

# Eine offene Entwicklungsplattform

Die Java-Lösung von OSGi

**Kürzere Entwicklungszeiten, die zunehmende Komplexität der Anwendungen und die wachsende Anzahl der zu berücksichtigenden Technologien führen zu einem technologischen Umbruch bei Telematik- und Infotainment-Systemen.**

**Über 80 Unternehmen haben sich daher zu einem Konsortium zusammengeschlossen – der OSGi – und einen offenen Standard spezifiziert, der auf Java basiert.**

Von Joachim Ritter

Das Angebot fertig entwickelter und funktionsfähiger Telematik- und Infotainment-Systeme für Fahrzeuge ist vielseitig und groß. Dennoch gehen Fachleute davon aus, dass die nächste Generation dieser Systeme technisch auf grundlegend anderen Ansätzen beruhen wird – wieso? Weshalb beschäftigen sich derzeit alle großen Automobilhersteller und -zulieferer intensiv mit alternativen Basistechnologien für derartige Systeme? Wie sehen diese aus?

Für die Beantwortung dieser Fragen sind für den Markt der intelligenten Embedded-Systeme drei aktuelle Problemstellungen zu betrachten:

- zunehmende Komplexität der Anwendungen,
- kürzere Entwicklungszeiten,
- zunehmende Anzahl zu berücksichtigender Technologien.

Die Komplexität und Dichte der integrierten Funktionen nimmt rasant

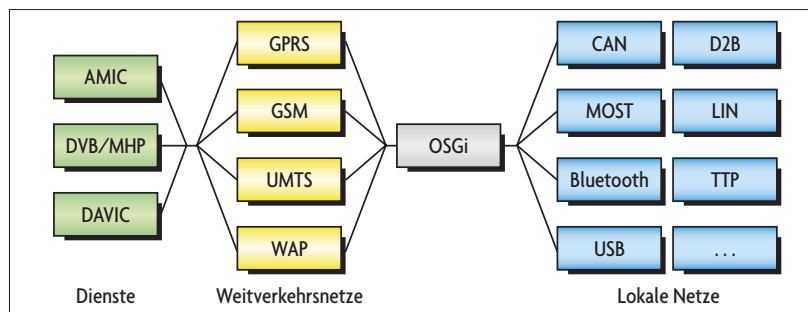
zu. Anwendungen aus dem Unterhaltungsbereich (z.B. Fernseher, DVD, Musik) und dem Internet (E-Mail, Surfen, Chatten) sollen ergänzend zu fahrzeugspezifischen Systemen (Navigation, Bordcomputer, Klima etc.) auf einer „Plattform“ zusammengeführt werden. Zudem werden den Geräten der nächsten Generation auch Dienste zur Nachrüstbarkeit, Administration sowie Fernzugriffe für Diagnose-, Wartungs- oder Datentransferfunktionen abverlangt. Hinzu kommt, dass die gewaltige Innovationskraft des Software- und Elektronikmarktes neben der ständigen Entwicklung neuer Technologien auch immer kürzer werdende Produktlebenszyklen verursacht. Um konkurrenzfähig zu bleiben, stehen dementsprechend Herstellern innovativer Systeme immer kürzere Entwicklungszeiten zur Verfügung.

Der dritte Punkt beschreibt eine Problemstellung, die auch auf Entwicklungsfeldern anderer Industriezweige

vorherrscht und intensiv diskutiert wird: Wie sollen die vielen verschiedenen Funktionsgruppen beziehungsweise Gerätetypen miteinander vernetzt werden?

Betrachtet man die jeweils speziellen Anwendungsanforderungen, dann hat sicherlich jedes Übertragungsverfahren seine Existenzberechtigung, und nicht wenige werden in den jeweiligen Einsatzbereichen zum Standard erhoben. CAN, MOST, USB, Bluetooth sind nur einige der nennenswerten Protokolle. Um einen durchgängigen Informationsfluss durch alle Anwendungsschichten zu gewährleisten, müssen also mehrere heterogene, verteilte Systeme miteinander vernetzt werden. Der Funktionsumfang einer zentralen Plattform wird dabei um in Software realisierte Netzwerkverbindungen erweitert, die nachfolgend „Service Gateway“ genannt werden.

Hersteller und Zulieferer müssen den Entwicklungsprozess der Software so gestalten, dass fertige Module wiederverwendet werden können und darüber hinaus eine produktive Entwicklungsumgebung geschaffen wird. Die Wiederverwendbarkeit von Softwaremodulen (und damit auch die Portierbarkeit) steigert z.B. die Sicherheit und spart Entwicklungsressourcen. Wenn man bereits bestehende Codes weiter verwenden kann, wird hierdurch die Entwicklung neuer Software schnell und effizient. Zu den Anforderungen an eine zentrale Softwareplattform der nächsten Generation gehört zunächst, dass diese nicht an eine bestimmte Anwendung gebunden sein darf; sie muss also offen sein für alle Anwendungen und Verfahren der Software-Entwicklung. Die rasche technische Entwicklung zieht die dynamische Nachrüstbarkeit von Funktionen und Diensten unmittelbar nach sich. Dass eine offene Plattform skalierbar und modular aufgebaut sein muss, ist eine selbstverständliche Voraussetzung, dass sie eine sichere und robuste Laufzeitumgebung darstellt, eine Herausforderung. Darüber hinaus sind Schnittstellen zur Fernparametrierung und -administration vorzusehen. Insgesamt muss eine solche Plattform von ihren Anwendern als Systemspezifikation akzeptiert und als Standard begriffen werden.



**Bild 1. Das OSGi-Framework vermittelt beliebige Dienste (Services) zwischen den Mobilfunk-Weitverkehrsnetzen und den lokalen Netzen im Fahrzeug.**

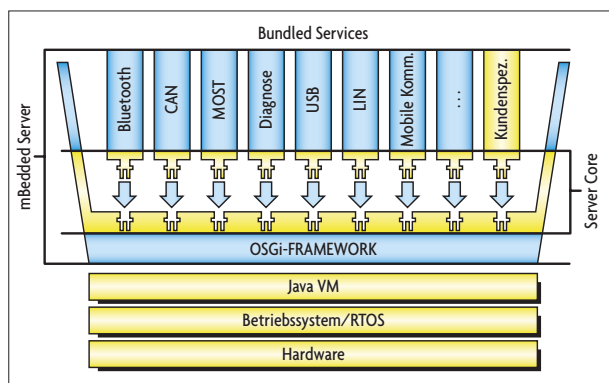
## Produktvielfalt als Motor

Im Automobilbereich ist die Produktvielfalt eines Herstellers in aller Regel groß – es gibt mehrere Fahrzeugtypen, die zudem in verschiedenen Ausstattungsklassen angeboten werden. Die Skalierbarkeit der eingebauten Systeme ist also von großer Bedeutung. Für die Technologieanbieter ist die Auswahl der Softwareplattform von entscheidender strategischer Bedeutung, denn Innovation und „State of the art“ lassen sich nur bei gleichzeitiger Zu-

beispielsweise Sun Microsystems, IBM, Lucent Technologies, Cisco, Nokia, Teia, Sonera, Deutsche Telekom, BMW, Motorola, NCI, Nortel Networks, Oracle, Philips, Sony, ProSyst, Toshiba sowie weitere 70 hochrangige Industrieunternehmen sind dort vertreten. Seit der Gründung 1997 hat sich das Konsortium zum Ziel gesetzt, einen gemeinsamen Kommunikationsstandard für die Vernetzung der unterschiedlichsten Geräte zu definieren. OSGi ist branchenübergreifend und kann im Home-Networking, in der Industrie-

produktion, im Auto oder der Medizin eingesetzt werden.

Das Resultat der Bestrebungen von OSGi liegt in Form einer Spezifikation des OSGi-Framework vor, deren Version 2.0 Anfang Oktober freigegeben wurde. Unter Einbezug der genannten Anforderungen wird hierin



**Bild 2. Software drives Hardware. Das OSGi-Framework bedient sich dazu des Konzepts der Virtual Machine (VM) von Java, mit dem die erforderliche Flexibilität für die Vielzahl der Dienste und Anforderungen erreicht wird.**

kunftsabsicherung verkaufen. Für standardisierte Umgebungen sind zudem mehr Entwickler verfügbar als für proprietäre Lösungen. Die Entwicklung neuer Anwendungen wird dadurch beschleunigt und damit kostengünstiger.

Ein weiterer Gesichtspunkt ist die Administrierbarkeit: Offene und verteilte Systeme müssen konfiguriert, administriert und gewartet werden. In vielen Applikationen fallen diese Aufgaben sicherlich Anbietern zu, die nicht gleichzeitig auch die Hersteller der Geräte oder Fahrzeuge sind. Das Service Gateway muss neben seinen anwendungsbezogenen Modulen demnach auch über definierte Schnittstellen zur Fernwartung verfügen.

## Das OSGi- Framework

Diese oben geschilderten Probleme des „Embedded-Markts“ für Service Gateways hat sich die OSGi (Open Service Gateway Initiative) zugewandt, auf deren Lösungsansatz im Folgenden näher eingegangen wird. Firmen wie

ein detailliertes System-Design für Service Gateways definiert, das bis hinunter zu den Application Programming Interfaces (API) reicht. Das Konsortium OSGi definiert dabei nur den Standard. Die Implementierung des „Frameworks“ ist den Anwendern überlassen, wobei die Verwendung professioneller und fertiger Implementierungen empfehlenswert ist, denn die Entwicklung eines Frameworks einschließlich der erforderlichen Werkzeuge kann mehrere hundert Mannjahre erfordern.

*Bild 1* verdeutlicht die Position des Service Gateways im Verbund verschiedener Datennetze und der darauf laufenden Protokolle. Die modulare Beschaffenheit des Gesamtsystems ermöglicht das Einbinden beliebiger Protokollstandards. Die Protokolltreiber werden durch Anwenderkomponenten miteinander verbunden, die ebenfalls in das Framework integriert werden. Der grundlegende Ansatz des Frameworks ist die Abstraktion aller Abhängigkeiten der Hardware und des Be-

triebssystems durch eine modular aufgebaute Softwareplattform. *Bild 2* veranschaulicht eine detailliertere Sicht des OSGi-Frameworks, das aus den Modulen „Server Core“, „Bundles“ und „Services“ besteht.

Der Server Core stellt fundamentale Funktionen zur Verwaltung und Administration von Bundles bereit. Ein Bundle ist ein Archiv, in dem Java-Code, nativer Code, statische Seiten (z.B. HTML) oder sonstige Dateien enthalten sein können. Sofern eine Klasse das Interface „BundleActivator“ enthält, handelt es sich um ein ausführbares Bundle, in dessen start()- und stop()-Routinen beliebige Java-Anwendungen initialisiert werden können. Um anderen Applikations- oder Treiber-Bundles einen Dienst verfügbar zu machen, können Services implementiert und im Framework registriert werden. Fortan können diese referenziert und aufgerufen werden. Auch Klassenbibliotheken sind untereinander referenzierbar.

Zur Standardausstattung eines Frameworks zählen die in *Tabelle 1* beschriebenen Bundles. Es handelt sich dabei um Funktionalitäten, die in den meisten Anwendungen benötigt werden. Eine OSGi-Framework-Implementierung liefert damit einen großen Funktionsumfang. Anwendungsbezogene Module und Protokolltreiber werden in Form von Bundles in das Framework geladen. Mittlerweile ist das Angebot fertiger Bundles sehr groß und deckt die Bedürfnisse vieler Branchen ab (*Tabelle 2*).

## Kleines Glossar

AMIC	Automobile Multimedia Interface Collaboration
CAN	Controller Area Network
D2B	Domestic Digital Bus
DAVIC	Digital Audio Video Interoperability Council
DVB	Digital Video Broadcast
GPRS	General Packet Radio System
GSM	Global System for Mobile Communication
LIN	Local Interconnect Network
MHP	Multimedia Home Platform
MOST	Media Oriented Systems Transport
OSGi	Open Service Gateway Initiative
TTP	Time Triggered Protocol
UMTS	Universal Mobile Telephone System
USB	Universal Serial Bus
WAP	Wireless Application Protocol

Bundle	Beschreibung
HTTP	Webserver mit integrierter Servlet Engine (Servlet 2.1 API) Dieses Bundle implementiert einen vollständigen Webserver, der auch Java-Servlets ausführen kann und somit alle Framework-Informationen und -Dienste über HTTP/HTML zugänglich macht.
Log	Dienste zum Setzen und Lesen von Log-Einträgen Events und benutzerdefinierte Einträge werden protokolliert.
Device Access	Erkennung neuer Geräte im Netz und Treiberverwaltung Der Device Access Manager erkennt neue Geräte im Netzwerk und kann für diese dann die erforderlichen Treiber bereitstellen, die ggf. auch dynamisch von einem entfernten System geladen werden können.
Config. Management (seit 2.0)	Standardisierte Konfigurationsschnittstelle Bundles können über diese Schnittstelle intern oder von außen konfiguriert werden. Bundle-bezogene Benutzeroberflächen zur Konfiguration können in Management-Programme geladen werden.
User Management (seit 2.0)	Dienste für die Benutzerverwaltung Das Bundle beinhaltet einen Service zum Verwalten von Benutzern. Daten werden in einer internen Datenbank gehalten. Authentifizierung, Rollenverteilung und Vergabe von Berechtigungen sind ebenfalls enthalten.
Preferences (seit 2.0)	Dienste für Benutzereinstellungen Sitzpositionen in Autos oder Punktstände von Spielen sind typische benutzerbezogene Informationen, die mit Hilfe dieses Bundles einfach verwaltet werden können.

Tabelle 1. Standard-Bundles des OSGi-Frameworks

### ► Java als Basistechnologie

Mit welchen Programmiersprachen und -verfahren lässt sich nun ein solcher Bezugsrahmen implementieren? Die OSGi stellte fest, dass die Programmiersprache „Java“ im Kontext der Anforderungen von Service Gateways derzeit die meisten Vorzüge aufzuweisen hat. Folglich wurde die Spezifikation 1.0 zunächst in Java formuliert, und auch in der Version 2.0 gibt es noch keine Erweiterung für andere Sprachansätze. Java für den Embedded-Bereich ist ein derzeit viel diskutiertes Thema und kann hier nicht umfassend erörtert werden. Da es sich jedoch um die „Schlüsseltechnologie“ des OSGi-Ansatzes handelt und maßgeblichen Einfluss auf die Gesamtarchitektur des integrierten Systems ausübt, sollen hier die Vor- und Nachteile Javas für den Einsatz in Service Gateways kurz beschrieben werden.

Die meisten Stärken und Schwächen Javas ergeben sich aus dem grundlegenden konzeptuellen Unterschied, der die Sprache maßgeblich von den im Embedded-Feld am häufigsten verwendeten Sprachen C/C++ abhebt. Darüber sollten auch die vielen syntaktischen Parallelen nicht hinwegtäuschen. Anders als in C/C++ wird der Sourcecode nicht in native Binärcodes compiliert, die dann direkt ausführbar sind, son-

dern zunächst in „Bytecode“ übersetzt. Dieser wird dann zur Laufzeit von der Java-Laufzeitumgebung, genannt Virtual Machine (VM), zunächst interpretiert und dann in Form nativer Aufrufe ausgeführt. Zu den wichtigsten Aspekten von Java im Kontext von Embedded-Systemen zählt, dass der Anwendungscode portierbar ist und Klassen dynamisch geladen werden können. Interessant ist auch die Möglichkeit, aus Java direkt auf die Hardware zugreifen zu können. Nach den bisherigen Erfahrungen können mit Java sichere, robuste und zudem echtzeitfähige Systeme erstellt werden, die hinsichtlich der Geschwindigkeit und des Speicherplatzbedarfs den heutigen Anforderungen gerecht werden. Die bestehenden Entwicklungsumgebungen sorgen für eine ausreichende Produktivität bei der Erstellung neuer Applikationen.

Die Virtual Machine (VM) abstrahiert weitestgehend von den Prozessor- und Betriebssystem-Eigenschaften. Durch diese Entkopplung ergibt sich die gute Portierbarkeit des Bytecodes und damit die Plattformunabhängigkeit von Java-Anwendungen. Mittlerweile

Kategorie	Bundles
Kommunikation	CAN, LON, EIB, EHS, X10, USB, HTTP, SMS, WAP, FTP, Jini, UPnP, RS 232, Mail, Bluetooth
Sicherheit	SSL, TLS, X501
Netzfunktionen	DHCP, SNMP, Jini-Lookup, UPnP Control Point

Tabelle 2. Auszug verfügbarer Bundles von ProSyst

gibt es für nahezu alle bekannten Betriebssysteme und Prozessoren VMs. Dies ist ein entscheidender Vorteil, da in Fahrzeugen die unterschiedlichsten Hardwaresysteme eingesetzt werden. Das Framework und erhebliche Teile der Anwendung können so schnell und effizient portiert und damit wiederverwendet werden.

Ein weiterer, ganz entscheidender Vorteil, der sich aus der Laufzeitinterpretation von Bytecode ergibt, ist das dynamische Laden von Klassen. So kann während des Ablaufes eines Programmes neuer Bytecode angefordert und unverzüglich ausgeführt werden – ein großer Vorteil, wenn man z.B. über Fernwartung neue Dienste oder Software-Updates verteilen will. Das OSGi-Framework macht gerade von dieser Eigenschaft regen Gebrauch, denn Services sind in Bytecode übersetzte Java-Anwendungen. Bei Zugriffen auf Hardwareschnittstellen wie z.B. auf einen seriellen Port oder ein Bussystem handelt es sich auch bei Java in der Regel um systemspezifische, proprietäre Lösungen. Diese müssen bei der Portierung den VM- und Betriebssystem-Eigenschaften angepasst werden.

### ► Offenheit versus Sicherheit – ein Problem?

Die Zuverlässigkeit der mobilen Systeme ist für die Fahrzeughersteller unheimlich wichtig. Java beinhaltet eine Vielzahl von Maßnahmen zur Steigerung der Sicherheit und Robustheit. Hierzu zählen integrierte Spracheigenschaften wie Exceptions, reine Objektorientierung, keine Globalvariablen und Zeiger, Verhinderung von Mehrfachvererbung und vieles mehr. Der gut strukturierte Aufbau und die Syntax machen Java zu einer schnell zu erlernenden und dennoch sehr „mächtigen“ Sprache, die wegen ihrer Beliebtheit mittlerweile über ein großes Reservoir an Klassenbibliotheken verfügt.

Die Leistungsanforderung zukünftiger Telematik-Plattformen ist jedoch sehr hoch. Daher ist die Ausführungs-

geschwindigkeit von Java eine relevante Kenngröße. Java-Code wird vor der Ausführung zunächst von der VM interpretiert, daher ist ein Java-Programm zwangsläufig immer langsamer als ein direkt ausgeführter Binärcode. Bei Vergleichen der Programmausführungszeiten sind binär compilierbare Sprachen wie C/C++ im Vorteil, je nach Anwendung sind diese 2- bis 10-mal schneller. Allerdings haben die Studien von Lutz Prechelt [1] gezeigt, dass die Kompetenz des Programmierers immer noch den größten Einfluss ausübt und solche Nachteile durchaus auszugleichen vermag. Um die Leistungsfähigkeit von Java zu steigern und die Technologie auch für Systeme mit sehr beschränkten Hardwareressourcen zugänglich zu machen, arbeiten sowohl die VM-Hersteller als auch die Chip-Designer intensiv an Lösungen zur Leistungssteigerung. Im Bereich der VMs gibt es grundsätzlich drei verschiedene Ansätze:

- „Just In Time“-Compiler (JIT),
- „Ahead of Time“-Compiler (oder „Upfront“-Compiler),
- „Dynamic Adaptive“-Compiler.

Allen drei Ansätzen ist das Prinzip gemein, Java-Bytecode schon vor dessen Ausführung durch die VM in Binärcode zu compilieren, um wiederholtes Interpretieren durch die VM zu verhindern. Diese Vorgehensweise führt zu erheblichen Steigerungen der „Performance“, sie ist jedoch in allen drei Fällen mit nennenswerten Nebeneffekten behaftet: JIT-Compiler halten den einmal compilierten Binärcode im RAM und brauchen dadurch zur Laufzeit ganz erheblich mehr Speicher (bis 32 Mbyte zusätzlich). „Ahead of Time“-Compiler haben einen höheren statischen Speicherbedarf, da die Binärcores wesentlich größer sind als Bytecode. Ferner ist durch das komplette Vorcompilieren jegliches dynamische Nachladen von Codes ausgeschlossen und somit für Anwendungen wie das OSGi-Framework uninteressant. Die „Dynamic Adaptive“-Compiler bieten hingegen einen guten Kompromiss aus Leistung und Speicherbedarf, da sie nur die am häufigsten verwendeten Code-Abschnitte in compiliert Form im Speicher halten.

Auch die Chip-Hersteller sehen große Potentiale in Java und entwerfen Java-fähige Prozessoren. Hier gibt es zwei Ansätze: reine Java-CPU's, die ausschließlich Bytecode verarbeiten können, und Java-Coprozessoren, die parallel zu herkömmlichen Prozessoren arbeiten und nur für die Java-Anwendungen angesprochen werden. Auch hier sind im Einzelfall die Nachteile zu beachten. Im Allgemeinen jedoch bringen die Hardware-Lösungen sehr gute Ergebnisse. Am Rande sei erwähnt, dass an einer Erweiterung der ARM-Prozessorarchitektur gearbeitet wird, um neben den beiden Befehlssätzen ARM und Thumb auch etwa 70 Prozent der rund 230 Java-Bytecode-Befehle direkt auf dem Chip zu unterstützen.

Was den Speicherbedarf betrifft, so schneiden Java-Programme im Vergleich zu anderen Sprachen schlechter ab. Die sichere und für den Programmierer einfachere, integrierte Speicherverwaltung der VMs wird mit einem erheblich größeren „Heap“ bezahlt. Java-Programme brauchen nicht selten mehr als das Doppelte an Speicher. Hinzu kommt noch der zusätzliche Flash- oder ROM-Bedarf, der durch die VM-Implementierung verursacht wird. Ferner führt die automatische Speicherbereinigung (garbage-collection) zu einem nicht deterministischen Verhalten der Applikation, was im Rahmen der Echtzeit-Fähigkeit problematisch ist.

### ► Echtzeit-Fähigkeit bleibt eine Herausforderung

In verschiedenen Anwendungsfällen besteht die Notwendigkeit der Echtzeit-Fähigkeit, d.h., das System muss bestimmte Antwortzeiten auf physikalische oder durch die Software ausgelöste Ereignisse garantieren können. Gerade bei maschinennahen Anwendungen, wie beispielsweise in der Automatisierungstechnik, müssen die Systeme streng deterministisch sein. Zwei Mechanismen sind für die Echtzeit-Fähigkeit eines Systems entscheidend: Multithreading oder -tasking und die Prioritätenverteilung der parallelen Ausführungspfade. Die Java-Spezifikation beinhaltet an dieser Stelle keine strikten Vorgaben und überlässt es den VM-Herstellern, diese Punkte im Zusammenhang mit dem unterlagerten Betriebssystem zu implementieren.

Bezüglich des Multithreading gibt es zwei Realisierungen: Java-Threads werden ausschließlich von der VM verwaltet und auf einen Betriebssystem-Task abgebildet (mapping). Hierbei ist die Echtzeit-Fähigkeit offensichtlich stark begrenzt, da zu viele

Softwareschichten zwischen dem Ereignis und der Bearbeitungsroutine liegen. Bei dem zweiten Ansatz werden die Java-Threads einzeln auf Betriebssystem-Threads abgebildet und unterliegen dadurch dessen Taskverwaltung. Hierbei ist die Prioritätenverwaltung des Betriebssystems entscheidend für die Grenzen der Echtzeit-Fähigkeit. Herkömmliche Betriebssysteme wie Windows NT oder Linux erlauben es Anwendungs-Threads nicht, höhere Prioritäten als Betriebssystem-Threads anzunehmen, was zu einem nicht-deterministischen Verhalten



**Dipl.-Ing. Joachim Ritter**

hat an der Berufsakademie Mosbach in Verbindung mit der Firma Braun GmbH, Kronberg, studiert und 1999 zum Dipl.-Ing. Elektrotechnik abgeschlossen. Während seiner fünfjährigen Anstellung bei Braun erwarb er fundierte Kenntnisse in hardwarenaher Softwareentwicklung sowie in Java. Seit September 2001 ist er als technischer Berater bei der ProSyst Software AG, Software-Anbieter für die internetbasierte Vernetzung stationärer und mobiler Elektrogeräte, angestellt.  
 ► E-Mail: j.ritter@prosynt.com

Anwendung	RAM (Mbyte)	Flash (Mbyte)	MIPS
Framework ohne Anwendungen	2	0,5	30
Diagnose- und Fernwartungsdienste ohne Grafik	4	2	80
Kommunikationsdienste und einfache Grafik	8	4	200
Multimedia-Anwendungen mit aufwendiger Grafik	>16	>8	>350

**Tabelle 3. Framework-Systemanforderungen**

führen kann. Das Verhalten steht bei diesem Prinzip in Abhängigkeit zur Systembelastung.

So genannte Echtzeit-Betriebssysteme (z.B. Siemens RMOS3) hingegen unterscheiden bei der Prioritätenverteilung nicht zwischen Anwendungs- und Betriebssystem-Threads und ermöglichen damit den tiefsten Eingriff in das Verhalten des Gesamtsystems. In einer solchen Umgebung kann Java hervorragende Antwortzeiten garantieren. Allen Ansätzen gemein ist die Problematik der Java-Speicherverwaltung (garbage-collection), die zu unregelmäßiger und starker Belastung der VM führt und die Echtzeit-Fähigkeit mitunter beeinträchtigen kann.

### ► Framework konkret

Die Bedeutung der oben beschriebenen Anforderungen an Java-Plattformen lassen sich im Bezug auf OSGi-Frameworks konkretisieren. Tests mit der Implementierung des OSGi-Frameworks der Firma ProSyst Software AG, genannt „mBedded Server 5.0“, resultierten in dem anwendungsbezogenen Anforderungsprofil an die Leistungsfähigkeit des Prozessors und die Ausstattungsmerkmale der Hardwareplattform, die in *Tabelle 3* dargestellt sind. Zu beachten ist, dass die Zielplattform hinsichtlich des Prozessors, des Betriebssystems und der VM zu bewerten ist.

Der „mBedded Server“ läuft auf nahezu allen gängigen Kombinationen (32-bit-Prozessoren und Kompatibilität zu JDK 1.1.7 vorausgesetzt). Hinsichtlich der Prozessoren werden derzeit u.a. StrongArm-, PPC-, x86-, SH3/4-Architekturen unterstützt. Neben allen Windows-Betriebssystemen und Linux können auch Echtzeit-Systeme wie VxWorks, QNX oder Jbed eingesetzt werden. Die Auswahl der VMs ist ebenfalls groß: Sun JDK 1.1.7 und höher, CVM, Personal Java, Geode, Jbed, J9 sind ein Teil der unterstützten Systeme.

me. Aus den Erläuterungen zum OSGi-Framework und der Umsetzung dieser Software in bereits laufenden

Projekten ergibt sich, dass die innovative Entwicklung einen Meilenstein erreicht hat.

### ► OSGi mit Perspektive

Die wachsende Mitgliederzahl der OSGi und die zunehmende Nachfrage nach Implementierungen zeigt, dass Service Gateways nach OSGi-Standard Marktakzeptanz finden. Mit anderen Worten liefert der OSGi-Ansatz Fahrzeugherstellern und Zulieferern einen Mehrwert, der die technologisch bedingten höheren Hardwareanforderungen übersteigt. Eine Vielzahl der eingangs dargelegten Problemstellungen, in denen sich Entwickler intelligenter Systeme der nächsten Generation befinden, werden durch den Java-basierten OSGi-Ansatz gelöst. Typische Zielplattformen sind Infotainment- und Entertainment-Systeme in Automobilen, aber auch mobile Router, Set-Top-Boxen, medizintechnische Geräte oder hochwertige Unterhaltungselektronik.

Die technische Machbarkeit wird durch die Programmiersprache Java sichergestellt, deren weitere Entwicklung sich positiv auf den Anwendungsbereich der Gateways durchschlagen wird. Umgekehrt wird Java durch OSGi-Gateways einen noch umfangreicheren Einzug in „embedded“-Systeme erhalten. jw

### Literatur

- [1] Prechelt, L.: Comparing Java vs. C/C++ Efficiency Differences to Inter-Personal Differences. Publikation der Fakultät für Informatik. Universität Karlsruhe. März 1999.
- [2] Brich, P.; Hinske, G.; Krause, K.-H.: Echtzeitprogrammierung in Java. Publicis MCD Verlag. Oktober 2000.
- [3] Perrier, V.: Embedded Java. Internet Publikation. The O'Reilly Network. August 2001, [www.oreillynet.com](http://www.oreillynet.com)