

# Ein Echtzeit-Framework für Bewegungssteuerungen

Dipl.-Phys. A. Traub<sup>\*</sup>, Fraunhofer IPA, Stuttgart  
Dipl.-Ing. C. Schaeffer, Fraunhofer IPA, Stuttgart  
Prof. Dr.-Ing. R. D. Schraft, Fraunhofer IPA, Stuttgart

*Das „Echtzeit-Framework“ ist eine Bibliothek von Software-Modulen zur Entwicklung von verteilten Echtzeit-Bewegungssteuerungen. Mit dieser Bibliothek wird die Portabilität, Wiederverwendbarkeit und Modularität von Steuerungssoftware stark verbessert und damit die Entwicklungs- und Wartungskosten des Steuerungssystems drastisch gesenkt. In diesem Beitrag werden die wesentlichen Eigenschaften des „Echtzeit-Frameworks“ vorgestellt und die praktischen Erfahrungen von einer Anwendung des Frameworks zur Steuerung eines mobilen Roboters gezeigt.*

*Das Framework beinhaltet Module zur Client-Server-Kommunikation, zur Steuerung des Programmablaufs sowie Interfaces für Low-level Input/Output. Darüber hinaus stellt es Routinen für Echtzeit-Funktionen zur Verfügung. Ein Interface für Betriebssystem-Aufrufe gewährleistet die Portabilität der darauf aufbauenden Anwendung. Mikro-Architekturen von Klassen (design patterns) wurden eingesetzt, um Lösungen für typische Probleme bei Steuerungsanwendungen zu implementieren und zu dokumentieren.*

## 1 Einleitung

Moderne Steuerungen in der Robotik und Automatisierungstechnik umfassen in der Regel eine Vielzahl unterschiedlicher Sensoren und Aktoren. Vielfach werden heute „intelligente“ Sensoren mit Microcontrollern eingesetzt, um die Datenübertragungsraten durch Vorverarbeitung der Sensordaten zu reduzieren. Ebenso können „intelligente“ Aktoren durch komfortable high-level-Befehlssätze gesteuert werden. Verteilte Bewegungssteuerungen vereinen „intelligente“ Sensoren, Aktoren und Steuerungsrechner, und erreichen so einen hohen Grad an Hardware-Modularität.

In der industriellen Steuerungsentwicklung werden jedoch noch vielfach prozedurale Programmieretechniken angewandt, mit denen die Komplexität solcher heterogen verteilter Software-Architekturen kaum beherrscht werden kann. Software, die nicht durchgängig modular aufgebaut ist, kann nur schlecht von großen Programmiererteams entwickelt werden, oder an neue Hardware-Plattformen bzw. neue funktionale Anforderungen angepasst werden. Außerdem leidet die Robustheit der prozeduralen Software unter der Schwierigkeit, einzelne Software-Komponenten unabhängig von anderen zu testen. Neben den prozeduralen Programmieretechniken besteht ein zweites großes Problem der gegenwärtigen Steuerungsentwicklung darin, dass die Kommunikationsprotokolle in verteilten Systemen abhängig vom Hersteller, vom Betriebssystem und vom jeweiligen Netzwerk oder verwendeten Bussystem stark variieren. „Plug and Play“ ist deshalb für Hersteller von Steuerungskomponenten beinahe unmöglich. Alle die oben genannten Faktoren führen zu exzessiven Kosten in der Software-Entwicklung und -Wartung.

---

<sup>\*</sup> e-mail: [traub@ipa.fhg.de](mailto:traub@ipa.fhg.de)

Um diese Kosten zu senken und gleichzeitig eine hohe Qualität (Sicherheit, Robustheit) der Software zu gewährleisten, müssen bei der Software-Entwicklung folgende Punkte angestrebt werden:

- einfache und effiziente Kommunikation zwischen unterschiedlichen Hardware-Plattformen auf Bussystemen oder in Netzwerken, wie sie in der industriellen Automation gebräuchlich sind
- Wiederverwendbarkeit von Software-Komponenten
- Portabilität zwischen Betriebssystemen und Hardware-Plattformen
- schnelle Einarbeitung neuer Teammitglieder in den Entwicklungsprozess und in bestehende Software
- Robustheit gegenüber internen Software-Fehlern und unvorhergesehenen äußeren Bedingungen
- hohe Laufzeit-Performance.

Dieser Beitrag stellt das *Echtzeit-Framework* vor, eine umfassende Bibliothek objekt-orientierter Software-Komponenten, mit der Steuerungsanwendungen aufgebaut werden können, die insbesondere die obigen Anforderungen erfüllen. Das *Echtzeit-Framework* erlaubt eine leichte und einfache Implementierung der für verteilte Bewegungssteuerungen typischen Kommunikationsprozesse. Die Hardware-Komponenten können über ein TCP/IP Netzwerk oder einen CAN [1] Feldbus verbunden werden, wie sie in industriellen Automationssystemen häufig verwendet werden. Die modulare Struktur des Frameworks erhöht die Wiederverwendbarkeit der Software und schirmt sie vor betriebssystem-abhängigen Schnittstellen ab. Außerdem enthält es Software-Entwurfsmuster [2], die die Einarbeitungszeit neuer Mitglieder im Entwicklungsteam verkürzen. Bei der Entwicklung des Frameworks wurden die Fragen der Software-Robustheit und der Echtzeit-Performance umfassend berücksichtigt.

Im 2. Abschnitt des Beitrags wird der Stand der Technik zu diesem Thema vorgestellt. Abschnitt 3 beschreibt die Gesamtarchitektur einer mit dem *Echtzeit-Framework* erstellten Anwendung und Abschnitt 4 beschreibt die Komponenten des *Echtzeit-Frameworks* im einzelnen. Nach der Diskussion der Implementierung in Abschnitt 5 beschreiben wir in Abschnitt 6 eine Beispielanwendung des Frameworks für eine mobile Robotersteuerung. Schlussfolgerungen sind in Abschnitt 7 zusammengefasst.

## 2 Stand der Technik

Für Steuerungen sind schon einige Frameworks entwickelt worden, darunter NEXUS [3] und OSACA [4]. Sie sind dem in diesem Beitrag vorgestellten *Echtzeit-Framework* ähnlich, insofern sie Client-Server-Architekturen für die Kommunikation benutzen. NEXUS ist ein dezentrales Kommunikationssystem mit hierarchischer Fehlererkennung zur Erhöhung der Robustheit. Es ist zwar für mobile Roboter entwickelt worden, ist aber allgemein genug, um auch in anderen Steuerungen eingesetzt zu werden. OSACA ist ein Kommunikations-Framework für Bewegungssteuerungen unter Windows NT und VxWorks. Sowohl NEXUS als auch OSACA beschränken sich jedoch auf TCP/IP Protokolle und können nicht direkt zur Kommunikation über industrielle Feldbussysteme genutzt werden, wie sie häufig in der Automation verwendet werden, z.B. CAN, [1], Interbus [10] und Profibus [11].

Kommerziell verfügbare Produkte sind ControlShell™ [5] und Rhapsody™ [6]. Sie verwenden grafische Editoren für Datenfluss- und Zustandsdiagramme, aus denen Quellcode generiert werden kann. Automatisch erzeugter Quellcode ist jedoch oft schwer verständlich. Deshalb wurde diese Vorgehensweise nicht für das *Echtzeit-Framework* übernommen. ControlShell™ hat die Besonderheit, dass es keine Client-Server-Architektur zur Kommunikation benutzt, sondern eine Kommunikation gleichberechtigter Partner erlaubt, genannt NDDS. Wie NEXUS und OSACA können ControlShell™ [5] und Rhapsody™ nicht zur Kommunikation über industrielle Feldbussysteme genutzt werden.

ACE [7] ist ein allgemeines Kommunikations-Framework für Client-Server-Architekturen, wie sie hauptsächlich in TCP/IP Netzwerken konzentriert sind. Es ist als Quellcode für verschiedene Windows- und UNIX/POSIX-Plattformen frei zugänglich und gut dokumentiert. Unter den in diesem Abschnitt vorgestellten Frameworks benutzt es als einziges explizit Software-Entwurfsmuster. Im Gegensatz zum *Echtzeit-Framework* enthält es jedoch keine komfortablen Klassen für die Art von high-level Kommunikation, wie sie in Bewegungssteuerungen typisch ist.

Einige Organisationen haben Kommunikationsstandards für spezifische Feldbussysteme entwickelt, darunter CANOpen [1], DeviceNET [8, 9], Interbus [10] und Profibus [11]. Ein vielversprechender neuer Standard ist „OLE for Process Control“ (OPC) (OLE = Object Linking and Embedding) [12]. Kommerzielle OPC Client-Server-Software ist für eine Vielzahl von Netzwerken und Feldbussystemen erhältlich. Der große Nachteil von OPC ist, dass es speziell für Microsoft-Betriebssysteme entworfen wurde. Das Windows DCOM Protokoll, auf dem OPC aufbaut, wurde bisher jedoch nicht einmal in Windows CE [13] implementiert, Microsoft's einzigem Echtzeit-Betriebssystem.

Obwohl nicht speziell für Produktionsanlagen entworfen, gewinnt die „Common Object Request Broker Architecture“ (CORBA) an Bedeutung in der industriellen Automation. Eine Echtzeitversion von CORBA basierend auf ACE [7] ist frei verfügbar.

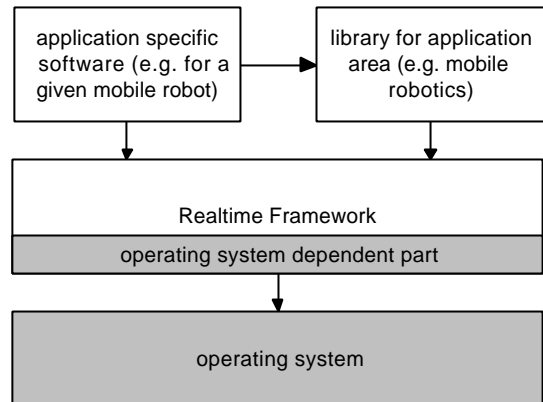
Solange diese Aufgaben zur Protokollstandardisierung nicht gelöst sind, haben die Hersteller von verteilten Bewegungssteuerungen nur die Wahl zwischen der Entscheidung für einen spezifischen Kommunikationsstandard oder der Einführung einer zusätzlichen Abstraktionsschicht in ihrer Software, die an verschiedene Protokolle angepasst werden kann. Der letztgenannte Ansatz wurde für das *Echtzeit-Framework* gewählt. Doch in den folgenden Abschnitten wird sich zeigen, dass die Funktionalität des *Echtzeit-Frameworks* weit über die von Protokollen wie OPC oder CORBA hinausgeht, da es auch Mechanismen für die sehr effiziente Client-Server-Kommunikation zwischen Prozessen und Threads (Tasks) auf derselben Maschine beinhaltet, Entwurfsmuster zur Integration wiederverwendbarer Komponenten in Steuerungsanwendungen und anderes mehr.

### **3 Architektur einer Anwendung des *Echtzeit-Frameworks***

Das *Echtzeit-Framework* ist als eine Schicht von Komponenten entworfen worden, auf der die Echtzeitanwendungen und andere Bibliotheken für spezifischere Gebiete aufgebaut werden können. (Abb. 1). Gegenwärtig ist eine Bibliothek für mobile Roboter und Industrieroboter basierend auf dem *Echtzeit-Framework* verfügbar, die Komponenten wie Bewegungsplanung, Lageregelung und Antriebsansteuerung enthält. Mit dieser Architektur

kann der Umfang des applikationsspezifischen Codes durch Nutzung des Frameworks und anderer Bibliotheken klein gehalten werden.

Abb. 1: Architektur einer typischen *Echtzeit-Framework* Anwendung. Pfeile symbolisieren eine „verwendet“-Beziehung. Betriebssystemabhängige Teile sind grau hinterlegt.



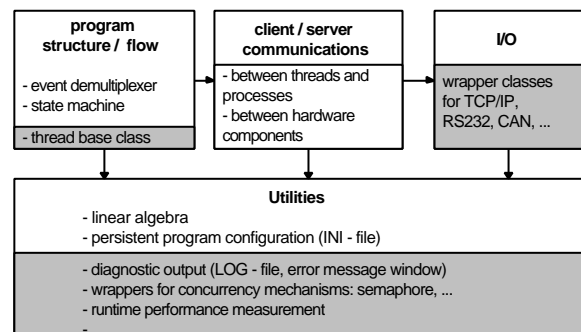
Ein weiterer Vorteil dieser Architektur ist, dass der vom Betriebssystem abhängige Code auf einer dünnen Schicht des *Echtzeit-Frameworks* konzentriert ist. Dies vereinfacht die Übertragung der Anwendung in ein neues Betriebssystem erheblich. Leistungseinbußen durch diese Extrachicht wurden mit Hilfe von C++ inline-Funktionen für Betriebssystemaufrufe klein gehalten.

#### 4 Komponenten des *Echtzeit-Frameworks*

Wie in Abb. 2 zu sehen, ist das *Echtzeit-Framework* in verschiedene Pakete von Klassen aufgeteilt. Die Pfeile bedeuten „verwendet“, was auch bedeutet „ist abhängig von“. Zirkulare Abhängigkeiten konnten vermieden werden, so dass zum Beispiel das I/O-Paket und das Utility-Paket ohne den Client-Server und das Programmstruktur / Programmfluss-Paket getestet und genutzt werden kann.

Das I/O-Paket liefert vor allem eine Schnittstelle zu betriebssystemabhängigen I/O-Operationen für TCP/IP, serielle Kommunikation und CAN Feldbus. Low-level Funktionen wie Diagnoseausgabe sind im Utility-Paket konzentriert. Wie bereits erwähnt, ist ein Hauptproblem in der derzeitigen Entwicklung verteilter Bewegungssteuerungen die Kommunikation zwischen verschiedenen Hardware- und Software-Einheiten. Die Client-Server-Bibliothek umfasst die wichtigsten Arten der Client-Server Kommunikation, wie Daten- oder Befehlsaustausch und Ereignismeldungen.

Abb. 2: Struktur des *Echtzeit-Frameworks*. Die Framework-Klassen sind in 4 Pakete aufgeteilt. Pfeile symbolisieren eine „verwendet“-Beziehung. Betriebssystemabhängige Teile sind grau hinterlegt.



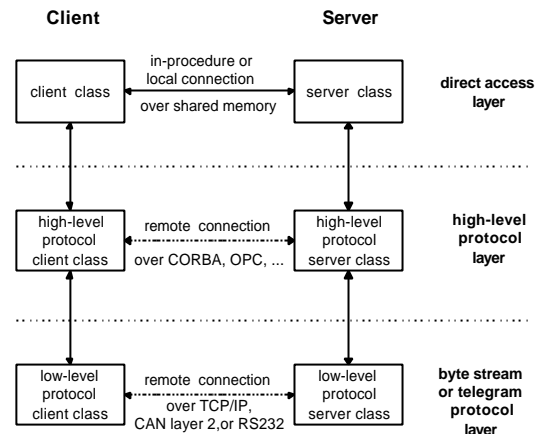
Zusätzlich wurde ein „Remote Procedure Call“-Mechanismus (RPC) entwickelt. Die Kommunikation kann entweder synchron (blockierend) oder asynchron (nicht-blockierend) erfolgen. Jeder Nachricht kann eine Priorität zugeordnet werden.

Unabhängig von dem verwendeten Übertragungsprotokoll (siehe Abb. 3) wird dem Benutzer der Client-Server-Klassen eine einfache und flexible Schnittstelle zur Verfügung gestellt. Zum Beispiel kann der Typ von Daten, die zwischen einem Client und einem Server übermittelt

werden, durch einen C++Template-Parameter spezifiziert werden. Sowohl einfache Datentypen wie Integer als auch komplexe Typen (Strukturen) sind erlaubt. Ebenso können Felder (Arrays) von einfachen und komplexen Datentypen erzeugt werden. Die Client-Server-Bibliothek erlaubt auf diese Art die Trennung der übertragenen Daten von der spezifischen Art des verwendeten Kommunikationsprotokolls; dies ist ein großer Schritt hin zu mehr Software-Modularität und Wiederverwendbarkeit.

Abb. 3: Schichten der Client-Server-Kommunikation. Wenn die Client-Server Verbindung zwischen zwei Threads (tasks) desselben Prozesses (in-process) oder zwischen zwei Prozessen auf einem Computer (lokal) besteht, kann die Kommunikation unter Verwendung des gemeinsamen Speichers in der obersten Schicht stattfinden. Ansonsten laufen die Daten über die mittlere Kommunikationsschicht, die über high-level Protokolle wie OPC oder CANOpen Fernverbindungen unterhält, oder auf die untere Schicht, die Kommunikation über Bytestreams oder Telegrammprotokolle umfasst.

Weil die in-process Kommunikation über einen gemeinsamen Speicher realisiert wurde, gibt es dabei nur einen sehr kleinen Performanceverlust. In Abschnitt 6, Abb. 8 findet sich ein Beispiel zur Verwendung der Client-Server-Klassen.



Die Fluktuation im Entwicklungsteam ist ein anderes typisches Problem bei der Entwicklung von Steuerungssoftware, das oft unterschätzt wird. Moderne Steuerungen sind so komplex, dass es oft sehr schwierig ist, Module und Anwendungen zu verstehen, die von jemand anderem entwickelt wurden. Dies führt zu dem allseits bekannten Phänomen, dass Software-Komponenten eher erneut erfunden als wieder verwendet werden. Neben einer guten Dokumentation können Software-Entwurfsmuster [2] dieses Problem entschärfen. Entwurfsmuster beschreiben auf informelle Weise die Verwendung von mehreren zusammenhängenden Klassen. Zum Beispiel sind sie bei der Dokumentation wiederkehrender Mikro-Architekturen von Klassen hilfreich, die sich beim Aufbau von Steuerungssoftware als nützlich erwiesen haben.

Ein wichtiges Entwurfsmuster, das in Verbindung mit dem *Echtzeit-Framework* entwickelt wurde, ist der *Management Tree* (Abb. 4), der als Skelett für die meisten mit dem *Echtzeit-Framework* realisierten Anwendungen verwendet wird. In diesem Muster sind anwendungsspezifische Klassen und Klassen einer auf dem Framework aufbauenden Bibliothek, die von entsprechenden Framework-Klassen abgeleitet sind, in einer baumartigen, hierarchischen Struktur angeordnet (alle *Echtzeit-Framework*-Klassen beginnen mit RL\_ um Doppelnamen zu vermeiden). Die Pfeile in Abb. 4 symbolisieren nicht eine vererbte Beziehung, sondern eine „verwaltet“-Verknüpfung, das heißt Initialisierung, Ausführung, Reset und De-Initialisierung. Abb. 5 zeigt wie dieses Entwurfsmuster intern durch das *Echtzeit-Framework* realisiert wird.

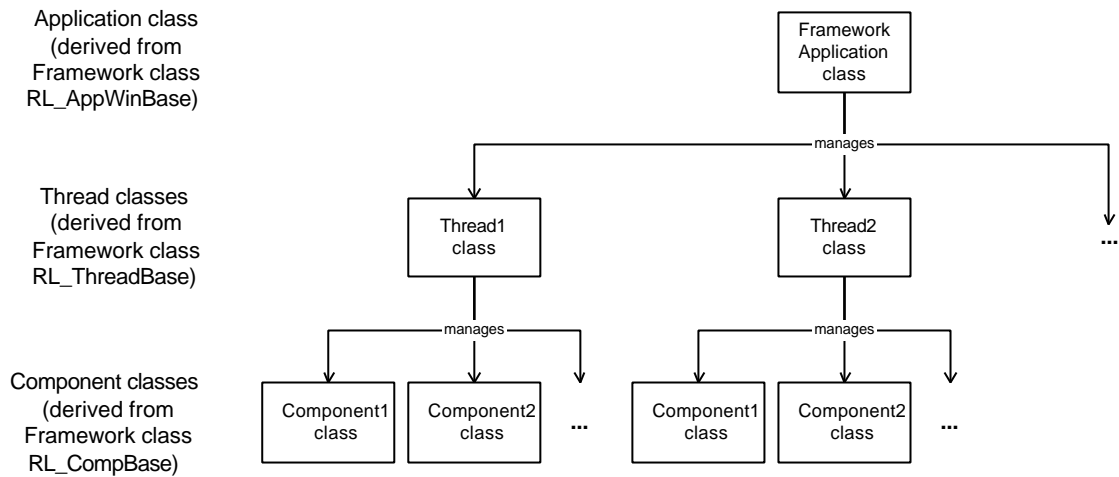


Abb. 4: *Management Tree* Entwurfsmuster. Anwendungsspezifische Klassen (Rechtecke) werden von entsprechenden Framework-Klassen abgeleitet, die die für das Management benötigte Funktionalität beinhalten.

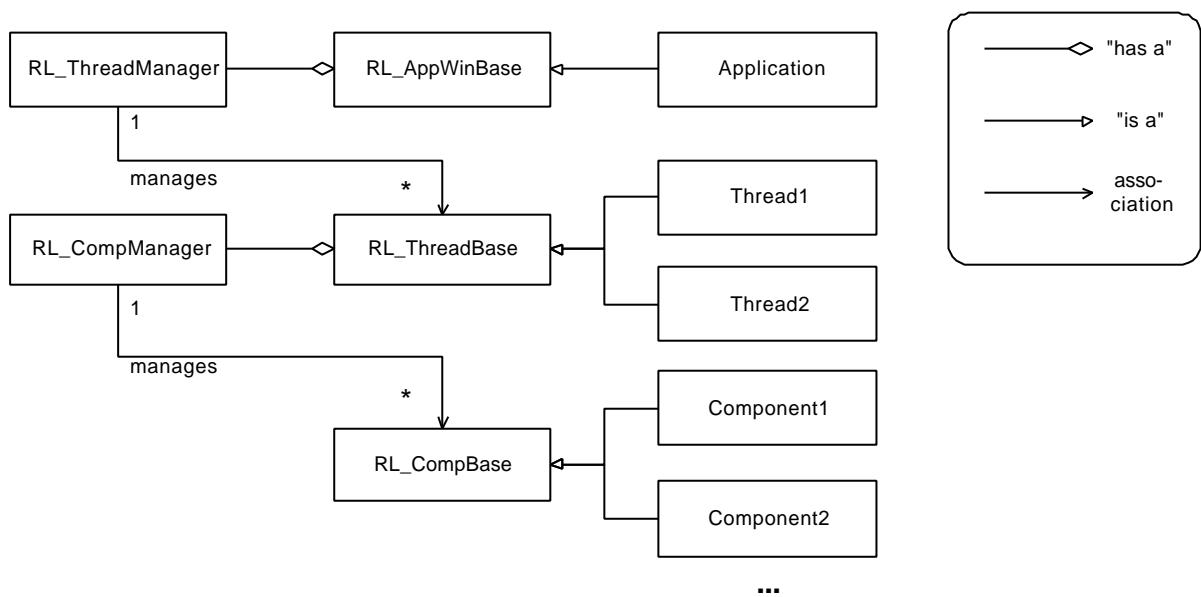


Abb. 5: Vereinfachtes UML-Diagramm der in das Entwurfsmuster *Management Tree* involvierten Klassen. Die linke und mittlere Spalte der Rechtecke sind *Echtzeit-Framework-Klassen*. Anwendungsspezifische Klassen befinden sich rechts. Die Anwendungsbasis-Klasse `RL_AppWinBase` enthält ein Objekt des Typs `RL_ThreadManager`. Damit kann man eine beliebige Anzahl von Thread Klassen (abgeleitet von der Thread-Basisklasse `RL_ThreadBase`) verwalten. Analog dazu enthält jede Thread-Klasse ein `RL_CompManager`-Objekt zur Verwaltung von Komponenten.

Eine wichtige Funktion des *Management Tree* Entwurfsmusters ist die Fehlerbehandlung. Wenn z.B. eine Komponente einen Fehler meldet, den sie nicht alleine beheben kann, wird er von der Applikations-Klasse (abgeleitet *RL\_AppWinBase*) abgefangen, die daraufhin alle anderen Threads und Komponenten zurücksetzt (Reset). Dadurch kann die Steuerung auch auf unvorhergesehene Bedingungen robust reagieren. Durch das *Management Tree* Entwurfsmuster können die wichtigsten Phasen des Programmstarts, des Resets und des Shutdown in *Echtzeit-Framework* Anwendungen, die von anderen Leuten entwickelt wurden, leichter verstanden und weiterentwickelt werden.

Neben den im Entwurfsmuster *Management Tree* verwendeten Klassen umfasst das Paket „Programmstruktur / Programmfluss“ eine konfigurierbare Zustandsmaschine und das *Reactor* Entwurfsmuster [14] mit einem Event-Demultiplexer für eine flexible, prioritätsbasierte Behandlung von gleichzeitigen Ereignissen in einem Programm-Thread.

## 5 Implementierung

Das *Echtzeit-Framework* wurde mit der Sprache Unified Modeling Language (UML) [15] entworfen und in C++ auf Windows NT und Windows CE implementiert. Während seiner Entwicklung wurde auf die Portabilität zu POSIX-konformen Betriebssystemen wie VxWorks

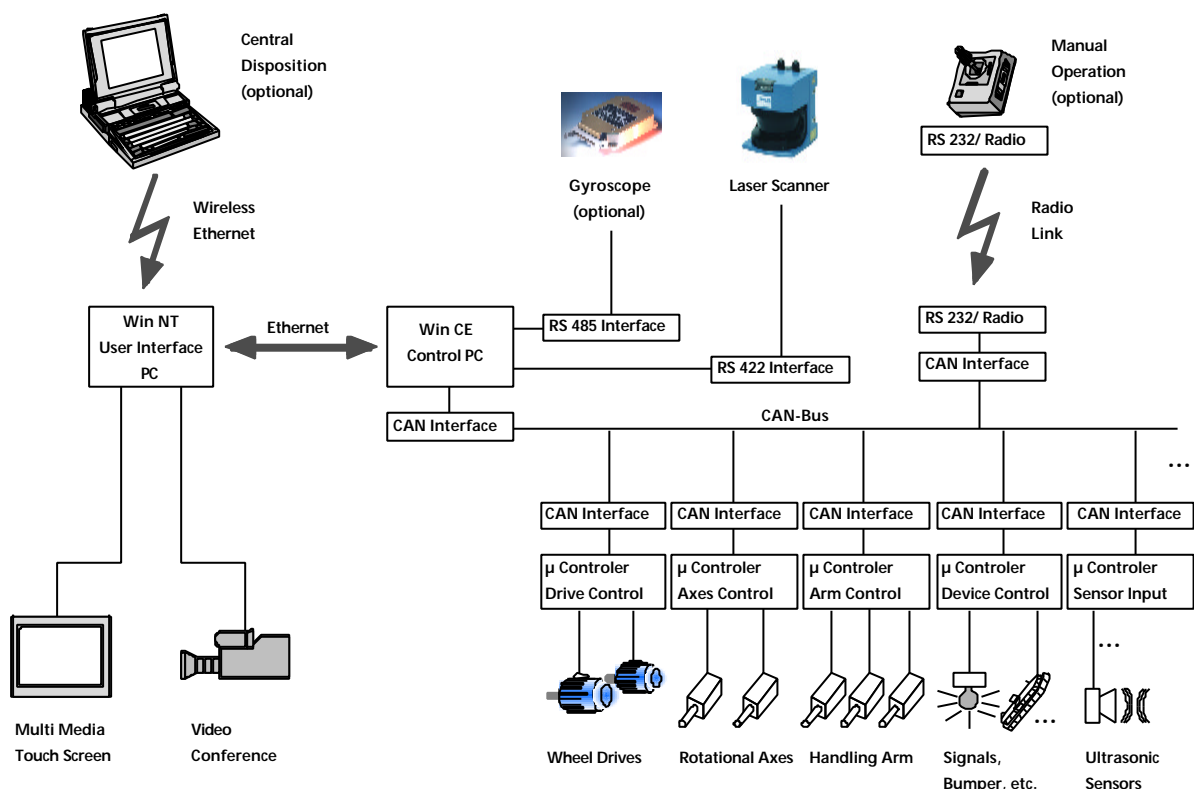


Abb. 6: Bewegungssteuerung des Care-O-bot™. Care-O-bot™ ist ein mobiles System zur häuslichen Pflege von älteren und behinderten Personen. Care-O-bot™ hat eine Zweiachsen-Kinematik und mehrere Achsen zur Objekthandhabung. Seine Bewegungssteuerung besteht aus drei Pentium PCs und mehreren Microcontrollern, die durch TCP/IP und einen CAN Feldbus verbunden sind.

Rücksicht genommen. Zum Beispiel wurde der Systemaufruf WaitForMultiple Objects() von Windows vermieden, weil er nicht von POSIX unterstützt wird.

*Echtzeit-Framework* Anwendungen werden erstellt durch statisches oder dynamisches Linken der *Echtzeit-Framework* Bibliotheken mit dem Rest der Anwendung und durch Aufruf einer Initialisierungsfunktion beim Programmstart. Die meisten *Echtzeit-Framework*-Anwendungen leiten auch anwendungsspezifische Klassen von Framework-Basisklassen ab, entsprechend dem *Management Tree* oder anderen Entwurfsmustern (siehe Abschnitt 4).

Die Echtzeitfähigkeit von Betriebssystemen wie Windows CE wird im *Echtzeit-Framework* erhalten durch Garantie deterministischer Antwortzeiten für Framework-Funktionsaufrufe. Des Weiteren werden die Laufzeit-Performance und die Speichereffizienz des Framework erhöht durch weitgehende Vermeidung von Klassen mit automatischer Typenkonversion und Speicherallokation, wie z.B. String-Klassen.

## 6 Anwendung des *Echtzeit-Frameworks* bei einem mobilen Roboter

Das *Echtzeit-Framework* wurde als Basis für die Steuerungssoftware des Care-O-bot™ (Abb. 7, [16]) verwendet, ein mobiles System zur häuslichen Pflege älterer und behinderter Menschen, das am Fraunhofer Institut IPA entwickelt wird.

Abb. 7: Care-O-bot™. Unter dem beweglichen Multimedia Touchscreen (oben) sind seitlich zwei Handgriffe für den Einsatz als Gehhilfe angebracht.

Care-O-bot™ ist mit einer Zweiradkinematik und einem Multimedia Touchscreen ausgestattet und kann als ein mobiles Kommunikationsterminal und Gehhilfe eingesetzt werden. Ein zukünftiges Modell mit einem Manipulatorarm wird in der Lage sein, einfache Hausarbeiten durchzuführen, wie z.B. Medikamente an ein Bett zu bringen. Die Steuerung des Care-O-bot™ (Abb. 6) umfasst drei PCs sowie mehrere „intelligente“ Sensoren und Aktoren, die über ein TCP/IP Netzwerk und einen CAN Feldbus miteinander verbunden sind. Das *Echtzeit-Framework* wurde als Basis der Steuerungssoftware verwendet. Es verwaltet die Kommunikation zwischen verschiedenen Hardware-Komponenten und liefert weitere Echtzeitsteuerungsfunktionalitäten für die Anwendungen, die auf den PCs laufen.

Der Benutzerschnittstellen-PC nimmt vom Benutzer Eingaben vom Touchscreen oder durch Spracherkennung an und liefert ein audiovisuelles Feedback. Über eine Client-Server-Verbindung ist der PC mit dem





Während der Entwicklung eines mobilen Roboters wurden praktische Erfahrungen mit dem *Echtzeit-Framework* gesammelt. Dabei stellte sich heraus, dass objekt-orientierte Frameworks und Entwurfsmuster die Software-Entwicklung und die Wartungskosten entscheidend reduzieren können und gleichzeitig die Softwarequalität erhöhen.

### Danksagung:

Dieser Beitrag wurde im Projekt IQ 2000 vom Bundesministerium für Forschung und Technologie, Deutschland, unterstützt.

## Literaturhinweise

- [1] CAN in Automation e.V. Home Page, <http://can-cia.de>
- [2] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software. Reading: Addison-Wesley, 1995
- [3] J. A. Fernandez, J. Gonzalez: NEXUS: A Flexible, Efficient and Robust Framework for Integrating Software Components of a Robotic System. Proceedings of the 1998 IEEE ICRA, Leuven, Belgium, 1998
- [4] OSACA Association: The OSACA Project at a Glance. <http://www.osaca.org/>
- [5] Real-Time Innovations Inc (RTI) and Stanford University: Control Shell: Object-Oriented Framework for Real-Time System Software. <http://128.102.240.17/products/cs.html>, 1996
- [6] I-Logix Inc.: Rhapsody product overview. [http://www.ilogix.com/fs\\_prod.htm](http://www.ilogix.com/fs_prod.htm)
- [7] Schmidt, D. C.: ACE: an Object-Oriented Framework for Developing distributed Applications. In: Proceedings of the 6<sup>th</sup> USENIX C++ Technical Conference, Cambridge, Massachusetts, USENIX Association, April 1994; see also: <http://www.cs.wustl.edu/~schmidt/ACE.html>
- [8] The Open DeviceNET Vendors Association, <http://208.147.96.36/mrop/Organizations>
- [9] Interesting links and information to DeviceNET. [http://www.infoside.de/infida/wissen\\_devicenet.htm](http://www.infoside.de/infida/wissen_devicenet.htm)
- [10] INTERBUS Online, <http://www.ibsclub.com/index.htm>
- [11] Profibus Home Page, <http://www.profibus.com>
- [12] OPC Foundation Home Page, <http://www.opcfoundation.org/>
- [13] Microsoft Windows CE product information, <http://www.microsoft.com/windowsce/default.asp>
- [14] Schmidt, D. C., „The Object-Oriented Design and Implementation of the Reactor: A C++ Wrapper for UNIX I/O Multiplexing (Part 2 of 2)“, C++ Report, vol. 5, Sept. 1993
- [15] G. Booch, J. Rumbaugh, I. Jacobsen, „Unified Modeling Language User Guide“, Addison Wesley, Longman, 1997
- [16] R. D. Schraft, C. Schaeffer, T. May, „The Concept of a System for Assisting Elderly or Disabled Persons in Home Environments“, Proceedings of the 24<sup>th</sup> IEEE IECON, Vol. 4 Aachen (Germany), 1998