Gesellschaft
für Informatik e.V.

Leitung der Fachgruppe 3.1.4
Betriebssysteme

Einladung

zum Frühjahrstreffen
am 16./17. März 1998 in Berlin

Die Schwerpunkte des Treffens sind diesmal:

"Betriebssysteme für Eingebettete Systeme" und
"Objektorientierung in Betriebssystemen"

Zur Teilnahme füllen Sie bitte das beiliegende Anmeldeformular aus und schicken es bis spätestens

2. 3. 1998

an den lokalen Organisator

Friedrich Schön
GMD FIRST
Rudower Chaussee 5
D-12489 Berlin

Tagungsort: GMD FIRST
Rudower Chaussee 5
D-12489 Berlin
Gebäude 13.10
Seminarraum 006
Beginn: 16. 3. 1998, 13.00 Uhr
Ende: 17. 3. 1998, 16.00 Uhr

Hotelreservierungen müssen selbst vorgenommen werden. Für weitere Hotelhinweise und An- und Abreisemöglichkeiten steht Ihnen die Web-Seite `http://www.first.gmd.de/~fs/gifg314` zur Verfügung. Herr Schön (Tel. `030-6392-1838`) oder Herr Dr. Nolte (Tel. `030-6392-1841`) helfen Ihnen bei Fragen gerne weiter. Die Fax-Nr. in beiden Fällen lautet `030-6392-1805`.

Gesellschaft
für Informatik e.V.

Fachgruppe 3.1.4
Betriebssysteme

Herrn
Friedrich Schön
GMD FIRST
Rudower Chaussee 5
D-12489 Berlin

Anmeldung

zum Treffen der GI-Fachgruppe "Betriebssysteme" am 16./17. März 1998 in Berlin.

Name: ....
Vorname: ....
Institution: ....
Anschrift: ....

Tel.: .... Fax.: ....
E-Mail: ....

Ankunft: .... Abreise: ....

(Hiermit ist keine Hotelreservierung verbunden)

Unterschrift: .... Datum: ....

Gesellschaft für Informatik e.V.

Leitung der Fachgruppe 3.1.4
Betriebssysteme

Programm

Montag, 16. März 1998	
13:00 – 13:45	**OSEK/VDX: Ein Betriebs- und Kommunikationssystem für den Einsatz in Kraftfahrzeugen**, R. Kern, Georg-Simon-Ohm-Fachhochschule Nürnberg
13:45 – 14:30	**Ein Kommunikationsrahmenwerk als Teil eingebetteter Systeme**, F. Schön, GMD FIRST
14:30 – 15:00	Kaffeepause
15:00 – 15:45	**Generische Komponenten zur Konstruktion anwendungsangepaßter Laufzeitplattformen**, L. Baum, M. Becker, L. Geyer, G. Molter, Universität Kaiserslautern
16:30 – 17:15	**Werkzeuggestützte Konfigurierung und Adaptierung**, R. Meyer, Universität Potsdam
17:15 – 18:00	**Werkbank zum Bau maßgeschneiderter Betriebssysteme**, D. Beuche, U. Haack, W. Schröder-Preikschat, O. Spinczyk, Otto-von-Guericke-Universität Magdeburg
19:00 – ??:??	Abendessen
Dienstag, 17. März 1998	
08:30 – 09:15	**Experiences with a PC SMP Cluster**, A. A. Fröhlich, GMD FIRST
09.15 – 10:00	**Die Erforschung der Langsamkeit: Vernachlässigte System-Resourcen und ihre Beherrschung**, F. Bellosa, Universität Erlangen-Nürnberg
10:00 – 10:30	Kaffeepause
10:30 – 11:15	**Reflexion als Strukturierungsprinzip objektorientierter verteilter Systeme: Das metaXa Projekt**, M. Golm, J. Kleinöder, S. Reitzner, Th. Riechmann, Universität Erlangen-Nürnberg
11:15 – 12:00	**Objekte in Windows-NT**, W. Kalfa, Technische Universität Chemnitz-Zwickau
12:00 – 12:45	**A Topology-Based Approach to Coordinated Multicast Operations**, J. Nolte, GMD FIRST
12:45 – 14:00	Mittagspause
14:00 – 16:00	Sitzung der Fachgruppe

Kurzfassungen

der Beiträge zum Frühjahrstreffen
am 16./17. März 1998 in Berlin

1. **OSEK/VDX: Ein Betriebs- und Kommunikationssystem für den Einsatz in Kraftfahrzeugen**, R. Kern, Georg-Simon-Ohm-Fachhochschule Nürnberg

   OSEK (Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug) = OSAN (Open Systems in Automotive Networks). VDX (Vehicle Distributed Executive, französisches Partnerprojekt).

   Bei OSEK handelt es sich um ein Gemeinschaftsprojekt von Automobilindustrie, Hochschulen, Hardware- und Software- Herstellern mit dem Ziel, einen Industrie-Standard für eine „offene" (herstellerunabhängige) Architektur für verteilte Systeme zur Steuerung von Fahrzeugen zu entwickeln. Die Implementierung der spezifizierten Schnittstellen bleibt frei und ist damit dem Wettbewerb unterworfen. Sie soll mit minimalem Hardware-Aufwand (insbesondere beim RAM) auskommen, aber engen Zeitbedingungen genügen können. Die OSEK-Spezifikation umfaßt drei Hauptbestandteile:

   (a) OSEK-OS (OSEK operating system): Konzept und Aufrufschnittstelle (API) für ein spezialisiertes (lokales) Echtzeit-Betriebssystem. Hauptmerkmale:
      - Nur für eingebettete (statische) Anwendungssysteme gedacht, daher
      - keine dynamische Betriebsmittelverwaltung,
      - sehr stark eingegrenzter Funktionsumfang,

- eingeschränkte Fehlerbehandlung,
- stattdessen auf geringen Speicherbedarf und schnelle Reaktionsfähigkeit getrimmt,
- flexible Konfigurationsmöglichkeiten,
- Erleichterung der Portierbarkeit auf Anwendungsebene durch vorgegebene Konformitätsklassen.

(b) OSEK-COM (OSEK communication): Kommunikationssystem auf Feldbusbasis im Kraftfahrzeug (in Anlehnung an den CAN-Bus, aber auch für andere Bussysteme offen). In diesem Teil ist auch die lokale Interprozeßkommunikation enthalten (Ortsunabhängigkeit).

(c) OSEK-NM (OSEK network management): Netz-Management auf der Basis von OSEK-COM.

Die OSEK-Komponenten sollen durch komfortable Werkzeuge für

- Entwicklung,
- Konfiguration,
- statische Fehlererkennung und
- Test und Simulation

ergänzt werden, die aber durch OSEK nicht näher spezifiziert sind.

2. **Ein Kommunikationsrahmenwerk als Teil eingebetteter Systeme**, F. Schön, GMD FIRST

   Beitrag wird nachgeliefert.

3. **Generische Komponenten zur Konstruktion anwendungsangepaßter Laufzeitplattformen**, L. Baum, M. Becker, L. Geyer, G. Molter, Universität Kaiserslautern

   Betriebssysteme bilden die Realisierungsgrundlage für annähernd jede Art von Softwarelösung. Das Spektrum der von Betriebssystemen zu erbringenden Leistungen beispielsweise betreffend Funktionalität, Performance oder adquate Programmierschnittstellen variiert dabei u.U. erheblich. Gerade bei eingebetteten Systemen besteht aufgrund verschräfter Ressourcenknappheit ein besonderes Interesse, die eingesetzten Laufzeitplattformen optimal auf die Anforderungen der Anwendung abzustimmen.

   Eine derartige Maschneiderung eines Betriebssystems ist im weitesten Sinne eine Konfigurationsaufgabe. Unser vornehmliches Augenmerk liegt dabei auf einer statischen Konfiguration, d.h. der Zusammenstellung und Anpassung einer geeigneten Laufzeitumgebung zum Entwicklungszeitpunkt des Systems. Die Maschneiderung soll sowohl bezüglich funktionaler Aspekte (erbrachte Funktionalität, Schnittstellen, etc.) als auch nicht-funktionaler

Aspekte (Zeit- und Speicherkomplexität, Fehlertoleranz, etc.) durch Anwendungsanforderungen gesteuert erfolgen. Hierbei ist eine weitgehende Werkzeugunterstützung des Vorgangs bis hin zu einer Teilautomatisierung wünschenswert. Der von uns verfolgte Ansatz beruht auf der Komposition maßgeschneiderter Laufzeitplattformen aus geeignet zu konfigurierenden generischen Komponenten. Generische Komponenten sind hierbei Softwarebausteine, deren funktionale Schnittstellen und nichtfunktionale Eigenschaften in begrenztem Umfang angepaßt werden können, ohne notwendigerweise manuelle Veränderungen am Komponentenkode vornehmen zu müssen. Wesentliche Bestandteile generischer Komponenten sind sogenannte generische Parameter, die der Wahl der Komponenteneigenschaften dienen. Wir unterscheiden hierbei grundsätzlich drei Typen solcher Parameter: Selektionsparameter, Generierungsparameter und Kodeparameter.

Selektionsparameter wählen diskrete Elemente wie Datenstrukturen, Algorithmen, kleinere Kodeteile oder ganze Komponenten aus einer endlichen Menge vordefinierter Varianten. So läßt sich beispielsweise über einen Selektionsparameter einer Speicherverwaltungskomponente eine geeignete Variante wählen, die sowohl den zu Verfügung stehenden Ressourcen wie Speicherplatz und Rechenzeit, als auch den diesbezüglichen Charakteristika der geplanten Anwendung möglichst gut Rechnung trägt. Diesem Parametertyp liegt ein Ansatz zugrunde, der generische Komponenten als eine Menge vorimplementierter Komponentenvarianten auffaßt. Selektionsparameter wählen dabei genau eine solche Variante aus, ohne daran Veränderungen vorzunehmen. In einer verfeinerten Sichtweise kann eine solche Auswahl nicht nur vollständige Komponenten, sondern auch einzelne Komponententeile betreffen.

Generierungsparameter steuern die werkzeugunterstützte Erzeugung von Komponenten mit gewünschten Eigenschaften. Einfache Generatoren können auf diese Weise z.B. Zahlenkonstanten einfügen, Quelltextoptimierungen vornehmen oder Funktionsnamen anpassen. Leistungsfähigere Generatoren sind in der Lage, Komponentenkode aus abstrakteren Beschreibungen zu erzeugen. Beispielsweise kann für eine Scheduler-Komponente die Implementierung einer anwendungsspezifischen Schedulingstrategie aus einer vom Benutzer vorgegebenen SDL-Spezifikation generiert werden. Im Gegensatz zur Verwendung vorimplementierter Varianten werden hierbei generische Parameter als Platzhalter in einer abstrakten formalen Beschreibung, der sogenannten Gerstinformation, aufgefaßt. Diese Gerstinformation kann selbst z.B. aus Quelltext oder einer SDL-Spezifikation bestehen. Die Anpassungsmöglichkeiten beschränken sich in diesem Fall nicht mehr auf den Austausch existierender Komponentenvarianten bzw. ihrer Bestandteile, sondern werden durch die Fähigkeiten der Generatoren bestimmt.

Wo hochgradig individuelle Anpassungen erforderlich sind, die nicht automatisiert durch Generatoren vorgenommen werden knnen, erlauben es Kodeparameter, individuelle Veränderungen am Komponentenkode vorzunehmen. Der Benutzer einer Komponente kann mit diesen Parametern z.B. eine selbstdefinierte Prozeßkontrollblock-Datenstruktur zur Feinanpassung einer Scheduling-Komponente oder einen nicht vorimplementierten Algorithmus einfügen. Kodeparameter sind mit einem Realisierungsansatz für generische Komponenten korelliert, der die Bereitstellung vordefinierter Implementierungslükken vorsieht. Der Anwender hat an diesen Stellen die Möglichkeit, selbst in die Komponentenimplementierung einzugreifen, muß sich im Regelfall aber auch an bestimmte Randbedingungen wie vorgegebene Schnittstellen halten. Allen drei genannten Parametertypen ist die Tatsache gemein, da Komponenten mit den gewünschten Eigenschaften in einem werkzeugunterstützten Instanziierungsvorgang aus generischen Komponenten respektive ihren Gerstinformationen und den Belegungen ihrer generischen Parameter gewonnen werden. Die unterschiedlichen Parametertypen spiegeln sich lediglich im Ablauf des Instanziierungsschrittes wieder. Im Falle von Selektionsparametern besteht die Gerstinformation aus den vorimplementierten Varianten, von denen entsprechend der Parameterbelegung eine unverändert übernommen wird; das Instanziierungsergebnis ist identisch mit einem Teil der Gerstinformation. Bei der Verwendung von Generierungsparametern wird aus Gerstinformation und Parameterwerten neuer Kode erzeugt, und im Falle von Kodeparametern sind Parameterwerte unverändert zu übernehmende und in die Gerstinformationen einzufügende Kodeteile. Ein Instanziierungsschritt kann das Entstehen einer wiederum generischen Komponente mit neuen, anderen generischen Parametern zur Folge haben, was eine iterative Anwendung der Instanziierung nach sich zieht. Die genannten Parametertypen können dabei in beliebiger Kombination auftreten.

Auf Implementierungsebene können generische Parameter durch eine Reihe von Mechanismen unterstützt werden. Selektionsparameter lassen sich im einfachsten Fall durch die Verwendung separater Quelltextdateien modellieren. Präprozessoren erlauben darüber hinaus die einfache Auswahl von Quelltextteilen und unterstützen somit feingranularere Selektionsparameter. Spezialisierte Werkzeuge können mit Hilfe geeigneter Markierungen im Quelltext die Funktion von Standard-Präprozessoren übernehmen und spezifisch erweitern. Auf diese Weise werden neben Selektionsparametern auch Generierungsparameter unterstützt, sofern die Gerstinformation selbst aus Quelltext besteht. Neben diesen Verfahren zur Auswahl und Manipulation von Quelltext bietet die Mehrzahl der Programmiersprachen Konzepte zur generischen Programmierung an, beispielsweise mittels Templates, Vererbung oder Casting. Mit diesen programmiersprachlichen Mitteln werden die Voraussetzungen für Kodeparameter geschaffen, z.B. durch Vorgabe von

Interfaces oder Spezifikationen.

Die generische Auslegung von Komponenten erlaubt deren einfache Anpaßbarkeit an unterschiedliche Anforderungen und somit ihren Einsatz in verschiedenen Systemumgebungen. Einem erhöhten Anfangsaufwand für die initiale Bereitstellung derart flexibler Softwarekomponenten steht somit eine Steigerung des Wiederverwendungspotentials gegenüber, die eine Vereinfachung der Konstruktion anwendungsangepaßter Laufzeitplattformen für eingebettete Systeme bis hin zu einer Teilautomatisierung verspricht.

4. **Werkzeuggestützte Konfigurierung und Adaptierung**, R. Meyer, Universität Potsdam

Betriebssysteme operieren in einer extrem heterogenen Umwelt. Dies gilt sowohl für die zugrundeliegende Hardware als auch für die durch Applikationen gestellten Anforderungen. Zum einen wird eine effizienzorientierte Betriebssystemsicht immer danach trachten, die Lücke zwischen den Applikationsanforderungen und der existierenden Hardware durch jeweils dediziert entwickelte Abstraktionen schließen zu wollen. Dem steht jedoch der Wunsch nach Wiederverwendbarkeit und vertrauten und damit beherrschbaren Programmiermodellen entgegen. Einen hoffnungsvollen Ansatz zur Lösung dieses Dilemmas bietet das Konzept der Programmfamilien. Die Idee, jeweils in einzelnen Familienmitgliedern nur eine minimale Basis an Funktionalitäten zu implementieren, die jeweils nur um unumgänglich notwendiges minimal erweitert werden, ließ sich erfolgreich auch auf Betriebssysteme übertragen (PEACE, PURE).

Bisher kaum erforscht in diesem Umfeld ist ein konsistenter Ansatz zur Konfigurierung solcher Systeme, der sowohl eine Erstkonfigurierung (incl. Generierung) als auch die dynamische Rekonfigurierung, insbesondere Adaptierung zur Laufzeit umfaßt.

Im Vortrag soll ein im Rahmen eines Promotionsvorhabens entwickelter Ansatz vorgestellt werden, der die dynamische Rekonfigurierung (von außen gesteuert) sowie die automatische Adaptierung an eine sich ändernde Umgebung ermöglicht. Hierzu wird ein eigenständiges Konfigurierungsmodell angewandt, welches Applikations- bzw. Systemcode von konfigurierendem Code explizit unterscheidet. Konfigurierungscode wird in diesem Modell in Konfigurierungsobjekten gekapselt. Da sich Konfigurierungsmaßnahmen durchaus ähneln, wiederholen oder gar repliziert erscheinen, wird Wiederverwendbarkeit von Konfigurierungsobjekten durch erweiterbare Konfigurierungsklassen angestrebt, welche in Bibliotheken gesammelt werden sollen.

Dieser Ansatz wird in einen an den Universitäten Magdeburg und Potsdam in der Entwicklung befindlichen Betriebssystembaukasten integriert werden.

Dieses Projekt einer in C++ implementierten „Werkbank“ zur Generierung und Rekonfigurierung von familienorientierten Betriebssystemen wird auch einen grafischen Editor zur dynamischen Konfigurierung beinhalten, der es dem Benutzer erlaubt, Rekonfigurierungsmaßnahmen direkt auszulösen und zu kontrollieren. Hierbei wird das zugrundeliegende Konfigurierungsmodell direkte Anwendung finden.

5. **Werkbank zum Bau maßgeschneiderter Betriebssysteme**, D. Beuche, U. Haack, W. Schröder-Preikschat, O. Spinczyk, Otto-von-Guericke-Universität Magdeburg

   Unter „maßgeschneiderten Betriebssystemen“ verstehen wir Betriebssysteme, die sich in Struktur und Funktionalität den Bedürfnissen gegebener Anwendungen und Hardware-Plattformen optimal anpassen können. Entsprechend der klassischen Vermittlerrolle zwischen Anwendungen und Hardware bedeutet dies die Anpaßbarkeit „nach oben“ *und* „nach unten“. Dabei wird die Forderung nach dynamischer Anpaßbarkeit zur Laufzeit zunächst einmal als zweitrangig angesehen, obgleich ihre technische Umsetzung eine große Herausforderung im Betriebssystembau darstellt. Im Gegensatz dazu wird der statischen Anpaßbarkeit des Betriebssystems zur Übersetzungs- bzw. Bindezeit eine vorrangige Bedeutung beigemessen, da sie die Voraussetzung für die dynamische Variante darstellt. Die Güte der statischen Anpaßbarkeit ist eine Frage der Software-Struktur, insbesondere des Grades der Modularität und der Komplexität der Moduln. Maßgeschneiderte Betriebssysteme sind anwendungsorientierte und „schlanke“ Betriebssysteme, die genau aus nur den Moduln bestehen, die zur Erbringung der durch den Einsatzbereich geforderten Funktionalität auch notwendig sind.

   Der Trend geht hin zu anwendungsbezogenen Betriebssystemen, d.h. es werden Systeme angestrebt, die in Art und Umfang der angebotenen Funktionalität variabel sind, um so eine Vielzahl verschiedenartigster Anwendungen möglichst optimal unterstützen zu können. Dabei gibt es unterschiedliche Auffassungen über den Umfang einer gemeinsamen Basis der verschiedenen Betriebssystemausprägungen. Die Werkbank setzt dabei auf einen extrem kleinen Umfang („Skalierung nach unten“). Auf diese Weise wird effiziente System-Software insbesondere auch für kleinste eingebettete Systeme bereitgestellt. Es soll ein *Baukasten* und dazugehörige Werkzeuge entwickelt werden für die Konstruktion *maßgeschneiderter*, problemorientierter *Betriebssysteme*. Dadurch soll ein Anwender bzw. ein Systemkonstrukteur in die Lage versetzt werden, sich aus einer Menge extrem feingranular strukturierter Betriebssystemfunktionalitäten das für seine Zwecke optimal passende System zu generieren. Darüber hinaus sollen dynamisch Veränderungen im System vorgenommen werden können, d.h., es sollen bei Bedarf zur Laufzeit Funktionalitäten hinzugefügt bzw. entfernt werden können.

6. **Experiences with a PC SMP Cluster**, A. A. Fröhlich, GMD FIRST

   This talk presents preliminary results on our experiences to support high performance computing using a PC SMP Cluster. Our cluster comprises eight dual-Pentium II PCs interconnected by Myrinet and Fast Ethernet networks. For the first experiments we selected Solaris as the operating system and some user level communication packages, including Illinois Fast Messages and Berkeley's Active Messages, to support applications and higher level packages, like MPI. With this this environment we collected statistics that pointed out weakness on both, hardware and software. However, we believe we can achieve reasonable performance with this inexpensive software, as long as the proper software, mostly still to be developed, is adopted.

7. **Die Erforschung der Langsamkeit: Vernachlässigte System-Resourcen und ihre Beherrschung**, F. Bellosa, Universität Erlangen-Nürnberg

   Die Umgebung, innerhalb der ein Computer sinnvolle Arbeit verrichten kann, ist durch die Hardwareressourcen und die Software bestimmt, welche diese Ressourcen verwaltet. In den letzten 40 Jahren konzentrierte sich die Betriebssystemforschung auf die Verwaltung der Basis-Ressourcen Prozessor, Hauptspeicher und Ein/Ausgabe. Abstraktionen wie Aktivitätsträger, Adreßräume, Dateisysteme und virtuelle Kommunikationskanäle wurden geschaffen, um die Handhabung dieser Basis-Ressourcen zu vereinfachen und allgemeine Schnittstellen anzubieten.
   Doch neue Anforderungen an Rechensysteme zwingen zur Einführung von neuen Hardwarekomponenten und zur Entwicklung von Verwaltungssoftware für Ressourcen, die bislang als nicht als „manage-würdig“ eingestuft wurden. Exemplarisch soll an den beiden Ressourcen **Energie** und **Speicherbandbreite** aufgezeigt werden, daß Betriebssysteme noch lange nicht erforscht sind, sondern sich die Forschung auf diesem Gebiet immer wieder neuen Herausforderungen stellen muß.

   Stromversorgung und Kühlung ist eine Voraussetzung für den Betrieb von Rechnern, die in der Vergangenheit als stets und konstant verfügbare Umgebungsbedingung unabhängig von der Art des Betriebs angenommen wurde. Diese Annahme ist nicht mehr zutreffend für tragbare Rechner mit beschränkter Stromversorgungs- und Kühlkapazität.
   Die **Energie**, die integrierte Schaltungen für ihren Betrieb brauchen, ist proportional zur Anzahl der Gatter und der Taktfrequenz, mit der sie betrieben werden. Ein Hochleistungsprozessor benötigt zur Zeit zwischen 26W (UltraSPARC-II mit 250Mhz) und 60W (alpha 21264 mit 300 MHz). Angesichts des Trends zu steigenden Taktfrequenzen und steigender Chipkomplexität, tauchen zwei Fragen auf: Können zukünftige Systeme noch

mit genügend Energie versorgt werden (z.B. in Notebooks und PDAs) und kann die Energie durch passive Kühlung noch abgeführt werden? Ist es mögliche, die Energieaufnahme durch ausgeklügelte Prozeß- und Energie-Verwaltungstechniken zu drosseln?
Einige der heutigen Prozessorarchitekturen bieten die Möglichkeit der reduzierten Taktfrequenz, um Energie zu sparen. Die Reduzierung der Taktfrequenz bewirkt eine lineare Reduzierung der Energieaufnahme, aber auch eine entsprechende Reduktion der Rechenleistung. So bleibt ein einfaches Maß für die Rechenleistung pro Energieeinheit definiert in Millionen Instruktionen pro Joule (MIPJ) konstant. Bei einer Reduzierung der Taktfrequenz läßt sich aus technischen Gründen auch die Betriebsspannung reduzieren. Damit bietet eine reduzierte Taktfrequenz die Möglichkeit zu einer quadratischen Energieeinsparungen, da die Energie quadratisch zur Betriebsspannung ist. Die Prozessverwaltung kann daher die Taktfrequenz für jeden Rechenauftrag bestimmen und so den Stromverbrauch kontrollieren. Der Stromverbrauch bestimmt einerseits die Betriebszeit eines Rechensystems als auch die Leistungsreserven des Systems für die nahe Zukunft, da der Rechner nur während kurzer Hochlastphasen mit voller Leistung gefahren werden kann. Die Kunst des Betriebssystems besteht also darin, mit minimaler Taktfrequenz eine bestimmte Dienstgüte aufrecht zu erhalten und zeitabhängige Dienste vor Ihrer Deadline auszuführen. Bislang wurde dieser Aspekt der „slow and cool execution“ nicht in Betriebssystemen berücksichtigt.

Die Fortschritte in der Speichertechnologie in Bezug auf Geschwindigkeit konnten nicht mit denen in der Prozessortechnologie mithalten. Prozessoren, die mit mehreren hundert Megahertz getaktet sind, können wesentlich mehr Anforderungen an das Speichersubsystem stellen, als dieses erfüllen kann. Die **Speicherbandbreite** ist die Datenmenge, die vom/zum Speicher in einem Zeitintervall transferiert werden kann. Die Anzahl der erfüllbaren Speicheranforderungen pro Zeitintervall und damit die Speicherbandbreite ist begrenzt.
Ein Prozessor, der mehr Speicheranforderungen stellt, als erfüllt werden können, muß Wartezyklen einlegen, welche die Effizienz reduzieren. In einem Multiprozessorsystem wird die verfügbare Speicherbandbreite von allen Prozessoren und DMA Geräten geteilt. Folglich können sich die Prozessoren gegenseitig behindern und in ihrer Effizienz beeinflussen. Wir nennen diesen Effekt *Memory Preemption.*
Die Anzahl der Speicherzugriffe, die ein Prozessor pro Zeitintervall tätigen kann, wird durch die Taktfrequenz und die Speicher-Referenz-Muster bestimmt. Wenn wir die Kenngrößen des Hauptspeichers kennen und die individuellen Zugriffsmuster einzelner Tasks messen können, kann diese Information im Scheduler genutzt werden, um den Effekt der Memory Pre-

emption vorherzusagen und zu kontrollieren. Die Kontrolle kann durch das Einfügen von Idle-Zyklen im Prozessor erfolgen, um den Speicherzugriff von Tasks mit geringer Priorität zu drosseln. Den gleichen Effekt bei gleichzeitiger Energieeinsparung hätte eine Reduzierung der Taktfrequenz.

Die zwei vorgestellten Beispiele zeigen eindrucksvoll, daß es Ressourcen wie Energie, Kühlung und Hauptspeicherbandbreite gibt, die bislang nicht von Betriebssystemen behandelt wurden. Im Vortrag werden Lösungsansätze vorgestellt und erste Ergebnisse und Messungen präsentiert.

8. **Reflexion als Strukturierungsprinzip objektorientierter verteilter Systeme: Das metaXa Projekt**, M. Golm, J. Kleinöder, S. Reitzner, Th. Riechmann, Universität Erlangen-Nürnberg

   Objektorientiertes Design und objektorientierte Programmierung haben sich bei der Entwicklung komplexer Systeme als vorteilhaft erwiesen. Die Unterteilung der Applikation in Objekte bzw. Klassen erleichtert das Verständnis und die Erweiterbarkeit der Applikation. Jede objektorientierte Sprache beruht auf einem bestimmten Objektmodell. Das Objektmodell beschreibt die syntaktischen und semantischen Eigenschaften der Sprache. Das Objektmodell legt zum Beispiel fest, ob Objekte persistent und aktiv sein können, ob Methodenaufrufe an entfernten Objekten oder Objektmigration unterstützt werden. Ein fest vorgegebenes Objektmodell kann die klare Strukturierung einer Anwendung aber auch behindern, wenn bestimmte, für die Anwendung benötigte Objekt-Eigenschaften im Objektmodell fehlen und dehalb bei der Anwendungsprogrammierung Objektmodell-Mechanismen nachgebildet werden müssen. Es existiert kein Objektmodell, welches für alle Anwendungen ausreichend wäre. Traditionell werden Systeme durch ein Schichtenmodell strukturiert. Untere Schichten (z.B. ein Betriebssystem, darüber evtl. Middleware) implementieren Dienste, die höhere Schichten (z.B. eine Anwendung) über Schnittstellen (Application Programmer Interface, API) benutzen. Die Trennung durch fest definierte Schnittstellen sichert die Unabhängigkeit der Implementierungen der verschiedenen Ebenen, hat aber den Nachteil, daß höhere Schichten Dienste tieferer Schichten lediglich benutzen, aber kaum Einfluß auf deren Verhalten oder Implementierung nehmen können. Die Folge sind Performanceverluste des Gesamtsystems sowie die Neuprogrammierung nicht passender Dienste in anderen Schichten. Performanceverluste werden besonders deutlich, wenn über einer Betriebsystemschicht beispielsweise eine virtuelle Maschine (wie die Java VM) läuft, die die eigentlichen Anwendungen ausführt. In aktuellen Forschungsprojekten wurde versucht, den Performanceverlust durch Integration der virtuellen Maschine in das Betriebsystem zu beseitigen - d.h. es werden zwei Schichten verschmolzen. Das metaXa-Projekt versucht, eine Lösung für dieses Problem durch Erweiterung und Öffnung der Java

Virtual Machine (JVM) für die Anwendungsebene zu finden.

Auch die JVM implementiert ein bestimmtes Objektmodell. So sind Objekte z.B. transient und passiv, können als Monitor verwendet werden, etc. Die metaXa-VM erlaubt es, diese Eigenschaften an Anwendungsanforderungen anzupassen. Für diesen Zweck wird Reflexion verwendet. Reflexion ermöglichst es Programmen, ihre Struktur und ihr Verhalten zu beobachten und zu verändern. Die Implementierung des Objektmodells, die üblicherweise durch ein Laufzeitsystem erfolgt, ist für Anwendungsprogramme zugreifbar und änderbar. MetaXa unterstützt zum einen Reflexion, indem es Metaobjekten den Zugriff auf Namen und Typen von Instanzvariablen, Methodennamen und Signaturen sowie Typinformationen von Objektreferenzen ermöglicht (*structural reflection*). In diesem Sinne ähnelt metaXa recht stark dem Reflection-API von Sun JDK 1.1. Zusätzlich bietet metaXa aber auch die Möglichkeit, in den Berechnungsvorgang einzugreifen (*behavioral reflection*). Dies geschieht dadurch, daß interessante Aktionen der Anwendung ein Ereignis (*Event*) auslösen können. An ein Java-Objekt (*Basisobjekt*) kann zur Laufzeit des Programms ein *Metaobjekt* gebunden werden. Reflexion über das Basisobjekt-Verhalten wird möglich, indem bei folgenden Mechanismen ein Event-Handler des Metaobjektes aufgerufen wird: Methodenaufrufe, Variablenzugriffe, Betreten und Verlassen von Objektmonitoren, Erzeugen von Objekten, Laden von Klassen. Diese Mechanismen können durch das zuständige Metaobjekt abweichend vom Java-Objektmodell implementiert werden. Metaobjekte modellieren Basisebenen-Objekte, d.h. sie enthalten Informationen bezgl. Struktur und Verhalten der Objekte.

Die Verwendung von Metaobjekten bedeutet im Schichtenmodell, daß die Anwendungsschicht eigene Implementierungen für Teile der Schicht der virtuellen Maschine installiert. Durch die Trennung von Basisobjekten und Metaobjekten bleibt die Gliederung in verschiedene Schichten aber erhalten.

Um die Vorteile unserer reflexiven Architektur zu demonstrieren, haben wir Metasysteme für Fernaufruf, Replikation, Sicherheit, Koordinierung, Persistenz, aktive Objekte und Just-in-time Compiler implementiert.

Replikation Replikation wird im Java Objektmodell nicht unterstützt. Wir haben Metaobjekte implementiert, die einen Methodenaufruf in einen Multicast umwandeln (aktive Replikation), oder den Zugriff auf Instanzvariablen an die Replikate verteilen (passive Replikation). Zur Koordinierung nebenläufiger Zugriffe ersetzt das Metaobjekt den Lock-Mechanismus des replizierten Objektes durch eine eigene Implementierung. Weitere, spezielle Replikationsprotokolle können jederzeit hinzugefügt werden und innerhalb einer Anwendung können für verschiedene Objekte unterschiedliche Replikationsstrategien eingesetzt werden.

Sicherheit Java verwendet Capabilities als primären Sicherheitsmechanismus: Eine Objektreferenz in Java ist eine Capability für die Methoden des Zielobjektes. Leider kann man diese Capabilities in keiner Weise einschränken. Restriktion auf bestimmte Methoden, Revokation, Konfiguration begrenzter Gültigkeit und Verbreitungskontrolle fehlen. Diese zusätzlichen Konfigurationsmöglichkeiten lassen sich durch sogenannte Sicherheitsmetaobjekte erreichen: Objektreferenzen können mit Sicherheitsmetaobjekten versehen werden.

Alle sicherheitsrelevanten Operationen werden dann zunächst von dem Metaobjekt überprüft. Ein solches Metaobjekt kann Methodenaufrufe zulassen oder verweigern und dadurch Restriktion, Revokation und begrenzte Gültigkeit der Capabilities realisieren. Es kann übergebene Parameter und Rückgabewerte ebenfalls mit Metaobjekten versehen und dadurch eine Verbreitungskontrolle von Objektreferenzen und transitive Sicherheitsstrategien realisieren. Die Programmierung der Metaobjekte kann unabhängig von der Applikationsprogrammierung stattfinden: Man kann zunächst die Applikation ohne spezielle Sicherheitsbetrachtungen entwickeln oder Bibliotheken verwenden, die ohne Sicherheitsanforderungen entwickelt wurden. Diese können dann nachträglich durch passende Sicherheitsmetaobjekte auf die speziellen Sicherheitsanforderungen, unter denen die Applikation oder die Bibliothek eingesetzt werden soll, angepaßt werden. Die Konfigurierung ist extrem feingranular möglich: bei jeder Objektreferenz kann separat eine Sicherheitsstrategie festgelegt werden.

Koordinierung Die Integration von Koordinierung in eine objektorientierte Sprache geschieht meist auf der Ebene der Objekte oder sogar der Methoden. Diverse Arbeiten haben gezeigt, daß das objektorientierte Paradigma der Vererbung mit Koordinierungsmechanismen kollidieren kann. Das Resultat ist, daß man bei der Vererbung von einer koordinierten Klasse einen Teil der Basisklasse reimplementieren muß, nur weil sich die Koordinierung in der Subklasse geringfügig ändert. Diese Erscheinung wird als *Vererbungs Anomalie* bezeichnet.

Unsere Idee ist es, das nebenläufige Objekte in ein Objekt, das den algorithmischen Teil enthält (Basisobjekt) und ein Koordinierungsobjekt (Metaobjekt) aufzuspalten. Das Basisobjekt spezifiziert die Stellen, an denen Koordinierungs-Events ausgelößt werden sollen. Treten solche Events auf, so ist im algorithmischen Objekt spezifiziert, welche Aktionen im Koordinierungsobjekt angestoßen werden müssen. Das Koordinierungsobjekt blockiert die Ausführung dieser Aktionen solange, bis das Ereignis koordiniert ist und gibt dann die Kontrolle an das algorithmische Objekt zurück. Dieses kann die Arbeit fortsetzen und braucht sich um die Koordinierung nicht weiter zu kümmern.

Die Abbildung von Ereignissen auf Koordinierungsaktionen nennt man das *Event Mapping*. Dieses Mapping ist virtuell, was bedeutet, daß eine Unterklasse das Mapping von Basisklassen-Ereignissen verändern kann. Das gewährleistet, daß eine Änderung der Koordinierung in der Subklasse nicht die Reimplementierung der gesamten Basisklassenmethode erzwingt, sondern nur der Koordinierungsanteil verändert werden muß. Dieses Vorgehen bewirkt eine erhebliche Reduzierung der Anomalien.

9. **Objekte in Windows-NT**, W. Kalfa, Technische Universität Chemnitz-Zwickau

   Beitrag wird nachgeliefert.

10. **A Topology-Based Approach to Coordinated Multicast Operations**, J. Nolte, GMD FIRST

    Multicasts are a powerful means to implement coordinated operations on distributed data-sets as well as synchronized reductions of multiple computed results. In this paper we present a topology based approach to implement parallel operations on distributed data-sets as multicasts. Multicast groups are described as reusable application-specific topology classes that coordinate both the spreading of multicast messages and the collection (reduction) of the computed results. Thus global operations are controllable through applications and existing communication topologies as well as synchronization patterns can effectively be reused.