Objektorientierung in Parallelen Betriebssystemen

Wolfgang Schröder-Preikschat
GMD FIRST an der Technischen Universität Berlin
Hardenbergplatz 2, W-1000 Berlin 12
schroeder-preikschat@kmx.gmd.dbp.de

28. Februar 1992

Vorwort

Objektorientierung ist eines der Schlagworte der achtziger Jahre in der Informationstechnologie. Das Objektparadigma übt damit auch zunehmenden Einfluß auf den noch relativ jungen Bereich der parallelen Betriebssysteme aus. Daß Objektorientierung in diesem Kontext die geeignete Form des Entwurfs und der Implementierung flexibler und dennoch leistungsfähiger Systemsoftware darstellt, versucht der vorliegende Beitrag am Beispiel der PEACE Betriebssystementwicklung zu demonstrieren.

1 Einleitung

Um darzustellen, was Objektorientierung in parallelen Betriebssystemen bedeutet, sind zuerst einmal die beiden in diesem Thema enthaltenen Begriffe für sich allein und hinsichtlich ihres Ursprungs zu beleuchten. Für den Begriff der *Objektorientierung* gibt es die Definition [8], die besagt, daß Objekte Instanzen von Klassen darstellen und daß darüberhinaus Vererbung einzusetzen ist, um Klassenhierarchien zu bilden. Klassen beschreiben die Implementierung abstrakter Datentypen, deren Komposition durch Vererbung erfolgt. Danach bedeutet Objektorientierung dann im einzelnen:

$$\textit{Abstraktion} + \textit{Klassen} + \textit{Vererbung}.$$

Die Hierarchiebildung dient der Spezialisierung sogenannter Basisklassen (auch Oberklassen oder Elternklassen). Basisklassen legen grundlegende Objekteigenschaften fest und werden durch Vererbung an konkrete Anwendungsbedürfnisse angepaßt. Derart spezialisierte, d.h. abgeleitete Klassen (auch Unterklassen oder Kindklassen) erben damit alle Eigenschaften ihrer Basisklassen und legen zusätzliche Eigenschaften fest. Unterschieden wird hierbei zwischen Einfachvererbung, bei der immer nur eine Basisklasse zur Komposition der neuen Hierarchiestufe herangezogen wird, und der Mehrfachvererbung, die mehrere Basisklassen vorsieht [4].

Ein objektorientiertes System besteht damit nicht nur aus einer Vielzahl von Objekten. Es stellt vielmehr auch und gerade Abstraktionen in Form von Klassen zur Verfügung, die benutzergerecht spezialisierbar sind. Objektorientiert bedeutet vor allen Dingen Vererbung und damit Spezialisierung. Systeme, die diese Eigenschaft nicht aufweisen, sind damit nicht objektorientiert, sondern wären höchstens klassenbasiert (abstrakte Datentypen) oder objektbasiert (Datenabstraktion) [1].

Parallele Betriebssysteme sind speziell für Parallelrechner entwickelt worden und hier für solche, die auf dem Prinzip des verteilten Speichers basieren. Diese Rechner stellen extrem leistungsfähige verteilte Systeme dar, bei denen die Sichtweise eines gemeinsamen, globalen Speichers physikalisch nicht mehr gegeben ist. Parallele Betriebssysteme sind damit auch notwendigerweise verteilte Betriebssysteme.

Paralleles Rechnen dient vornehmlich dem Ziel der Leistungssteigerung. Es ist naheliegend, daß dies auch eine Zielvorstellung der dann zur Problemlösung einzusetzenden Komponenten Rechner, Betriebssystem, Übersetzer und Sprache sein muß. Für die Betriebssysteme gelten etwa auf Grundlage eines 40 MIPS Prozessors Leistungsanforderungen, die das Aufsetzen der netzwerkweiten Nachrichtenkommunikation in der Größenordnung von 10 μsec vorsehen. Der lokale Verlust an Skalarleistung für das Anwendungsprogramm muß auf das absolute Minimum reduziert werden. Die obere Schranke für den in der Natur der Sache liegenden Leistungsschwund muß in einem "vernünftigen" Verhältnis zur Latenzzeit bei der netzwerkweiten Kommunikation liegen. Dies sind enorme Leistungsanforderungen, die für verteilte Betriebssysteme in der Form nicht gelten (bzw. bei weitem nicht erreicht werden), sich aber sehr wohl fördernd auswirken würden.

Der vorliegende Beitrag behandelt Objektorientierung in parallelen Betriebssystemen am Beispiel PEACE. Zunächst wird erläutert, was unter einem objektorientierten Betriebssystem überhaupt zu verstehen ist. Danach wird demonstriert, wie Objektorientierung in PEACE zum Einsatz kommt.

2 Objektorientierte Betriebssysteme

Die dem Kontext von Programmiersprachen entnommene Klassifikation [8] eignet sich sehr gut, um eine klare Position für PEACE in der Welt sogenannter objektorientierter Betriebssysteme zu finden, ohne dabei auf neue Definitionen zurückgreifen zu müssen. In letzter Zeit ist hier eine rege Diskussion darüber entbrannt, ob und ggf. welches Betriebssystem als objektorientiert gilt. Hier sei nur auf USENET verwiesen, das ein nahezu unerschöpfliches Reservoir an Definitionen zu Tage bringt. Alle Welt spricht von Objektorientierung und viele lassen dabei die ursprüngliche Bedeutung außer Acht. Ein Blick in artverwandte Fachgebiete hat jedoch auch im Betriebssystembereich noch nie geschadet.

Die Diskussion entbrannte u.a. um Mach und Chorus, die derzeit erfolgreichsten Vertreter auf Mikrokernen basierender Betriebssysteme. Während die Chorus Entwickler zögern, ihr System objektorientiert zu klassifizieren, ist diese Zurückhaltung bei Mach nicht spürbar. Mach wird als objektorientiert postu-

liert, entspricht jedoch höchstens den Anforderung eines objektbasierten Systems. Objektbasiert sind nahezu alle Betriebssysteme moderner Ausprägung, d.h., die sogenannten prozeßstrukturierten Systeme. Systemprozesse stellen Dienstleistungen zur Verfügung und treten als aktive Objekte in Erscheinung. Der interne Zustand des Systemprozesses (aktiven Objektes) kann dabei nur über Systemaufrufe (Operationen bzw. Methoden) verändert werden. Ein *File Server* ist z.B. als typischer Vertreter solcher aktiven Objekte zu nennen.

Einmal von dieser gegenwärtig nicht unbedingt als konstruktiv zu bezeichnende Diskussion um objektorientierte Betriebssysteme abgesehen, sind sowohl bei Mach und Chorus Signale wahrnehmbar, die die Weichen eindeutig in Richtung Objektorientierung stellen. Inwieweit dies tatsächlich mit den (auch kommerziell) etablierten Systemen im Nachhinein erfolgreich sein wird, bleibt abzuwarten. Objektorientierung bedeutet ja nicht nur, z.B. alles von C in C++ umzukodieren bzw. ein System von Anfang an in C++ zu programmieren. Das System muß vor allen Dingen objektorientiert entworfen werden. Die Wahl der Implementierungssprache ist dann erst der zweite Schritt. Wenn sowohl Mach als auch Chorus also als objektorientierte Betriebssysteme postuliert werden wollten, müßten Entwurf und Umsetzung entsprechend ausgelegt sein. Dies kommt zwangsläufig einer völligen Neustrukturierung und auch -implementierung gleich, wenn nicht bereits vorher der objektorientierte Systementwurf vorlag. Aber genau *so* ist die gegenwärtige Situation beider Systeme zu umschreiben.

Was ist ein objektorientiertes Betriebssystem wirklich? Nun, es muß sich aus Abstraktionen zusammensetzen, die für den Anwender spezialisierbar sind. Es muß darüberhinaus Mechanismen zur Verwaltung großer Objektbestände zur Verfügung stellen. Abstraktionen sind Klassen - und Klassen werden durch Vererbung spezialisiert. Ein Betriebssystem, das streng objektorientiert entworfen und implementiert worden ist, muß damit auch zwangsläufig über Mechanismen zur Verwaltung großer Objektbestände verfügen. Diese Bestände setzen sich aus den internen Betriebssystemobjekten zusammen. Die Objektverwaltung wird selbst über eine Klasse beschrieben. Damit besteht das Betriebssystem ausschließlich aus einer Vielzahl von Klassen, von denen eine ausgewählte Menge auch für den Anwender spezialisierbar sind. Diese Klassen definieren dann die Betriebssystemschnittstelle.

Ein objektorientiertes Betriebssystem ist also objektorientiert entworfen und implementiert und es bietet eine objektorientierte Schnittstelle an. Es ist damit offen und anwendungsbezogen, weil spezialisierbar. Nur sehr wenige moderne Betriebssysteme genügen diesen Anforderungen, so z.B. Choices [2] und auch PEACE [7].

3 Objektorientierung vs. Programmfamilien

PEACE ist ursprünglich als objektbasiertes Betriebssystem für SUPRENUM entwickelt worden. Erst mit Beendigung dieses Projektes begann für PEACE die Ära der Objektorientierung. Basierend auf den mit der SUPRENUM Entwicklung gesammelten Erfahrungen, erfolgte eine konsequente Neustrukturierung

nach objektorientierten Entwurfsprinzipien. Objektorientierung in PEACE hat also nicht bedeutet, dem System nur eine objektorientierte Oberfläche zu verschaffen - es ist damit kein *object support* System wie z.B. COOL, dem *Chorus object-oriented layer*. Auch das Innenleben von PEACE gestaltet sich grundsätzlich anders als in der ersten Version.

Ohne Abstriche beibehalten wurde das für den ursprünglichen Entwurf von PEACE maßgebliche Konzept der Programmfamilien [6]. Dieses Konzept zielt auf das Maßschneidern, die Erweiterbarkeit und Wartbarkeit von Softwaresystemen allgemein und Betriebssystemen speziell. Es ist damit bereits objektorientierten Ideen sehr nah. Die Distanz ist sogar derart gering, daß von einer Renaissance des Familienansatzes für den Betriebssystembereich gesprochen werden kann [3]. Der Begriff der minimalen Basis von Systemfunktionen spiegelt sich direkt im Konzept der Basisklassen wider und der Begriff der minimalen Systemerweiterung findet sein Gegenüber in der Vererbung und der dadurch ermöglichten Vielgestaltigkeit (Polymorphismus) eines objektorientierten Systems. Auch eine Betriebssystemfamilie ist vielgestaltig.

Damit hat sich der Übergang für PEACE von einem objektbasierten zu einem objektorientierten System sanft gestaltet. Die wesentlichen Änderungen bezogen (und beziehen) sich tatsächlich auch auf den Akt der Programmierung, indem einerseits C++ anstatt C konsequent zum Einsatz kam und sich andererseits die Implementierung der Komponenten (vormals Moduln nun Klassen) einzelner Familienmitglieder neu gestaltete. Das globale Weltbild blieb mit Ausnahme der dann auch objektorientierten Systemschnittstelle gleich.

Die Vorteile des objektorientierten Modells bei diesem "PEACE *lifting*" waren (und sind) tiefgreifend. Eine wesentliche Erkenntnis aus SUPRENUM besagt, daß mikrokernbasierte Betriebssystemstrukturen nicht das *non plus ultra* zur Ausführung paralleler Programme darstellen. Dies vorwiegend deshalb, weil aufgrund des Ansatzes, Dienstleistungen in eigene Prozeßadreßräume abzukapseln, die Mikrokerne immer klassischen Mehrprogrammbetrieb implementieren. Das bedeutet nicht nur die individuelle (horizontale) Isolation der untereinander konkurierenden Prozesse sondern auch die (vertikale) Isolation des Mikrokerns von diesen Prozessen. Dieser Ansatz erzwingt auch gerade dann diese selbstgeschaffenen Grenzen, wenn auf jedem Knoten des Parallelrechners nur ein einziger Anwendungsprozeß ablaufen soll. Auch wenn sich Anwendungsprozesse den Knoten nicht mit Systemprozessen teilen müssen, besteht immer noch die Grenze zwischen dem Anwendungsprogramm und demselben mehrprogrammfähigen Mikrokern. Dies hat enorme Konsequenzen hinsichtlich der Dauer des vom Anwendungsprogramm angestoßenen Aufsetzvorgangs der netzwerkweiten Kommunikation. Vertikal gesehen, laufen für solche Konfigurationen sehr viel unnötige Aktivitäten beim Eintritt in den Mikrokern und in ihm selbst ab. Das Resultat sind lokale Leistungsverluste, die sich negativ auf die Leistungsfähigkeit des gesamten Systems auswirken.

Nun sind solche Konfigurationen nicht unbedingt die Seltenheit, wie zumindest die SUPRENUM Erfahrung zeigt. Gut unterstützt werden mit dem Mikrokernansatz alle Anwendungen, deren Prozesse nicht eins-zu-eins auf Prozessorknoten abbildbar sind. Andere Anwendungen werden benachteiligt - und dies widerspricht dem Familiengedanken. Eine objektorientierte Systemschnittstelle

und -implementierung führt auf jedem Fall dazu, daß sich die Systemgrenzen durch den Mechanismus der Vererbung, d.h., vermöge von Klassenhierarchien, fließend gestalten. Hier ist gerade der Vorteil von C++ als objektorientierter Implementierungssprache zu sehen, bei der nahezu keine "Erbschaftssteuer" zu zahlen ist. Wie hoch sich der Aufwand durch die Klassenhierarchie wirklich erweist, ist im wesentlichen davon abhängig, ob die Klassen auf dynamisches Binden von Methoden hinweisen. Ohne dynamisches Binden kann mit C++ sogar eine "Steuerrückzahlung" erreicht werden, was bedeuten soll, daß eine effizientere und darüberhinaus auch noch erheblich besser strukturierte Implementierung möglich ist. Dadurch, daß die Operationen der Klassen als *inline* Methoden auslegbar sind, können die nötigen "Steuerabzüge" geltend gemacht werden.

4 Systemtransparenz

Die PEACE Systemschnittstelle wird ausschließlich über Klassen definiert. Um Dienstleistungen von PEACE in Anspruch zu nehmen, werden diese Klassen über Vererbung anwendungsbezogen spezialisiert oder so genommen, wie sie sind. Es muß aber in jedem Fall eine oder mehrere (PEACE) Instanzen von der betreffenden Klasse geschaffen werden, bevor Interaktionen mit dem System stattfinden können. Diese Objekte sind damit auch direkter Bestandteil des Anwendungsprogramms und, was von wesentlicher Bedeutung für die Programmierbarkeit des Systems ist, es sind Objekte, mit denen das Anwendungsprogramm auch im Falle eines nicht objektorientierten Systems arbeiten müßte.

Zur Verdeutlichung, wie in PEACE Objektorientierung zum Einsatz kommt, sei z.B. die Modellierung leichtgewichtiger Prozesse betrachtet. Hierbei handelt es sich allgemein um Prozesse, die sich mit anderen Prozessen denselben Adreßraum teilen. Es ist oft festzustellen, daß diese Prozesse neben dem im Stapelspeicher gehaltenen, lokalen Datenbereich auch über Daten des globalen Adreßraums bzw. (dynamische) Daten verfügen, die der Halde entnommen wurden. Diese sogenannten *own* Variablen sind geeignet dafür, in Klassen abgekapselt zu werden und sich von diesen Klassen dann **aktive Objekte** zu schaffen, die nichts anderes als leichtgewichtige Prozesse darstellen. Eine **autonome Methode**, d.h. eine Methode ohne Signatur, könnte den Prozeßrumpf repräsentieren und damit die vom Prozeß auszuführenden Instruktionssequenzen beschreiben (Bild 1).

Die autonome Methode (**action()**), sofern sie gestartet wird, hat implizit Zugriff auf die Daten des Objektes, auf das sie angewendet wird. Sie wäre dem System sinnvoll über **dynamisches Binden** bekannt zu geben. Die Spezialisierung einer Systemklasse ist die Folge. Das System definiert eine **virtuelle Funktion** (eben **action()**), die dann aktiviert wird, wenn ein Objekt dieser Klasse erzeugt worden ist. Diese Funktion wird im Zuge der Vererbung auf Benutzerebene redefiniert und damit dynamisch dem System bekannt gegeben. Prozesse werden dann rein deklarativ erzeugt, einzeln oder gar als Felder (Bild 2). Genau so stellt sich die Modellierung von Prozeßstrukturen in PEACE dar.

Das in PEACE umgesetzte Prozeßmodell hat den enormen Vorteil, daß dem

```
#include "thread.h"

class myThread : public thread {
    void action();      // process body
    ...                 // any private/protected code/data
public:
     myThread();        // default process construction
     myThread(...);     // parameterized process construction
    ~myThread();        // process destruction
    ...                 // any other public code/data
};
```

Abbildung 1: Definition eines aktiven Objektes

System wesentliche Unterstützung zur kontrollierten Entsorgung des Parallelrechners gegeben wird. In C++ können klassenbezogene Konstruktor- und Destruktormethoden definiert werden, deren Aufruf automatisch bei der Erzeugung bzw. Zerstörung eines Objektes dieser Klasse erfolgt. Über den Konstruktor können dem Prozeßobjekt dann initiale Daten übergeben werden. Die Signatur des Konstruktors ist dabei frei wählbar. Der **thread** Destruktor z.B. sorgt implizit für die Betriebsmittelfreigabe, wenn ein aktives Objekt zerstört wird. Dies ist von unschätzbarem Wert, wenn an massiv parallele Systeme gedacht wird, die aus mehreren tausend autonomer Prozessoren und einer durchaus noch größeren Anzahl von Prozessen bestehen können.

Ein zweites Beispiel betrifft die Interprozeßkommunikation. Die zu sendende oder zu empfangende Nachricht ist hier ein sehr einleuchtendes Beispiel dafür, daß eine Anwendung implizit mit Systemobjekten umgehen muß (Bild 3).

Um eine Nachricht senden zu wollen, muß dem System der betreffende Speicherplatz bekannt gegeben werden, in dem diese Nachricht liegt. Die Nachricht muß also als Objekt bereits vorhanden sein. Der Sendevorgang bedeutet sodann, eine entsprechende Methode auf dem betreffenden Objekt anzuwenden. Für den Empfangsvorgang gilt ähnliches wie für den Sendevorgang (Bild 4).

Beide Beispiele zeigen, daß im objektorientierten Ansatz eine sehr enge Bindung zwischen Anwendung und Betriebssystem allgemein, bzw. PEACE speziell, erreicht wird. Es besteht zunächst kein Zwang, eine schwer zu überwindende Grenze künstlich festschreiben zu müssen. In PEACE wird die Anwendung grundsätzlich als Systemspezialisierung aufgefaßt - im Sinne des Familiengedankens ist sie die natürliche Erweiterung des Systems überhaupt. Ob eine Grenze tatsächlich zu ziehen ist, etwa im Sinne eines Mikrokernansatzes, bleibt allein eine Konfigurationsentscheidung.

5 Ortstransparenz

Ob PEACE als objektorientierte Laufzeitumgebung dem Adreßraum des Anwendungsprogramms als Bibliothek zugeordnet ist oder die aktiven Systemobjekte

```
#include "myThread.h"

myThread two[2];          // 2 static processes

void main(...) {
    myThread  one(...);   // 1 temporary process
    myThread  six[6];     // 6 temporary processes
    myThread* son;        // a dynamic process
    ...
    son = new myThread(...);
    ...
    delete son;           // destruct dynamic process
    ...
}                         // destruct temporaries
                          // destruct static processes
```

Abbildung 2: Instanzenbildung aktiver Objekte

isoliert vorliegen (wie im Mikrokernansatz), ist eine Frage der Systemkonfiguration. Auch diese Form der Vielgestaltigkeit ist im PEACE Familienkonzept von zentraler Bedeutung. Wie sich das System nach außen hin darstellt, ist überwiegend transparent für die Anwendung. Erreicht wird diese Konfigurationsfähigkeit einerseits durch die Organisation und Struktur von PEACE, andererseits werden sprachliche Hilfsmittel eingesetzt, um Objekte bestimmter Klassen als nicht im eigenen Adreßraum vollständig resident auszeichnen zu können [5].

Durch Annotationen von Klassen, Methoden und Argumenten werden Hinweise darauf gegeben, daß Objekte dieser Klassen nicht lokal (d.h., im anderen Adreßraum oder auf einem anderen Knoten) vorhanden sind. Diese Annotationen sind softwaretechnisch verträglich ausgelegt, d.h., die annotierten Klassen können von jedem C++ Übersetzer verarbeitet werden und erlangen nur ihre Bedeutung bei Anwendung des P++ Klassengenerators [5]. Objektgebundene Fernaufrufe sind dann die Folge, um Instanzen annotierter Klassen zu konstruieren, zu zerstören und um auf ihnen Methoden anzuwenden.

Ein solches Objekt wird in PEACE als **duales Objekt** [5] bezeichnet. Die Dualität kommt dadurch zustande, daß eine zweigeteilte Repräsentation des Objektdatenraums herbeigeführt wird. Die öffentlichen Daten (**public** und **protected** in C++) sind immer im Kontext der Anwendung direkt zugänglich. Wohingegen die nicht-öffentlichen Daten (**private** in C++) dort verbleiben, wo das Objekt tatsächlich physikalisch abgelegt worden ist. Der Zugriff auf den privaten Objektdatenbereich muß über Methoden erfolgen, die dann bei dualen Objekten in Fernaufrufe automatisch umgesetzt werden.

Der öffentliche Objektdatenraum wird über ein Vertreterobjekt (**Proxy**) nach der Konstruktion nach außen bekannt gegeben. Es liegt in der eigenen Entscheidung derjenigen Instanz, die ein duales Objekt deklariert hat und daraufhin ein Proxy erhält, dieses Proxy an dritte weiterzugeben. Der objektgebundene

```
#include "client.h"
#include "server.h"
#include "parcel.h"

void client::action() {
    server  sink;                   // create receiver
    opaque* peer;                   // endpoint identifier
    parcel  data;                   // query/reply message
    ...
    peer = sink->gateway();         // where to go
    for (;;) {
        ...
        peer = data.request(peer);  // ask for a reply
        ...
    }
}
```

Abbildung 3: Sender als aktives Objekt

Fernaufruf sorgt immer dafür, daß öffentliche und nicht-öffentliche Objektdaten verschmolzen werden und damit die Konsistenz des gesamten Objekts wieder hergestellt ist. Diese Konsistenzerhaltung wird nicht erzwungen, wenn ein Proxy an dritte weitergegeben wird. Es hängt letztlich vom Anwendugsprofil ab, ob Konsistenz immer zu garantieren ist.

Die dualen Objekte sind der Schlüssel in PEACE, um auch tatsächlich objektorientiert in verteilten Umgebungen arbeiten zu können. Vererbung wird in PEACE netzwerkweit unterstützt, ohne auch hier mit überhöhter "Erbschaftssteuer" rechnen zu müssen.

6 Zusammenfassung

Objektorientierung in parallelen Betriebssystemen bedeutet, daß Entwurf, Implementierung und Schnittstelle objektorientierten Prinzipien genügen müssen. Das heißt aber in erster Linie Vererbung auch dazu einsetzen zu können, um Systemschnittstellen anwendungsbezogen spezialisieren zu können. In verteilten Umgebungen, wie im Falle massiv paralleler Systeme, ist Vererbung darüberhinaus netzwerkweit zu unterstützen.

PEACE entspricht diesen Anforderungen und ist ein Beispiel dafür, wie Objektorientierung und das Konzept der Programmfamilien erfolgreich eingesetzt werden kann, um flexible und leistungsfähige Systemstrukturen aufzubauen.

Objektorientierung wird auch verschiedentlich daran gemessen, aus wieviel Klassen das System besteht. Nun, der PEACE Kern umfaßt gegenwärtig 116 Klassen, ca. 3000 Zeilen (incl. Kommentare), wovon 5 Klassen die Systemschnittstelle darstellen. Die tiefste Vererbungshierarchie, zur Modellierung des Prozeßkontrollblocks, besteht dabei aus 10 Stufen mit insgesamt 17 Klassen. Der

```
#include "server.h"
#include "ticket.h"
#include "parcel.h"

void server::action() {
    ticket  gate;                       // communication handle
    opaque* peer,self;                  // endpoint identifiers
    parcel  data;                       // query/reply message
    ...
    self = (*this)->gateway();          // where am i
    for (;;) {
        ...
        peer = data.receive(gate);      // accept query
        ...
        (void)data.respond(peer,self); // return reply
        ...
    }
}
```

Abbildung 4: Empfänger als aktives Objekt

aus den Klassendefinitionen herausgezogene Programmtext beläuft sich auf ca. 3800 Zeilen (incl. Kommentare).

Der Kern und das gesamte ROI System laufen derzeit als Gastebene oberhalb von SunOS und werden u.a. in der Lehre eingesetzt. Die Gastebene dient auch als Plattform zur Integration von virtuell gemeinsamem Speicher in PEACE. Erste Portierungsschritte für den auf i860 Prozessoren basierenden MANNA Rechner sind erfolgreich durchgeführt worden. Die Portierung auf die beim FIRST befindlichen SUPRENUM und Transputer Systeme ist im Gange. Implementierungsarbeiten zur dynamischen Rekonfiguration des Systems und zum inkrementellen Laden stehen jetzt an. Mit Konzeptarbeiten zur Persistenz von Objekten wurde begonnen.

Literatur

[1] G. S. Blair, J. J. Gallagher, J. Malik, "Genericity vs Inheritance vs Delegation vs Conformance vs...", *Journal of Object-Oriented Programming*, Vol. 2, No. 3, pp. 11-17, 1989

[2] R. Campbell, G. Johnston, V. Russo, "Choices (Class Hierarchical Open Interface for Custom Embedded Systems)", *ACM Operating Systems Review*, 21, 3, pp. 9-17, 1987

[3] J. Cordsen, W. Schröder-Preikschat, "Object-Oriented Operating Systems Design and the Revival of Program Families", Proceedings of the Second

International Workshop on Object Orientation in Operating Systems, IEEE 91TH0392-1, pp. 24-28, Palo Alto, CA, October 17-18, 1991

[4] D. C. Halbert, P. D. O'Brien, "Using Types and Inheritance in Object-Oriented Languages", *IEEE Software*, 9, pp. 71-79, 1987

[5] J. Nolte, "Language Level Support for Remote Object Invocation", Technical Report, GMD FIRST, Berlin, Germany, 1991

[6] D. L. Parnas, "Designing Software for Ease of Extension and Contraction", *IEEE Transaction on Software Engineering*, Vol. SE-5, No 2, 1979

[7] W. Schröder-Preikschat, "Scalable Operating System Design", Technical Report, GMD FIRST, Berlin, Germany, 1991

[8] P. Wegner, "Classification in Object-Oriented Systems", *SIGPLAN Notices*, 21, 10, pp. 173-182, 1986