Jan Jurczynski

Konzeption und Entwurf einer eingebetteten Kommunikationsschnittstelle für verteilte Robotikanwendungen

Otto-von-Guericke-Universität
Magdeburg
Fakultät für Informatik
Institut für Verteilte Systeme

Studienarbeit

Konzeption und Entwurf einer eingebetteten Kommunikationsschnittstelle für verteilte Robotikanwendungen

Autor:	Jan Jurczynski
Betreuer:	Maik Mory (IFF) Sebastian Zug (OvG)
Verantwortlicher Hochschullehrer:	Prof. Dr. Jörg Kaiser

SS 2007

Universität Magdeburg
Fakultät für Informatik
Postfach 4120
D-39016 Magdeburg
Germany

Jurczynski, Jan:
Konzeption und Entwurf einer eingebetteten Kommunikationsschnittstelle für verteilte Robotikanwendungen
Studienarbeit, Otto-von-Guericke-Universität
Magdeburg, 2007.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADC	Analog - Digital - Converter
ARM	Acorn Risc Machine
CAN	Controller Area Network
CF	CompactFlash
CORBA	Common Object Request Broker Architecture
I/O	Input/Output (dt. Eingabe und Ausgabe)
IDE	Integrated Drive Electronics
KSPS	Kilosamples per Second
MHz	Megahertz
RAM	Random Access Memory
SDRAM	Synchronous Dynamic Random Access Memory
TCP	Transmission Control Protocol
UART	Universal Asyncronous Receiver Transmitter
USB	Universal Serial Bus

1 Einleitung

1.1 Motivation

Roboter werden eingesetzt um dem Menschen gefährliche, monotone oder unliebsame Aufgaben abzunehmen. Zu diesen Aufgaben gehören zum Beispiel die Erkundung unzugänglicher Räume, wie Abwasserkanäle oder Luftschächte. Sie führen die Überwachung von Objekten oder die Reinigung von Fassaden durch. Dazu erledigen sie einfache Aufgaben automatisiert.

Die Anforderungen an ein Robotersystem sind hoch. Es muss sehr zuverlässig arbeiten, robust gegenüber Störungen und Umwelteinflüssen und zudem flexibel Einsetzbar sein.

Am Fraunhofer Institut für Fabrikbetrieb und -automatisierung, IFF, in Magdeburg, werden Service Roboter mit Hilfe von Verteilten Systemen realisiert. Die Kommunikation zwischen den Rechnern wird durch eine "Common Object Request Broker Architecture" kurz CORBA realisiert.

Um die Effizienz der Roboter zu steigern, können Embedded Systeme integriert werden. Sie erhöhen aufgrund kompakter Bauweise die Mobilität der Robotersysteme.

1.2 Aufgabenstellung

Ziel der Arbeit ist der Aufbau einer leistungsfähigen, vielfältig einsetzbaren Hardwareplattform. Die Plattform soll Anwendungen in einem verteilten System verschiedene Dienste zur Verfügung stellen. Als Basis soll ein Embedded System dienen.

Damit das Embedded System auf einem breiten Anwedungsfeld einsetzbar ist, muss es diverse Schnittstellen vorweisen können. Die anzubietenden Dienste reichen von der analogen Ein - und Ausgabe, über Ethernet und

USB bis hin zu Interfaces für Bussysteme, wie z.B. CAN. Zur Erfüllung der Aufgaben, muss eine geeignete Embedded Plattform mit den verschiedenen Schnittstellen ausgewählt werden.

Die Kommunikation in dem verteilten System erfolgt über eine CORBA basierte Service-Architektur (siehe 4). In Abhängigkeit von der Leistungsfähigkeit der Plattform und dem verwendeten Betriebssystem wird eine Portierung der Service-Architektur notwendig.

Um die Dienste des Embedded Systems dann letztendlich Anwendungen des verteilten Systems zu Verfügung zu stellen, muss eine Möglichkeit für eine einheitliche Dienstbeschreibung definiert werden.

2 Grundlagen

In diesem Kapitel werden Grundlagen vermittelt, die notwendig für das Verständnis der Arbeit sind. Zunächst stehen Hardwarebeschreibungen der Anforderungen an ein Embedded System im Mittelpunkt. Dann werden wichtige Softwaregrundlagen erklärt und zum Ende wird auf Benchmarking, die hier verwendete Bewertungsmethode, eingegangen.

2.1 Hardware technische Grundlagen

2.1.1 Eingebettete Systeme

Embedded Systems (dt. eingebettete Systeme) sind eine Kombination aus Hardware und Software, die zur Lösung einer ganz bestimmten Aufgabe dienen [18]. Das einfachste Embedded System stellt ein Single-Chip Microcontroller dar. Hier sind neben Prozessor und Speicher weitere Ein- und Ausgabemöglichkeiten wie serielle Schnittstellen auf einem einzigen Chip vereinigt.

Das Herzstück eines Embedded Systems ist ein Microcontroller. Dieser benötigt weitere, externe, Speicherbausteine. Diese Speicherbausteine umfassen einen nichtflüchtigen, Read Only Memory (ROM), und einen temporären Zwischenspeicher, Random Access Memory (RAM). Im ROM wird die Firmware des Embedded Systems abgelegt, hier kommt ein so genannter FLASH Speicher zum Einsatz. Während der Laufzeit dient der RAM zum Zwischenspeichern von Daten und zur Ablage des Prozessor Stack.

Als bekannte Vertreter sind hier die Microcontroller der Atmel-AVR-Familie zu nennen. Sie verfügen im Allgemeinen über Digitale I/Os, Interrupteingänge, mehrere UARTs, Zeitgeber, und einen Watchdog Timer. Außerdem besitzen viele einen integrierten Analog-Digital-Umsetzer (ADU) und Digital-Analog-Umsetzer (DAU) (siehe 2.1.6) sowie einen CAN-Controller (siehe 2.1.5).

Acorn Risc Machine (ARM) sind hier als Vertreter für Single-Chip-Microcontroller zu nennen. Sie besitzen ähnliche Schnittstellen wie die zuvor beschriebenen Microcontroller. Außerdem sind sie noch mit Speicherbausteinen ausgestattet. Arbeitsspeichergrößen von 32 MB sind häufig zu finden.

Embedded Systeme zeichnen sich durch eine geringe Größe und einem geringerem Strombedarf aus. Außerdem kommen sie ohne rotierende und mechanische Laufwerke aus, was einen langfristigen, wartungsarmen Betrieb ermöglicht. Zudem können sie leicht in andere Systeme integriert werden. So findet man sie z.B. als Steuereinheiten u.a. als Controller des Anti Blockier Systems im Auto oder als Zeitschaltuhr in der Mikrowelle. Sie sind in vielen alltäglichen Geräten eingebettet ohne dass sie bemerkt werden.

Für die meisten dieser Anwendungen werden 8 oder 16 Bit Single-Chip-Microcontroller verwendet. Diese werden in der Regel ohne Betriebssystem verwendet. Die auf ihnen implementierten Anwendungen laufen direkt auf der Hardware. Sinnvoll wird dies erst bei großen Stückzahlen. So ist ohne Betriebssystem eine längere Entwicklungsdauer vorzusehen. Trotzdem können Kosten eingespart werden, da weniger Ressourcen auf dem Embedded System benötigt werden und Lizenzkosten für das Betriebssystem gänzlich entfallen [18].

Sollen jedoch komplexere Aufgaben bewerkstelligt werden, ist ein Betriebssystem unabdingbar. Dazu können, und werden auch, 32 Bit Prozessoren eingesetzt werden. Es gibt spezielle Embedded Betriebssysteme wie z.B Windows CE, Windows XP Embedded, QNX, und diverse Linux Distributionen.

2.1.2 Prozessor und Speicher

Wie in 2.1.1 bereits erwähnt, stellt die Central Processing Unit (CPU), den zentralen Hauptprozessor eines Rechnersystems dar. Sie übernimmt die Steuerung und Koordination der Hard - und Software [12].

Der Arbeitsspeicher (RAM), eines Rechners bedingt den Einsatz von Betriebssystemen und Programmen. Es ist ein relativ schneller Speicher, in dem die CPU oft benötigte Daten ablegt um schnell auf diese zugreifen zu können. Je mehr Speicher vorhanden ist, desto umfangreichere Programme können ausgeführt werden. Von daher ist es vorteilhaft über einen großen Speicher zu verfügen.

2.1.3 Serielle Schnittstellen

Asynchrone serielle Schnittstellen (UART, engl. Universal Asynchronous Receiver Transmitter) dienen der Kommunikation mit anderen Systemen [18]. Sie gehören zu den standardmäßigen Eingabe- und Ausgabefunktionen und stehen auf vielen Embedded Systemen zur Verfügung. Als Beispiel sei hier die RS232-Schnittstelle genannt. Sie ist Desktop-PCs als COM1- oder COM2-Port bekannt.

Die Kommunikation erfolgt über getrennte Leitungen zum Senden und Empfangen. Die mechanische Verbindung zwischen zwei UARTs erfolgt standardmäßig über sogenannte DE9 (D-Sub 9-polig) Stecker und Buchsen.

2.1.4 Universal Serial Bus

Der Universal Serial Bus (USB) ist eine standardisierte und weitverbreitete I/O-Schnittstelle [12]. Mit USB können verschiedene Geräte, wie Festplatten, Tastatur, Maus und Drucker angeschlossen werden. Der Host Adapter ermittelt selbstständig welches Gerät angeschlossen ist und lädt die entsprechenden Treiber. Zudem kann die Stromversorgung von Geräten geringer Leistung direkt über das USB Kabel erfolgen.

Es können bis zu 127 Geräte, mit entsprechenden USB Hubs, an einen Host angeschlossen werden. Ein automatisches Laden der Gerätetreiber ermöglicht eine Hot-Plugging Funktion. Hot-Plugging ist die Fähigkeit Komponenten während des Betriebes anzuschließen oder zu entfernen.

Es besteht die Möglichkeit eine garantierte Übertragungsleistung zur Verfügung zu stellen. Die Gesamttransferleistung wird dabei auf 1,5 MByte/s reduziert. Die Fähigkeit zu einer garantierten Übertragungsleistung versetzt den Universal Serial Bus in die Lage zeitkritische Übertragungen durchführen zu können [12].

Es gibt drei Versionen des USB-Standards. Sie unterscheiden sich in der Übertragungsgeschwindigkeit. Die USB Version 1.0 hat eine maximale Übertragungsgeschwindigkeit von 1,5 MBit/s. Mit der Version 1.1, können bereits maximal 12 MBit/s erzielt werden. Die neueste Version - USB 2.0 - kann Daten mit bis zu 480 MBit/s übertragen.

Zu bemerken ist die volle Abwärtskompatbilität eines USB 2.0 Ports. Jedoch kann dann nur die maximale Übertragungsgeschwindigkeit der älteren Version genutzt werden. Zwischen einem USB 1.1 Gerät und einem USB 2.0 Host sind also maximal 12 MBit/s möglich. [4]

2.1.5 Feldbusse und Rechnernetze

Feldbusse werden zum Verbinden unterschiedlicher Geräte wie z.B. Aktoren, Sensoren und einem Steuergerät eingesetzt [13]. Sie kennzeichnen sich im Wesentlichen durch das Vorhandensein eines einzigen Übertragungsweges, den jeder Teilnehmer benutzt, aus [11]. Aufgrund der seriellen Verdrahtung können Kosten, u.a. für Planung und Material, eingespart und Fehler, wie z.B. bei der Verdrahtung, vermieden werden.

Feldbussysteme erhöhen die Flexibilität der Anwendungen, da einzelne Komponenten sehr einfach ausgetauscht, repariert oder hinzugefügt werden können. Das System kann sich dazu selbst diagnostizieren und den Ausfall einzelner Komponenten anzeigen. Die Flexibilität und die Fähigkeit der Selbstdiagnose erhöhen die Zuverlässigkeit und das System bietet so eine höhere Verfügbarkeit. Eine weitere wichtige Eigenschaft von Feldbussystemen ist ihre Echtzeitfähigkeit [7]. Das heißt, dass maximale Übertragungszeiten von Nachrichten garantiert werden können.

Es gibt mehrere Protokolle die auf einem Feldbussystem implementiert sind. Ein sehr bekanntes und weit verbreitetes Protokoll ist das Controller Area Network (CAN) (siehe 2.1.5 auf der nächsten Seite). Genannt seien neben CAN noch PROFIBUS, IINTERBUS und LIN. Diese Mannigfaltigkeit der Protokolle ist nachteilig anzusehen, denn so muss ein System eventuell mehrere Protokolle unterstützen. Dadurch wird das System sehr komplex und natürlich auch teurer.

Rechnernetze hingegen stellen einen Zusammenschluss mehrerer autonomer Computer dar [17]. Sei bestehen oftmals aus mehreren Teilnetzen. Während Feldbusse für kurze Übertragungswege (bis 100 m [9]) verwendet werden, können Rechnernetze sehr weite Strecken und Flächen abdecken. Als bekannteste Vertreter sind hier das Local Area Network (LAN) und das Internet zu nennen. Für die drahtgebundene Datenübertragung wird das Ethernet verwendet (siehe 2.1.5 auf der nächsten Seite). Beide basieren auf dem Internet Protocol (IP, siehe 2.2.3).

Feldbusse und Netze unterscheiden sich im Wesentlichen durch die zu übertragenen Datenmengen, der Häufigkeit einer Übertragung, der Reaktionszeit und der Gleichmäßigkeit der Netzbelastung [9]. In einem Rechnernetz werden weitaus mehr Daten als in einem Feldbussystem übertragen. In einem Bussystem finden regelmäßig Übertragungen kleiner Datenmengen statt, während ein Netz große Datenmengen zu nicht vorhersehbaren Zeitpunkten übertragen muss. Ebenso unterscheiden sich beide im Hinblick auf die Echtzeitfähigkeit. Diese spielt bei einem Netz eine untergeordnete Rolle, ist aber unabdingbar für die meisten Feldbussprotokolle.

Ethernet

Das Ethernet (IEEE 802.3) ist ein Busnetzwerk [10]. Es stellt den aktuellen Übertragungsstandard in drahtgebundenen Rechnernetzen dar. Ebenso wie ein Feldbussystem verfügt Ethernet über nur einen Kommunikationskanal.

Der Übertragungsstandard wurde ursprünglich von der Firma Xerox entwickelt und von Xerox, Intel und Digital Equipment auf eine gemeinsame Spezifikation gebracht. Es setzt die untersten zwei Schichten des ISO/OSI Modells um.

Mit Ethernet sind mehrere verschiedene Topologien möglich. Es kann u.a. stern- oder auch ringfömig aufgebaut sein. Zur Kommunikation zwischen zwei Endknoten werden Punkt-zu-Punkt Verbindungen aufgebaut [9]. Liegen die Knoten in verschiedenen Teilnetzen, geschieht die Weiterleitung der Pakete von bestimmten zentralen Stationen (Router).

Zur Drahtanbindung können unterschiedliche Kabeltypen eingesetzt werden. Als bekannteste Vertreter sind hier das Twisted-Pair-Kabel mit sogenanntem RJ45-Stecker und das Koaxialkabel mit sogenanntem BNC Stecker. Des weiteren können Lichtwellenleiter und auch eine drahtlose Übertragung eingesetzt werden.

Mit dem kabelgebundenem IEEE 802.3 Standard können Übertragungsraten von 10 MBit/s, 100 MBit/s, oder auch bis zu 1000 MBit/s, bei drahtloser Übertragung sind 54 MBit/s möglich, erzielt werden.

Controller Area Network

Das Controller Area Network (CAN) ist ein definierter Standard nach (ISO/DIS 11898 und ISO/DIS 11519-1) [9]. Der CAN Bus wurde von Bosch für Automobilanwendungen entwickelt [7]. Das serielle Bussystem findet zudem auch Verwendung in der Industrieautomatisierung. Von Vorteil ist eine internationale Normierung, ein weites Anwendungsfeld und ein weite Verbreitung.

Die Übertragungsraten liegen zwischen 1 MBit/s und 5 KBit/s. Die Übertragungslänge hängt von der Übertragungsrate ab. Je höher die Rate, desto kürzer fällt die Übertragungslänge aus. Bei 1 MBit/s beträgt die Übertragungslänge ca. 40 Meter und bei einer Rate von 5 KBit/s sind bis ca. 10 Kilometer möglich.

Der CAN zeichnet sich durch eine hohe Übertragungssicherheit, einer prioritätsgesteuerten Übertragung und Echtzeitfähigkeit aus. Es existieren verschiedene Versionen der CAN Spezifikation. Zum Einen die CAN Version 2.0A mit einem 11 Bit Identifier und zum Anderen CAN 2.0B, welches über einen 29 Bit Identifier verfügt. Werden beide Versionen in einem Feldbussystem verwendet, können effektiv nur 11 Bit Identifier genutzt werden.

Die Arbitrierung [13] im CAN erfolgt, indem die Identifier der einzelnen Kommunikationsteilnehmer bitweise verglichen werden. Rezessive Bits (logisch "1") werden hierbei von dominanten Bits (logisch "0") überschrieben. Auf diese Weise wird eine Priorisierung der Nachrichten erreicht, da kleinere Identifier einen jeweils Höheren überschreiben. Jeder Identifier darf daher auf Senderseite nur einmal vergeben werden. Auf Empfängerseite hingegen merhfach.

Das Controller Area Network hat aufgrund hoher Fertigungszahlen den Vorteil von kostengünstigen Chips. Diese ermöglichen die Entwicklung eigener angepasster Baugruppen. Ein hier nennenswerter Vertreter ist beispielsweise der SJA1000 von Phillips. Es gibt aber auch einzelne Controller, die an einen Prozessor angeschlossen werden können. Miteinander verbunden werden CAN Controller u.a. über sogenannte DE9 (D-Sub 9polig) Stecker und Buchsen.

2.1.6 Analoge Ein- und Ausgabe

Um Sensordaten mit einem Rechner auslesen und ihre Daten digital verarbeiten zu können, werden Analog-Digital-Umsetzer (ADU) eingesetzt. Sensoren geben ein analoges und stetiges Signal aus. Es besitzt zu jedem Zeitpunkt einen bestimmten Wert. Ein ADU erhält dieses als Eingangssignal und quantisiert es in ein digitales, für einen Rechner verständliches Signal. Ein digitales Signal hingegen liefert lediglich zeitlich diskrete Werte.

Analog-Digital-Wandler werden nach ihrer Abtastrate, der Genauigkeit, und der Wandlungsgeschwindigkeit charakterisiert. Mit der Abtastrate (oder auch Abtast-Frequenz) wird der minimale zeitliche Abstand zwischen zwei aufeinander folgenden Konvertierungen angegeben. Laut dem Nyquist-Abtasttheorem ergibt sich für jede Anwendung eine kleinste zulässige Abtastrate [16]. Diese soll demnach minimal das 2,5 fache der höchsten Frequenz des Eingangssignals betragen. Um eine große Bandbreite von Anwendungsfällen abdecken zu können, erweist sich also eine möglichst hohe Abtastrate als vorteilig.

Die Genauigkeit des digitalen Wertes wird in Bit angegeben. Häufig verwendete Auflösungen liegen bei Werten von 1, 8, 12, 16 oder 24 Bit. Für eine Vielzahl von industriellen Anwendungen hat sich eine Auflösung von 12 Bit als Standard herausgestellt [16]. Die Erfassung zeitlich rasch veränderlicher Messgrößen erfordert eine hohe Wandlungsgeschwindigkeit. Zur Digitalisierung von z.B. Videodaten wird eine Wandlungsgeschwindigkeit von mindestens 20 MHz benötigt [11].

Digital-Analog-Umsetzer (DAU) haben die gleichen Anforderungen an Wandlungszeit, Genauigkeit und Geschwindigkeit wie Analog-Digital-Umsetzer. Sie erhalten, im Gegensatz zu einem ADU, ein digitales Eingangssignal und wandeln dieses in ein stetiges, analoges Signal. Das analoge Signal wird dann z.B. zur Ansteuerung eines Motors verwendet.

2.2 Softwaretechnische Grundlagen

2.2.1 Verteilte Systeme

Ein verteiltes System besteht aus mehreren miteinander verbundenen, autonomen Computern. Für einen Benutzer sind diese nicht sichtbar. Ein Benutzer hat keine Kenntnis über die Existenz mehrerer Prozessoren. Das verteilte System stellt sich ihm als ein virtuelles Einzelprozessorsystem dar [17]. Es arbeiten verschiedene Softwarekomponenten auf veschiedenen Hardwaresystemen zusammen. Diese müssen ihre Aktionen durch den Austausch von Nachrichten koordinieren [3].

Neben der Transparenz gegenüber dem Vorhandensein mehrerer Prozessoren, bleibt u.a auch das Betrtiebssystem oder die Art der Kommunikation verborgen. Auf diese Weise muss sich ein Entwickler nur mit der Implementation seiner Algorithmen und Applikationen befassen. Die Verteilung auf verfügbare Ressourcen übernimmt eine spezielle Software für ihn, die Middleware. Als Beispiele für eine Middleware sind hier die Common Object Request Broker Architecture (CORBA, siehe 2.2.2) und die Oracle Fusion Middleware zu nennen.

Charakteristisch für ein verteiltes System ist neben der bereits erwähnten transparenten Darstellung, der einzelnen, miteinander verbundenen Hardwarplattformen, die sogenannte Nebenläufigkeit. Allgemein formuliert, handelt es sich bei Nebenläufigkeit um eine gleichzeitige Nutzung der

vorhandenen Ressourcen. Somit ist es u.a. möglich mehrere Prozesse gleichzeitig auf mehreren Prozessoren laufen und diese auf einen gemeinsamen Speicher zugreifen zu lassen.

Damit der Ausfall einzelner Komponenten, Prozessen, Hardwareplattformen oder Netzwerkverbindungen nicht zum Ausfall des gesamten Systems führt, verfügen alle Komponenten über eine Fehlererkennung und -verwaltung. Dazu besitzt jede Komponente Kenntnis über seine Abhängigkeiten zu anderen und wie ein aufgetretener Fehler behandelt werden kann.

Ein großes Problem in einem verteilten System stellt die Zeitsynchronisation dar. Synchronisierte Uhren sind u.a. wichtig für einen gemeinsamen Speicherzugriff. Da die einzelnen Komponenten ihre Aktionen mittels Austausch von Nachrichten koordinieren, benötigen sie auch hier einen einheitlichen Zeitstempel. Zur Synchronisation der Uhren gibt es spezielle Algorithmen wie z.B. das Network Time Protocol oder den Algorithmus von Christian.

Die Komponenten des verteilten Systems, stellen aus Sicht der Middleware Objekte dar. Ein Objekt bietet anderen Objekten speziell definierte Schnittstellen an. Über diese Schnittstellen stehen die Dienste des Objektes anderen zur Verfügung. Alle Objekte können sich mit Hilfe bestimmter Dienste wie z.B. einem CORBA Name-Service finden und somit die Dienste anderer Objekte nutzen [14].

Die Implentation der Schnittstellen bleibt anderen Objekten verborgen. Auf diese Weise kann z.B. die Implementation oder sogar die gesamte Komponente verändert bzw. ausgetauscht werden.

2.2.2 Common Object Request Broker Architecture (CORBA)

Die Common Object Request Broker Architecture (CORBA) ist ein Standarmechanismus zur Definition von Schnittstellen zwischen den Komponenten eines verteilten Systems (siehe 2.2.1 auf der vorherigen Seite) [14]. Entwickelt und verbreitet wird CORBA von der Object Management Group. Diese Gruppe definiert außerdem bestimmte Standarddienste wie z.B. Name-Services, Persistent Object Services und Transaction Services. Jeder dieser Dienste steht allen Anwendungen im Verteilten System zur verfügung.

Eine wichtige Eigenschaft ist seine Plattformunabhängigkeit. Es steht für beliebige Betriebssysteme und Programmiersprachen zur Verfügung. CORBA

ist objektorientiert. Die Kommunikation zwischen einzelnen CORBA Objekten, die über das Internet Inter-ORB Protocol (IIOP) geschieht, ist ebenso unabhängig vom Kommunikationsprotokoll. Jedoch hat sich hier TCP/IP (siehe 2.2.3) herauskristallisiert.

Werden Remote-Methodenaufrufe gesendet bzw. empfangen, muss das Remote Object aufgefunden werden und das Format von Parametern und Rückgabewerten (Marshaling) übertragen werden. Dazu gibt es den Object Request Broker (ORB). Dies ist eine Software Komponente, die eine einfacherere Kommunikation zwischen den Objekten ermöglicht.

Die Definition der Schnittstellen eines CORBA-Objektes erfolgt mit der Interface Definition Language (IDL). Mit der IDL werden keine Anwendungen sondern lediglich Interfaces festgelegt. Sie können auf jede beliebige Programmiersprache abgebildet und von dieser implementiert werden. So kann beispielsweise ein in C++ implementierter Server mit einem Java Client direkt über ein solches Interface kommunizieren.

Das CORBA Kommunikationsmodell basiert auf der Verwendung von Objektreferenzen, den Interoperable Object References (IORs), welches ebenfalls zur einfacheren Kommunikation zwischen den Objekten beiträgt. Möchte nämlich eine Anwendung auf ein Objekt zugreifen erhält sie eine Referenz auf das Objekt, die IOR. Eine Anwendung kann als Client eines Objekts betrachtet werden, da sie dessen angebotene Dienste in Anspruch nimmt. Eine Komponente kann zudem gleichzeitg als Server, indem es Dienste zur Verfügung stellt, und auch als Client, durch Inanspruchnahme von Diensten anderer Objekte, fungieren.

Für den Zugriff auf ORB Funktionen wie z.B. der Objektaktivierung oder der Objektpersistenz stellt CORBA einen sogenannten Basic Object Adapter (BOA) bereit. Der BOA stellt eine Schnittstelle zwischen einem Objekt und dem ORB dar.

2.2.3 Kommunikationsprotokolle

Die Kommunikation in verteilten System geschieht u.a. über das Ethernet (siehe 2.1.5). Es bedient sich dazu verschiedenen Protokollen.

Das Internet Protokoll (IP) ist auf der Internetschicht des TCP/IP-Modells implementiert. Es hat die Aufgabe Daten von der Quelle zum Ziel zu befördern. Unter Umständen müssen diese auch durch dazwischenliegende Netze hindurch, bis in das Zielnetz geroutet werden.

Durch IP-Adresse und Subnetzmaske (subnet mask) werden Computer innerhalb eines Netzwerkes in logische Einheiten, so genannte Subnetze, gruppiert. Auf dieser Basis ist es möglich, Computer in größeren Netzwerken zu adressieren und Verbindungen zu ihnen aufzubauen, da logische Adressierung die Grundlage für Routing (Wegewahl und Weiterleitung von Netzwerkpaketen) ist. Das Internet Protocol stellt die Grundlage des Internets dar.

Das Transmission Control Protocol (TCP) ist auf der sogenannten Transportschicht des TCP/IP-Modells implementiert [17]. Die Kommunikation via TCP ist zuverlässig und verbindungsorientiert. Die Zuverlässigkeit wird durch Bestätigungssignale (Acknowledgement (ACK)) aller korrekt erhaltenen Nachrichten gewährleistet. Hat ein Empfänger eine Nachricht korrekt erhalten, sendet er ein ACK an den Sender zurück. Erhält der Sender das ACK weiß er, dass seine Nachricht korrekt übertragen wurde und er nun die nächste Nachricht senden kann. Sollte kein ACK ankommen, ist die Nachricht nich korrekt übertragen worden und der Sender verschickt die Nachricht erneut.

Wie bereits erwähnt ist die Datenübertragung bei TCP verbindungsorientiert. Die Datenübertragung, zwischen zwei Kommunikationspartnern, erfolgt hier über einen logischen Kanal. Dazu wird zuerst eine Verbindung zwischen beiden aufgebaut. Dann werden Nachrichten ausgetauscht und zum Ende muss die Verbindung wieder abgebaut werden.

TCP kann die Geschwindigkeit der Datenübertragung steuern. Kann ein Empfänger nicht alle Daten in der Geschwindigkeit verarbeiten, in der sie eintreffen, kommt es zum Datenverlust. Um dies zu verhindern wird der Algorithmus der Flusssteuerung angewendet. Die Sendegeschwindigkeit des Senders wird hierzu an die Empfangsgeschwindigkeit des Empfängers angepasst. Auf diese Weise kann dieser nicht mit Nachrichten überschwemmt werden und ein Datenverlust wird verhindert.

Das User Datagram Protocol (UDP) hingegen ist unzuverlässig und verbindungslos. Werden also Daten mit UDP übertragen, erhält der Sender keinerlei Informationen über den Erhalt, oder auch Nichterhalt, einer Nachricht. Auch wird hier keine Verbindung wie bei TCP aufgebaut. Sollen Daten gesendet werden, werden diese einfach unter Angabe der entsprechenden Zieladresse versendet. UDP unterstütz demzufolge auch keine Flusskontrolle. Verwendet wird es hauptsächlich für einmalige Abfragen und Situationen, in denen Schnelligkeit wichtiger als Genauigkeit ist.

Das Internet Control Message Protocol (ICMP) ist ein Steuerprotokoll. Wenn etwas unerwartetes passiert, wird dies von ICMP berichtet. Es gibt zahlrei-

che definierte ICMP-Nachrichten. Ein wichtiges Beispiel ist die Nachricht TIME EXCEEDED. Diese wird von ICMP versendet, wenn der Timer eines Paketes Null erreicht hat und es verworfen wird. Die ICMP-Nachrichten werden, in einem IP Paket verkapselt, übertragen.

2.2.4 Betriebssysteme

Ein Betriebssystem bildet eine Schicht zwischen Hardware und Anwendungssoftware in Rechnersystemen. Es stellt der Anwendungssoftware Schnittstellen zur Hardware bereit [18]. Im Allgemeinen wird die Hardware vom Betriebssystem vollständig isoliert und stellt einer Anwendung bestimmte Hardwarezugriffsfunktionen zur Verfügung.

Neben der Bereitstellung von Schnittstellen, verwaltet ein Betriebsystem die vorhandenen Ressourcen. Dazu gehören Task Management und Benutzerverwaltung, Speicher- und Dateiverwaltung sowie die Verwaltung von Ein- und Ausgaben [1].

Ein Betriebssystem ist multitaskingfähig, wenn es mehrere Programme gleichzeitig ausführen kann. Multitasking erfordert ein entsprechendes Task Management (Scheduling). Das Betriebssystem muss die CPU unter den konkurrierenden Tasks aufteilen. Ebenso muss die CPU unter mehreren angemeldeten Benutzern aufgeteilt werden.

Die Datei- und Speicherverwaltung dient der Vermeidung von Zugriffskonflikten. Diese entstehen, wenn mehrere Prozesse auf eine Speicherstelle oder eine Datei zugreifen wollen. Des weiteren muss das Betriebssystem eine Ein- bzw. Ausgabeeinheit für andere Prozesse blockieren, sobald ein Prozess Zugriff auf eine solche fordert.

Bekannte und weit verbreitete Betriebssysteme sind Microsoft Windows und Linux. Der wesentliche Unterschied liegt in den Anschaffungskosten. Während es zahlreiche kostenlose Linux Distributionen im Internet als einfachen Download gibt, fallen für die Verwendung von Windows Lizenzkosten an. Zudem ist Linux Open Source, das heißt der Linux Quelltext kann von jedermann gelesen, geändert und angepasst werden. Zu erwähnen ist, dass Windows Anwendungen nicht mit Linux kompatibel sind. Ebenso gilt die inverse Richtung.

2.3 Benchmarking

Benchmarking ist ein Standard zum Vergleich von Objekten. Ausgewählte Eigenschaften werden dazu gegenübergestellt. Anhand der erhaltenen Infomationen wird das optimale Objekt aus einer Menge von Objekten bestimmt.

Um einen Vergleich durchführen zu können, müssen einheitliche Größen ermittelt werden. Dies kann auf zweierlei Arten geschehen. Zum Einen können die wesentlichsten Unterschiede beschrieben werden. Zum Anderen besteht die Möglichkeit der Quantifizierung von Merkmalen eines Objektes [2]. Eine Quantifizierung ermöglicht eine analytische Messung der Größen.

Benchmarking findet Anwendung auf mehrere Aspekte. Es kann auf die Herstellung von Produkten, auf die Produkte selbst, auf angebotene Dienstleistungen und auf die dazu erforderlichen Geschäftsprozesse angewendet werden.

In der Computerindustrie wird ein Benchmark zur Messung von Leistungsdaten, von Software- und Hardwaresystemen verschiedener Hersteller angewendet. Das Ergebnis dient als Entscheidungsgrundlage zwischen den alternativen Angeboten. Die Angebote besitzen unterschiedliche Merkmale und Funktionen. Die Gesamtanforderung kann letztlich durch eine unterschiedliche Ausprägung und Mischung der Leistungsdaten erreicht werden [2].

Es existieren vier Grundfragen einer Benchmarking-Studie [15]:

1. Welches Objekt soll verglichen werden?
2. Welche Kriterien dienen der Bewertung?
3. Mit welchen Objekten wird verglichen?
4. Mit welchen Methoden wird bewertet?

Bei Benchmarkingobjekten kann es sich um Produkte, Unternehmensprozesse, Organisationsstrukturen oder Unternehmensstrategien handeln [15]. Im Mittelpunkt dieser Arbeit steht ausschließlich das Produktbenchmarking.

Allgemein können Produkte als Hardware, Software oder Dienstleistungen angesehen werden. Ihre Bewertung hängt dabei u. a. von der Produktstruktur, dem Systemcharakter und der Produkttechnologie ab. In der Robotik beispielsweise sind Produkte eine Zusammensetzung von Hardware, Software und Dienstleistungen.

Die Bewertungskriterien hängen vom Objekt und vom Umfang des Benchmark ab. Die Kriterien umfassen wirtschaftliche, technische und organisatorische Größen. Bei einem Vergleich von Embedded Systemen beläuft es sich in erster Linie auf technische Kriterien. An zweiter, wenn auch nicht weniger wichtiger, Stelle stehen wirtschaftliche Aspekte (siehe 3.2).

Für einen Benchmark müssen immer Referenzobjekte in den Vergleich einbezogen werden. Außerdem muss die Vergleichbarkeit der Objekte gegeben sein. In dieser Arbeit werden verschiedene, vergleichbare Embedded Systeme gegenübergestellt.

Das Ziel einer Benchmark-Studie ist die optimale Erfüllung der gewünschten Anforderungen (siehe 3.1). Mit Hilfe von Bewertungsmethoden kann der Grad der Anforderungserfüllung bestimmt werden. Dazu müssen zunächst exakte und aktuelle Informationen der Objekte und Referenzobjekte ermittelt und gegenübergestellt werden.

3 Auswahl der Hardware

In diesem Kapitel wird der Auswahlprozess eines Embedded Systems (siehe 2.1.1) beschrieben. Dazu wird zunächst auf dessen Anforderungen näher eingegangen. Im Folgenden wird beschrieben wie der Auswahlprozess durchgeführt wurde. Abschließend wird der Testsieger vorgestellt.

3.1 Anforderungen

Der eingebettete Charakter impliziert eine Menge von Schnittstellen, angeordnet auf kleinstem Raum. Der Idealfall stellt ohne Zweifel eine Plattform mit sehr geringen Abmessungen dar. Für die Integration einer eingebetteten Kommunikationsschnittstelle in ein verteiltes System müssen weitere bestimmte Kriterien erfüllt sein.

Damit überhaupt ein Betriebssystem wie Linux oder Windows verwendet werden kann, müssen die folgenden drei Vorraussetzungen mindestens erfüllt sein. Zum Einen wird ein 32 Bit Microcontroller bzw. Microprozessor mit einer Floating Point Unit (FPU) oder einer Memory Management Unit (MMU) benötigt. Um die Entscheidung ob Linux oder Windows eingesetzt wird erst bei Bedarf diskutieren zu können, muss die CPU (siehe 2.1.2) eine Wortbreite von mindestens 32 Bit beherrschen, da kleinere Busbreiten nicht von Linux unterstützt werden [18].

Zur Integration in ein verteiltes System muss eine Middleware (siehe 2.2.1) auf dem Embedded Board betrieben werden können. Die Middleware und laufende Applikationen benötigen für ihren reibungslosen Betrieb ausreichend Arbeitsspeicher. Einige Robotikanwendungen im Frauhofer IFF benötigen, je nach Anwendungsgebiet, eine Kapazität von mindestens 16 MB. Für Anwendungen mit hohem Datenaufkommen empfiehlt sich eine Größe von mindestens 32 Megabyte.

Um der Eigenschaft einer Kommunikationsschnittstelle zu entsprechen, muss zumindest eine Anbindung über Ethernet (siehe 2.1.5) möglich sein. Da in der Industrie der CAN Bus (siehe 2.1.5) sehr verbreitet ist, sollte ein

Embedded System bereits einen solchen Controller vorweisen oder entsprechende Erweiterungsmöglichkeiten vorsehen.

Zur Anbindung peripherer Geräte, müssen standardisierte Ein- und Ausgabemöglichkeiten, wie serielle Schnittstellen (siehe 2.1.3) oder USB (siehe 2.1.4) vorhanden sein. Um Sensordaten auszuwerten

Die Auswertung von Sensordaten erfordert einen Analog-Digital-Wandler (siehe 2.1.6). Dieser sollte die im industriellen Umfeld verwendete Auflösung von 12 Bit erlangen können. Zudem ist es von Vorteil neben dem ADU über einen Digital-Analog-Wandler (siehe 2.1.6) zu verfügen. Damit erweitert sich der Einsatzbereich der Embedded Plattform vom Messen, bis hin zum Steuern eines Aktors. Des weiteren sollen auch spezielle, digitale Signale von der Plattform ausgewertet werden können, was natürlich entsprechende Ein- und Ausgabemöglichkeiten erfordert.

Neben den aufgeführten technischen Eigenschaften spielen natürlich auch wirtschaftliche Faktoren eine entscheidende Rolle. Die finanziellen Daten sollten sich in einem moderaten Rahmen bewegen. Erfüllen mehrere Produkte die technischen Anforderungen gleichermaßen, rückt jeweils das kostengünstigere in die engere Auswahl vor.

3.2 Bewertung der Kriterien

Zunächst mussten Informationen zu einzelnen Anbietern von Embedded Systemen ermittelt werden. Dann wurden einzelne Produkte gegenübergestellt. Detailierte Informationen wurden zusammengetragen und eine Marktübersicht verschafft. Aus dieser Übersicht musste nun ein geeignetes Embedded System, das den Kriterien einer eingebetteten Kommunikationsschnittstelle (siehe 3.1) am besten nachkommt, ausgewählt werden.

Dazu wurden die einzelnen Embedded Systeme einem Benchmark (siehe Abschnitt 2.3) unterzogen. Der Benchmark sollte folgende Fragen klären:

- Wenn, welches Betriebssystem wird unterstützt?
- Welche Spezifikationen besitzen ADU und/oder DAU?
- Kann das Embedded System über eine vorhandene Ethernet Schnittstelle in ein Netzwerk integriert werden?
- Welche Möglichkeit der CAN Anbindung besteht?
- Sind serielle Schnittstellen und USB Ports verfügbar?

- Welche mechanischen Dimensionen sind zu erwarten?
- Welchen Preis hat das System?

Die Gesamtbewertung einer Plattform setzt sich aus den Punkten der einzelnen Kriterien und deren Gewichtungsfaktor zusammen. So gehen die bei einer Betriebssystemunterstützung erreichten Punkte mit dem Faktor 20 in die Gesamtbewertung ein, eventuelle Ethernet-, USB-, CAN- und UART Schnittstellen jeweils mit dem Faktor 10. Ein Analog-Digital-Umsetzer onboard geht ebenfalls mit dem zehnfachen ein. Die Breite und die Länge machen ein Gewicht von jeweils fünf aus und der Preis letztendlich hält einen Anteil von 20 Prozent.

Wird in der Produktdokumentation mit einem **Betriebssystem** (Tabelle 3.1) geworben, hängt die erreichte Punktzahl von der Vielfalt unterstützter Betriebssysteme ab. Sollte kein Betriebssytem erwähnt werden, muss von einer Embedded Plattform ohne Betriebssystemunterstützung ausgegangen werden. Gibt es Unterstütung für Linux oder Windows erhält es 100 Punkte. Wird entweder nur Linux oder nur Windows unterstützt, gibt es 80 Punkte. Ist es weder möglich Linux noch Windows zu betreiben, bekommt es Null Punkte. Die Null Punkte rechtfertigen sich, da eine CORBA basierte Service Architektur ohne komplettes Betriebssystem nicht ohne weiteres einsetzbar ist. Prinzipiell ist es zwar egal ob Linux oder Windows betrieben wird, jedoch ist es besser die Wahl selbst treffen zu können. Deshalb 100 bzw. 80 Punkte um den Unterschied zu kennzeichnen.

Betriebssystem	Punkte
Windows oder Linux	100
nur Windows	80
nur Linux	80
kein Betriebssystem	0

Tabelle 3.1: Betriebssystemunterstützung

Beim **Ethernet** (Tabelle 3.2 auf der nächsten Seite) kommt es auf die Geschwindigkeit und den mechanischen Kontakt an. Ist eine RJ45-Buchse bereits auf dem Board vorhanden oder muss diese erst an Pins angeschlossen werden? Hat also ein Board eine Geschwindigkeit von einem Gigabit pro Sekunde mit mechanischem Kontakt onboard, sind dies die vollen 100 Punkte. Bei gleicher Geschwindigkeit, jedoch ohne Buchse, werden 80 Punkte vergeben. Beträgt die maximale Geschwindigkeit 100 Megabit pro Sekunde mit entsprechender Buchse, bekommt es 80 Punkte und 70 Punkte

ohne Buchse. Bei nur zehn Megabit pro Sekunde mit Buchse onboard bekommt es noch 60 Punkte und ohne Buchse dann nur noch 50 Punkte.

Ausführung \ Geschwindigkeit	1 GBit	100 MBit	10 MBit
Buchse	100	80	60
Pins	90	70	50

Tabelle 3.2: Wertung des Ethernet

Der Hauptaugenmerk der **USB**-Schnittstelle (Tabelle 3.3) liegt auf dessen Ausführung. Da zum Anschluss anderer Geräte ein USB-Host notwendig ist, muss auch ein solcher auf dem System integriert sein. Ist also mindestens eine USB-Host-Buchse vorhanden, wird die volle Punktzahl vergeben. Ist die Host-Funktion gegeben aber kein mechanischer Kontakt vorhanden, können maximal 80 Punkte erreicht werden. Da an einem USB-Port mehrere Geräte angeschlossen werden können, stellt die Anzahl an Schnittstellen kein entscheidendes Kriterium dar.

Ausführung \ Anzahl	≥ 1
Buchse	100
Pins	80

Tabelle 3.3: Wertung der USB-Schnittstelle

Sind auf einem Board mehr als ein **CAN-**Bus-Interface (Tabelle 3.4) mit DE9-Kontakten vorhanden, wurden die vollen 100 Punkte erreicht. Kann auf diese nur über Stiftleisten zugegriffen werden, können noch 90 Punkte erreicht werden. Ist genau ein Interface vorhanden, gibt es 80 Punkte bei Steckverbindungen und 70 Punkte bei herausgeführter Stiftleiste.

Ausführung \ Anzahl	≥ 2	1
Buchse	100	80
Pins	90	70

Tabelle 3.4: Wertung des CAN Bus Interface

Hat ein Board mehr als drei **serielle Schnittstellen** (Tabelle 3.5) mit 9-poliger D-Sub Buchse, verdient es in dieser Kategorie die vollen 100 Punkte,

ohne Buchse noch 90 Punkte. Bei drei Schnittstellen mit Buchse gibt es 80, ohne noch 70 Punkte. Zwei UART, jeweils mit Buchse ergeben 60 und ohne Buchse noch 50 Punkte. Einmal UART mit Buchse erhält 40 und mit Stiftleisten nur noch 30 Punkte. Wichtig ist auch hier die generelle Verfügbarkeit der Schnittstellen.

Ausführung \ Anzahl	≥4	3	2	1
Buchse	100	80	60	40
Pins	90	70	50	30

Tabelle 3.5: Wertung RS-232

Bei der Bewertung eines **Analog-Digital-Wandlers** (Tabelle 3.6)**,** lag das Hauptaugenmerk auf der Verfügbarkeit. So wurde zuerst die Auflösung bewertet, dann die Anzahl der Kanäle und beide Werte addiert. Bei einer Auflösung von mehr als 12 Bit können 90 Punkte und bei genau 12 Bit 80 Punkte. Bei einer Auflösung von weniger als 12 Bit, können noch 70 Punkte erreicht werden.

Die zu erreichende Punktzahl für die Anzahl der Kanäle wurde an zehn der signifikantesten Werte skaliert. Es werden maximal 10 Punkte für die Anzahl der Kanäle gegeben. So kann auch in dieser Kategorie jedes Board maximal 100 Punkte erreichen.

Bei einem integrierten ADU mit einer Auflösung von z.B. 12 Bit und 16 Kanälen, errechnen sich die Punkte wie folgt. Für die Auflösung werden 80 Punkte vergeben und mit 16 Kanälen kommen noch einmal 8 Punkte hinzu. Beide Werte addiert ergeben eine Punktzahl von 88 in der ADU Kategorie.

Auflösung	Grundwert	Anzahl der Kanäle									
		< 4	4	5	8	10	11	13	16	32	40
<12 Bit	70	71	72	73	74	75	76	77	78	79	80
12 Bit	80	81	82	83	84	85	86	87	88	89	90
>12 Bit	90	91	92	93	94	95	96	97	98	99	100

Tabelle 3.6: Wertung A/D - Wandler

Die **Abmessungen** (Tabelle 3.7) orientierten sich an dem größten Wert aller Boards. Die kleineren wurden diesem Wert entsprechend skaliert bepunktet. Je größer der Wert desto weniger Punkte wurden vergeben. Liegt ein

Testobjekt mit seinen Abmessugen z.B. bei einer Länge von 40 mm und einer Breite von 50 mm, so erhält es 90 Punkte für die Länge und 80 Punkte für die Breite.

Maße in mm	Punkte
15	100
30,5	90
46	80
61,5	70
77	60
92,5	50
108	40
123,5	30
139	20
154,5	10
170	0

Tabelle 3.7: Wertung der Abmessungen

Beim **Preis** (Tabelle 3.8) verhält sich dies ähnlich, Boards unter 100 Euro verdienten die vollen 100 Punkte. Kostet ein Board zwischen 100 und 200 Euro, gibt es nur noch 90 Punkte usw. Preise jehnseits der 1000 Euro rechtfertigen Null Punkte.

Preis in Euro	Punkte
100	100
200	90
300	80
400	70
500	60
600	50
700	40
800	30
900	20
1000	10
ab 1000	0

Tabelle 3.8: Wertung des Preises

3.3 Ergebnis des Benchmark

Anhand der Bewertungsregeln in Abschnitt 3.2 wurden die Embedded Systeme nun bewertet. Die erreichten Punktzahlen liegen im Bereich von 2450 bis 7740 (Abb. 3.1). Der Testsieger mit 7740 erreichten Punkten ist das Modell GESBC-9315 der Firma Glomation .

Es gibt nicht viele Boards, die die Anforderungen an eine eingebettete Kommunikationsschnittstelle erfüllen. Einige verfügen zwar über integrierte ADUs lassen aber gleichzeitig einen CAN-Controller vermissen.

Andere Testkandidaten verfügen zwar über alle gewünschten Schnittstellen und Eigenschaften haben aber einen sehr hohen Preis. Wieder andere stimmen zwar vom Preis aber unterstützen kein Betriebssystem.

Die Ausstattung mit seriellen Schnittstellen und USB war im Allgemeinen immer gegeben. Auch eine Ethernet Schnittstelle ist auf den meisten Boards integriert. Auf manchen sogar in zweifacher Ausführung. Fast alle unterstützen eine Geschwindigkeit von 100 MBit/s einige auch bis zu 1 GBit/s.

Vielfältige Werte sind auch bei den Abmessungen zu vermerken. Der Kleinste Wert belief sich auf gute 15 mm. Nach oben gingen die Werte bis zu 170 mm.

Letzten Endes galt es einen Kompromiss zwischen Ausstattung, Größe und Preis zu finden.

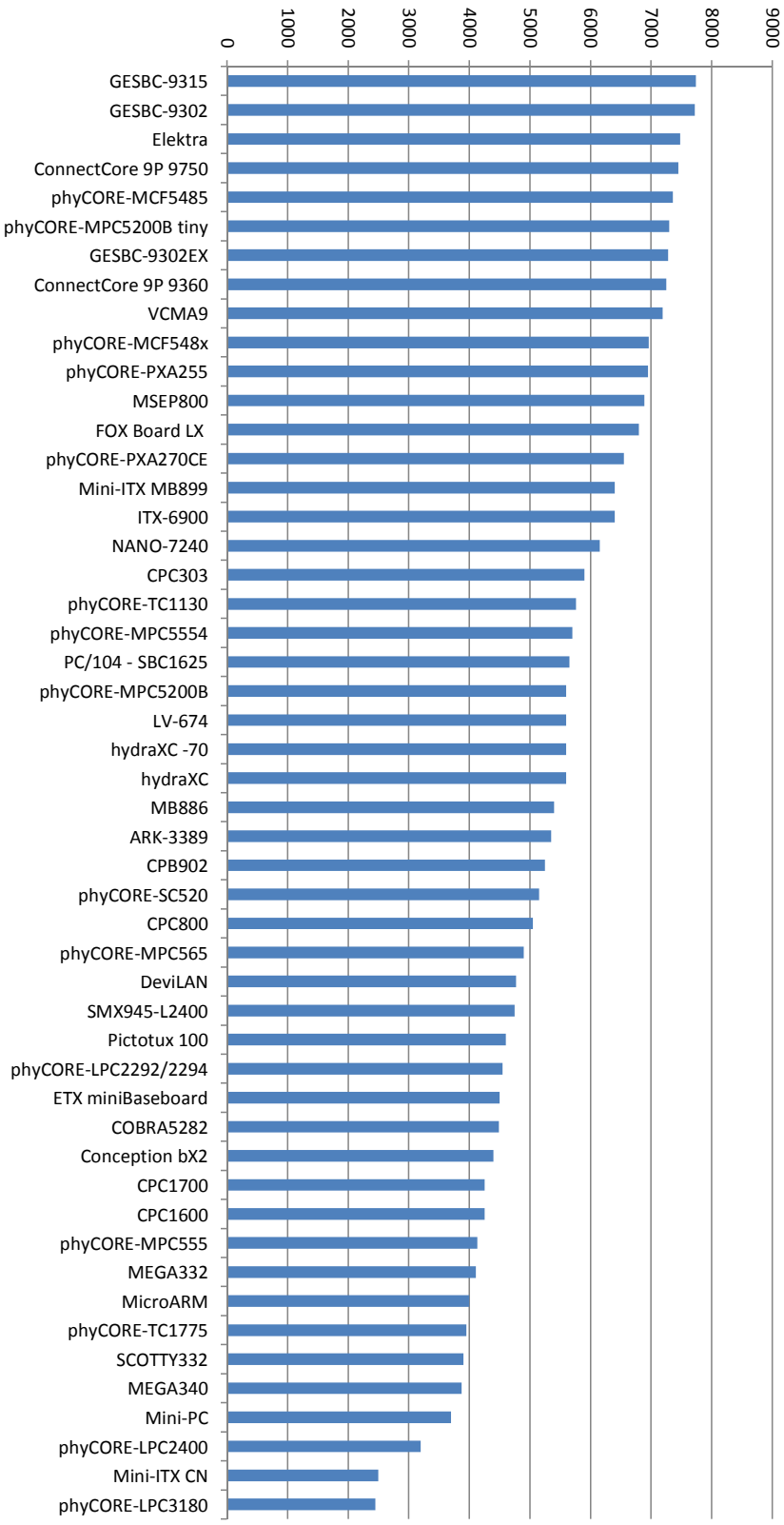

Abbildung 3.1: Vergleich der untersuchten Hardware [8]

3.4 Vorstellung des Spitzenmodells

Das Glomation GESBC-9315 (Abb. 3.2) erfüllt die Anforderungen an eine eingebettete Kommunikationsschnittstelle am ehesten. Es verfügt über einen 200 Megahertz Cirrus Logic EP-9315 Acorn Risc Machine (ARM) Prozessor mit mathematischem Co-Prozessor. Einen Arbeitsspeicher in Höhe von 64 Megabyte Synchronous Dynamic Random Access Memory (SDRAM) und bis zu 32 Megabyte Intel Flashspeicher. Für die Datenspeicherung steht eine Integrated Drive Elektronics (IDE) Schnittstelle mit CompactFlash-Card (CF) Unterstützung zur Verfügung.

Abbildung 3.2: Glomation GESBC-9315 [6]

Es besteht die Möglichkeit Windows CE oder Linux auf dem Board zu betreiben. Der Betrieb von Windows wird durch ein Windows CE Board Support Package (BSP)[1] ermöglicht. Standardmäßig wird eine ARM basierte Debian Linux Distribution mitgeliefert.

Zur Kommunikation besitzt es eine Ethernet Schnittstelle mit einer Übertragungsrate von 100 Megabit pro Sekunde. Zur Einbindung in einen Feldbus steht ein CAN-Controller zur Verfügung. Weitere Geräte können an einen

[1] Ein BSP ist eine Hardware-Abstraktionsschicht, die zwischen Hardware und Betriebssystem liegt. Sie stellt dem Betriebssystem eine hardwareunabhängige Plattform zur Verfügung.

der drei USB-Ports vom Typ A angeschlossen werden. Neben den USB-Ports sind noch andere, mit drei UARTs genügend, serielle Schnittstellen vorhanden.

Die analoge Eingabe findet mit einem auf dem Embedded Board integrierten Analog-Digital-Wandler statt. Der ADU besitzt acht Kanäle bei einer Auflösung von 12 Bit und einer Wandlungsgeschwindigkeit von 52 KSPS. Optional kann ein Digital-Analog-Wandler mit 4 Kanälen ausgewählt werden. Die Anzahl digitaler Eingänge beläuft sich auf 16. Auf ihnen können zudem Interrupts ausgelöst werden. Weiterhin sind zwei interruptfähige 16 Bit Timer, ein 32 Bit Timer und ein 40 Bit debug Timer vorhanden

Eine grafische Ausgabe ist über den integrierten LCD-Controller, einer Standard VGA Schnittstelle oder mittels S-Video auf einem Fernseher möglich. Außerdem befinden sich Audio Ein- und Ausgänge onboard. Zur Erweiterung des Embedded Board stehen ein PC/104 Bus und eine PCMCIA-Schnittstelle zur Verfügung.

Mit Abmessungen von 106 x 158 mm gehört das GESBC-9315 nicht gerade zu den kleinen im Test. Die Leistungsaufnahme des Boards liegt mit 5V bei 1A bzw. 12V und 1A bei Anschluss einer Festplatte im, für diese Leistungsklasse, normalen Bereich. Der Vorteil dieser Embedded Plattform liegt im niedrigen Preis[2].

[2] $150 (Stand 18.September 2007)

4 Die CORBA basierte Service-Architektur

Den Servicerobotern am Fraunhofer IFF liegt ein einheitliches Softwarekonzept zu Grunde. Aus informationstechnischer Sicht stellen die Roboter ein verteiltes System dar. Die Kommunikation des verteilten Systems baut auf dem Observer-Observable Verhaltensmuster [5] auf. Realisiert wird das Konzept mit Hilfe von CORBA (siehe 2.2.2).

4.1 Service Schnittstellen

Services sind die in einem verteilten System angebotenen Dienste. Über ihre Schnittstellen zum CORBA System findet der Austausch von Daten, Systemparametern und Steuerbefehlen zwischen Observer und Observable statt. Jeder Service implementiert die gleichen Schnittstellen.

Ein Service implementiert die Funktionen `get()`, `set()`, `ctrl()` und `info()` (Abb. 4.1). Es gibt Methoden zur Übertragung von Steuerdaten und Methoden zur Übertragung von Messdaten. Die `ctrl()` Methode übermittelt Systemparameter und Steuerbefehle. Zur Übertragung von Messdaten dienen `get()` und `set()`. Wobei `get()` zum Abholen von Daten und `set()` zum Setzen verwendet wird. Mit Hilfe von `info()` können die Eigenschaften und Funktionen einer Komponente abgefragt werden.

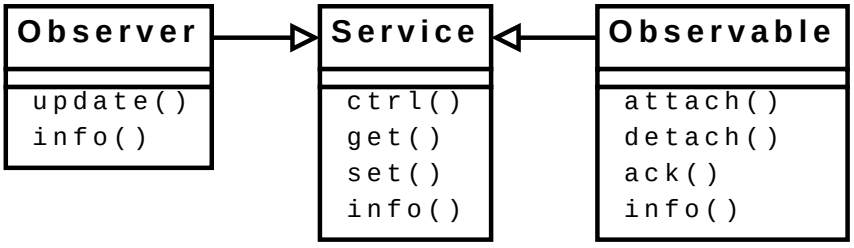

Abbildung 4.1: Klassendiagramm Service Schnittstellen

Die Parameter der Methoden sind allgemein definiert. So stellt beispielsweise der jeweilige erste Parameter, `what:string`, einer Methode die Beschreibung eines Dienstes dar. Ein Beispiel hierzu wäre die Bezeichnung `"cam"` um einen Kameraservice anzusprechen.

Die Steuerungsmethoden `ctrl()` und `info()` verfügen neben dem Dienstbezeichner über weitere Parameter die der näheren Beschreibung eines Dienstes dienen. Mit der Steuerungsmethode `ctrl()` (Listing 4.1) können bestimmte Systemzustände einer Komponente gesetzt werden. Beispielsweise könnte mit `ctrl("cam","AF_ON")` der Autofokus einer Kamera angeschaltet und mit `ctrl("cam","AF_OFF")` wieder ausgeschaltet werden. Mit `info("cam","AF")` (Listing 4.2) kann dann der aktuelle Zustand des Autofokus abgefragt werden.

```
long ctrl(in string what, in any param,
          out any result);
```

Listing 4.1: ctrl()

```
long info(in string what, in any param,
          out any result);
```

Listing 4.2: info()

Eine erhöhte Aufmerksamkeit soll hier den Methoden zum Austausch der Messdaten, `get()`(Listing 4.3) und `set()` (Listing 4.4), obliegen. Beide Methoden besitzen die gleichen Parameter. Neben dem Dienstbezeichner `what:string`, werden auch die angeforderten Daten mit `paramsXML:string` beschrieben. Hier werden die Daten mit ihren Zeitstempeln beschrieben. Den Zeitstempeln können dann Daten aus dem Puffer zugeordnet werden.

```
long get(in string what, in string paramsXML,
         out string dataXML, out OctetSeq dataBin);
```

Listing 4.3: get()

```
long set(in string what, in string paramsXML,
         in string dataXML, in OctetSeq dataBin);
```

Listing 4.4: set()

Der Parameter `dataXML:string` enthält Metadaten im XML Format. Hier werden u.a. informationen über Datenformat und Zeitstempel angegeben. Bei einem Kameraservice werden also Informationen über ein Bild z.B. zum Datentyp (Jpeg oder Bitmap), der Dateigröße und Zeitpunkt der Bilderstellung sowie Kameraeinstellungen, wie z.B. das Zoomlevel, übermittelt. Die eigentlichen binären Messdaten können mit diesem Parameter in XML eingebettet und übertragen werden. Bei größeren Daten jedoch, wie etwa Bilder eines Kameraservers, wird die Einbindung der binären Daten in XML ineffizient. Deshalb kann auch auf eine Übertragung der binären Daten ausgewichen werden. Dazu dient der vierte Parameter `dataBin:OctetSeq`.

4.2 Schnittstellen von Observer und Observable

Das Observer-Observable Modell (Abb. 4.2) geht von einer zweiseitigen Kommunikation aus. Eine Seite stellt den Observer dar. Er überwacht einen Observable und wird benachrichtigt, wenn an diesem neue Daten vorliegen. Die andere Seite bildet der Observable. Er benachrichtigt einen Observer, wenn für diesen neue Daten vorliegen.

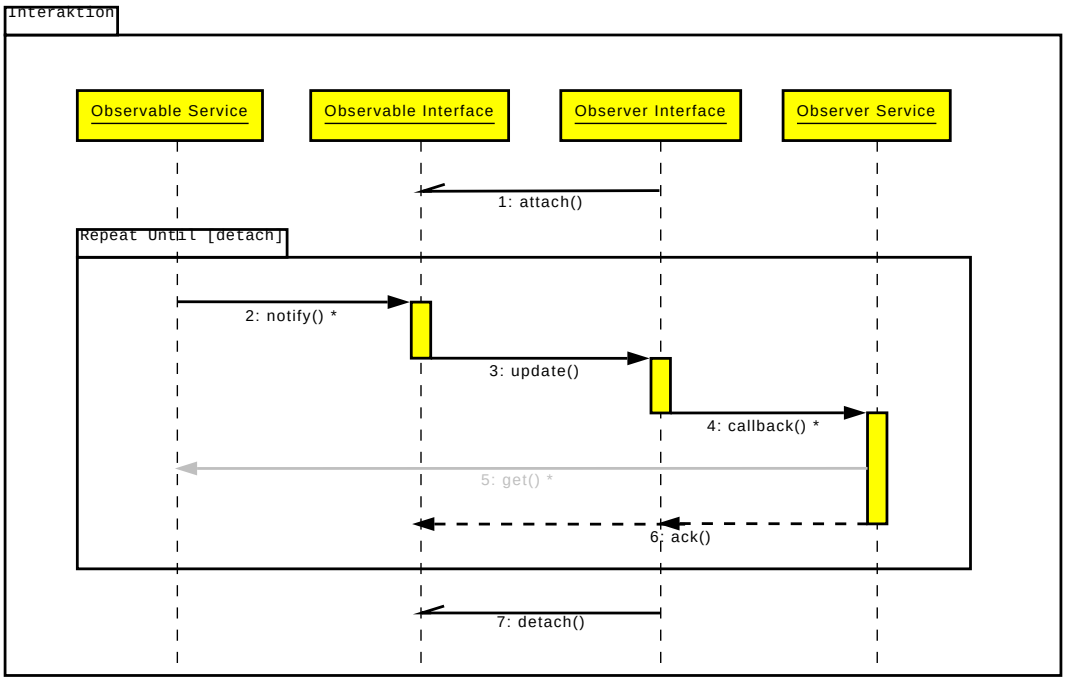

Abbildung 4.2: Sequenzdiagramm Observer - Observable Modell

Mit `attach()` (Listing 4.5) meldet sich ein Observer bei einem Observable an und signalisiert diesem, dass er benachrichtigt werden möchte, sobald neue Daten vorliegen. Er übermittelt dem Observable seinen Namen, für welche Daten er sich interessiert und wieviele Daten sich maximal beim Observable ansammeln dürfen.

```
long attach(in string observer, in string what,
            in long maxAsync);
```

Listing 4.5: attach()

Liegen neue Daten vor, gibt der Observable Service seinem Observable Interface mittels `notify()`[1] bekannt, dass ein neuer Datensatz vorliegt. Das Observable Interface benachrichtigt alle abonnierenden Observer Interfaces via `update()` (Listing 4.6) von dem neuen Datensatz. Er sendet seinen Namen, welche neuen Daten vorliegen und eine Liste mit den Zeitstempeln der Daten.

```
long update(in string observable, in string what,
            in StringSeq timestamps);
```

Listing 4.6: update()

Das entsprechende Observer Interface unterrichtet seinen Beobachter Service über das Vorliegen neuer Daten. Dem Beobachter Service steht es frei, den vollständigen Datensatz via `get()` vom Observierten Service zu beziehen. Zurück erhält der beobachtende Service einen Datensatz, der aus den Binärdaten und Metadaten, in denen der Zeitstempel enthalten ist, besteht.

Mit `ack()` (Listing 4.7) quittiert der Observer Service die vollständige Verarbeitung der Datensätze.

```
long ack(in string observer, in string what,
         in StringSeq timestamp);
```

Listing 4.7: ack()

Die eben beschriebenen Vorgänge wiederholen sich solange, bis sich der Observer mit `detach()` (Listing 4.8) wieder vom Observable abmeldet und signalisiert ihm damit, dass er nicht mehr über neue Daten benachrichtigt werden will.

[1] notify() ist CORBA intern

```
long detach(in string observer, in string what);
```

Listing 4.8: detach()

5 Kommunikationsszenarien

In diesem Kapitel wird der Frage nachgegangen wie eine Embedded Plattform in eine Observer-Observable-Struktur eingebunden werden kann. Zunächst wird betrachtet, wie sich die Struktur ohne Embedded System verhält. Soll nun Observer oder Observable auf einer Embedded Plattform betrieben werden, ist dies nicht ohne weitere Maßnahmen möglich.

5.1 Exklusive Embedded

Die Kommunikation zwischen Observer und Observable auf jeweils leistungsstarken Plattformen (Abb. 5.1), stellt den bisherigen Einsatzfall in den Robotersystemen des Fraunhofer IFF dar. Es besteht die Möglichkeit das komplette CORBA-System (siehe Kapitel 4) auf den beteiligten Plattformen zu betreiben. Es sind keine Anpassungen der verwendeten Software für eine Embedded Plattform notwendig. Auf allen beteiligten Kommunikationspartnern kann das gleiche Softwaresystem verwendet werden.

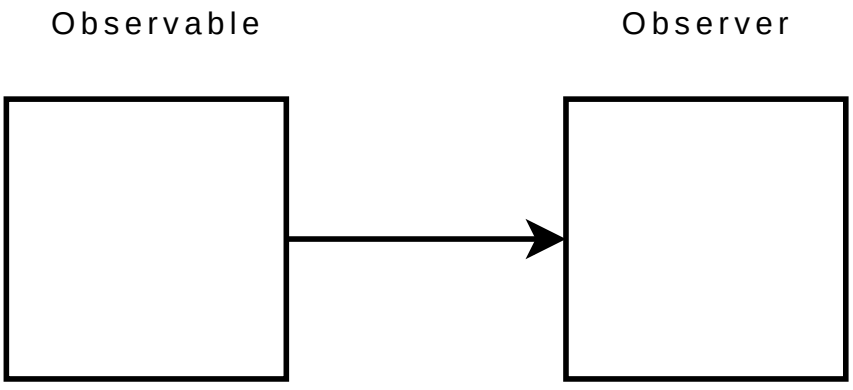

Abbildung 5.1: Leistungsstarker Observable und leistungsstarker Observer

Die Datenverarbeitung findet direkt auf den jeweiligen beteiligten Plattformen statt. Die Daten werden vom Observable gesammelt und zwischengespeichert. Hat ein Observer bestimmte Datensätze abonniert, versendet der

Observable diese an ihn. Der Observer kann die empfangenen Daten zwischenspeichern und verarbeiten.

Diese Konstellation ist allerdings für mobile Roboter sichtlich ungeeignet. Ein Robotersystem besteht aus einer Vielzahl an Observer und Observable Plattformen. Die Integration leistungsstarker Plattformen erfordert viel freien Bauraum und eine beachtliche Stromversorgung.

5.2 Embedded Observable

Um ein Embedded System in die Observer-Observable-Struktur zu integrieren, kann ein solches als Observable fungieren (Abb. 5.2). Ein Observer kann auf einer leistungsstarken Plattform z.B. in der Service Station des mobilen Roboters betrieben werden und drahtlos mit ihm kommunizieren. Für diese Konstellation muss die verwendete CORBA-Service-Architektur (siehe 4) auf das Embedded System angepasst werden.

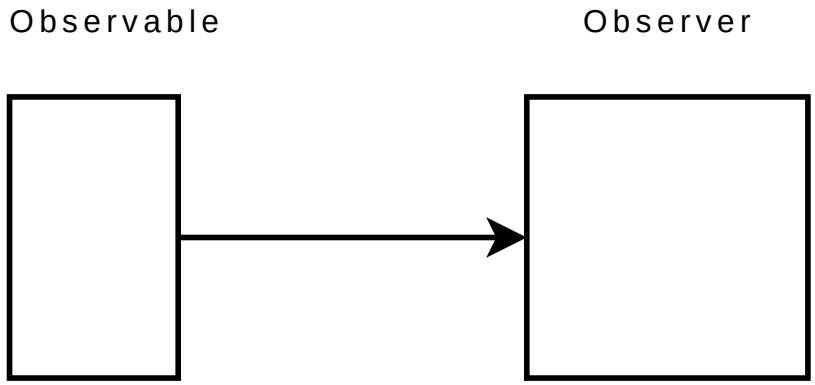

Abbildung 5.2: Embedded Observable und leistungsstarker Observer

Der Embedded Observable befasst sich lediglich mit der Datenerfassung. Die Zwischenspeicherung und die Vermittlung von abonnierten Daten erfolgt durch einen Proxy Service. Die Kommunikation zwischen Observer und Observable findet dazu über diesen Stellvertreter statt.

Ein Embedded System erhebt im Gegensatz zu einer leistungsstarken Plattform, wie in 5.1 verwendet, deutlich weniger Anspruch bezüglich Bauraum und Stromverbrauch. Auf diese Weise bewegen sich die Dimensionen eines mobilen Roboters in weitaus kleineren Rahmen. Zudem verfügt er aufgrund der geringeren Leistungsaufnahme über eine längere Akkulaufzeit.

Das Problem besteht in der Anpassung des Softwaresystems. Hier muss entschieden werden, welche Teile der Software auf der Embedded Plattform benötigt werden und welcher Teil ausgelagert werden kann.

5.3 Embedded Observer

Eine weitere Möglichkeit zur Migration von Embedded Systemen ist mit der Realisierung des Observer auf diesem gegeben (Abb. 5.3). Auf diese Weise kann ein Observable z.B. auf einer leistungsstarken Plattform in der Service Station realisiert werden. Auch in diesem Fall muss das Softwaresystem auf die Embedded Plattform angepasst werden.

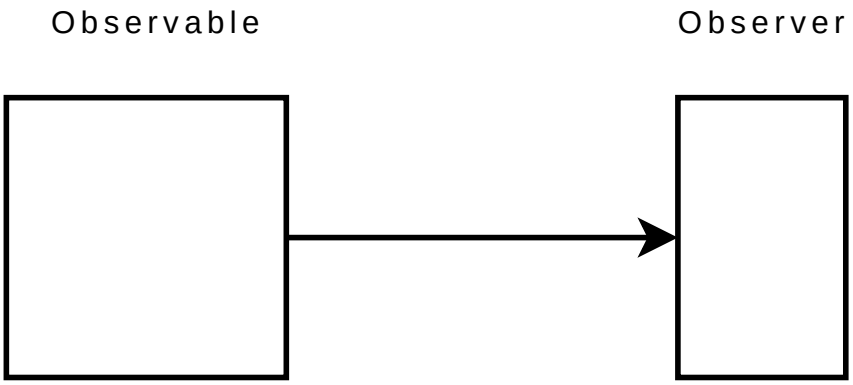

Abbildung 5.3: Leistungsstarker Observable und Embedded Observer

Der Observable sendet abonnierte Daten an den entsprechenden Observer. Bei diesem Observer handelt es sich um einen Proxy Service. Der Observer Stellvertreter speichert die erhaltenen Daten zwischen und leitet sie an den Embedded Observer weiter.

Auch hier wirken sich die Eigenschaften der Embedded Plattform positiv aus. So kann auch in diesem Fall eine höhere Akkulaufzeit erzielt werden. Zudem kann ein Roboter mit einer kleinen und leichten Bauweise aufwarten.

5.4 Exklusiv Embedded

Folgt man nun der logischen Kette zum Ende, erhält man eine Anordnung bestehend aus jeweils einer Embedded Plattform, sowohl für den Observer als auch für den Observable (Abb. 5.4). Für beide wird ein Proxy Service benötigt, der die Zwischenspeicherung und Koordination der Daten übernimmt. Die Proxy Plattformen können in der Basisstation eines Roboters untergebracht werden.

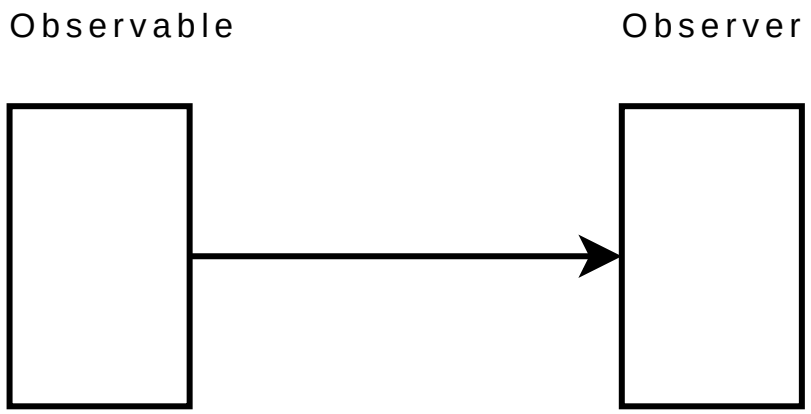

Abbildung 5.4: Embedded Observable und Embedded Observer

Der Embedded Observable sammelt Daten und schickt diese an seinen Proxy Service. Sein Stellvertreter, sendet diese an einen Observer. In diesem Fall handelt es sich erneut um einen Proxy Service. Also speichert der Observer Stellvertreter die empfangenen Daten zwischen und leitet sie an den Embedded Observer weiter.

Auch hier sind wieder ganz klar die geringen Abmessungen und der geringere Stromverbrauch der Embedded Systeme zu nennen. In Bezug auf den Bauraum, könnte alternativ auch eine Proxy Plattform erneut auf verschiedene Embedded Systeme aufgeteilt werden und in das Robotersystem eingebaut werden. So müsste der Roboter mit keiner Basis kommunizieren.

6 Anpassung des Softwaresystems

In diesem Kapitel werden die in Kapitel 5 hergeleiteten Kommunikationsszenarien näher besprochen und Konzepte einer Realisierung angegeben. Eine Umsetzung setzt eine Anpassung der CORBA basierten Service Architektur (siehe Kapitel 4) voraus. Dazu werden Teile des Observable bzw. Observer ausgelagert und auf leistungsstarken Plattformen ausgeführt.

6.1 Ohne Anpassung

Das in 5.1 beschriebene Szenario, stellt den bisher realisierten Einsatzfall im Fraunhofer IFF dar. Er geht von zwei leistungsstarken Plattformen aus (Abb. 6.1). Eine Anpassung des Softwaresystems entfällt. Die Service-Architektur aus Kapitel 4 kann direkt auf den, an der Kommunikation beteiligten Plattformen, ausgeführt werden.

Der Observable sammelt Daten und hält sie in einem Puffer vor. Leistungsstarke Rechner können auf diese Weise effizient genutzt werden. Der Datenaustausch findet wie in Kapitel 4 beschrieben statt.

6.2 Verteilung des Observable Service

Die logische Observable Einheit wird nach dem Stellvertreterprinzip [5] in zwei physikalische Teile (Abb. 6.2) eingeteilt. Einen Teil bildet eine Embedded Plattform. Der andere Teil stellt einen Proxy mit Observable Interface und Observable Service, der auf einer leistungsstarken Plattform ausgeführt wird, dar. Die Embedded Plattform dient der Datenerfassung. Sie sendet ihre gesammelten Daten direkt an den Observable Service. Der Proxy realisiert den in Kapitel 4 standardisierten Observable Service mit Puffer.

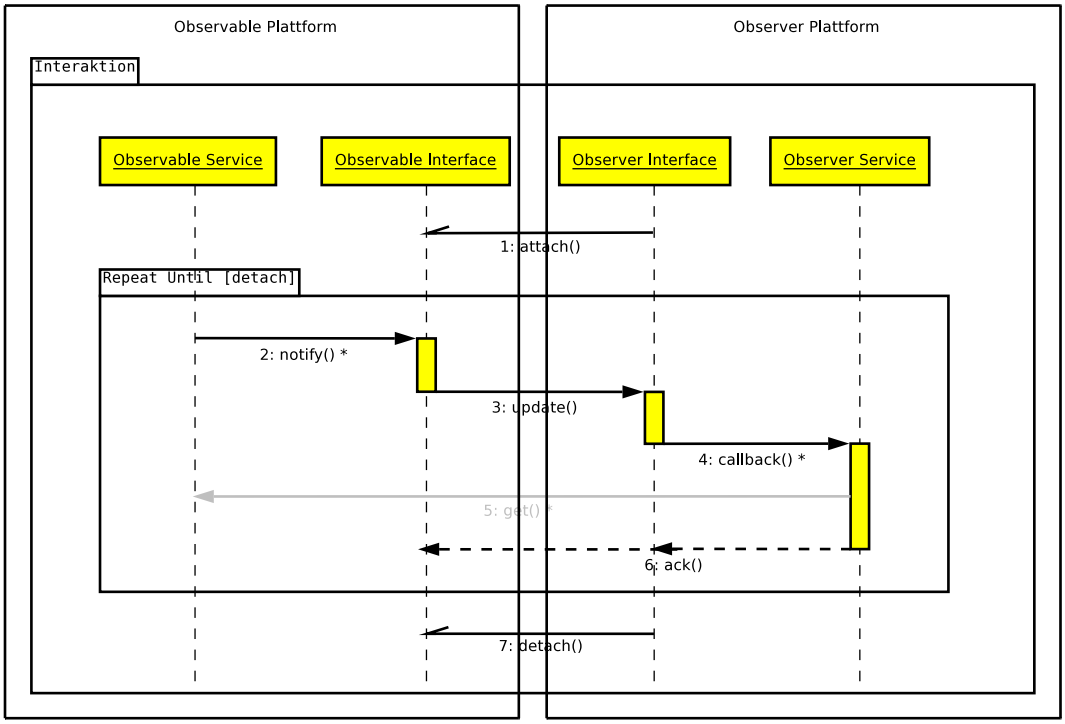

Abbildung 6.1: bisheriger Einsatzfall

Zur Kommunikation mit dem Proxy Service, werden zwei TCP (siehe 2.1.5) Kanäle verwendet. Die Entkopplung von Mess- und Steuerdaten wird nötig, da wegen hohen Datenaufkommens wichtige Steuerbefehle unnötig verzögert werden könnten. Fähigkeiten wie Flusskontrolle und Bestätigungen bei korrektem Empfang von Daten ziehen TCP einer UDP orientierten Verbindung vor.

Ein Beispiel zur Veranschaulichung ist ein Embedded Videoserver im Fraunhofer IFF. Dieser nimmt einzelne Bilder auf und komprimiert sie in das Jpeg-Format. Neben den Bilddaten bindet er zusätzliche Informationen wie Zeitstempel oder Zoomlevel zu Struct zusammen. Das fertige Struct sendet er über den Datenkanal an den Observable Proxy. Dieser bereitet die empfangenen Daten zu XML-Metadaten (siehe 4) auf.

Mit Hilfe des Steuerdatenkanals, kann der Observable Proxy Statusinformationen des Embedded Observable abrufen. Dieser wiederum nutzt den Kanal, um u.a. seine Uhr mit dem Proxy zu synchronisieren. Für eine Uhrensynchronisation auf dem Embedded Observable sind Zeitmanagement Funktionen des angepassten Softwaresystems Voraussetzung.

Da in der Regel XML-Metadaten übertragen werden, ist ein XML Parser

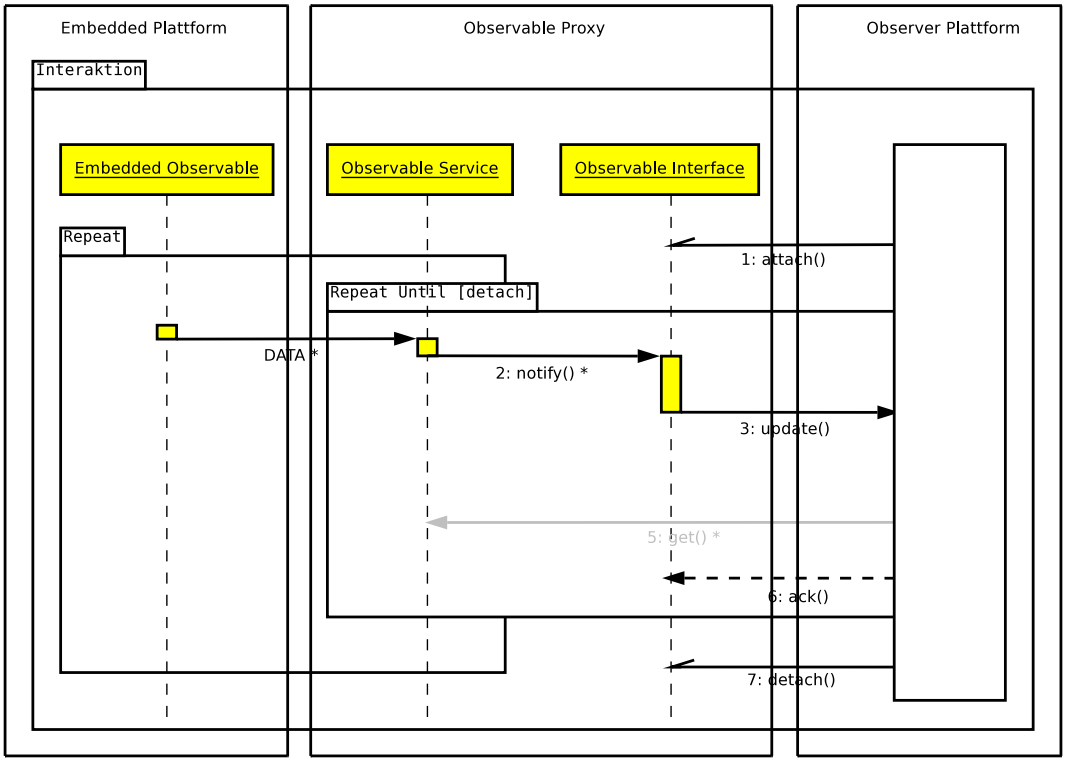

Abbildung 6.2: Observable Proxy

vorteilig. Zwingend notwendig ist dieser nicht, da die Embedded Plattform im Allgemeinen keine XML-Metadaten empfängt bzw. sendet. Um Anfragen auf dem Steuerkanal zu erkennen und zu bearbeiten, muss das Softwaresystem über einen Event Handler verfügen. Damit die Datenkompression, das Erstellen und Versenden der Metadaten gleichzeitig stattfinden kann, muss ein entsprechendes Task Management ermöglicht werden.

6.3 Verteilung des Observer Service

Analog kann zu 6.2, kann die logische Observable Einheit ebenfalls nach dem Stellvertreter Prinzip in zwei physikalische Einheiten unterteilt werden (Abb. 6.3). Der Observer Proxy ist auf einer leistungsstarken Plattform implementiert. Er ist mit der Datenaufbereitung und Koordinierung betraut. Der Embedded Server ist mit dem Empfang und der korrekten Umsetzung ausgelastet.

Der Observer Proxy bereitet die abgeholten Datensätze für den Embedded

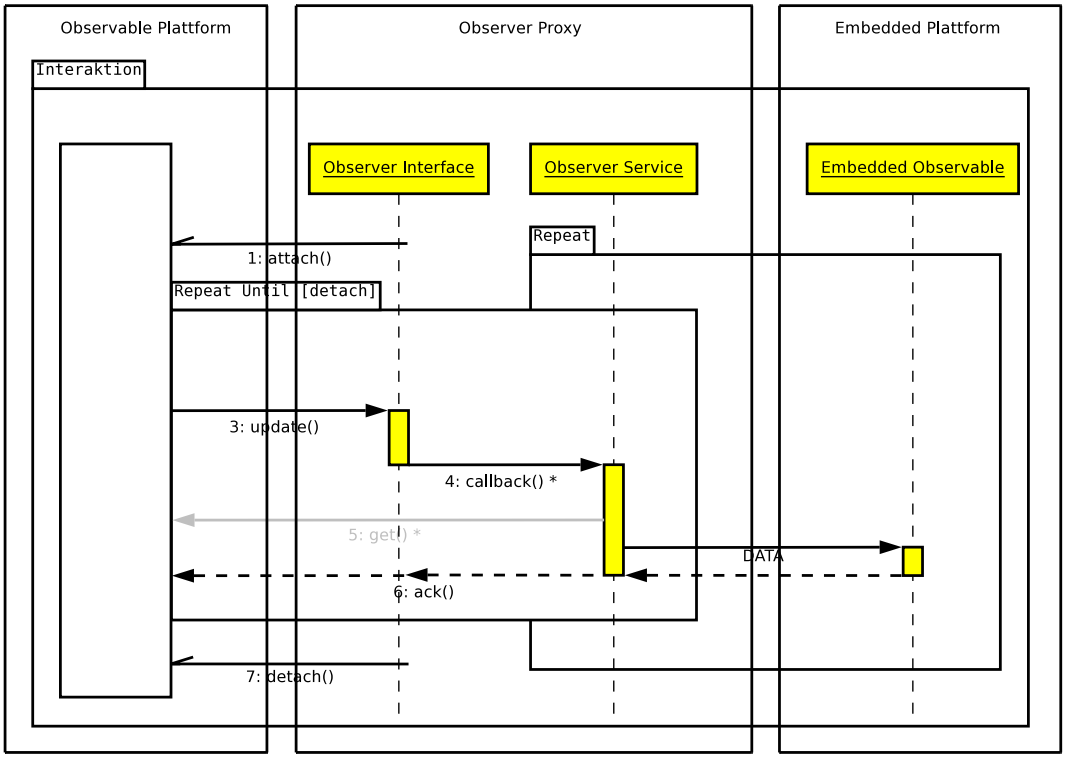

Abbildung 6.3: Observer Proxy

Observer auf. Die erhaltenen Daten müssen in eine korrekte zeitliche Reihenfolge gebracht werden. Des weiteren werden die Daten auf ihre Richtigkeit und ihren Inhalt überprüft. So können fehlerhafte Daten und Daten, die einen ungewünschten Systemzustand hervorrufen können, gefiltert werden. Außerdem trennt er die XML-Metadaten auf und sendet ein Struct mit den binären Daten an den Embedded Observer.

Dieses Szenario soll am Beispiel einer Pan-Tilt-Zoom-Kamera (PTZK) im Fraunhofer IFF veranschaulicht werden. Der Zoom der PTZK kann von einem Bedienstand aus mit dem Mausrad, über eine Benutzeroberfläche oder auch mit Knöpfen an einem Schaltpult verändert werden. Die Bedienung von Mausrad und Knöpfen kann gleichzeitig erfolgen.

Der Observer Proxy empfängt die Zoombefehle und bringt sie zunächst in eine valide zeitliche Ordnung. Dann wertet er den neuen erwünschten Zoomlevel aus und sendet einen zusammengerechneten Befehl über die Steuerleitung an die Embedded Plattform. Diese steuert den Motor der Kamera an und stellt so den Zoom ein.

Auf dem Embedded System werden auch hier wieder Zeitmanagement

Funktionen zur Synchronisation benötigt. Für eine eventuelle Verarbeitung der XML-Metadaten ist auch hier einen XML Parser von Vorteil. Ein Event Handler muss einen Interrupt auslösen um die geforderte Steuerung der Kamera auszuführen. Damit mehrere Prozesse gleichzeitig ablaufen können, müssen mehrere Threads gehändelt werden können.

6.4 Verteilung des Observer und Observable Service

Da die Softwareschnittstellen eines Embedded Systems stark Anwendungsbezogen ausgelegt sind, müssen diese manuell angepasst werden, um zwei dieser Systeme miteinander kommunizieren zu lassen. Findet die Kommunikation über einen Proxy Service statt, kann der Aufwand der manuellen Anpassung reduziert werden. Die Kommunikation über Observer und Observable Proxy (Abb. 6.4) leitet sich aus 6.2 und 6.3 ab.

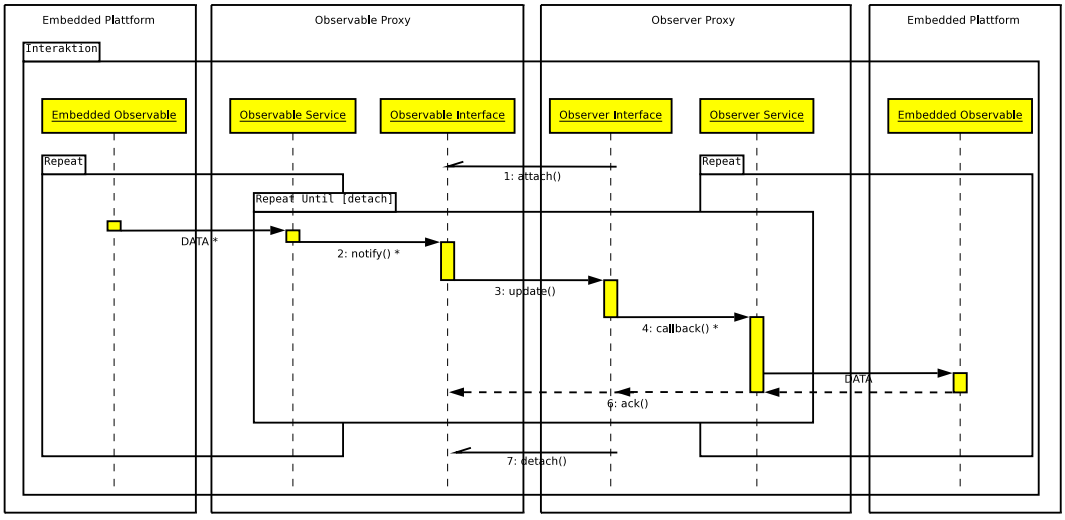

Abbildung 6.4: Observer und Observable Proxy

Die Embedded Systeme tauschen Daten mit ihren Proxy Services aus. Die Stellvertreter übernehmen die Aufbereitung zu XML-Metadaten und deren Zustellung.

6.5 Portierung

Damit die CORBA basierte Service-Architektur mit der in Kapitel 3 ausgewählten Plattform zusammenarbeitet, müssen diverse Änderungen an ihr vorgenommen werden. Standardmäßig ist auf dem GESBC-9315 der Betrieb eines Debian Linux vorgesehen. Da das vorliegende Softwaresystem unter Verwendung windowsspezifischer Funktionen implementiert wurde, müssen seine zentralen Basisklassen auf Standard C++ bzw. Posix Funktionen angepasst werden.

Portiert wurde das CORBA-Framework auf einem i386 Debian. Als Compiler diente der gcc. Die wichtigsten vorzunehmenden Veränderungen beliefen sich auf eine Anpassung spezifischer Datentypen und das Ersetzen Microsoft eigener Funktionen durch äquivalente Linux Funktionen. Dem erfolgreichen ersten Compilieren, schloss sich ein erheblicher Debugging Aufwand an.

Bei der Portierung Thread und Event behandelnder Klassen und Methoden, kam es auf Semaphoren und Mutexes an. Ein großer Unterschied zwischen Windows und Linux Funktionen, konnte bei der Prozesspriorisierung festgestellt werden. Windows verwendet hier Werte im Bereich von 1 bis 15, wobei 15 die höchste Priorität darstellt, um einem Prozess ein Priorität zu zuweisen. Unter Linux werden Prozessprioritäten im Bereich von -20 bis 20 angegeben. Prozessprioritäten im negativen Bereich können nur vom Superuser vergeben werden. Hier gilt, je niedriger der Wert, desto höher die Priorität.

Die für Zeitmessung benötigten Funtionen unterscheiden sich ebenfalls sehr stark zwischen Linux und Windows. Viele Zeitfunktionen unter Linux, basieren auf der Unix-Zeit, welche die Anzahl der vergangenen Sekunden seit dem 01.01.1970 zurückgibt. Es gibt für Windows zwar auch derartige Funktionen, jedoch wurden diese bei der Implementierung nicht verwendet. Hier wurden eher absolute Zeitfunktionen bevorzugt, so dass Umrechnungen von relativer Unix-Zeit auf die absolute Zeit nötig waren. Die Funktionen von Windows als auch Linux speichern die Uhrzeit in einem Struct. Unterschiede gibt es hier jedoch im Aufbau der Structs.

Die Anpassung des XML Parser war nicht so tief gehend wie bei den Zeitfunktionen oder dem Thread Handling. Hier musste im Prinzip auf unterschiedliche Stringfunktionalitäten zwischen beiden Betriebssystemen geachtet werden. Wesentliche Differenzen gab es in Bezug auf die Nullterminierung von Strings sowie spezieller Makros. So mussten einige Stringfunktionen selbst angepasst werden.

7 Zusammenfassung

Diese Arbeit liefert eine Herangehensweise für effizientere Robotersysteme. Durch den Einsatz von Embedded Systemen in einem Roboter, ist eine optimalere Ausnutzung des vorhandenen Bauraums gegeben.

Auf die zum Verständnis der Arbeit relevanten Grundlagen wird in Kapitel 2 eingegangen. Dort wird neben Hardwarebeschreibungen, u.a. Embedded Systeme, Feldbusse und ADUs, und wichtigen Softwarekonzepten, wie Verteilte Systeme und CORBA, auf das Bewertungsverfahren - Benchmarking - im dritten Kapitel, eingegangen.

Das dritte Kapitel beinhaltet den Prozess zur Auswahl eines geeigneten Embedded Systems. Dazu werden Anforderungen an das Produkt, spezifiziert. Zur Feststellung des Erfüllungsgrades wird ein Benchmark angewendet. Mit Abschluss des Benchmark, geht das GESBC-9315 der Firma Glomation als Testsieger hervor.

Im vierten Kapitel wird das Observer-Observable Entwurfsmuster beschrieben. Zur Veranschaulichung dient die Kommunikation zwischen CORBA Objekten in einem verteilten Robotersystem am Fraunhofer IFF.

Möglichkeiten der Integration eines Embedded Systems in ein Verteiltes System, werden im fünften Kapitel aufgezeigt. Das sechste Kapitel geht dann auf eine Implementierung der Integration ein.

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

Literaturverzeichnis

[1] Rüdiger Brause. *Betriebssysteme*. Number 3540675981. Springer, Berlin, 2., überarb. Aufl. (2004).

[2] R.C. Camp. *Benchmarking*. Number 3446176063. Hanser, 1994.

[3] G. Coulouris, J. Dollimore, and T. Kindberg. *Verteilte Systeme: Konzepte und Design (Auflage: 3)*. Number 3827370221. Pearson Studium, 2005, Auflage: 3 (15. März 2002).

[4] ELKO. das eletronik-kompendium.

[5] Erich Gamma. *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Number 3827321999. Addison-Wesley, München, Juli 2004.

[6] Glomation.

[7] Phoenix Contact (Hrsg.). *Grundkurs Sensor/ Aktor- Feldbustechnik. InterBus, Feldbustechnik nach DIN 19 528 (Vogel Fachbuch) (Gebundene Ausgabe)*, volume 2. überarb. Auflage. Vogel (1998), 1998.

[8] Jan Jurczynski. interne datenanlage. 2006.

[9] Tim Lüth. *Technische Multi-Agenten-Systeme (Verteilte autonome Roboter- und Fertigungssysteme)*. Carl Hanser Verlag München Wien, 1998.

[10] Jürgen Nehmer. *Softwaretechnik für verteilte Systeme*. Springer Verlag, 1985.

[11] Wolfgang Pfeiffer. *Digitale Meßtechnik, Grundlagen, Geräte, Bussysteme*. Number 3540639047. Springer, 1998.

[12] Kleinlein Precht, Meier. *EDV-Grundwissen*. Number 3827314224. Addison-Wesley, Auflage: 5., aktualis. u. erw. A. (1999).

[13] Bernd Reißenweber. *Feldbussysteme*. Number 3486245368. Oldenbourg Wissensch.Vlg, August 2001.

[14] Jeremy Rosenberger. *CORBA in 14 Tagen. Ihr professioneller Schritt-für-Schritt- Einstieg in CORBA (Gebundene Ausgabe)*. Number 3827220319. Markt und Technik, 1998.

[15] H. Sabisch and C. Tintelnot. *Integriertes Benchmarking für Produkte und Produktentwicklungsprozesse.* Number 3540619631. Springer, 1997.

[16] Peter Stuhlmüller. *A/D-Wandler in Embedded Systemen. Aufbau und Anwendung*. Number 3772353002. Franzis, 2004.

[17] Andrew S. Tannenbaum. *Computernetzwerke*. Number 3827370116. Addison-Wesley, 2000, Auflage: 3. Aufl. (15. November 2000).

[18] Klaus-Dieter Walter. *Messen, Steuern, Regeln mit Linux. Einsatzmöglichkeiten für Linux in Embedded Systems*. Number 3772344844. Franzis Verlag, 2001.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Magdeburg, den 24. September 2007 Jan Jurczynski