Ein objekt-orientierter Ansatz für die Echtzeitdatenverarbeitung

Lambis Tassakos, Klaus W. Pleßmann

Lehr- und Forschungsgebiet für Verfahren der
Prozeßdatenverarbeitung und Prozeßführung
(pdv)
RWTH Aachen, Steinbachstr. 54a, D-5100 Aachen

Zusammenfassung

Der objekt-orientierte Ansatz stellt heute eine echte Alternative zur Entwicklung von großen Software-Systemen dar. Gleichzeitig mit der noch andauernden intensiven Grundlagenforschung auf diesem Gebiet werden die ersten kommerziellen Systeme angeboten und in verschiedenen Projekten mit der Ambition eingesetzt, die auch heute noch vorhandene Softwarekrise in den Griff zu bekommen. Unsere Beitrag beschäftigt sich mit der Frage, ob und mit welchen Voraussetzungen sich diese Entwicklungen auch im Bereich der Echtzeitsysteme in der Automatisierungstechnik einsetzen lassen. In diesem Zusammenhang stellen wir die Konzepte eines parallelen, objekt-orientierten Modells vor, das im Hinblick auf seinen Einsatz in komplexen Anwendungen in der Prozeßdatenverarbeitung entwickelt worden ist. Dabei werden sowohl verschiedene Entwurfsentscheidungen erläutert als auch die wichtigsten Aspekte vorgestellt und diskutiert. Zuvor wird zum allgemeinen Verständnis und zu einer klaren Definition und Abgrenzung der entsprechenden Begriffe kurz auf die grundlegenden Prinzipien des objekt-orientierten Ansatzes eingegangen, wie er heute allgemein auf internationaller Ebene verstanden wird.

1. Einleitung

Die objekt-orientierte Programmierung wird heute als eine praktische und nützliche Methode angesehen, große und komplexe Programme herzustellen /1,2/. Mit diesem Begriff wird nicht nur eine bestimmte Programmiersprache sondern auch die allgemeinere Programmierumgebung und -technik der Softwareproduktion mit aufgefaßt /3/. Diese innovative Programmtechnik unterstützt insbesondere einen modularen Entwurf in Zusammenhang mit einer inkrementellen Realisierung des Systems und bringt gleichzeitig die Voraussetzungen dafür, dem Traumziel einer allgemeinen, ausgedehnten Wiederverwendbarkeit von Software näher zu kommen. Gerade wegen dieser Eigenschaft verspricht die objekt-orientierte Methodik, ein wirksames Mittel gegen den größten Feind eines großen, komplexen Software-Systems zu sein: die ständig auftretenden Änderungen der Anwendungsgegebenheiten während des gesamten Software-Zyklus.

Die Motivation für die Anwendung einer objekt-orientierten Programmierung in der Prozeßdatenverarbeitung ist die gleiche, wie für andere Software-Systeme. Die mit der Entwicklung und vor allem mit der Wartung von Software zusammenhängenden Probleme sind in diesem Feld auch vorhanden. Gerade aus der Tatsache der ständigen Änderungen in allen Stadien des Lebens eines großen Software-Systems resultiert die heutige unbefriedigende Situation im Bereich der Software-Entwicklung, die mit dem Schlagwort Software-Krise bezeichnet wird. Während der Hardware-Ingenieur in den letzen Jahren gelernt hat, in Termen von wiederverwendbaren Modulen zu denken, was akkumulativ einen exponentiellen Zuwachs seiner Produktivität erlaubte, realisiert der Programmierer immer weiter seine Programme durch Hintereinanderschreiben von einmal verwendbaren Programmzeilen.

Durch die spezifischen Eigenschaften des objekt-orientierten Ansatzes, nämlich ausgedehnte Wiederverwendbarkeit, erhöhte Lesbarkeit, inhärente Flexibilität und Modularität, erhofft man sich auch im Bereich der Softwareproduktion einen ähnlichen exponenziellen Zuwachs, wie es für den Hardware-Entwurf heute der Fall ist, zu erhalten. Dementsprechend ist es nicht von ungefähr, daß man in diesem Zusammenhang nicht nur von Programmpaketen sondern auch gern von "Software-ICs" spricht /4/.

Mit einem gezielten Einsatz eines objekt-orientierten Modells für die Modellierung und Programmierung von Realzeitsystemen kann eine Verbesserung der Software-

Produktivität und eine Herabsetzung der Entwicklungs- und Wartungskosten dieser Systeme herbeigeführt werden. Echtzeitsysteme haben oft ein relativ langes Systemleben, so daß die Verminderung der Wartungskosten einen deutlichen Beitrag zur Verminderung der Gesamtkosten bedeutet. Die gestiegenen Anforderungen an die Flexibilität solcher Systemen können nämlich nur mit einer entsprechenden Methodik bzgl. der Wartung begegnet werden, was für moderne, mit lokaler "Intelligenz" ausgestatteten Systeme zum größten Teil mit der Wartung der Software sehr eng zusammenhängt.

2. Der objekt-orientierte Ansatz

Die Entstehung und die Evolution des objekt-orientierten Ansatzes hat sich in zwei Bereichen abgespielt, die ähnliche Ziele verfolgt haben. Der erste Bereich, innerhalb dessen die Ideen des objekt-orientierten Ansatzes entwickelt worden sind, ist der Bereich der Systemmodellierung und der Programmiersprachen /5/. Der andere Bereich liegt im Gebiet des Operationsprinzips der Rechnersysteme und ist gleichzeitig aus der Organisation der Systemzuverlässigkeit und der Protektion von Rechnerarchitekturen geprägt /6/. In beiden Fällen stand die Intention im Vordergrund, durch Erhöhung der Abstraktionsebene aus der Sicht des Benutzers den Einsatz von Rechnern plausibler, überschaubarer, effizienter, zuverlässiger und wirtschaftlicher zu verwirklichen.

Die Erhöhung der Abstraktion basiert im Gegensatz zum aktions-orientierten konventionellen Ansatz auf dem Objekt-Modell. Dabei wird die Ansicht hervorgehoben, daß ein Programm im Grunde genommen die Definition, Kreation, Manipulation und Wechselwirkung zwischen Mengen von autonomen Komponenten beschreibt, die Objekte genannt werden /7/. Dieser Ansicht sollten die entwickelten Operationsprinzipien der Rechner Rechnung tragen und dementsprechend ein objekt-orientiertes Berechnungsmodell unterstützen. Damit könnten diese Prinzipien dazu beitragen, die große semantische Lücke zwischen Hardware und Software zu schließen.

Bei der traditionellen Sicht werden die Softwaresysteme als eine Sammlung von Daten und eine Menge von Prozeduren angesehen. Die Daten präsentieren die Informationen im System, und die Prozeduren die Manipulation der Information. Dabei werden Daten und Prozeduren so behandelt, als ob sie voneinander unabhängig wären, was natürlich nicht der Fall ist. Beim objekt-orientierten Ansatz hat

man dagegen nur einen grundlegenden Begriff, der sowohl Daten als auch Prozeduren repräsentiert: das Objekt. Die Objekte besitzen in dieser Hinsicht einen dualen Charakter: Sie können als Daten angesehen werden, da sie manipulierbare Information darstellen. Sie verhalten sich aber auch als Prozeduren, da sie auch die Manipulationen der Daten beschreiben. Die Objekte können also sowohl passive als auch aktive Rollen übernehmen. Dies stellt einen Dualismus von passiven Daten und aktiven Prozeduren dar. Durch diese Integration (encapsulation) von Daten und ihren assoziierten Aktionen innerhalb der Objekte wird das Prinzip der Informationsabkapselung (information hiding) direkt unterstützt.

Objekt-orientierte Systeme betrachten die Welt als eine Ansammlung von Objekten, wobei sich die Welt dadurch verändert, indem ihre Objekte ihren inneren Zustand verändern und indem sie durch das Senden und Empfangen von Nachrichten in Wechselwirkungen mit den anderen Objekten treten /8/. Die Evolution des Informationsprozesses findet durch die Zustandstransitionen der Objekte statt.

Obwohl es sehr schwierig ist, den Begriff "Objekt" exakt zu definieren, kann man im allgemeinen ein Objekt folgendermaßen beschreiben:

> Ein Objekt ist eine abgekapselte Einheit, die sowohl aus einer Informationsmenge (lokalen Daten) als auch aus den diese Informationen manipulierenden Operationen (Prozeduren) besteht.

Die Objekte beinhalten also die Informationen, die nur ihnen gehören. Damit verkörpern die objekt-orientierten Systeme das Prinzip der Datenabkapselung auf eine natürliche Weise. Information wird durch das Senden einer Nachricht an das Objekt, das diese Information enthält, manipuliert. Diese Nachricht benennt die gewünschte Manipulation aus der Menge der im Objekt vorhandenen Manipulationsmöglichkeiten (Operationen). Für diesen Zweck enthalten die Nachrichten einen sogenannten Selektor, der vom empfangenden Objekt dazu benutzt wird, die entsprechende Operation aus den ihm zur Verfügung stehenden Operationen herauszufinden. Diese Operationen werden im objekt-orientierten Jargon Methoden genannt.

Durch den Begriff der Nachricht werden gleichzeitig zwei Dinge realisiert. Zum einem werden durch das Senden von Nachrichten die Objekte zu Verarbeitungsaktivitäten aufgefordert, wobei diese Aktivitäten innerhalb der Objekte stattfinden. Bei einer solchen "Aktivierung" führt das Objekt eine seiner Manipula-

tionsroutinen auf seine eigenen Daten aus und kann währenddessen auch weitere Nachrichten an andere Objekte senden, um weitere Aktivitäten anzufordern. Zum anderen stellen die Nachrichten die Basis zur Kommunikation zwischen den Objekten dar, weil eine Nachricht außer als Ankündigung einer Anforderung zusätzlich als Träger beliebiger Informationen (damit auch ganzer Objekte !) benutzt werden kann.

3. Echtzeitverarbeitung und der objekt-orientierte Ansatz

Entsprechend dem Prinzip der Objektabkapselung können die Teile eines Echtzeitsystems als Objekte mit einer konkreten Funktionalität und mit bestimmten Eigenschaften herausgehoben und spezifiziert werden. Der Entwerfer eines Objektes gestaltet es entsprechend seiner spezifizierten Funktionalität und bietet den potentiellen Benutzern dieses Objektes eine abstrakte Schnittstelle für seine Interaktion mit der übrigen Welt an. Dies geschieht durch die Angabe seines Protokolls (d.h. die Menge aller seiner aufrufbaren Methoden). Damit verbirgt er gleichzeitig alle intern programmierten, für den Benutzer unnötigen Einzelheiten des Objektes. Auf diese Art kann in einem logischen Sinn sowohl eine Zerlegung des gesamten Systems in seine natürlichen Teile, als auch seine Synthese aus seinen Bestandteilen erreicht werden, wobei diese beiden Möglichkeiten normalerweise kombiniert verwendet werden.

Als Ausgangsbasis für den Aufbau der Objektwelt einer Applikation sollte das Ergebnis der Anforderungsphase und der Systemanalyse genommen werden. Dadurch kann man danach in die Entwurfsphase des Systems in einer homogenen Weise ohne aufwendige und fehleranfällige semantische Transformationen zwischen Anforderungen und Entwürfen übergehen. Eins der wesentlichen Merkmale ist hierbei, daß man keine starren und abgekapselten Projektphasen mehr hat, da der oben erwähnte fließende Übergang solche Grenzen aufhebt und eine flexible Rückkopplung und offene Wechselwirkung zwischen diesen Phasen ermöglicht /9/.

Durch die Benutzung eines entsprechenden objekt-orientierten Systems, welches das grundlegende Modell in der Software und eventuell sogar mit seiner Hardware unterstützt, kann man den oben geschilderten natürlichen Phasenübergang bzw. die traditionelle Phasenauflösung bis hin zu der Realisierung und sogar Wartung des Systems ausdehnen. Die daraus entstehende Homogenität der angewandten Methodiken und der verwendeten Werkzeuge während des gesamten Zyklus eines Pro-

jektes stellt ein sehr wirksames Mittel zur Bekämpfung und Reduzierung der Komplexität und der Kosten eines Projekts dar.

Um einen glatten Übergang (smooth transition) zwischen den einzelnen Projektphasen zu ermöglichen, muß allerdings der zugrundeliegende objekt-orientierte Ansatz mit seiner konzeptionellen Mächtigkeit und seinen Spezifikationsmöglichkeiten die entsprechenden Bedürfnisse des jeweiligen Einsatzgebietes und in unserem Fall die der Echtzeitverarbeitung abdecken. Dabei ist die Hervorhebung eines aktiven Charakters der Bestandteile des Echtzeitsystems von elementarer Bedeutung. In diesem Umfeld sind viele Systemkomponenten ständig aktiv und befinden sich in einer kontinuierlichen Wechselwirkung miteinander. Der mit dem objekt-orientierten Ansatz verbundene Dualismus der Objekte bezüglich Information und Manipulation bzw. Aktivität bietet hierfür eine hervorragende Grundlage. Dafür reicht allerdings nicht das übliche, nach Smalltalk-Art gestaltete Modell aus, wo in jedem Zeitpunkt nur ein Objekt aktiv ist und wo ein Objekt erst nach Beendigung seiner aktuellen Methode in der Lage ist, weitere an ihm gerichtete Nachrichten zu empfangen. Damit wird angedeutet, daß für den Einsatz des objekt-orientierten Ansatzes in der Prozeßdatenverarbeitung eine Variante benötigt wird, die eine selbständige und parallele Aktivitätsentfaltung der Objekte in ihrem Grundkonzept beinhaltet.

Aufgrund des interpretativen Charakters des Selektionsmechanismus, der mit der Assoziierung einer Objektmethode zu einer angekommenen Nachricht verbunden ist, bietet sich der objekt-orientierte Ansatz im Hinblick auf die Laufzeiteffizienz der Software auf konventionellen Prozeßrechner eher für lose-gekoppelte Strukturen. Bei Echtzeitsystemen können die Vorteile des dynamischen Bindens voll ausgenutzt werden, wo keine strengen Anforderungen an eine 100% Einhaltung von Zeitlimits vorliegen. Moderne, komplexe Echtzeitsysteme enthalten allerdings eine Mischung von eng und lose gekoppelten Strukturen, so daß ein objekt-orientiertes Modell hybrider Natur (d.h., es muß keine reine objekt-orientierte Welt aufgebaut werden, sondern es wird innerhalb von Objekten auch eine effiziente, auf konventionelle Art durchzuführende Programmausführung ermöglicht) für die Prozeßdatenverarbeitung vom besonderen Interesse zu sein scheint. Die für die Berechnung der Bahn eines Roboterarmes benötigten Operationen und die Natur ihrer Operanden ist ein Beispiel von einer eng gekoppelten Struktur, wo alles schon bei der Compilierzeit bekannt ist und harte zeitliche Anforderungen gesetzt werden. Im Gegensatz dazu präsentiert eine flexible Produktionsinsel ein lose gekoppeltes System, wo

einerseits die zeitlichen Anforderungen anderen Charakters sind und anderseits eine hohe Flexibilität bzgl. Modifikationen und Erweiterbarkeit gefordert wird.

Die Ausnahme- und Unterbrechungsbehandlung gehören zu den grundlegenden Aspekten in Echtzeitsystemen. Im Gegensatz zum normalen objekt-orientierten Modell müssen die Objekte hier u.U auch in der Lage sein, ihre aktuellen Aktivitäten zu unterbrechen und sie entsprechend den ständig verändernden Gegebenheiten des technischen Prozesses anzupassen. Eine besondere Anforderung an die Spezifikationsmöglichkeiten des Modells stellt die Notwendigkeit dar, sogenannte Unterbrechungsobjekte als solche zu spezifizieren und in eine einfache und effiziente Art an das Unterbrechungssystem des Rechners anzubinden. Effizient bedeutet hierbei meistens eine feste und hardwaremäßig unterstützte Verbindung, was wiederum die Notwendigkeit einer harmonischen Koexistenz einer statischen mit einer dynamischen Struktur in großen, komplexen Echtzeitsystemen zum Ausdruck bringt.

Einer der Vorteile des objekt-orientierten Ansatzes für die Spezifikation und Programmierung von technischen Systemen ist, daß er ein natürliches Modell für die Abbildung der realen Welt des technischen Prozesses auf die Software anbietet. Mit seiner Hilfe kann man reale Objekte aus der Prozeßumgebung direkt in entsprechende Software-Objekte in das Rechnersystem abbilden: Sensor A sendet der Maschine B die Nachricht "Überschreitung der Temperatur" und löst damit eine entsprechende Reaktion der Maschine B aus, deren Natur und Einzelheiten nur die Maschine B selbst kennt und aufgrund der übertragenen Parameter und ihres internen Zustandes entscheidet (design decision and information hiding). Auf diese Weise hat man zweierlei Gewinn: einerseits einen adäquaten und logischen Systemaufbau und anderseits einen modularen und hochqualitativen Entwurf, der einfach zu warten ist.

4. Ein paralleles, objekt-orientiertes Modell für die Echtzeitverarbeitung

4.1 Grundlagen

Als Grundelemente in diesem Modell stellen die Objekte aktive, autonome Aktoren dar, die Fähigkeiten zur eigenen Aktivitätsentfaltung in der Objektwelt und ein eigenes Gedächnis zum Festhalten von Daten, Informationen und sogar Wissen inhärent besitzen können. Durch die Entfaltung eigener Aktivitäten demon-

striert jedes Objekt ein individuelles Verhalten zu seiner Umwelt, wobei es durch den Kontakt mit seiner unmittelbaren Umwelt in Wechselwirkungen mit der gesamten übrigen Objektwelt kommt. Dieser Kontakt äußert sich insbesondere durch eine auf Nachrichtenaustausch basierte Kommunikation mit den übrigen Objekten. Jedes Objekt entspricht einer logischen oder physikalischen Einheit im Problembereich. Damit kann die konzeptionelle Lösung des Problems in einer unifizierten und natürlichen Weise stattfinden, wobei das gleiche auch für die Implementation in Software und Hardware gilt.

Um gefährliche interne Interferenzen vorzubeugen, wird jedem elementaren Objekt erlaubt, zu jedem Zeitpunkt nur eine Aktion durchzuführen. Es kann z.B. nicht gleichzeitig mehrere seiner Manipulationsroutinen auf seinen inneren (permanenten und/oder temporären) Zustandsraum anwenden, oder gleichzeitig senden und empfangen. Die Parallelität kommt erst durch das Zusammenspiel mehrerer Objekte zustande, wie es im nächsten Unterkapitel erläutert wird. Zusammengesetzte Objekte bieten auch die Möglichkeit, mehrere parallele Aktivitäten innerhalb eines einzigen (nicht mehr elementaren) Objektes zu modellieren.

Der wesentliche Unterschied zu anderen objekt-orientierten Modellen besteht allerdings darin, daß ein Objekt parallel zu seiner operationellen Tätigkeit in der Lage ist, ankommende Nachrichten wahrzunehmen. Ein solches Objekt kann dann gegebenenfalls die aktuell ausgeführte Routine (Methode) abbrechen und mit der Ausführung einer anderen Routine beginnen. Entsprechend der Semantik und der Dringlichkeit der angekommenen Nachricht und in Abhängigkeit seines internen Zustandes kann nämlich das Objekt selbständig entscheiden, ob es die aktuelle Tätigkeit fortsetzt oder sie abbricht und mit der Ausführung der mit der angekommenen Nachricht assoziierten Methode beginnen soll. Damit sind insbesondere Aspekte der Behandlung von Ausnahme- und Fehlersituationen einerseits und von Unterbrechungen andererseits verbunden. Diese inhärente Objektfähigkeit eröffnet erst den Weg zu einem echtzeitkonformen Operationsmodell.

4.2 Parallelität

Wir betrachten die Parallelität als eine grundlegende Eigenschaft unseres Modells, die von Anfang an zusammen mit den übrigen Grundkonzepten entwickelt und nicht wie bei anderen Modellen erst später als eine zusätzliche Erweiterung addiert wurde (z.B. /Concurrent Smalltalk/). Ein paralleles Operationsprinzip ist nämlich die intuitive und natürliche Art, auf die die vielen eigenständigen Objekte eines

technischen Prozesses bzw. eines großen, verteilten Systems ihre Aktivitäten entfalten /10/. Motiviert auch von dem Wunsch, das hier diskutierte Modell für Anwendungen der Prozeßdatenverarbeitung benutzen zu wollen, wird hier die Parallelität durch das Hinzufügen eines jeweils eigenen Rumpfes (Body) in die Objekte gestaltet. Das Objekt ist dann in der Lage, diesen Body unabhängig vom Eintreffen von Nachrichten auszuführen, und damit als eine unabhängige, aktive Einheit eigenständig zu fungieren. Auf diese Weise können die ständig laufenden Prozesse und die kontinuierlichen Aktivitäten, wie sie in der technischen Umgebung vorkommen, auf eine natürliche Art modelliert werden, womit eine wesentliche Forderung der Prozeßdatenverarbeitung auf eine für den Benutzer intuitive Weise erfüllt wird.

Die Aufname eines Body in ein Objekt ist optional. Nach seiner Erzeugung beginnt jedes Objekt zunächst mit der Ausführung seines Body. In diesem Fall gibt es durch den "accept"-Befehl explizite Stellen in den Body-Routinen, wo die Bedienung von externen, angekommenen Nachrichten erlaubt ist. Dies läßt die Konstruktion und Einhaltung von internen Invarianten zu. Nur Ausnahme-Nachrichten (wie sie im folgenden Unterkapitel erläutert werden) können die normale Programmausführung auch in anderen Stellen unterbrechen und eine Ausnahmebehandlung auslösen.

Die eventuell notwendige Garantie der internen Variableninvarianten innerhalb eines Objektes und/oder die Koordination und Synchronisation seiner Aktivitäten obliegt der alleinigen Verantwortung des Objektes selbst. Das bedeutet, daß hier keine Semaphoren-Objekte für die Koordinierung des Zugriffes auf gemeinsam bekannte Objekte verwendet werden, wie es z.B. in parallelen Versionen von Smalltalk der Fall ist. Wenn ein Objekt ein eingeschränktes Betriebsmittel darstellt (z.B. ein eingeschränkter Puffer), muß es für die benötigte Koordination mit Hilfe seines Body selbst sorgen.

4.3 Kommunikation und Synchronisation

Das Kommunikationsmodell ist für einen parallelen, objekt-orientierten Ansatz von zentraler Bedeutung, da damit nicht nur ein flexibler Aufruf- und Kommunikationsmechanismus sondern insbesondere die Synchronisation und die Koordination der kooperativen Aktivitäten der parallelen Objekte verbunden ist. In unserem Modell unterscheiden wir bewußt zwischen der Wahrnehmung über das Ankommen einer Nachricht und dem Beginn der Ausführung der assoziierten Methode. Weil die Objekte als eigenständige Aktoren fungieren, sollen sie in der Lage sein, ihre Aktivi-

täten zu entfalten und gleichzeitig das Eintreffen einer Nachricht zu registrieren. Dies erlaubt den Objekten selektive Aktivitäten auf der Basis von vorprogrammierten Schemata zu verwirklichen.

Es gibt sowohl eine synchrone als auch eine asynchrone Kommunikationsart, die allerdings auf den gleichen semantischen Prizipien aufbauen. Damit kann der Benutzer auf eine einfache Weise die für ihn jeweils geeignete Art auswählen. Vier Kommunikationsprimitiven bilden die Basis für unser Kommunikationsmodell:

- **send** (destination object, message)
- **reliableSend** (destination object, message)
- **ask** (destination object, message) und
- **reply** (destination object, expression)

Mit Hilfe des **send**-Befehls findet eine asynchrone Kommunikation statt. Nachdem die Nachricht vom Sender (Objekt A) konstruiert und gesendet wurde, kann er seine weitere Aktivitäten fortsetzen (Bild 1). Die Nachricht wird nach einer gewissen Zeit an ihrem Empfangsort ankommen (tB1), vom Empfänger (Objekt B) wahrgenommen (tB2), und noch später (vielleicht auch direkt, wenn das empfangende Objekt auf Nachrichten gewartet hat und keine andere Nachricht vorliegt) die Selektion und Ausführung einer Methode des Empfängers auslösen (tB3).

Obwohl durch den send-Befehl der maximale Parallelitätsgrad erreicht wird, wird die Gültigkeit zweier Annahmen bzgl. der Nachrichtenübertragung nicht garantiert,

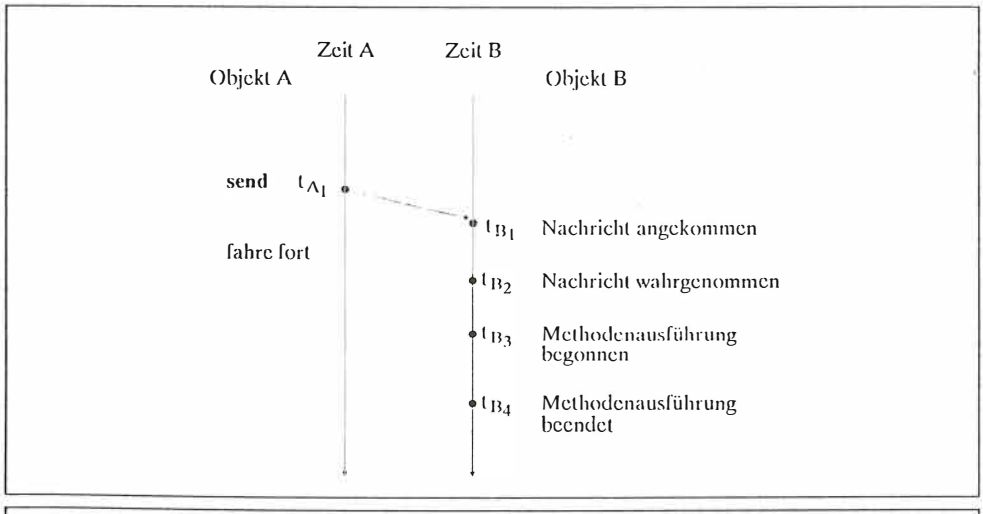

Bild 1. Der send-Befehl

nämlich:

- die tatsächliche Lieferung der Nachricht an den Empfänger und
- die Einhaltung der zeitlichen Ordnung zweier Nachrichten bei ihrer Übertragung /11/

Der **reliableSend**-Befehl realisiert inhärent beide Annahmen. Das sendende Objekt A wartet mit der Entfaltung weiterer Aktivitäten, bis es eine Bestätigung für die Wahrnehmung der gesendeten Nachricht von Seite des Empfängers erhalten hat (Bild 2). Damit kann der Sender sicher sein, daß der Empfänger seine Nachricht erhalten hat. Darüber hinaus ist die zweite obige Forderung auch erfüllt, weil bei einer wiederholten Ausführung solcher reliableSend-Befehle vom Objekt A jeweils auf die entsprechende Bestätigung gewartet werden muß, bevor der eine Befehl abgeschlossen und der nächste begonnen werden kann.

Es gibt eine spezielle Form des reliableSend-Befehls, die sich von der normalen dadurch unterscheidet, daß das empfangende Objekt nach der Registrierung der Nachricht seine bisherige Aktivität abbricht und sofort mit der Ausführung der mit der Nachricht assoziierten Methode anfängt (Bild 3, tB2 = tB3):

- **exceptionalSend** (destination object, message)

Die durch diesen Befehl erzeugten Nachrichten heißen Ausnahmenachrichten (ex-

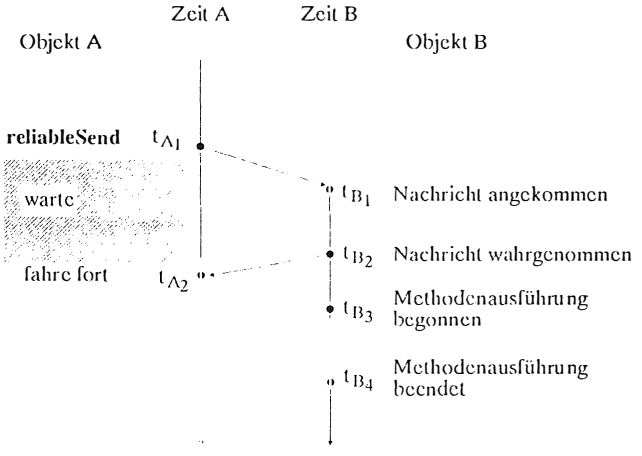

Bild 2. Der reliableSend-Befehl

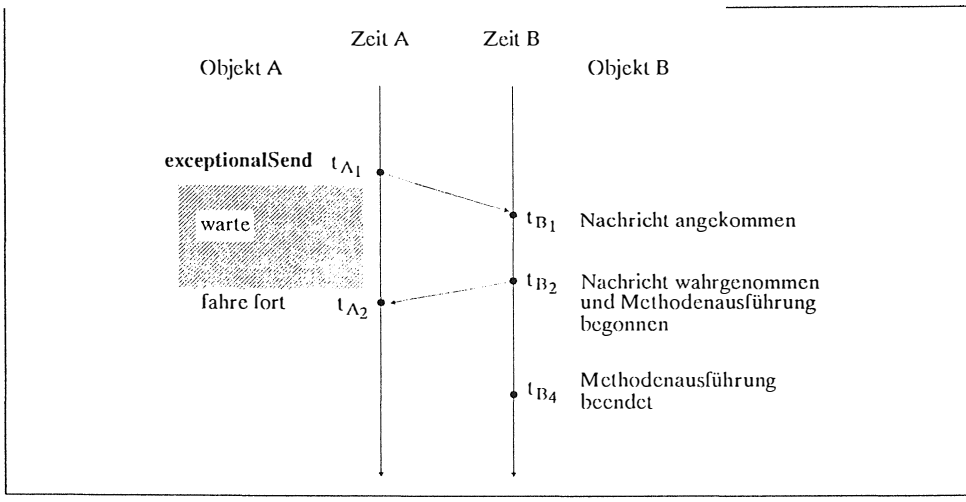

Bild 3. Der exceptionalSend-Befehl

ceptional messages). Es kann auch vereinbart werden, daß diese Nachrichten die Stimulierung von nur einer bestimmten Sorte von "Ausnahme-Methoden" zur Folge hat, welche den Aspekten von Ausnahme- und Unterbrecherbehandlung dienen. Diese Ausnahmemethoden können durch das Ankommen weiterer Nachrichten nicht mehr unterbrochen werden; das Objekt kapselt sich von der Außenwelt ab, um sich auf die Ausnahmebehandlung zu "konzentrieren".

Wenn das sendende Objekt die Ergebnisse seiner Nachrichtentransaktion benötigt, um die Ausführung seiner aktuellen Routine fortzusetzen, dann benutzt es den **ask**-Befehl. Dabei wartet der Sender nicht nur, bis seine Nachricht an den Empfänger angekommen ist, sondern solange, bis dieser seine Anforderung bearbeitet hat und die Resultate an ihn durch den **reply**-Befehl zurückgesendet hat (Bild 4). Damit verkörpert das Befehlspaar ask/reply die Funktionalität eines ("remote") "procedure call"-Befehls. Die durch den reply-Befehl erzeugte Nachricht unterscheidet sich dadurch von einer normalen Nachricht, daß sie bei ihrem Ankommen an das anfordernde Objekt keine Methodenselektion und -ausführung auslöst, sondern einfach das Resultat der gestellten Anforderung überträgt. Damit sieht man, daß es eigentlich zwei Stellen gibt, wo Nachrichten an ein Objekt eintreffen können.

Die an den Empfänger einer Nachricht gestellte Dienstleistungsanforderung muß er nicht unbedingt selbst weiter bearbeiten. Wenn er ein anderes Objekt kennt, das diese Dienstleistung (besser) erbringen kann, kann er durch eine weitere entsprechende Nachricht die Anforderung an dieses Objekt **weiterleiten**. Durch gleichzeiti-

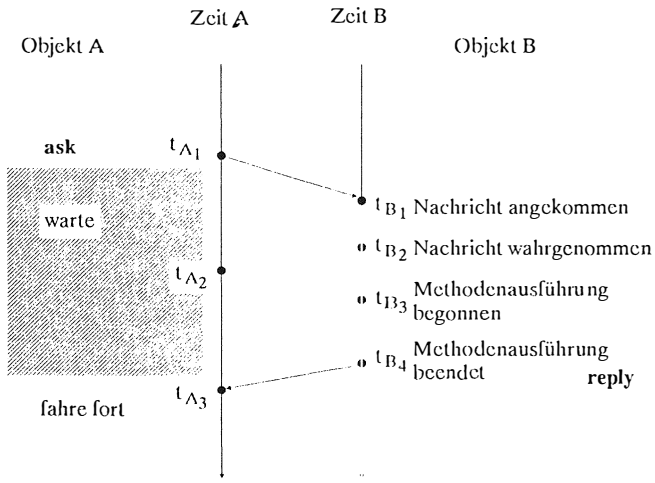

Bild 4. Das Befehlspaar ask/reply

ge Bekanntmachung des Namens des ursprünglich anfordernden Objektes kann er sogar erreichen, daß er sich von jeglicher Weiterverarbeitung der Anforderung befreit, da jetzt das dritte Objekt die Antwort direkt an das erste Objekt schicken kann. Damit entsteht eine hervorragende Grundlage für die Einführung globaler Server in das System, die mit Hilfe paralleler "Helfer-Objekte" sehr schnell mehrere Anfragen gleichzeitig bearbeiten können.

4.4 Zusammengesetzte Objekte

In vielen Fällen besteht der Wunsch und z.T. auch die Notwendigkeit, eine bestimmte Anzahl bzw. Gruppe von Objekten als eine einzige und logisch zusammenhängende Einheit zu betrachten und anzusprechen /12/. Man könnte so auf eine natürliche Art eine Maschine modellieren, bei der die einzelnen Teile trotz der Zusammenfassung zu einem zusammengesetzten Objekt immer noch als eigenständige (und kooperierende) Objekte definiert werden können. Die meisten objekt-orientierten Systeme unterstützen heute keine Spezifikationen und Wechselwirkungen solcher Art, sondern sie erlauben lediglich den Ausdruck von Kollektionen von Objekten, wobei die Zusammengehörigkeit nur über lose Verweise zum Erscheinen kommt.

Ein komplexes Objekt kann so angesehen werden, daß es aus einer strukturellen Hierarchie von Komponentenobjekten besteht /13/. Im Gegensatz zu der "ist-ein"-

Relation, die man in der Klassenhierarchie findet, handelt es sich hier um eine "ist-Teil-von"-Relation. Es ist wichtig dabei anzumerken, daß alle Komponentenobjekte eines zusammengesetzten Objektes in der vorgesehenen Weise vom Eintreffen einer an das zusammengesetzte Objekt gesandten Nachricht beeinflußt werden. Der Begriff des zusammengesetzten Objektes unterstützt einen parallelen Programmentwurf in unserem Modell in einer zuverlässigen Weise. Er trägt eine wesentliche Rolle zur Erhaltung einer ausgedehnten Parallelität trotz der Definition und Einführung von komplexen Objekten bei.

Ein solches Objekt könnte zwar durch eine ausgiebige und strukturierte Ausnutzung der Vererbungsmechanismen eine komplizierte Funktionalität auf modulare und zuverlässige Weise realisieren, könnte aber jederzeit nur eine einzige Aktivität entfalten, womit es in einer eher "verklemmten" Weise in Wechselwirkungen mit seiner Umwelt treten würde. Da es jederzeit z.B. eine einzige Nachricht empfangen bzw. bearbeiten kann, könnte es oft nur nach einer gewissen "Hysterese" auf das Eintreffen weiterer Nachrichten reagieren. Durch seine Aufteilung in eine Anzahl von kleineren, eigenständigen und doch zusammenhängenden Objekten kann es mehrere Aktivitäten gleichzeitig entfalten und dies vor allem auf eine direktere Art, was für die Belange der Prozeßdatenverarbeitung ja von außerordentlicher Bedeutung ist.

Nicht nur das Stammobjekt sondern auch die Komponentenobjekte eines zusammengesetzten Objektes können nämlich für die übrigen Objekten einer Applikation sichtbar sein. Die Interaktion zwischen einem zusammengesetzten Objekt und einem anderen Objekt oder sogar zwischen zwei aus mehreren Teilobjekten bestehenden Objekten kann auf eine natürliche und hoch parallele Weise stattfinden, ohne daß die jeweiligen Stammobjekte (an der Spitze des jeweiligen Abstraktionsbaums) für alle Interaktionen der Objekte ihrer abhängigen Objektbäume immer intervenieren müssen (Bild 5). Dies resultiert in einer Leistungssteigerung des Systems, weil die Nachrichten, die zwischen zwei Objekten unterschiedlicher Teilobjektbäume ausgetauscht werden sollen, direkt gesendet werden können und nicht durch alle Abstraktionsschichten laufen müssen. Die Konfiguration der Nachrichtenwege findet zwar auf der Basis der Funktionalität der gesamten zusammengesetzten Objekte statt, die Nachrichtentransaktionen aber werden danach nur in der alleinigen Verantwortung der betreffenden Teilobjekte durchgeführt.

Ein wichtiger Aspekt ist hierbei sicherlich die Spezifikation einer solchen Konfiguration. Die Konfiguration ist sogar in der Lage, aufgrund des dynamischen

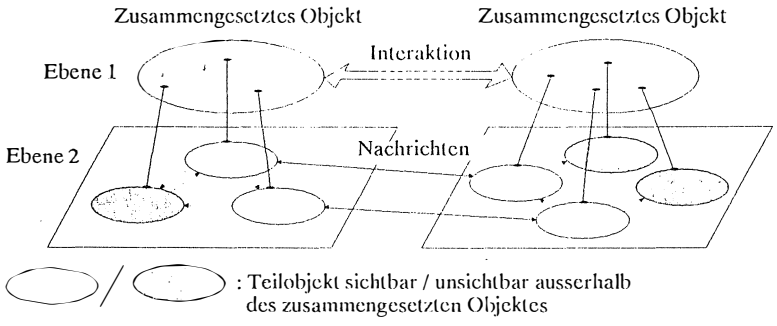

Bild 5. Interaktion zwischen zwei zusammengesetzten Objekten

Charakters des objekt-orientierten Ansatzes sich im Laufe der Zeit zu verändern. Auf diese leichte Änderbarkeit werden u.a. die Weichen zur Konstruktion flexibler Systeme gestellt. Dafür müssen wir aber die Übersicht behalten und uns mit der erhöhten Komplexität auseinandersetzen. Dazu werden strukturierte Mittel und Methoden gebraucht; das Konzept der zusammengesetzten Objekte soll ein solches Mittel darstellen. Die parallelen, echtzeitkonformen, vom Benutzer des zusammengesetzten Objektes zum großen Teil abgekapselten Aktivitätsentfaltungen, die gleichzeitig ein hohes Maß an Objektintegrität besitzen, waren die größte Motivation für die Einführung und die Entwicklung des Begriffes der zusammengesetzten Objekte in diesem Modell.

5. Schlußbemerkungen

Es ist sicherlich sehr interessant, ein komplettes Echtzeitsystem in Termen von aktiven, autonomen und parallel auf das gesamte verteilte System wirkende Einheiten zu modellieren, zu programmieren und zu realisieren. Der objekt-orientierte Ansatz bietet vielversprechende Grundlagen zur Realisierung einer solchen Vorgehensweise. In unseren Beitrag haben wir unsere Vorstellungen darüber erläutert, in welcher Art sich ein objekt-orientiertes Modell in der Prozeßdatenverarbeitung einsetzen ließe. Dazu reichen die heute übliche Modelle nicht aus, die eher für den Bereich der symbolischen Verarbeitung ausgelegt sind. Wichtig für die erfolgreiche Einführung eines Ansatzes in die Praxis ist allerdings u.a. auch seine Effizienz sowohl bzgl. der Modell- und Programmentwicklung aber auch bzgl. der Programmausführung. Für die erste Komponente bieten die spezifischen Eigenschaften des objekt-orientierten Ansatzes eine hervorragende Grundlage. Für die zweite

Komponente ist es vor allem wichtig, daß es eine effiziente Kooperation zwischen der Software und der zugrundeliegenden Hardware existiert, da nur auf diese Weise besonders elegante und effiziente Systeme entstehen können. In diesem Sinn arbeiten wir gleichzeitig an einer objekt-orientierten Architektur, die den grundlegenden Rahmen für die effiziente Ausführung unseren Modells zur Verfügung stellen soll /14/.

6. Literaturverzeichnis

1. Wegner, P., Shriver, B. (Eds.): Object-Oriented Programming Workshop, IBM Yorktown Heigts, 9-13 June 1986, SIGPLAN Notices, Vol. 21, No. 10, Oct. 1986
2. Meyrowitz, N. (Ed.): Proc. ACM Conf. on Object-Oriented Systems, Languages, and Applications. Orlando, Florida, 1987. Special Issue of SIGPLAN Notices, Vol. 22, No. 12, Dec. 1987
3. Cook, S.: Languages and object-oriented programming. Software Engineering Journal, March 1986, 73-80
4. Cox, B.J.: Object-Oriented Programming: An Evolutionary Approach. Reading, Mass. Addison-Wesley Publishing Company, 1986
5. Nygaard, K.: Basic Concepts in Object Oriented Programming. SIGPLAN Notices, Vol. 21, No. 10, Oct. 1986, 128-132
6. Levy, H.M.: Capability-Based Computer Systems. Bedford, Mass.: Digital Equipment Corp., 1984
7. Organick, E.I.: A Programmer's View of the Intel 432 System. New York, NY, McGraw-Hill Book Company, 1983
8. Goldberg, A., Robson, D.: Smalltalk 80: The Language and its Implementation. Reading, MA: Addison-Wesley, 1983
9. Matsumoto, Y.: Requirements Engineering and Software Development. In: Güth, R. (Ed.): Computer Systems for Process Control, New York: Plenum Press, 1986, 241-264
10. Pleßmann, K.W., Tassakos, L.: Concurrent, object-oriented program design in real-time systems. Proc. EUROMICRO 88, Microprocessing and Microprogramming, Vol. 24, Nrs. 1-5, 1988, 257-265
11. Yonezawa, A., Shibayama, E., Takada, T., Honda, Y.: Modelling and Programmming in an Object-Oriented Concurrent Language ABCL/1. In: Object-Oriented Concurrent Programming, Yonezawa, A., Tokoro, M. (Eds.), Cambridge, Mass., London, The MIT Press (Series in Computer Sciences), 1987, 55-89
12. Stefik, M., Bobrow, D.G.: Object-Oriented Programming: Themes and Variations. AI Magazin, Jan. 1986, 40-62
13. Banerjee, J. et al.: Data Model Issues for Object-Oriented Applications. ACM Trans. on Office Information Systems, Vol. 5, No. 1, Jan. 1987, 3-26
14. Tassakos, L., Pleßmann K.W.: pdvPool: A real-time object-oriented multiprocessor system. Presented in Short Notes of EUROMICRO 88, will published in a special issue of "Microprocessing and Microprogramming", the Euromicro Journal, in spring 89.