Lehrstuhl für Messsystem- und Sensortechnik

Systemkonzept für die Heimautomatisierung

Yuriy Kyselytsya

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. techn. P. Russer
Prüfer der Dissertation:	1. Univ.-Prof. Dr.-Ing., Dr.-Ing. habil. F. Schneider, i.R.
	2. Prof. Dr.-Ing. J. M. Tuz, NTUU Kiew, Ukraine
	3. Univ.-Prof. Dr.-Ing. J. Eberspächer

Die Dissertation wurde am 15.10.2003 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 15.04.2004 angenommen.

Technische Universität München
Lehrstuhl für Messsystem- und Sensortechnik

Systemkonzept für die Heimautomatisierung

Dipl.-Ing. Yuriy Kyselytsya

Concept of a system for the Homeautomation

Abstract

This thesis describes a universal, modular concept of a management system for homeautomation. Since bus systems are spreading into private homes as result requirements have been defined for an universal management system, which unifies all in home installed systems to one logical system and through it make interworking possible beyond the system boundaries. Unlike the solutions for management systems in industrial automation and even for similar systems in building automation special attention in system concept has been laid on the fact, that the management system will be used mainly by technical not prepared users (tenants) and therefore must meet the corresponding requirements. In this thesis the suitable protocols for the system components will be selected. These protocols will be used in the realization example of the management system where the European Installation Bus (EIB) is used. With an outlook of the further development and the reinforcement of the management system the thesis closes.

Systemkonzept für die Heimautomatisierung

Kurzfassung

Die vorliegende Arbeit beschreibt ein universelles modulares Konzept des Managementsystems für die Heimautomatisierung. Ausgehend von der wachsenden Verbreitung der Bussysteme in privaten Heimen werden Anforderungen an ein universelles Managementsystem definiert, das alle im Haus installierten Systeme zu einem logischen System vereint und dadurch ihre systemübergreifende Zusammenarbeit ermöglicht. Im Unterschied zu den Managementsystemen in der Industrieautomatisierung und sogar zu den eng verwandten Systemen der Gebäudeautomatisierung wurde bei der Entwicklung des vorgestellten Konzepts besonders darauf geachtet, dass das Managementsystem überwiegend von technisch nicht versierten Anwendern (Hausbewohnern) benutzt wird und deshalb den entsprechenden Anforderungen gerecht werden muss. In der Arbeit wird eine Auswahl der passenden Protokolle für die Systemkomponenten getroffen. Diese Protokolle werden bei dem PC-basierten Realisierungsentwurf des Managementsystems am Beispiel des Europäischen Installationsbusses (EIB) eingesetzt. Ein Ausblick auf die Weiterentwicklung und die Erweiterung des Managementsystems schließt die Arbeit ab.

Inhaltsverzeichnis

1. Einführung

Seitdem der Mensch angefangen hat, sich ein Heim zu bauen, versucht er es auch zu vervollkommnen, um seinen Aufenthalt dort noch bequemer und komfortabler zu machen. Das ist nicht verwunderlich, da der Mensch den größten Teil seiner freien Zeit zu Hause verbringt.

Unser Heim hat während seiner Entwicklung mehrere Entwicklungsstufen passiert. Zuerst wurde es mechanisiert – die ersten Vorrichtungen für die Zustellung des Wassers ins Haus, für das Öffnen/Schließen des Tores oder für das Bewegen anderer sperriger Gegenstände im Haushalt sind entstanden. Die nächste wichtigste und gewissermaßen revolutionäre Entwicklungsstufe unseres Heimes war seine Elektrifizierung – als der elektrische Strom zu jedem Haushalt zugeführt wurde und es hell wurde. Auf der Entwicklungsstufe „Medialisierung" sind unterschiedliche Informationsquellen in unserem Heim aufgetaucht, wie z. B. das Radio, das Fernsehen oder das Internet. Zurzeit befinden wir uns auf der Entwicklungsstufe „Automatisierung". Die Stufe ist im Vergleich zu der Elektrifizierung nicht revolutionär, sondern eher von evolutionärem Charakter. Bei diesem Entwicklungsschritt wird die Entstehung von „vernetzten" Häusern erwartet, die so eine Art „Nervensystem" bekommen, das in der Lage ist, einzelne „Organe" des Hauses zu steuern – wie z. B. die Beleuchtung oder die Heizung [1]. Dies wird auch später zur Erscheinung von „intelligenten" Häusern beitragen. Solche Häuser sind das eigentliche Ziel des Automatisierungsschrittes. Ein „vernetztes" Haus wird erst dann zum „intelligenten", wenn es eine zentrale intelligente Instanz – „das Gehirn" – besitzt. Eine indirekte Bestätigung der richtigen Entwicklungsrichtung ist auch eine wachsende Akzeptanz der Idee bei der Bevölkerung (Bild 1.1) [2].

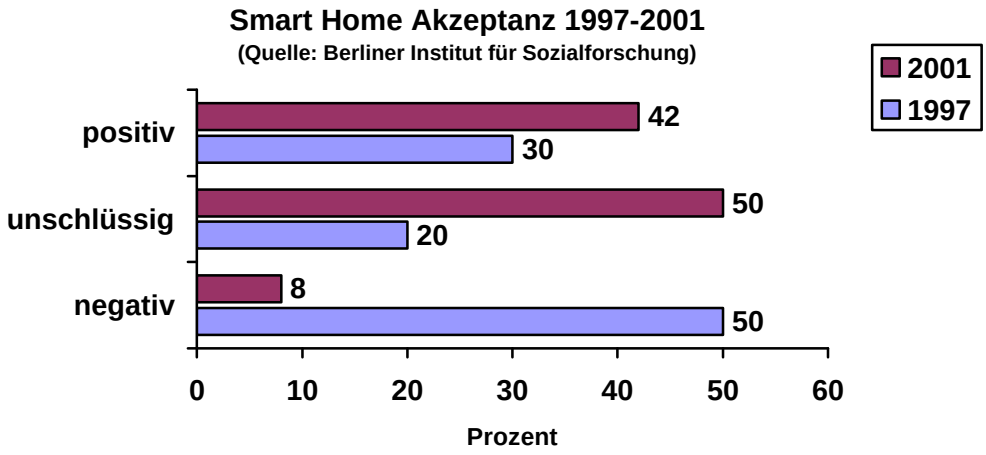


Bild 1.1. Akzeptanz der Idee „intelligentes Haus" im Zeitraum 1997-2001

Schon jetzt lässt sich vermuten, dass die nächste Entwicklungsstufe des Hauses seine Roboterisierung sein wird, d.h. einige Haushaltsfunktionen werden auf Roboter übertragen werden.

Die vorliegende Arbeit beschäftigt sich mit den Fragen der „Intelligisierung" des Heimes und stützt sich dabei auf gewonnene Erfahrungen aus den Forschungsprojekten „VIMP", „IWOBAY" und „*tele*-Haus", die unter der Leitung von Herrn Professor Schneider am Lehrstuhl für Messsystem- und Sensortechnik der Technische Universität München durchgeführt wurden.

1.1. Bussysteme im modernen Heim

Schon heutzutage und in noch größerer Anzahl in nächster Zukunft, wird es nicht schwer sein, ein Haus zu finden, wo ein oder mehrere Steuerungssysteme oder Datenübertragungssysteme installiert sind. Das erste solche weitverbreitete System, das Eingang in unsere Häuser gefunden hat, obwohl auch in einer vereinfachten Form, war die infrarote Fernbedienung, die dem Menschen erlaubt, verschiedene Haushaltsgeräte zu steuern, ohne den Lieblingssessel verlassen zu müssen. Einige Zeit später wurden viele Menschen von der Idee inspiriert, sich während ihrer Telefongespräche frei im Haus zu bewegen. Und so ist ein drahtloses System fürs Telefonieren „Digital Enhanced Cordless Telecommunications (DECT)“ in unser Heim gekommen.

In den letzten Jahren jedoch finden die „echten“ Automatisierungssysteme [3], wie z.B. European Installation Bus (EIB) [4,5,6], Local Operating Network (LON) [7,8], European Home System (EHS) [9] u.a., mehr und mehr den Weg ins private Heim. Diese Systeme verbinden unterschiedliche Geräte von verschiedenen Herstellern und ermöglichen es, ein nie dagewesenes Niveau an Bequemlichkeit und Komfort für den Endverbraucher zu erreichen. Die Systeme, die bis jetzt vor allem aus der Gebäudesystemtechnik bekannt waren und ihre Verwendung in großen Offices oder industriellen Gebäuden fanden, waren für den Endverbraucher wegen ihrer technischen Komplexität und den hohen Preisen unzugänglich.

Man muss auch ein weitverbreitetes System erwähnen, das sich nicht direkt im Haus, sondern nebendran in der Garage, befindet. Es geht um das Auto – jedes mehr oder weniger moderne Fahrzeug stellt ein komplexes System aus vielen Sensoren und Controllern dar, die zu einem gemeinsamen Netz, meist mit Hilfe von Controller Area Network (CAN) [10,11], verbunden sind.

Nachdem die meisten privaten Haushalte ein oder mehrere Computer besitzen, findet man dort öfter auch das Ethernet – das Computernetz für die Datenübertragung [12]. Die weite Verbreitung des Ethernets und die damit verbundene günstige Hardware hat zu der Idee geführt, das Ethernet auch als Automatisierungsnetz zu benutzen [13].

Die Vielfalt der möglichen Systeme im privaten Heim wird noch dadurch ergänzt, dass einige von ihnen unterschiedliche und mehrere Medien für die Informationübertragung nutzen. Solche meist verbreiteten Medien sind Twisted Pair (Übertragung über Zweidraht), Powerline (Übertragung über die elektrische 230V-Leitung) und Wireless (drahtlose Übertragung, in der Regel per Funk).

Und so kann man zum jetzigen Zeitpunkt feststellen, dass es keinen einheitlichen Standard für das Heimnetz gibt. Es lässt sich auch bezweifeln, dass es einen solchen Standard in Zukunft geben wird. Allzu groß ist der Unterschied in den Anforderungen der einzelnen Gruppen von Geräten an ein solches System und noch dazu eine Anpassung der vorhandenen Geräte an das neue System oder eine Entwicklung von neuen Geräten wird aus wirtschaftlicher Sicht für den Hersteller praktisch unzumutbar.

Bedenklich ist auch die Akzeptanz eines neuen Systems beim Endverbraucher, weil nur wenige von ihnen bereit sein werden, die komplette technische Infrastruktur des Hauses zu erneuern. Die Mehrheit wird einfach die bestehende Infrastruktur weiter ausbauen und/oder durch neue Produkte, Dienste oder Geräte ergänzen. Dabei werden ihre Nutzbarkeit, die Kosten und die Einfachheit im Vordergrund stehen.

Ein fortgeschrittenes Niveau des „intelligenten“ Hauses kann man dann erreichen, wenn man dem Haus die Möglichkeit gibt, mit der Außenwelt zu kommunizieren, d.h. es mit dem globa-

len Netz – dem Internet zu verbinden. Und tatsächlich von Zeit zu Zeit kündigen die Hersteller im Bereich der Heimautomatisierung neu entwickelte intelligente Geräte an, die das Haus mit dem Internet verbinden und die sog. „Services“ in jedes Haus liefern. Diese Services sollten dem Hausbewohner ein bis jetzt unerreichtes Niveau an Komfort und Bequemlichkeit bieten. Aber bei genauer Betrachtung entpuppen sich diese Geräte als eine proprietäre Lösung der einzelnen Hersteller und sind nicht miteinander kompatibel. Das bedeutet, dass der Anwender, der die Services verschiedener Hersteller nutzen will, auch die Geräte von verschiedenen Herstellern erwerben muss. Dies ist nicht nur ein technisch aufwendiges sondern auch ein kostspieliges Vergnügen.

1.2. Zielsetzung der Arbeit

Unter Berücksichtigung der oben beschriebenen Situation ist das Ziel der vorliegenden Arbeit nicht die Entwicklung eines neuen standardisierten Systems für das private Heim, sondern ein Versuch, ein einheitliches flexibles Managementsystem im privaten Heim zu schaffen, indem man alle dort installierten Systeme zu einem logischen System zusammenschließt. Dabei versucht man alle Vorteile der jeweiligen Systeme beizubehalten und die Nachteile zu verringern.

Für ein solches System kann man zwei Lösungen vorschlagen: die erste – ein „mächtiges“ Gateway mit angeschlossenen internen Netzen und einem externen Interface; die zweite – mehrere „schlanke“ Gateways, bei dem jedes ein gewisses „piece of mind“ des kompletten Systems beinhaltet.

Bei der Zielsetzung der Arbeit wurden ebenfalls, die in [2] beschriebenen Vorstellungen der Bewohner eines „intelligenten“ Heimes berücksichtigt. Es muss:

- ✓ kinderleicht zu bedienen sein
- ✓ technisch begreifbar sein
- ✓ einen günstigen Einstiegspreis haben
- ✓ modular erweiterbar sein
- ✓ einfach zu installieren sein
- ✓ altbautauglich sein
- ✓ sicher in der Handhabung sein
- ✓ Zeit sparen und nicht Zeit fressen
- ✓ Datensicherheit gewährleisten

Wenn man die Tatsache in Betracht zieht, dass der Anwender als ein Systemintegrator oftmals mit den modernen Systemen der Automatisierung überfordert ist, wurde bei der Entwicklung des Systemkonzeptes den benutzerfreundlichen Werkzeugen und Mitteln eine besondere Aufmerksamkeit geschenkt. Diese sollen es sogar dem unvorbereiteten Anwender ermöglichen, wenn nicht gerade die Erstinstallation und Inbetriebnahme, dann zu mindest die Erweiterung des Systems oder kleine Änderungen ihrer Konfiguration vorzunehmen.

Noch ein wichtiger Aspekt, der bei der Entwicklung des einheitlichen Managementsystems verfolgt wurde, war: den Anwender in den Entwurf und den Aufbau des Systems miteinzubeziehen. Dies soll es ermöglichen, ein höheres Niveau an Vertrauen zwischen Mensch und System zu erreichen und so die Angst der Bewohner überwinden, nicht mehr Herr im eigenen Haus zu sein.

Zum Schluss muss man anmerken, dass das eigentliche Ziel der in dieser Arbeit beschriebenen Konzeptimplementierung nicht die Entwicklung eines marktreifen Produktes war, sondern mehr der Nachweis der Realisierbarkeit und der Tragfähigkeit des Konzepts. Als Folge wurden einige Komponenten zum jetzigen Zeitpunkt nicht im vollen, sondern im dafür notwendigen Umfang implementiert. Die volle Umsetzung des Systemskonzeptes mit dem Entwurf aller Software- und Hardwarekomponenten gilt für den Autor als langfristiges Ziel, das, er hofft, in Zukunft verfolgen zu dürfen.

1.3. Gliederung der Arbeit

Die vorliegende Arbeit beschreibt das Konzept eines einheitlichen Managementsystems für das private Heim und dessen Realisierung am Beispiel des Europäischen Installationsbusses. Kapitel 2 „Stand von Wissenschaft und Technik“ versucht die verschiedene Ideen aus den anderen Forschungsarbeiten und Projekten und die vorhandenen technischen Lösungen, vor allem sind es Gateways, die man in einem solchen System verwenden könnte, vorzustellen.

Der wichtigste Teil der Arbeit und zwar die Beschreibung des Konzepts eines einheitlichen Managementsystems für das private Heim wird im Kapitel 3 „Systemkonzept“ dargestellt. Hier sind die wichtigsten Hardware- und Softwarekomponenten des Systems und die Forderungen an ihre Realisierung aufgelistet.

Basierend auf diesen Forderungen werden im folgenden Kapitel 4 „Auswahl der Protokolle“ kurze Beschreibungen der passenden Protokolle für die einzelnen Komponenten vorgestellt und auf dieser Grundlage werden die geeignetsten ausgewählt.

Diese Protokolle werden dann bei der im Kapitel 5 „Realisierungsentwurf“ beschriebenen Realisierung des Konzeptes des Managementsystems am Beispiel des EIB Untersystems benutzt. Hier sind auch die Besonderheiten des internen Aufbaus der einzelnen Komponenten, ihre Schnittstellen und die Zusammenarbeit beschrieben. Dieses Kapitel enthält außerdem viele Screenshots der Softwarekomponenten.

Zum Schluss im Kapitel 6 „Zusammenfassung und Ausblick“ wird eine Bilanz der durchgeführten Arbeit gezogen und es wird der Versuch unternommen, die Tragfähigkeit des Konzepts abzuschätzen. Hier sind auch einige Vorschläge zur Verbesserung und Erweiterung des Systems aufgeführt. In diesem Kapitel sind auch einige Bilder der Hardwarekomponenten des implementierten Systems zu finden.

Zum besseren Verständnis des im Kapitel 5 beschriebenen Realisierungsentwurfes der einzelnen Komponenten und des Strukturaufbaus von Services wird im Anhang A eine kurze technische Beschreibung des Europäischen Installationsbusses gegeben.

2. Stand von Wissenschaft und Technik

Wie schon in dem vorhergehenden Kapitel erwähnt wurde, ist das Ziel der vorliegenden Arbeit die Schaffung eines flexiblen Managementsystems für das private Heim. Dabei sollte nicht ein neues System entwickelt werden, sondern alle vorhandenen Netze zu einem logischen Netz zusammen gefasst werden.

In diesem Kapitel werden die technischen Ideen, die in verschiedenen anderen Forschungsarbeiten und Projekten verwendet wurden, die vorhandenen Technologien für das Verbinden von unterschiedlichen Netzen und auch marktverfügbare Lösungen in Bezug auf die Automatisierungssysteme betrachtet.

2.1. Stand der Wissenschaft

Im Laufe der Vorbereitung der vorliegenden Arbeit ist es dem Autor nicht gelungen, eine grundliegende allgemein anerkannte Theorie für das Verbinden und die Zusammenarbeit der unterschiedlichen heterogenen Netze für die Steuerung und Datenübertragung zu finden. Als Folge davon wurden verschiedene Ideen und Entwicklungen, die in anderen Forschungsprojekten und Arbeiten verwendet werden, als theoretische Grundlagen benutzt. Einige von ihnen werden in den nächstfolgenden Kapiteln betrachtet und es wird auch ein Versuch unternommen, aus ihnen die positiven Aspekte für die weitere Verwendung bei der Entwicklung des eigenen Konzeptes für die Heimautomatisierung herauszufinden.

2.1.1. Energiemanagementkonzept für Büros

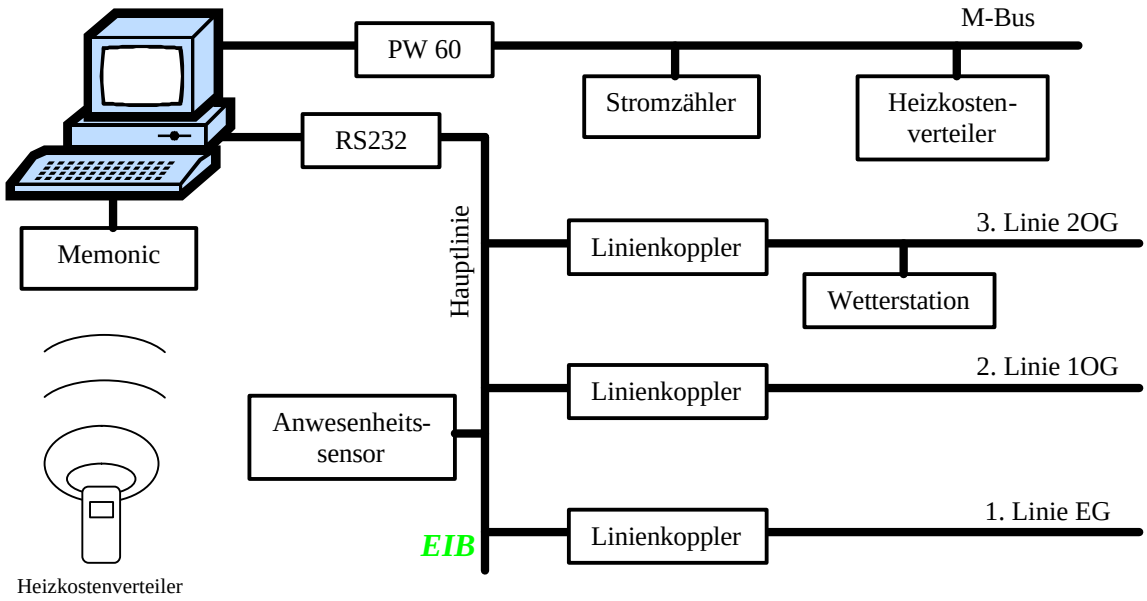


Bild 2.1. Systemaufbau des Energiemanagemntkonzepts

Das System wurde im Institut für Elektrische Energietechnik der Universität Gesamthochschule Kassel in Rahmen des internationalen Forschungsprojektes „Building Energy Management With Smart Systems“ (BySyS) [14] entwickelt.

Eine zentrale Rolle im Systemaufbau (Bild 2.1) spielt ein Computer. In diesem sind Hardwareschnittstellen für den Zugriff auf die einzelnen Untersysteme eingebaut. Auf diesem läuft die ganze Systemsoftware unter der Laufzeitumgebung „LabView“. Der Informationsfluss im System ist in Bild 2.2 zu sehen.

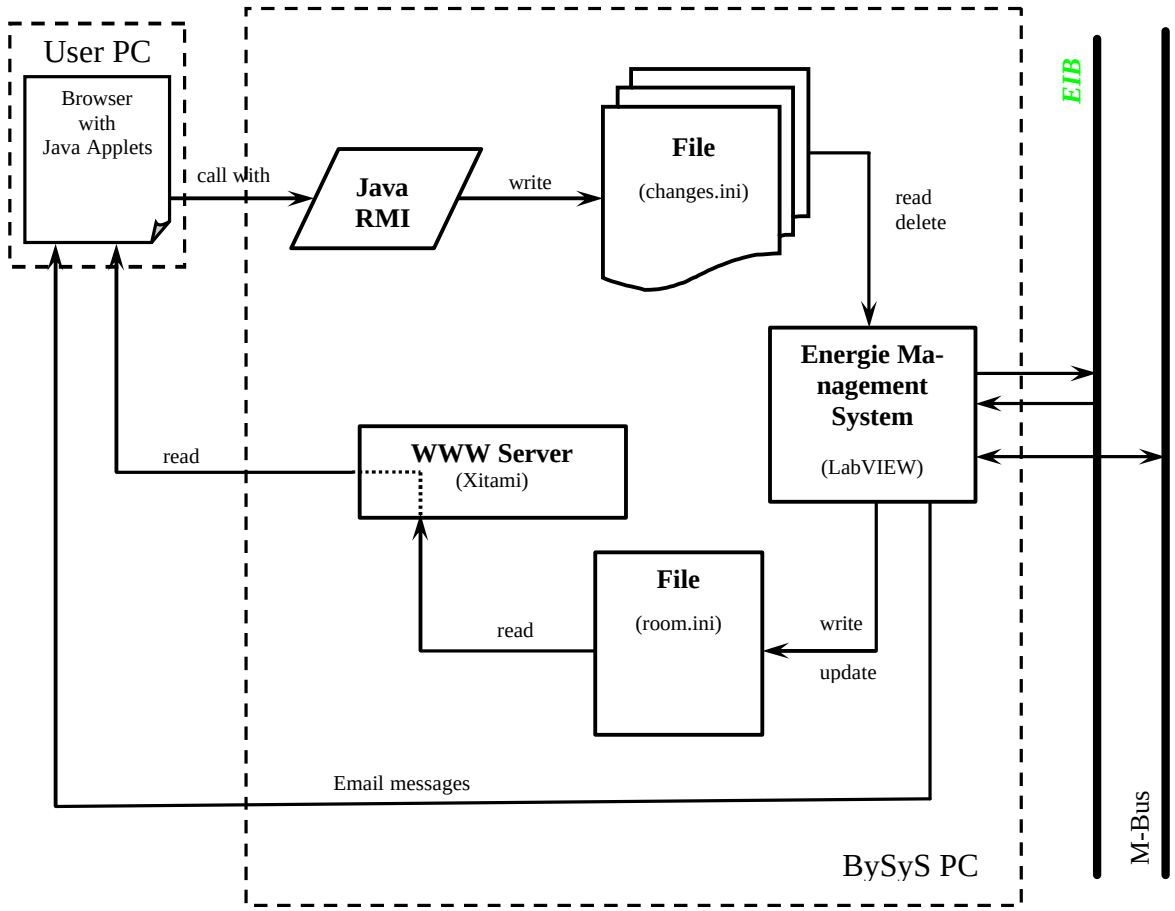


Bild 2.2. Informationsfluss im Energiemanagementsystem

Ein solcher Aufbau des Systems hat sowohl positive als auch negative Aspekte. Zu den positiven gehören ohne Zweifel seine Universalität (Integration der drei unterschiedlichen Untersysteme) und Erweiterbarkeit (Einbau weiterer Untersysteme mit geringem Aufwand möglich). Auch hat der Einsatz des Internetbrowsers als Visualisierungssoftware und die damit verbundene Möglichkeit, verschiedene Hardwareplattformen zu nutzen, eine sichere Zukunft. Zweifelhaft ist andererseits der Einsatz von „LabView“ als Softwareplattform. Diese Tatsache beschränkt die Auswahl der passenden Hardwareplattformen wegen der notwendigen Ressourcen (Rechnerleistung, Speicher, etc.) nur auf eine – den Computer. Die Verwendung der dateibasierenden Austauschprotokolle gehört eindeutig in die Vergangenheit und findet kaum noch Einsatz in modernen Systemen, weil sie den Datenaustausch nur auf eine lokale Plattform begrenzen und diesen in der Netzwerkumgebung praktisch unmöglich machen.

2.1.2. Integriertes Managementkonzept für die Gebäudesystemtechnik

Im Rahmen einiger Forschungsprojekte am Lehrstuhl für Messsystem- und Sensortechnik der Technischen Universität München wurde ein Softwarekonzept für die Automatisierungssysteme im privaten Heim „IMOS-*home*“ [15] entwickelt und realisiert. Die Komponenten dieser Software unterstützten den Anwender bei allen Automatisierungsschritten: von Installation und Inbetriebnahme, bis zu den alltäglichen Aufgaben der Steuerung und Visualisierung des Systemszustandes, oder auch bei der Fehlersuche (Bild 2.3).

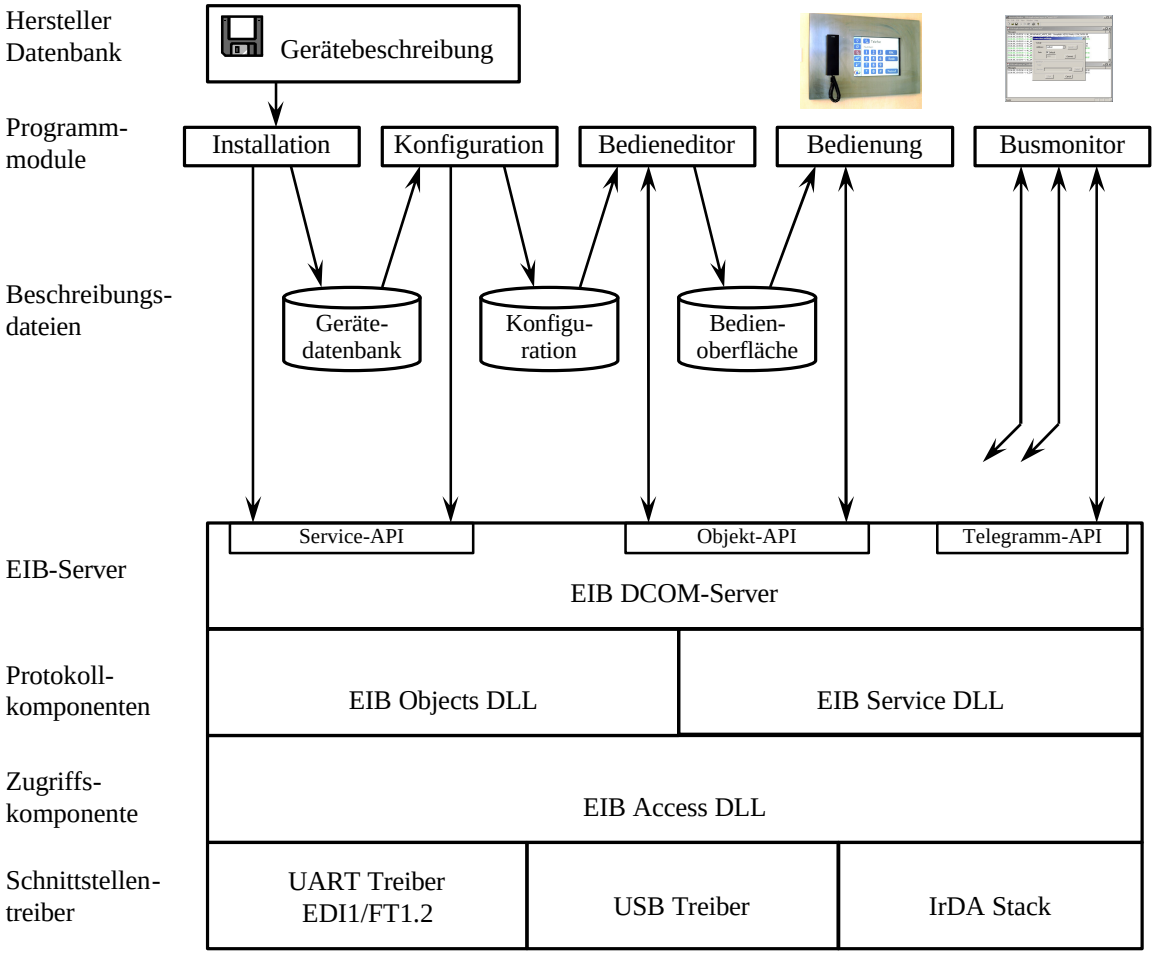


Bild 2.3. Strukturüberblick der „IMOS-*home*“ Software

In diesem Konzept sind der fast ideale Aufbau der Software und ihre Aufteilung in die einzelnen Komponenten (da fehlt vielleicht nur das Konzept und die Schnittstelle zur Integration von Komponenten von Drittherstellern) besonders lobenswert. Die Definition der einzelnen Module und ihrer Zusammenarbeit lässt kaum noch etwas an Wünschen übrig. Auch bei der Entwicklung der Visualisierungs- und Steuerungsoberflächen sind die Akzente – auf die Einfachheit und das intuitive Verstehen sogar durch den unerfahrenen Benutzer – richtig gesetzt worden.

Zu den Nachteilen des Konzeptes kann man seine ungenügende Flexibilität als Folge der festen lokalen Verbindung zwischen dem Schnittstellentreiber, der Zugriffskomponente und den Protokollkomponenten zählen. Es ist aus heutiger Sicht zu fragen, ob tatsächlich jede Schnitt-

stelle ein eigenes Prozessabbild (EIB Objects DLL) braucht und ob es nicht vorteilhafter wäre, wenn sie ein gemeinsames Prozessabbild nutzen könnten? Hinzu kommt, dass die entsprechenden Mittel und Werkzeuge bei der Realisierung des Konzeptes nicht optimal ausgewählt worden sind. So zum Beispiel begrenzt die Anwendung des Distributed Component Object Model (DCOM) als Serverschnittstelle für die Clientprogramme die Auswahl der Softwareplattformen für die lokalen und fernzugreifenden Visualisierungs- und Konfigurierungskomponenten des Automatisierungssystems praktisch auf eine Plattform – Windows. Dies seinerseits, verstärkt noch durch den Einsatz der proprietären Protokolle und Formate in den Visualisierungskomponenten, macht die Verwendung der kleinen tragbaren Geräte (wie z.B. PDA, Handy, etc.) als Hardwareplattform für die Steuerungs- und Visualisierungssoftware unmöglich.

2.1.3. Neues Framework für die EIB/KNX Anwendungen

Die im Folgenden beschriebene Softwareumgebung für die EIB/KNX Anwendungen [16] wurde im Institut für Rechnergestützte Automation der Technischen Universität Wien entwickelt. Dieser Lösung liegt ein eigenentwickeltes Softwarekonzept zugrunde, das auf dem sog. „virtual shared object space“ (VSOS) basiert (Bild 2.4).

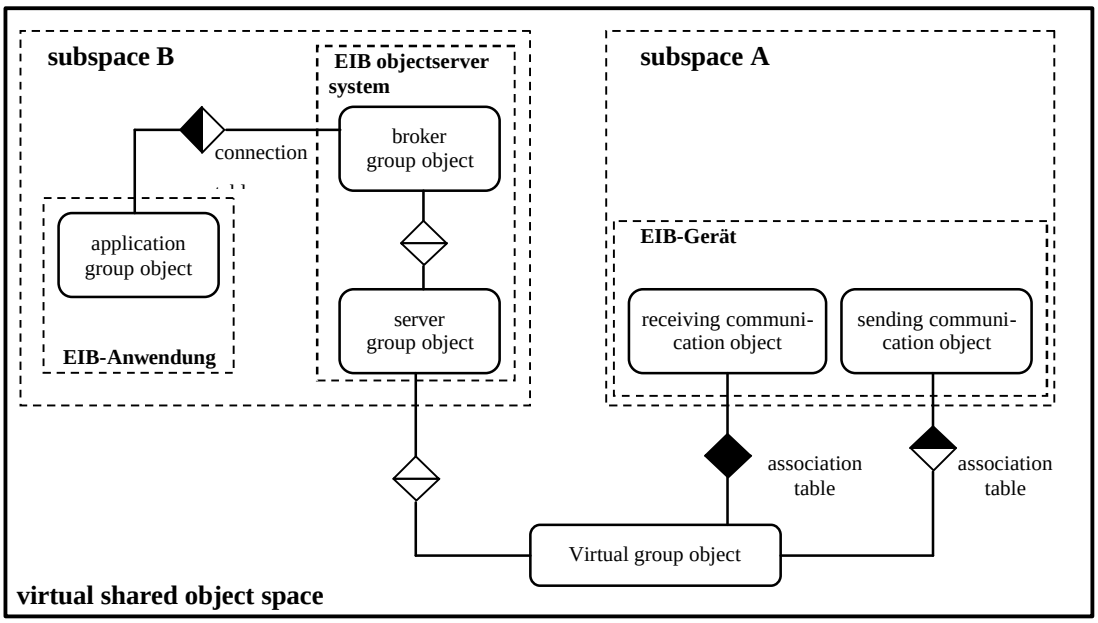


Bild 2.4. VSOS Modell

Dieses Konzept sieht im System eine abstrakte Ebene vor, die der Anwendungssoftware erlaubt, nur gewisse „virtual shared objects“ zu manipulieren. Diese aber werden transparent für den Anwender mit den physikalischen Objekten im System verbunden. Die Zugriffe auf die „virtual shared objects“ werden vom „objectserver“ verwaltet (Bild 2.5), der für die Anwendungsprogramme eine universelle Zugriffsschnittstelle zur Verfügung stellt.

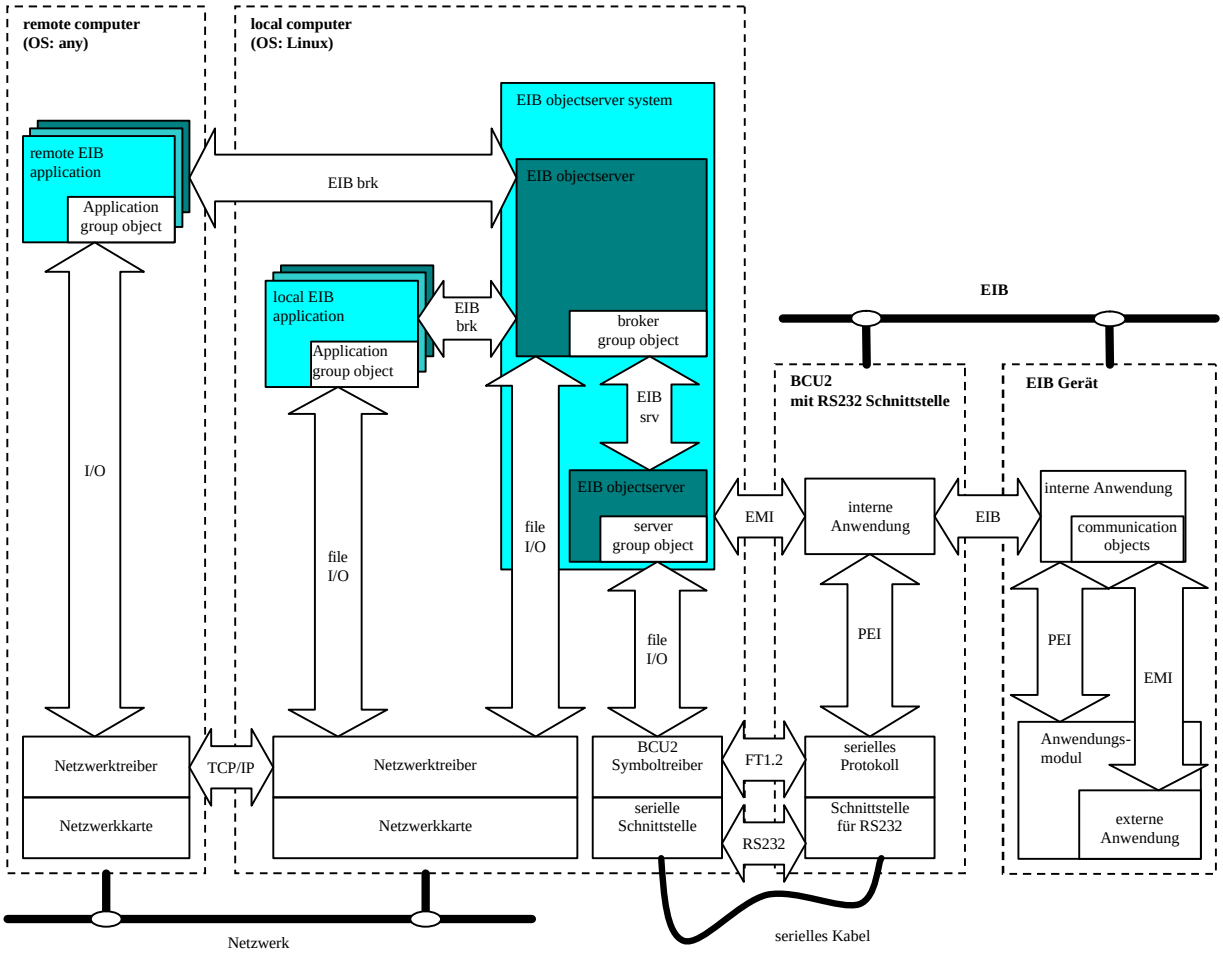


Bild 2.5. Systemaufbau des Frameworks für die EIB/KNX Anwendungen

Zu den Vorteilen dieses Systems gehört eindeutig seine Erweiterbarkeit (dank der Abstraktionsebene) und die flexible Skalierbarkeit der Software, die ihrerseits erlauben, es auf einem breiten Spektrum vorhandener Hardwareplattformen einzusetzen. Ein derartiges System hat auch Nachteile. Das sind an erster Stelle – die Benutzung der proprietären Protokolle für den Fernzugriff und das Fehlen des direkten Zugriffes auf das Untersystem (zum Zweck der Konfigurierung und der Suche nach Fehlern), dazu noch der statische Charakter des Systems – d.h. eine beliebige Veränderung in der Struktur des Systems oder der einzelnen Untersysteme zieht die nötigen Anpassungen in der Anwendersoftware nach sich oder die Rekonfigurierung des Systems. So zum Beispiel wird eine Ergänzung des Systems durch ein Untersystem mit eigenem „objectserver“ den Anwender vor das schwer lösbare Problem stellen – welche Objekte muss er jetzt auf welchem Server suchen? Auch die Flexibilität des Systems kann entscheidend verbessert werden, indem man die lokale feste Verbindung zwischen dem „objectserver“ und der physikalischen Zugriffsschnittstelle des Untersystems entfernt. Dies würde gleichzeitig ermöglichen, die unterschiedlichen räumlich getrennten Untersysteme mit einem „objectserver“ zu betreiben.

2.2. Verfügbare Protokolle und Busse

In den folgenden Abschnitten werden die verschiedenen Bussysteme und Protokolle kurz erläutert, die in unterschiedlichen Automatisierungssystemen ihre Verwendung finden. Diese Bussysteme wurden für den Einsatz in einem bestimmten Marktsegment entwickelt und sind deshalb in der Regel entsprechend aufgebaut [17,18]. Dabei werden auch weniger bekannte Bussysteme in Betracht gezogen werden, um die Vielfalt der Möglichkeiten zu zeigen.

Die Eigenschaften der betrachteten Bussysteme werden im Hinblick auf das zu entwickelnde Konzept des Managementsystems dargestellt. Dabei wird den physikalischen Eigenschaften: Topologie, unterstützte Übertragungsmedien, Übertragungsgeschwindigkeit, etc. und auch dem Aufbau des Protokollstacks des jeweiligen Bussystems eine besondere Beachtung geschenkt, da diese die Auslegung des Konzepts entscheidend beeinflussen können.

2.2.1. CAN

Das Bussystem CAN wurde im Jahre 1982 von der Firma Bosch vorgestellt und wurde vor allem für den Einsatz in Kraftfahrzeugen entwickelt. Dieses Anwendungsgebiet des Busses bestimmt seine Haupteigenschaften. Eine der wichtigsten von ihnen ist die Fehlertoleranz des Bussystems. Dies bedeutet nicht nur die Fähigkeit verschiedene Fehler zu tolerieren, sondern auch die Verfügbarkeit der mehrstufigen Mechanismen zur Fehlererkennung und auch die Mechanismen der Selbstkontrolle und Selbststeuerung. Das heißt, dass in CAN die Sicherheit an erster Stelle steht, im Gegensatz zu vielen anderen Systemen, wo zum Beispiel mehr Wert auf die hohe Verfügbarkeit gelegt wird. In CAN bestätigt nicht nur jeder Teilnehmer jede empfangene Nachricht, sondern hat die Möglichkeit, eine „falsche“ Nachricht zu überschreiben und damit zu verhindern, dass andere Teilnehmer sie empfangen. Diese Möglichkeit, dass einzelne Busknoten Nachrichten zerstören können, birgt in sich die Gefahr, dass fehlfunktionierende Knoten den Bus blockieren. Um Letzteres zu verhindern, enthält jeder Knoten einen Fehlerzähler, der beim Erkennen des Empfangs- oder Sendefehlers inkrementiert und beim erfolgreichen Senden oder Empfangen dekrementiert wird. Der Zustand dieses Zählers lässt Schlussfolgerungen über die Zuverlässigkeit des jeweiligen Knotens zu und erlaubt ihm abhängig vom Wert des Zählers entweder aktiv an der Buskommunikation teilzunehmen oder mit bestimmten Einschränkungen. Im Extremfall kann sich der Knoten vollständig vom Bus abschalten. Ein nächster Schritt zur Erhöhung der Buszuverlässigkeit von CAN stellt die Möglichkeit dar, die physikalische Datenübertragung, zum Beispiel beim Bruch einer der beiden Busleitungen, von einem differenziellem Übertragungsverfahren auf eine massebezogene Übertragung umzustellen.

Für die physikalische Datenübertragung in CAN wird ein auf dem RS485-Standard basierendes differentielles Verfahren mit dem CSMA/CA Buszugriffsverfahren eingesetzt. Dieses Verfahren ermöglicht auf dem Bus Datenraten bis zu 1 MBit/s zu erreichen, dabei darf die Länge der Busleitung 40m allerdings nicht überschreiten.

Im Unterschied zu den meisten anderen Feldbussen verwendet CAN nicht die teilnehmerbezogene Adressierung, sondern eine nachrichtenorientierte. Dies veranlasst die Klassifizierung von CAN als ein Multi-Master System und eher als ein ereignisorientiertes System. Der CAN Standard definiert im Übertragungsprotokoll nur die Schichen 1 und 2 des ISO/OSI Kommunikationsmodells.

Der nächste wichtige Aspekt des CAN ist der Verzicht auf die Flexibilität bezüglich Konfiguration oder Parametrisierung der einzelnen Busknoten und als Folge dessen das Erreichen

einer einfachen Integration der Busknoten in ein existierendes System. Dies ermöglicht seinerseits eine enorme Wirtschaftlichkeit des Busses und gerade deshalb findet CAN heutzutage nicht nur im Kraftfahrzeug Verwendung, sondern in der Medizintechnik, in der industriellen Automatisierung usw.

2.2.2. EIB

Der EIB basiert auf dem Standard aus dem Gebiet der Installationstechnik und genießt besonders in Europa eine große Anerkennung und daher Verbreitung, vor allem in der Gebäudeautomatisierungstechnik. Im Jahre 1990 wurde die EIB Assoziation (EIBA) von den führenden europäischen Firmen im Bereich der Installationstechnik gegründet, die eine unabhängige Standardisierungs- und Zertifikationsstelle darstellt. Der EIB war Basis des im Jahre 1992 in Deutschland veröffentlichten Vorstandards DIN V VDE 0829 „Elektrische Systemtechnik für Heim und Gebäude (ESHG)“.

Das Haupteinsatzgebiet des EIB ist die Gebäudeautomatisierung und die Gebäudeleittechnik. Entsprechend ist die physikalische Topologie der EIB Netze aufgebaut, sie besteht aus Bereichen und Linien. Von den physikalischen Übertragungsmedien unterstützt EIB Zweidraht, Powerline, Infrarot- und Funkwellen. Die Übertragungsgeschwindigkeit variiert je nach Übertragungsmedium und entspricht, zum Beispiel, für Zweidraht 9600 Bit/s. Als Zugriffsverfahren wird das CSMA/CA Verfahren verwendet. Im Protokollstack des EIB sind die Schichten von 1 bis 4 und 7 des ISO/OSI Kommunikationsmodells definiert.

Ein typisches EIB Gerät besteht aus einer BCU (Bus Coupler Unit) und einem Applikationsmodul. Die BCU, die die physikalische und logische Verbindung zwischen unterschiedlichen Applikationen und dem EIB gewährleistet, umfasst ein Sende- und Empfangsmodul, einen Mikrokontroller mit Systemsoftware einschließlich EIB Protokollstack und eine PEI Schnittstelle, die zum Kontakt mit der Außenwelt dient. Jedem EIB Gerät wird nach seinem Anschluss ans System eine eindeutige physikalische Adresse zugeteilt, die aus einer Bereich-, Linien- und Gerätenummer zusammengestellt ist. Neben der physikalischen Adressierung, die hauptsächlich für die Initialisierung, Programmierung und Diagnostik des einzelnen Gerätes verwendet wird, wird beim EIB auch eine „Gruppen“-Adressierung eingesetzt. Ein wichtiger Vorteil eines solchen Adressierungsverfahrens besteht darin, dass dies die Kommunikation mit mehreren Geräten gleichzeitig ermöglicht.

Um eine vernünftige Kommunikation zwischen den Geräten unterschiedlicher Hersteller zu erreichen, hat die EIBA einen EIB Interworking Standard (EIS) verabschiedet, der genau definiert, welche Datentypen und auf welche Weise auf dem Bus zu übertragen sind. Eine genauere Beschreibung des EIBs findet sich in Anhang A.

2.2.3. Interbus

Der Interbus wurde im Jahre 1987 von der Firma Phoenix Contact entwickelt und ist mittlerweile in der CENELEC-Norm EN 50254 und in der IEC-Norm 61158 standardisiert. Der Interbus unterscheidet sich erheblich von allen anderen Feldbussen. Er besitzt eine Ringtopologie und gehört zu den Master-Slave Systemen. Alle Slaves werden vom Master als seine eigene Peripherie (genauer gesagt als ein großes Schieberegister) betrachtet. Jeder Slave stellt eine einzige Zelle dar, durch die der Master die Daten nacheinander „durchschiebt“. Am Ende der Schiebeperiode gelangen die Daten zurück zum Master. Der Interbus ist ein Aktor/Sensor

Bus, der für den Prozessdatenaustausch auf der untersten Ebene der Automatisierungshierarchie optimiert ist. Diese Prozessdaten fallen in der Regel zyklisch an und müssen in einem begrenzten Zeitintervall verarbeitet werden.

Als Folge seiner Ringtopologie hat der Interbus einen großen Nachteil – seine Tendenz zum Gesamtausfall des Feldbusses beim Ausfall eines Teilnehmers. Dieser Nachteil fällt nicht besonders ins Gewicht, da dort, wo der Interbus meistens eingesetzt wird, nämlich in der industriellen Automatisierung, der Ausfall eines Busteilnehmers zum Stopp des gesamten Prozesses führt, weil eine Teilfunktion des Prozesses keinen Nutzten bringt. Der Interbus hat infolge seiner relativ einfachen Funktionsweise auch Vorteile. Diese sind die fehlende Notwendigkeit in der Adressierung der einzelnen Busteilnehmer, weil ihre „Adresse“ durch ihre Position in dem Systemring bestimmt wird und in der Steuerung der Zugriffe auf das Übertragungsmedium, da in jedem Systemsegment bloß ein Sender und Empfänger existiert. Infolgedessen fehlen auch die Kollisionen auf dem Bus.

2.2.4. LonWorks

LonWorks ist ein von der nordamerikanischen Firma Echelon entwickelter Feldbus, der für ein möglichst breites Einsatzspektrum konzipiert wurde. Heutzutage wird LonWorks sowohl in der industriellen Automatisierung als auch in der Gebäude- und Heimautomatisierung eingesetzt.

Als Übertragungsmedium kann LonWorks Zweidraht, Powerline, Infrarot-, Funkwellen und den Lichtwellenleiter benutzen. Die Auswahl des einen oder anderen Mediums hängt von der Auswahl des entsprechenden Transceivers ab, der den Knoten mit dem Übertragungsmedium verbindet und für die Anpassung der Spannungspegel und die Datenmodulation auf der physikalischen Ebene des Übertragungsprotokolls verantwortlich ist. Die Datengeschwindigkeit variiert bei LonWorks in Abhängigkeit vom Übertragungsmedium von wenigen Kilobits bis einigen Megabits und beträgt z.B. bei Zweidraht 78 Kbit/s.

In LonWorks wird das LonTalk Protokoll verwendet, das seit 1998 offengelegt ist und als Standard EIA 709.1 und ENV 13154-2 veröffentlicht ist. Das LonTalk Protokoll definiert die Schichten von 1 bis 4, 6 und 7 des ISO/OSI Kommunikationsmodells. Die meisten LonWorks Geräte sind auf der Basis des Neuron-IC Chips aufgebaut, der das softwaremäßig implementierte LonTalk Protokoll enthält und in der Lage ist, die Anwenderapplikationen auf dem Chip auszuführen.

Der Datenaustausch in einem LonWorks System basiert auf Kommunikationsobjekten, wobei das meist verbreitete von ihnen die Netzvariable ist. Alle standardisierten Typen der Netzvariablen werden von der LonMark Interoprability Association verwaltet. Diese hat zum Ziel eine bessere Kompatibilität zwischen den Geräten unterschiedlicher Hersteller zu erreichen.

2.2.5. Profibus

Profibus ist über CENELEC in EN 50170 und in IEC-Norm 61158 standardisiert. Das Ziel bei der Entwicklung des Busses war es, eine solche Flexibilität zu erreichen, um einfache Knoten oder aber auch komplexe Steuerungsgeräte miteinander verbinden zu können. Infolgedessen gibt es drei verschiedene Typen von Profibus: FMS (Fieldbus Messaging Specification), DP (Decentralized Peripherals) und PA (Process Automation).

Die Haupteigenschaften jedes einzelnen Typs des Profibusses werden in [6] wie folgt zusammengefasst:

- FMS ist ein Multimaster-System für mittlere Datenmengen auf Zellebene, bei dem Echtzeit gefordert ist, aber keine extrem kleinen Reaktionszeiten
- DP ist ein Single-Master-System für kleine Datenmengen mit kleinen Reaktionszeiten zwischen 2 bis 5 ms. DP ist einfach zu programmieren und effizient einzusetzen
- PA ist ein speziell für den eigensicheren Bereich konzipiertes System für kleine Datenmengen, einfach zu programmieren und ideal im gemeinsamen Betrieb mit DP

Als Übertragungsmedium wird der Lichtwellenleiter oder RS485 eingesetzt. Letzterer ist der meist verwendete und stellt eine verdrillte, abgeschirmte Zweidrahtleitung dar. Die Übertragungsgeschwindigkeit variiert von 9,6 Kbit/s bis zu 500 Kbit/s (Profibus/DP 12 Mbit/s). Das Protokollstack des Profibusses definiert nur die Schichten 1, 2 und 7 des ISO/OSI Kommunikationsmodells.

Da es keine fertigen Chipsätze für Profibus gibt, ist die Implementierung des Standards jedem Hersteller selbst überlassen. Daraus ergibt sich das aktuelle Problem der Kompatibilität zwischen den Geräten verschiedener Hersteller. Das ist einer der Gründe, weshalb viele Profibus Zertifikationszentren entstanden sind, die die Einhaltung der Norm und der definierten Profiles überwachen.

2.3. Technische Grundlagen für das Zusammenschließen getrennter Netze

Für das Zusammenschließen von verschiedenartigen Netzen oder von verschiedenen Segmenten eines gleichartigen Netzes gibt es unterschiedliche Gerätetypen. All diese Geräte enthalten zwei oder mehr physikalische Schnittstellen für das Anschließen der entsprechenden Netze und unterscheiden sich in der Regel nur darin, auf Grund welcher Informationen der Datenweg von einer Schnittstelle zur anderen ausgewählt wird, oder darin, welche Datenumwandlung bei der Übermittlung durchgeführt wird.

Ausgehend davon, dass diese Geräte für ihre Arbeit teilweise die aus den übermittelten Daten gewonnen Informationen nutzen, muss sichergestellt werden, dass sie diese auch “verstehen” und aus ihnen die notwendige Information herausfiltern können. Dies lässt sich wesentlich vereinfachen, wenn der Protokollstack des entsprechenden Netzes auf dem von der internationalen Standardorganisation (ISO) empfohlenen Schichtenmodell [19,20] (Bild 2.6) basiert, das den strukturellen Aufbau von im Netz übertragenen Daten standardisiert.

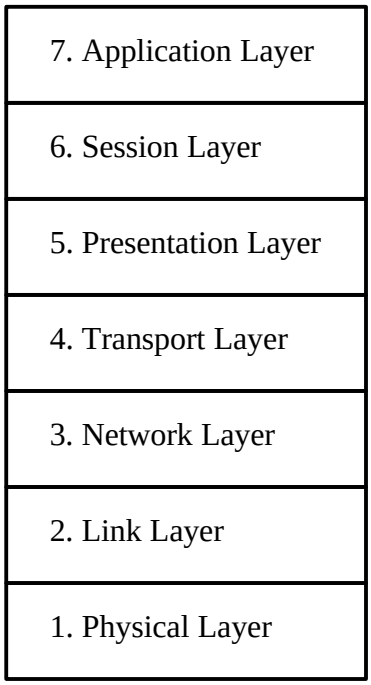


Bild 2.6. Aufbau des Protokollstacks nach dem ISO/OSI Schichtenmodell

2.3.1. Repeater

Die einfachsten Geräte für das Verbinden von Netzen sind die Repeater (Bild 2.7). In der Regel haben sie nur zwei physikalische Schnittstellen und einen kleinen Hardwareteil – der Softwareteil fehlt ganz. Als Folge dessen können solche Geräte nur Daten auf der untersten Schicht des Schichenmodells – der physikalischen Schicht übertragen. Die Repeater werden hauptsächlich zur Überbrückung der physikalischen Grenze der Länge eines Netzsegmentes (des Abstandes zwischen zwei benachbarten Geräten im Netz) benutzt.

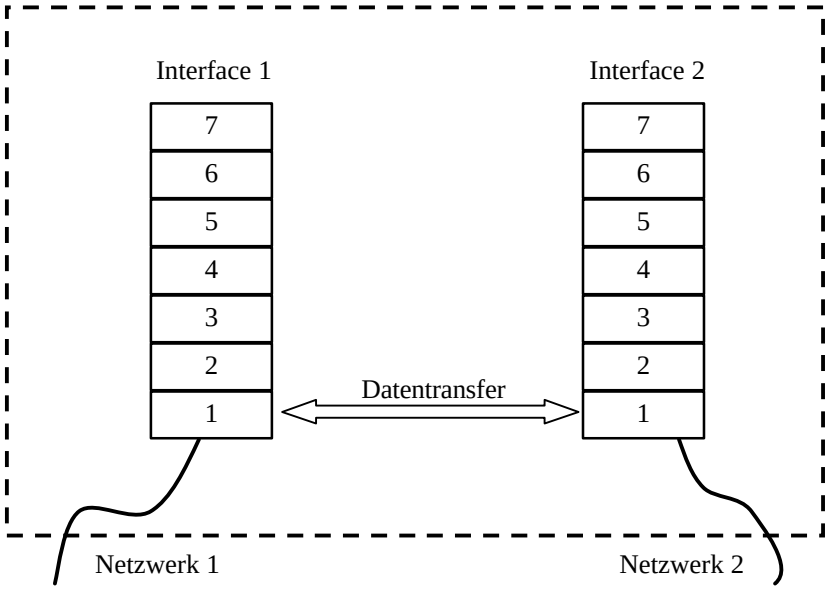


Bild 2.7. Aufbau und Funktionsweise des Repeaters

2.3.2. Bridge

Die Bridges dienen zum Verbinden von Netzen, die zwar ein und dasselbe Protokoll für die Datenübertragung, aber dafür verschiedene Übertragungsmedien (Twisted pair, Funk, LWL oder andere) nutzen. Im Unterschied zu den Repeatern haben die Bridges einen kleinen Softwareteil, der für den Datentransfer von einer Schnittstelle zur anderen verantwortlich ist. Dieser kann auch einfachere Funktionen für die Filterung auf der Basis von MAC-Adressen von Datenpaketen enthalten, um unnötige Transfers von Daten zu verhindern – wenn zum Beispiel der Datenempfänger sich nicht im über die Bridge angeschlossenen Netz befindet. Als Folge dessen arbeiten die Bridges auf der zweiten Schicht des OSI Schichtenmodells (Bild 2.8).

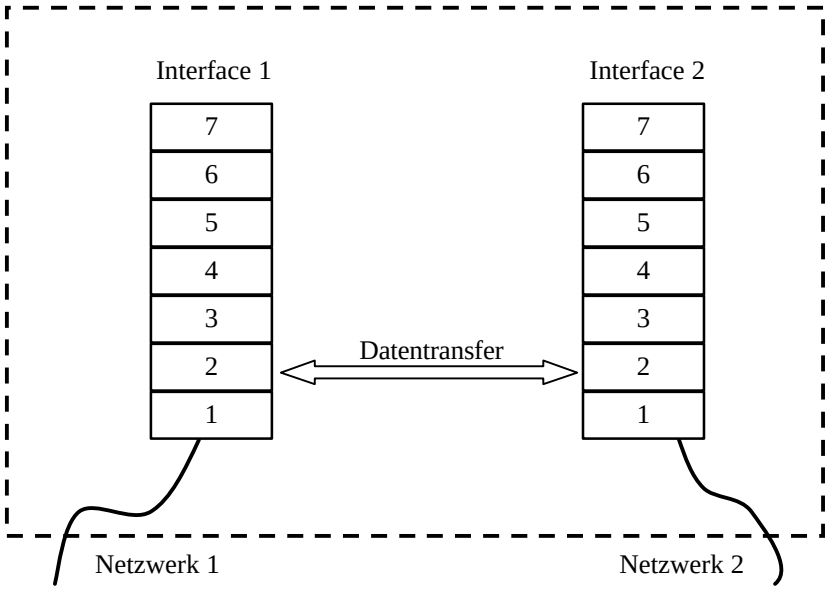


Bild 2.8. Aufbau und Funktionsweise der Bridge

Eine besondere Art der Bridge stellt ein Switch (Bild 2.9) dar. Er wird für das Verbinden verschiedener Segmente von gleichartigen Netzen benutzt. Ein Switch hat in der Regel mehrere Schnittstellen, an jeder von ihnen wird nur ein Netzteilnehmer angeschlossen. Ausgehend von der MAC Adresse des Datenpaketes bestimmt der Softwareteil des Switches, an welche Schnittstelle es geschickt wird, oder aber einfach ignoriert wird.

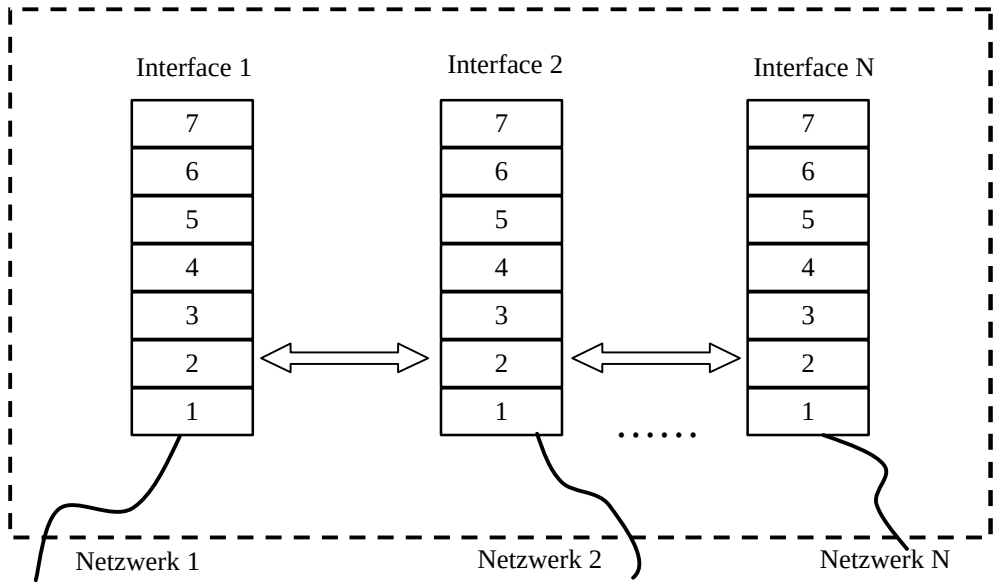


Bild 2.9. Aufbau und Funktionsweise des Switches

2.3.3. Router

Der Router arbeitet auf der dritten Schicht des OSI Schichtenmodells (Bild 2.10) und dient zur Übertragung der Daten zwischen Sender und Empfänger über mehrere Netze, unabhängig von ihrer Art. Als Folge davon ist der Softwareteil des Routers in der Lage die Informationen über mögliche Übertragungswege der Daten und ihrer Effektivität (Übertragungszeit, Übertragungskosten, u.a.) dynamisch zu sammeln. Um dieses Ziel zu erreichen, kann der Router mit Hilfe eines speziellen Protokolls mit anderen Routern im Netz kommunizieren, um den optimalen Übertragungsweg zu finden.

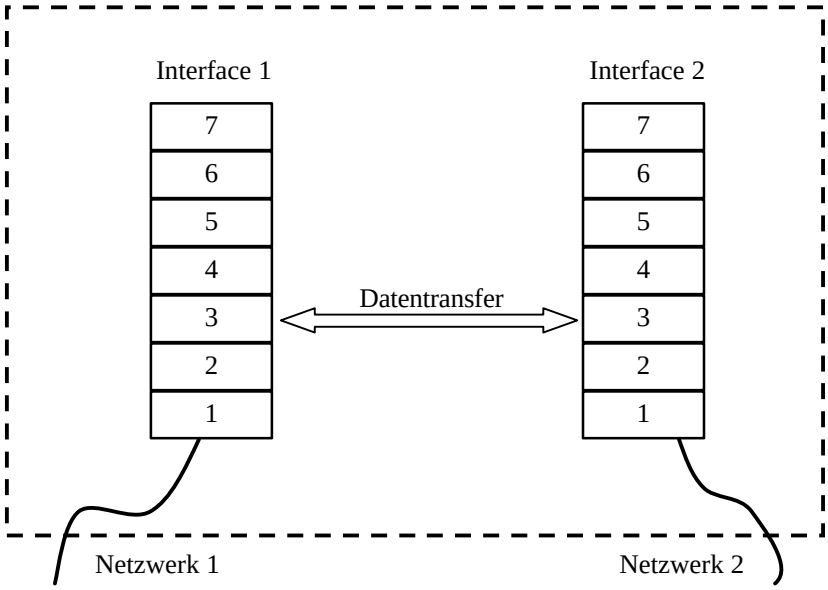


Bild 2.10. Aufbau und Funktionsweise des Routers

2.3.4. Gateway

Eines der komplexesten Geräte für das Verbinden von verschiedenen Netzen ist ein Gateway. Es dient dazu, zwei Netze mit völlig unterschiedlicher Architektur auf der siebten Schicht des OSI Schichtenmodells zu verbinden (Bild 2.11). Das Gateway hat in der Regel zwei Hardwareschnittstellen und seine Software beinhaltet komplette Implementierungen des Protokollstacks der beiden Netze – und dies macht das Gateway so komplex. Das Gateway arbeitet mit den Anwenderdaten und sammelt und trennt sie von den verschiedenen Netzwerkinformationen (Adresse des Empfängers und des Senders, benutzter Netzwerkservice u.a.).

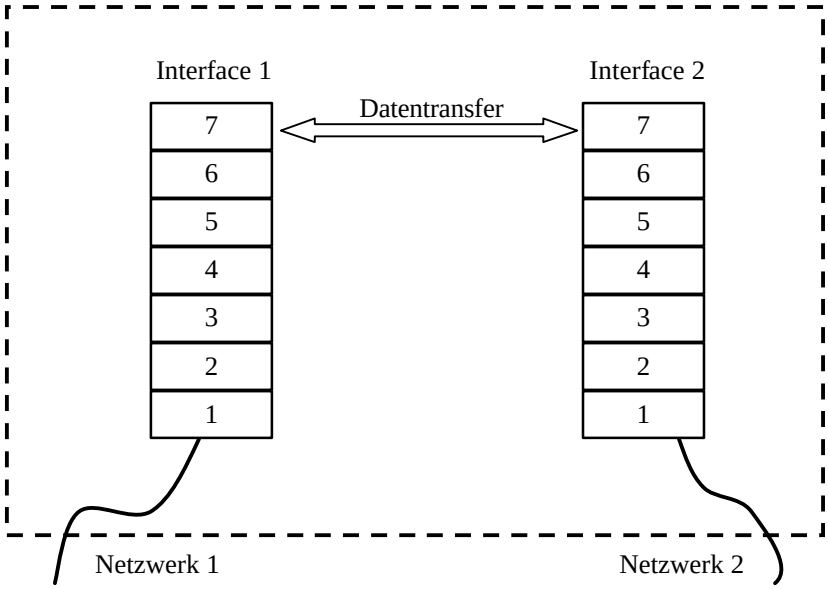


Bild 2.11. Aufbau und Funktionsweise des Gateways

2.4. Marktverfügbare Lösungen für die Zusammenarbeit von verschiedenen Bussystemen in der Heimautomatisierung

Im Bereich der Heimautomatisierung werden ausschließlich Gateways eingesetzt. Folglich kommt es zu gravierenden Unterschieden in der Struktur der Automatisierungsnetze und der Datenübertragungsnetze.

In den nächsten Unterkapiteln werden die verschiedenen, auf dem Markt verfügbaren Geräte für das Zusammenschließen verschiedener Bussysteme betrachtet. Dabei wird das größte Augenmerk auf die meist herstellerspezifischen Besonderheiten des jeweiligen Gerätes gerichtet, die für die Zusammenarbeit im System, für die Konfigurierung des Gerätes und für die Visualisierung der vom Gerät gelieferten Daten relevant sind.

2.4.1. EIB – BACnet Gateway

Dieses Gateway, welches von der Firma Amann GmbH in Oberhaching hergestellt wird, erlaubt es, den EIB mit dem BACnet (Building Automation and Control Network) – einem nationalen nordamerikanischen Standard im Bereich der Gebäudeautomatisierung – zu verbinden. Es wird in zwei Varianten als „EWMS I Gateway“ (Bild 2.12) [21] und als „EWMS II Gateway“ (Bild 2.13) [22] produziert. Diese unterscheiden sich in der verschiedenen Hardwareausstattung und somit in der unterschiedlichen Anzahl der unterstützten Datenpunkte oder Objekte (bis zu 1000 bzw. 1500 Datenpunkte).

Bild 2.12. EWMS I Gateway

(Quelle: Amann)

Bild 2.13. EWMS II Gateway

(Quelle: Amann)

Sie unterscheiden sich ebenfalls in der Art ihrer Konfigurierung. Während „EWMS I“ ein spezielles Konfigurationsprogramm benötigt, lässt sich „EWMS II“ dank des eingebauten Web-Servers aus jedem beliebigen Web-Browser konfigurieren.

2.4.2. EIB – ISDN Gateway

Es gibt einige Firmen, die sich mit der Herstellung von Gateways zum Verbinden des EIB mit dem ISDN (Integrated Services Digital Network) – das Netz des digitalen Telefonierens – beschäftigen. Zu ihnen gehören: Siemens (Abteilung Automation & Drives) (Bild 2.14) [23], ASTON GmbH in Oberhausen (Bild 2.15) [24], FS SoftwareConsult in Meinerzhagen u.a.

Bild 2.14. ISDN-Schnittstelle der FA. Siemens

(Quelle: Siemens)

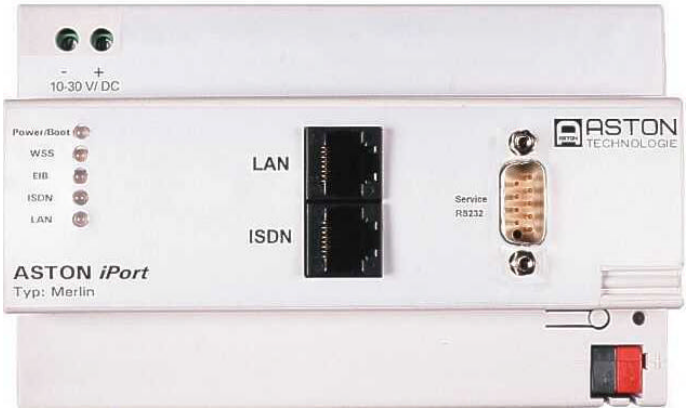


Bild 2.15. ASTON iPort MAX - ISDN

(Quelle: ASTON)

Diese Gateways ermöglichen dem Anwender den Zugriff auf das im Gebäude installierte EIB Automatisierungsnetz über das öffentliche Telefonnetz. Dieser Zugriff hat einen zweiseitigen Charakter, das heißt er erlaubt nicht nur dem Einwohner sein Haus, sondern auch dem Haus bestimmte Nummern anzurufen, oder SMS an sie zu schicken, um eine Veränderung des Normalzustandes aufzuzeigen oder einen Notruf zu betätigen. Ein solches Gateway kann ebenfalls für das Verbinden von zwei getrennten EIB-Installationen mit Hilfe von ISDN verwendet werden.

Dabei ist die ISDN-Schnittstelle der Fa. Siemens vornehmlich für die Aufgaben des Fernzugriffs zur Überwachung und Rekonfigurierung von EIB-Netzen ausgelegt. Als Folge beinhaltet sie einen eingebauten Web-Server und unterstützt das iETS Zugriffsprotokoll der iETS Software zur Konfigurierung von EIB-Netzen. Es verwaltet deshalb auch nur 64 Kommunikationsobjekte. Eine interessante Eigenschaft des Gateways ist die Spracherkennung/Generierung, die es erlaubt, Sprachnachrichten zu verschicken oder EIB-Geräte zu steuern.

„iPort MAX – ISDN“ der Fa. ASTON ist wiederum eher für die Fernsteuerung und Fernvisualisierung von EIB-Installationen ausgelegt. Es unterstützt bis zu 65534 unterschiedliche Datenpunkte/Objekte. Für die Konfigurierung wird eine proprietäre Software mit dem Zugriff über eine lokale serielle Schnittstelle des Gateways benutzt.

2.4.3. EIB – IP Gateway

Zu einer der weitverbreitetsten Gruppen gehören Gateways, die die Automatisierungsnetze mit dem IP-Netz verbinden. Die Ursache dafür ist offensichtlich: die relative Kostengünstigkeit und Verbreitung der IP-Netze. So z.B. werden diese Geräte von folgenden Firmen hergestellt, Siemens [25] (Bild 2.16), IPAS GmbH [26] (Bild 2.17) und viele andere.

Diese Geräte bieten dem Anwender die Möglichkeit des automatischen Monitorings bestimmter Ereignisse im EIB-Netz und die Übertragung von Informationen darüber ins IP-Netz. Ebenfalls bieten sie die Möglichkeit, auf das EIB-Netz zuzugreifen. Die Fortschrittlicheren unter ihnen stellen dem Anwender auch einen iETS-Server zur Verfügung. Dies erlaubt aus

der Ferne die Diagnosestellung, das Debugging und die Rekonfigurierung von EIB-Netzen mit Hilfe der Standardsoftware ETS (siehe Anhang A).

Bild 2.16. IP-Schnittstelle der Fa. Siemens

(Quelle: Siemens)

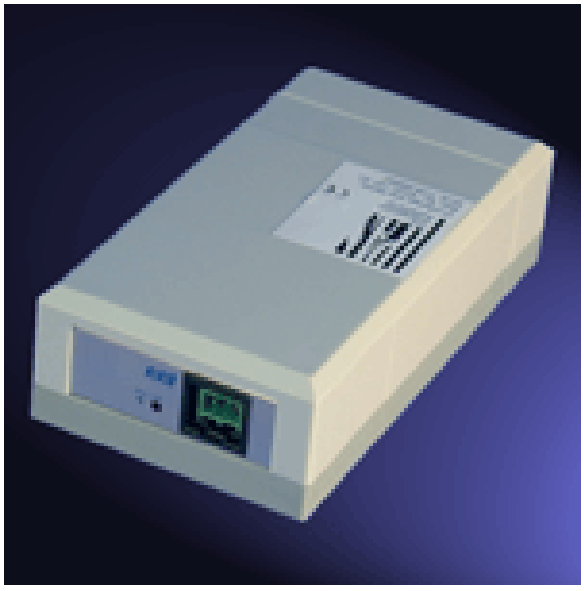


Bild 2.17. „EIB/IP Combrige“ der Fa. IPAS GmbH

(Quelle: IPAS)

2.4.4. LON – IP Gateway

Genauso wie bei den EIB-Netzen, gibt es eine große Vielfalt von LON – IP Gateways, einige von ihnen kann man z.B. in der Datenbank der LON-Nutzer-Organisation in Deutschland [27] finden. Es werden zwei dieser Geräte genauer betrachtet: „LON Works Bridge“ [28] der Fa. FieldServer Technologies aus den USA (Bild 2.18) und „LON Internet Connector 1000“ der Fa. SVEA Building Control Systems GmbH & Co. aus Hamburg (Bild 2.19) [29].

Bild 2.18. FieldServer LonWorks Bridge

(Quelle: FieldServer Technologies)

Bild 2.19. LON Internet Connector 1000

(Quelle: SVEA Building Control Systems)

Diese Geräte erlauben den Zugriff auf die LON-Geräte aus dem IP Netz, dabei werden unterschiedliche proprietäre Protokolle eingesetzt. Diese Gateways haben in der Regel einen eingebauten Web-Server für den Fernzugriff über den Standard Web-Browser. Die Konfigurierung wird über die proprietäre serielle Schnittstelle der Geräte vorgenommen. Beide Gateways können das OPC Protokoll unterstützen, aber das Gateway der Fa. SVEA bietet außerdem eine CORBA-Schnittstelle an.

2.4.5. CAN Gateways

Trotz der großen Verbreitung von CAN-Systemen ist im Vergleich zu Gebäudeautomatisierungssystemen eine geringere Vielfalt an CAN-Gateways auf dem Markt verfügbar. Die am meisten verbreiteten sind entweder einfache Protokollumsetzer von CAN zur serielle Schnittstelle RS232 des PC, so wie z.B. CAN232 (Bild 2.20) der Fa. LAWICEL aus Schweden [30] oder ein CAN-IP Gateway, wie z.B. CAN@net (Bild 2.21) der Fa. IXXAT Automation GmbH aus Weingarten [31]. Das letztere bietet einen weltweiten Zugriff auf das angeschlossene CAN Netz oder aber es erlaubt zwei räumlich getrennte CAN Netze über das TCP/IP oder über das eingebaute GSM Modem zu verbinden.

Bild 2.20. CAN232 der Fa. LAWICEL

(Quelle: LAWICEL)

Bild 2.21. CAN@net der Fa. IXXAT Automation GmbH

(Quelle: IXXAT Automation)

2.5. Schlussfolgerungen

Im Allgemeinen lässt sich anmerken, dass es zurzeit eine große Anzahl von unterschiedlichen Lösungen für das Verbinden von verschiedenen Systemen gibt. Diese Realisierungen stellen ausnahmslos Lösungen für ein bestimmtes System dar, die nur von angepasster Software unterstützt werden. Daher ist es nicht verwunderlich, dass diese Mittel hauptsächlich die Sicht der Hersteller des jeweiligen Systems oder Software darstellen. Ein besonders schwacher Punkt ist dabei der Datenaustausch zwischen den einzelnen Lösungen. Obwohl hier das Bestreben zum Einsatz einer allgemein gültigen Beschreibungssprache (so z.B. XML) beobachtet wird, so hat es sich noch lange nicht überall durchsetzen können.

Andererseits erlaubt das Entwicklungsniveau moderner Hardware- und Softwarewerkzeuge, das Erstellen eines Konzepts und die Realisierung von universellen Mitteln zum Zusammenführen mehrerer existierender Untersysteme zu einem logischen Managementnetz. Dabei muss man die konsequente Verwendung der üblichen objektorientierten modularen Technologien und offen verfügbarer Lösungen aus ähnlichen Einsatzgebieten (wie z.B. lokale Netze) anstreben. Ausgehend davon wird auch das Ziel zur Unterstützung von Maßnahmen zur Vereinheitlichung der unterschiedlichen Systeme als wesentlich betrachtet. Dabei wird im gleichen Maße wie die Standardisierung auch die Definition der systemunabhängigen Profile eine entscheidende Rolle spielen.

Für die weitere Betrachtung eines universellen Managementsystems ist es notwendig, zuerst basierend auf den bereits vorhandenen Technologien und Lösungen ein passendes Konzept auszuarbeiten. Es wird später als Grundlage für den konkreten Realisierungsentwurf des Managementsystems dienen.

3. Systemkonzept

Wie schon im Kapitel 1.2 formuliert wurde, ist das Ziel der vorliegenden Arbeit nicht die Entwicklung eines vollkommen neuen Managementsystems für das private Heim gewesen, sondern eher der Versuch, ein einheitliches flexibles Managementsystem zu schaffen, indem alle im Haus installierten Netze zu einem logischen System vereinigt werden.

Für den Aufbau eines solchen Systems wird meistens eine von den zwei folgenden Lösungen verwendet. Bei der ersten Lösung werden alle existierenden Systeme an ein „großes“ Gerät (welches „mächtiges“ Gateway genannt wird) angeschlossen, das alle notwendigen physikalischen Schnittstellen für den Zugriff auf jedes System beinhaltet und einen internen softwarebasierten Datenaustausch zwischen den einzelnen Systemen durchführt. Bei der zweiten Lösung wird jedes der installierten Systeme an ein eigenes Gateway (welches „schlankes“ Gateway genannt wird) angeschlossen. Diese Gateways werden in der Regel mit einem eigenen Netz (dies könnte auch eines der schon installierten Netze sein) miteinander verbunden. In diesem Netz findet der Datenaustausch zwischen den einzelnen Untersystemen statt und es wird häufig als „Backbone“ bezeichnet.

Beide Lösungen haben sowohl Vorteile wie auch Nachteile, die wir nun aus Sicht unseres Anwendungsfalles betrachten werden. Zu den Vorteilen der Lösung mit einem Gateway kann man die relative Günstigkeit und wenige Schwierigkeiten beim Austesten und bei der Wartung rechnen. Zu den Nachteilen einer solchen Lösung gehören die Schwierigkeiten bei der Nachrüstung, weil nicht in jeder Installation alle Systeme an einer Stelle physikalisch präsent sind; Schwierigkeiten bei der Systemerweiterung oder bei der Änderung der Systemkonfiguration wegen des nicht modularen Aufbaus der Lösung, infolge des letzteren auch ein relativ hoher fixierter Einstiegspreis; ein relativ komplizierter Gatewayaufbau und als Folge davon fehlende Möglichkeiten, das Gateway durch den Anwender zu warten. Die Lösung mit mehreren „schlanken“ Gateways hat folgende Vorteile: ein modularer Systemaufbau, eine unkomplizierte Installation und Nachrüstung, eine einfache Erhöhung der Zuverlässigkeit des Systems durch die Redundanz der Systemgeräte, ein niedriger und flexibler Einstiegspreis. Zu den Nachteilen gehören ein etwas höherer Preis der gleichen Komplettlösung im Vergleich zum System mit einem „mächtigen“ Gateway, und auch die Schwierigkeiten beim Austesten und bei der Suche nach Fehlern in den Systemkomponenten, weil diese im System „verstreut“ sind.

Ausgehend von den in Kapitel 1.2 definierten Zielen dieser Arbeit und von den oben beschriebenen Eigenschaften der einzelnen vorgeschlagenen Lösungen wird das hier beschriebene Konzept des Managementsystems für das private Heim auf der Lösung mit mehreren „schlanken“ Gateways und einem Backbone basieren.

Generell ist ein Managementsystem mit einem derartigen Aufbau ein zentrales System, obwohl einzelne Untersysteme dezentral sein können. Die Funktionalität jedes angeschlossenen Untersystems hängt in keiner Weise vom Backbone ab, d.h. die Untersysteme funktionieren autonom. Das System macht es möglich, die Funktionalität der einzelnen Untersysteme sogar nach dem Ausfall des Backbones zu erhalten, wenn auch nicht in dem Umfang, den das gesamte Managementsystem bietet.

In einem solchen System ist keine direkte Kommunikation der Geräte aus einem Untersystem mit Geräten aus einem anderen Untersystem zulässig. Die Geräte aus verschiedenen Untersystemen ahnen nicht einmal voneinander. Der Informationsaustausch zwischen den einzelnen Untersystemen findet nur über den Backbone statt, der in diesem Fall ein „Supervisor” ist, d.h. nur ihm ist bekannt – welche Information in welches System übertragen werden muss.

Ein solches Modell kann man in der Heimautomatisierung nicht nur für neue Installationen verwenden, sondern auch für den kosteneffektiven Zusammenschluss vorhandener Untersysteme bei der Modernisierung bzw. Nachrüstung.

Im Gegensatz zur einfachen Definition des Backbones stellt er in dieser Arbeit einen Satz von Hardware- und Softwarekomponenten dar und beinhaltet somit nicht nur die physikalische Verbindung der Hardwarekomponenten, sondern enthält ebenfalls verteilte Funktionalitäten, um die Aufgaben für die Heimautomatisierung zu erfüllen. Nun müssen noch diese einzelnen Hardware- und Softwarekomponenten ausgegrenzt und definiert werden, um das Managementsystem flexibel und effektiv zu gestalten.

Für die Funktionalität des Gesamtsystems muss man nicht nur die physikalische (Hardware-) Anpassung der Signale der einzelnen Untersysteme und die logische (Software-) Anpassung ihrer Protokolle vornehmen, sondern auch ein einheitliches Protokoll für den Zugriff auf jedes Untersystem aus dem Backbone festlegen. Die Ausführung dieser Aufgaben übernimmt ein „*Services*-Gateway“ (Bild 3.1), das im Kapitel 3.1 beschrieben wird.

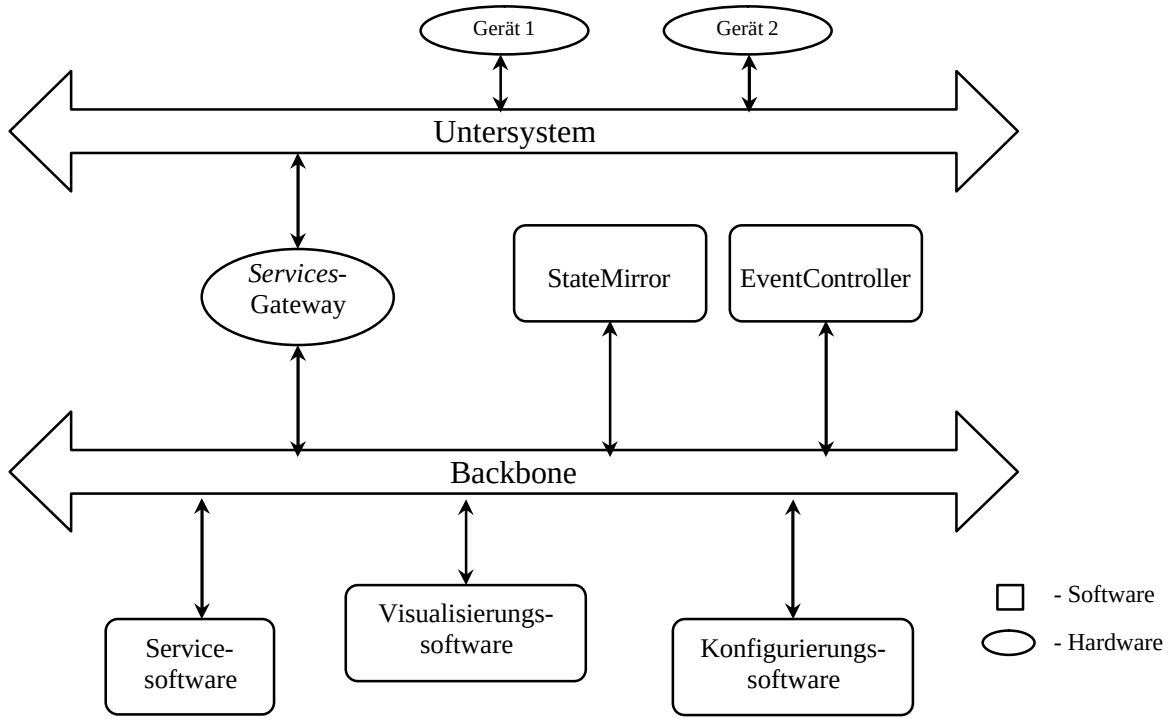


Bild 3.1. Anbindung eines Untersystems an den Backbone

Ein weiterer wichtiger Erfolgsbestandteil eines solchen Managementsystems ist eine universelle Software, welche die Funktionsweise der einzelnen Untersysteme und ihre Software- und Hardwarebesonderheiten für den Anwender unsichtbar macht. So wird, zum Beispiel, ein Schalter für den Benutzer als ein Modul mit einem Ausgang und zwei möglichen Zuständen – „Ein“ oder „Aus“ dargestellt, unabhängig davon, in welchem Untersystem der Schalter installiert ist und welche Telegramme er bei der Änderung seines Zustandes dort überträgt.

Die Software des Systems (Kapitel 3.4) lässt sich erfahrungsgemäß in drei Gruppen aufteilen:

- Die Konfigurationssoftware, beschrieben im Kapitel 3.4.1, führt die Konfigurierung der installierten Systemkomponenten und im Idealfall die Konfigurierung der einzelnen Untersysteme durch
- Die Visualisierungssoftware, Kapitel 3.4.2, gibt dem Nutzer die Möglichkeit verschiedene Größen und Prozesse in den Untersystemen zu beobachten, und auch ihre Werte zu ändern bzw. sie zu steuern
- Servicesoftware, Kapitel 3.4.3, erlaubt dem Anwender die aufgetauchten Fehler oder Störungen in der Arbeit des Systems oder der einzelnen Untersysteme zu lokalisieren, den Funktionstest der Geräte bzw. Services oder die Fehlersuche in den Geräten bzw. Services zu unterstützen, unterschiedliche statistische Informationen (Buslast, Nutzzeit der einzelnen Geräte u.a.) zu sammeln

Im Kapitel 3.2 wird die Softwarekomponente „*StateMirror*“ beschrieben, die in den Untersystemen mit den sog. Prozessvariablen ein Proxy[1] darstellt, d.h. sie speichert den letzten, auf dem Bus übertragenen Wert der Prozessvariablen. Bei der Anfrage des Variablenwertes vom Client prüft der *StateMirror* zunächst seinen Puffer. Falls der Wert dort nicht vorhanden ist, fordert er ihn über den Bus an, anderenfalls liefert er einen Wert aus dem Puffer. So verkürzt der *StateMirror* die Antwortzeit und senkt die Buslast.

Die letzte enthaltene Softwarekomponente ist ein «*EventController*» (Kapitel 3.3), der alle notwendigen Mittel enthält, um die Reaktionen auf die Ereignisse im Untersystem durchzuführen. Dies kann eine Übertragung des Telegramms im Untersystem, aber auch das Versenden einer E-Mail sein.

3.1. *Services*-Gateway

Wie man aus Bild 3.1 deutlich erkennen kann, ist das *Services*-Gateway ein Schlüsselelement des Systems. Es ist die einzige Systemkomponente, die physikalisch ein beliebiges Untersystem mit dem Backbone verbindet. Als Folge hat ein *Services*-Gateway immer zwei Hardwareinterfaces, um sowohl im Backbone wie auch im Untersystem kommunizieren (senden/empfangen von Nachrichten) zu können. Softwaremäßig stellt das *Services*-Gateway ein an ihm angeschlossenes Untersystem im Backbone als einen Satz von Services dar.

Bei der Definition dieser Services gilt es, einen gemeinsamen Nenner der in Frage kommenden existierenden Systeme und eventuell der in Zukunft neu zu entwickelnden Systeme zu finden. Dabei lassen sich bestimmt mehrere Ansätze verfolgen. In der vorliegenden Arbeit wurde davon ausgegangen, dass ein Protokollstack von allen mehr oder weniger modernen und höchst wahrscheinlich auch von den neu zu entwickelnden Systemen auf dem sog. OSI (Open System Interconnection) Schichtenmodell (Tabelle 3.1) [19,20] basiert. Das Modell wurde von der internationalen Organisation für die Standardisierung (ISO) empfohlen.

[1] Proxy – eng. Stellvertreter, wird in der Informationstechnik als Bezeichnung für einen Puffer zwischen dem Client und Server verwendet, der strukturierte Inhalte zwischenspeichert. Siehe z.B. Proxy-Server.

Tabelle 3.1. ISO/OSI Schichtenmodell

Layer	Kurze Beschreibung	Austauschelement
Physical	Verbindet das Gerät mit dem Bus. Bestimmt physikalische und elektrische Eigenschaften des Systems	Bits
Link	Bestimmt Zugriffs- und Teilungsstrategie. Korrektur von Übertragungsfehlern	Frames
Network	Gewährleistet Öffnung, Unterstützung und Schließung der Verbindungen. Leitet die Information im System	Datagrams
Transport	Gewährleistet Zuverlässigkeit und Integrität der Daten. Ermöglicht „Punkt-zu-Punkt“ Verbindungen	Packets
Session	Steuert den Informationsaustausch zwischen zwei Objekten. Beseitigt Fehler der nicht-kommunikativen Art.	Messages
Presentation	Wandelt die Information um (Packen/Entpacken, Verschlüsseln/Entschlüsseln u.a)	Messages
Application	Bietet verschiedene Services für Applikationen/Anwender	Messages

Somit wurden alle Services in acht Gruppen aufgeteilt:

- LinkLayer services
- NetworkLayer services
- TransportLayer services
- PresentationLayer services
- SessionLayer services
- ApplicationLayer services
- UserLayer services
- Auxiliary services

Wie man leicht erkennen kann, entsprechen die ersten sechs Gruppen den jeweiligen Schichten des Schichtenmodells. Hier fehlt nur der physikalische Layer, weil die Manipulation von Informationen auf der Bitebene für diese Anwendung nicht interessant ist.

Jede Gruppe enthält die Services für die Kommunikation auf der entsprechenden Ebene im Untersystem. So zum Beispiel, wenn in einem Untersystem eine Kommunikation auf dem Application Layer definiert ist, kann (soll) ein *Services*-Gateway die entsprechenden Applica-

tionLayer-Services zur Verfügung stellen. Wenn die Kommunikation im Untersystem auf irgendeinem Layer nicht definiert ist, oder die Services des Layers im *Services*-Gateway nicht implementiert sind, bleibt die entsprechende Gruppe der Services leer.

Die Gruppe „UserLayer services" enthält die Services, die mehrere Aufrufe von anderen Services benötigen oder zeitkritisch sind. Die letzte Gruppe „Auxiliary services" enthält alle Services, die zu keiner anderen Gruppe passen.

Für die optimale und zuverlässige Arbeit des Gesamtsystems muss das Zugriffsprotokoll des *Services*-Gateways aus dem Backbone folgendes gewährleisten:

- Eine Betriebsart mit einer festen Adresse im Backbone oder mit ihrer automatischen Zuteilung bei der Inbetriebnahme
- Das automatische Entdecken der installierten Services-Gateways im Backbone
- Authentifizierung von Clients, Zuteilung von Zugriffsrechten
- Versenden der Beschreibung von unterstützten Services auf Anfrage vom Client
- Bidirektionaler asynchroner Informationsaustausch mit dem Client

3.2. *StateMirror*

Wie schon erwähnt wurde, ist in den Untersystemen mit Prozessvariablen der *StateMirror* ein Proxy, der die letzten Variablenwerte speichert. Damit verkürzt er die Antwortzeiten und senkt die Buslast. Der *StateMirror* arbeitet eng mit dem *Services*-Gateway zusammen. (Bild 3.2)

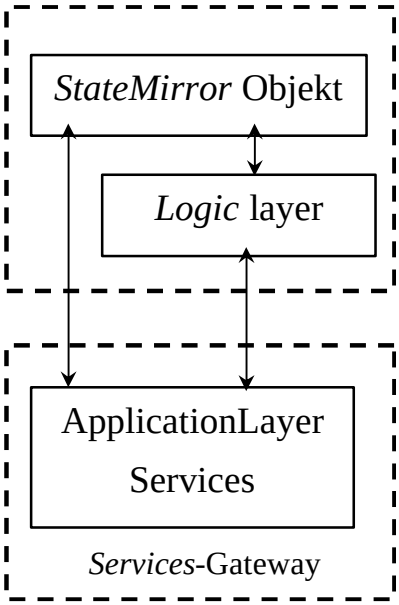


Bild 3.2. Zusammenarbeit von *StateMirror* und *Services*-Gateway

Ein wichtiger Teil des *StateMirrors*, den es in dieser Form bisher nicht gibt, ist die Einführung einer Verknüpfungsebene, des sog. „*LogicLayer*", der logische Prozessvariablen verwal-

tet. Diese Variablen existieren nicht physikalisch in einem Untersystem, sondern werden durch logische und mathematische Operationen und aus den physikalischen Prozessvariablen gebildet. In Bild 3.3 sind einige solcher Variablen zu sehen.

LogicalVariable1 = (~ProzessVariable1) | (ProzessVariable2 & ProzessVariable3)

LogicalVariable2 = (ProzessVariable4 + ProzessVariable5) * ProzessVariable6

LogicalVariable3 = (ProzessVariable7 < 100) & ($TIME > 20:00)

Bild 3.3. Logische Prozessvariable

Der *StateMirror* muss für die optimale Ausführung seiner Aufgaben folgende Funktionen beinhalten:

- die installierten *Services*-Gateways automatisch entdecken, sich in sie einloggen und ihre Services abonnieren können.
- die Datenbank der logischen und physikalischen Prozessvariablen verwalten und einen Lese/Schreib-Zugriff für den Client bieten
- die Clients und ihre Zugriffsrechte authentifizieren
- den „Multisession“ Betrieb unterstützen. D.h. der *StateMirror* sollte in der Lage sein, mit einer theoretisch unbegrenzten Anzahl von *Services*-Gateways gleichzeitig zusammenzuarbeiten und eine theoretisch unbegrenzte Anzahl von Clients gleichzeitig zu bedienen. (Bild 3.4)

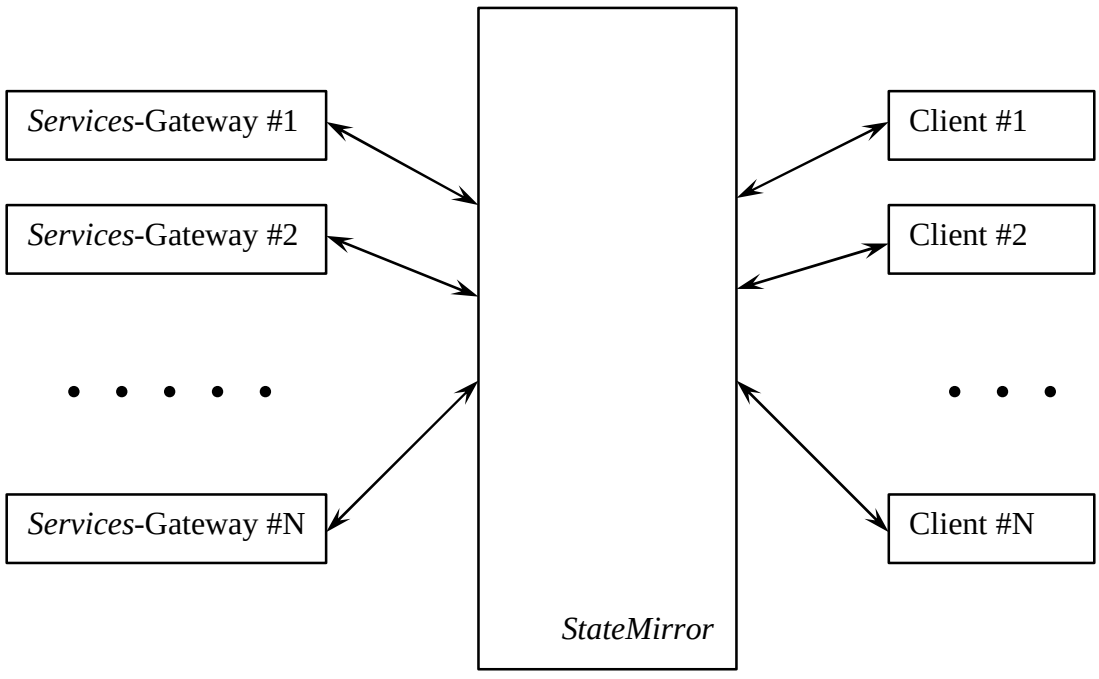


Bild 3.4. “Multisession“ Betrieb des *StateMirrors*

3.3. *EventController*

Der *EventController* ist verantwortlich für die Ausführung von verschiedenen „Aktionen“ als Reaktion auf die Ereignisse im Untersystem. Um diese Komponente nicht zu komplex werden zu lassen und zur Vereinheitlichung ihrer Schnittstelle wurden folgende Voraussetzungen getroffen:

1) Der *EventController* kann nur mit dem *StateMirror* kommunizieren, d.h. er kann nicht direkt mit dem *Services*-Gateway arbeiten
2) Der *EventController* stellt sich für den Anwender als eine Bibliothek von Komponenten dar, jede von ihnen ist nur für eine bestimmte „Handlung“ verantwortlich
3) Jede der Komponenten hat nur einen Booleschen Eingang. Beim Eintreffen des Wertes „wahr“ wird die entsprechende Handlung ausgeführt, beim Eintreffen des Wertes „falsch“ passiert nichts.

So zeigt die in Bild 3.5. vorgestellte Zusammenarbeit zwischen dem StateMirror und dem EventController zum Beispiel die Ausführung einer Funktion zum Ausschalten des Lichtes in der Abwesenheit von Personen im Zimmer. Dafür wird das „Aktion“-Modul „SendMessageToBus“ verwendet. Dieses Modul wird vom StateMirror über eine Änderung der logischen Variable „Variable 1“ benachrichtigt. Die Variable verknüpft ihrerseits zwei physikalische Variablen (Zustand der Zimmerbeleuchtung und Anzahl der anwesenden Personen) und liefert den Wert „wahr“ wenn das Licht im Zimmer eingeschaltet ist und keine Personen anwesend sind. Darauf reagiert das Modul mit dem Setzen der physikalischen Variablen „Variable 2“, die für die Beleuchtung im Zimmer zuständig ist, in den Zustand „AUS“, was wiederum zum Ausschalten der Zimmerbeleuchtung führt.

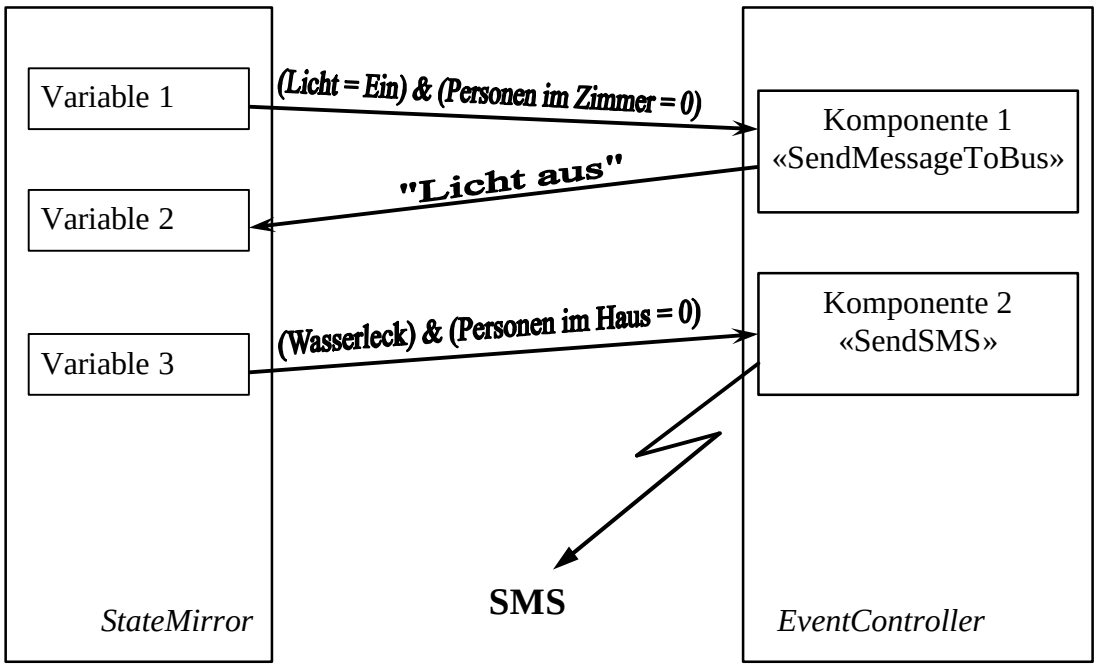


Bild 3.5. Funktionsweise des *EventControllers*

Ein anderes in Bild 3.5. dargestelltes „Aktion“-Modul „SendSMS“ reagiert auf das Eintreffen des Wertes „wahr“ an seinem Eingang mit dem Versenden einer SMS-Nachricht. Dieses Modul kann für die Reaktion auf kritische Ereignisse (Alarme) im Managementsystem eingesetzt werden.

3.4. Software des Systems

Da das hier beschriebene Systemkonzept sich an den privaten und so meist technisch nicht versierten Anwender wendet, spielt eine bedienerfreundliche und intuitiv verständliche Benutzeroberfläche (siehe [32,33]) eine große Rolle in der Akzeptanz der Systemsoftware. Das bedeutet, dass man nicht erwarten kann, dass ein Anwender über ausreichend technische Kenntnisse über den internen Aufbau und die Funktionsweise des Systems und der einzelnen Komponenten verfügt. Und trotzdem sollte er mit Hilfe der Bedienoberfläche mit entsprechenden Elementen des Mensch-Maschine-Interfaces [34] in der Lage sein, das System zu konfigurieren bzw. zu steuern. Als ein gutes Beispiel für ein solches Interface kann man die integrierte Entwicklungsumgebung „LabView“ (Bild 3.6) der Fa. Texas Instruments nennen. Um dort das notwendige Programm zu entwickeln, braucht der Anwender keine Kenntnisse über irgendwelche Programmiersprachen haben. Er zieht einfach die notwendigen Module per „Drag&Drop“ aus einem Bibliotheksfenster in sein Programmfenster und verbindet sie zu einem Funktionsdiagramm.

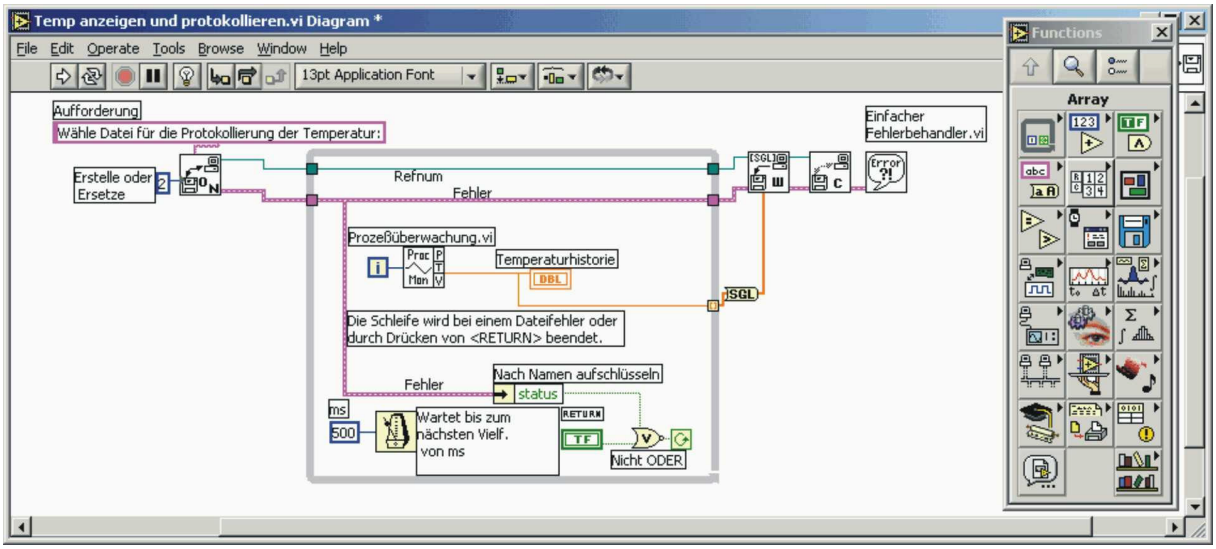


Bild 3.6. Integrierte Entwicklungsumgebung „LabView“

Kleine Änderungen in der Funktionsweise der Module oder ihres Zusammenspiels kann man über die Eigenschaften des Moduls bzw. der Verbindung vornehmen. Nach dem „Zusammenklicken“ des Programms drückt der Anwender den Knopf „Ausführen“ und kann schon nach kurzer Zeit die Ergebnisse seiner Arbeit betrachten. Diese Art Programme zu entwickeln wird oft als visuelle Programmierung [35,36] bezeichnet.

Eine weitere wichtige Eigenschaft des Systems muss seine Flexibilität und Erweiterungsfähigkeit sein. Das kann man in der Regel mit einem modularen Aufbau der Systemssoftware erreichen. Ein Beispiel für einen solchen Aufbau ist in Bild 3.7 zu sehen. Die Systemsoftware

stellt einen unveränderbaren Kern des Systems dar, der mit Hilfe der dynamisch ladbaren Module an die Bedürfnisse der jeweiligen konkreten Installation des Systems angepasst werden kann. Dabei werden gleichartige Module in den entsprechenden Bibliotheken zusammengefasst. So sind in Bild 3.7 die Bibliothek der Untersysteme und die Gerätebibliothek der einzelnen Untersysteme dargestellt, die hauptsächlich von der Konfigurationssoftware des Systems benutzt werden. Ebenso sind dort die Bibliothek der Visualisierungs- und Steuerungskomponenten für die Visualisierungs- und Steuerungssoftware des Systems und die Bibliothek der „Aktion“-Module für den *EventController* zu sehen. Ein solcher Aufbau der Systemsoftware ermöglicht Module von Drittanbietern (je nach Komplexität entwickelt von großen Firmen und Verbänden, oder von kleineren Firmen, oder vom Endverbraucher) ins System zu integrieren und damit den Marktwert des Systems deutlich zu steigern, weil in dem Fall die Weiterentwicklung des Systems nicht von einer Firma oder Institution abhängig ist.

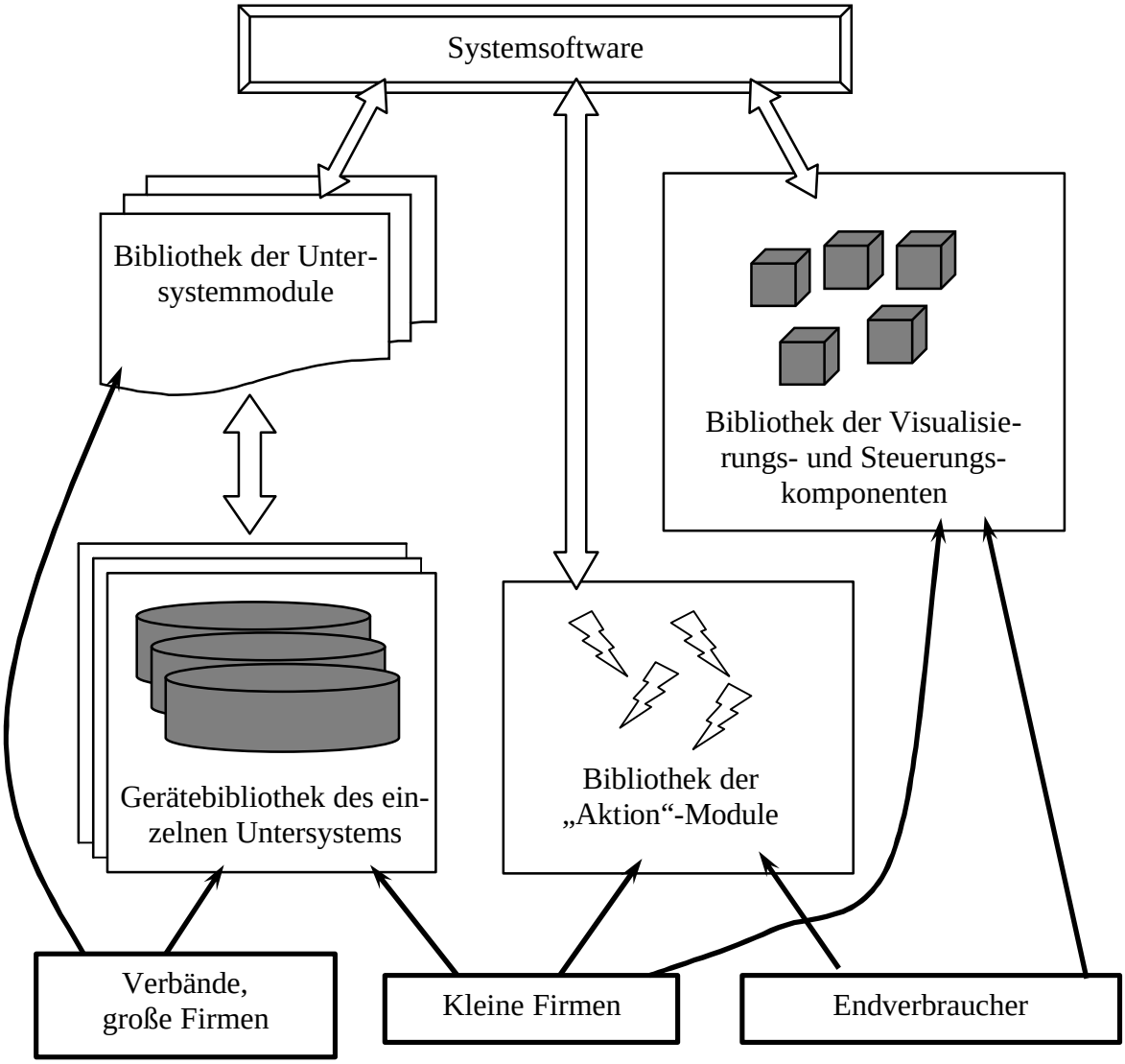


Bild 3.7. Modularität der Systemssoftware

So erlaubt die Standardisierung und noch wichtiger die freie Verfügbarkeit des Bibliotheksformats dem Anwender, selber neue Bibliotheken für Visualisierungs- und Steuerungskomponenten oder vielleicht sogar die Bibliotheken von Systemkomponenten zu schaffen. Dies

führt dazu, dass man das System später relativ mühelos durch neue Untersysteme und einzelne Untersysteme durch neue Geräte erweitern kann.

In den folgenden drei Unterkapiteln (3.4.1 – 3.4.3) wird die gewünschte Funktionalität jeder Gruppe der Systemsoftware im Einzelnen erläutert.

3.4.1. Konfigurierungssoftware

Die Hauptaufgabe der Konfigurierungssoftware ist die Konfigurierung bzw. Parametrierung aller installierten Systemkomponenten. Nach Möglichkeit kann sie auch die Konfigurierung der einzelnen Untersysteme übernehmen. Diese Aufgabe kann, je nachdem wie komplex das Untersystem aufgebaut ist, einen beträchtlichen technischen und zeitlichen Aufwand bedeuten, deshalb soll ihre Implementierung optional bleiben. Das bedeutet, dass die Konfigurierung des Untersystems sowohl die Konfigurierungssoftware des Systems als auch eine andere speziell für das Untersystem gefertigte Software durchführen kann.

Da die Konfigurierungssoftware hauptsächlich bei der Installation und Inbetriebnahme, und ab und zu später bei der Rekonfigurierung der installierten Komponenten oder beim Einfügen neuer Komponente benutzt wird, kann sie plattformabhängig sein. Die Konfigurierungssoftware kann für eine, maximal zwei, meist verbreitete Softwareplattformen implementiert sein, z.B. für Windows und Linux.

Die größte Schwierigkeit bei der Implementierung der Konfigurierungssoftware dürfte ein vernünftiger Kompromiss zwischen der einfachen Bedienbarkeit und der vielfältigen technischen Funktionalität bereiten. Um ihn zu erreichen, muss die Konfigurierungssoftware zwischen drei Anwendertypen (Tabelle 3.2) unterscheiden und an sie angepasste Werkzeuge für die Unterstützung des Konfigurierungsvorganges bieten.

Tabelle 3.2. Anwendertypen der Konfigurierungssoftware

Anwendertyp	Vorhandene Kenntnisse über das Funktionieren des Systems	Wofür wird die Konfigurierungssoftware genutzt?	Art der Konfigurierung	Wahrscheinlichkeit eines Anwenderfehlers
Anfänger	Minimale oder gar keine	Für die Personalisierung der Visualisierungsseiten	Alle Änderungen in der Systemkonfiguration (z.B. Einfügen von neuen Geräte) sind ausschließlich über Wizards[2] möglich	Sehr gering

[2] Wizard – eng. Zauberer – ist die Bezeichnung für die Dienstprogramme, die den Anwender bei komplexen Abläufen unterstützen. Meistens fragen sie dabei Schritt für Schritt die erforderlichen Daten ab und bieten dem Anwender anschließend eine Lösung an.

Fortgeschrittener	Allgemeine technische Kenntnisse und Vorstellungen über das Funktionieren des Systems	Für die Personalisierung der Visualisierungsseiten, für kleine Änderungen im System (Änderung der Verknüpfungen „Schalter - Lampe“, Einfügen von neuen Ereignissen und Reaktionen auf sie u.a.) oder Ergänzung durch neue Geräte	Direkt oder über Wizards	Hoch
Experte	Genaue Kenntnisse über das Funktionieren und den Aufbau des Systems	Für die Inbetriebnahme des Systems, für große Änderungen der Systemskonfiguration und für das Einfügen von neuen Geräten und Untersystemen	Direkt	Niedrig

Jeder der drei in Tabelle 3.2 beschriebenen Anwendertypen kann mit einer zusätzlichen Option „Entwickler“ kombiniert werden, die es dem Anwender erlaubt, das System durch eigene Module für die Visualisierungs- und Steuerungssoftware oder durch Module der Reaktionen auf Ereignisse im System zu ergänzen.

3.4.2. Visualisierungssoftware

Der Zweck der Visualisierungssoftware ist es, dem Anwender zu ermöglichen und möglichst bequem zu machen, den Zustand der verschiedenen Variablen oder Prozesse im System (ob das Licht im Zimmer aus oder an ist, ob die Waschmaschine schon fertig ist u.a.) zu beobachten, ihre Werte zu ändern bzw. sie zu steuern. Dies alles geschieht unabhängig davon, in welchem Untersystem die Variablen oder Prozesse existieren.

Die Hauptanforderung an die Visualisierungssoftware ist eine einfache und intuitiv verständliche Benutzeroberfläche, die ermöglichen soll, dass sie nicht nur von einem erfahrenen Anwender (z.B. dem Hausbewohner) genutzt werden kann, sondern auch von einem Laien, wie einem Gast oder einem Kind. Dabei hat man auch auf die „Usability“ (Deutsch: Gebrauchstauglichkeit) der Visualisierungssoftware zu achten, die in [37] definiert wird als „das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen“.

Eine weitere wichtige und diesmal technische Anforderung an die Visualisierungssoftware ist ihre Plattformunabhängigkeit, d.h. sie sollte in der Lage sein, auf einer möglichst großen Anzahl von unterschiedlichen Softwareplattformen ausgeführt zu werden. Das wird dem Anwender wiederum erlauben, sie ohne jegliche Veränderungen auf dem Heim-PC oder dem eingebauten Bedienterminal, und auch auf einem digitalen Assistent (PDA) oder einer Settopbox mit einem Fernseher als Ausgabegerät zu nutzen. Als Folge ihrer Plattformunabhängig-

keit muss die Visualisierungssoftware möglichst viele verschiedene Steuerungsmöglichkeiten bieten, wie z. B. per Tastatur, Maus, Touchscreen, Sprache oder anderes.

3.4.3. Servicesoftware

Die Servicesoftware ist eine Gruppe von unterschiedlichen Programmen und Modulen. Diese Gruppe ist nicht abschließend definiert und kann durch eigene Programme von Anwendern oder von dritten Anbietern ergänzt werden.

Auf jeden Fall ist es sinnvoll ein Programm den sog. „*Services*-Monitor“ zur Verfügung zu stellen. Das Programm stellt eine Art des Busmonitors für das *Services*-Gateway dar und erlaubt dem Anwender, sowohl mit einem an das *Services*-Gateway angeschlossenen Untersystem zu kommunizieren bzw. seine Services zu nutzen wie auch die in der Entwicklung befindliche *Services*-Gateways zu testen.

In der Gruppe Servicesoftware kann man sich auch unterschiedliche statistische Programme vorstellen, die verschiedene in den Gateways gesammelte Informationen darstellen oder analysieren. Diese könnten zum Beispiel die Buslast in einem Untersystem oder die zeitliche Nutzung der einzelnen Geräte (die Zeit zwischen Einschalten und Ausschalten des Gerätes) sein.

3.5. Zusammenfassung

Zusammenfassend ist der vorgeschlagene Systemaufbau in Bild 3.8 dargestellt. Die Funktionsweise der einzelnen Softwarekomponenten und ihre Zusammenarbeit ist dort überschlägig angegeben. (Die Verbindungen der Servicesoftware sind mittels des in Kapitel 3.4.3 erwähnten Programms *Services*-Monitor dargestellt)

Die Software im System kann man in drei Gruppen aufteilen: die Systemsoftware (Konfigurierungs-, Visualisierungs- und Servicesoftware), die Software der Systemkomponenten (*Services*-Gateway, *StateMirror*, *EventController*) und die Systemerweiterungsmodule (Module des einzelnen Untersystems, Gerätebibliotheken, Bibliotheken der Visualisierungs- und Steuerungskomponenten und Bibliotheken der „Aktion“-Module).

Die für jedes System spezifischen Daten, die man jedes Mal anpassen (konfigurieren) muss, sind auf Konfigurierungsdaten (Projektdatei) und Visualisierungsdaten (Dateien der Visualisierungs- und Steuerungsseiten) aufgeteilt.

Die Zusammenarbeit zwischen den einzelnen Softwarekomponenten ist in Form eines Pfeils dargestellt. Die Pfeilrichtung zeigt, von welchem auf welches Modul zugegriffen wird und/oder die Richtung des Datenaustausches. Die verschiedenen Pfeiltypen deuten auf einzelne Zusammenarbeitsarten der Softwarekomponenten. Ein dicker Pfeil bedeutet einen Datenaustausch zwischen Komponenten über den Backbone, ein dünner Pfeil – direkte Lese- oder Schreibzugriffe auf der Komponente und ein dünner gestrichelter Pfeil – einen Softwarezugriff (call) auf die einzelne Unterprogramme in der Komponente.

Bei einer genauren Behandlung des im Bild 3.8 dargestellten Schemas kann man einige Besonderheiten des Systemaufbaus erkennen:

- Das *Services*-Gateway hat keinen Zugriff auf die Konfigurierungsdaten des Systems, d.h. es muss ohne spezielle Einstellungen auf einem konkreten Systementwurf auskommen können und somit “Plug&Play“-fähig sein;
- Die Konfigurierungssoftware muss in der Lage sein, auf alle ans System angebundene Softwaremodule zugreifen zu können, um eine Beschreibung der Module (Anzahl der Modulparameter, ihren Typ u.a.) zu bekommen;
- Ein Zugriff auf die Gerätebibliothek ist nur für die Untersystemmodule gewährleistet. Das kann man damit erklären, dass jedes Untersystem eine eigene Beschreibung der Geräte und ihrer Eigenschaften nutzt. Als Folge können die Module unterschiedlicher Untersysteme und manchmal unterschiedliche Module des gleichen Untersystems verschiedene Formate der Gerätebibliothek verwenden. In einigen Fällen ist ein Zugriff auf die Gerätebibliothek auch für die Konfigurierungssoftware sinnvoll. Dies kann man mit einem standardisierten Interface des Untersystemmoduls erreichen, d.h. die Konfigurierungssoftware greift nicht direkt sondern über das Untersystemmodul auf die Gerätebibliothek zu;

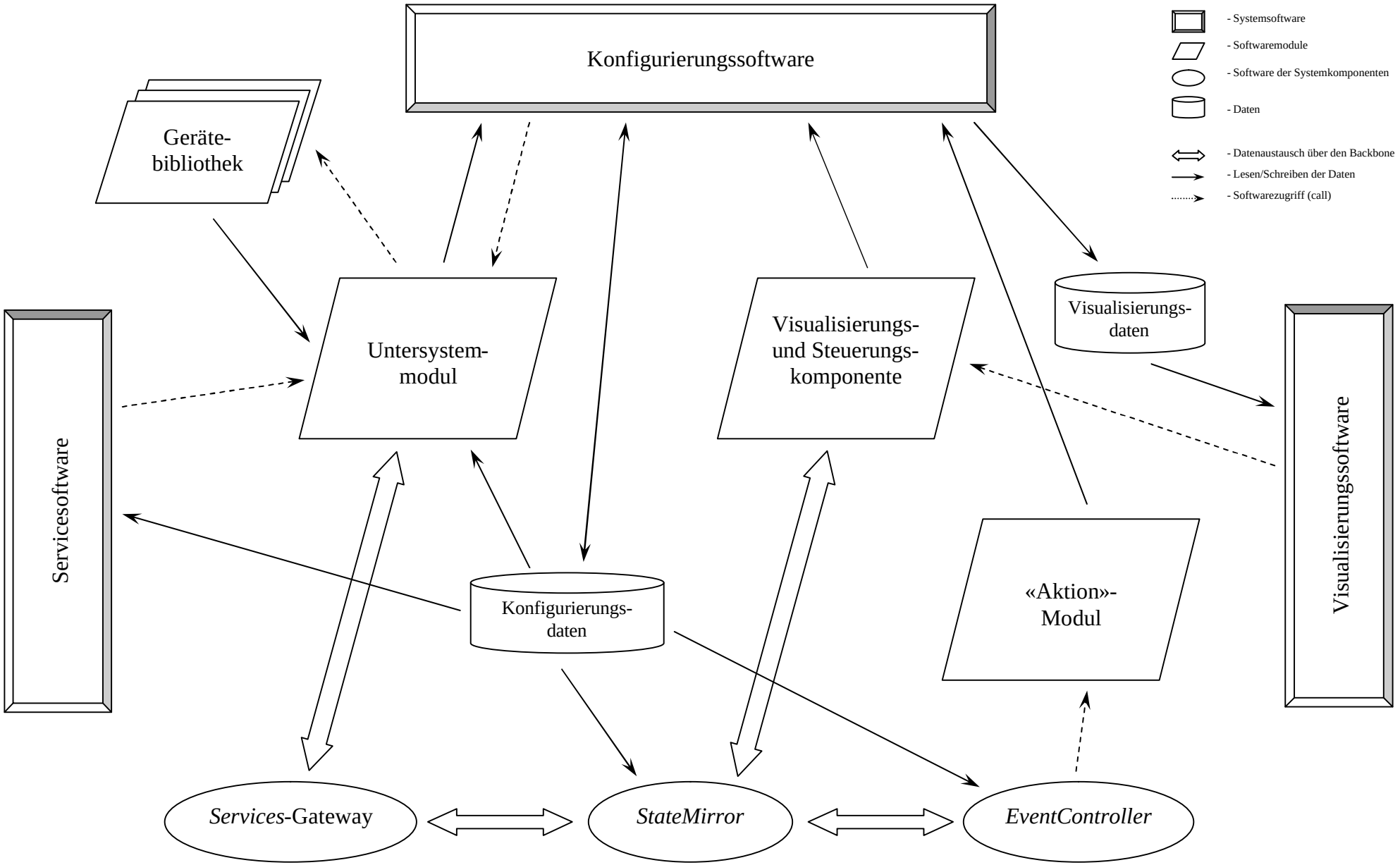


Bild 3.8. Zusammenarbeit der Softwarekomponenten im System

4. Auswahl der Protokolle für die Systemimplementierung

In den letzten Jahren haben einige Netzwerkstandards für Lokal-/Heimnetze eine große Verbreitung gefunden. Das sind bei den Kabelnetzen: Ethernet, token ring, Appletalk, u.a. und bei den kabellosen: DECT, Wireless Lan, Bluetooth, u.a. Die größte Verbreitung und den „De facto"-Standard unter den Datennetzen besitzt das Ethernet. Aber Ethernet ist nicht gleich Ethernet. Es gibt mehrere Ethernetstandards, die sich im Wesentlichen dadurch unterscheiden, auf welchem Medium (Koaxialkabel, Zweidraht, Glasfaser) und mit welcher Geschwindigkeit (1Mbit, 10Mbit, 100Mbit, 1Gbit oder 10 Gbit) die Übertragung stattfindet.

Oft existieren in einem Haus mehrere unterschiedliche Netzsegmente nebeneinander, die sich im Laufe der Zeit gebildet haben. Da ist der Wunsch des Anwenders verständlich, sie zu einem logischen Netz zu vereinigen, anstatt alle durch ein einheitliches Netz zu ersetzen. Diesen Wunsch kann man am einfachsten auf der Basis eines einheitlichen Transportprotokolls für die Datenübertragung realisieren. Das meist verbreitete Transportprotokoll ist heutzutage das Internetprotokoll (IP), das sich über lange Jahre hinweg und in vielen Anwendungen als zuverlässig und robust erwiesen hat. Die Auswahl des IP's als Transportprotokoll erlaubt auch die Telekommunikationsnetze (Analog, ISDN und DSL) oder die Energieübertragungsnetze (Powerline) für die Datenübertragung von der Außenwelt ins Heimnetz und umgekehrt zu nutzen. Von dieser Tatsache ausgehend, werden in diesem Kapitel nur die Protokolle als mögliche Kandidaten berücksichtigt, die auf der IP-Kommunikation basieren.

Die allgemeine Forderungen, die an alle Kandidaten gestellt wurden, waren ihre weite Verbreitung (d.h. die Protokollimplementierungen müssen auf einem breiten Spektrum an Hardwareplattformen vorhanden sein) und ihre Offenheit (frei verfügbare Protokollspezifikation).

Falls mehrere passende Kandidaten vorhanden sind, wird die Auswahl anhand der Bewertungstabelle (Beispiel Tabelle 4.1) getroffen. In einer solchen Tabelle werden alle Anforderungen an das Protokoll aufgeführt und anschließend ihre Erfüllung durch das Protokoll bewertet.

Tabelle 4.1. Ein Beispiel der Protokollbewertungstabelle

	Protokoll 1	Protokoll 2	Protokoll 3
Anforderung 1	++	**O**	-
Anforderung 2	--	+	**O**

Die Notenskala der Anforderungserfüllung hat fünf Werte: "++" – ausgezeichnet; "+" – gut; "**O**" – ausreichend; "-" – schlecht; "--" – die Anforderung wird nicht erfüllt.

Zum Schluss muss noch angemerkt werden, dass, obwohl der Autor die Meinung vertritt - die in diesem Kapitel ausgewählten Protokolle für das hier beschriebene System aus heutiger Sicht optimal sind, können diese nicht als endgültige Lösung betrachtet werden, sondern als eine Ausgangsbasis, die mit fortschreitender Entwicklung und Wissenserweiterung verbessert werden kann.

4.1. Protokoll für das *Services*-Gateway

Zusätzlich zu den allgemeinen Kriterien, die im letzten Abschnitt aufgezählt sind, muss ein Protokoll für das *Services*-Gateway auch die im Kapitel 3.1 aufgestellten Anforderungen erfüllen.

Damit sieht die vollständige Liste der Anforderungen folgendermaßen aus:

- weite Verbreitung des Protokolls
- Offenheit des Protokolls
- Unterstützung einer Betriebsart mit einer festen Adresse im Backbone oder mit ihrer automatischen Zuteilung bei der Inbetriebnahme
- Gewährleistung des automatischen Entdeckens der installierten Services-Gateways im Backbone
- Authentifizierung von Clients, Zuteilung von Zugriffsrechten
- Versenden der Beschreibung von unterstützten Services auf Anfrage vom Client
- Unterstützung des bidirektionalen asynchronen Informationsaustausches mit dem Client

Passende Kandidaten, die in den nächstfolgenden Abschnitten betrachtet werden, sind: „Universal Plug and Play“, „Open Services Gateway“ und „OLE for Process Control“.

4.1.1. Universal Plug and Play (UPnP)

Das Protokoll UPnP wurde im Jahre 1999 von der Firma Microsoft vorgeschlagen. Im Mittelpunkt des neuen Protokolls stand die Entwicklung einer standardisierten, flexiblen und leicht verwendbaren Möglichkeit, unterschiedliche Geräte an ein sog. „unverwaltetes“ Netz sowohl zu Hause, im Büro oder in öffentlichen Plätzen, als auch direkt ans Internet anschließen zu können. Dabei wurde besondere Aufmerksamkeit auf die Unterstützung der „Nullkonfigurierung“, dem „unsichtbaren“ Netzwerkbetrieb und der automatischen Entdeckung und Zusammenarbeit von verschiedenen Geräten unterschiedlicher Hersteller gerichtet. Das heißt, dass Geräte dynamisch ans Netz angeschlossen werden können, automatisch eine gültige Netzadresse zugewiesen bekommen und dann ihre Eigenschaft den anderen Netzteilnehmern mitteilen können. Im Gegenzug können die angeschlossenen Geräte die Informationen über andere im Netz vorhandene Geräte und ihre Eigenschaften bekommen.

Dabei ist das UPnP Protokoll kein neuer Standard, vielmehr kombiniert es die vorhandenen und bewährten Netzwerkstandards, wie: IP, TCP, UDP, HTTP, XML und andere (Bild 4.1). Dies seinerseits erlaubt dem UPnP Protokoll ein breites Spektrum an physikalischen Übertragungsmedien zu verwenden und damit eine Synergie zwischen dem Internet und den Heim- und Büronetzen zu erreichen.

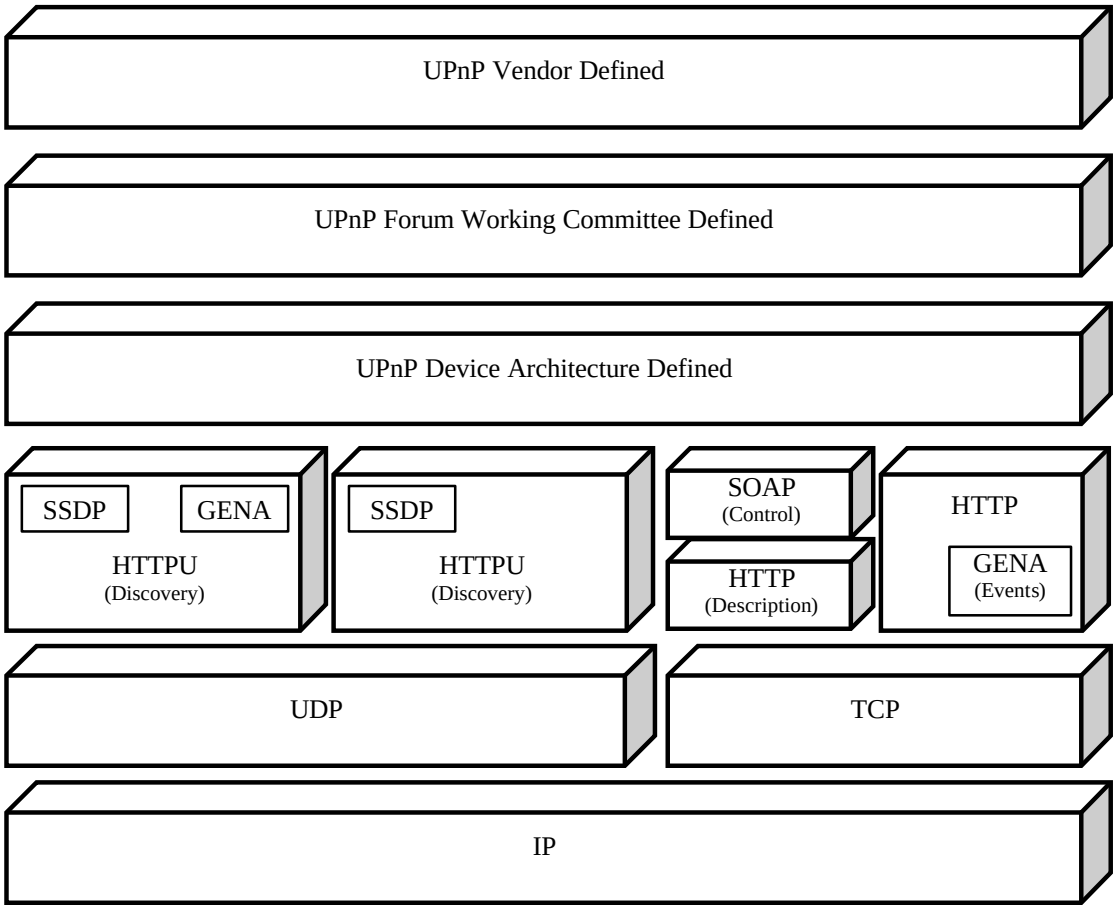


Bild 4.1. Der UPnP Protokollstack

Unter „Universalität“ des Protokolls verstehen die Schöpfer, dass UPnP Geräte keinen speziellen Treiber benötigen, sondern sie nutzen stattdessen ein gemeinsames Protokoll.

Der Netzbetrieb des UPnP’s hängt nicht vom Übertragungsmedium ab. UPnP Geräte können in einer beliebigen Programmiersprache und für ein beliebiges Betriebsystem implementiert werden. Das UPnP Protokoll stellt keine Anforderungen an den internen Softwareaufbau des UPnP Gerätes und an seine Softwareschnittstelle. Damit bietet es dem Hersteller und dem Anwender die notwendige Realisierungsfreiheit.

Die Grundelemente des Aufbaus von UPnP Netzen sind: Dienste, Geräte und Steuerungspunkte (Bild 4.2).

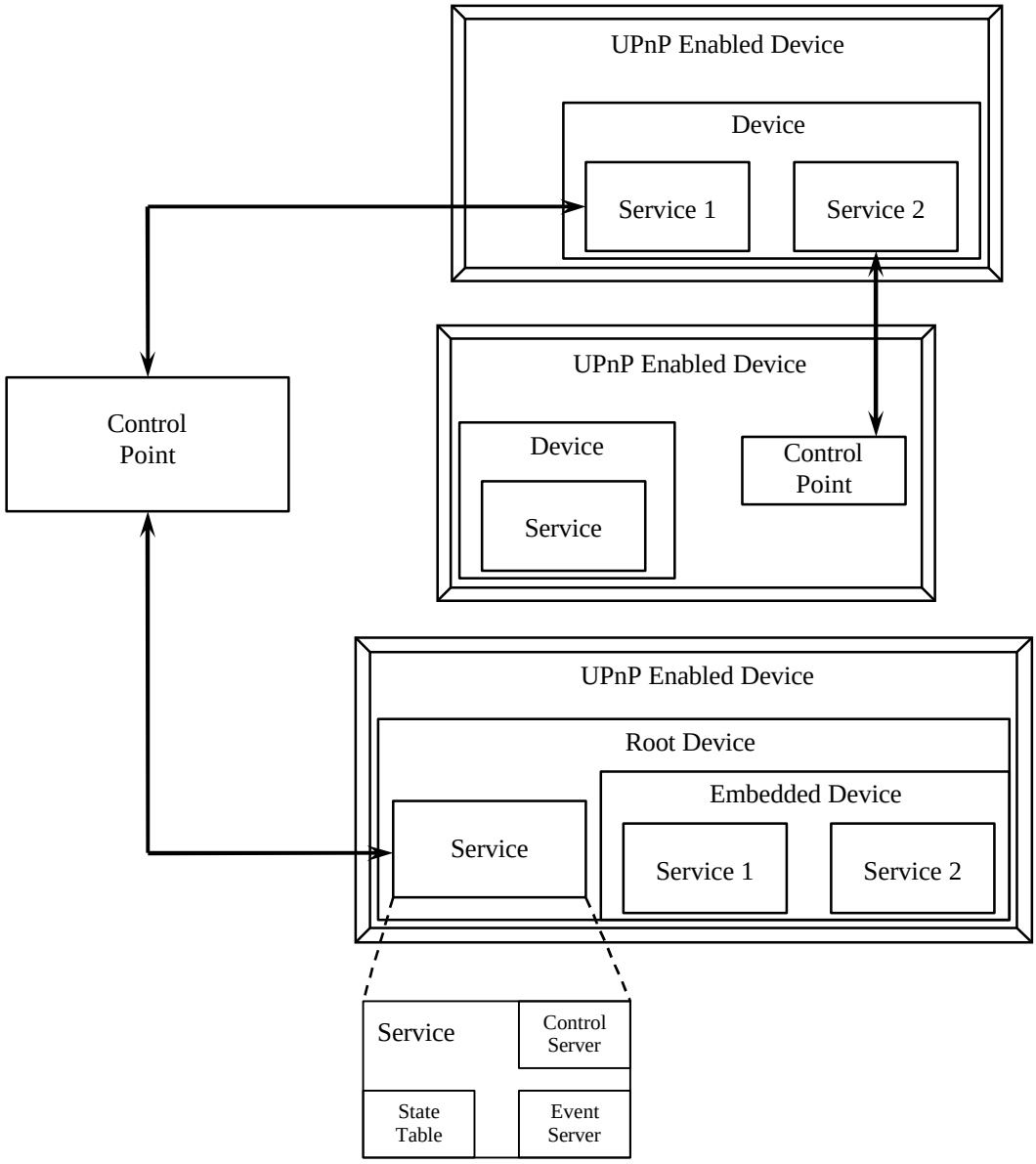


Bild 4.2. Die UPnP Steuerungspunkte, Geräte und Dienste

Das Gerät stellt einen gewissen Container für Dienste und andere integrierte Geräte dar. So kann zum Beispiel, eine Uhr mit eingebautem Radio einen Dienst „Uhr“ und ein integriertes Gerät „Radio“ beinhalten. Letzteres seinerseits kann noch weitere Dienste bieten, solche wie: „Tuner“, „Lautstärkeregelung“ u.a.

Die kleinste unteilbare Steuerungseinheit im UPnP Netz stellt ein Dienst dar. Er bietet normalerweise verschiedene Schnittstellen an und kann seinen Zustand mit Hilfe der Zustandsvariablen variieren. Zum Beispiel kann der Dienst „Uhr“ eine Variable „aktuelle Zeit“ und zwei Schnittstellen „Zeit lesen“ und „Zeit schreiben“ zur Verfügung stellen.

Jeder UPnP Dienst beinhaltet

- eine Zustandstabelle, die den Zustand des Dienstes durch Zustandsvariablen steuert oder sie nach der Änderung des Dienstzustandes aktualisiert;

- einen Steuerungsserver, der über die Schnittstellen verschiedene Steuerungsanfragen entgegennimmt, die Zustandstabelle aktualisiert und das Anfrageergebnis zurückschickt;
- einen Ereignisserver, der bei Änderung des Dienstzustandes eine entsprechende Nachricht an die Teilnehmer verschickt, die sich dafür angemeldet haben

Ein Steuerungspunkt in UPnP Netzen ist ein Steuerungsgerät, das in der Lage ist, andere Netzteilnehmer aufzuspüren und sie zu steuern. Nach dem Entdecken des neuen Gerätes im Netz kann der Steuerungspunkt:

- eine Beschreibung des Gerätes und der unterstützen Services von ihm anfordern
- die Unterstützung der ihn interessierenden Services abfragen
- Schnittstellen des neuen Gerätes zur Steuerung des Dienstes benutzen
- sich für die ihn interessierenden Dienstereignisse anmelden und bei ihrem Zustandekommen eine entsprechende Nachricht vom Ereignisserver des Dienstes bekommen

Den Netzbetrieb eines UPnP Gerätes kann man in sechs Schritte untergliedern.

Der erste von ihnen heißt „Adressierung“. Dadurch bekommt das Gerät eine gültige Netzwerkadresse. Um dies zu erreichen, muss jedes Gerät einen DHCP-Client besitzen und beim ersten Anschluss ans Netz einen DHCP-Server aufsuchen. Wenn dieser Vorgang jedoch fehlschlägt, muss das Gerät, um die Adresse zu erlangen, auf das „AutoIP“ Verfahren zurückgreifen. Kurzgefasst, bestimmt das AutoIP, wie man aus dem reservierten Bereich der Netzwerkadressen eine gültige für sich heraussucht. Die Kombination aus diesen beiden Verfahren ermöglicht eine freie Bewegung der Geräte zwischen den verwalteten und unverwalteten Netzen. Dem Entwickler ist auch die Möglichkeit überlassen, hierzu andere Verfahren oder Protokolle so wie z.B. Domain Name Service (DNS) einzusetzen.

Der zweite Schritt im Netzbetrieb des UPnP Gerätes heißt „Discovery“. Der Entdeckungsprozess basiert auf dem Protokoll „Simple Service Discovery Protocol“ (SSDP) [38]. Ein erstmals ans Netz angeschlossenes Gerät kann dadurch seine Dienste dem Steuerungspunkt anbieten. Oder umgekehrt, wenn ein Steuerungspunkt ans Netz angeschlossen wird, ermöglicht ihm das SSDP die vorhandenen, ihn interessierenden Geräte aufzuspüren. In beiden Fällen wird im Netz eine „Discovery“ Nachricht verschickt, die eine für das entsprechende Gerät oder den entsprechenden Dienst spezifische Information und den Verweis auf das Dokument mit voller Gerätebeschreibung beinhaltet.

Der nächste dritte Schritt im Netzbetrieb des UPnP Gerätes heißt „Beschreibung“. Nachdem der Steuerungspunkt das Gerät aufgespürt hat, verfügt er über nur wenige Informationen. Um weitere Auskünfte über das Gerät oder seine Fähigkeiten zu erhalten, oder um seine Schnittstellen nutzen zu können, muss der Steuerungspunkt zuerst die volle Beschreibung des Gerätes anfordern und dafür den Verweis aus der „Discovery“ Nachricht nutzen. Die UPnP Beschreibung des Gerätes wird im Extensible Markup Language (XML) Format verbreitet und beinhaltet in der Regel die Herstellerinformation (Gerätebezeichnung, Seriennummer, Name des Herstellers u.a.), die Liste der Dienste und integrierten Geräte und auch die Verweise für die Steuerung, die Ereignisse und die Präsentation.

Der vierte Schritt ist die „Steuerung“. Nachdem der Steuerungspunkt die Gerätebeschreibung bekommen hat, verfügt er über die wesentlichen Informationen, um das Gerät steuern zu können. Was ihm noch fehlt, ist eine detaillierte Schilderung der verfügbaren Services.

Diese Schilderung wird auch im XML Format verbreitet und beinhaltet die Liste der unterstützten Befehle und Aktionen, und auch ihre Parameter und Argumente. In der Beschreibung stehen auch alle Variablen des Services, die seinen gegenwärtigen Zustand widerspiegeln, ihre Typen, ihre Wertbereiche usw. Um das Gerät zu steuern, sendet der Steuerungspunkt eine Anfrage an dessen Dienst. Dabei wird eine entsprechend aufgebaute Nachricht an die in der Gerätebeschreibung vorhandene Adresse für die Steuerung geschickt. Die Steuerungsnachricht ist im XML Format und benutzt das Simple Object Access Protocol (SOAP) [39,40]. Als Antwort auf die Steuerungsnachricht schickt das Gerät das Ergebnis oder den Fehlercode zurück.

Der fünfte Schritt im Netzbetrieb eines UPnP Gerätes heißt „Eventing“. Wie schon erwähnt, enthält die UPnP Beschreibung des Geräteservices unter anderem auch die Liste der Zustandsvariablen. Jedes Mal, wenn sich der Zustand irgendeiner Variable ändert, veröffentlicht der Service im Netz eine Information darüber und jeder Netzteilenehmer kann sie abonnieren und bekommen. Eine Ereignisnachricht enthält den Namen einer oder mehrerer Zustandsvariablen und ihre gegenwärtigen Werte. Diese Nachricht wird im XML Format verbreitet und benutzt Generic Event Notification Architecture (GENA) [41]. Um die Unterstützung von mehreren Steuerungspunkten zu gewährleisten, verschickt der Dienst die abonnierten Ereignisse an alle Abonnenten. Dabei enthalten die Ereignisse keinen Hinweis auf den Grund, weshalb die Zustandsvariable ihren Wert geändert hat (der Abfrage vom Steuerungspunkt zufolge oder infolge einer Änderung des Dienstzustandes).

Der sechste Schritt ist die „Präsentation“. Falls das Gerät einen gültigen Präsentationsverweis hat, kann der Steuerungspunkt die Präsentationsseiten von der Stelle anfordern, auf welche der entsprechende Verweis zeigt. Als nächstes müssen die Seiten in den Internetbrowser geladen werden und bieten dem Anwender dadurch die Möglichkeit, entsprechend den Seiteneigenschaften, das Gerät zu steuern oder seinen Zustand zu beobachten.

Unter Berücksichtigung der oben erwähnten Eigenschaften des UPnP Protokolls kann man folgende Bewertungen für die Entsprechung des Protokolls an die in Kapitel 4.1 aufgestellten Anforderungen abgeben. Da das UPnP Protokoll sich auf weitverbreitete Protokolle stützt, deren Implementierungen für viele verschiedene Plattformen existieren, erfüllt es bestens (Note „ausgezeichnet (++)“) die Anforderung an die weite Verbreitung des Protokolls. Für die Offenheit des UPnP Protokolls gibt es ebenfalls die höchste Note, weil die Spezifikationen der beim UPnP verwendeten Protokolle offen gelegt sind und damit für jedermann zugänglich sind. Das UPnP Protokoll unterstützt den Netzwerkbetrieb sowohl mit einer festen Netzwerkadresse als auch mit ihrer automatischen Zuteilung (siehe Schritt „Adressierung“ im UPnP Netzbetrieb) und deswegen verdient es in der entsprechenden Kategorie die beste Note. Dank der im UPnP Protokoll integrierten Unterstützung des automatischen Entdeckens der UPnP Geräte (siehe Schritt „Discovery“ im UPnP Netzbetrieb) erhält das UPnP Protokoll auch hier die höchste Bewertung. Da es aber keine implizite Unterstützung der Authentifizierung der Clients und ihrer Zugriffsrechte aufweist, wird die entsprechende Anforderung nicht erfüllt (Bewertung „--“). In den letzten zwei Kategorien bekommt das UPnP Protokoll die höchste Note, weil es das Versenden der Dienstbeschreibungen und den asynchronen Datenaustausch mit den Clients unterstützt.

Zusammenfassend lässt sich die folgende Bewertungstabelle für das UPnP Protokoll (Tabelle 4.2) erstellen.

Tabelle 4.2. Die Bewertungstabelle für das UPnP Protokoll

Anforderungen	UPnP
weite Verbreitung des Protokolls	++
Offenheit des Protokolls	++
Unterstützung einer Betriebsart mit einer festen Adresse im Backbone oder mit ihrer automatischen Zuteilung bei der Inbetriebnahme	++
Gewährleistung des automatischen Entdeckens der installierten Services-Gateways im Backbone	++
Authentifizierung von Clients, Zuteilung von Zugriffsrechten	--
Versenden der Beschreibung von unterstützen Services auf Anfrage vom Client	++
Unterstützung des bidirektionalen asynchronen Informationsaustausches mit dem Client	++

4.1.2. Open Services Gateway (OSG)

Dieser Standard wurde von der Open Services Gateway Initiative (OSGi) [42] im Jahre 2000 vorgeschlagen. Das Hauptziel des neuen Protokolls war es, eine offene Spezifikation für die Zustellung verwalteter und verteilter Dienste zu lokalen Netzen und zu einzelnen Geräten zu schaffen.

Der Standard OSG definiert eine Umgebung (Framework) für die Dienste [43,44]. Dieses Framework stellt die softwaremäßig herunterladbaren Dienste einer Laufzeitumgebung zur Verfügung. Solche Dienste heißen im OSG „Bundles“. Die runtergeladenen Bundles werden im Rahmen und unter der Kontrolle des Frameworks ausgeführt und finden eine genau definierte und sichere Umgebung (Bild 4.3). Dieses Framework enthält eine Java Laufzeitumgebung, einen Festdatenspeicher, Versionsverwaltungsmittel und eine Registry (Verwaltungsdatenbank) des Dienstes.

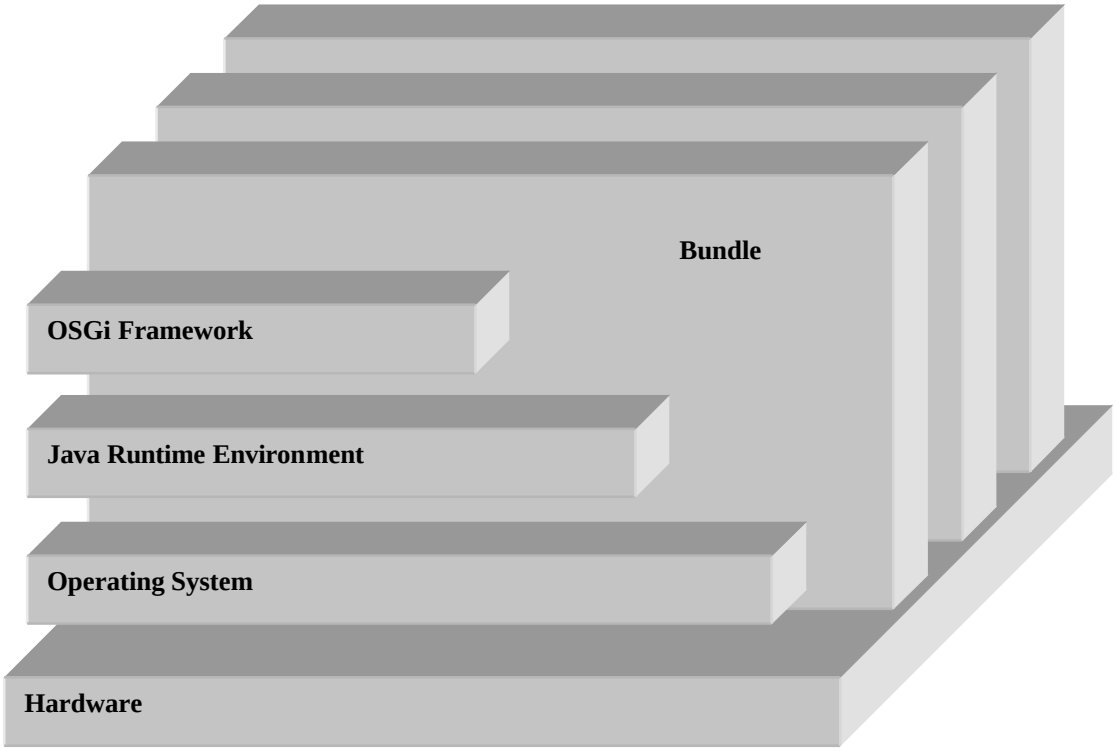


Bild 4.3. Die OSG Umgebung

Die Services stellen die Javaobjekte mit kurzen spezifischen Schnittstellen dar. Das Registry des Dienstes des OSG Frameworks wird dazu genutzt, um einen sicheren und kontrollierten Austausch von Diensten zwischen den einzelnen Bundles zu ermöglichen. Über das Registry können die Bundles anderen beliebige Dienste zur Verfügung stellen oder auch deren Dienste selber benutzen. Das Registry gewährleistet die höchste Sicherheitsstufe und bietet damit dem Operator die volle Kontrolle über die Plattform.

Mittlerweile existiert schon die zweite Version des OSG Standards. Die erste Version hat nur definiert, auf welche Weise die verwalteten Dienste über das globale Netz zu den lokalen Netzen und Geräten gelangen. Diese Version hat drei Dienste definiert: Protokollierungs-, Webserver- und Gerätezugriffsdienst. Die zweite Version, die im Jahre 2001 erschienen ist, verbessert und erweitert die Softwareschnittstellen. Alle Änderungen sind rückwärts kompatibel. Dies ermöglicht den Anwendungen, die für die erste Version geschrieben sind, auch unter der zweiten Version der Umgebung ohne irgendeine Veränderung ausgeführt zu werden. Die wichtigsten Erweiterungen in der zweiten Version sind die Vereinfachung des Bundleaufbaus und die Erhöhung der Sicherheit. So wurde zum Beispiel noch ein getrenntes Bundle definiert, das die Zugriffs- und Sicherheitsrechte zur Laufzeit bestimmt und kontrolliert.

Die Bewertungen des OSG Standards sehen folgendermaßen aus: für die Verbreitung des Standards erhält das OSG die Note „ausreichend (**O**)“, weil die OSG Software unter der Java Virtual Machine läuft und deshalb nur auf einer begrenzten Anzahl von Plattformen eingesetzt werden kann; da der OSG Standard offen ist, bekommt er in der Kategorie „Offenheit des Protokolls“ die Note „ausgezeichnet (++)“; die Anforderung an eine feste Adressierung im Netz oder die automatische Zuteilung der Netzwerkadresse bleibt vom OSG Stan-

dard unerfüllt, weil der Standard diese an die darunter liegende Software der Hardwareplattform delegiert bzw. eine Adresse von dieser vorgeschrieben bekommt; für die Gewährleistung des automatischen Entdeckens der OSG Geräte bekommt der OSG Standard nur eine ausreichende Note, weil er nicht über eine solche eingebaute Funktionalität verfügt, dafür aber bietet er genug Flexibilität und Erweiterungsfähigkeit, um diese Funktion von Seiten des Anwenders als getrenntes Bundle nachrüsten zu lassen; die folgenden drei Kategorien werden jeweils mit der besten Note bewertet, da der OSG Standard eine implizite Unterstützung der Authentifizierung von Clients, der Zuteilung von Zugriffsrechten, des Versandes von Dienstbeschreibungen und des asynchronen Informationsaustausches vorweist.

Somit sieht die Bewertungstabelle des OSG Standards folgendermaßen aus (Tabelle 4.3).

Tabelle 4.3. Die Bewertungstabelle des OSG Standards

Anforderungen	OSG
weite Verbreitung des Protokolls	**O**
Offenheit des Protokolls	++
Unterstützung einer Betriebsart mit einer festen Adresse im Backbone oder mit ihrer automatischen Zuteilung bei der Inbetriebnahme	--
Gewährleistung des automatischen Entdeckens der installierten Services-Gateways im Backbone	**O**
Authentifizierung von Clients, Zuteilung von Zugriffsrechten	++
Versenden der Beschreibung von unterstützten Services auf Anfrage vom Client	++
Unterstützung des bidirektionalen asynchronen Informationsaustausches mit dem Client	++

4.1.3. OLE for Process Control (OPC)

Das OPC Protokoll wurde im Jahre 1996 von einer Initiativgruppe von Herstellern vorgeschlagen. Im Mittelpunkt des neuen Protokolls stand die Bewältigung des bekannten Problems der Ein-/Ausgabe Treiber. Dieses Problem der Industrieautomatisierung liegt in der Vielfalt der unterschiedlichen Systeme, Programmiersprachen, Protokolle usw. Dazu existieren auch noch viele verschiedene industrielle Netzwerkstandards, solche wie: Interbus, Profibus, Devicenet u.a. Als Folge davon fehlt im Herstellungsprozess ein einheitlicher Standard, der es ermöglicht, die von den Geräten erhaltenen Daten einlesen und bearbeiten zu können. Dies bedeutet seinerseits, dass alle Hersteller unabhängig voneinander einen eigenen Weg ausarbeiten und pflegen müssen, um auf ein und dasselbe Gerät zugreifen zu können. So etwas ist natürlich unlogisch und ziemlich teuer. Deshalb wurde ein Versuch unternommen, einen neuen Standard für die Softwareschnittstelle (z.B. Druckertreiber im Betriebsystem Windows) zu entwickeln. Der neue Standard sollte den Anschluss der Steuerungskomponenten und anderer nach dem „Plug&Play“-Prinzip ermöglichen [45].

Wie das Wort „OLE“ schon verrät, basiert OPC auf dem Komponentenmodell der Fa. Microsoft, des Herstellers des Betriebssystems Windows™. In der Zeit, als das Protokoll ausgearbeitet wurde, wurde der Begriff „OLE“ für den Gesamtkomponentenaufbau des Betriebssystems verwendet. Heutzutage hingegen wird der Begriff „Component Object Model (COM)“ eingesetzt. COM bietet den Programmen eine bestimmte Umgebung für die Zusammenarbeit der verschiedenen Softwarekomponenten an. Später wurde auch die Zusammenarbeit der Komponenten ermöglicht, die sich auf unterschiedlichen Rechnern im Netz befinden. Der Mechanismus, der dafür zuständig ist, heißt „Distributed COM (DCOM)“.

Das OPC Protokoll definiert nicht das Informationaustauschverfahren im Netz, sondern die Softwareschnittstellen und Komponenten. So wurde für den Lieferanten der OPC Dienste der Begriff „OPC-Server“ verwendet, und für ihre Verbraucher – „OPC-Client“. Der Dienst stellt sich im Sinne der Objektorientierung als ein Satz von Eigenschaften und Methoden dar. Jeder Dienst enthält einen bestimmten Basissatz von Eigenschaften und Methoden und bietet damit dem Client ein gewisses Abstraktionsniveau und ermöglicht ihm die Nutzung von unterschiedlichen Diensten verschiedener Server (Bild 4.4).

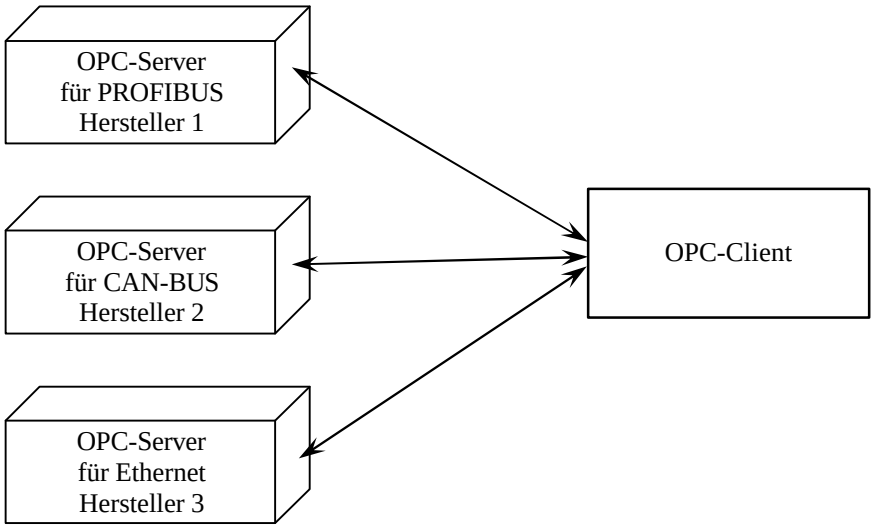


Bild 4.4. Die Benutzung von Diensten verschiedener Server durch den Client

Das OPC Protokoll bietet dem Hersteller die Freiheit in der Auswahl der Dienstsyntax (die Namen der Methoden und Variablen). Um den Clients trotzdem den Zugang zur Liste der verfügbaren Dienste zu verschaffen, wurde eine „Browsing“-Methode definiert, die dem Client eine Liste der verfügbaren Methoden und Eigenschaften liefert. Das OPC bietet dem Client auch die Möglichkeit, beliebige Variable zur Überwachung zu registrieren. Damit bleibt dem Client das ständige Abfragen ihres Wertes erspart und, falls der Wert einer Variablen sich ändert, wird der Client automatisch vom Server darüber benachrichtigt.

Unter Berücksichtigung der obenbeschriebenen Eigenschaften wird das OPC Protokoll für die Erfüllung der im Kapitel 4.1 beschriebenen Anforderungen bewertet. Für die Verbreitung des Protokolls bekommt das OPC die Note „schlecht (-)“, weil es Implementierungen nur für das Betriebssystem „Windows“ vorweist. Obwohl es seit einiger Zeit Anstrengungen gibt, das OPC Protokoll auf das Betriebssystem Linux zu portieren [46], befinden sich diese immer noch in einem experimentellen Stadium. Die Anforderung an die Offenheit des Protokolls wird nur mäßig erfüllt und daher mit der Note „ausreichend“ bewertet. Dies liegt daran,

dass die Spezifikation des OPC Protokolls unzugänglich bleibt, es lassen sich jedoch einige Rückschlüsse aus dem oben erwähnten Linux Projekt ziehen, da es im Quelltext verfügbar ist. Die folgenden drei Anforderungen an die Adressierung, automatisches Entdecken, Authentifizierung von Clients und ihrer Zugriffsrechte werden vom OPC Protokoll nicht erfüllt, daher erhält es hierbei die entsprechende Bewertung (--). Im Gegensatz dazu wird dem Protokoll für das Versenden der Dienstbeschreibungen und für die Unterstützung des bidirektionalen und asynchronen Informationsaustausches die höchste Note vergeben, denn das OPC Protokoll verfügt über die entsprechenden spezifizierten Funktionalitäten.

Die Ergebnisse dieser Beurteilung lassen sich der folgenden Bewertungstabelle des OPC Protokolls entnehmen (Tabelle 4.4).

Tabelle 4.4. Die Bewertungstabelle des OPC Protokolls

Anforderungen	OPC
weite Verbreitung des Protokolls	-
Offenheit des Protokolls	**O**
Unterstützung einer Betriebsart mit einer festen Adresse im Backbone oder mit ihrer automatischen Zuteilung bei der Inbetriebnahme	--
Gewährleistung des automatischen Entdeckens der installierten Services-Gateways im Backbone	--
Authentifizierung von Clients, Zuteilung von Zugriffsrechten	--
Versenden der Beschreibung von unterstützten Services auf Anfrage vom Client	++
Unterstützung des bidirektionalen asynchronen Informationsaustausches mit dem Client	++

4.2. Protokoll für den *StateMirror*

Die volle Liste der Anforderungen an das Protokoll für den *StateMirror* besteht aus den allgemeinen Anforderungen an alle hier betrachteten Protokolle und den in Kapitel 3.2 erwähnten Anforderungen. Diese Liste sieht wie folgt aus:

- weite Verbreitung des Protokolls
- Offenheit des Protokolls
- Gewährleistung des automatischen Entdeckens der installierten Services-Gateways, der Möglichkeit in sie einzuloggen und ihre Dienste abonnieren zu können
- Verwaltung der Datenbank durch logische und physikalische Prozessvariablen, Gewährleistung des Lese/Schreib-Zugriffs für Clients
- Authentifizierung der Clients und ihrer Zugriffsrechte

– Unterstützung des „Multisitzungbetriebs“

Die möglichen Kandidaten, die in den nächsten Abschnitten betrachtet werden, sind „Simple Network Management Protocol“ und der frei in Quellkode verfügbare und kostenlose Vertreter der Datenbankwelt – „MySQL“.

4.2.1. Simple Network Management Protocol (SNMP)

Das SNMP Protokoll wurde von der Internet Engineering Task Force (IETF) [47] in der zweiten Hälfte der 80’er Jahre vorgeschlagen. Das IETF beschäftigt sich mit der Standardisierung von Internetprotokollen in Form sog. „Requests For Comments (RFCs)“. Sein Erscheinen verdankt das SNMP dem exponentiellen Wachstum von Computernetzen. Nachdem die Computer keine einzelnen Rechengeräte mehr waren, sondern angefangen haben, sich in große und örtlich weit verteilte Netze zusammenzuschließen, wurde auch schnell das Problem erkannt, diese Computer und die Netzgeräte verwalten (d.h. überwachen und betreuen) zu müssen. So wurde das Bedürfnis nach einem neuen Protokoll klar, das die Verwaltung von Netzteilnehmern unabhängig von ihrem Platz im Netz und dem Netztyp ermöglichte.

Der Informationsaustausch des SNMPs findet mittels kurzer Nachrichten statt, die als „Protocol Data Units (PDUs)“ bezeichnet werden. Jede Nachricht kann als ein Satz von Variablen betrachtet werden, die sowohl ihre Namen als auch ihre Werte enthalten. Es gibt drei Typen von SNMP Nachrichten: Lese-PDU, Schreib-PDU und asynchrone Nachrichten der Variablenänderung (Traps). Das SNMP Protocol schließt drei Schlüsselelemente ein: MIB (Management Information Base), Agenten und Manager (Bild 4.5).

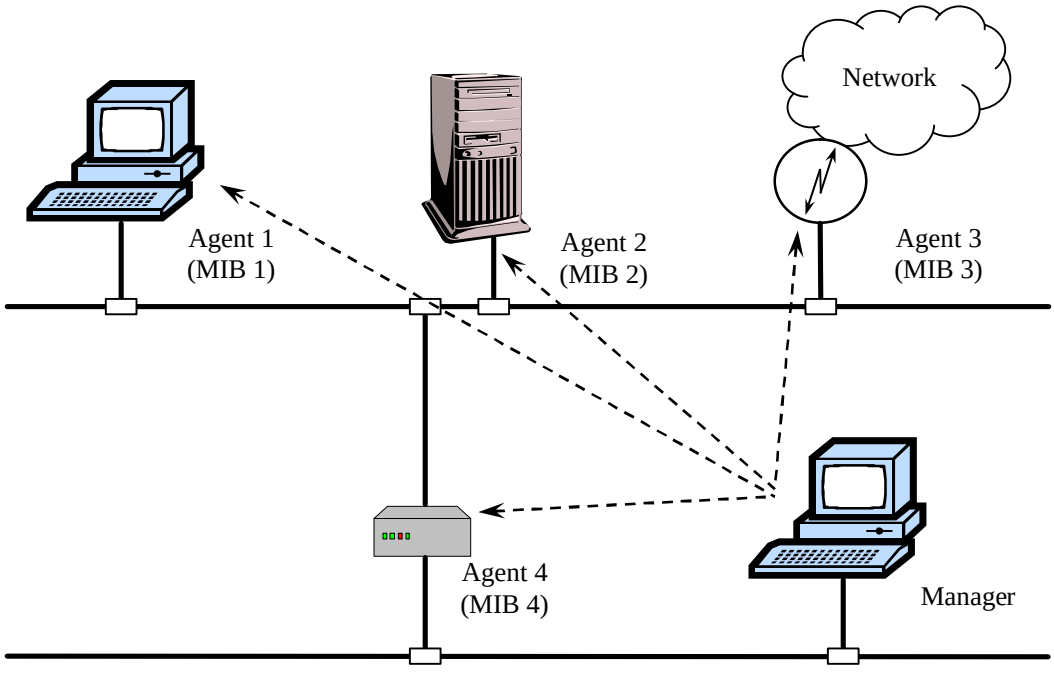


Bild 4.5. Die SNMP Komponenten

Jeder Netzteilnehmer besitzt einen eingebauten SNMP Agent, der entsprechend seiner MIB Lokal- und Netzinformationen sammelt. Der Manager, der sich in jedem beliebigen Netzsegment befinden kann, ist in der Lage alle Agenten abzufragen und damit die notwendigen

und für ihn interessanten Informationen zu sammeln. Er kann die einzelnen Agenten auch durch die Änderung ihrer Zustandsvariablen steuern.

Einzelne Variablen des Agenten werden Steuerungsobjekte genannt. Ein Satz solcher Steuerungsobjekte in einem Agenten stellt die MIB dieses Agenten dar. Die MIB kann auch als eine Art Datenbank betrachtet werden. Alle MIB Objekte besitzen die mit ihnen assoziierten Zahlen, die für ihre eindeutige Identifikation verwendet werden. Diese Zahlen nennt man Objekt ID's.

Die Objekt ID basiert ihrerseits auf einem hierarchischen Nummerierungsschema und erlaubt, die MIB Variablen einzuordnen (Bild 4.6). Die Einordnung der Variablen im MIB bedeutet, dass, wenn man im Besitz nur einer einzigen Objekt ID ist, man dennoch die Objekt ID's der folgenden Variablen im MIB bestimmen kann.

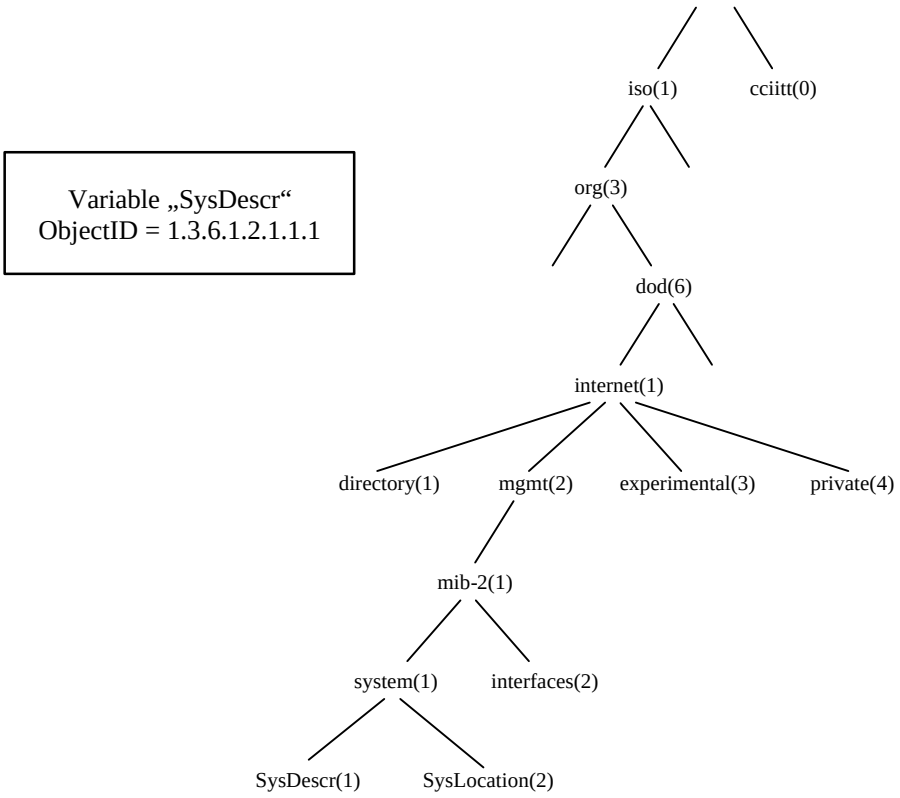


Bild 4.6. Die hierarchische Struktur des MIB's

Die Einordnung der Variablen im MIB entspricht konzeptionell der Elementeindizierung in einer Datenbank.

Jede MIB Variable hat ein Statusbit, das bestimmt, ob die Variable nur zum Lesen oder zum Lesen und Schreiben zur Verfügung steht.

Einige MIB Variable oder sogar einzelne Zweige des MIB Baums sind standardisiert und damit für bestimmte Variablen reserviert. Die Mehrheit ist aber frei für weitere Nutzung verfügbar.

Zurzeit befindet sich die dritte Version des Protokolls (SNMPv3) in der Einführung. In dieser Version wurde die Sicherheit des Protokolls nochmals verbessert, das Protokoll durch das

Verfahren zur Authentifizierung der Clients erweitert und die Unterstützung für die neue Version des IP Protokolls IPv6 eingebaut.

Basierend auf den obenerwähnten Eigenschaften des SNMP Protokolls kann man folgende Bewertungen für die Erfüllung der in Kapitel 4.2 gestellten Anforderungen verteilen. Da das SNMP Protokoll eine offene Spezifikation hat und Implementierungen für alle gängigen Plattformen vorweist, erhält es in den entsprechenden Kategorien die höchste Note. Die nächste Anforderung an das automatische Entdecken, Einloggen- und die Abonniermöglichkeit der Dienste wird nicht erfüllt und deswegen bekommt das SNMP Protokoll hier die schlechteste Note. In allen übrigen Kategorien kann das SNMP Protokoll jeweils die beste Bewertung erreichen, weil es eine datenbankähnliche Verwaltung der Prozessvariablen und einen Lese- und Schreibzugriff, genauso wie die Authentifizierung von Clients und ihrer Zugriffsrechte als auch die Unterstützung des „Multisession“ Betriebs aufweist.

Aufgrund dieser Informationen lässt sich eine Bewertungstabelle für das SNMP Protokoll (Tabelle 4.5) zusammenstellen.

Tabelle 4.5. Die Bewertungstabelle des SNMP Protokolls

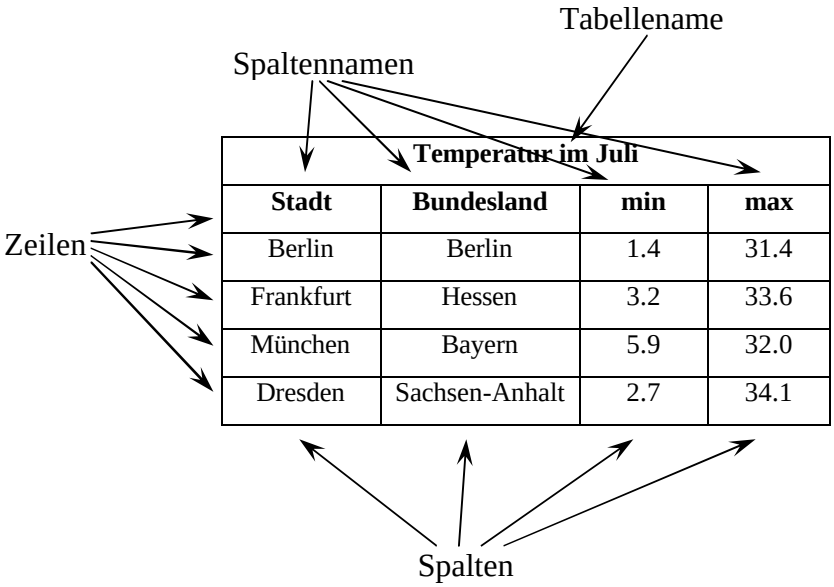
Anforderungen	SNMP
weite Verbreitung des Protokolls	++
Offenheit des Protokolls	++
Gewährleistung des automatischen Entdeckens der installierten Services-Gateways, der Möglichkeit in sie einzuloggen und ihre Dienste abonnieren zu können	--
Verwaltung der Datenbank durch logische und physikalische Prozessvariablen, Gewährleistung des Lese/Schreib-Zugriffs für Clients	++
Authentifizierung der Clients und ihrer Zugriffsrechte	++[3]
Unterstützung des „Multisession“ Betriebs	++

4.2.2. Datenbank MySQL

Die Datenbank MySQL ist ein Vertreter der relationalen Datenbanken und, wie der Name schon verrät, basiert sie auf der Structured Query Language (SQL) [48,49]. Die SQL-Sprache wird zur Kommunikation mit der Datenbank benutzt und ist entsprechend ANSI (American National Standards Institute) eine Standardsprache für die Verwaltung relationaler Datenbankensysteme. Die Operatoren der SQL-Sprache erlauben Operationen wie Lesen oder Schreiben von Daten. Datenbanken mit großer Verbreitung, welche die SQL-Sprache verwenden, sind: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc.

[3] Ab SNMPv2

Eine relationale Datenbank speichert ihre Daten oder Informationen im Gegensatz zu anderen Typen von Datenbanken in Objekten, die „Tabellen“ heißen. Die Tabellen werden anhand ihrer eindeutigen Namen identifiziert und bestehen aus Zeilen und Spalten. Die Spalten haben einen Namen und einen Typ von Daten, die sie aufbewahren. Die Zeilen beinhalten Sätze oder Daten der entsprechenden Spalten (Bild 4.7).



Temperatur im Juli

Stadt	Bundesland	min	max
Berlin	Berlin	1.4	31.4
Frankfurt	Hessen	3.2	33.6
München	Bayern	5.9	32.0
Dresden	Sachsen-Anhalt	2.7	34.1

Bild 4.7. Ein Beispiel einer Tabelle einer relationalen Datenbank

Die Daten in Tabellen können auch auf andere Daten in anderen Tabellen verweisen und somit, bei der Betrachtung von außen, große und komplexe Strukturen darstellen. Dabei werden alle Daten in relativ kleinen und einfach aufgebauten Tabellen gespeichert. Dadurch werden die Hauptvorteile relationaler Datenbanken – große Flexibilität und Datenbearbeitungsgeschwindigkeit - garantiert.

Das Protokoll zum Datenaustausch mit der relationalen Datenbank besteht hauptsächlich aus dem Versenden von kurzen Programmen (SQL-Anfragen) an den Server und aus dem Empfang der Antworten durch den Client. In Bild 4.8 ist eine Anfrage nach Städten mit der maximal registrierten Temperatur über 33 Grad Celsius und entsprechende Ergebnisse aus der Beispieltabelle von Bild 4.7 zu sehen.

```
select Stadt, max
from „Temperatur im Juli“
where max > 33;
```

Frankfurt	Hessen	3.2	33.6
Dresden	Sachsen-Anhalt	2.7	34.1

Bild 4.8. Ein Beispiel einer SQL-Anfrage und ihre Ergebnisse

Der in diesem Kapitel betrachtete Vertreter der relationalen Datenbanken MySQL ist die populärste und am weitesten verbreitete Datenbank mit offenem Quellkode. Dieses anfangs von einer Enthusiastengruppe entwickelte Projekt ist mit der Zeit zu einem leistungsfähigen, flexiblen und für den kommerziellen Einsatz reifen Produkt herangewachsen.

Die Hauptbesonderheiten von MySQL sind:

- Verfügbarkeit des Quellkodes auf C und C++, die auf einem breiten Spektrum von Compilern getestet wurden
- Unterstützung einer großen Anzahl von unterschiedlichen Softwareplattformen: Linux, Windows, FreeBSD, SunOS, Solaris, MacOS X und vielen anderen
- Verfügbare API für die Programmiersprachen C, C++, Java, Perl, Tcl, PHP u.a.
- nach Geschwindigkeit und Speicherbedarf optimierte Unterprogramme, Unterstützung von Multiprozessorsystemen
- flexibles System an Zugriffsprivilegien und Kennworten. Verschlüsselter Kennwortaustausch zwischen Server und Client
- Unterstützung riesiger Datenbanken, in Abhängigkeit von der benutzten Plattform bis zu 8 Terrabyte (10^{12} Byte). Ein Beispiel aus der Praxis: eine Datenbank mit 60,000 Tabellen und 5,000,000,000 Zeilen
- Unterstützung der ODBC (Open Database Connectivity) für die Win32 Plattformen

Anhand der oben aufgeführten Eigenschaften lässt sich die Erfüllung der in Kapitel 4.2 beschriebenen Anforderungen von Seiten der MySQL Datenbank bewerten. In der ersten Kategorie, der Verbreitung der Datenbank, erhält MySQL die Note „ausreichend“, da sie, obwohl relativ weit verbreitet, jedes Mal eine gewisse Anpassung an die neue Plattform benötigt. Dafür bekommt sie für die Offenheit die beste Note, weil sie im Quellkode verfügbar ist. Da die MySQL Datenbank eine gewöhnliche Applikation des Betriebssystems darstellt, delegiert sie die Funktionen des automatischen Entdeckens, Einloggens- und die Abonniermöglichkeiten an das Betriebssystem weiter und erfüllt daher nicht die entsprechende Anforderung. In allen übrigen Kategorien erhält die MySQL dank ihrer Datenbankherkunft die beste Bewertung, weil die Verwaltung der Variablen, die Authentifizierung von Clients und ihrer Zugriffsrechte und ein „Multisession“ Betrieb zu den Grundfunktionen einer Datenbank gehören.

Daraus ergibt sich die folgende Bewertungstabelle für die MySQL Datenbank:

Tabelle 4.6. Die Bewertungstabelle der MySQL

Anforderungen	MySQL
weite Verbreitung des Protokolls	**O**
Offenheit des Protokolls	++
Gewährleistung des automatischen Entdeckens der installierten Services-Gateways, der Möglichkeit in sie einzuloggen und ihre Dienste abonnieren zu können	--
Verwaltung der Datenbank durch logische und physikalische Prozessvariablen, Gewährleistung des Lese/Schreib-Zugriffs für Clients	++
Authentifizierung der Clients und ihrer Zugriffsrechte	++
Unterstützung des „Multisession“ Betriebs	++

4.3. Protokoll für die Visualisierungssoftware

Bei der Auswahl des Protokolls für die Visualisierungssoftware wurden sowohl die allgemeinen als auch die in den Kapiteln 3.4 und 3.4.2 formulierten Anforderungen berücksichtigt. Diese sehen wie folgt aus:

- weite Verbreitung des Protokolls
- Offenheit des Protokolls
- Gewährleistung der flexiblen und freien Positionierung von Visualisierungselementen
- Integration vorgefertigter Visualisierungsmodule
- Unterstützung des bidirektionalen Informationsaustauschs zwischen der Visualisierungssoftware und den anderen Softwarekomponenten des Systems
- Unterstützung der verschiedenen Benutzerschnittstellen für Eingabe/Ausgabe

Bei der Auswahl möglicher Kandidaten für das Protokoll der Visualisierungssoftware wurde der Blick zuerst auf das Protokoll des World Wide Web’s (WWW) – die Hyper Text Markup Language (HTML) (detailliert beschrieben in [50,51]) – mit dem entsprechenden Protokoll zu seiner Übertragung HTTP (Hyper Text Transfer Protocol) [52,53] gerichtet. Ein Beispiel für eine Seite in HTML und ihres Aussehen in einem Internetbrowser ist aus Bild 4.9 zu entnehmen.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Text um Grafik fliessen lassen</title>
</head>
<body>

<h1><img src="haus2.jpg" width="311" height="194" align="left" vspace="10"
hspace="20" alt="Text?">Ein Text</h1>

<p>Manche Texte erschlie&szlig;en sich nur aus der n&ouml;tigen Distanz. So-
gar dann, wenn sie selber genau das Gegenteil davon behaupten. Aber so ist das
eben mit den Widerspr&uuml;chen.</p>

</body>
</html>
```

Bild 4.9. Beispiel für eine Seite in HTML und ihres Aussehen in einem Internetbrowser

Bei genauer Betrachtung der Eigenschaften der HTML/HTTP Kombination hat sich herausgestellt, dass sie bei der Verwendung von Cascading Style Sheet (CSS), Extensible Markup Language (XML), Javascript und Java applet allen Anforderungen gerecht wird und dafür die besten Noten in allen Punkten erhalten (Tabelle 4.7).

Tabelle 4.7. Die Bewertungstabelle der HTML

Anforderungen	HTML
weite Verbreitung des Protokolls	++
Offenheit des Protokolls	++
Gewährleistung der flexiblen und freien Positionierung von Visualisierungselementen	++
Integration der vorgefertigten Visualisierungsmodule	++
Unterstützung des bidirektionalen Informationsaustauschs zwischen der Visualisierungssoftware und den anderen Softwarekomponenten des Systems	++
Unterstützung der verschiedenen Benutzerschnittstellen für Eingabe/Ausgabe	++

In einem solchen Fall macht die weitere Suche nach möglichen Kandidaten wenig Sinn, da sie bestenfalls genau so gut wie die HTML/HTTP Kombination sein können.

4.4. Schlussfolgerungen für die Auswahl der Protokolle

In Tabelle 4.8 sind die wesentlichen Eigenschaften aller in den letzten Kapiteln betrachteten Protokolle zusammengefasst.

Tabelle 4.8. Zusammenfassende Bewertungstabelle möglicher Protokolle für das *Services*-Gateway

Anforderungen	UPnP	OSG	OPC
weite Verbreitung des Protokolls	++	**O**	-
Offenheit des Protokolls	++	++	**O**
Unterstützung einer Betriebsart mit einer festen Adresse im Backbone oder mit ihrer automatischen Zuteilung bei der Inbetriebnahme	++	--	--
Gewährleistung des automatischen Entdeckens der installierten Services-Gateways im Backbone	++	**O**	--
Authentifizierung von Clients, Zuteilung von Zugriffsrechten	--	++	--
Versenden der Beschreibung von unterstützten Services auf Anfrage vom Client	++	++	++
Unterstützung des bidirektionalen asynchronen Informationsaustauschs mit dem Client	++	++	++

Aufgrund dieser Tabelle ergibt sich folgende Reihenfolge: 1. UPnP; 2. OSG; 3. OPC.

Die zusammenfassende Bewertungstabelle für den *StateMirror* (Tabelle 4.9) führt zu folgender Reihenfolge: 1. SNMP; 2. MySQL

Tabelle 4.9. Zusammenfassende Bewertungstabelle der Protokolle für den *StateMirror*

Anforderungen	SNMP	MySQL
weite Verbreitung des Protokolls	++	**O**
Offenheit des Protokolls	++	++
Gewährleistung des automatischen Entdeckens der installierten Services-Gateways, der Möglichkeit in sie einzuloggen und ihre Dienste abonnieren zu können	--	--
Verwaltung der Datenbank durch logische und physikalische Prozessvariablen, Gewährleistung des Lese/Schreib-Zugriffs für Clients	++	++
Authentifizierung der Clients und ihrer Zugriffsrechte	++	++
Unterstützung des „Multisession“ Betriebs	++	++

Somit werden für den in Kapitel 5 beschriebenen Realisierungsentwurf folgende Protokolle verwendet:

*Services*Gateway → **UPnP**

StateMirror → **SNMP**

Visualisierungs-software → **HTML/HTTP**

5. Realisierungsentwurf

In den vorherigen Kapiteln wurden das Konzept des Managementsystems für die Heimautomatisierung und die Vorstellungen über ihren modularen Aufbau vorgestellt, und die Protokolle für die Hauptkomponenten ausgewählt. Damit ist ein Hauptteil der zum Realisierungsentwurf benötigten Informationen vorbereitet. Als Nächstes muss man sich für eine bestimmte Hardware- und Softwareplattform für die Implementierung entscheiden. Um die Entwicklungszeit und die Ausgaben für die Entwicklung der neuen Hardware zu reduzieren, wurde der PC als Hardwareplattform für die einzelnen Komponenten ausgewählt. Obwohl diese Plattform für unsere Zwecke einen „Overkill“ darstellt, ist sie jedoch ein unverzichtbares Mittel in den Anfangsentwicklungsstadien, weil es anfangs äußerst schwierig ist, die für das zukünftige System benötigten Hardware- und Softwareressourcen richtig einzuschätzen. Die Vorteile des Computers aus dieser Sicht sind unwiderlegbar – seine offene Architektur erlaubt schnell und relativ günstig die eine oder andere Hardware- oder Softwarekomponente auszutauschen oder nachzurüsten. Später, als der Entwurf der Softwarekomponenten abgeschlossen war, wurde der Versuch unternommen, die Software auf andere „embedded“ Hardwareplattformen zu portieren. Darüber wird eingehend im nächsten Kapitel berichtet.

Als Softwareplattform wurde am Anfang das Betriebssystem „Windows 2000 Professional“ der Fa. Microsoft eingesetzt, das später mit dem Erscheinen der neueren Version durch „Windows XP Professional“ ersetzt wurde. Die Auswahl dieses Betriebssystems wurde in erster Linie von am Lehrstuhl vorhandenen Entwicklungswerkzeugen und von in langen Jahren gesammeltem „Know-how“ bestimmt. Auch haben die gute Dokumentation und die verschiedenen Informationsquellen über „Windows“ als Folge seiner weiten Verbreitung zur Entscheidung beigetragen. Aus wirtschaftlicher Sicht ist auch eine Implementierung des Managementsystems für das Betriebssystem „Linux“ zweifellos von Interesse.

Basierend auf dem vorhandenen „Know-how“ wurde der in diesem Kapitel beschriebene Realisierungsentwurf des Managementsystems für die Heimautomatisierung am Beispiel des Automatisierungssystems „EIB“ (siehe Anhang A) ausgeführt. Der EIB wurde in den Projekten des Instituts der letzten Jahre eingesetzt.

5.1. EIB-*Services*-Gateway

Zuerst muss, als Folge seiner zentralen Rolle, ein *Services*-Gateway für die Anbindung des EIBs an den Backbone – EIB-Services-Gateway – realisiert werden. In Bild 5.1 ist die schematische Darstellung des Zusammenspiels der benutzten Computerhardware und der Software des EIB-*Services*-Gateways dargestellt.

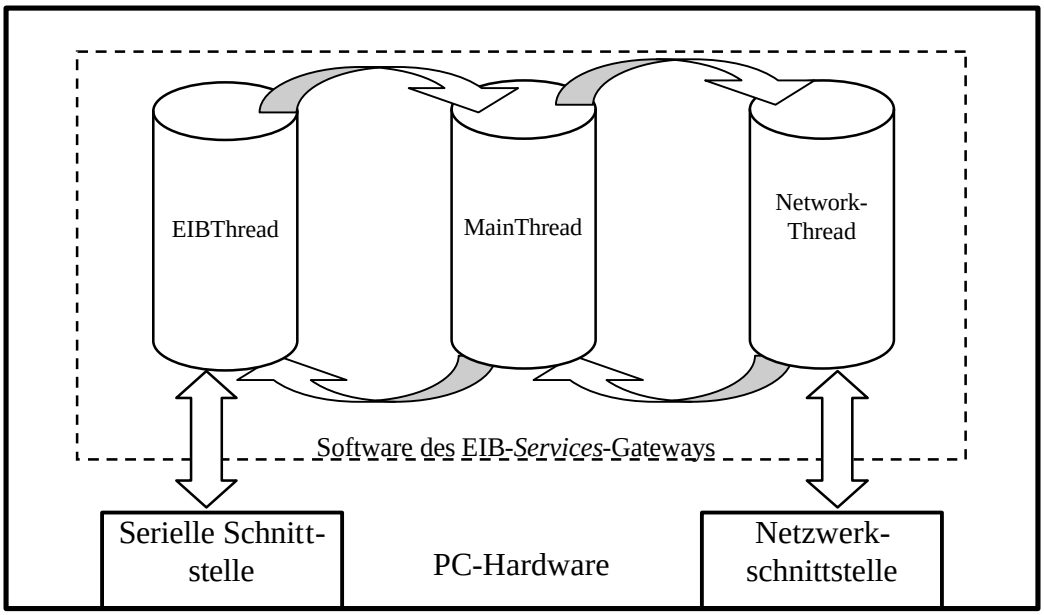


Bild 5.1. Komponenten des EIB-*Services*-Gateways

Das EIB-*Services*-Gateway benutzt die Netzwerkschnittstelle (Ethernet) des Computers für die Kommunikation mit den Clients. Der Zugriff auf den EIB erfolgt über die serielle Schnittstelle (RS232). Der Aufbau der Software des EIB-*Services*-Gateways und seine Zusammenarbeit mit den Systemkomponenten des Betriebssystem „Windows“ werden detailliert im nächsten Unterkapitel erläutert.

5.1.1. Softwareaufbau

Das EIB-*Services*-Gateway wurde als sog. „Dienst“ des Systems „Windows“ („EIB-NET“ in Bild 5.2) implementiert. Ein Dienst ist ein spezielles Programm, das schon am Start des Betriebssystems ausgeführt wird und das die ganze Zeit unsichtbar für den Anwender im Hintergrund weiter läuft.

Die Softtware des EIB-*Services*-Gateways besteht aus drei parallelen Programmen: NetworkThread – verantwortlich für die Kommunikation mit den Clients, EIBThread – gewährleistet den Informationsaustausch mit dem EIB und MainThread – synchronisiert die Arbeit der ersten beiden Programme und enthält die Hauptvariablen des Gerätes.

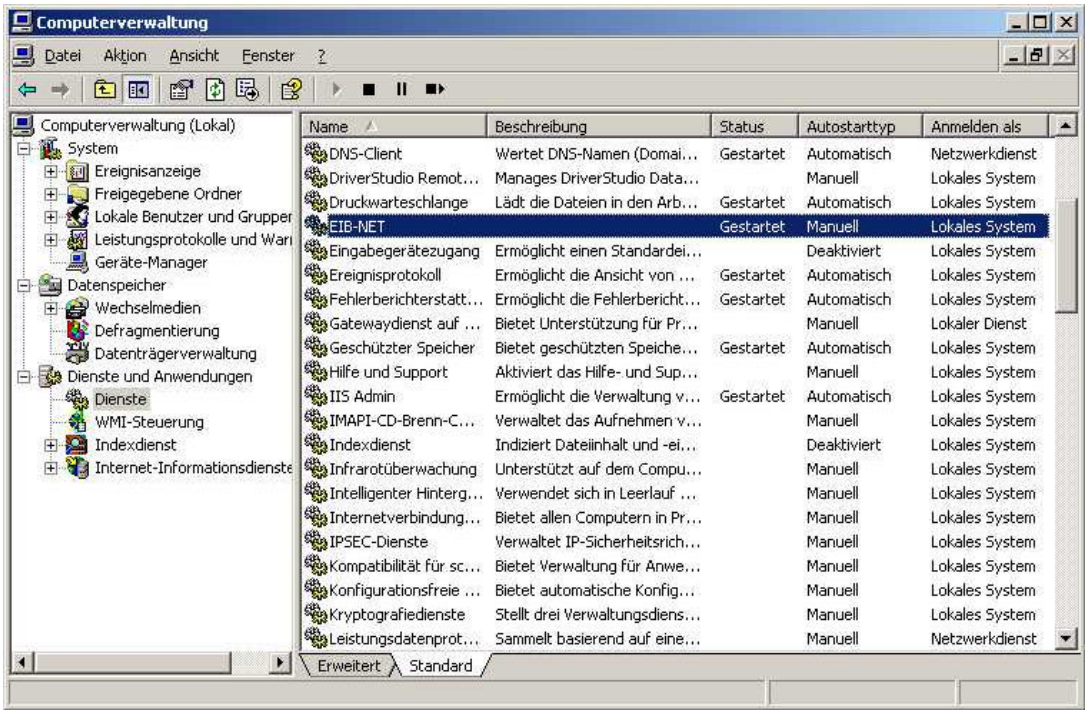


Bild 5.2. Dienst EIB-*Services*-Gateway

5.1.1.1. NetworkThread

Um die auf ihn übertragenen Aufgaben auszuführen, arbeitet der NetworkThread eng mit den Systemkomponenten des UPnP-Stacks des Betriebssystems „Windows“ zusammen. Dabei wird die „UPnP Device API“ [54] verwendet.

Bei der Entwicklung der Software des EIB-*Services*-Gateway wurde ein Beispiel des UPnP Gerätes „X10Light“ aus dem „UPnP Development Kit“ (Teil des Platform SDK) benutzt, das von der Fa. Microsoft zur Verfügung gestellt wurde.

Dieses Beispiel enthält ein strukturiertes Gerüst, bei dem jede wichtige Funktion (XML Parser, SSDP Protocol Parser, Event handler, usw.) in einer getrennten Datei abgelegt wird. Die Mehrzahl von ihnen wurde ohne jegliche Veränderung ins neu entwickelte Gerät übernommen und der Rest - durch die spezifische Implementierung der einzelnen Module (Bild 5.3) ergänzt.

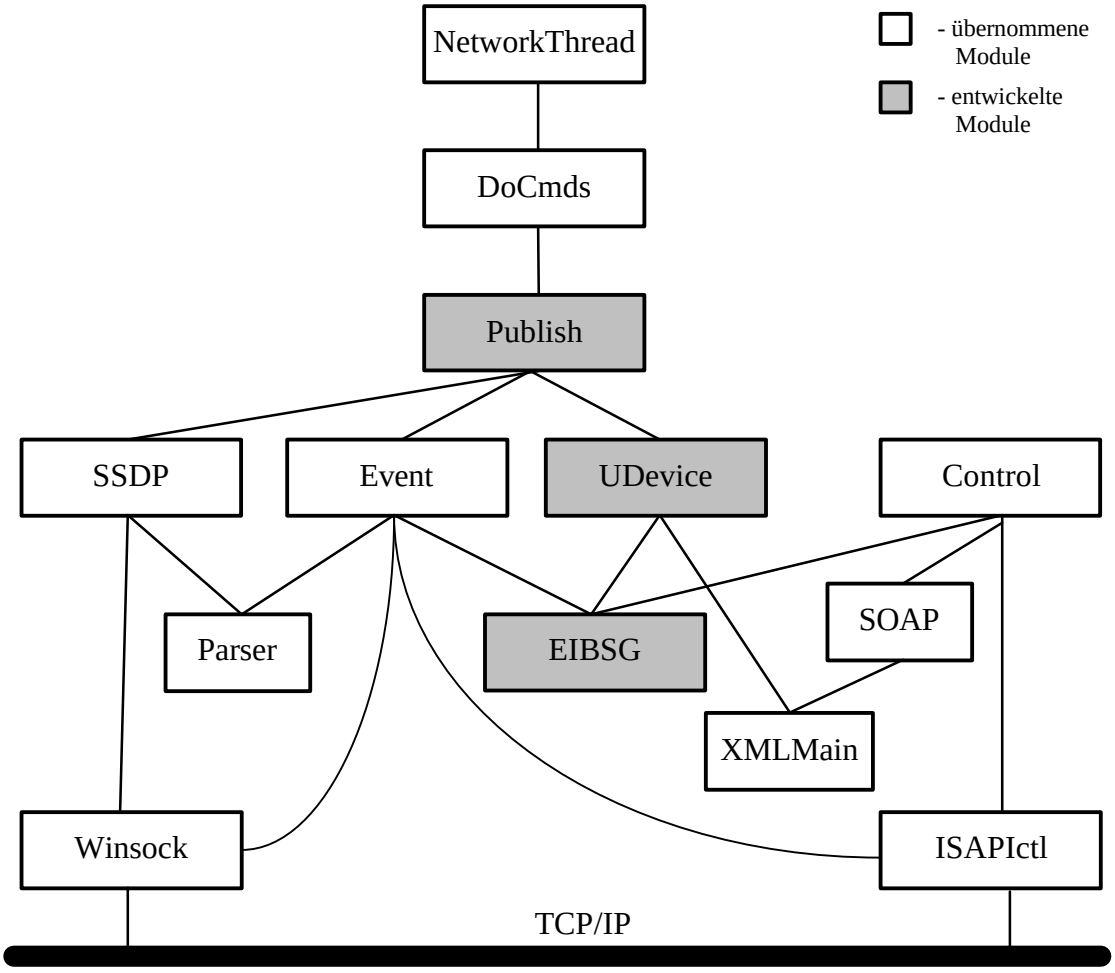


Bild 5.3. Der Aufbau des Netzwerkprogrammstroms

Die einzelnen Module führen folgende Funktionen aus:

- *Control* – bearbeitet Anfragen zur Steuerung des Geräts
- *DoCmds* – führt die Anwenderanweisungen aus
- *EIBSG* – Implementierung des EIB-*Services*-Gateways
- *Event* – steuert die Ereignismechanismen des Geräts
- *ISAPIctl* – enthält die Web Server (Microsoft IIS) Erweiterungen
- *Parser* – wertet die SSDP-Pakete aus
- *Publish* – annonciert das Gerät im Netz
- *SOAP* – bearbeitet Aktions- und Beschreibungsanfragen
- *SSDP* – bearbeitet die SSDP-Nachrichten
- *UDevice* – beschreibt die Gerätstruktur
- *Winsock* – eine Standardbibliothek von Windows zur Netzwerkkommunikation
- *XMLMain* – bearbeitet XML Dokumente

5.1.1.2. EIBThread

Der EIBThread koordiniert den Informationsaustausch mit dem gleichnamigen Automatisierungsbus. Im Verlauf des Realisierungsprozesses wurde der ganze EIB Stack nachgebildet, um eine vollwertige und flexible Kommunikation mit dem Bus zu gewährleisten. Der Stack befindet sich in einer getrennten Bibliothek „EIBLIB“ (Bild 5.4).

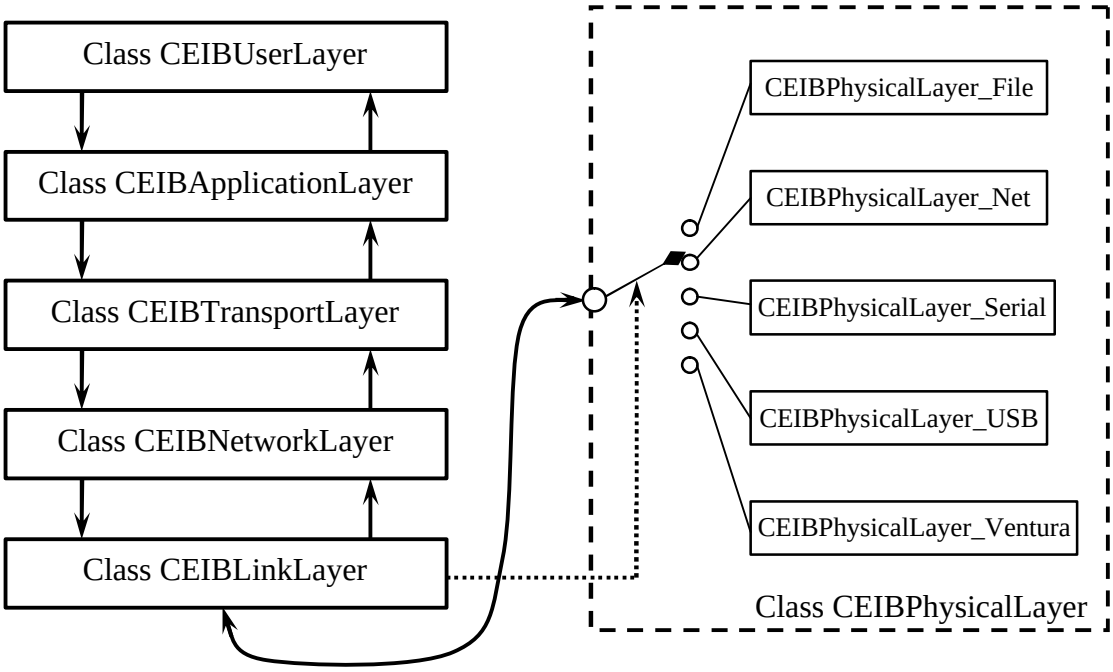


Bild 5.4. Aufbau der Bibliothek „EIBLIB“

Die Bibliothek „EIBLIB“ besteht aus mehreren Klassen, die, wie aus ihren Namen hervorgeht, den Ebenen des EIB Protokollstacks entsprechen. So zum Beispiel, enthält die Klasse `CEIBLinkLayer` unter anderem die Definition der Funktion `reqL_Data` (Bild 5.5), die dem Anwender ermöglicht, die Daten der Linkebene (`lsdu`) mit der Übertragungspriorität `prclass` an die Gruppen- (`DAF=1`) oder physikalische (`DAF=0`) Adresse `destaddr` zu verschicken.

Die Benachrichtigungen der Klasse `CEIBLinkLayer` kann der Anwender auf zwei Wegen erlangen: entweder übergibt er bei der Klasseninitialisierung (Aufruf der Funktion `Init`) einen nicht Nullzeiger auf die Ereignisfunktion `event_proc` und eine entsprechende Maske für die gewünschten Ereignisse `EventMask` oder er erzeugt eine eigene von der `CEIBLinkLayer` geerbte Klasse und überschreibt die entsprechenden Ereignisfunktionen. Dies könnte zum Beispiel `indL_Data`, um eine Nachricht über jedes auf dem Bus übertragene Telegramm zu bekommen, oder `conL_Data`, um eine Nachricht über die erfolgreiche oder fehlgeschlagene Datenübertragung zu empfangen.

```
class CEIBLinkLayer
{
       friend class CEIBPhysicalLayer;
public:
       void reqL_Data(int destaddr, char DAF, eib_priority prclass,
                        const unsigned char *lsdu);
       void reqL_Poll_Data(int destaddr, int nPollDataExpected);
       int Init(eib_pl_type pl, void *param, eib_event_proc event_proc,
   BYTE EventMask);
       ... ... ...
protected:
       virtual void indL_Data(int srcaddr, int destaddr, char DAF,
                              eib_priority prclass, const unsigned char *lsdu);
       virtual void conL_Data(con_status l_status, const unsigned char *lpdu);
       ... ... ...
private:
       ... ... ...
       CEIBPhysicalLayer *eib_pl = NULL;
};
```

Bild 5.5. Definition der Klasse `CEIBLinkLayer`

Die Klasse `CEIBLinkLayer` enthält auch einen Zeiger `eib_pl` auf die Klasse der physikalischen Ebene `CEIBPhysicalLayer`, der für die physikalische Übertragung der in der Klasse `CEIBLinkLayer` vorbereiteten Daten verantwortlich ist. Bei der Klasseninitialisierung (Funktion `Init`) wird dem Zeiger `eib_pl` eine entsprechende Klasseninstanz einer von den fünf zurzeit unterstützten Buszugriffschnittstellen zugewiesen. Welche von ihnen wirklich zum Einsatz kommt, bestimmt der Anwender in dem Parameter `pl` des Typs `eib_pl_type` beim Aufruf der Funktion `Init`. Gegenwärtig werden folgende Typen unterstützt:

- `EIB_PL_TYPE_FILE` – der Buszugriff wird über eine Datei simuliert
- `EIB_PL_TYPE_NET` – der Buszugriff übers Netz
- `EIB_PL_TYPE_SERIAL` – über die standardisierte serielle Schnittstelle der Fa. Siemens
- `EIB_PL_TYPE_USB` – über die USB Schnittstelle
- `EIB_PL_TYPE_USER` – über eine beliebige Anwenderschnittstelle (in diesem Fall muss der Anwender eine eigene Implementierung der Klasse `CEIBPhysicalLayer` für diese Schnittstelle vornehmen)
- `EIB_PL_TYPE_VENTURA` – über die am Lehrstuhl entstandene Schnittstelle „Ventura“ (siehe Kapitel 6)

Die nächste Klasse `CEIBUserLayer` aus der Bibliothek `EIBLIB` bietet dem Anwender eine Reihe von „Highlevel“ Funktionen für den Buszugriff (Bild 5.6).

```
class CEIBUserLayer : public CEIBApplicationLayer
{
public:
       eib_error GetDeviceDescriptor(int addr, int *device_descriptor);
       eib_error ReadMemory(int addr, int memory_addr,
                            int memory_number, unsigned char *buf);
       eib_error GetADCValue(int addr, unsigned char channel_num,
                             unsigned char read_count, int *sum);
       ... ... ...
       eib_error SetGroupValue(int groupaddr, eib_data_type typ,
                               unsigned char *data);
       eib_error GetGroupValue(int groupaddr, eib_data_type typ,
                               unsigned char *data);
       ... ... ...
private:
       ... ... ...
};
```

Bild 5.6. Definition der Klasse `CEIBUserLayer`

So kann der Anwender, z. B. die Funktion `ReadMemory` benutzen, um den Speicher eines BCU auszulesen, anstatt selber eine Verbindung zum Gerät aufzubauen, die Anfragen zu verschicken, den Empfang von Antworten zu bestätigen usw.

5.1.1.3. MainThread

Die Aufgaben des Hauptprogramms liegen hauptsächlich in der Synchronisierung der Zusammenarbeit der beiden oben beschriebenen Programme und in der Verwaltung der globalen Gerätevariablen, so z.B., der Datenbestand von Clients `Clients[MAX_CONNECTIONS]` oder den Eventhandle `m_hEIBEvent` und `m_hNetworkEvent` für die Synchronisierung mit den entsprechenden Programmen. Der Datenbestand der Clients hat den Typ `recEibNetClient` und ist folgendermaßen definiert:

```
struct recEibNetClient
{
     int SessionID;             // Session ID Nummer, zur
                                    Identifizierung des Clients
     int LastCallTime;          // Zeitpunkt des letzten
                                    Aufrufs (Timeoutdetektion)
     CEIBUserLayer *eiblayers; // Zeiger auf den EIB Stack des
                                    Clients
};
```

Der Programmablauf des Hauptprogramms ist aus Bild 5.7. zu entnehmen.

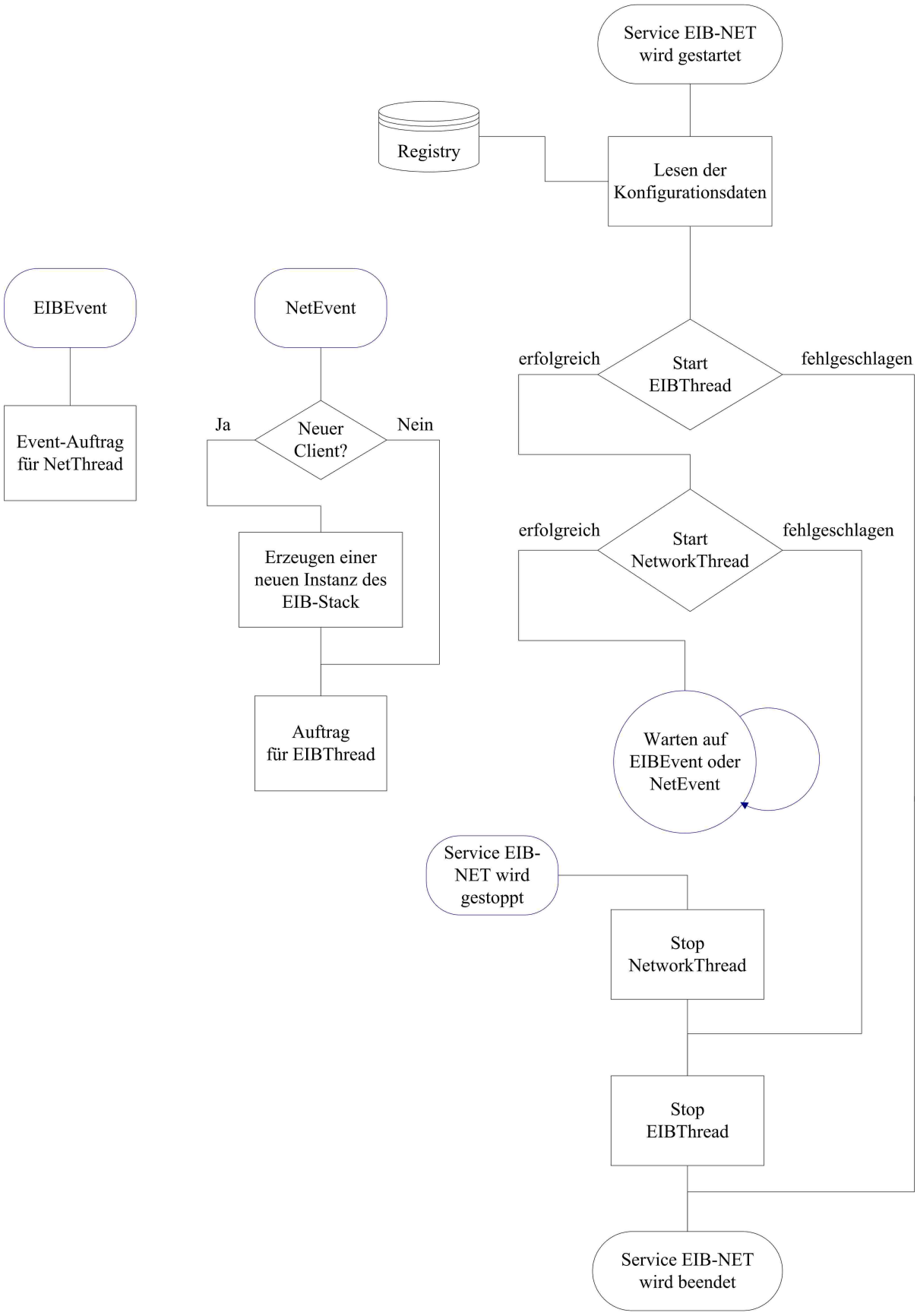


Bild 5.7. Flussdiagramm des Hauptprogramms

5.1.2. Networking

Die Betriebsart des EIB-*Services*-Gateways im Netz entspricht der UPnP Spezifikation und ist dadurch bedingt in sechs Schritte aufgeteilt, die in Kapitel 4 bei der Protokollübersicht aufgeführt sind.

Der erste Schritt „Adressierung“ wird in unserem Fall deutlich vereinfacht, weil das EIB-*Services*-Gateway die IP Adresse des Computers übernimmt, auf dem es gerade läuft.

Im zweiten Schritt startet der im Netz aufgetauchte Kontrollpunkt die Suche nach allen verfügbaren oder bestimmten UPnP Geräten. Dabei meldet das EIB-*Services*-Gateway seine Präsenz im Netz.

Im dritten Kommunikationsschritt „Beschreibung“ fordert der Kontrollpunkt die Beschreibung von allen Geräten, die sein Interesse erweckt haben. Wenn es das EIB-*Services*-Gateway ist, schickt es an ihn seine Beschreibung (Versionsnummer, Herstellername, Modellbeschreibung etc.) und die Beschreibung der verfügbaren Dienste (Bild 5.8).

```
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>

  <device>
    <UDN>uuid:d21586ad-abe7-46be-961f-b0f04b25e195</UDN>
    <friendlyName>EIB-Services-Gateway</friendlyName>
    <deviceType>urn:schemas-upnp-org:device:gateway:1</deviceType>
    <presentationURL>../presentation/NoPresentation.html</presentationURL>
    <manufacturer>Ky</manufacturer>
    <manufacturerURL>http://www.emt.ei.tum.de/</manufacturerURL>
    <modelName>EIB-Services-Gateway</modelName>
    <modelNumber>A1</modelNumber>
    <modelDescription>Gateway to the EIB network</modelDescription>
    <modelURL>http://www.emt.ei.tum.de/</modelURL>
    <UPC>000000000001</UPC>
    <serialNumber>0000001</serialNumber>
    <iconList>
      <icon>
        <mimetype>image/png</mimetype>
        <width>32</width>
        <height>32</height>
        <depth>8</depth>
        <url>../images/EIBSG.png</url>
      </icon>
    </iconList>

    <serviceList>
      <service>
        <serviceType>urn:schemas-upnp-org:service:LL-Services:1</serviceType>
        <serviceId>urn:upnp-org:serviceId:LLService</serviceId>
        <controlURL>../control/isapictl.dll?LLService</controlURL>
        <eventSubURL>../control/isapictl.dll?LLService</eventSubURL>
        <SCPDURL>../SCPD/EIBSG_LL-SCPD.xml</SCPDURL>
      </service>
```

Bild 5.8. Die UPnP Beschreibung des EIB-*Services*-Gateways

```
      <service>
        <serviceType>urn:schemas-upnp-org:service:AL-Services:1</serviceType>
        <serviceId>urn:upnp-org:serviceId:ALService</serviceId>
        <controlURL>../control/isapictl.dll?ALService</controlURL>
        <eventSubURL>../control/isapictl.dll?ALService</eventSubURL>
        <SCPDURL>../SCPD/EIBSG_AL-SCPD.xml</SCPDURL>
      </service>

      <service>
        <serviceType>urn:schemas-upnp-org:service:UL-Services:1</serviceType>
        <serviceId>urn:upnp-org:serviceId:ULService</serviceId>
        <controlURL>../control/isapictl.dll?ULService</controlURL>
        <eventSubURL>../control/isapictl.dll?ULService</eventSubURL>
        <SCPDURL>../SCPD/EIBSG_UL-SCPD.xml</SCPDURL>
      </service>

    </serviceList>
  </device>
</root>
```

Bild 5.8 (fort.). Die UPnP Beschreibung des EIB-*Services*-Gateways

In Bild 5.9 kann man eine Standardapplikation „Generic UPnP Control Point (Generic UCP)“ aus dem UPnP Development Kit mit dem automatisch gefundenen EIB-*Services*-Gateway und mit der Liste seiner Dienste sehen.

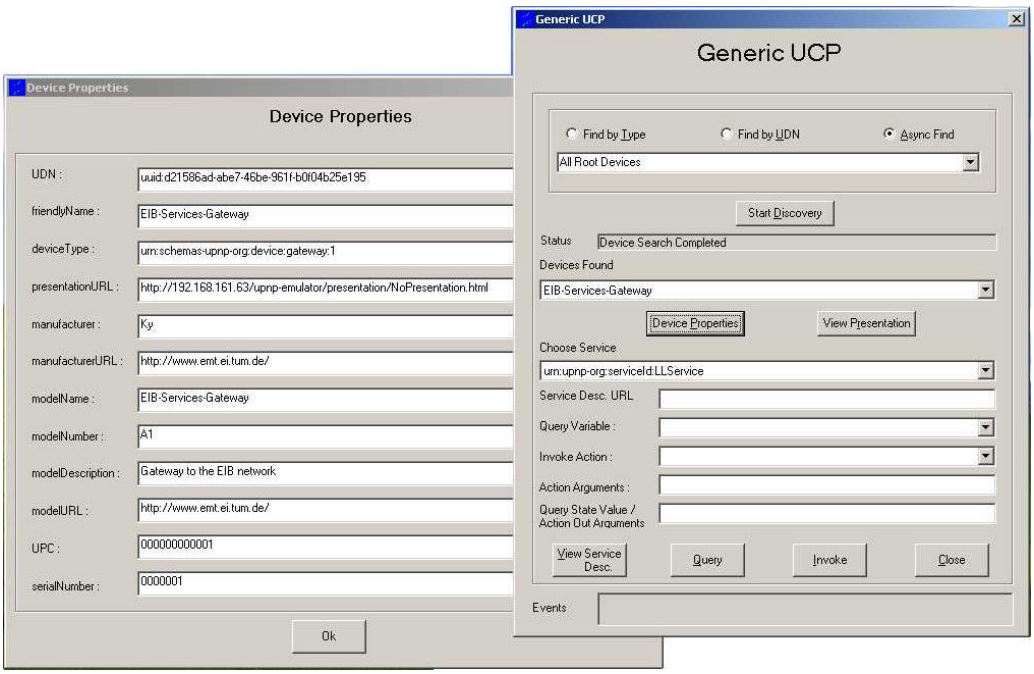


Bild 5.9. Beschreibung des EIB-*Services*-Gateways in der Kontrollpunktapplikation

Nun besitzt der Kontrollpunkt alle notwendigen Informationen, um zum nächsten Schritt „Steuerung“ überzugehen. Dafür nutzt er einen der verfügbaren Dienste des Gerätes. Die detaillierte Schilderung der stichprobenartig im EIB-*Services*-Gateway implementierten Dienste wird in Kapitel 5.1.3 angegeben.

Im fünften Schritt „Ereignisse“ der Netzbetriebsart des EIB-*Services*-Gateways werden an die Clients bestimmte Ereignisnachrichten verschickt, die sie vorher abonniert haben. Die detaillierte Beschreibung der stichprobenartig im EIB-*Services*-Gateway implementierten Ereignisse wird in Kapitel 5.1.4 aufgeführt.

Der sechste Schritt „Präsentation“ wird vom EIB-*Services*-Gateway nicht unterstützt, weil die Darstellung im Browserfenster der im EIB-*Services*-Gateway implementierten Dienste wenig Sinn macht. Die Aufgaben der Darstellung der einzelnen Zustandsvariablen des Systems sind völlig in die Visualisierungssoftware delegiert, die im Kapitel 5.4 beschrieben wird.

5.1.3. EIB Dienste

Die im EIB-*Services*-Gateway implementierten Dienste sind in Tabelle 5.1 aufgeführt. Dies sind nur drei Dienste, die aber genügen, um eine Demonstration der ausgewählten Möglichkeiten des Systems vorzuführen.

Tabelle 5.1. Liste der implementierten Dienste und ihre Beschreibung

Dienstebene	Unterdienst	Beschreibung
Link	LL_DATA_REQ	Anfrage zur Nachrichtenübertragung auf Linkebene auf dem Bus
Application	AL_GROUPVALUE_READ_REQ	Anfrage zum Lesen des Wertes der Gruppenvariable
	AL_GROUPVALUE_WRITE_REQ	Anfrage zum Schreiben des Wertes der Gruppenvariable
User	UL_READMEMORY_REQ	Anfrage zum Lesen des Gerätespeichers

5.1.4. EIB Ereignisse

Entsprechend den im EIB-*Services*-Gateway implementierten Diensten sind auch die Ereignisse (Tabelle 5.2) realisiert.

So zum Beispiel, wenn der Kontrollpunkt die Ereignisse der Linkebene abonniert, wird er in Kenntnis gesetzt, falls seine Daten erfolgreich oder nicht auf dem Bus übertragen werden (Ereignis „LL_DATA_CON“) oder irgendeine neue Nachricht auf dem Bus auftaucht (Ereignis „LL_DATA_IND“).

Tabelle 5.2. Liste der implementierten Ereignisse und ihre Beschreibung

Dienstebene	Ereignis des Dienstes	Beschreibung
Link	LL_DATA_CON	Bestätigung der Nachrichtenübertragung der Linkebene auf dem Bus
	LL_DATA_IND	Neue Nachricht auf dem Bus
Application	AL_GROUPVALUE_READ_CON	Bestätigung der Anfrageübertragung zum Lesen des Wertes der Gruppenvariable
	AL_GROUPVALUE_READ_RES	Ergebnis der Anfrage zum Lesen des Wertes der Gruppenvariable
	AL_GROUPVALUE_WRITE_CON	Bestätigung der Anfrageübertragung zum Schreiben des Wertes der Gruppenvariable
	AL_GROUPVALUE_WRITE_IND	Neue Anfrage zum Schreiben des Wertes der Gruppenvariable auf dem Bus
User	UL_READMEMORY_RES	Ergebnis der Anfrage zum Lesen des Gerätspeichers

5.2. *StateMirror*

Der *StateMirror* speichert die letzten Werte der Busvariablen und stellt sie seinen Clients zur Verfügung. Der *StateMirror* ist als eine dynamische Bibliothek (eng. „Dynamic Link Library (DLL)“) implementiert, die eine Systemkomponente des Betriebssystems „Windows“ darstellt und beim Systemstart geladen wird.

Um seinen Aufgaben nachzukommen, arbeitet der *StateMirror* eng mit dem EIB-*Services*-Gateway zusammen. Beim Start führt der *StateMirror* zunächst die Suche nach den im Netz verfügbaren *Services*-Gateways durch, dann loggt er sich in diese ein und abonniert ihre Ereignisse der Applikationsebene. Um die Werte der Variablen abzufragen oder neue Werte zu schreiben, benutzt der *StateMirror* zwei Dienste der Applikationsebene: AL_GROUPVALUE_READ_REQ bzw. AL_GROUPVALUE_WRITE_REQ.

Die Liste aller Variablen, die der *StateMirror* zu verarbeiten hat, sind in seiner Konfigurationsdatei (Bild 5.10) anzutreffen. Dort findet er auch die Informationen über die benötigten Services-Gateways und die Aufteilung der Variablen pro Gateway. Die Konfigurationsdatei wird im XML Format gespeichert.

```
<?xml version="1.0"?>
<root name=”StateMirror-config-file”>
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>

  <gatewayList>
    <gateway>
      <gatewayID>1</gatewayID>
      <deviceType>urn:schemas-upnp-org:device:gateway:1</deviceType>
      <modelName>EIB-Services-Gateway</modelName>
      <serialNumber>0000001</serialNumber>
    </gateway>

    <gateway>
      <gatewayID>2</gatewayID>
      <deviceType>urn:schemas-upnp-org:device:gateway:1</deviceType>
      <modelName>EIB-Services-Gateway</modelName>
      <serialNumber>0000002</serialNumber>
    </gateway>
  </gatewayList>

  <variableList>
    <variable>
      <variableID>1</variableID>
      <variableFlags>READ:WRITE</variableFlags>
      <variableType>UINT1</variableType>
      <variableDesc>Lampe im Netz 1</variableDesc>
      <variableAddr>1_1_0</variableAddr>
    </variable>

    <variable>
      <variableID>2</variableID>
      <variableFlags>READ:WRITE</variableFlags>
      <variableType>UINT1</variableType>
      <variableDesc>Lampe im Netz 2</variableDesc>
      <variableAddr>2_1_3</variableAddr>
    </variable>

    <variable>
      <variableID>3</variableID>
      <variableFlags>LOGICAL:READ</variableFlags>
      <variableType>UINT1</variableType>
      <variableDesc>Ist das Licht irgendwo eingeschaltet?</variableDesc>
      <variableAddr>ID_1 | ID_2</variableAddr>
    </variable>
  </variableList>
</root>
```

Bild 5.10. Beispiel der Konfigurationsdatei des *StateMirrors*

Die Konfigurationsdatei ist in mehrere Sektionen aufgeteilt. So beschreibt die Sektion „gatewayList“ die benötigten *Services*-Gateways, die der *StateMirror* versuchen wird, bei seinem Start ausfindig zu machen. Um ihre bessere Identifizierung und die saubere Unterscheidung von den anderen gefundenen Services-Gateways zu erreichen, befinden sich in der Sektion unter anderem die für das gegebene Gateway spezifischen Felder (deviceType, modelName, serialNumber), die auch in seiner Beschreibung (siehe Bild 5.8) zu finden sind.

Die Sektion „variableList“ enthält die Liste aller Variablen, die der *StateMirror* verwaltet. Jede Variable hat eine eigene systemweit eindeutige Identifikationsnummer, die im Feld „variableID“ gespeichert wird. Die Nummer wird bei der Variablenadressierung von Clients

oder bei der Verknüpfung der physikalischen Variablen mit den logischen Variablen benutzt. Das Feld „variableFlags“ enthält die durch Doppelpunkt getrennten Variableneigenschaften. Zurzeit sind drei Eigenschaften definiert: „READ“ – der Wert der Variable kann ausgelesen werden; „WRITE“ – die Variable erlaubt den Schreibzugriff; „LOGICAL“ – für eine logische Variable. Der Variablentyp wird vom Feld „variableType“ bestimmt, dessen mögliche Werte in der Tabelle 5.3. aufgelistet sind.

Tabelle 5.3. Die vom *StateMirror* unterstützten Variablentypen

Typ	Variablenlänge
INT1	1 Bit (boolesche Variable)
INT2	2 Bits
INT4	4 Bits
INT8	1 Byte
INT16	2 Bytes
INT32	4 Bytes
INT64	8 Bytes
DATA16	16 Bytes
DATA256	256 Bytes
FLOAT	4 Bytes (float point Format)
STRING	Die Zeichenkette der variablen Länge (zurzeit maximal 4 KBytes)

Im Feld „variableDisc“ befindet sich die Variablenbeschreibung. Das Feld „variablenAddr“ enthält für die physikalischen Variablen die Information, wie sie im System zu adressieren sind. In diesem Fall besteht das Feldformat aus der Gatewaykennung, über das die Variable zu erreichen ist, und aus der für den konkreten Typ des Services-Gateways spezifischen Variablenadresse im Untersystem. So z.B., enthält das Adressierungsfeld der Variable „Lampe im Netz 1“ (siehe Bild 5.10) eine Zeile „1_1_0“ – dies bedeutet, dass die Variable sich in einem Untersystem befindet, das ans Gateway mit der Kennung „gatewayID=1“ angeschlossen ist, und ihre interne (nur für das jeweilige Gateway gültige) Adresse „1_0“ ist. Dies wiederum entspricht für das EIB-*Services*-Gateway einer Gruppenvariable des EIB Systems (siehe Anhang A) in der Hauptgruppe „1“ und der Untergruppe „0“.

Falls das Feld „variableAddr“ eine logische Variable beschreibt, enthält es einen regulären Ausdruck für die Berechnung der Variablen. Ein solcher Ausdruck kann die Verknüpfungen zu den physikalischen Variablen (in Form: Präfix „ID_“ + Wert des Feldes „variableID“ der entsprechenden Variable) und auch folgende arithmetische und logische Operatoren: „+“, „-“, „*“, „/“, „&“, „|“, „^“ und „~“ aufweisen. Zum Beispiel ist der Wert der in der Liste als letzte aufgeführten Variable (variableID=3) ein Ergebnis der logischen Operation „Oder“ mit den Werten der zwei anderen Variablen – ID_1 (variableID=1) und ID_2 (variableID=2).

5.2.1. Class CStateMirror

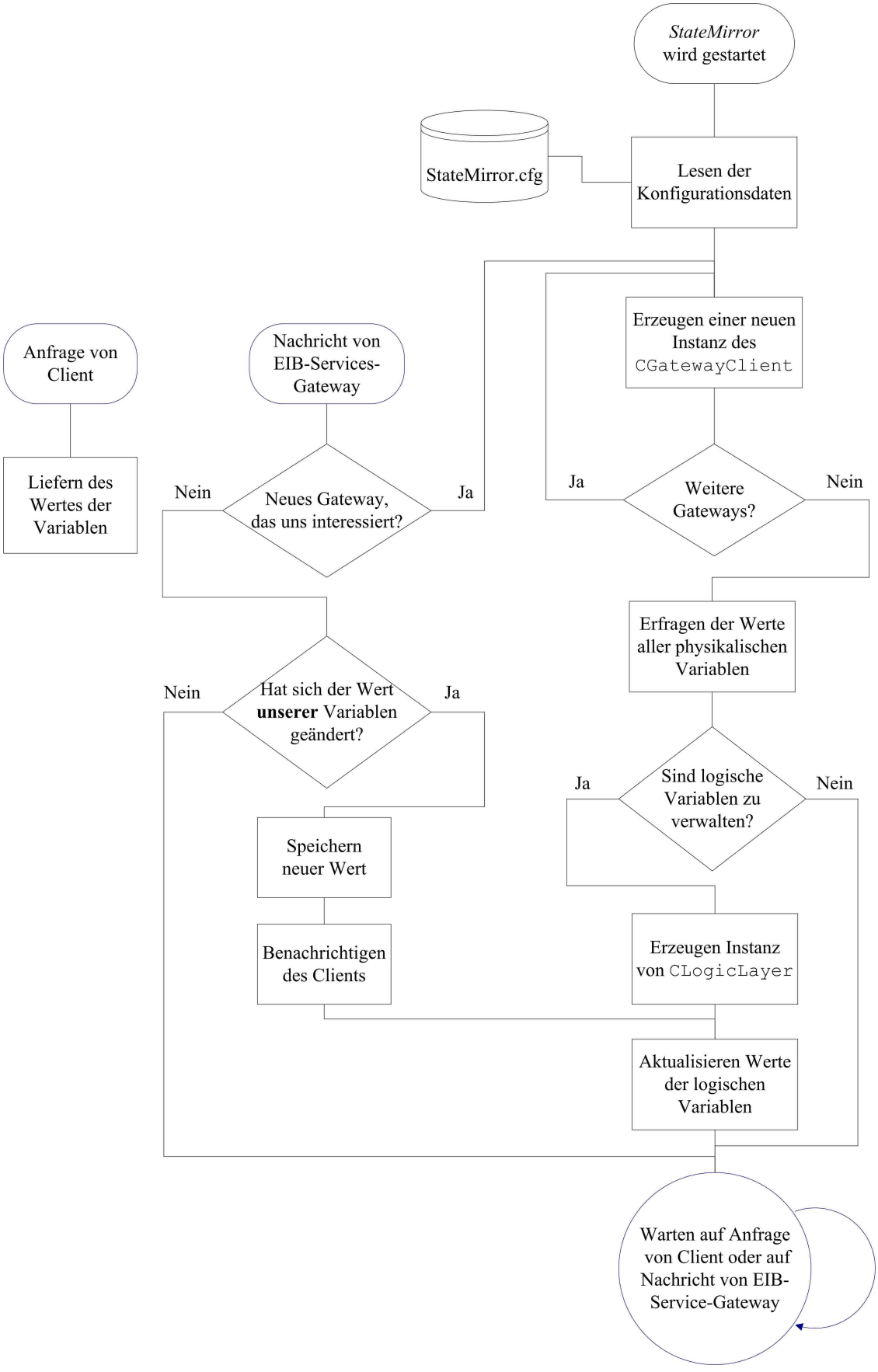


Bild 5.11. Funktionsweise der Klasse `CStateMirror`

Die Klasse `CStateMirror` enthält den Hauptteil der Funktionalität des *StateMirrors*. Seine Funktionsweise ist im Bild 5.11 dargestellt.

Als Erstes nach seinem Start liest der *StateMirror* die Konfigurationsdaten aus der Datei „StateMirror.cfg“. Danach erzeugt er für jedes dort aufgeführte Gateway eine eigene Instanz der Klasse `CGatewayClient`, die für die Kommunikation mit dem entsprechenden Gateway verantwortlich ist. Dabei wird der UPnP Control Point API [55] des Betriebssystems „Windows“ genutzt. Als Nächstes lädt der *StateMirror* alle Variablen, die mit den gefundenen Gateways assoziiert sind (die restlichen Variablen können später nachgeladen werden, nachdem entsprechende Gateways im Netz auftauchen) und fragt ihre Werte ab. Falls in der Variablenliste auch die logischen Variablen vorhanden sind, erzeugt der *StateMirror* für ihre Verwaltung eine Instanz der Klasse `CLogicLayer`, die im nächsten Unterkapitel detailliert beschrieben wird. Zur Kommunikation mit den Clients wird, wie in Kapitel 4 erwähnt wurde, das SNMP Protokoll verwendet. Dabei greift der *StateMirror* auf den SNMP Stack des Betriebssystems zurück, mit dem über die SNMP Extension Agent API [56] kommuniziert wird.

5.2.2. Class CLogicLayer

Die Klasse `CLogicLayer` ist für die Erzeugung, Speicherung und Berechnung der Werte von logischen Variablen verantwortlich. Die Definition des Schnittstellenteils der Klasse ist in Bild 5.12 dargestellt.

```
class CLogicalLayer
{
	friend class CStateMirror;
public:
	BOOL AddVariable(LPSM_VARIABLE lpCO);
	LPSM_VARIABLE GetVariableByID(int id);
	... ... ...
private:
	... ... ...
protected:
	void Event_ValueChanged(int VarID);
};
```

Bild 5.12. Schnittstellenteil der Klassendefinition von `CLogicLayer`

Die Kommunikation mit der Klasse `CStateMirror` findet über folgende drei Funktionen statt:

- `AddVariable` – fügt eine neue logische Variable hinzu
- `GetVariableByID` – gibt die Werte der Variablen mit der entsprechenden ID zurück
- `Event_ValueChanged` – wird von `CStateMirror` bei der Änderung einer beliebigen physikalischen Variable aufgerufen

Bei der Berechnung der Ausdrücke, die den Wert einer logischen Variablen beschreiben, wird die sog. „polnische“ Notation (Bild 5.13) eingesetzt. Diese Notation wurde in den 20-er Jahren vom polnischen Mathematiker Jan Lukasiewicz eingeführt. Er hat damit gezeigt, dass beim Schreiben der Operatoren vor den Operanden anstatt zwischen ihnen, man auf den Einsatz der Klammern verzichten kann. Später in den 50-er Jahren hat der Australier Charles L. Hambil ein umgekehrtes Schreibschema angeboten, in dem die Operatoren nach den Operanden stehen (postfix operators). Dieses Schema wurde „umgekehrte polnische Notation“ (eng. „Reverse Polish Notation (RPN)“) genannt. Der Vorteil eines solchen Schemas liegt darin, dass alle Operatoren in einer zur Berechnung notwendigen Reihenfolge stehen. Diese Form der „polnischen“ Notation wird auch in der Regel bei der Berechnung von Ausdrücken in Rechnern angewendet.

Algebraischer Ausdruck:	Polnische Notation:	Die umgekehrte polnische Notation:
$\frac{8+4*2}{1+3}$	/ + 1 3 + * 2 4 8	8 4 2 * + 1 3 + /

Bild 5.13. „Polnische“ Notation

Bei der Berechnung der Ausdrücke reserviert die Klasse `ClogicLayer` zuerst einen Speicherbereich für die Speicherung der Operatoren. Dieser Bereich wird nach dem Stackprinzip „letzter hinein – erster heraus“ genutzt.

Das Programm bearbeitet die Ausdrücke von links nach rechts. Wenn es einen Operanden findet, speichert das Programm ihn im Stack, dabei werden natürlich alle andere dort befindlichen Operanden verschoben (Bild 5.14).

Speicher / Operation	Stack[0]	Stack[1]	Stack[2]	Stack[3]
8	8			
4	4	8		
2	2	4	8	
*	8	8		
+	16			
1	1	16		
3	3	1	16	
+	4	16		
/	4			

Bild 5.14. Ein Beispiel für die Berechnung von regulären Ausdrücken in der umgekehrten „polnischen“ Notation

Wenn als nächstes ein Operator im Ausdruck steht, führt das Programm die entsprechende Operation mit den zwei letzten im Stack befindlichen Operanden aus. Dabei werden sie aus dem Stack entfernt und das Ergebnis der Operation im Stack gespeichert. Am Ende der Bearbeitung befindet sich im Stack nur das Ergebnis der Berechnung.

5.3. Konfigurationsprogramm

Die im Rahmen dieser Arbeit realisierte Konfigurationssoftware (genannt „*HomeConfigurator*“) ist eine WIN32 GUI Applikation für das Betriebssystem „Windows“ (Bild 5.15). Die implementierte Funktionalität des *HomeConfigurators* erlaubt dem Anwender, ihn bei der Inbetriebnahme für die Konfiguration der einzelnen Komponenten des Managementssystems einzusetzen.

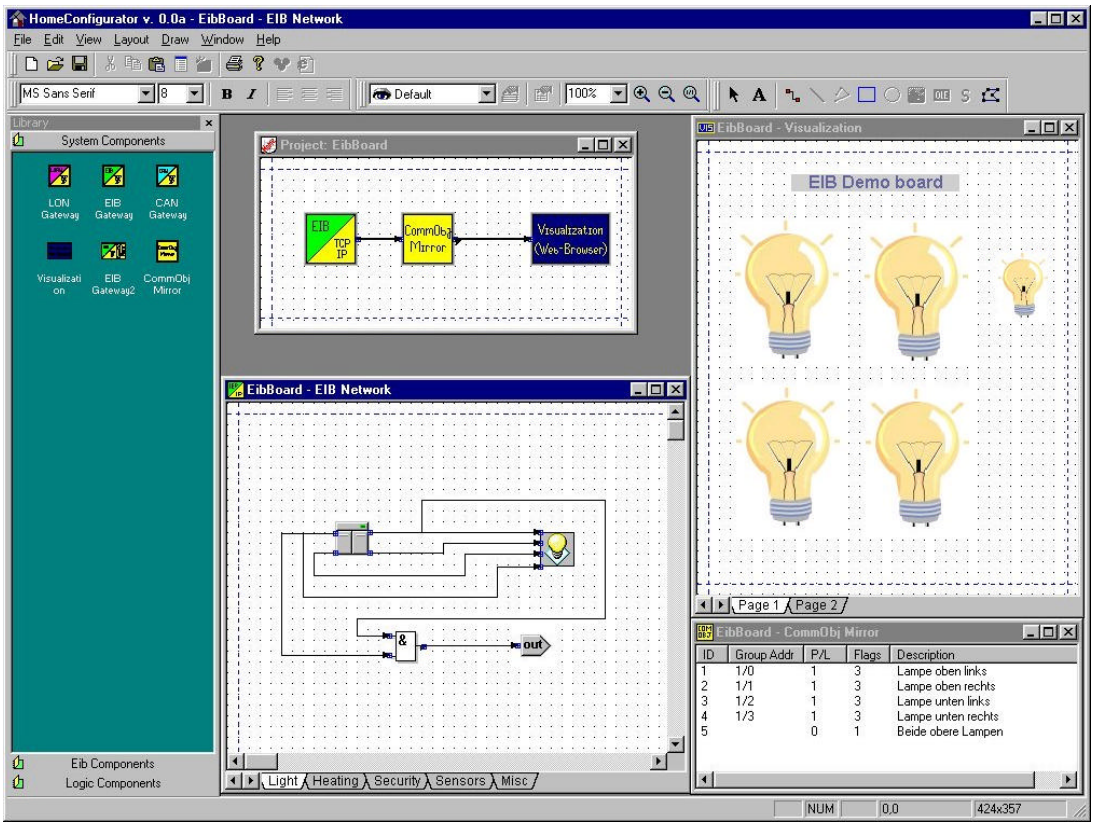


Bild 5.15. Die Bedienoberfläche der Konfigurationssoftware „*HomeConfigurator*“

Das Hauptfenster der Konfigurationssoftware beinhaltet zusammen mit den üblichen Elementen der Benutzeroberfläche (Menü, Toolbox, Instrumentenpanel) im oberen Fensterbereich auch ein Fenster mit den Komponenten der Bibliothek auf der linken Seite. In diesem Fall sind zum Beispiel drei Module geladen: das Modul der Systemkomponenten „System Components“, das Modul der EIB Komponenten „EIB Components“ und das Modul der logischen Komponenten „Logic Components“. All diese Module stellen entsprechend dem Systemkonzept (siehe Kapitel 3.4) standardisierte Softwarekomponenten dar, die beim Start des *HomeConfigurators* vom Bibliothekmanager geladen werden. Dank dieser Tatsache kann die Bibliothek in Zukunft relativ einfach erweitert und ergänzt werden.

Die weiteren vier Fenster der Benutzeroberfläche des *HomeConfigurators* gehören zu dem geladenen Anwenderprojekt „EibBoard“. Dieses Projekt beschreibt die Konfiguration der im Institut benutzten tragbaren Miniinstallation des EIB Systems (Bild 5.16).

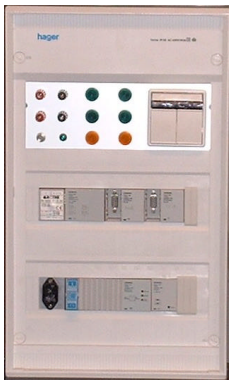


Bild 5.16. Eine tragbare EIB Miniinstallation

Dies wurde zum Kennenlernen und zum Test der experimentellen Hardware- oder Softwarekomponenten eingesetzt. Die Miniinstallation enthält einen Schalter mit vier Tasten, einen Aktor mit vier an ihn angeschlossenen Lampen und eine oder zwei serielle Schnittstellen für den Buszugriff. Außerdem enthält sie die für den elektrischen Betrieb notwendigen Komponenten, wie z.B. einen Busspannungsregler.

Das Hauptfenster des Projekts „EibBoard“ befindet sich oben links, relativ zu den anderen Fenstern des Projekts und zeigt in einer visuellen Form die aktuelle Konfiguration des Projekts. So enthält das System in diesem Fall drei Systemkomponenten: ein EIB-*Services*-Gateway (dargestellt mit einem gelb-grünen Icon), einen *StateMirror* (gelbes Icon) und eine Visualisierungskomponente für den Web-Browser (blaues Icon).

Jede der im Hauptfenster befindlichen Systemkomponenten entsprechen einem der drei anderen Projektfenster. So stellt das rechts unten befindliche Fenster die gegenwärtige Konfiguration des *StateMirrors* dar und wird in Unterkapitel 5.3.2 näher betrachtet. Das Fenster oben rechts nimmt die einzelnen (in diesem Fall - zwei) Visualisierungsseiten auf und wird in Unterkapitel 5.3.3 beschrieben. Das Fenster unten links gehört zum EIB-*Services*-Gateway, zeigt aber nicht seine Konfiguration, sondern die Konfiguration des ans Gateway angeschlossenen Systems. Das Fenster wird detailliert im nächsten Unterkapitel untersucht.

Zum Schluss muss noch angeführt werden, dass obwohl der in diesem Kapitel beschriebene Realisierungsentwurf der Konfigurationssoftware „*HomeConfigurator*“ wegen der begrenzten Zeit bei seiner Verwirklichung nur eine eingeschränkte Funktionalität besitzt, basiert er auch in diesem Entwicklungszustand auf den benutzerfreundlichen und intuitiv verständlichen Elementen der Benutzeroberfläche. Dank dieser Tatsache und seines modularen Aufbaus kann der *HomeConfigurator* als ein Gerüst zur Entwicklung eines vollfunktionalen und leistungsfähigen Konfigurationsinstruments für das Managementsystem in der Heimautomatisierung dienen. In der aktuellen Version des *HomeConfigurators* wurde die Verwaltung für die früher beschriebenen Bibliothekmodule, aber auch für die Module der einzelnen Untersysteme (z.B. EIB) und für die Module der Visualisierungssoftware (Beispiel – die Lampen im Konfigurationsfenster der Visualisierungssoftware in Bild 5.15) implementiert.

5.3.1. Konfiguration des EIB-*Services*-Gateways

Dank des Einsatzes der „Plug&Play“ Prinzipien hat die Konfiguration des EIB-*Services*-Gateways einen ziemlich einfachen Charakter und besteht nur in der Auswahl des bestimmten Gateways aus einer dem Anwender angezeigten Liste der gefundenen Gateways. Sobald der Anwender per „Drag&Drop“ Methode ein neues *Services*-Gateway des bestimmten Typs aus der Bibliothek der Systemkomponenten ins Projektfenster hinzugefügt hat, startet der

HomeConfigurator ein Entdeckungsverfahren der im Netz verfügbaren Services-Gateways und bietet dem Anwender anschließend einen Auswahldialog (Bild 5.17).

Bild 5.17. Dialog zur Auswahl des *Services*-Gateways

Um dem Anwender die Auswahl zu erleichtern, entfernt die Konfigurationssoftware aus der Liste der gefundenen Gateways die, die nicht den gewünschten Typ besitzen, und die, die bereits im Projekt verwendet werden. Zusätzlich zur Auswahlliste zeigt der Auswahldialog noch einige Hauptmerkmale des aktuellen (in der Liste ausgewählten) Gateways an, was dem Anwender erlaubt, gleiche Gateways voneinander zu unterscheiden.

5.3.2. Konfiguration des *StateMirrors*

Die Konfiguration des *StateMirros* besteht in der Vorbereitung einer Konfigurationsdatei (siehe Bild 5.10, Seite 77) für ihn. Die dafür benötigte Information befindet sich teilweise im Hauptprojektfenster (für die Sektion „gatewayList“) und in den Konfigurationsfenstern der einzelnen Untersysteme (Information darüber, in welchem Untersystem welche Variablen zu finden sind). Ihr Hauptteil, und zwar die Liste der verwalteten Variablen (für die Sektion „variableList“), befindet sich im Konfigurationsfenster des *StateMirrors* (Bild 5.18).

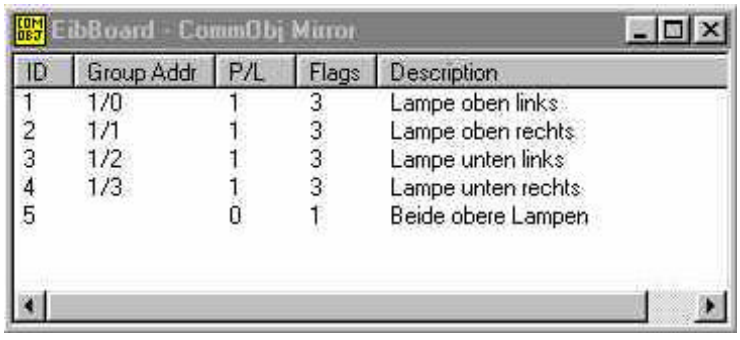


Bild 5.18. Konfigurationsfenster des *StateMirrors*

Dieses Fenster ist wie eine Tabelle dargestellt und enthält die zur Darstellung einer oder anderen Variablen wichtigen Eigenschaften (ID, die Adresse, der Variablentyp, die Beschreibungszeile).

Um den Konfigurationsprozess für den Anwender zu vereinfachen und den Konfigurationsaufwand zu senken, fügt die Software die Variablen automatisch zur Liste hinzu, nachdem entsprechende physikalische oder logische Verbindungen im Konfigurationsfenster der einzelnen Untersysteme hergestellt wurden. Dem Anwender steht auch eine Möglichkeit zur Verfügung, die Variablen manuell hinzuzufügen (Bild 5.19), sie zu ändern oder zu entfernen.

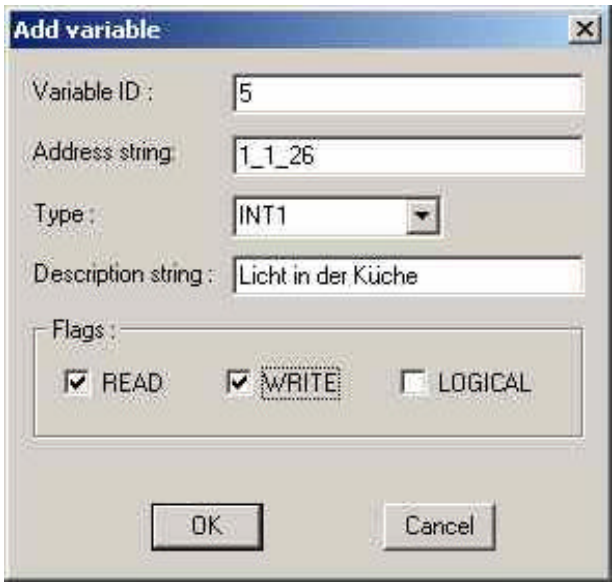


Bild 5.19. Dialog zur Einfügung der Variablen

5.3.3. Herstellung der Visualisierungsseiten

Die Visualisierungsseiten werden in einem entsprechenden Fenster (Bild 5.20) hergestellt.

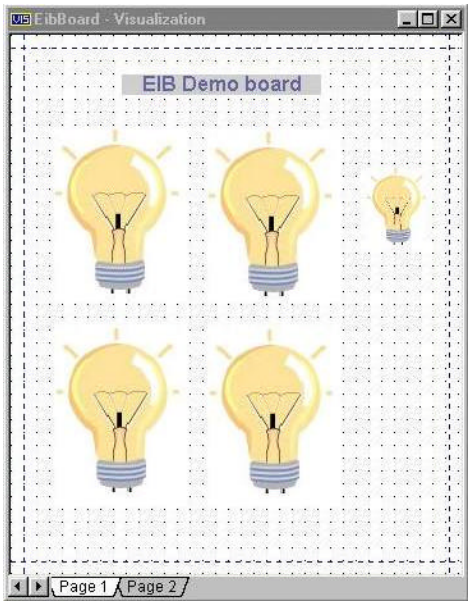


Bild 5.20. Konfigurationsfenster für die Visualisierungsseiten

Die Entwicklung der Visualisierungsseiten besteht aus der Sicht des Systems in der Vorbereitung der HTML-Seiten für die Visualisierungssoftware, die im nächsten Kapitel näher erläutert wird. Aus der Sicht des Anwenders besteht sie in der Positionierung der verschiedenen Visualisierungsmodule mit der Methode „Drag&Drop“ auf der entsprechenden Seite im Fenster des *HomeConfigurators*. Diese Module stellen entweder eine statische (Beispiel – Zeile „EIB Demo Board“ in Bild 5.20) oder eine dynamische (die Busvariablen) Information dar. Um die modulare Aufteilung der Visualisierungssoftware so bequem wie möglich zu gestalten, wurde eine Aufteilung auf die einzelnen Visualisierungsmodule nach dem Variablentyp vorgenommen, d.h. ein Visualisierungsmodul ist für die Darstellung der Busvariablen des einen bestimmten Typs (siehe Tabelle 5.3, Seite 78) zuständig.

Um eine flexiblere Verwendung der Visualisierungsmodule zu gewährleisten, besteht die Möglichkeit einige ihrer Parameter anzupassen. So zum Beispiel, hat das Visualisierungsmodul der Variablen des Typs „INT1“ folgende anpassbare Eigenschaften (Bild 5.21): „Name“ – Der intermodulare Variablenname; „Type“ – der Variabletyp (unveränderbar); „Oid“ – die SNMP Adresse der Variablen; „Position X,Y“ – die Position des Visualisierungsmoduls auf der Seite; „Bitmap_0“ und „Bitmap_1“ – die Bilder der Variblenzustände, die das Modul auf der Visualisierungsseite zeigt, um den Wert „0“ bzw. Wert „1“ visuell darzustellen.

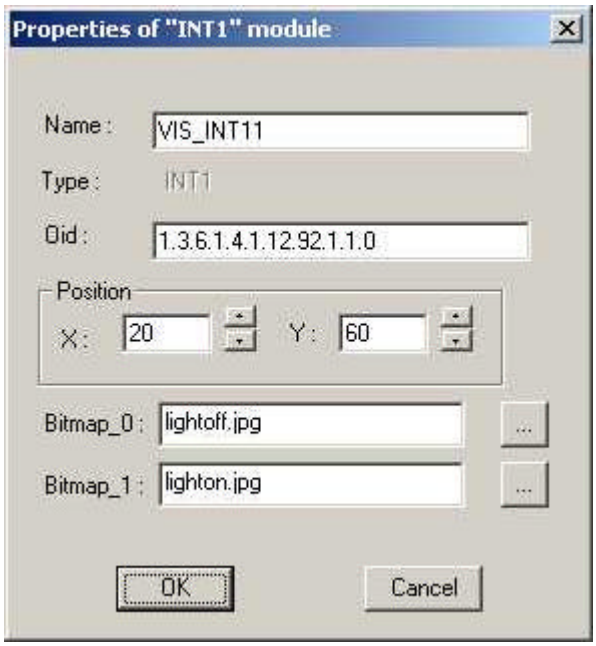


Bild 5.21. Eigenschaftendialog des Visualisierungsmoduls des Typs „INT1“

Es muss noch angemerkt werden, dass der *HomeConfigurator* bei der Vorbereitung der Visualisierungsseiten (Erzeugung der HTML Dateien) versucht, entsprechend den „WYSIWYG“ (What You See Is What You Get) Prinzipien das Seitengesamtlayout (Positionen der Module relativ zu einander und zur Seite) der vom Anwender entworfenen Visualisierungsseiten so nahe wie möglich beizubehalten.

5.3.4. Konfiguration der EIB-Komponenten

Die Konfiguration der EIB Komponenten stellt einen technisch komplizierten Prozess dar, der noch durch die Tatsache erschwert wird, dass die dafür benötigten Informationen praktisch unzugänglich sind. So sind die für die Programmierung der EIB Geräte notwendigen Daten in der von der EIBA (siehe Anhang A) verbreiteten Produktdatenbank der EIB Produkte zu finden. Der Zugriff auf sie ist allerdings wegen ihres geschlossenen proprietären Formats praktisch unmöglich. Deswegen ist auch der Import von vorbereiteten oder früher realisierten Projekten erschwert. Die Lösung dieses Problems hat man eher im politischem als im technischen Bereich zu suchen.

Obwohl die oben beschriebenen Schwierigkeiten für die Implementierung des Moduls für das EIB Untersystem, das für die Konfiguration der EIB Komponenten verantwortlich ist, ein gewisses Problem darstellen, bleiben sie für das Gesamtsystem unkritisch. Dies wird durch die Tatsache erreicht, dass das Systemkonzept erlaubt, die Konfiguration der einzelnen Untersysteme mit Hilfe der speziell dafür entwickelten Werkzeuge, so wie z.B. das Softwarepaket „ETS“ (siehe Anhang A) für den EIB, durchzuführen und sie trotzdem ins Gesamtsystem zu integrieren, allerdings mit einem größeren Aufwand für den Anwender.

Das Modul für das EIB Untersystem selber wurde im Rahmen des Realisierungsentwurfs nur in einer minimalen für die Machbarkeitsbestätigung benötigten Form implementiert. Die Bibliothek der EIB Komponente enthält zurzeit nur zwei Geräte – das sind, ein Schalter und ein Aktor, die in der im Kapitel 5.3 beschriebenen Miniinstallation des EIB Systems enthalten sind.

Die Konfiguration des EIB Untersystems findet auf mehreren Seiten im entsprechenden Fenster des *HomeConfigurators* (Bild 5.22) statt.

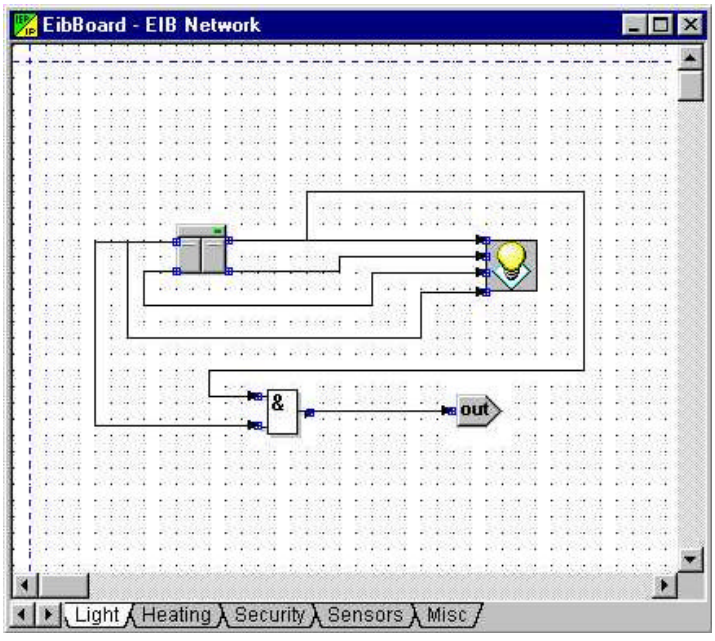


Bild 5.22. Das Fenster für die Konfiguration des EIB Untersystems

Die Aufteilung des Fensters auf die einzelnen Seiten kann nach beliebigen, dem Anwender passenden Kriterien geschehen. Im Bild 5.22 ist die Aufteilung zum Beispiel nach den Ge-

werken (Beleuchtung, Heizung, Sicherheitstechnik usw.) vorgenommen. Dem Anwender ist es überlassen, die vorhandenen Seiten zu löschen und eigene hinzuzufügen. Die Aufteilung des Konfigurationsfensters auf die Seiten spielt für die Software absolut keine Rolle – sie betrachtet alle auf verschiedenen Seiten befindlichen Geräte als ein System. Diese Aufteilung wurde nur dafür gemacht, um dem Anwender zu helfen, in großen und komplizierten Untersystemen die Übersicht zu behalten.

Der Konfigurationsprozess selber wurde so einfach wie möglich gehalten. Um ein neues Gerät zum Untersystem hinzuzufügen, überträgt der Anwender sein Icon aus dem Bibliothekfenster der EIB Komponenten (für die logischen Komponenten – aus dem Bibliothekfenster der logischen Komponenten) ins Konfigurationsfenster des EIB Untersystems. Danach hat der Anwender noch die Möglichkeit die Eigenschaften des eingefügten Geräts (Bild 5.23) zu verändern oder anzupassen und dadurch zu bestimmen – wie viele Eingänge und Ausgänge das Gerät besitzen wird.

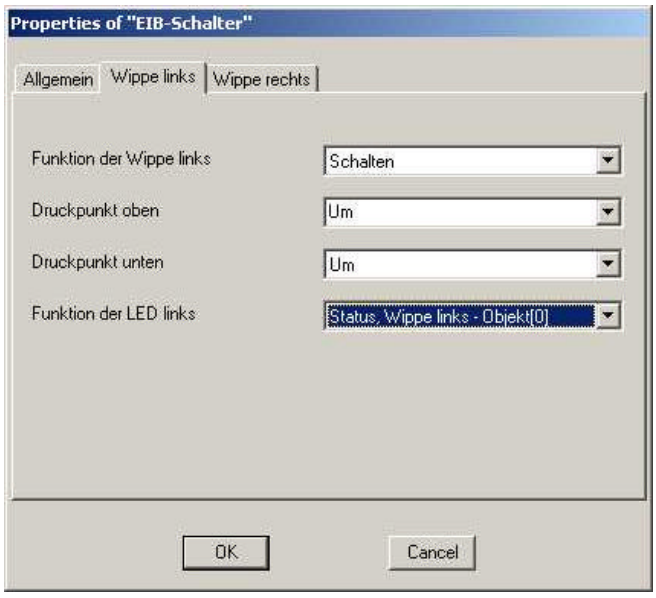


Bild 5.23. Eigenschaften des EIB Geräts

Weiter kann der Anwender die Verbindung der einzelnen Geräte untereinander durchführen. Dabei kontrolliert die Software die Richtigkeit (vom Ausgang des Geräts zum Eingang) der Verbindung. Es gibt auch die Möglichkeit die Eigenschaften der gerade vorgenommenen Verbindung (Bild 5.24) zu ändern.

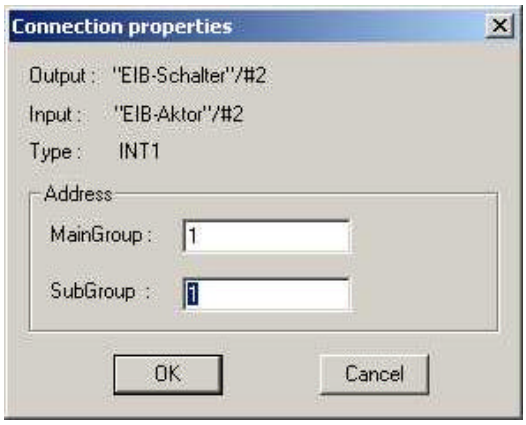


Bild 5.24. Verbindungseigenschaften

Für jede physikalische oder logische Verbindung erzeugt der *HomeConfigurator* eine neue Variable mit den entsprechenden Attributen im Konfigurationsfenster des *StateMirrors*.

Ein spezielles Verbindungselement mit der Aufschrift „Out“, das im Bild 5.22 zu sehen ist, dient dem Schaffen der Busvariablen bzw. Verbindungen, die mit Hilfe des Elements „In“ auf den anderen Seiten im gleichen Untersystem oder in einem anderen Untersystem eingesetzt werden können.

Wenn der Anwender die Konfiguration des Untersystems abgeschlossen hat, kann er alle Konfigurationsdaten oder nur ihren geänderten Teil mit einem „Klick“ ins Untersystem runterladen. Dabei bleibt der technisch komplizierte Ladevorgang für ihn völlig unsichtbar.

5.4. Visualisierungs- und Steuerungsprogramm

Wie schon erwähnt, wurde die Kombination HTML/HTTP als Protokoll für die Visualisierungs- und Steuerungssoftware ausgewählt, so dass zur Darstellung ein gewöhnlicher Webbrowser verwendet werden kann. Die Vorteile einer solchen Lösung sind offensichtlich: es gibt Implementierungen des Webbrowsers für alle mehr oder weniger verbreiteten Plattformen; der größte Teil von ihnen ist kostenlos (manche sogar im Quellkode); eine einheitliche Darstellung der Visualisierungs- und Steuerungsseiten u.a.

Dabei existieren manchmal gravierende Unterschiede zwischen den einzelnen Webbrowsern oder sogar zwischen den Implementierungen desselben Webbrowsers für verschiedene Plattformen. Hauptsächlich liegen die Unterschiede in der Unterstützung der unterschiedlichen Erweiterungstechnologien der Webseiten von Webbrowsern und sind in der Regel auf die verfügbaren Ressourcen der jeweiligen Plattform zurückzuführen.

In Anbetracht dieser werden in den nächsten Unterkapiteln die Lösungen für die folgenden Hauptgruppen von Webbrowsern vorgestellt: mit einer Java Virtual Machine (JVM) – größter Teil der verbreiteten Webbrowser; ohne eine Java Virtual Machine – in der Regel sind es die Webbrowser auf einer embedded Plattform, z.B. Pocket PC; WAP Browser – ein speziell für das Handy entwickelter Webbrowsertyp.

Bei der Implementierung wurde besonderer Wert auf die erste Gruppe von Webbrowsern gelegt – die Webbrowser mit JVM, weil sie die größte Verbreitung auf den typischen Heimplattformen – PCs und PC-basierten Steuerungsterminals – erreicht haben.

5.4.1. Web-Browser mit Java Virtual Machine (Java Version)

Die in diesem Kapitel vorgestellte Lösung basiert auf dem Einsatz der Java Applets als Visualisierungs- und Steuerungsmodule. Das Java Applet stellt eine besondere Art des Java Programms dar, das zur Ausführung in einem Webbrowser bestimmt ist.

In der Regel enthält eine Visualisierungs- und Steuerungsseite des Managementsystems ein unsichtbares Visualisierungsmodul „Manager“, das für die Kommunikation mit dem *State-Mirror* verantwortlich ist und mehrere Visualisierungsmodule für bestimmte Variablentypen, die eine oder andere Variable auf der Visualisierungs- und Steuerungsseite graphisch darstellen.

Zum Beispiel enthält die Visualisierungs- und Steuerungsseite des früher erwähnten Projekts „EibBoard“ ein unsichtbares „Manager“ Modul, ein Modul für die Darstellung der Zeichenkette und fünf Visualisierungsmodule des Variablentyps „INT1“. Ein Teil des HTML Quellkodes der Visualisierungs- und Steuerungsseite ist in Bild 5.25 zu sehen. Die Darstellung der Seite in einem Webbrowser ist aus Bild 5.26 zu entnehmen.

```
... ... ...
<APPLET codebase=.. code="Classes/VisualManager.class" NAME=MANAGER1 width=0
height=0>
<PARAM NAME=StateMirror VALUE="redbull" >
<PARAM NAME=Name VALUE="MANAGER1" >
</APPLET>

<APPLET codebase=.. code="Classes/Visual_Str.class" NAME=VIS_STR1 width=400
height=50>
<PARAM NAME=String VALUE="EIB Demo board" >
<PARAM NAME=Size VALUE="5" >
<PARAM NAME=Color VALUE="#10A060" >
<PARAM NAME=Name VALUE="VIS_STR1" >
</APPLET>

... ... ...
<APPLET codebase=.. code="Classes/VisualINT1.class" NAME=VIS_INT11 width=120
height=140>
<PARAM NAME=Manager VALUE="MANAGER1" >
<PARAM NAME=Oid VALUE="1.3.6.1.4.1.12.92.1.1.0" >
<PARAM NAME=Bitmap_1 VALUE="Pics/lighton.jpg" >
<PARAM NAME=Bitmap_0 VALUE="Pics/lightoff.jpg" >
<PARAM NAME=Name VALUE="VIS_INT11" >
</APPLET>
... ... ...
```

Bild 5.25. Teil des HTML Quellkodes der Visualisierungs- und Steuerungsseite des Projekts „EibBoard“

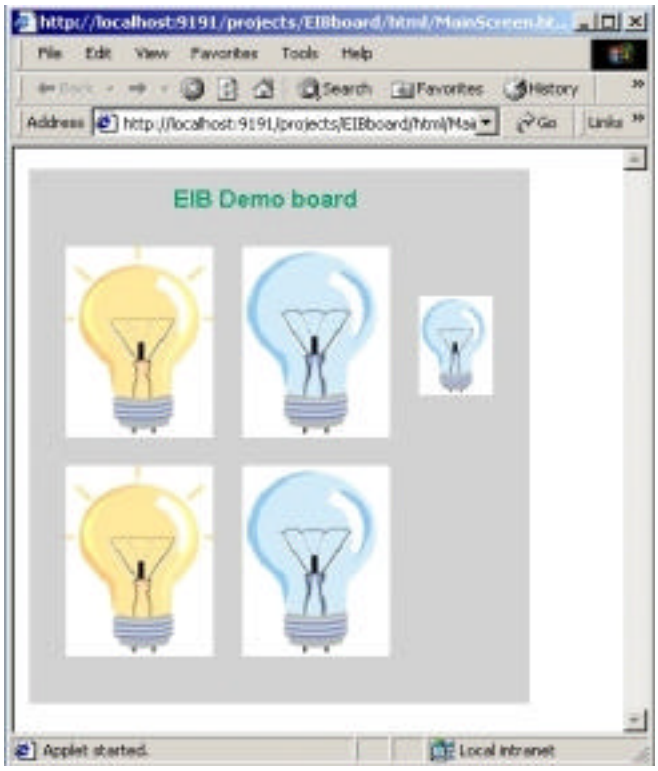


Bild 5.26. Darstellung der Visualisierungsseite in einem Webbrowser

Das erste Visualisierungsmodul, das auf der Seite beschrieben wird, muss ein „Manager“ Modul sein. Dieses Modul wurde in der Java Klasse `VisualManager` implementiert (Bild 5.27).

```
public class VisualManager extends java.applet.Applet implements iSnmpTrap
{
    ... ... ...
    public void init ();
    public void stop();

    public void SnmpTrapEvent(SnmpVarBind var);

    public int RegisterClient(String NameOfClient, String OidOfClient);
    public void ChangeValueFromClient(int ClientID, int newValue);

    ... ... ...

    private Vector VectorOfNames = new Vector();
    private Vector VectorOfOids = new Vector();
    private Vector VectorOfValues = new Vector();
}
```

Bild 5.27. Definition der Klasse VisualManager

Bei seiner Initialisierung (Webbrowser ruft die Funktion `Init` der Klasse auf) versucht das „Manager“ Applet einen *StateMirror* im gleichnamigen Parameter (siehe Bild 5.25) per übergebener Adresse oder über Netzwerknamen im Netz zu erreichen. Falls erfolgreich, abon-

niert das „Manager“ Applet die Ereignisnachrichten (werden im SNMP „Traps“ genannt) für die Änderung der Variablenwerte und geht in den „Warte“ Zustand über.

Danach können die Clientapplets sich durch den Aufruf der Funktion `RegisterClient` des „Manager“ Applets bei ihm registrieren lassen. Dabei melden sie ihm ihren eindeutigen intermodularen Namen und die SNMP Adresse der Variablen, die sie zu verwalten haben. Diese ganze Information und dazu noch der letzte bekannte Wert der Variablen speichert die Klasse `VisualManager` in drei Vektoren: `VectorOfNames`, `VectorOfOids` und `VectorOfValues` entsprechend. Als Ergebnis der Registrierung erhält jeder Clientapplet seine persönliche Identifikationsnummer, mit der er sich gegenüber dem „Manager“ später ausweisen kann.

Falls die Registrierung erfolgreich war, kann das Clientapplet jederzeit dem „Manager“ Applet durch den Aufruf seiner Funktion `ChangeValueFromClient` die Änderung des Variablenwerts durch den Anwender mitteilen. Im Gegensatz dazu, wenn eine asynchrone Ereignisnachricht der Änderung des Variablenwerts vom *StateMirror* beim „Manager“ eintrifft, leitet er sie an den entsprechenden Clientapplet weiter. Diese Ereignisnachricht erfolgt über die Funktion `SnmpTrapEvent` der Programmschnittstelle `iSnmpTrap`, die in der Klasse `VisualManager` implementiert wurde, des Java SNMP Stacks.

Der Aufbau des Clientapplets wird am Beispiel des Visualisierungsmoduls für den Variablentyp „INT1“ gezeigt. Die Implementierung eines solchen Visualisierungsmoduls befindet sich in der Klasse `VisualINT1` (Bild 5.28).

```
public class VisualINT1 extends java.applet.Applet
{
    ... ... ...
    public void init ();
    public void paint (Graphics g)
    public void ValueChangedEvent(int newValue)
    ... ... ...
}
```

Bild 5.28. Definition der Klasse `VisualINT1`

Bei seiner Initialisierung liest das Visualisierungsmodul zuerst die ihm übergebenen Parameter ein und versucht anschließend mit dem „Manager“ Modul in Verbindung zu treten und sich bei ihm zu registrieren. Danach hat das Clientapplet, wie schon vorher erwähnt wurde, die Möglichkeit, dem „Manager“ Applet die Änderung des Variablenwerts zu melden oder auch selber über die eigene Funktion `ValueChangedEvent`, die vom „Manager“ aufgerufen wird, diese zu erfahren.

Zum Schluss muss noch angemerkt werden, dass der Aufbau der Visualisierungsmodule der anderen Variablentypen (zurzeit sind die Module für nahezu alle Variablentypen verfügbar) sehr ähnlich sind und sich nur in der Implementierung der Funktion `Paint` der entsprechenden Applets unterscheiden, die für die graphische Darstellung (Zeichnung) des Variablenwerts auf dem Bildschirm verantwortlich ist.

5.4.2. Web-Browser ohne Java Virtual Machine (HTML Version)

Für den Webbrowser ohne JVM ist es auch möglich, obwohl mit einigen Beschränkungen, die Visualisierungs- und Steuerungsseiten herzustellen. Dies wurde mit Hilfe der „Common Gateway Interface (CGI)“ [57,58] erreicht. Die Technologie CGI bietet eine standardisierte Schnittstelle zur Integration der Ausgabe eines beliebigen Programms in einer oder anderen HTML Seite. Dabei können diese Programme in unterschiedlichen Programmiersprachen realisiert werden. Im Unterschied zu den Java Applets, die auf dem Clientrechner ausgeführt werden, laufen die CGI Programme auf dem Server.

So könnte, zum Beispiel, der Aufruf eines CGI Programms für die Darstellung des Variablentyps „INT1“ im HTML Kode folgendermaßen aussehen:

```
http://MyServer/cgi-bin/EIB_Light.cgi?“.1.3.6.1.4.1.12.92.1.1.0“
```

Mit dieser Zeile ruft der Server das CGI Programm `EIB_Light.cgi` auf, das den Wert der als Parameter übergebenen SNMP Variable „.1.3.6.1.4.1.12. 92.1.1.0“ beim *StateMirror* anfragt und dann abhängig vom erhaltenen Varibalenwert die Zeile „`<img src="LightOn.jpg" alt="Light is ON">`“, falls der Variablenwert „1“ ist, oder andernfalls die Zeile „`<img src="LightOff.jpg" alt="Light is OFF">`“ in seiner Ausgabe (der HTML Strom für den Client) schreibt. Das führt im ersten Fall zur Darstellung einer eingeschalteten Lampe im Fenster des Webbrowsers des Clients, im anderen Fall zur Darstellung einer ausgeschalteten Lampe.

Der größte Nachteil der Lösung mit den CGI Programmen ist ihre synchrone Ausführung, d.h. die CGI Programme werden nur nach einer Anforderung durch den Anwender (Abruf der HTML Seite) oder einer Aktion (Klick auf dem Seitenelement) ausgeführt. Und was ist, wenn die Variable ihren Wert geändert hat, nachdem die Seite aufgerufen wurde? Dieses Problem kann durch die Integration des Webbrowserbefehls zum automatischen Nachladen der Seite nach Ablauf eines bestimmten Zeitintervalls im HTML Kode gelöst werden. Obwohl dadurch die Visualisierungs- und Steuerungssoftware auf die asynchrone Änderungen reagieren kann, ist es keine optimale Lösung, weil ein zu kurzes Nachladeintervall zu einer viel zu häufigen Neuzeichnung der Seite und ein zu langes Intervall zu einer Verzögerung der Reaktion der Software auf die Änderung des Variablenwerts führt.

5.4.3. Mobile Pocket Web-Browser (WAP Version)

Bild 5.29. Bedienoberfläche des WAP-Browsers

Die am meisten verbreitete Mobileplattform ist ohne Zweifel das mobile Telefon. Statistisch gesehen, hat jeder zweite Deutsche ein Handy. In letzter Zeit werden mehr und mehr von ihnen mit einer speziell für die Plattform vorbereiteten Version des Webbrowsers (sog. WAP-Browser) ausgestattet, bei der zum Datenaustausch ein Wireless Application Protocol (WAP) verwendet wird.

Deshalb ist es konsequent, die Integration dieses Browsertyps in die Visualisierungs- und Steuerungssoftware für den Zugriff auf das Managementsystem zuhause und zum Datenaustausch mit ihm vorzunehmen. Obwohl diese Lösung die Herstellung von zusätzlichen WAP Visualisierungs- und Steuerungsseiten erfordert, bietet sie dem Hausbewohner ein erhöhtes Komfort- und Sicherheitsniveau. Er kann jederzeit und von jedem Ort sein Haus „anrufen“ und erfahren, ob alle elektrischen Geräte ausgeschaltet sind oder alle Fenster geschlossen sind, oder dem Haus auch den Befehl erteilen, ein heißes Bad vorzubereiten.

Obwohl die Bedienoberfläche des WAP-Browsers (Bild 5.29) weniger attraktiv als die seines „größeren“ PC-Kollegen aussieht, füllt sie jedoch die auf sie übertragenen Aufgaben vollständig aus.

5.5. Diagnostikprogramme

In diesem Kapitel werden alle anderen Programme vorgestellt, die beim Realisierungsentwurf des Managementsystems für die Heimautomatisierung entstanden sind. Als erstes ist der *Services*-Monitor zu nennen – ein Programm zur Diagnostizierung und zum Austesten der Funktionsweise des *Services*-Gateways. Das zweite hier vorgestellte Programm ist keine selbständige Software, sondern ist als ein Modul für das Tabellenkalkulationsprogramm „Microsoft Excel“ realisiert, durch das es möglich wird, die vom EIB-*Services*-Gateway vorbereiteten Protokolldaten einzulesen. Somit kann der Anwender die leistungsfähigen graphischen Werkzeuge von Excel für die Erstellung von unterschiedlichen Graphiken und Diagrammen nutzen.

5.5.1. *Services*-Monitor

Der *Services*-Monitor (Bild 5.30) ist ein universelles Testwerkzeug, das es den Entwicklern möglich macht, die von ihnen entwickelten *Services*-Gateways auf richtige Funktionsweise zu testen und den Anwendern Fehlfunktionen im Betrieb des einen oder des anderen Untersystems ausfindig zu machen.

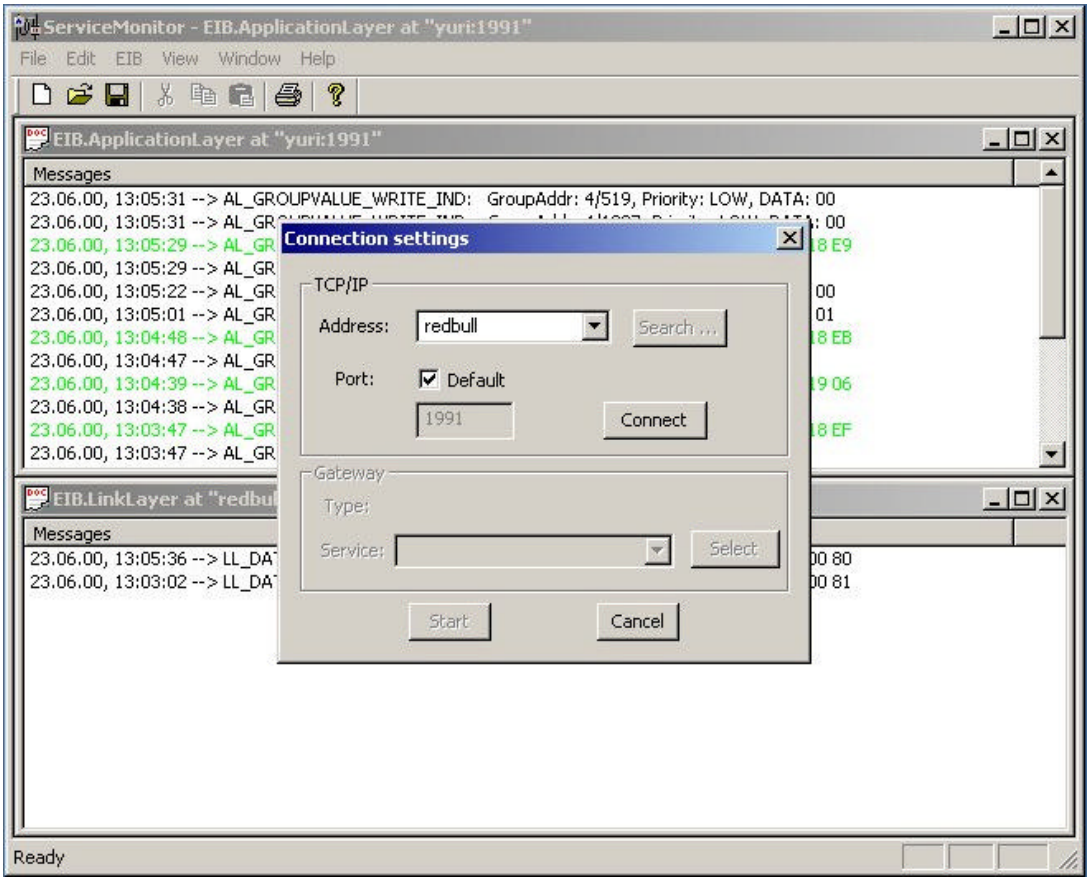


Bild 5.30. Testsoftware *Services*-Monitor

Nach dem Start des *Services*-Monitors hat der Anwender die Möglichkeit entweder eine früher gespeicherte Sitzung fortzusetzen oder eine neue zu starten. Im zweiten Fall wird dem Anwender ein Dialog mit den Verbindungsparametern (Bild 5.30, im Vordergrund) angezeigt. Dort kann er die Suche nach dem ihn interessierenden *Services*-Gateway initiieren oder aber seine Adresse direkt eingeben. Nach dem erfolgreichen Einloggen ins entsprechende Gateway wird die untere Hälfte des Dialog aktiviert, in der die Information über den Typ des *Services*-Gateways steht und in der der Anwender einen vom Gateway angebotenen Dienst auswählen kann. Nach der Auswahl eines bestimmten Dienstes lädt der *Services*-Monitor das Modul des entsprechenden Untersystems, das die Umwandlung der vom *Services*-Gateway kommenden Nachrichten zu einer vom Anwender leicht verständlichen Darstellung übernimmt. Diese wird dann im Verbindungsfenster angezeigt. Dieses Untersystemsmodul, das auch in der Konfigurationssoftware verwendet wird, bietet dem Anwender in der Regel über ein entsprechendes Menü die Möglichkeit, den einen oder anderen Dienst des *Services*-Gateways zu benutzen.

5.5.2. Protokollauswertungsmodul

Dieses Modul macht es möglich, die vom EIB-*Services*-Gateway gespeicherten Protokolldaten (den Busverkehr) in die Tabellenkalkulationssoftware Microsoft Excel zu importieren. Damit lassen sich unterschiedliche statistische Auswertungen der Daten und verschiedene graphische Darstellungen vornehmen.

So ist, zum Beispiel, in Bild 5.31 eine Graphik der zeitlichen Buslast zu sehen.

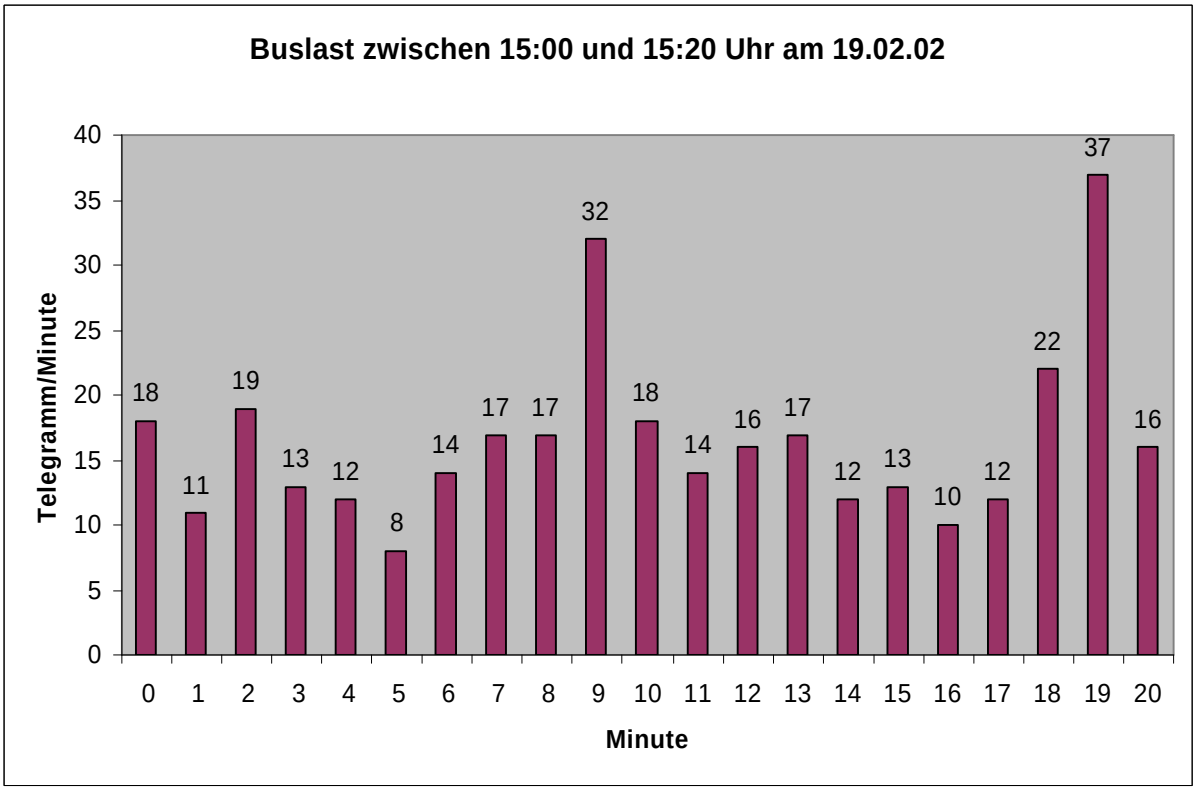


Bild 5.31. Graphik der zeitlichen Buslast

5.6. Zusammenfassendes Beispiel

Abschließend wird das Zusammenspiel der in den vorhergehenden Kapiteln beschriebenen Komponenten des realisierten Systems an einem praktischen Beispiel betrachtet, genauso wie die Reihenfolge ihrer Konfiguration und das Erstellen der Visualisierungsseiten. Als ein solches Beispiel dient uns die typische Basiskonfiguration eines Zimmers des privaten Heimes (in diesem Fall – das Kinderzimmer), das über Beleuchtung, Heizungskreis, Lüftung, Sonnenschutz sowie auch einen Temperatursensor verfügt.

Für das Funktionieren des Managementsystems für das private Heim ist die Installation des *Services*-Gateways in allen vorhandenen Untersystemen notwendig. Das Gateway wurde nach dem „Plug&Play“ Prinzip konzipiert und braucht daher keinerlei Konfiguration, sondern ist gleich nach dem Einschalten betriebsbereit.

Nach der Installation der Services-Gateways kann man zur Konfiguration der übrigen Systemkomponenten übergehen. Dafür verwendet man die in Kapitel 5.3 beschriebene Konfigurationssoftware des Managementsystems. Für unser Beispiel wird ein neues Projekt unter dem Namen „Kinderzimmer“ erstellt und ihm werden EIB *Services*-Gateway, *StateMirror* und ein Visualisierungsmodul für den Webbrowser hinzugefügt (Bild 5.32).

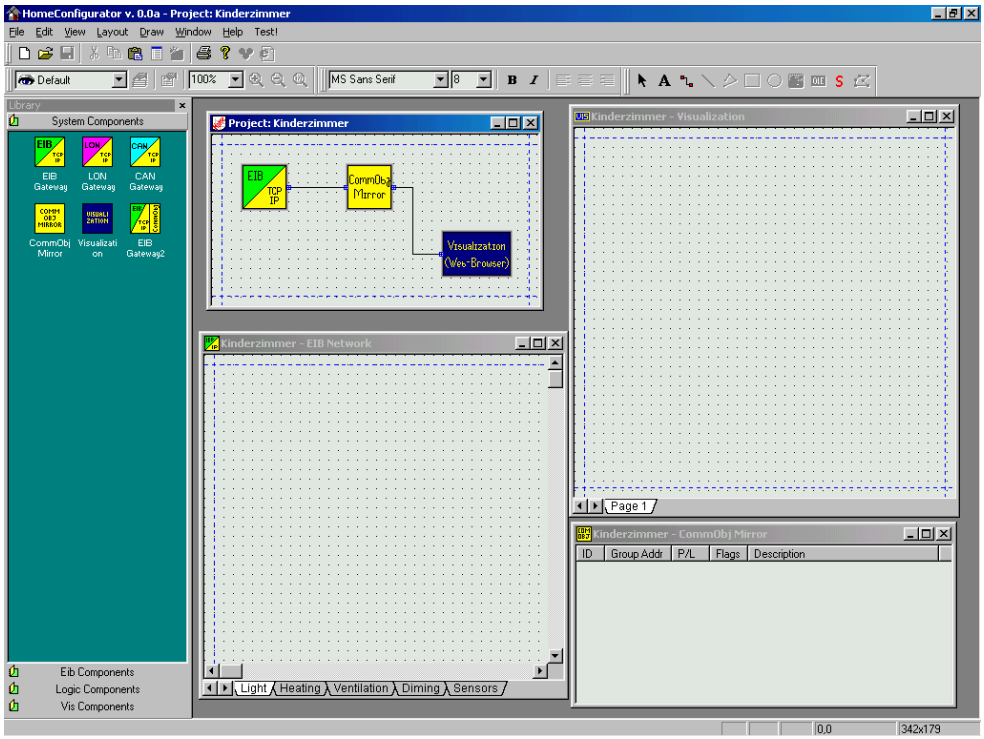


Bild 5.32. Konfigurationssoftware mit dem neuen Projekt „Kinderzimmer“

Der nächste Schritt ist die Konfiguration des EIB Untersystems, die wegen der unvollständigen Implementierung des EIB Untersystemmoduls und der entsprechenden Bibliothek der EIB Geräte mit Hilfe der speziellen EIB Konfigurationssoftware „ETS“ (s. Anhang A) durchgeführt werden muss. Nachdem die Konfiguration der EIB Geräte erfolgreich abgeschlossen ist, muss man alle physikalischen Variablen des EIB Untersystems manuell ins Konfigurationsfenster des *StateMirrors* eingeben. Dafür wird der Dialog „add variable“ (s. Kapitel 5.3.2) der Konfigurationssoftware des Managementsystems verwendet. Die notwendigen Daten für die eine oder andere Variable kann man aus dem entsprechenden Fenster (Bild 5.33) der Software „ETS“ entnehmen.

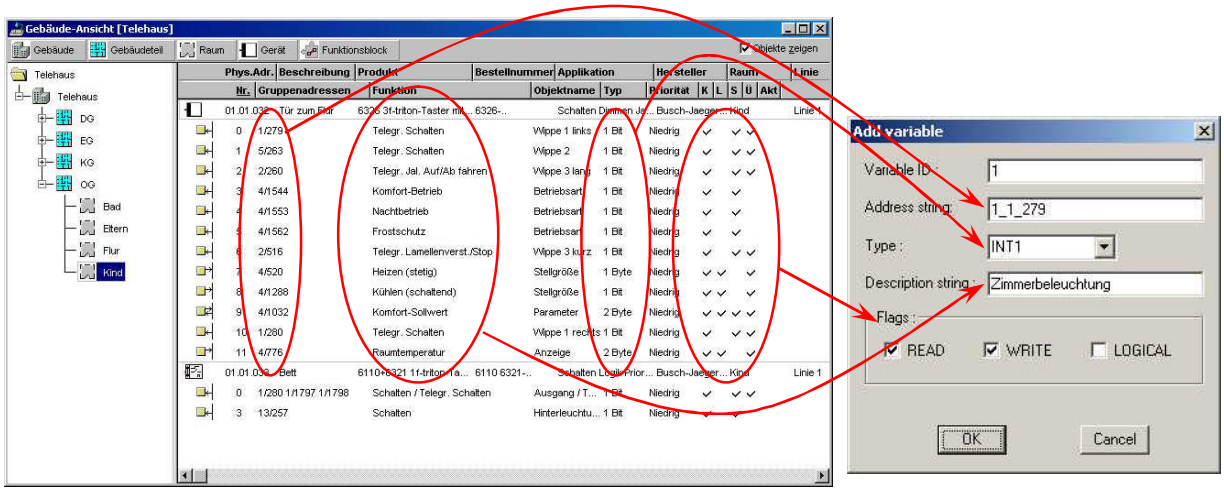


Bild 5.33. Die Entsprechung der Variableneigenschaften zwischen „ETS“ und der Konfigurationssoftware des Managementsystems

Nachdem alle physikalischen Variablen des EIB Untersystems dem Konfigurationsfenster des *StateMirrors* hinzugefügt wurden, sieht dieses ungefähr wie in Bild 5.34 dargestellt aus.

ID	Group Addr	P/L	Flags	Description
1	1/279	1	3	Zimmerbeleuchtung
2	1/280	1	3	Bettlicht
3	4/776	1	1	Temperatur IST-Wert
4	4/1032	1	3	Temperatur SOLL-Wert
5	4/520	1	3	Steuerung des Heizkreises
6	5/263	1	3	Lueftung
7	2/260	1	2	Jalousie Auf/Ab fahren
8	2/516	1	2	Jalousie Stop

Bild 5.34. Das Konfigurationsfenster des *StateMirrors*

Nun kann man das Managementsystem durch einige logische Variablen ergänzen, wie zum Beispiel der Variablen für das zeitweilige Belüften des Fensters. Angenommen, es wird das automatische und tägliche Lüften des Zimmers zwischen 3:00 und 4:00 Uhr nachts angestrebt, so muss man dafür ins Konfigurationsfenster des EIB Untersystems auf die Seite „Lüftung“ eine Zeitkomponente aus der Bibliothek der logischen Komponenten einfügen und ihre Eigenschaft „Trigger Time“ auf 3:00 Uhr (Bild 5.35.) stellen. Danach muss man auch die Komponente „Out“ ins Fenster einfügen und die mit ihr assoziierte Gruppenadresse gleich der Gruppenadresse der Belüftungssteuerung (in diesem Fall 5/263) setzen (Bild 5.36).

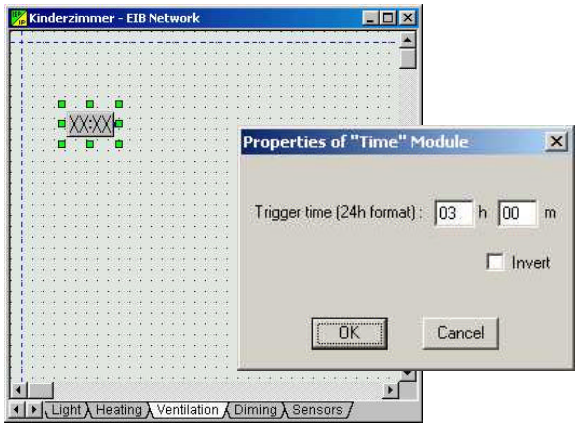


Bild 5.35. Zeitkomponente und ihre Eigenschaften

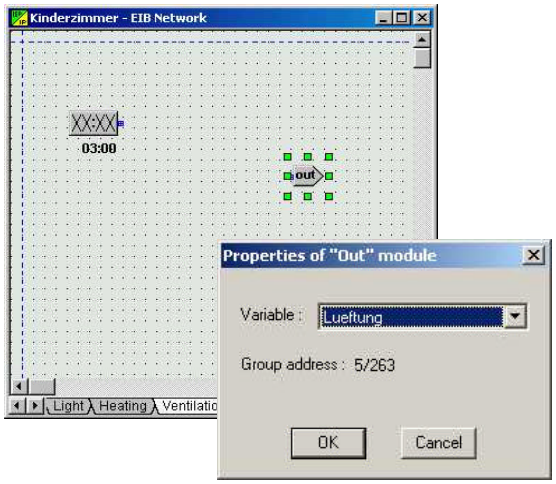


Bild 5.36. Komponente „Out“ und ihre Eigenschaften

Zum Ausschalten der Lüftung muss eine weitere Zeitkomponente eingefügt werden und ihre „Trigger Time“ muss auf 4:00 Uhr gestellt werden. Außerdem muss ihre Option „Invert“ aktiviert werden, damit die Komponente beim Erreichen des gesetzten Zeitpunktes den Wert „0“ auf dem Bus überträgt anstatt des Wertes „1“, was der Normalfall wäre. Zum Schluss müssen die beiden Zeitkomponenten mit der Komponente „Out“ verbunden werden. Als Ergebnis kann das Konfigurationsfenster des EIB Untersystems wie in Bild 5.37. aussehen.

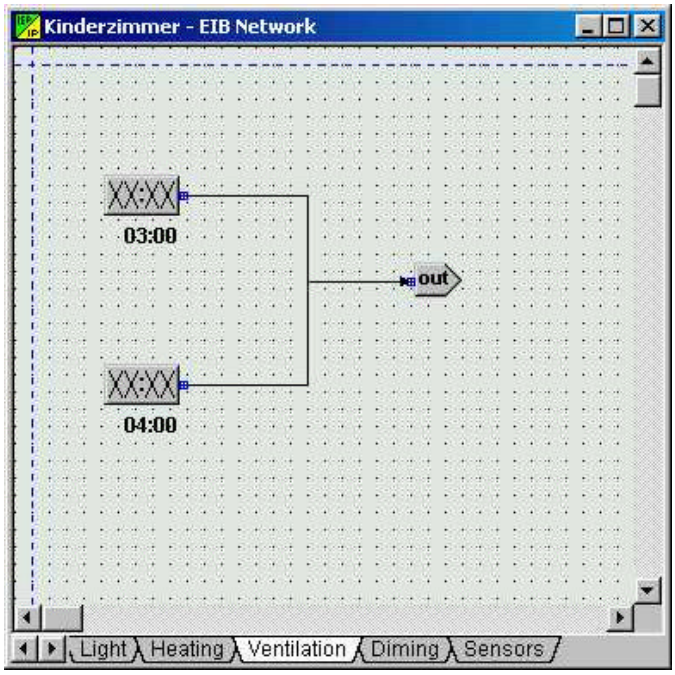


Bild 5.37. Konfigurationsfenster des EIB Untersystems

Der nächste Schritt der Systemkonfiguration ist die Herstellung der Visualisierungs- und Steuerungsseiten. In diesem Fall wird es nur eine Seite sein, auf der sich alle Visualisierungs- und Steuerungskomponenten für die im Zimmer vorhandenen Geräte befinden. Die Herstellung der Visualisierungsseiten besteht allgemein in der Positionierung der unterschiedlichen Visualisierungskomponenten und der Anpassung ihrer Eigenschaften. So müssen zum Bei-

spiel für die Belüftung (Bild 5.38) andere Bilder zur Darstellung des Variablenzustandes („Bitmap_0“ und „Bitmap_1“) ausgewählt werden als für die Beleuchtung.

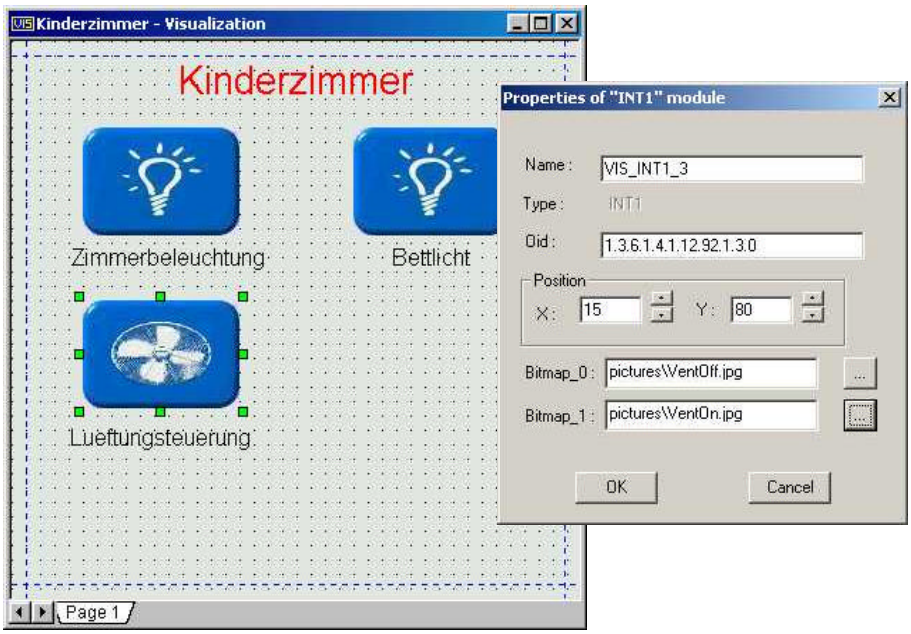


Bild 5.38. Visualisierungs- und Steuerungskomponente für die Belüftung und ihre Eigenschaften

Nachdem alle Visualisierungs- und Steuerungskomponenten im Konfigurationsfenster positioniert und angepasst wurden, könnte das Konfigurationsfenster wie in Bild 5.39. aussehen.

Bild 5.39. Die fertiggestellte Visualisierungs- und Steuerungsseite

Abschließend muss man das Projekt speichern (Menü „File/Save Project“), dabei wird die Konfigurationssoftware eine Konfigurationsdatei für den *StateMirror* (siehe Kapitel 5.2) und die HTML-Visualisierungsseiten erzeugen. Diese müssen dann in die entsprechenden Verzeichnisse der Softwarekomponenten kopiert werden. Nach dem Start des *StateMirrors* und des Webservers ist das Managementsystem für das private Heim zum Einsatz bereit.

6. Zusammenfassung und Ausblick

Mehr und mehr finden heutzutage unterschiedliche Systeme den Einzug in private Heime. Nun kann man dort überwiegend multimediale Systeme und Datenübertragungssysteme vorfinden, aber schon in nächster Zukunft kann ihr Spektrum deutlich erweitert werden. Die weite Verbreitung der Automatisierungssysteme hat in letzter Zeit auch die privaten Haushalte erreicht. Die fortschreitende Entwicklung der Elektronik und Informationstechnik verursacht einen Wandel der Automatisierungslösungen von den klassischen eigenständigen Systemen für Regelung und Steuerung zu komplexen verteilten Informationssystemen. Besonders die ständig wachsende Leistungsfähigkeit der Hardwarekomponenten begünstigt diesen Prozess und dient gleichzeitig als Plattform für komplexere Softwarelösungen. Diese Lösungen werden ohne Zweifel gefordert, wenn die vorhandenen verschiedenen Systeme miteinander zusammenarbeiten sollen und die Daten ausgetauscht werden.

In der vorliegenden Arbeit wurde ein Konzept für ein einheitliches Managementsystem für die Heimautomatisierung entwickelt, das es ermöglicht, die Zusammenschließung der unterschiedlichen im Haus koexistierenden Systeme zu einem logischen System durchzuführen und dadurch einen Mehrwert für den Bewohner im Vergleich mit der Nutzung der einzelnen Systeme zu erreichen. Besondere Aufmerksamkeit im Systemkonzept ist dem Aspekt gewidmet, dass die Benutzung des Systems und seiner Komponenten von einem technisch nicht versierten Anwender erfolgen soll. Deswegen fanden nur anwenderfreundliche und intuitiv verständliche Elemente für das MMI[4] Anwendung bei den Softwarekomponenten. So erhofft sich der Autor, dass das Einbeziehen des Anwenders in den Prozess der Systemkonfiguration (Systemänderung) ein größeres Maß an Vertrauen durch den Anwender bzw. Bewohner in das System hervorrufen wird.

Zur Demonstration der Realisierbarkeit wurde ein Entwurf mit bestimmten ausgewählten Protokollen für die Systemkomponenten präsentiert. Die Integration der einzelnen Untersysteme ins Managementsystem wurde am Beispiel des Automatisierungssystems EIB erläutert.

Um das Verhalten des Systems im alltäglichen Einsatz zu testen, wurde es in einem Forschungshaus (Bild 6.1) in Neubiberg bei München installiert. Dieses Haus wurde der Technischen Universität München im Rahmen des Forschungsprojekts „*tele*-Haus“ vom Projektpartner – Firma Bauland GmbH – zur Verfügung gestellt.

Bild 6.1. Das Forschungshaus „*tele*-Haus“ in Neubiberg bei München

[4] Eine Mensch-Maschine-Schnittstelle (eng. Man-Machine-Interface)

Die in Kapitel 5 beschriebenen Komponenten (EIB-*Services*-Gateway, *StateMirror* und *ManagementController*) wurden auf einem Computer im Haus installiert, der über das eigenentwickelte Zugriffsmodul „Ventura“ (Bild 6.2) auf den Automatisierungsbus EIB zugreift.

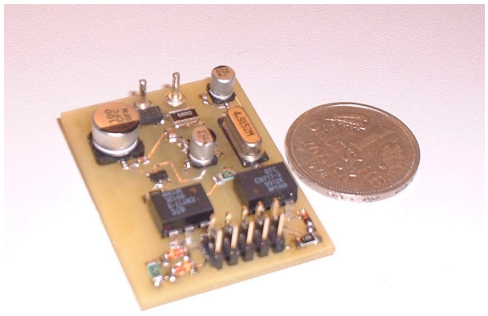


Bild 6.2. Modul „Ventura“ für den computerbasierten Zugriff auf das EIB System

Der Zugriff auf das System, die Steuerung des Systems und seine Beobachtung war dank der festen Anbindung des Hauses ans Internet auch vom Lehrstuhl aus möglich. Diese Tatsache hat sich als äußerst nützlich für die Suche nach Fehlern im System erwiesen. So zum Beispiel, ist es gelungen, ein fehlfunktionierendes Gerät im System ausfindig zu machen. Das Gerät hat auf alle ihn erreichenden Busnachrichten eine negative Bestätigung zurückgeschickt, die wiederum zur dreifachen Wiederholung aller Nachrichten auf dem Bus geführt hat. Die Suche nach dem Gerät wurde zusätzlich dadurch erschwert, dass die Gerätefehlfunktion nur in einem relativ kurzen Zeitintervall und sporadisch aufgetreten ist.

Um das fehlerhafte Gerät zu finden, wurde eine zusätzliche Funktion in der Testsoftware „*Services*-Monitor“ implementiert. Das Programm lief ununterbrochen auf einem Rechner am Lehrstuhl und benachrichtigte den Anwender, wenn wiederholte Nachrichten auf dem Bus auftraten. Auf diese Weise ist es gelungen, in kurzer Zeit das fehlfunktionierende Gerät zu lokalisieren und die Arbeit des Systems zu normalisieren.

In Zukunft, um die Tragfähigkeit der Systemstrukturen zu beweisen, wäre die Integration von anderen heterogenen Untersystemen ins Managementsystem von Interesse.

Als nächste Entwicklungsstufe des Systems kann man die Miniaturisierung der Hardwareplattform für das *Services*-Gateway und die dazu gehörige Anpassung der Systemsoftware vorschlagen. Als eine solche Plattform könnte z.B. PC-104 dienen, der durch seine Verwandtschaft mit der Computerplattform fast die gleichen Hardwareschnittstellen aufweist und den Einsatz von verbreiteten embedded Betriebssystemen, wie „Embedded Linux“, „Windows CE“ u.a., erlaubt. Letzteres Betriebssystem wird wegen seiner Zugehörigkeit zur Betriebssystemfamilie „Windows“ nur eine minimale Anpassung der Software erfordern. Die Portierung der anderen Softwarekomponenten des System (*StateMirror, ManagementController*) auf die neue Plattform ist auch denkbar.

Dem Autor scheint eine Integration von persönlichen Daten ins Managementsystem, wie hausbezogene und persönliche Termine des Bewohners, und die Finanzverwaltung (Strom-, Wasserverbrauch des Hauses, Kraftstoffverbrauch des Autos, etc.) möglich und nützlich zu sein.

Eine qualitativ neue Stufe der Entwicklung kann das Managementsystem durch die weitere Entwicklung des *Services*-Gateway und durch die Abbildung von embedded Geräten und „highlevel“ Services des an ihn angeschlossenen Untersystems (Bild 6.3) erreichen.

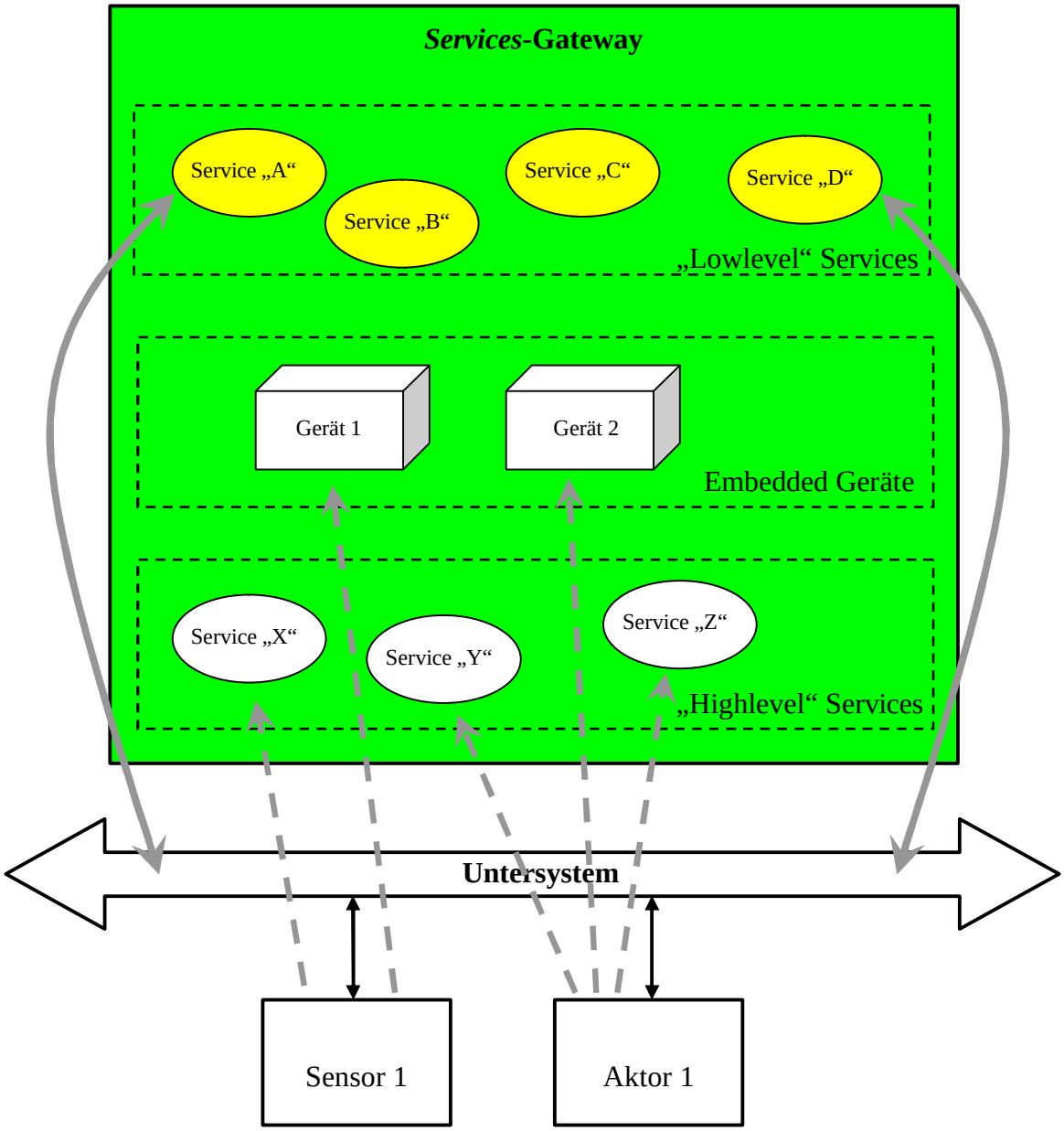


Bild 6.3. Integration von embedded Geräten und „highlevel“ Services ins *Services*-Gateway

Dabei muss eine vernünftige standardisierte Struktur zur Darstellung von embedded Geräten und „highlevel“ Services definiert werden, wie es bei der Definition der „lowlevel“ Services in dieser Arbeit der Fall war. Die größte Schwierigkeit, die man dabei zu überwinden hat, ist ein Kompromiss zwischen der Flexibilität (der Möglichkeit, sie auch für neue Geräte einzusetzen) und der Einfachheit.

7. Literaturverzeichnis

[1] J. Uetrecht: „Das vernetzte Haus“, Franzis, Poing, 2000

[2] S. Meyer: „Was will der Kunde – was nutzt der Kunde? Trends, Wünsche, Vorbehalte“, Berliner Institut für Sozialforschung, Kongressband e-home 2002, Berlin

[3] G. Schnell (Hrsg.): „Bussysteme in der Automatisierungs- und Prozesstechnik. Grundlagen und Systeme der industriellen Kommunikation“, Vieweg, Wiesbaden, 2000

[4] M. Abel, T. Lücke: „Einführung in die EIB-Gebäudesystem-technik“, Europa-Lehrmittel, Haan-Gruiten, 1998

[5] M. Rose, W. Kriesel, J. Rennefahrt: „EIB für die Gebäudesystemtechnik in Wohn- und Zweckbau“, Hüthig, Heidelberg, 2000

[6] D. Dietrich, W. Kastner, T. Sauter: „EIB Gebäudebussystem“, Hüthig, Heidelberg, 2000

[7] D. Dietrich, D. Loy, H.-J. Schweinzer: „LON-Technologie. Verteilte Systeme in der Anwendung“, Hüthig, Heidelberg, 1999

[8] D. Dietrich, K. Kabitzsch, G. Pratl: „LonWorks Gewerkeübergreifende Systeme“, Vde-Verlag, Berlin, 2002

[9] A. Kung, B. Jean-Bart, u.a.: „The EHS European Home Systems Network“, Trialog, Braunschweig, 1995

[10] W. Lawrenz: „CAN. Controller Area Network. Grundlagen und Praxis“, Hüthig, Heidelberg, 2000

[11] R. Hoffmann, K. Etschberger (Hrsg.) u.a.: „Controller- Area-Network. Grundlagen, Protokolle, Bausteine, Anwendungen“, Fachbuchverlag, Leipzig, 2002

[12] J. Rech: „Ethernet. Technologien und Protokolle für die Computervernetzung“, Dpunkt, Heidelberg, 2002

[13] F. Furrer: „Industrieautomation mit Ethernet- TCP/IP und Web-Technologie“, Hüthig, Heidelberg, 2002

[14] I. Stadler: „Dialogfähige Energiemanagementsysteme im Kontext von Energieverbrauch und Nutzerverhalten“, Dissertation, Universität Gesamthochschule Kassel, 2001

[15] T. Weinzierl: „Integriertes Managementkonzept für die Gebäudesystemtechnik“, Pflaum, München, 2001

[16] W. Kastner, W. Tumfart: „A New Framework for EIB/KNX Applications“, Konnex Scientific Conference 2002, München

[17] R. Busse: „Feldbussysteme im Vergleich“, Pflaum, München, 1996

[18] B. Reißenweber: „Feldbussysteme“, Oldenbourg, 2001

[19] Deutsche Industrienorm, DIN ISO 7498: „ISO-OSI-Schichtenmodell)“

[20] H. Kerner: „Rechnernetze nach OSI“, Addison-Wesley, 1992

[21] Amann GmbH: „EWMS I Gateway“, http://www.amann-net.de/produkte/ewms-gatewayI.htm

[22] Amann GmbH: „EWMS II Gateway“, http://www.amann-net.de/produkte/ewms-gatewayII.htm

[23] Siemens AG: „ISDN-Schnittstelle N147“, Technische Produkt-Information, 2001

[24] ASTON GmbH: „ASTON iPort MAX - ISDN“, http://www.aston-technologie.de/martin1.html

[25] Siemens AG: „Schnittstelle AP 146 für Ethernet-UDP/IP“, Technische Produkt-Information, 2002

[26] IPAS GmbH: „EIB/IP Combridge“, http://www.ipas-gmbh.de/d/63101-14-01.htm

[27] „Produktdatenbank der LON Nutzer Organisation e.V.“, http://www.lno-db.de

[28] FieldServer Technologies: „FieldServer LonWorks Bridge FS-B2011“, Product Manual, 2001

[29] SVEA Building Control Systems: „LON Internet Connector 1000“, Product Manual, 2000

[30] LAWICEL: „CAN232“, Product Manual, 2001

[31] IXXAT Automation GmbH: „Leistungfähiges Gatewaymodul für CAN und Ethernet/Modem (TCP/IP, PPP)“, http://www.ixxat.de/deutsch/produkte/canprod/gate-way/canatnet_gateway.shtml

[32] Deutsche Industrienorm, DIN ISO 13407: „Benutzerfreundliche Gestaltung von interaktiven Systemen“

[33] Deutsche Industrienorm, DIN ISO 9241-10: „Grundsätze der Dialoggestaltung“

[34] C. Stary: Interaktive Systeme, Software-Entwicklung und Soft-ware-Ergonomie, Vieweg, Braunschweig, 1996

[35] S. Schiffer: Visuelle Programmierung, Grundlagen und Einsatzmöglichkeiten, Addison Wesley Longman, Bonn, 1998

[36] J. Poswig: Visuelle Programmierung, Computerprogramme auf grafischem Weg erstellen, Hanser, München, 1996

[37] Deutsche Industrienorm, DIN ISO 9241-11: “ Richtlinien zur Gebrauchstauglichkeit“

[38] Y. Goland, T. Cai, u.a.: „Simple Service Discovery Protocol/1.0“, Internet-Draft, IETF, 1999

[39] M. Gudgin, M. Hadley, u.a.: „Simple Object Access Protocol (SOAP) 1.1“, W3C Working Draft <http://www.w3.org/TR /SOAP >, 2000

[40] S. Scribner: „SOAP – Developer’s Guide. Spezifikation, XML, BizTalk-Server“, Markt+Technik, 2000

[41] J. Cohen, u.a.: „General Event Notification Architecture Base“, Internet-Draft, IETF, 2000

[42] www.osgi.org

[43] OSGi: „OSGi Service Platform“, Release 2, 2000

[44] K. Chen, L. Gong: „Programming Open Service Gateways with Java Embedded Server ™ Technology“, Addison-Wesley Pub Co, 2001

[45] F. Iwanitz, J. Lange: „OPC. Grundlagen, Implementierung und Anwendung“, Hüthig, Heidelberg, 2002

[46] “Light OPC. The Free OPC Server Toolkit”, http://www.ipi.ac.ru/lab43/lopc-en.html

[47] http://www.ietf.org

[48] C. J. Date, H. Darwen: „A Guide to SQL Standard“, Addison-Wesley Pub Co, 1997

[49] S. W. Dietrich: „Understanding Relational Database Query Languages“, Prentice Hall, 2001

[50] L. A. Phillips: „Using HTML 4“, Que, 1998

[51] S. Münz, W. Nefzger: „HTML & Web-Publishing Handbuch. HTML, JavaScript, CSS, DHTML“, Franzis, Poing, 2002

[52] RFC2616: „Hypertext Transfer Protocol -- HTTP/1.1“, 1999

[53] C. Wong: „HTTP kurz und gut“, O’Reilly, 2000

[54] Microsoft MSDN Library: „Device Host API“, http://msdn.microsoft.com/library/default.asp?url=/library/en-

us/upnp/upnp/device_host_api.asp

[55] Microsoft MSDN Library: „Control Point API“, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/upnp/upnp/control_point_api.asp

[56] Microsoft MSDN Library: „SNMP Functions“, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/snmp/snmp/snmp_functions.asp

[57] D. Gourley, B. Totty: „HTTP: The Definitive Guide“, O’Reilly & Associates, 2002

[58] S. Guelich, S. Gundavaram, G. Birznieks: „CGI Programmierung mit Perl“, O’Reilly, 2001

[59] EIBA: „Handbook for Development. Pre-Release Issue EIB 3.0“, Brussel, 1998

Anhang A. Europäische Installationsbus (EIB)

Der Europäische Installationsbus (EIB) ist ein Automatisierungsbus für Häuser und Gebäude. Der EIB erlaubt dem Anwender die klassischen Gebäudesysteme, wie zum Beispiel Heizung, Beleuchtung, zu steuern und die "unsichtbaren" Prozesse im Gebäude (die Luftqualität, den Energieverbrauch usw.) zu beobachten.

Der EIB gehört zu den dezentralen Systemen und besteht in der Regel aus einzelnen EIB-Geräten, die untereinander mit Buslinien verbunden sind. Diese Buslinien können verschiedene Übertragungsmedien nutzen: Zweidraht, Powerline, Funk- oder Infrarotwellen. Das EIB Protokoll wird von allen Busteilnehmern unterstützt und verwaltet. Es wird im Mikrokontroller jeder einzelner Komponente implementiert und ausgeführt, d.h. es gibt keinen Bedarf zur Steuerung der Busfunktionalität in irgendeiner zentralen Komponente.

Alle Busteilnehmer kommunizieren direkt miteinander. So wenn zum Beispiel ein Sensor aktiviert wird, wird sein Applikationsprogramm der Systemsoftware einen Befehl geben, ein entsprechendes Telegramm auf dem Bus zu verschicken. Dieses Telegramm wird von einem oder mehreren Aktoren empfangen und als Reaktion darauf wird eine der vorprogrammierten Handlungen entsprechend der Konfiguration der Applikationsparameter ausgeübt.

Die Applikationsprogramme, ihre Parameter und die Beschreibung der Geräte von praktisch allen EIB Herstellern, d.h. denjenigen, die zu der Europäischen Installationsbus Association (EIBA) gehören, werden in einer zentralen Datenbank gesammelt. Die Projektierung der EIB Installation bedeutet dann, ein Produkt aus dieser Datenbank zu wählen, die benötigte und zu ihm passende Applikation zu finden und die Art der Zusammenarbeit zwischen den einzelnen Geräten zu bestimmen. Alle diese Schritte können mit Hilfe des Softwarepakets ETS – EIB Tool Software – ausgeführt werden.

Ein EIB Gerät besteht fast immer aus einer „Bus Access Unit"[5] (BAU) und einem Applikationsmodul (AM) (Bild A.1). Das können entweder zwei getrennte Module sein, die über das „Physical External Interface"[6] verbunden sind, oder ein Einheitsmodul.

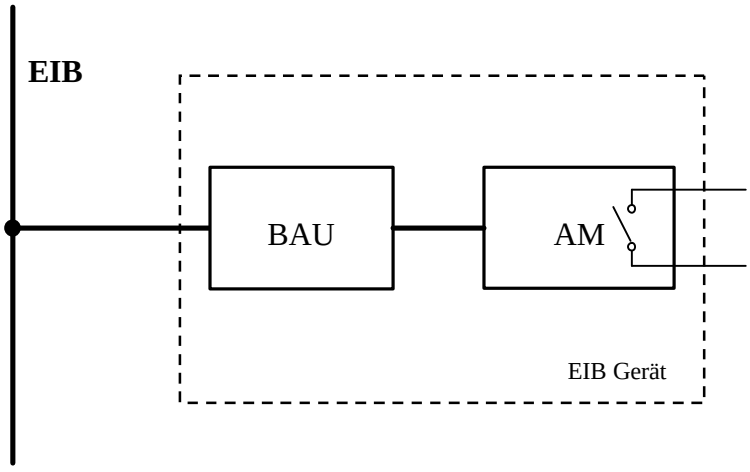


Bild A.1. Der Aufbau der meisten EIB Geräte

[5] Ein Buszugriffsmodul

[6] Eine physikalische externe Schnittstelle

A.1. Topologie und Bus Access Unit (BAU)

Wie bereits erwähnt wurde, hat der EIB Standard zurzeit die Signalübertragung über die folgenden Medien definiert: Zweidraht, Powerline, Funk- und Infrarotwellen. Unter der Topologie des Systems versteht man den physikalischen Weg, über den eine Übertragung des Kommunikationssignals möglich ist. Bei einigen Medien, z.B. Funk- und Infrarotwellen, ist die physikalische Verbreitung dieses Signals nicht an einem Draht gebunden, bei anderen Medien (Zweidraht, Powerline) pflanzt sich das Signal dagegen drahtgebunden fort. Dabei können einzelne Segmente der Topologie eine beliebige Form haben: Bus-, Stern-, Baum-, Ring- oder kombinierte Topologie (Bild A.2).

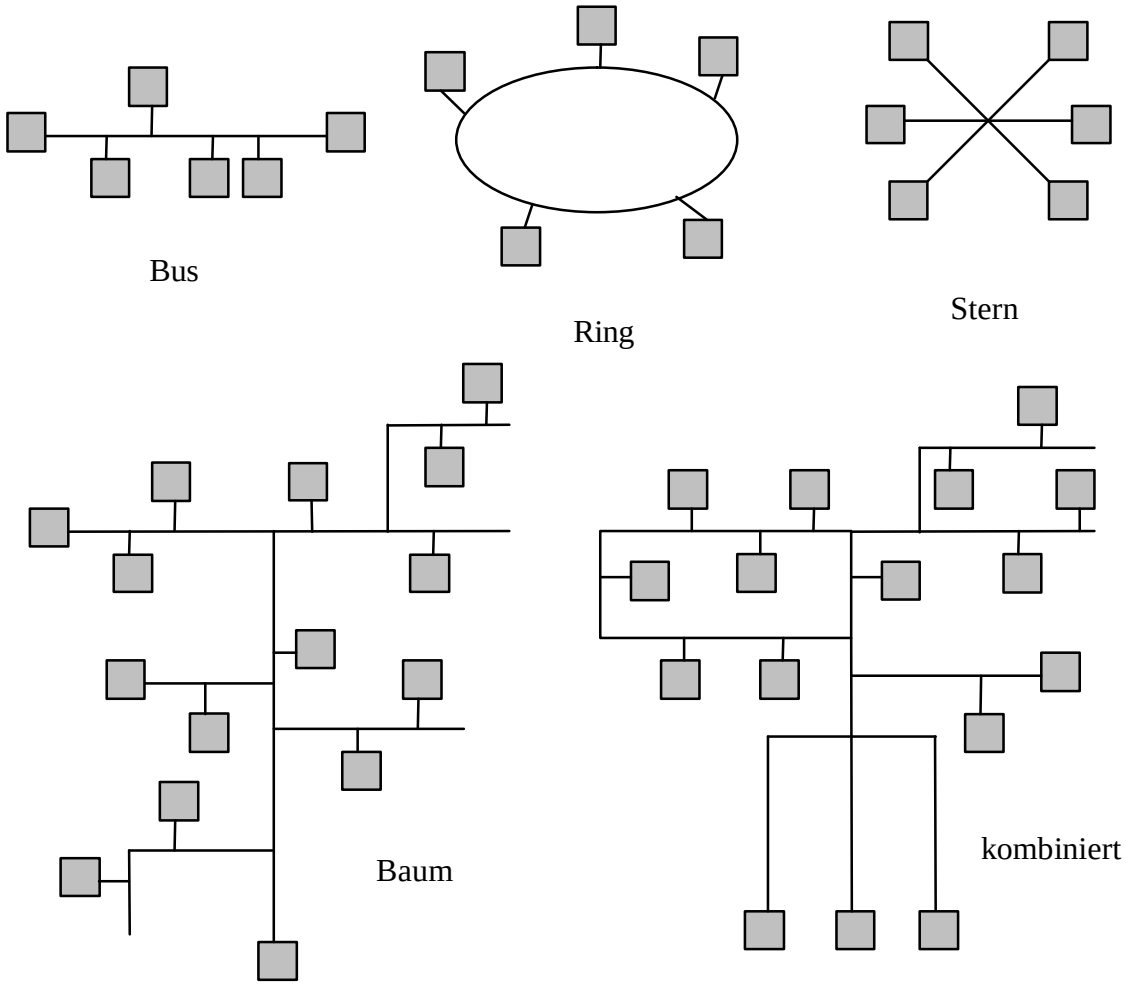


Bild A.2. Topologie der elektrischen Segmente

Das BAU ist zusammen mit dem Repeater und dem Linienkoppler eine Basiskomponente des EIBs. Sie besteht aus zwei Teilen: dem Zugriffsmodul auf das Übertragungsmedium und dem Kommunikationskontroller.

Das Zugriffmodul:

- dient als eine Datenschnittstelle, d.h. es moduliert die Daten bzw. sondert sie aus einem für das spezielle Medium spezifischen Signal aus
- kann optional die Versorgungsspannung für die anderen Hardwareteile (Kommunikationskontroller, Applikationsmodul o.a.) bereitstellen, falls die Spannung über das genutzte Übertragungsmedium zur Verfügung steht
- generiert spezielle Signale (Resetsignal, Savesignal u.a.) für den Kommunikationskontroller

Der Kommunikationskontroller ist in der Regel ein Mikrokontroller, der RAM, ROM und EEPROM enthält. Dabei:

- beinhaltet der ROM eine vorprogrammierte Systemsoftware, die nicht verändert werden kann
- speichert der EEPROM die Systemparameter, die den Systemzustand bestimmen, die applikationsspezifische Daten und normalerweise auch die Applikation selbst und ihre Parameter
- wird der RAM zum Teil von der Systemsoftware und der Applikation genutzt

Eine „Bus Coupler Unit“ (BCU) ist eine BAU, die zusätzlich die standardisierte Schnittstelle PEI anbietet. Diese ist ein 10 oder 12-poliger Stecker, an den die unterschiedlichen Applikationsmodule angeschlossen werden können und welcher zu den verschiedenen Betriebsarten konfiguriert werden kann (von mehreren Ein-/Ausgangskanälen, bis zu einer seriellen Schnittstelle).

Die BAU existiert in verschiedenen Variationen, diese unterscheiden sich voneinander in der Integrationsstufe. Das Spektrum reicht von der BCU im geschlossenen Gehäuse – vollkommen komplette Einheiten, in die nur ein Applikationsmodul eingesteckt werden muss – über „Bus Interface Module“ (BIM) – die eine vereinfachte Hardware haben und eine engere Zusammenarbeit mit dem Kommunikationskontroller ermöglichen – bis zu einzelnen Chipsätzen, die ganz einfach gehalten sind und eine begrenzte Funktionalität für den Buszugriff bieten.

Die BCU’s gibt es in verschiedenen Gehäusen, je nach den Geräteausführungen, in denen sie integriert sind (Bild A.2-A.4).

Bild A.2. BCU für die UP-Montage

Bild A.3. BCU für die AP-Montage

Bild A.4. BCU für die DIN Schiene

Bild A.5. BIM

Auch unterscheiden sich die BCU's in der Hardwareausstattung, die wichtigsten von ihnen sind in der Tabelle A.1 aufgeführt.

Tabelle A.1. Überblick über die existierende BCU-Hardware

	mask 001x	mask 002x	mask 070x	mask 101x
	BCU 1 BIM M111	BCU 2	BIM M112	PL-BA
Mikrokontroller	68HC05B6	68HC05BE12	68HC11E9	68HC05B16
RAM (bytes)	18	40 – 90	7K oder 15K	18
EEPROM (bytes)	230	850	8K oder 31K	230
Maximalle Anzahl von Gruppenadressen	64	64	254	64
Maximalle Anzahl von Kommunikationsobjekten	12	28 – 48	255	12
Zugriffschutz	Nein	Ja	Ja	Nein

A.2. Übertragungsprotokoll und Interworking

Der Informationsaustausch zwischen zwei EIB Geräten findet auf der Basis der Datenpakete statt. Dabei muss der Empfänger jedes erfolgreich empfangene Paket dem Sender bestätigen. Für jedes der unterstützten Übertragungsmedien sieht der Nachrichtenframe wie in Bild A.6 aus.

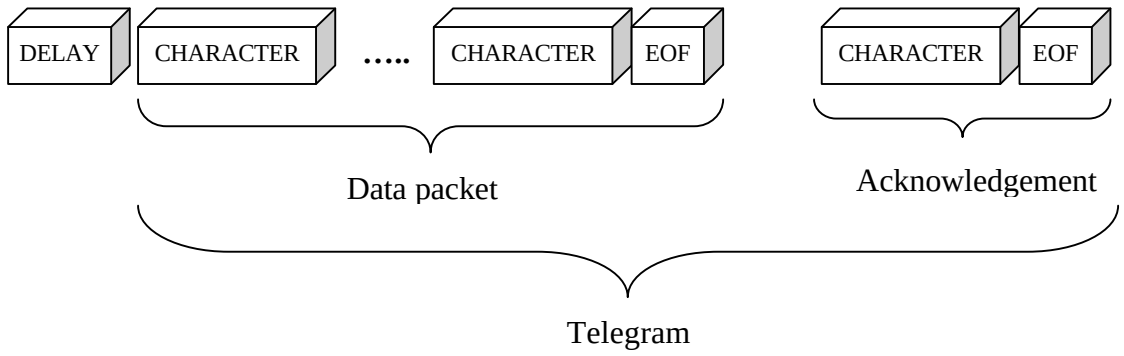


Bild A.6. Nachrichtenframe

In manchen Medien werden vor oder nach dem Frame noch einige zusätzliche spezifische Sequenzen übertragen, die entweder die Charakteristik des Übertragungsmediums oder die Mechanismen zur Fehlerkorrektur beinhalten.

Der Aufbau des Datenpakets selbst ist aus Bild A.7 zu entnehmen.

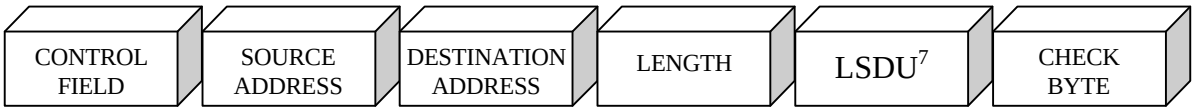


Bild A.7. Felder des Datenpakets[7]

Für den Fall der Übertragung von Notfallnachrichten oder anderen wichtigen Nachrichten erlaubt das EIB System, jedem zu übertragenden Datenpaket eine eigene Priorität zu geben. So ist die Priorität einer Alarm-Nachricht höher als die Priorität aller allgemeinen Pakete. Und die Datenpakete, die infolge einer Störung im System oder aus einem anderen Grund wiederholt werden müssen, besitzen eine Priorität, die zwischen den allgemeinen und den Alarm-Paketen liegt.

Die an den Bus angeschlossenen Geräte können auf zwei Arten adressiert werden – mittels der physikalischen oder der Gruppenadresse.

Die physikalische Adresse stellt eine einzigartige ID des Gerätes in einer bestimmten Installation dar. Die physikalische Adresse entspricht der physikalischen Lage des Gerätes in der Installationstopologie und ist eine Kombination aus den Nummern der Zone und der Linie, in welchen sich das Gerät befindet, und der Gerätenummer in der jeweiligen Linie (Bild A.8).

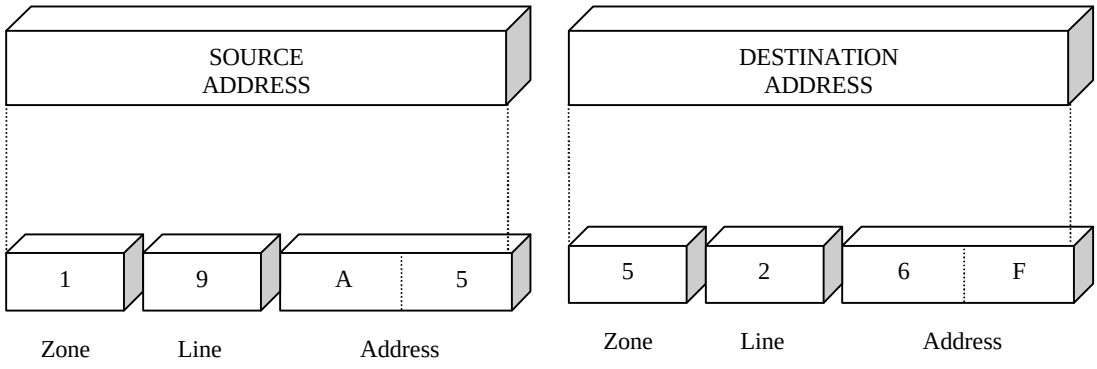


Bild A.8. Das Adressfeld bei physikalischer Adressierung

Die physikalische Adresse wird jedem Geräte bei seiner Inbetriebnahme zugeteilt. Danach kann das Gerät im System eindeutig adressiert werden, z.B. um ins Gerät eine neue Applikation herunterzuladen oder dem Gerät eine neue Gruppenadresse zuzuweisen.

Die EIB Busteilnehmer, die eine gemeinsame Funktion erledigen (z.B. der Schalter und die Lampe), werden in sog. „Gruppen“ zusammengeführt. Solche Geräte kommunizieren untereinander mittels der Gruppentelegramme, die durch die Gruppenadressen adressiert werden (Bild A.9).

[7] LSDU (Link Service Data Unit), d.h. die Daten, die übertragen werden müssen

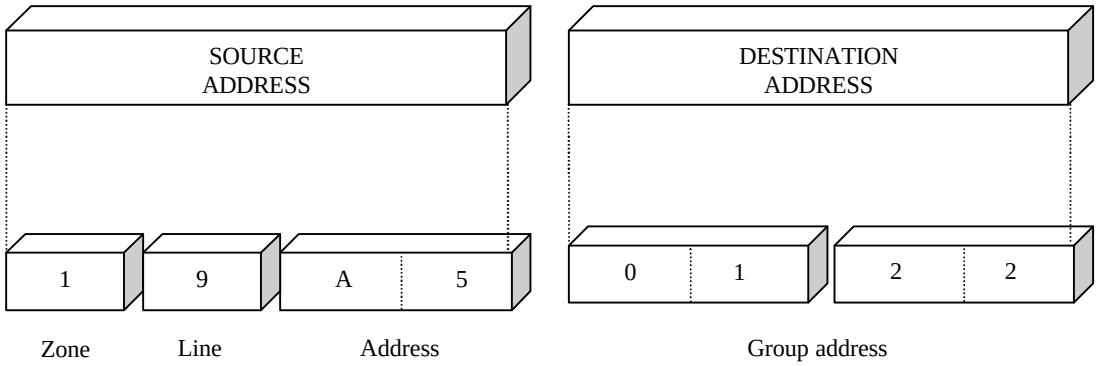


Bild A.9. Das Adressfeld bei der Gruppenadressierung

Im Gegensatz zur physikalischen Adresse enthält die Gruppenadresse keinerlei Information über die Lage des Gerätes im System. Somit können die Geräte, die zu einer Gruppe gehören, sich physikalisch in einem beliebigen Systemteil befinden. Der Vorteil der Gruppenadressierung der Geräte im System besteht darin, dass ein Gruppentelegramm, das von irgendeinem Mitglied einer Gruppe gesendet wurde, gleichzeitig von allen Mitgliedern dieser Gruppe empfangen und bearbeitet wird. So wird die Kommunikation zwischen den Busteilnehmern vereinfacht und dazu noch die Buslast gesenkt.

Das EIB Kommunikationsstack ist auf der Basis des ISO/OSI Referenzmodells [19,20] aufgebaut (Bild A.10).

Die erste physikalische Ebene des EIB Protokolls steuert die physikalischen Mechanismen der Datenübertragung. Sie ist für die physikalische Übertragung der einzelnen Datenbits auf dem Bus zuständig. Deshalb ist diese Ebene ausschließlich in Hardware implementiert. Bei der Benutzung des Zweidrahts als Übertragungsmedium wird der CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) Mechanismus für den Buszugriff eingesetzt. Dieser Mechanismus verhindert die Datenübertragung von irgendeinem Gerät auf dem Bus, wenn ein anderes Gerät das gemeinsame Medium bereits benutzt. Dies wird dadurch erreicht, dass bei gleichzeitiger Übertragung von zwei Geräten die Daten von dem einen mit den Daten von dem anderen überschrieben werden (im EIB System überschreibt ein „0“-Bit das „1“-Bit). Falls ein beliebiger Busteilnehmer des EIB Systems merkt, dass seine Daten überschrieben werden, hört er unverzüglich mit ihrer weiteren Übertragung auf und ermöglicht damit dem anderen Gerät, seine Datenübertragung erfolgreich zu beenden.

Die zweite Linkebene sammelt die einzelnen Datenbits in Blöcke, ergänzt sie durch die Start-Stop- und Paritätsbits, und versieht sie mit einer Checksumme, um ihre sichere Übertragung zu gewährleisten. Die Linkebene ist unter anderem auch für die Bestätigung der richtig empfangenen Daten zuständig.

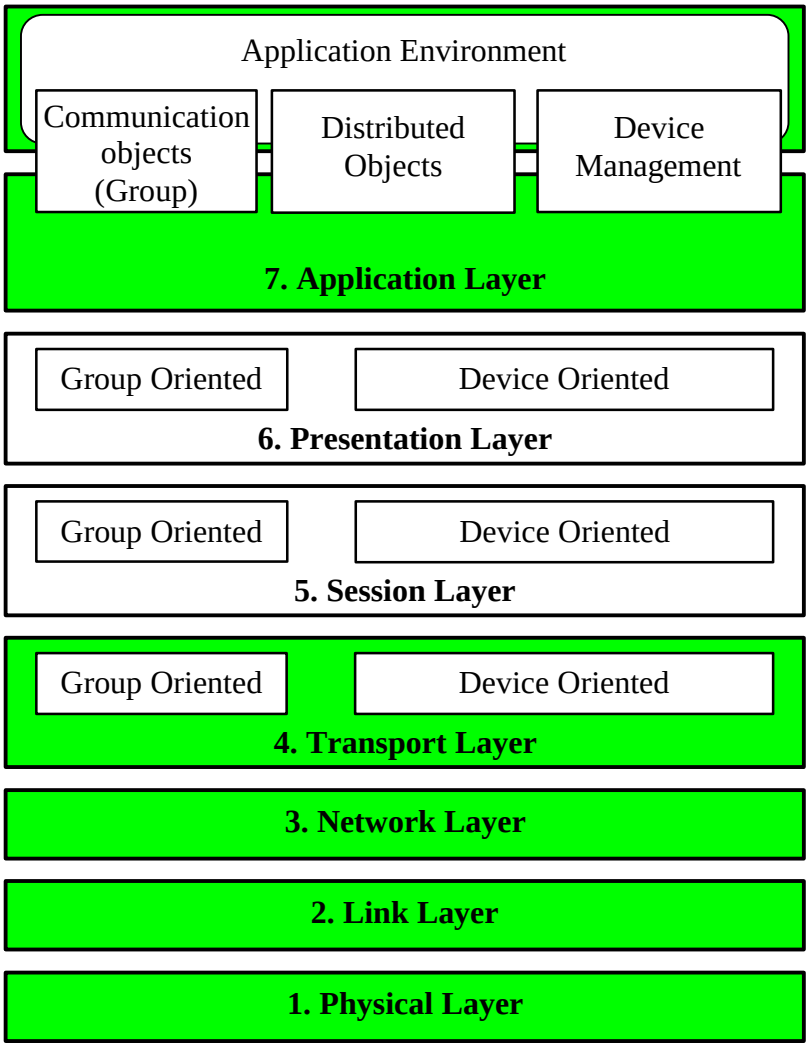


Bild A.10. EIB Kommunikation entsprechend dem ISO/OSI Referenzmodell

Die dritte Netzwerkebene gewährleistet eine sichere Datenübertragung aus einem Netzsegment (im EIB – Bereiche und Linien) des Systems in ein anderes, d.h. ihre Weiterleitung durch die Linien- und Bereichskoppler. Um die kreisförmige Zirkulation ein und desselben Telegramms im System zu verhindern, definiert die Netzwerkebene des EIB Protokolls einen Weiterleitungszähler, der sich jedes Mal um Eins verringert, wenn das Telegramm durch die Linien- oder Bereichskoppler passiert. Ein Telegramm mit dem Wert „0“ des Weiterleitungszählers wird von den Kopplern dann einfach ignoriert.

Die vierte Transportebene des EIB Protokolls ist für die „Punkt-zu-Punkt“ Datenübertragung im System verantwortlich. Dabei werden zwei Arten unterstützt: die verbindungslose und die verbindungsorientierte Datenübertragung. Die erste Art wird vorzugsweise für die Gruppenadressierung eines oder mehrerer Geräte eingesetzt. Bei dieser Kommunikationsart wird die Buslast bedeutend verringert, da das sendende Gerät das Telegramm nur ein Mal an alle Empfangsgeräte sendet, die dann ihren Empfang oder Nichtempfang auch gleichzeitig bestätigen. Dieser Vorteil wird durch die sinkende Übertragungszuverlässigkeit erreicht. So, wenn ein Empfangsgerät ausgeschaltet oder einfach physikalisch vom Netz getrennt ist, werden die positiven Bestätigungen von anderen Geräten aus derselben Empfangsgruppe dem Sender genügen, um diese als erfolgreich anzusehen. Die zweite Art der Kommunikation bietet eine

höhere Sicherheit und wird hauptsächlich für die Verwaltung (das Herunterladen von Tabellen oder Applikationen usw.) der einzelnen Geräte benutzt. Eine höhere Sicherheit wird dadurch erreicht, dass an der Kommunikation gleichzeitig nur zwei Geräte teilnehmen. Dabei werden auch noch alle Telegramme durchnummeriert und von der Empfangsseite zusätzlich bestätigt.

Der Hauptteil der Software der Applikationsebene dient zur Bereitstellung einer standardisierten Schnittstelle zwischen den Anwenderapplikationen und der Kommunikationssoftware des Systems. Der andere Teil der Software ist für die Verwaltung der Kommunikationsobjekte eines Gerätes zuständig.

Die EIB Komponenten werden von einer großen Anzahl von Herstellern produziert. Deshalb ist für ihre reibungslose Zusammenarbeit wichtig, dass sie eine gemeinsame „Sprache“ sprechen.

Gerade dafür werden die Hauptfunktionen der EIB Geräte definiert, die die wichtigsten Aufgaben einer EIB Installation in Gebäuden abdecken. Alle diese Funktionen sind im EIB Interworking Standard (EIS) aufgeführt. So, zum Beispiel, ist in Bild A.11 eine „Schalter“ - Funktion und eine Darstellung ihres Wertes auf dem Bus zu sehen.

Funktion	EIB_Switching
Wertgröße	1 Bit
Datengröße	1 Byte
Definition	Bits : 7 6 5 4 3 2 1 0 0 0 0 0 0 0 0 X
Belegung	X: 0 = “aus“, 1 = “ein“

Bild A.11. EIS Funktion „Schalter“ und die Darstellung ihres Wertes

Dadurch wird erreicht, dass, falls ein Sensor (in unserem Fall – ein Schalter) auf dem Bus den Wert „Ein“ an eine bestimmte Gerätegruppe schicken wird, alle Aktoren, die zur jeweiligen Gruppe gehören, dies als einen Befehl betrachten und die entsprechenden Kreise in ihren Applikationsmodulen einschalten bzw. kurzschließen. Dies könnte z.B. zum Einschalten der Beleuchtung oder etwas anderem führen.

Alle zum heutigen Zeitpunkt definierten EIB Funktionen und ihre Kurzbeschreibungen sind in Tabelle A.2 aufgeführt. Eine detaillierte Beschreibung und Definition der Übertragungsformate ihrer Werte sind aus [59] zu entnehmen.

Tabelle A.2. Existierende EIS Funktionen und ihre Kurzbeschreibung

Funktion	Beschreibung
EIS 1: „Switching“	Diese 1-bit Funktion dient zur Umschaltung des Aktors in einen von zwei möglichen Zuständen (Werte: 1/0)
EIS 2: „Dimming“	Ermöglicht die Steuerung eines Dimmers und enthält drei Unterfunktionen: „Position“ (einschalten, ausschalten und Status), „Steuerung“ (relatives Dimmen) und „Wert“ (absolutes Dimmen)
EIS 3: „Time“	Gewährleistet die Zustellung der aktuellen Systemzeit an alle Busteilnehmer
EIS 4: „Date“	Gewährleistet die Zustellung des aktuellen Datums an alle Busteilnehmer
EIS 5: „Value“	Dient hauptsächlich der Übertragung von Werten unterschiedlicher physikalischer Größen, dabei wird die Einheit der physikalischen Größe nicht mitübertragen
EIS 6: „Scaling“	Dient der Übertragung von relativen Werten mit einer maximalen Auflösung von 8 Bits
EIS 7: „Drive control“	Ermöglicht die Steuerung von Motoren und enthält zwei Unterfunktionen: „Bewegung“ (einschalten, ausschalten und Änderung der Fahrrichtung) und „Schritt“ (Schritt vorwärts, Schritt rückwärts und Stop)
EIS 8: „Priority“	Bietet die Möglichkeit, die Priorität einer beliebigen Operation zu steuern
EIS 9: „Float value“	Dient der Übertragung von Werten physikalischer Größen im IEEE Float Point Format
EIS 10: „16-bit Counter value“	Wird für die Übertragung von Werten von 16-bit-Zählern eingesetzt
EIS 11: „32-bit Counter value“	Wird für die Übertragung von Werten von 32-bit-Zählern eingesetzt

A.3. Software des Systems

Die ganze Software des Systems wird von der EIBA entwickelt und ausschließlich unter ihren Mitgliedern verbreitet. Diese Software kann in die folgenden Softwarepakete aufgeteilt werden:

- EIB Interoperability Test Tool (EITT)
- Bus monitor

– EIB Tool Software (ETS)

– EIB Integrated Development Environment (EIB IDE)

Das EITT (Bild A.12) ist ein Testtool, das zu Tests der neu entwickelten EIB Geräte und zur Vorbereitung auf ihre Zertifikation eingesetzt wird.

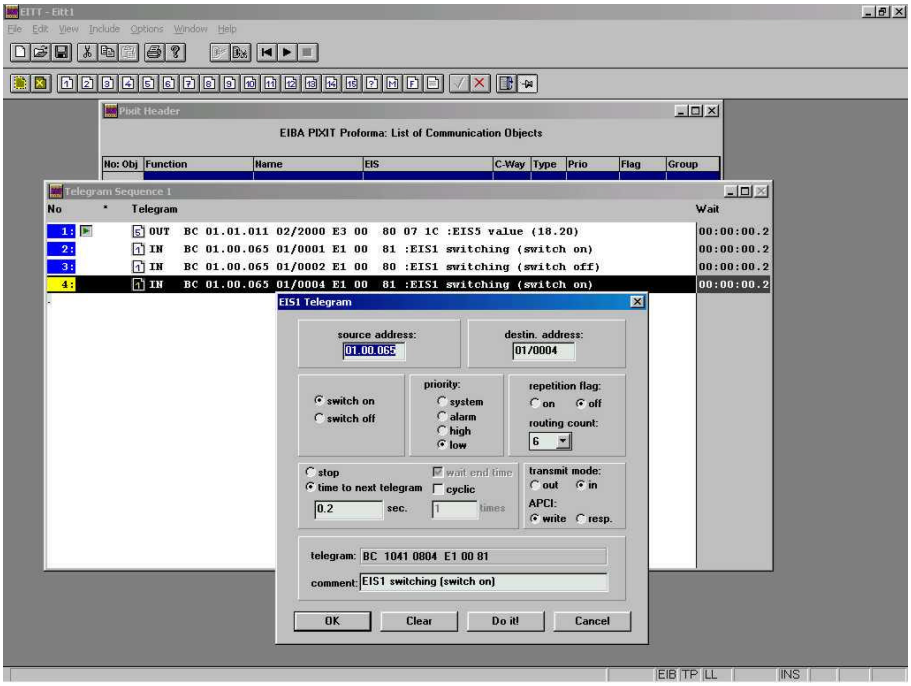


Bild A.12. Ein Beispiel der EITT Oberfläche

Der Busmonitor (Bild A.13) ist ein DOS Programm und dient der manuellen Verwaltung einzelner Geräte (Programmierung der physikalischen Adresse, Auffüllung von Kommunikations- und Gruppentabellen, Herunterladen der Applikation usw.). Er wird auch zur Beobachtung des Busverkehrs verwendet. Der Busmonitor bietet dem Anwender die Möglichkeit, ein beliebiges von ihm zusammengestelltes Telegramm auf dem Bus zu verschicken, um die einzelnen Geräte oder ihre Funktionen zu testen.

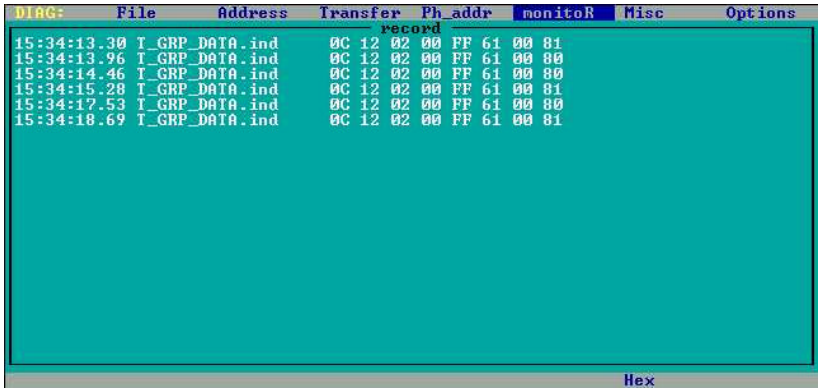


Bild A.13. Die Oberfläche des Busmonitors

Die ETS ist die am meisten eingesetzte und verbreitete EIB Software. Sie besteht aus getrennten Modulen (Bild A.14). Das Haupteinsatzgebiet der ETS ist die Projektierung und die Inbetriebnahme der einzelnen EIB Installationen. Sie enthält auch Mittel zum Testen der korrekten Programmierung von EIB Geräten und ihrer Funktionalität. In der Variation ETS+ enthält sie auch ein zusätzliches Modul zur Datenaufbereitung für die zentrale Datenbank der EIB Geräte.

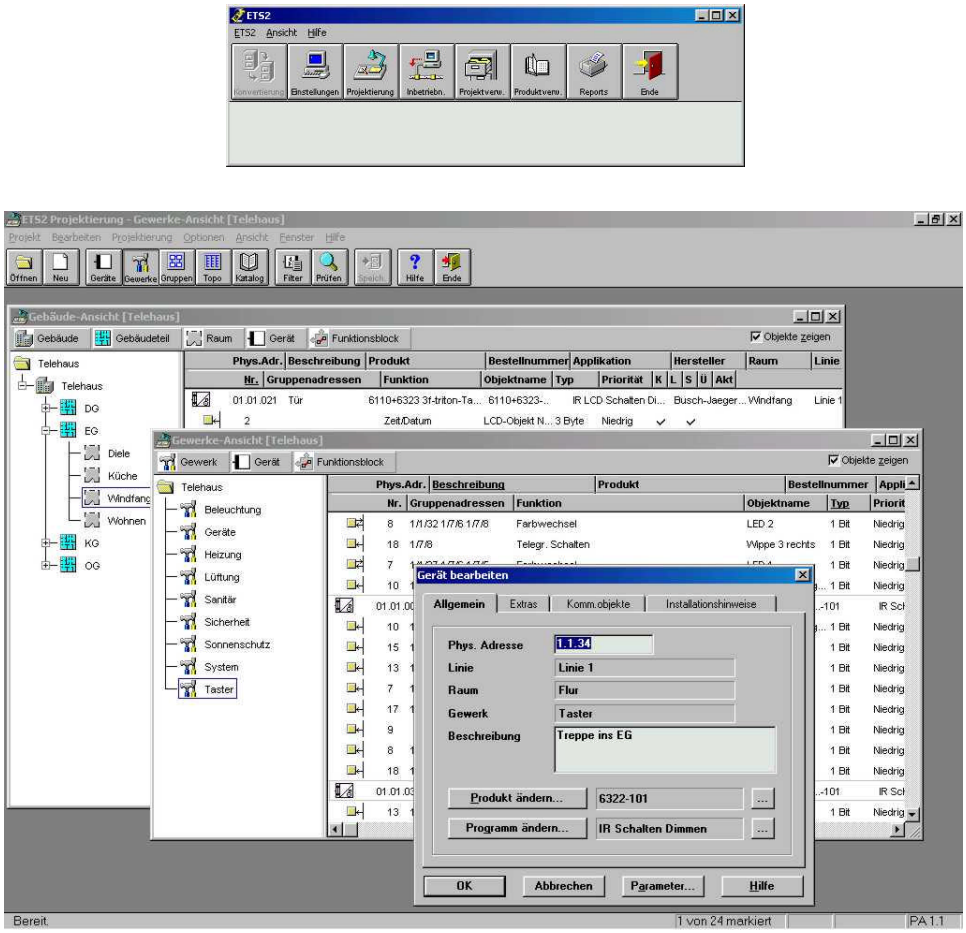


Bild A.14. Das Hauptmenü der ETS und die Oberfläche des Projektierungsmoduls

Das Softwarepaket EIB-IDE (Bild A.15) ist in erster Reihe für die Entwickler der EIB Applikationen für die BCU's und BAU's von Interesse. Es beinhaltet einen Quelltexteditor, einen Compiler für die in den BAU's eingesetzten Mikrokontroller, einen Debugger und auch alle nötigen Header- und Definitionsdateien, die die Entwicklung von neuen Applikationen deutlich vereinfachen.

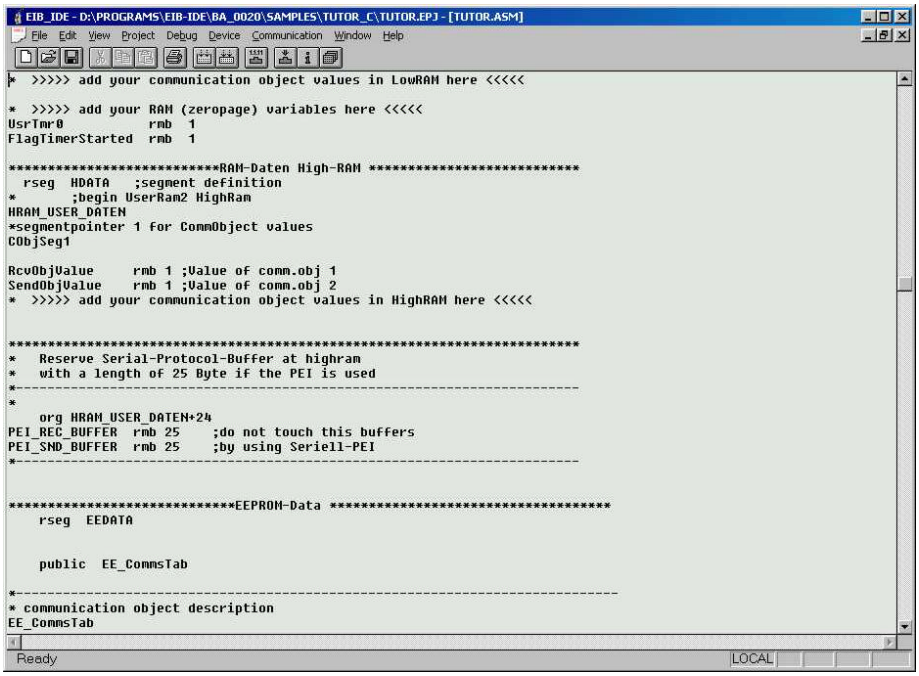


Bild A.15. EIB-IDE

Anhang B

Abkürzungsverzeichnis

AM	Application Module
ANSI	American National Standards Institute
API	Application Program Interface
BACnet	Building Automation and Control Network
BAU	Bus Access Unit
BCU	Bus Coupler Unit
BIM	Bus Interface Module
BySyS	Building Energy Management With Smart Systems
CAN	Controller Area Network
CENELEC	European Committee for Electrotechnical Standardization
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CSS	Cascading Style Sheets
DCOM	Distributed Component Object Model
DECT	Digital Enhanced Cordless Telecommunication
DHCP	Dynamic Host Configuration Protocol
DIN	Deutsche Industrienorm
DLL	Dynamic Link Library
DNS	Domain Name Service
DOS	Disk Operating System
DSL	Digital Subscriber Line
EEPROM	Electrically Erasable Programmable Read-Only Memory
EHS	European Home System
EIA	Electronic Industries Alliance
EIB	European Installation Bus
EIBA	European Installation Bus Association
EIS	EIB Interworking Standard
EITT	EIB Interoperability Test Tool
EN	European Norm

ETS	EIB Tool Software
GENA	Generic Event Notification Architecture
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IEC	International Engineering Consortium
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IMOS-*home*	Integriertes Management Offener Systeme für das private Heim
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISO	International Standardization Organization
IWO-BAY	Innovatives Wohnen in Bayern
JVM	Java Virtual Machine
LON	Local Operating Network
LWL	Lichtwellenleiter
MAC-Adresse	Media Access Control Adresse
MIB	Management Information Base
MMI	Man-Machine-Interface
ODBC	Open DataBase Connectivity
OLE	Object Linking and Embedding
OPC	OLE for Process Control
OSG	Open Services Gateway
OSGi	Open Services Gateway Initiative
OSI	Open Systems Interconnection
PDA	Personal Digital Assistant
PDU	Protocol Data Unit
PEI	Physical External Interface
RAM	Random Access Memory
RFC	Requests For Comments
ROM	Read-Only Memory
RPN	Reverse Polish Notation
SDK	Software Development Kit
SNMP	Simple Network Management Protocol

SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSDP	Simple Service Discovery Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UPnP	Universal Plug and Play
VDE	Verband der Elektrotechnik, Elektronik und Informationstechnik e.V.
VIMP	Verteilte Intelligente Mikrosysteme für den Privaten Lebensbereich
VSOS	Virtual Shared Object Space
WAP	Wireless Application Protocol
WWW	World Wide Web
WYSIWYG	What You See Is What You Get
XML	Extensible Markup Language

Tabellenverzeichnis

Abbildungsverzeichnis