# Ein selbstoptimierendes Echtzeitbetriebssystem für verteilte selbstoptimierende Systeme *

Simon Oberthür
Heinz Nixdorf Institut
Universität Paderborn
Fürstenallee 11
33102 Paderborn
simon@oberthuer.net

Carsten Böke
Heinz Nixdorf Institut
Universität Paderborn
Fürstenallee 11
33102 Paderborn

Franz Rammig
Heinz Nixdorf Institut
Universität Paderborn
Fürstenallee 11
33102 Paderborn

## ABSTRACT

Selbstoptimierende mechatronische Systeme passen sich an verändernde Bedingungen zur Laufzeit an. Statische Systemsoftware für eine solche Anwendungsklasse muss für den allgemeinen Fall konfiguriert sein und ist dadurch nicht optimal. Deshalb wurde ein als Multiagentensystem realisiertes selbstoptimierendes RTOS entwickelt, welches sich zur Laufzeit optimal an die sich dynamisch ändernden Anforderungen von selbstoptimierenden Anwendungen anpasst. Hierfür wurden Strategien auf RTOS-Ebene entwickelt, um die Dienstestruktur dynamisch an die Anforderungen der Anwendungen anzupassen. Das System muss dabei starken Echtzeit- und Sicherheitsbedingungen genügen.

## 1. EINLEITUNG

Steigende Komplexität ist eines der Hauptprobleme von modernen mechatronischen Echtzeitsystemen, wie beispielsweise beim Automobil. Um diese Komplexität zu beherrschen, ist ein Ansatz diese Systeme mit „Self-X-Eigenschaften“ auszustatten, wie Selbstoptimierung [7], Selbstreflexion bzw. der Fähigkeit der Selbstanpassung. Solche dynamischen Anwendungen stellen besondere Anforderungen und Herausforderungen an unterliegende Hardware/Software-Schichten verteilter Echtzeitsysteme [13].

Für diese Anwendungen wird in dem Teilprojekt C2 des SFB614 „Selbstoptimierende Systeme des Maschinenbaus“, ein Echtzeit-Betriebssystem (RTOS) für verteilte selbstoptimierende Systeme bereitgestellt. Hierfür wurden grundlegende Konzepte für ein solches selbstoptimierendes RTOS entwickelt. RTOS-Dienste werden auf einem Berechnungsknoten (CPUs bzw. FPGAs) lokal optimiert und Ressourcen dynamisch unter den Anwendungen selbstoptimierend verteilt.

*Diese Arbeit ist im Sonderforschungsbereich 614 – Selbstoptimierende Systeme des Maschinenbaus, Universität Paderborn – enstanden und wurde auf seine Veranlassung unter Verwendung der ihm von der Deutschen Forschungsgemeinschaft zur Verfügung gestellten Mittel veröffentlicht.

Als Basis für das selbstoptimierende RTOS wurde die konfigurierbare Betriebssystembibliothek DREAMS (Distributed Real-Time Extensible Application Management System) [5] verwendet. DREAMS war in der ursprünglichen Version lediglich offline konfigurierbar. Durch eine Konfigurationsbeschreibung wurden die konfigurierbaren Komponenten der Bibliothek ausgewählt. Nach dem Kompilieren des Betriebssystems waren somit nur die Komponenten und Eigenschaften in der konfigurierten Betriebssysteminstanz enthalten, die von der Anwendung benötigt werden. Techniken zur automatisierten Erzeugung einer solchen Konfigurationsbeschreibung wurden im Rahmen des DFG SPP1040 TEReCS (Tools for Embedded Real-Time Communication Systems) [2] erarbeitet.

TEReCS unterscheidet strikt zwischen Wissen über die Anwendung und Wissen von Experten über das zu konfigurierende Betriebssystem. Informationen über die Anwendungen wurden über eine Anforderungsbeschreibung spezifiziert. Diese Spezifikation beschreibt abstrakt das Verhalten und Bedingungen (Zeitschranken) der Anwendung. Das Verhalten der Anwendung wird durch die verwendeten Systemaufrufe des Betriebssystems beschrieben. Es beschreibt, welche Systemprimitiven zu welchem Zeitpunkt benötigt werden. Zusätzlich werden in der Anforderungsbeschreibung die benötigte Hardware-Plattform und -Topologie spezifiziert.

In einer Wissensbasis wird der zulässige Entwurfsraum des konfigurierbaren Betriebssystems über die Abhängigkeiten der Dienste in einem UND/ODER-Graph vollständig beschrieben. In dieser Domänen-spezifischen Beschreibung des Betriebssystems sind Optionen, Kosten und Randbedingungen spezifiziert und führen so zu einer Überspezifikation, die alternative Implementationsvarianten enthält. Aus der Domänen-spezifischen Beschreibung wurde durch den Konfigurationsprozess mit Hilfe der Anforderungsbeschreibung der Anwendung der Entwurfsraum auf das benötigte Minimum verkleinert. Der Entwurfsraum enthält die gültigen Konfigurationen für die zu generierende Betriebssystem-Laufzeitplattform.

## 2. ECHTZEITBETRIEBSSYSTEM FÜR SELBSTOPTIMIERENDE SYSTEME

Das hier um Selbstoptimierung erweiterte RTOS DREAMS passt sich gegebenen Umständen (variierendes Anwendungs-

feld und/oder variierende Anforderungen der Anwendungen) dynamisch an. Hiermit ergibt sich ein System, welches effizient die Verwaltung von selbstoptimierenden Anwendungen unterstützt und entsprechende Dienstleistungen zur Verfügung stellt.

Die Forderung nach Effizienz beinhaltet, dass nur die tatsächlich benötigten Dienste in das Betriebssystem integriert sind. In einem selbstoptimierenden System mit veränderlichen Parametern, Algorithmen oder gar Strukturen ändern sich diese Anforderungen an zu erbringende Dienste aber dynamisch, was bei Beibehaltung der Forderung nach Effizienz eine Dynamik der unterliegenden Dienstestruktur erfordert. Agentensysteme auf hohem Abstraktionsniveau und unter Echtzeit wirkende Reglersysteme haben stark unterschiedliche Anforderungen an ein Echtzeit-Betriebssystem (RTOS). Insbesondere aber die Vorgabe, dass sich Anwendungen selbstoptimierend an die Umgebung anpassen, führt zu stark variierenden Anforderungsprofilen.

Eine dynamische Adaption des RTOS an die Anwendung wird durch eine effiziente Instanziierung des Gesamtsystems, welches mit möglichst geringem Aufwand arbeitet, erreicht. Unterschiedlichste Anforderungen von Anwendungen zu disjunkten Zeitpunkten werden mit minimalen Ressourcen erfüllt. Durch das entwickelte selbstoptimierende RTOS werden nur die Anforderungen erfüllt und somit Dienste geladen, die aktuell benötigt werden.

Um ein dynamisch rekonfigurierbares RTOS aufzubauen, galt es folgende weiterführenden Fragestellungen zu beantworten: Zunächst musste der Entwurfsraum der Rekonfiguration des RTOS modelliert werden, um die Möglichkeiten der Anpassung des Systems an die Anwendungen zu modellieren. Die Rekonfiguration soll stattfinden, wenn sich die Anforderungen der Anwendungen verändern. Deshalb musste eine Schnittstelle zwischen Anwendung und Betriebssystem definiert werden, die den Austausch dieser Informationen ermöglicht. Mit diesen Informationen muss eine zu diesen Anforderungen adäquate Konfiguration des Systems ausgewählt werden, wozu ein Ressourcen-Management-System zu entwickeln war. Das von uns entwickelte System besteht daher aus den folgenden Modulen: *Profile Framework*, *Online-TEReCS* und *Flexible Resource Management*.

Insgesamt stellt das RTOS eine Menge konfigurierbarer RTOS-Komponenten zur Verfügung, die durch den Online-Konfigurator *Online-TEReCS* konfiguriert werden. Dieser wird durch Profile gesteuert, die durch ein *Profil-Framework* definiert werden können. Über das Profil-Framework werden auch die durch Deaktivierung von RTOS-Diensten frei gewordenen Ressourcen den Anwendungen zur Verfügung gestellt. Die strategischen Rekonfigurationsentscheidungen werden von dem o.a. flexiblen Ressourcen-Management-System getroffen.

## 2.1 Profile Framework

Als Eingabe für die Online-Konfiguration des RTOS wird der Bedarf an Ressourcen jeder ausgeführten Anwendung benötigt. Die Systemprimitiven des Betriebssystems werden hierbei ebenfalls als Ressourcen aufgefasst.

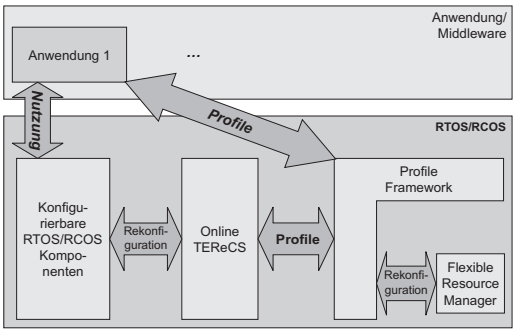


**Figure 1: Überblick selbstoptimierendes RTOS** DREAMS

Um die benötigten Dienste zur Bereitstellung der Ressourcen rechtzeitig zu instanziieren, ist es zusätzlich notwendig, dass Anwendungen den Bedarf möglichst frühzeitig mitteilen können. Je mehr Informationen das RTOS-System über die Anwendung besitzt, desto optimaler kann es sich auf die Anwendung einstellen. Deshalb ist es wichtig, eine Schnittstelle zur Anwendung zu schaffen, über die diese möglichst einfach zusätzliche Informationen bereitstellen kann.

Als Schnittstelle aus diesen Anforderungen ist ein Profile Framework [11] entwickelt worden. Mit diesem Framework kann eine Menge von Profilen pro Anwendung definiert werden. Je Profil wird eine Implementationsvariante der Anwendung mit einem unterschiedlichen Bedarf an Ressourcen beschrieben. Pro Ressource ist die maximale Belegung der Ressource für jedes Profil angegeben, die die Anwendung benötigt, wenn das Profil aktiviert ist. Jede Allokation von einer Ressource bedarf einer vorherigen Mitteilung an die Ressourcenverwaltung des Systems. Pro Ressource ist die maximale Verzögerung bei einer Allokation durch die Ressourcenverwaltung zu definieren. Zusätzlich muss definiert werden, zwischen welchen Profilen umgeschaltet werden darf bzw. kann. Hierbei wird unterschieden zwischen Profilwechseln, die von der Anwendung bzw. vom Ressourcenmanager initiiert werden. Das bedeutet, dass Anwendungen *On Demand* ihr Profil wechseln können. Um unter harten Echtzeitbedingungen zwischen den Profilen wechseln zu können, muss die Dauer der Rekonfiguration bekannt sein. Deshalb muss die maximale Dauer (WCET, Worst-Case-Execution-Time) bei vom Ressourcenmanager initiierbaren Wechseln zwischen Profilen angegeben werden. Jedes Profil erhält zusätzlich eine Gewichtung, die die Qualität der Implementationsvariante des Profils widerspiegelt.

Da selbstoptimierende Anwendungen sich mit der Zeit verändern können, können Profile dynamisch zur Laufzeit angelegt, entfernt und verändert werden. Der Entwurfs-Prozess von mechatronischen Systemen basiert heutzutage auf dedizierten Entwicklungswerkzeugen wie MATLAB/SIMULINK oder CAMeL-View. Durch Codegeneratoren wird automatisch Programmcode erzeugt. Die Modellierung von Anwendungen auf Basis des Profile Frameworks stellt, neben den Vorteilen, einen zusätzlichen Aufwand für den Entwickler dar. Um diesen Aufwand möglichst gering zu halten, wurde die Erzeugung von Profilen für selbstoptimierende Anwendungen in Entwurfstechniken und Entwurfsmethoden integriert [4]. So wurde ein Verfahren entwickelt, um aus er-

weiterten hybriden Real-Time Statecharts halb-automatisch Profile für eine Anwendungen zu generieren.

## 2.2 Online-TEReCS

Auf Basis des Offline-Konfigurators TEReCS wurde eine onlinefähige Version des Konfigurators entworfen [12]. Mit Hilfe dieses Konfigurators können sowohl die Möglichkeiten der Rekonfiguration wie auch die Abhängigkeiten der feingranularen und konfigurierbaren Dienstestruktur des Systems beschrieben werden. Außerdem wird die eigentliche Rekonfiguration über den Konfigurator vorgenommen.

TEReCS wurde ursprünglich für die Offline-Konfiguration von RTOS-Systemen auf Quelltext-Ebene mit feingranularen optionalen Komponenten entworfen. Da eine feingranulare Rekonfiguration zur Laufzeit zu komplex ist, werden nun grobgranulare Strukturen rekonfiguriert. Hierdurch wird der Konfigurationsraum eingeschränkt und somit die Laufzeit für die Online-Auswahl einer Konfiguration minimiert. Die Möglichkeit, in TEReCS auf dem Entwurfsraum eine Hierarchie zu definieren, ermöglichte die Wiederverwendung des Konfigurators. So können auf Sourcecode-Ebene grobgranulare vorkonfigurierte Cluster bzw. RTOS-Agenten (z. B. Systemhierarchie, Speicherverwaltung, Scheduling, Kommunikation, Gerätetreiber, etc.) bereitgestellt werden, die zur Laufzeit ausgewählt werden. Ein RTOS-Agent umfasst dabei eng zusammenhängende Komponenten. Die Ableitungshierarchie sowie Komponenten der Klassenobjekte eines RTOS-Agenten sind hierbei zur Laufzeit statisch, nur die Abhängigkeiten der Aufrufe von Funktionen von Objekten anderer RTOS-Agenten können sich ändern. RTOS-Agenten können in verschiedenen Ausprägungen offline vorkonfiguriert werden. Der RTOS-Agent Systemhierarchie kann beispielsweise einmal mit und einmal ohne Unterstützung von Multi-Threading vorkonfiguriert sein. Im ersten Fall benötigt dieser Agent Funktionalitäten des Agenten Scheduling, im zweiten Fall werden diese Funktionalitäten nicht benötigt.

Jede Abhängigkeit von RTOS-Agenten untereinander wird auf interne Primitiven abgebildet. Diese Primitiven können von anderen RTOS-Agenten verwendet werden. So modellieren auf der höheren Hierarchie-Ebene die alternativen RTOS-Agenten wie im ursprünglichen TEReCS-Ansatz einen UND/ODER-Graphen. Zusätzlich enthält der Graph auch Implementierungsalternativen von Agenten, die unterschiedlichen Ressourcenbedarf haben, z. B. Implementationen in Software und alternativ in rekonfigurierbarer Hardware.

Zur Laufzeit identifiziert der Konfigurator anhand der z. Z. benötigten Systemprimitiven, welche RTOS-Agenten benötigt werden. Konkret bedeutet dieses, dass die Dienste identifiziert werden, die für die Erfüllung der aktuell benötigten Systemprimitiven gebraucht werden. Hierbei wendet der Online-Konfigurator auf den UND/ODER-Graph der Ebene der RTOS-Agenten den selben Algorithmus an, wie der Offline-Konfigurator auf den UND/ODER-Graph der Menge der feingranularen Komponenten.

Informationen über die aktuell benötigten Systemprimitiven erhält das Online-TEReCS Modul indirekt über die aktivierten Profile der Anwendungen. Zu jedem RTOS-Agenten werden zu den Implementationsvarianten entsprechende Profile angelegt. Somit wird für die RTOS-Agenten ebenfalls – wie für die Anwendungen – pro Variante ein Profil angelegt. Zusätzlich wird pro RTOS-Agent ein Profil (Sperr-Profil) angelegt, in denen die Ressourcen von TEReCS, die von dem aktiven RTOS-Agenten zur Verfügung gestellt würden, als vollständig belegt markiert sind. Dieses Sperr-Profil wird aktiviert, wenn der RTOS-Agent nicht ins laufende System konfiguriert ist und die Ressourcen, die von diesem RTOS-Agenten zur Verfügung gestellt werden, somit nicht für Anwendungen zur Verfügung stehen. Die Aktivierung dieser Profile – und damit die Aktivierung der Rekonfiguration des RTOS – erfolgt durch den Flexible Resource Manager. Hierdurch wird gewährleistet, dass nie ein Profil einer Anwendung aktiviert wird, ohne dass alle benötigten Dienste – und somit durch den Dienst angebotene Ressourcen – zur Verfügung stehen.

## 2.3 Flexible Resource Manager

Das Hauptziel des Flexible Resource Managers (FRM) [11] ist es allgemein, die Auslastung der Ressourcen zu optimieren und die Qualität des Systems zu steigern. Als Basis verwendet der FRM die Informationen aus den Profilen der Anwendungen und RTOS-Agenten. Durch Deaktivierung und Aktivierung von Profilen einer Anwendung bzw. von RTOS-Agenten nimmt er aktiv Einfluss auf das System. Mit Hilfe der Qualität der Profile und einer Gewichtung der Anwendungen wird eine Qualitätsfunktion des Systems aufgestellt, die der FRM maximiert. Die Komplexität des zugrunde liegenden Optimierungsproblems des Systems ist NP-vollständig, da der Lösungsraum exponentiell zu der Anzahl der Profile wächst. In der ersten Phase wurden daher zwei einfache Optimierungsheuristiken implementiert. Eine Strategie, die nur für kleine Mengen von Profilen anwendbar ist, bestimmt durch eine vollständige Suche die optimale Konfiguration von Profilen und versucht diese zu aktivieren. Die zweite Strategie ist eine einfache Heuristik, die von der aktuellen Konfiguration – durch Aktivierung weniger Profile – eine lokale bessere Konfiguration auswählt.

Ursprünglich war eine Simulation der zu aktivierenden RTOS-Agenten geplant. Da eine komplexe Simulation zu viele Ressourcen verschwendet – und somit nicht den Anwendungen zur Verfügung steht – beschränkt sich der FRM auf eine Simulation des Ressourcenbedarfs mit anschließendem Akzeptanztest. Hierbei wird überprüft, ob alle Echtzeitschranken der Anwendungen bei der Rekonfiguration eingehalten werden können und genügend Ressourcen für die Anwendungen zur Verfügung stehen.

## 2.4 Überbelegung von Ressourcen

Der FRM optimiert nicht nur die benötigten RTOS-Systemagenten, indem er die von TEReCS bereitgestellten Profile aktiviert bzw. deaktiviert, sondern stellt außerdem zugesicherte, aber zeitweise ungenutzte Ressourcen unter harten Echtzeitbedingungen anderen selbstopimierenden Anwendungen zur Verfügung. Ressourcen können ungenutzt sein, wenn Anwendungen diese für Worst-Case Szenarien anfordern – aber zwischenzeitlich nicht benötigen – oder nur während Rekonfigurationen verwenden.

Durch den FRM wird eine weitere Ebene der Selbstoptimierung ermöglicht. Somit kann nicht nur innerhalb einer An-

wendung Selbstoptimierung oder eine speziell vordefinierte hierarchische Selbstoptimierung betrieben werden. Der FRM kann Ressourcen unter den Anwendungen vermitteln und somit die Qualität einer Gruppe von Anwendungen global erhöhen. Ein weiterer Vorteil ist, dass Selbstoptimierung auch zwischen Anwendungen erschlossen wird, zwischen denen keine explizite Abhängigkeit besteht. Durch die Modellierung im Profile Framework wird dies ermöglicht und eröffnet neues Potenzial zur Selbstoptimierung. Das RTOS passt sich somit nicht nur an die selbstoptimierenden Anwendungen an, sondern versucht die Qualität des Systems selbstständig zu optimieren, indem die Ressourcen möglichst so an die Anwendungen verteilt werden, dass die Qualität des Systems gesteigert wird. Als Grundlage für die Optimierung verwendet der FRM die erwähnte Qualitätsfunktion, in die die Qualitätskoeffizienten der Profile und eine Gewichtung der Anwendung eingehen. Liegt die Anwendung als Software-Implementation und als Variante in rekonfigurierbarer Hardware vor, aktiviert der FRM die aktuell qualitativ beste Implementationvariante.

# 3. ÄHNLICHE SYSTEME

Eingebettete Betriebssysteme [14], wie z. B. VxWorks [15], QNX [10], PURE [1] oder EPOS [8] haben eine feinkörnige Dienst-Architektur und ermöglichen eine auf die Anwendung angepasste Konfiguration der Dienste, womit sie für Architekturen mit geringen Ressourcen geeignet sind. Jedoch wird das dynamische Verhalten der einzelnen Knoten nicht unterstützt. Außerdem fehlt ein Mechanismus zur transparenten Kooperation zwischen den auf mobilen Knoten laufenden Tasks. Weiterhin sind solche Betriebssysteme nicht reflektorisch und können keine Selbstadaption auf neue Anforderungen der Anwendung oder Änderungen der Umgebung vornehmen.

Einige akademische Betriebssysteme wie etwa Apertos [16] können als reflektorisch bezeichnet werden. Diese reflektorischen (oder rekonfigurierbaren) Betriebssysteme haben die Möglichkeit, ihren aktuellen Zustand zu bewerten und darauf basierend ihre momentane Struktur zu ändern, um sich auf neue Anforderungen der Umgebung oder der Anwendung anzupassen. Ihre Dienste werden durch Objekte implementiert, die mit Meta-Objekten korrespondieren. Die Meta-Objekte analysieren das Objektverhalten und die Anforderungen der Anwendung zur Laufzeit und rekonfigurieren das Betriebssystem, um die Anforderungen besser zu unterstützen. Trotz dieser für das beschriebene Szenario wünschenswerten Eigenschaften haben die existierenden reflektorischen Betriebssysteme einen oder beide der oben erwähnten Nachteile: Sie wurden nicht konzipiert, um alle Charakteristika der beschriebenen Umgebung zu berücksichtigen, und bzw. oder stellen keine Transparenz für verteilte Anwendungen zur Verfügung.

Die Adaptive Resource Management (ARM) Middleware [6] stellt ein Optimierungsframework zur Beschreibung von verteilten Echtzeitsystemen für dynamische Umgebungen vor. Dieses Framework basiert ebenfalls auf einem Profil-Modell und modelliert dynamische Veränderungen anhand von Ankunftszeiten, Last- und Dienstgüte. Die Middleware versucht, die Systemauslastung zu maximieren, indem eine gültige und optimale Ressourcenbelegung gesucht wird. Der Übergang zwischen Profilen ist durch atomare Operationen mit konstanter und vernachlässigbarer Umschaltzeit realisiert. Dieser Ansatz ist beispielsweise für Echtzeit-Bildverarbeitung möglich (ein Szenario, in dem ARM eingesetzt wird), indem atomar zwischen Bildverarbeitungsalgorithmen umgeschaltet werden kann. Für selbstoptimierende mechatronische Controller ist dieser Ansatz nicht anwendbar. Zwischen Controller-Algorithmen kann nicht immer atomar rekonfiguriert werden; sie verlangen Überblenden oder können nur zu bestimmten Zeitpunkten atomar umgeschaltet werden.

Der Dynamic QoS Manager (DQM) [3] implementiert ein Quality of Service Level Model für weiche Echtzeitanwendungen. Er ist als Middleware auf einem Gerneral-Purpose-Betriebssystem realisiert. Der DQM optimiert die globale Auslastung, indem er entsprechende QoS Level der Anwendungen aktiviert. Der QoS Level einer Anwendung charakterisiert den Ressourcenbedarf und den Nutzen bei Aktivierung des Levels. Der Ressourcenbedarf wird hierbei durch Messungen zur Laufzeit bestimmt. Dieser Ansatz ist nur für weiche Echtzeitsysteme anwendbar.

# 4. ZUSAMMENFASSUNG

Dynamische Anwendungen verlangen nach innovativen neuen Dienstestrukturen auf der Ebene des Betriebssystems. Durch die vorgestellten Erweiterungen wurde unser Offline-konfigurierbares Echtzeitbetriebssystem DREAMS zu einem zur Laufzeit selbstoptimierenden Echtzeitbetriebssystem. Dieses deaktiviert ungenutzte Dienste und stellt freigewordene Ressourcen den Anwendungen zur Verfügung. Anhand einer selbstoptimierenden Anwendung und eines rekonfigurierbaren Echzeitkommunikationsagenten wird das System in [9] genauer erläutert.

# 5. REFERENCES


[1] D. Beuche, A. Guerrouat, H. Papajewski, W. Schröder-Preikschat, O. Spinczyk, and U. Spinczyk. On the development of object-oriented operating systems for deeply embedded systems - the pure project. In *ECOOP Workshops*, page 26, 1999.

[2] C. Böke. *Automatic Configuration of Real-Time Operating Systems and Real Time Communication Systems for Distributed Embedded Applications.* Dissertation, Universität Paderborn, Heinz Nixdorf Institut, Entwurf Paralleler Systeme, 2004. ISBN 3-935433-51-4.

[3] S. Brandt and G. J. Nutt. Flexible soft real-time processing in middleware. *Real-Time Systems*, 22(1-2):77–118, 2002.

[4] S. Burmester, M. Gehrke, H. Giese, and S. Oberthür. Making mechatronic agents resource-aware to enable safe dynamic resource allocation. In *Fourth ACM International Conference on Embedded Software (EMSOFT'2004)*, 27 - 29 Sept. 2004.

[5] C. Ditze. *Towards Operating System Synthesis.* Dissertation, Universität Paderborn, Heinz Nixdorf Institut, Entwurf Paralleler Systeme, 2000.

[6] K. Ecker, D. Juedes, L. Welch, D. Chelberg, C. Bruggeman, F. Drews, D. Fleeman, and D. Parrott. An optimization framework for dynamic, distributed real-time systems. *International Parallel and Distributed Processing Symposium (IPDPS03)*, page

111b, April 2003.

[7] U. Frank, H. Giese, F. Klein, O. Oberschelp, A. Schmidt, B. Schulz, H. Vöcking, and K. Witting. *Selbstoptimierende Systeme des Maschinenbaus - Definitionen und Konzepte.*, volume 155 of *HNI-Verlagsschriftenreihe.* Heinz Nixdorf Institut, Universität Paderborn, Paderborn, 2004.

[8] A. A. Fröhlich and W. Schröder-Preikschat. Epos: An object-oriented operating system. In *ECOOP Workshops*, page 27, 1999.

[9] B. Griese, S. Oberthür, and M. Porrmann. Component case study of a self-optimizing rcos/rtos system: A reconfigurable network service. In *Proceedings of International Embedded Systems Symposium 2005*, Manaos, Brazil, 15 - 17 Aug. 2005.

[10] R. Oakley. QNX microkernel technology: A scalable approach to realtime distributed and embedded system. In *Proc. of the Embedded Computer Conference (ECC'94)*, June 1994. http://www.qnx.com.

[11] S. Oberthür and C. Böke. Flexible resource management - a framework for self-optimizing real-time systems. In B. Kleinjohann, G. R. Gao, H. Kopetz, L. Kleinjohann, and A. Rettberg, editors, *Proceedings of IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES'04)*. Kluwer Academic Publishers, 23 - 26 Aug. 2004.

[12] S. Oberthür, C. Böke, and B. Griese. Dynamic online reconfiguration for customizable and self-optimizing operating systems. In *Fifth ACM International Conference on Embedded Software (EMSOFT'2005)*, 18 - 22 Sept. 2005. Jersey City, New Jersey.

[13] F. J. Rammig. Autonomic distributed real-time systems: Challenges and solutions. In *7th International Symposium on Object-oriented Real-time Distributed Computing, ISORC 2004*. IEEE Computer Society, IEEE Computer Society Press, 12 - 14 May 2004.

[14] J. A. Stankovic and R. Rajkumar. Real-time operating systems. *Real-Time Systems*, 28(2-3):237 – 253, Nov 2004.

[15] W. R. Systems. Vxworks 5.4 - product overview, June 1999.

[16] Y. Yokote. The apertos reflextive operating system: The concept and its implementation. In *OOPSLA Proceedings*, pages 414–434, 1992.