

INTEGRATION SPEICHERPROGRAMMIERBARER STEUERUNGEN IN VERTEILTE, OBJEKTORIENTIERTE AUTOMATISIERUNGSANWENDUNGEN

R. Kröger*, M. Wack⁺, Th. Gräff, M. Gröger, M. Núñez

Fachhochschule Wiesbaden, Fachbereich Informatik
Kurt-Schumacher-Ring 18, D-65197 Wiesbaden
email: kroeger@informatik.fh-wiesbaden.de

⁺Klößner-Moeller GmbH, Bereich Automatisierungsgeräte
Hein-Moeller-Straße 7-11, D-53115 Bonn

EINLEITUNG

Dezentrale Architekturen für Automatisierungssysteme haben in den letzten Jahren eine starke Verbreitung gefunden und werden mittlerweile auch zur Steuerung einzelner Maschinen eingesetzt, wenn diese aus mehreren kooperierenden, intelligenten Aggregaten bestehen. Entsprechende dezentrale Hardware-Strukturen können heute effizient auf der Basis Feldbus-verbundener Systeme aufgebaut werden. Wiederverwendbarkeit von Teillösungen in flexiblen Produktfamilien, Offenheit bezüglich der Integration von Fremdkomponenten sowie die Fähigkeit zum Informationsaustausch mit externen (auch nicht-echtzeitfähigen) Teilsystemen sind zunehmend gefragte Eigenschaften.

Die Integration von speicherprogrammierbaren Steuerungen (SPSen) in verteilte Netzwerke gewinnt im Einklang hiermit in zunehmendem Maße an Bedeutung. Eine konsequente Dezentralisierung von Automatisierungsstrukturen entspricht dem Trend der Technik, Intelligenz dort haben zu wollen, wo diese benötigt wird. Im Bereich der Büroanwendungen ist dieser Schritt schon vollzogen worden. Aus diesen neuen Strukturen entstehen aber auch weitergehende Anforderungen an die Vernetzung, die Integration sowie an die Kommunikation von SPS-Systemen. Zum einen werden SPSen i.d.R. über Feldbusse vernetzt. Diese sind von den Hardwarestrukturen und dem Durchsatz beschränkt und eher für kleine Datenmengen der E/A-Ebene geeignet. Außerdem sind die Kommunikationsbeziehungen vom Anwender bei jedem Projekt explizit zu programmieren. Zum anderen gibt es bisher nur eine lose Kopplung in die Welt der Büroanwendungen bzw. der Leitsysteme. Es ist nun die Aufgabe das Paradigma von SPSen so zu erweitern, daß der Informationsaustausch mit externen auch nichtechtzeitfähigen Teilsystemen einfach und transparent möglich wird. Ebenso wird der Offenheit bezüglich der Integration von Fremdkomponenten in SPS-Systemen eine immer höhere Bedeutung zugemessen.

Die kostengünstige Entwicklung verlässlicher Software stellt in diesem Zusammenhang aber noch eine große Herausforderung dar. Die herkömmlichen Methoden für Entwurf und Erstellung von Echtzeitsystemen sind diesen Anforderungen nicht mehr gewachsen. Die Programmierung von SPSen hat zwar mit Einführung der neuen Norm IEC 1131-3 deutliche Fortschritte in Hinblick auf die Modularisierung und Portierung von Steuerungsprogrammen gebracht, allerdings wird wenig und ausschließlich auf Nachrichtenaustausch ausgerichtete Unterstützung bei der Entwicklung dezentraler SPS-Anwendungen geleistet [5].

Objektorientierte Methoden für Analyse und Entwurf sind in den vergangenen Jahren in zahlreichen Varianten vorgestellt worden (z.B. [2], [8]). Sie versprechen, Entwickler zu leiten, die Effizienz ihrer Arbeit zu steigern und die Entwicklungskosten durch bessere Wartbarkeit

und Wiederverwendbarkeit von Bausteinen längerfristig zu reduzieren. Eine Unterstützung für die spezifischen Probleme von Automatisierungsanwendungen wird aber erst in Ansätzen geboten [9]. Für den Bereich der Büroanwendungen hat die Object Management Group (OMG), ein Konsortium führender Unternehmen der Informationstechnologie, mit der Object Management Architecture (OMA) ein weit beachtetes Referenzmodell für objektorientierte Verarbeitung in verteilten, heterogenen, offenen Umgebungen definiert [6], das die Basis für eine Reihe von Spezifikationen bildet. Ein Object Request Broker (ORB) ist dabei eine zentrale Komponente, die es Objekten erlaubt, in der verteilten Umgebung transparent Aufrufe an andere Objekte zu tätigen bzw. von diesen entgegenzunehmen. Die Common Object Request Broker Architecture (CORBA) Spezifikation identifiziert die Komponenten eines ORBs und spezifiziert deren Programmierschnittstellen [7].

Das DIRECT-Projekt (DIstributed REaltime ConTrol) an der Fachhochschule Wiesbaden versucht, die Prinzipien der objektorientierten verteilten Programmierung entsprechend dem OMA/CORBA-Referenzmodell der OMG für die Entwicklung verteilter Anwendungen in der Automatisierungstechnik zu nutzen [3], [4]. In einer Zusammenarbeit zwischen der Fachhochschule Wiesbaden und der Klöckner-Moeller GmbH, Bonn, werden diese Ansätze derzeit auf die Integration von SPS-gesteuerten Teilsystemen in komplexe objektorientierte Automatisierungsanwendungen sowie auf die Programmierung vernetzter SPS-Systeme übertragen.

DIE DIRECT-ARCHITEKTUR

Der Entwicklung einer Automatisierungsanwendung im Rahmen von DIRECT liegt das CORBA-Modell zugrunde [3]. Eine Anwendung wird durch eine Menge von kooperierenden Objekten modelliert und implementiert, die auf den verschiedenen Stellen des verteilten Systems residieren. Die Spezifikation von Objektschnittstellen geschieht in der C++-ähnlichen CORBA Interface Definition Language (IDL). Vererbung von Schnittstelleneigenschaften ist in CORBA IDL möglich. Objektimplementierungen erfolgen in C++ und C. Aus der Sicht des Anwendungsentwicklers erfolgt ein Aufruf einer Operation eines entfernten oder lokalen Objekts gleichermaßen durch einen üblichen Prozeduraufruf. Die Verteiltheit des Systems ist für den Anwendungsentwickler damit weitgehend transparent und spielt primär bei der Konfigurierung und Initialisierung des Systems eine wesentliche Rolle.

Schnittstellenbeschreibungen abstrahieren zunächst von der Implementierung der angebotenen Funktionalität. So sind z.B. für den Nutzer einer Laborgeräteschnittstelle die sehr vielfältigen unterlagerten physikalischen Schnittstellen der Laborgeräte nicht mehr von Interesse. Mittels Vererbung von Schnittstellen können Familien von Geräteklassen dargestellt werden. Spezielle Eigenschaften von Geräten bestimmter Hersteller lassen sich z.B. in abgeleiteten Klassen isolieren, während die gemeinsame Funktionalität der Geräteklasse in einer allgemeinen Schnittstelle zusammengefaßt wird. Vererbung erlaubt auch die Wiederverwendbarkeit von Implementierungsklassen.

Es wird angenommen, daß alle Komponenten eines komplexen Automatisierungssystems durch ein zweistufiges Kommunikationsnetzwerk verbunden sind (vgl. Abb. 1). Ein LAN (z.B. Ethernet) verbindet Server-Rechner, Bedienplätze und bietet die Möglichkeit zur Integration des Systems in höhere Schichten der CIM-Architektur. Alle Geräte sowie komplexe SPS-gesteuerte Subsysteme werden über µController-basierte Frontends an echtzeitfähige Feldbus-Segmente angebunden, welche selbst über Gateways mit dem LAN

verbunden sind. Als Feldbus wird derzeit in DIRECT der CAN-Bus (Controller Area Network) eingesetzt, ein Multicast-fähiges Feldbussystem, das ursprünglich für den Automobilbereich entwickelt wurde. Alle Knoten werden derzeit unter Verwendung des PXROS Echtzeit-Kerns (HighTec, Saarbrücken) betrieben. Insgesamt liegt damit eine verteilte Systemarchitektur vor.

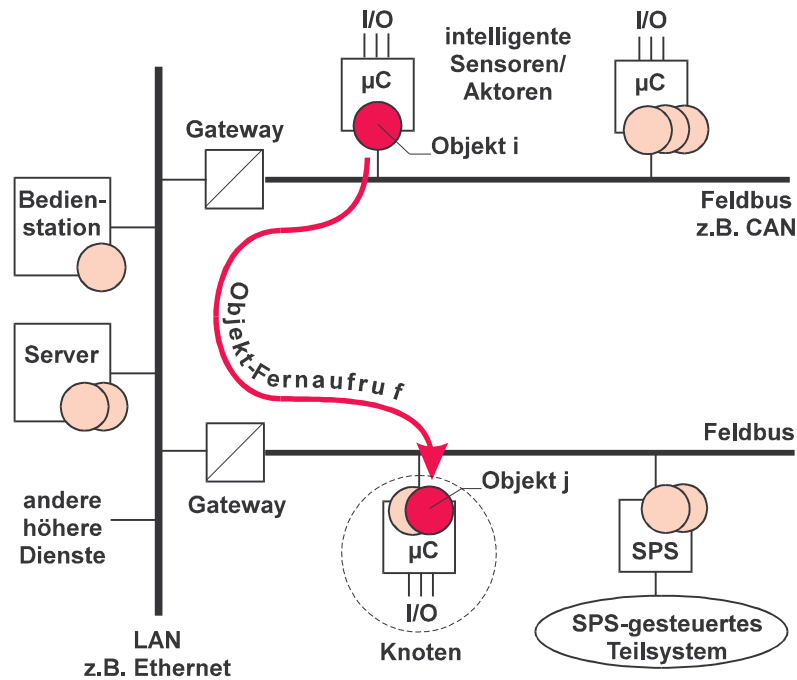


Abb. 1: Die DIRECT-Architektur

Für die Echtzeit-relevante Feldbus-Ebene sind objektorientierte Ansätze weitgehend neu, für die LAN-basierte Ebene existieren dagegen bereits mehrere CORBA-konforme Produkte. Im Rahmen von DIRECT wurde ein zur Erzielung von Echtzeiteigenschaften vereinfachter, nach CORBA-Prinzipien arbeitender Object Request Broker entworfen und implementiert, der als μ ORB bezeichnet wird. Er ermöglicht es, daß alle relevanten Einheiten als intelligente Objekte erscheinen. Für die Realisierung von knotenübergreifenden Fernaufrufen wurde ein RPC-Protokoll entwickelt, welches insbesondere Rücksicht auf die kleine Telegrammgröße in CAN-Netzwerken nimmt. Teil des RPC-Protokolls ist ein Segmentierungsverfahren, das einen anstehenden Request bei Bedarf in eine Folge von CAN-Nachrichtensegmenten zerlegt. Das RPC-Protokoll des μ ORB basiert aus Performance-Gründen auf einem elementaren CAN-Datagramm-Dienst. In Hinblick auf die geplanten Ergänzungen zur Durchsetzung von Echtzeiteigenschaften erlaubt das RPC-Protokoll bereits die Festlegung einer maximalen Antwortzeit für die Durchführung eines Fernaufrufs durch den Klienten, wobei bei nicht zusicherbarer Durchführung eine Ausnahme durch den Diensterbringer generiert wird. Der Zugriff zum CAN-übertragungsmedium ist darüberhinaus prioritätenbasiert. Jeder Klient kann im Rahmen des Binding zu einem Server eine für ihn geeignete Prioritätsklasse wählen. Details der Implementierung sind in [1] beschrieben.

INTEGRATION EINER SPS-GESTEUERTEN TEILANWENDUNG

Ein wichtiger Problembereich bei der Entwicklung komplexer Automatisierungsanwendungen ist die Integration von SPS-gesteuerten Teilsystemen. In diesem Kontext prallen häufig die

unterschiedlichen, durch die verschiedenen Verarbeitungsmodelle geprägten Sichtweisen der Beteiligten aufeinander. In DIRECT wird versucht, eine gemeinsame Basis durch CORBA-basierte "Wrapper" zu finden. Ein Wrapper verpackt eine zyklisch ablaufende SPS-Anwendung und verleiht ihr eine objektorientierte Hülle, die das SPS-Subsystem nach außen, etwa zum Zwecke der Parametrisierung oder Modusänderung, als eine Menge von Objekten erscheinen läßt. Die Integrierbarkeit der Teilanwendung in die Sichtweise der übergeordneten objektorientierten Anwendung ist dabei vollständig gegeben.

Die SPS kann sowohl eine Client-Rolle wie auch eine Server-Rolle einnehmen.

Die Integration einer Klöckner-Moeller SPS PS 416 in das CAN-Bus-basierte Experimentalsystem wurde erfolgreich abgeschlossen. Die Vorgehensweise ist in Abb. 2 dargestellt. Die SPS wird um eine zusätzliche Prozessor-Karte erweitert. Auf ihr läuft der μ ORB wie auf den anderen CAN-Knoten im System. Auf dem μ ORB residieren Objekte, die aus der Sicht entfernter Klienten als dienstbringende Objekte erscheinen, in Wirklichkeit aber nur sogenannte Proxy-Objekte sind, die einen eingehenden Aufruf über den Rückwandbus der SPS an zugehörige Funktionsbaustein-Instanzen in der eigentlichen SPS-CPU weitergeben. Die Proxy-Objekte bilden zusammen mit diesen Instanzen den o.a. Wrapper. Für jedes sichtbare Objekt kann zu einem Zeitpunkt höchstens ein Aufruf innerhalb der SPS in Bearbeitung sein.

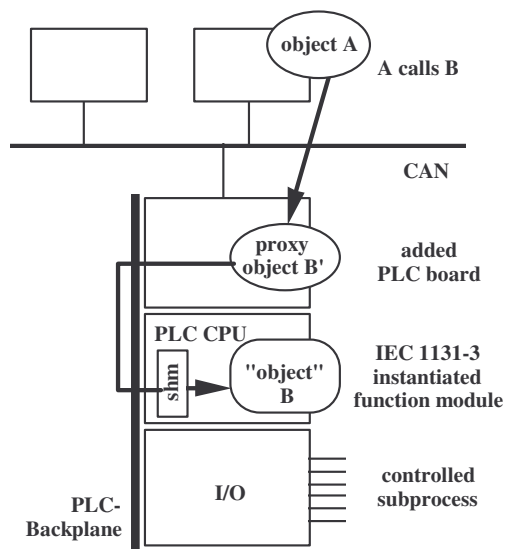


Abb. 2: SPS-Integration

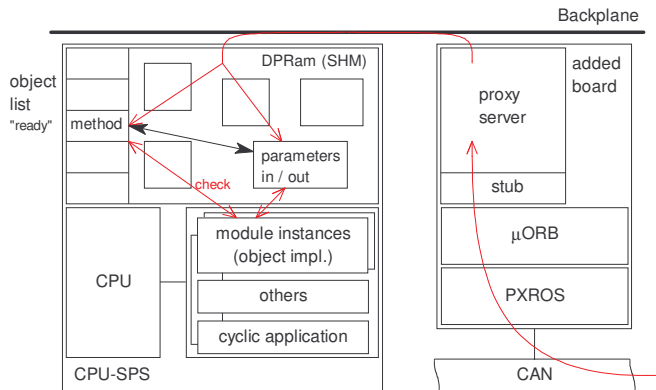


Abb. 3: Kommunikation zwischen SPS und µORB

Genauer wird die Kommunikation der Zusatz-Karte mit der SPS-CPU über ein spezielles dual-ported RAM abgewickelt, das für beide Teilnehmer zugreifbar ist. Es enthält eine sogenannte Objektliste mit einem Eintrag für jedes sichtbare Objekt. Dieser enthält ein sogenanntes Methoden-Byte, das die auszuführende Operation identifiziert. (In der derzeitigen Implementierung sind damit maximal 255 Methoden pro Klasse möglich, was für den praktischen Einsatz als ausreichend erscheint). Für jedes zu realisierende Objekt auf SPS-Seite wird ein spezieller Speicherbereich im DPRAM reserviert, in dem die Aufrufparameter bzw. Funktionsrückgabewerte abgelegt werden. Weiter existiert auf SPS-Seite ein generisches Funktionsmodul, das für die Bearbeitung der Methodenaufrufe im Rahmen der zyklischen Verarbeitung der SPS zuständig ist.

Zur Laufzeit schreibt das Proxy-Objekt im Falle einer durchzuführenden Operation für ein aufgerufenes Objekt zuerst alle Parameter in den hierzu vorgesehenen Speicherbereich im DPRAM. Anschließend wird die Verarbeitung durch das Schreiben der aufzurufenden Methode in die Objektliste freigegeben. Die SPS kann jetzt die übergebenen Parameter auslesen und den Methodenaufruf abarbeiten. Sobald ein Methodenaufruf komplett abgearbeitet wurde, werden alle Rückgabewerte in den vorgesehenen Speicherbereich im DPRAM geschrieben und das Objekt in der Objektliste als abgearbeitet gekennzeichnet (Methode 0). Der Proxy - Server kann jetzt die ermittelten Werte an den entsprechenden Client weiterleiten und einen neuen Aufruf an dieses Objekt starten.

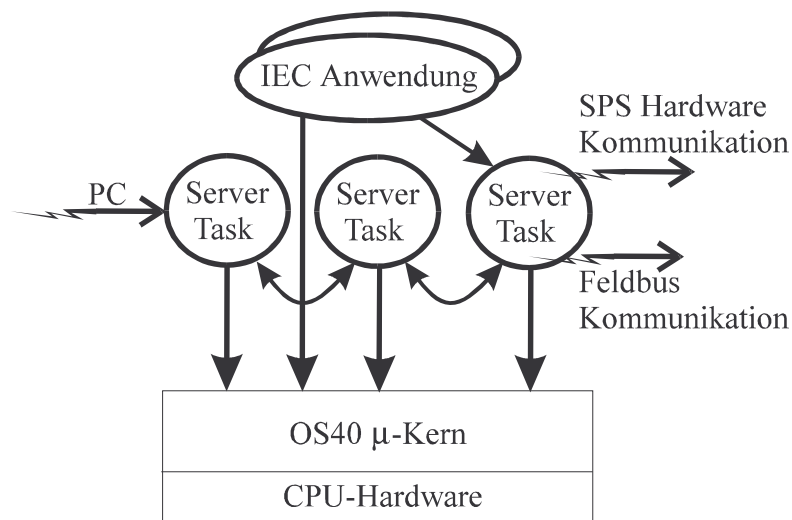
OBJEKTORIENTIERTE PROGRAMMIERUNG VERNETZTER SPS'EN

SPS-Modell

Der Ansatz der für den Entwurf einer dezentralen objekt-orientierten Automatisierungsanwendung vorgeschlagen wird, beruht auf einer dreistufigen Vorgehensweise. Zunächst wird eine Applikation als eine Menge von zu programmierenden Objekten angesehen. Die Objekte können dabei einen direkten Bezug zur Peripherie haben, wie z.B. Ventilinseln, Motoren und Achsen. Sie können aber auch ein Subkomponente der konkreten Anwendung mit bestimmten Eigenschaften repräsentieren (z.B. Temperaturzonenregelung). Eine aktuelle Problemstellung stellt sich also als ein Geflecht von einander abhängigen Objekten dar. Jedes Objekt wird dabei durch eine Instanz eines Funktionsbausteins im Sinne der IEC1131-3 dargestellt. Den Beziehungen von Objekten untereinander entsprechen Aufrufe zwischen den zugeordneten Funktionsbausteinen. Bei der Entwicklung muß auf die spätere räumliche Verteilung noch keine Rücksicht genommen werden. In einem zweiten Schritt wird durch eine Konfiguration

eine Zerlegung der Gesamtanwendung in Subsysteme und eine Zuordnung der Instanzen auf die einzelnen Automatisierungsgeräte vorgenommen. Die durch die Zerlegung notwendig gewordenen Kommunikationsbeziehungen werden von dem System automatisch generiert. In einem dritten Schritt werden die Subsysteme der verteilte Anwendung in die entsprechenden Automatisierungsgeräte geladen und dort zur Ausführung gebracht. Der notwendige synchronisierte Austausch von Daten wird vom System übernommen und im folgenden beschrieben.

Existieren zwischen zwei Objekten nur lokale Beziehungen, d.h. liegen beide Objekte auf dem selben Automatisierungsknoten sind keine weiteren Aktionen des Systems notwendig. Jede Beziehung zwischen verteilten Objekten erfordert dagegen vom Laufzeitsystem entsprechende Dienste. Die Basis für die Programmierung einer solchen verteilten Automatisierungsanwendung bildet ein System von geeignet vernetzten SPS-Knoten und das nachstehend beschriebene RPC-Protokoll. Jeder Knoten enthält die dabei gleichen Software Komponenten. Ein Klöckner-Moeller SPS-Knoten besitzt dabei die folgende Software Architektur auf Basis des Echtzeit-Multitasking Betriebssystem OS40.



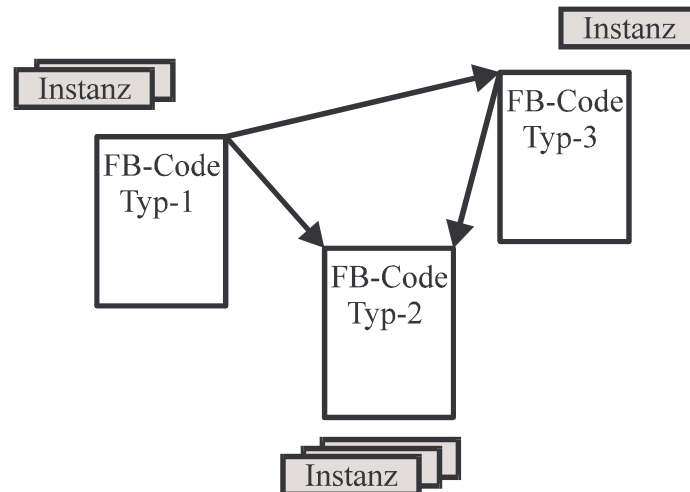
Software-Architektur in einer PS416-CPU400

Innere Struktur

Jedes Objekt welches ein Teil der verteilten Anwendung darstellt, muß geeignet auf Funktionsbausteine (FB) der IEC-Sprache abgebildet werden. In der Klöckner-Moeller Realisierung der IEC 1131-3 (Anweisungsliste AWL) besteht die Möglichkeit beliebig viele, nur durch den Speicherplatz beschränkte, typisierte Einheiten, die Instanzen genannt werden, aus einer abstrakten Spezifikation, der FB-Typdefinition, zu erzeugen. Dieser Vorgang wird Instanziierung genannt. Eine Instanz ist also die Inkarnation eines abstrakten Datentyps, bestehen aus einer Menge von Daten und aus einer Anzahl von erlaubten Operationen auf diesen Daten. Diese Operationen stellen die einzige korrekte Möglichkeit dar, Daten eines FBs zu modifizieren bzw. zu bearbeiten. Weiterhin ist es so, daß ein FB-Typ den Code für alle von ihm erzeugten Instanzen in Form einer funktionalen Schnittstelle bereitstellt. Der Ansatz von Klöckner-Moeller zeichnet sich also dadurch aus, daß alle Instanzen einer FB-Typdefinition die gleiche Struktur besitzen und den gleichen Code verwenden. Die Daten einer Instanz teilen sich in einen Ein/Ausgabebereich, den ein Aufrufer kennt und modifizieren kann, und einen lokalen, nur innerhalb der Operationen sichtbaren Datenbereich auf. Die verschiedenen Operationen eines Typs werden durch den Eingabebereich einer Instanz parametrisiert und ausgewählt.

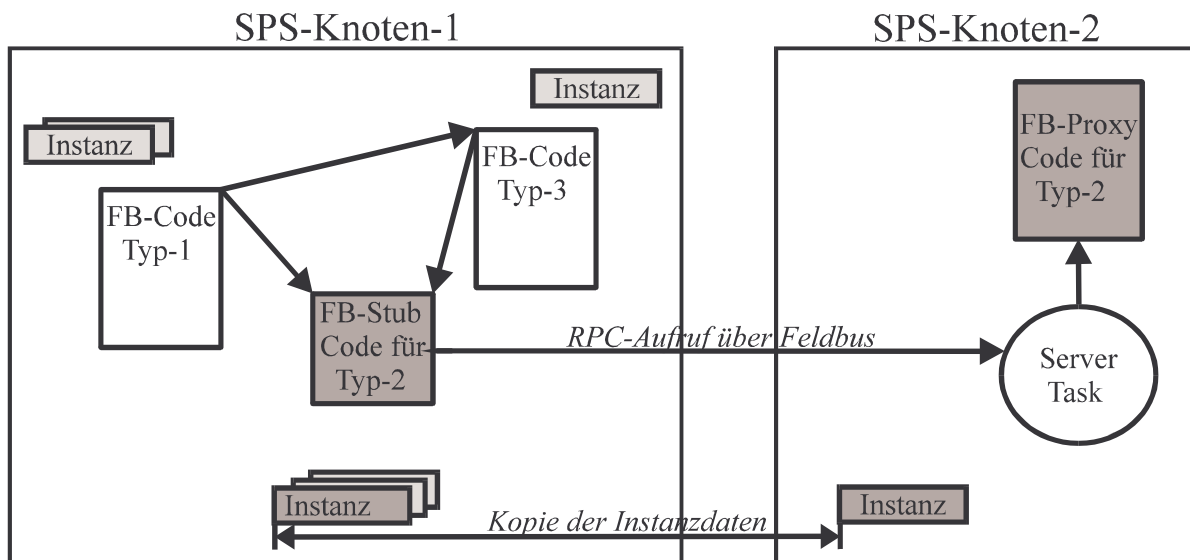
Strukturell kann daher jede Instanz genau ein Objekt einer verteilten, objektorientierten Anwendung repräsentieren. Dabei werden die Attribute eines Objektes auf die Daten und die Methoden auf die Operationen des FB-Typs abgebildet.

Ein typisches IEC 1131-3 AWL-Programm besteht also zum einen aus einer Menge von Einheiten (Dateien) die den Code der im Programm verwendeten FB-Typen enthalten. Zum anderen existiert eine durch die Programmstruktur exakt definierte Menge von Instanzen die zur Laufzeit nicht mehr veränderbar ist. Die Speicherbereiche für alle Instanzen werden vor dem Programmstart vom Betriebssystem erzeugt. Bei der zyklischen Programmverarbeitung wird ein FB-Code ein- oder mehrfach im Programm mit entsprechenden Instanzen aufgerufen und abgearbeitet.

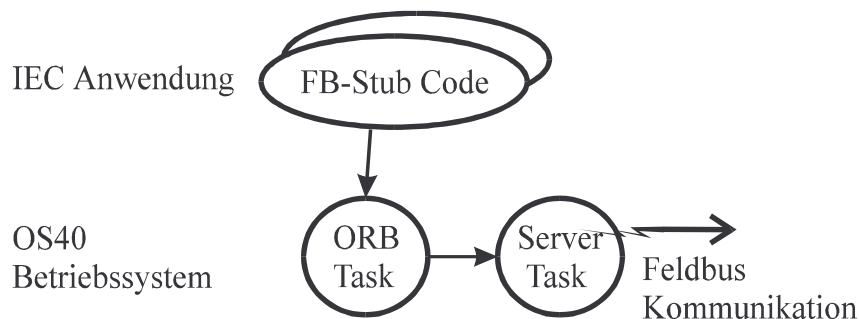


Beispiel: Drei FB-Typen mit beispielhaften Instanzen

Jedem entfernten Objekt ist auf dem eigenen Knoten ein lokales Stub-Objekt zugeordnet. Dieses besitzt eine zum eigentlichen Objekt identische Schnittstelle. Der Aufrufer oder Benutzer erkennt also keinen Unterschied zwischen einem lokalen und einem entfernten Objekt. Funktional hat ein Stub-Objekt die Aufgabe einen Methodenaufruf entgegen zu nehmen, die Argumente zu verpacken und diese über das unterliegende Netzwerk an den Knoten, der das eigentliche Objekt enthält, zu versenden. Dort werden die Argumente von einem Objekt-Repräsentanten (proxy-object) entgegengenommen, ausgepackt und damit ein Methodenaufruf zu dem gewünschten Zielobjekt durchgeführt. Dieses kann mit den Parametern und den aktuellen Attributen eine programmierte Aktion ausführen und zugehörige Ergebnisse liefern, die dann auf dem umgekehrten Weg zurück zum Aufrufer gelangen. Die folgende Abbildung zeigt beispielhaft die Auslagerung eines FB's auf einen entfernten SPS-Knoten.



Für jedes Stub-Objekt kann das System den ursprünglich definierten Typ verwenden. Dadurch ist gesichert, daß alle Instanzen in geeigneter Struktur erzeugt werden. Der Typspezifische Code wird jedoch durch einen Stub-Code ersetzt und stattdessen im Anwendungsprogramm verwendet. Für einen Aufrufer ergibt sich kein Unterschied, da die Aufrufumgebung identisch zur ursprünglichen ist. Der Stub-Code ist generisch, da die in Frage kommenden Instanzen als eine Menge von Bytes interpretiert werden können, die über den RPC-Mechanismus zu transportieren sind. Dabei ist lediglich die Größe und die physikalische Adresse einer konkreten Instanz von Interesse. Beides ist dem Stub-Code bekannt bzw. kann vom Laufzeitsystem erfragt werden.



Zur Realisierung einer solchen Objektverteilung verfügt das System über einen generischen ObjectRequestBroker (ORB), der die Aufrufe aller Client-Stubs entgegen nimmt, die Daten auf das unterliegende Netzwerk abbildet und an den Empfänger weiterleitet. Außerdem nimmt der ORB die Antworten aller Repräsentanten entgegen und verteilt sie geeignet auf die Client-Stubs.

ZUSAMMENFASSUNG

LITERATUR

- [1] Bauer, A.; RemŽdios, O.: "Objektorientierte Verarbeitung in verteilten Automatisierungssystemen", Diplomarbeit, Fachhochschule Wiesbaden, 1995
- [2] Coad, P.; Yourdan, E.: "Object-Oriented Analysis" (Second Ed.) and "Object-Oriented Design", Prentice-Hall, 1991

- [3] Kröger, R.; Bauer, A.; RemŽdios, O.; Thoss, M.: "Objektorientierte Programmierung verteilter Echtzeitanwendungen", Echtzeit'95, Karlsruhe, 20.-22. Juni 1995, Network GmbH, ISBN 3-924651-46-9, 1995
- [4] Kroeger, R.: "Using Object-Oriented Standards for Developing Realtime Control Applications", Proc. 3rd IFAC/IFIP Workshop on Algorithms and Architectures for Real-Time Control (AARTC'95), Ostend, May 31 - June 2, 1995
- [5] Neumann, P.; Grötsch, E.E.; Lubkoll, C.; Simon, R.: "SPS-Standard: IEC 1131-Programmierung in verteilten Automatisierungssystemen", Oldenbourg-Verlag, 1995
- [6] OMG: "Object Management Architecture Guide", Object Management Group, Framingham, MA, USA, 1992
- [7] OMG: "The Common Object Request Broker, Architecture and Specification, Rev.1.2", Object Management Group, Framingham, MA, USA, 1993
- [8] Rumbaugh, J. et al.: "Object-Oriented Modeling and Design", Prentice-Hall, 1991
- [9] Selic, B.; Gullekson, G.; Ward P. T.: "Real-Time Object-Oriented Modeling", J. Wiley & Sons, New York, 1994