Zeitschrift: Comtec : Informations- und Telekommunikationstechnologie = information and telecommunication technology

Herausgeber: Swisscom

Band: 80 (2002)

Heft: 4

Artikel: Das Internet wird mobil

Autor: Sellin, Rüdiger

DOI: https://doi.org/10.5169/seals-877191

Java für mobile Endgeräte

Das Internet wird mobil

Die Programmiersprache Java ist aus dem Internet nicht mehr wegzudenken. Viele Homepages wurden mit Java-Animationen deutlich attraktiver. Mobile Internet-Anwendungen (wie seit einiger Zeit bei i-Mode in Japan) erhalten mit Java ganz neue Möglichkeiten. Wird Java nun auch im Mobilfunk, etwa in GPRS-fähigen Endgeräten, Einzug halten?

RÜDIGER SELLIN

Seit sich das Internet ab Mitte der 90er-Jahre zum Massenmedium entwickelte, wurden die Browser auf den PCs und Apple Macintoshs immer ausgeklügelter. Vor allem animierte Abläufe beim Aufrufen einer bestimmten Internet-Adresse in Form von Bewegungen oder Videosequenzen sind heute kaum mehr wegzudenken. Diese Animationen basieren fast ausschliesslich auf der Java-Technologie von Sun Microsystems und tragen wesentlich zur Attraktivität einer gut gemachten Homepage bei. Java wurde damit zum De-facto-Standard im Zusammenspiel vom Browser im Endgerät mit der Anwendung auf dem Internet-Server.

Mobile Zusatzdienste

In der weltweiten Telekommunikation werden heute in den Festnetzen – je nach untersuchtem Land – zwischen 50 und 80% der Umsätze mit Sprachdiensten und 20 bis 50% mit Datendiensten erzielt. Im Vergleich dazu liegt der Umsatzanteil von Datendiensten in Mobilfunknetzen bei maximal 5% – allerdings mit steigender Tendenz. Einen wichtigen Beitrag daran leisten die mobilen Zusatzdienste, die mit SMS (Short Message Service) und WAP (Wireless Application Protocol) aber erst am Anfang ihrer Möglichkeiten stehen. Mit EMS (Enhanced Message Service) und MMS (Multimedia Message Service) können auch Bilder bzw. Videosequenzen von einem Handy zum anderen gesendet werden. Heute geschieht dies noch abhängig vom Endgerät, sodass ein Austausch derartiger Nachrichten zwischen Handys unterschiedlicher Hersteller noch nicht möglich ist.

Gerade die Anwendungen sind die treibenden Kräfte für eine intensive Nutzung der Endgeräte – und damit auch der Mobilfunknetze, was neue Einnahmequellen erschliessen hilft. Ein Blick auf den i-Mode-Dienst von NTT DoCoMo zeigt, welche Anwendungen die gefragtesten sind (Tabelle 1). Besonders die grosse Beliebtheit des Unterhaltungsbereichs ist auffallend. Einerseits müssen die Mobilfunknetzbetreiber genügend Bandbreite bereitstellen, was in Europa mit der Erweiterung der bestehenden GSM-Netze durch GPRS (General Packet Radio Service) und dem Aufbau der UMTS-Netze (Universal Mobile Telecommunications System) ja auch geschieht. Andererseits sollten die Dienstanbieter (Service Provider) mit den Inhaltsanbietern (Content Provider) gemeinsam dafür sorgen, dass die ersten Anwendungen (Spiele, Videos, animierte Wegweiser usw.) auf den gängigen Endgeräten herstellerunabhängig und zuverlässig laufen. Dazu soll die so genannte Java™ 2 Micro Edition (kurz J2ME™), auch Wireless Java oder Mobile Java genannt, beitragen.

Anwendung	Anteile der Zugriffe
Mobile Unterhaltung (Entertainment)	43%
Telefonbuch	29%
Tickets für Veranstaltungen	9%
Tele-Banking	7%
Öffentliche Verkehrsmittel (Fahrpläne, Tickets)	6,5%
City Guide (Stadtpläne, Informationen)	4,4%

Tabelle 1. Die populärsten Anwendungen des NTT-DoCoMo-Dienstes «i-Mode».

Neue Ära mit Wireless Java

Java™ ist eine objektorientierte Programmiersprache, die sich mittlerweile in drei Ausgaben (Java Editions, bisweilen auch Java Platforms genannt, Bild 1) präsentiert:

- Die Java™ 2 Enterprise Edition (J2EE™) zum Aufbau und Betrieb eines Internet Business Server (Internet, Intranet, Extranet).
- Die Java™ 2 Standard Edition (J2SE™) für den breiten Desktop-Computer-Markt (PCs und Laptops mit mehr als 16 MByte RAM und einem Prozessor mit mindestens 64 Bit).
- Die Java™ 2 Micro Edition (J2ME™, das Synonym für Wireless Java) für kleine mobile Endgeräte mit begrenzten Ressourcen in Bezug auf Gewicht und Stromversorgung (Psions, Communicators, Handys, Spielkonsolen).

Für jede der drei Kategorien wurde ein Satz von Werkzeugen und Technologien entwickelt, die für eine Endgerätekategorie und/oder für eine bestimmte Anwendung optimiert wurden. Zu diesem Satz gehören die Java Virtual Machines (JVM), Klassenbibliotheken mit vordefinierten Objekten, Methoden und Application Programming Interfaces (APIs) sowie Werkzeuge für die Inbetriebsetzung einer Anwendung und die anwendungsspezifische Konfiguration des Endgeräts. Letzteres geschieht über einen Methodenaufruf vom Server auf das Endgerät, um eine einwandfreie Funktion der Java-Anwendung in der lokalen Umgebung sicherzustellen. Dieser Ablauf wird mit Remote Method Invocation (RMI) bezeichnet und ist von Java-Anwendungen vom Internet her bekannt.

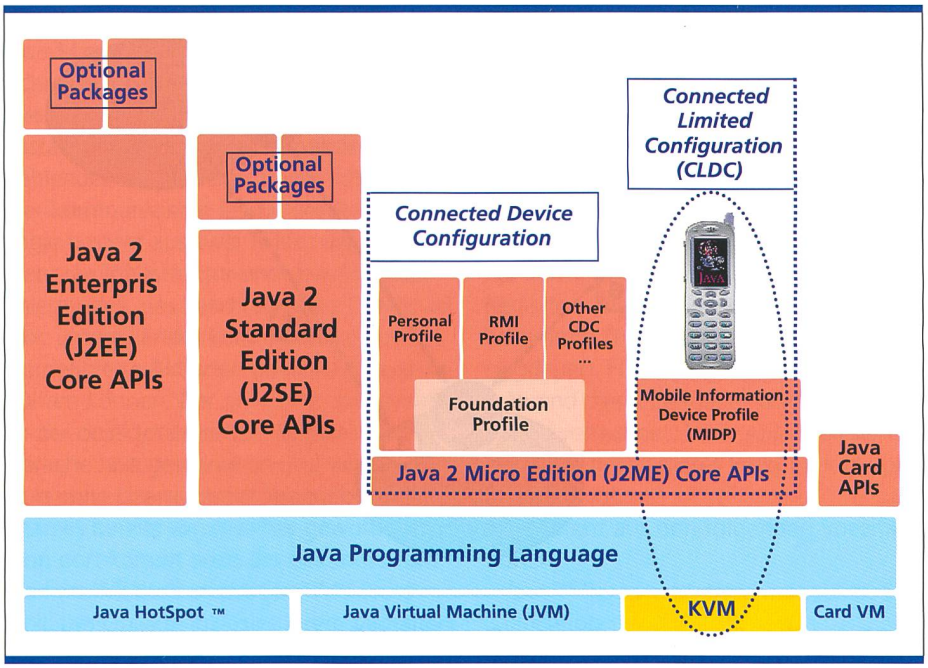

Bild 1. Drei verschiedene Java-Ausgaben – zugeschnitten auf den Endgerätetyp und die darauf laufende Anwendung (Quelle: Sun/modifiziert von Rüdiger Sellin).

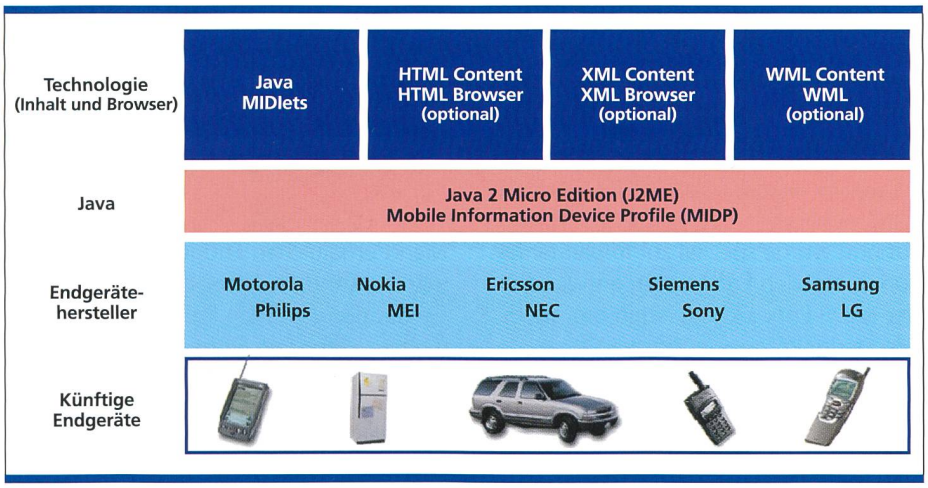

Bild 2. Einbindung von Java in verschiedene Browsertypen und mobile Endgeräte (Quelle: Sun).

Die drei Java-Editionen setzen auf dem Betriebssystem des jeweiligen Zentralprozessors (Host Operation System) jeweils drei Schichten auf, von denen die JVM die erste Schicht darstellt. Sie stellt das Bindeglied zwischen dem Betriebssystem und der eigentlichen Java-Technologie dar und ist sehr eng mit der nächsten Schicht «Configuration» verzahnt. Diese Schicht unterstützt eine grosse Anzahl von Anwendungen und definiert die kleinstmögliche Anzahl von Java-Plattformeigenschaften (Java Features, Core APIs) und Java-Klassenbibliotheken (Class Libraries). Sie konzentriert sozusagen den kleinsten gemeinsamen Nenner unter einer Vielzahl von Java-Funktionen in einem Gefäss. Die dritte Schicht hat die Bezeichnung «Profile» und beinhaltet einen Minimalsatz von anwendungsbezogenen APIs, die auf einer bestimmten Konfigurationsschicht aufsetzen und für diese optimiert wurden. Java-Anwendungen werden für ein spezifisches Profil entwickelt und sind auf verschiedene Endgeräte uneingeschränkt portierbar, solange das Endgerät das der Anwendung entsprechende Profil unterstützt. Ein Endgerät kann selbstverständlich mehrere Profile unterstützen und somit mehrere Anwendungen ermöglichen. Anschaulich gesprochen ist die Schicht «Profile» markt- und endgerätespezifisch, also vertikal fokussiert, während «Configuration» horizontal, also auf die Unterstützung mehrerer Anwendungen und Segmente ausgerichtet ist.

Da der Massenmarkt durch eine Vielzahl von unterschiedlichen Endgeräten gekennzeichnet ist, wird die J2ME™ noch weiter differenziert, um die stark differierenden Charakteristika hinlänglich genau abzubilden:

- Connected Device Configuration (CDC) für TV-Set-Top-Boxen, Internet-fähige Bildschirmtelefone, Highend Communicators und Psions sowie automobile Navigations- und Unterhaltungssysteme mit 2 bis 16 MByte RAM und 32- bis 64-Bit-Prozessor.
- Connected Limited Device Configuration (CLDC) für Mobiltelefone, Pager, Smart Phones (intelligente Telefone mit erweiterten Möglichkeiten für den Internet-Zugang oder Spiele) und Personal Digital Assistants (PDAs) mit 128 bis 512 kByte RAM und 16- bis 32-bit-Prozessor.

Tendenziell sind die teilweise durchaus durch den Kaufpreis bedingten Beschränkungen (geringes Gewicht, dadurch kleine Abmessungen und wenig Platz für grosse Displays oder Einschübe zur funktionalen Erweiterung, beschränkte Stromversorgung usw.) bei CLDC ausgeprägter als bei CDC. CDC und CLDC bilden die Basis für die Entwicklung von endgerätespezifischen Profilen, die auf den vordefinierten Konfigurationen aufsetzen. Ein Beispiel dafür ist das in den Bildern 1 und 2 erwähnte Mobile Information Device Profile (MIDP), auf das weiter unten noch näher eingegangen wird.

Daneben stellen Java-basierte Karten mit einem 8-Bit-Prozessor eine weitere Kategorie dar, in der gewisse Java-Funktionalitäten quasi fest verdrahtet vorhanden sind. Wegen der beschränkten Funktionalität beanspruchen diese Karten keine eigene Java-Edition, benutzen aber gleichwohl Java als Programmiersprache. Besonders für den Massenmarkt könnte man sich hier eine Reihe von Anwendungen, wie beispielsweise kleine Spielzeuge und Smart Cards (Zugang zu bestimmten Diensten, Servern oder auch nur zu Hotelzimmern) vorstellen.

Java 2 Micro Edition (J2ME)

Wie aus Bild 1 ersichtlich sind alle Komponenten und Systemschichten einer Java-Edition aufeinander abgestimmt. Dies gilt besonders für die J2ME, die be-

reits in der JVM-Schicht unnötige Features weglässt, nicht benötigte Funktionalität also vermeidet und damit die Beschränkungen mobiler Endgeräte berücksichtigt. Während die CDC noch die normale Java Virtual Machine (JVM) nutzt, wurde die JVM für die CLDC abgespeckt und mit KVM bezeichnet: K steht für Kilo und trägt dem Umstand Rechnung, dass bei kleineren Endgeräten der verfügbare Speicherplatz eher in Kilo- als in Megabyte gemessen wird. Das Abspecken wurde bei den Java-fähigen Karten noch weiter vorangetrieben, was in der funktional beschränkten CardVM reflektiert wird.
Sowohl die CDC als auch die CLDC sind bis heute die einzigen Konfigurationen, die innerhalb der J2ME definiert wurden. Dies geschah mit dem Hintergedanken, nicht zu viele unterschiedliche Profile zuzulassen, um Portabilität und Funktionstüchtigkeit uneingeschränkt garantieren zu können. Die CLDC ist eine Untermenge der CDC, die J2ME ihrerseits eine Untermenge der J2SE. Jede von der J2SE geerbte Objektklasse muss in der J2ME mit der Originaldefinition exakt übereinstimmen oder zumindest eine Teilmenge (Subset) der Originaldefinition darstellen. Zusätzlich wurden für die J2ME eine – allerdings wegen der Kompatibilität zu anderen Java-Editionen möglichst begrenzte – Anzahl von Zusatzfunktionen definiert, die den spezifischen Anforderungen der kleinen Endgeräte entsprechen. Im Design der KVM und der anschliessenden praktischen Erprobung hat sich gezeigt, dass eine zu beschränkte Auslegung der KVM ohne zusätzliche Systemklassen zu viel dynamischen Speicherplatz (Random Access Memory, RAM) belegt, was die Kosten in die Höhe treibt und die Geschwindigkeit des Endgeräts wegen des dann erforderlichen Downloads von Objekten senkt – beides unerwünschte Eigenschaften. Hingegen brachte die Erweiterung der KVM um Klassenbibliotheken mit spezifischen Objekten und Methoden einen deutlich geringeren RAM-Bedarf und eine höhere Geschwindigkeit mit sich, was den Benutzer schliesslich im täglichen Gebrauch seines Java-fähigen Endgerätes erfreuen dürfte.

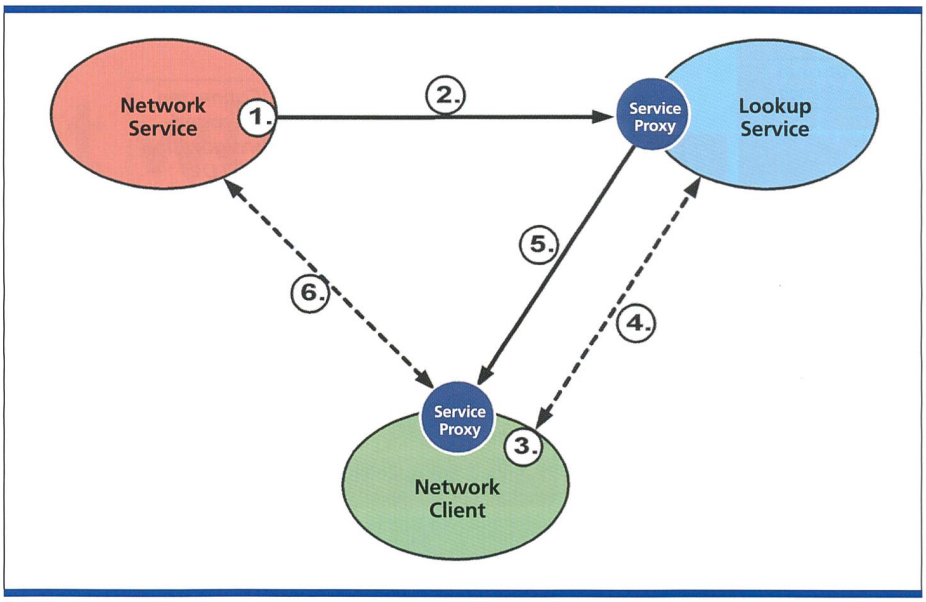

Bild 3. Jini-Flussdiagramm. 1. Discover (Entdecken): Der Network Service entdeckt die verfügbaren Look-up-Services. 2. Join (Anschliessen, Verbinden): Der Network Service sendet seinen Service Proxy zum lookup-Services (Anmerkung: Statt «Join» scheint der Begriff «Register» hier angebrachter zu sein, da unten dieser Begriff referenziert wird). 3. Discover (Entdecken): Der Network Client entdeckt die verfügbaren (registrierten) Look-up-Services. 4. lookup- (Nachschauen): Der Network Client sendet einen Request (Anforderung) zum lookup-Service, um den gewünschten Dienst in Form eines Service Proxy zu erhalten. 5. Receive (Empfangen): Der lookup-Service sendet den gewünschten Service Proxy zum Network Client. 6. Use (Benutzen): Der Network Client nutzt den Network Service über den empfangenen Service Proxy (Quelle: Sun modifiziert von Rüdiger Sellin).

MIDP – ein Profil für mobile Endgeräte

Das Mobile Information Device Profile (MIDP) setzt auf dem CLDC-Teil der J2ME auf und wurde speziell zur Implementation in mobilen Endgeräten entwickelt. Der MIDP-Entwicklung lagen drei wichtige Designkriterien zugrunde:
- Ein einmal geschriebener Programmiercode ist auf unterschiedlichen Endgeräten lauffähig (entsprechend der Java-Philosophie der vollen Portabilität).
- Die Darstellung auf den Endgeräten soll grafisch abwechslungsreich und der angebotene Inhalt interaktiv sein (ähnlich wie bei Java-basierten Internetseiten).
- Es werden mehrere Inhaltstypen unterstützt (z. B. mobile Spiele und Videos), die miteinander verknüpft werden können (etwa eine Videosequenz als Animation innerhalb eines Spiels).

Es sollte allerdings nicht verschwiegen werden, dass MIDP ähnlich wie CLDC eine recht generische Funktionalität definiert, um deren Anwendung auf möglichst vielen Endgeräten zu erlauben. Diese Endgeräte sind daher auf MIDP-Stufe recht allgemein gehalten und reichen von Handys über Navigationssysteme bis hin zu Autos und Schiffen (Bild 2). Die zurzeit gültige Version ist MIDP 1.0.3, die auch überwiegend in den am Markt erhältlichen Endgeräten eingesetzt wird. Unter dem Arbeitstitel «MIDP New Generation» werden im Laufe des Jahres weitere Versionen (vermutlich unter der Bezeichnung 2.x) folgen. Diese sollen besser auf spezifische GSM/GPRS-Funktionen abgestimmt werden, um die Integration von Java in den grossen GSM zu vereinfachen. Als neue MIDP-Funktionen sind unter anderem geplant:
- SIM-Toolkit-Integration
- SMS-Unterstützung
- Möglichkeit zum Anschluss einer Java Card
- Kommunikation mit dem GSM-Netz (z. B. für Java-basierte Location Based Services wichtig)
- End-to-End-Security (gerade für E-Banking- und M-Commerce-Anwendungen von Bedeutung)
- Streaming-Kommunikation (zwecks Herunterladens von Video-Sequenzen und Animationen)
- Dialer- und Browser-Schnittstelle
- Unterstützung von Sounds (Java-basierte Klingeltöne, zusammen mit der Streaming-Kommunikation Java-basierte Multimedia-Nachrichten)

Jini als weitere Java-Technologie

Sun Microsystems als Lizenzgeber beschreibt Jini als «eine Softwareinitiative, die Java-fähige Endgeräte ohne komplexe Installationen oder Kopfschmerzen wegen fehlender oder falscher Treiber untereinander kommunizieren lässt». Die Jini-Technologie besteht aus zwei Teilen, der so genannten Infrastruktur und dem Programmiermodell, das beschreibt, wie Clients ad hoc untereinander kommunizieren und vordefinierte Aktionen gemeinsam ausführen können. Der Jini-Code ist ebenfalls in der objektorientierten Programmiersprache Java geschrieben und wird von Sun ohne Lizenzgebühr abgegeben, wie man es bereits von Java her gewohnt ist. Von dort kommt auch das bekannte Java Remote-Method-Invocation(RMI)-Protokoll, um Objekte inklusive ihrem Verhalten im Netz zu bewegen. Ein solches Objekt wird hier Proxy oder Service- Proxy genannt, da dieser Proxy im Prinzip ja einen Service realisiert. Es besteht aus Dienstattributen und Anweisungen, wie der Proxy (das Objekt) im Netz zu kommunizieren hat. Mit Hilfe der Prozesse «Discover» (Entdecken) und «Join» (Zusammenkommen) können Dienste im Netz gefunden und regis-triert werden. Registrieren bedeutet in einer Jini-Umgebung, dass ein bestimmter Dienst seinen Service-Proxy (oder eine definierte Untermenge davon) an alle so genannten lookup-Services im Netz sendet. Unter einem «lookup-Service» kann man sich ein Verzeichnis (Directory) oder einen Index mit allen im Netz verfügbaren Diensten und ihren Proxies vorstellen. Wenn ein Client einen Dienst via RMI aufruft, so sendet der Dienst seinen Service-Proxy zu diesem Client. Mit diesem Proxy ist es dem Client möglich, mit dem gewünschten Service zu kommunizieren und dessen Eigenschaften zu nutzen (gesamter Ablauf: Bild 3). Für die Verrechnung könnte eine Art Leasingmodell zur Anwendung gelangen, bei dem die Regis-trierung des Service-Proxy an eine bestimmte Zeitdauer gebunden ist (Leasingdauer). Für die Nutzung des Dienstes während dieser Zeitdauer wird ein fester Betrag verrechnet. Nach Ablauf dieser Frist wird der Eintrag dieses Proxy im lookup-Service gelöscht.

Jini besteht aus den folgenden Spezifikationen:

- Java™ Remote Method Invocation Specification
- Java™ Object Serialization Specification
- Java™ Discovery and Join Specification
- Java™ Device Architecture Specification
- Java™ Distributed Events Specification
- Java™ Distributed Leasing Specification
- Java™ lookup-Service Specification
- Java™ lookup-Attribute Schema Specification
- Java™ Entry Specification
- Java™ Transaction Specification

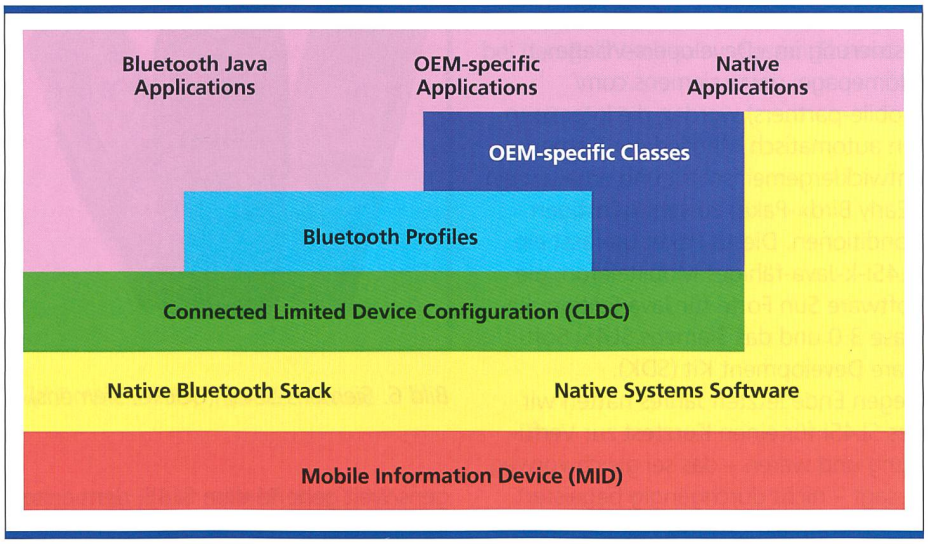

Bild 4. Zusammenspiel von Java und Bluetooth (Quelle: Rüdiger Sellin).

Weitere Ergänzungen kommen laufend hinzu, wobei sich alle Anbieter von Jini-basierten Lösungen verpflichten müssen, die Grundspezifikationen einzuhalten. Jini-Programmierer tun also gut daran, das Verhalten ihrer Jini-Objekte streng konform nachzubilden, nicht zuletzt aus Kompatibilitätsgründen der Jini-Lösungen untereinander.

Alternativen bzw. ergänzende Technologien zu Jini sind

- XML (eXtended Markup Language), die den Austausch von Informationen in einer vordefinierten Art und Weise erlaubt.
- UDDI, die eine Hilfe bei der Suche nach verfügbaren Diensten im jeweiligen Netz darstellen kann.

Bei XML scheiden sich übrigens häufig die Geister. Die einen sehen darin die Wunderwaffe bei der Beseitigung von Schwierigkeiten im Datenaustausch, etwa zwischen dem Hersteller einer Ware und dessen Zulieferer oder bei der Lagerbewirtschaftung. Andere werfen XML vor, dass die Sicherheit im Datenaustausch höher gewertet wird als Spontanität oder Flexibilität. Diese Attribute entsprechen eher dem Charakter einer mobilen Telekomdienstleistung.

Mit Jini lassen sich netzwerkbasierte Dienste realisieren, die auf der Software-Architektur von Jini aufbauen. So könnten beispielsweise Mobilfunknetzbetreiber Zusatzdienste basierend auf Jini anbieten, etwa das Ausdrucken von umfangreichen E-Mail-Attachments oder Faxnachrichten vom Handy des Benutzers auf einem öffentlichen Drucker oder einem Faxgerät in einem Restaurant. Jini ist die ideale Ergänzung zu Bluetooth (siehe nächster Abschnitt): Jini stellt die effektiven Softwarefunktionen und Bluetooth die lokale Funkschnittstelle bereit. Detaillierte Informationen zu Jini sind in den entsprechenden Whitepapers enthalten (Homepage: www.sun.com/jini/whitepapers).

Neben MIDP wird noch ein weiteres Profil zur Einbindung von Bluetooth erstellt (Bild 4). Bluetooth ist ein Protokoll zur drahtlosen Datenübertragung mit bis zu 1 Mbit/s und einem Radius von rund 10 m. Dieser Wert gilt für geschlossene Büroräume, grössere Hallen lassen Reichweiten von bis zu 100 m zu. Mit dem Java-Bluetooth-Profil kann die Java-Anwendung auf alle bestehenden Bluetooth-Funktionen und -Eigenschaften zurückgreifen, was einer erhöhten Kommunikationsfähigkeit im lokalen Bereich zugute kommt. Aber auch hier wird deutlich, dass die Handyhersteller eigene Java-Anwendungen erstellen und dazu auch herstellerspezifische Profile und Objektklassen einsetzen – nicht zuletzt zur Unterscheidung von ihren Mitbewerbern.

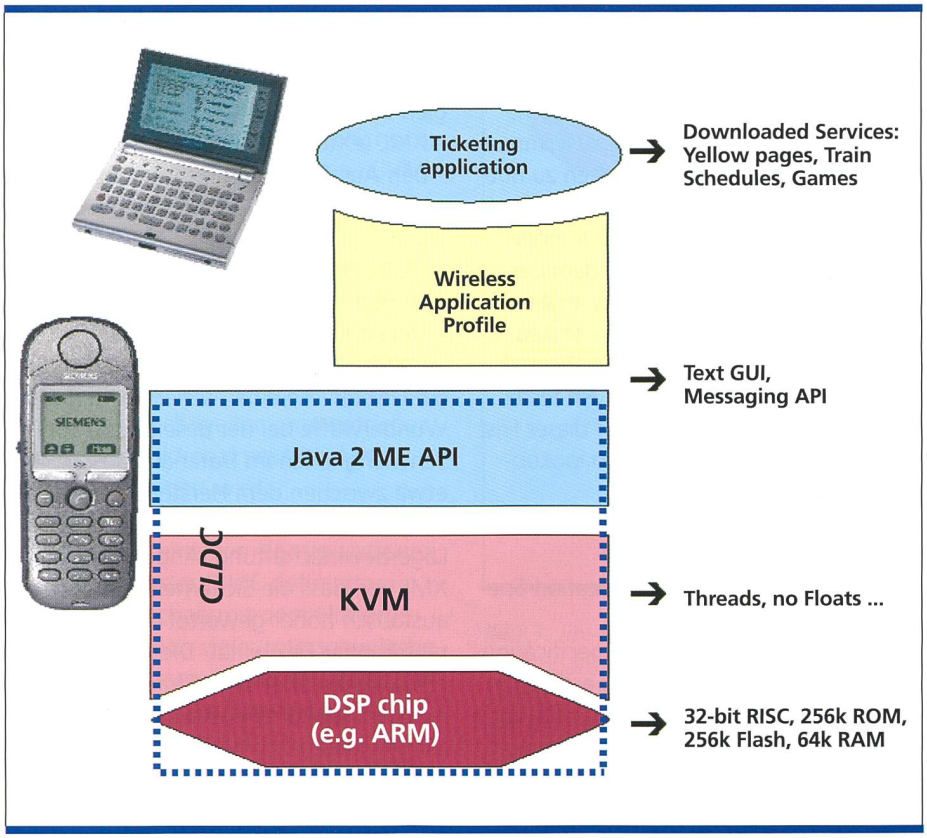

Bild 5. Einbindung von Siemens-spezifischen Java-Anwendungen in die Java 2 Micro Edition J2ME (Quelle: Siemens/modifiziert von Rüdiger Sellin).

Erste Implementierungen in Handys von Siemens und Nokia

Ein gutes Beispiel für diesen Implementierungsansatz liefert die Firma Siemens, welche die J2ME mit eigenen Elementen speziell für den mobilen Entertainmentbereich ergänzt hat (Bild 5). Die bereits seit 1999 währende Zusammenarbeit des Siemens-Geschäftsbereichs ICM (Homepage: www.siemens-mobile.de) mit Sun Microsystems führte Anfang 2000 zu einem ersten Entwicklungsauftrag, bei dem ein SL35 modifiziert und mit ersten J2ME-Funktionalitäten erweitert wurde. Mit dem SL45i brachte Siemens im Oktober letzten Jahres das erste kommerziell verfügbare GSM-Java-Handy der Welt auf den deutschen Markt (Bild 6). Grund genug für Siemens, ein in Telekomkreisen eher ungewöhnliches Partnerschaftsprogramm vorzustellen: «Early Bird» bietet allen interessierten Programmierern die Möglichkeit, ihre Entwicklungsideen in mobile Java-Anwendungen umzusetzen. Programme mit guten Marktchancen, die mittels «Early Bird» entwickelt werden, erhalten durch Siemens ICM volle Unterstützung bis zur Marktreife. Die Online-Vertriebskanäle von Siemens können zur Markteinführung dieser Programme ebenso genutzt werden wie die Möglichkeit zur Kontaktaufnahme mit den Netzbetreibern. Nach der Online-Registrierung im «Developers-Village» (Homepage: www.siemens.com/mobile-partners) werden die Interessenten automatisch Mitglied der Siemens-Entwicklergemeinschaft und erhalten ein «Early Bird»-Paket zu sehr günstigen Konditionen. Dieses Paket umfasst ein SL45i-k-Java-fähiges Mobiltelefon, die Software Sun Forte für Java Edition, Release 3.0 und das Siemens SL45i Software Development Kit (SDK).

Gegen Ende letzten Jahres hatten wir das SL45i für einen Kurztest zur Verfügung und waren – das sei gleich vorweg gesagt – nicht durchgängig begeistert. Gleichwohl faszinierend war die Tatsache, ein kleines Wunderwerk der Technik mit Java-Technologie in den Händen zu halten. Zu den Java-Anwendungen des SL45i gehören Spiele, ein erweiterter Taschenrechner, Tabellenvorlagen, eine Wettervorhersage und ein Cocktail-Manager mit Rezeptvorschlägen. Dieser Manager lässt allerdings nach dem Aufstarten mehr als eine Minute auf sich warten, bis er sich rührt und auf dem Display erscheint. Auch die anderen Java-Anwendungen starten recht langsam und bereiteten durchaus keine uneingeschränkte Freude über Wireless Java. Den Vogel schoss in dieser Beziehung die Anwendung World Wide Weather ab, die eine Weltkarte anzeigt und über WAP die aktuellen Wetterdaten zu den wichtigsten Metropolen einholt. So sollte Mitte Dezember in London eine Temperatur von 36 °C herrschen – was Zweifel an der Funktionstüchtigkeit hervorrief. Apropos WAP: Der WAP-Browser entspricht noch der Version 1.1 und ist damit nicht mehr up to date. Auch das Programm Micro-Mail zum Empfangen und Versenden von E-Mails funktionierte je nach Provider nicht immer. Zudem vermissten wir die Möglichkeit, über den Enhanced Messaging Service (EMS) ähnlich wie bei den neuen Ericsson-Handys Bilder versenden zu können. Dies gelingt nur mit dem eingebauten Siemens-Standard und das auch nur auf andere SL-Typen aus dem gleichen Hause. Etwas versöhnlich stimmte uns da die Möglichkeit, weitere Java-Anwendungen aus dem Netz herunterzuladen (z. B. ab der Homepage: www.midnight.org). Die Ausstattung des SL45i entspricht übrigens weit gehend dem SL45, dem ersten Mobiltelefon mit MP3-Player. Offenbar hat das SL45i auch gleich die Kinderkrankheiten des Vorgängers übernommen, so etwa das extrem langsame Herunterladen von MP3-Files ab PC über die serielle Schnittstelle auf die auswechselbare Speicherkarte im Handy. Auch von GPRS ist weit und breit nichts zu sehen – schade, denn gerade dieser Datenturbo würde die Wartezeiten beim Herunterladen von neuen Java-Anwendungen deutlich verkürzen.

Bild 6. Siemens SL45i (Quelle: Siemens).

Kurz vor Redaktionsschluss ereilte uns noch die Meldung, dass nun auch Nokia im Laufe des zweiten Quartals 2002 ein Java-enabled GSM-Handy auf den Markt bringen will. Das Nokia 7650 (Bild 7) unterstützt das MIDP von Sun und bietet ein Farbdisplay im Format 35×41 mm bei einer Auflösung von 176×208 Pixel. Eine unter der Tastatur versteckt eingebaute Kamera mit einer Auflösung von 640×480 Pixel (VGA) erlaubt es, selbst geschossene Bilder sofort via MMS (Multimedia Messaging Service) oder E-Mail zu versenden. Die diversen Möglichkeiten, neben dem klassischen GSM auch über GPRS (General Packet Radio Service), Bluetooth- oder Infrarot-Schnittstelle mit der Aussenwelt Kontakt aufzunehmen, erfreuen den Benutzer. Zudem können ambitionierte Entwickler für das eingebaute Betriebssystem «Symbian 6.1» eigene Softwareanwendungen in C++ schreiben.
Doch wo Licht ist, da ist auch Schatten. Das Nokia 7650 bietet keine Erweiterungsmöglichkeiten für zusätzliche Speicherplätze (Memory Slots) bei nur 4 MByte lokal verfügbarem Speicherplatz. Um also den mageren lokalen Speicherplatz zu leeren, müssen die gespeicherten Bilder schnell über MMS oder E-Mail/GPRS auf die Reise geschickt werden. Die fehlenden Erweiterungsslots verhindern ausserdem die Nutzung des Handys als MP3-Player. Last, but not least enthält die eingebaute Bluetooth-Schnittstelle kein Profil zur Unterstützung von Bluetooth-fähigen Freisprecheinrichtungen (wie das etwa bei den besseren Handys vom Erzrivalen Ericsson üblich ist). Weitere Infos sind unter der Homepage: www.nokia.com/phones/7650/index.html abrufbar.

Bild 7. Nokia 7650 (Quelle: Nokia).

Fazit

Trotz erster Ernüchterungen beim Einsatz der neuen Technologie bleibt die starke Hoffnung, dass der Einsatz von Wireless Java die europäische GSM-Welt ähnlich revolutionieren wird, wie das bei i-Mode in Japan bereits geschah. Die grösste Beschränkung dürfte dabei wohl die langsame Erweiterung des GSM-Standards darstellen, die aber unumgänglich ist, wenn Wireless Java auf allen GSM-Handys einwandfrei funktionieren soll. Dies wäre dann ein wichtiger Meilenstein auf dem Weg zum Erfolg. Die optimale Synthese scheint die Kombination Bluetooth (für die lokale Netztechnik) und Jini (für den Aufruf der lokalen Dienste) zu sein. Bis zum reibungslosen Funktionieren müssen wir uns allerdings noch mit den Geburtswehen beschäftigen – wie bei anderen neuen Technologien auch. [1,3]

Weiterführende Links für generelle Informationen:

www.sun.com/products/midp
www.sun.com/jini
www.javamobiles.com

Danksagung

Der Autor dankt Emil Diethelm und Reto Gantenbein von der Firma Sun Microsystems (Schweiz) AG für die Unterstützung bei den Recherchen zu diesem Thema.

Rüdiger Sellin, *Dipl.-Ing., ist PR-Manager bei den Portal Services von Swisscom Mobile. Davor war er unter anderem als Senior Consultant, Product Manager und Systems Engineer bei verschiedenen Telco- und IT-Firmen beschäftigt. Seit 1992 ist er ausserdem als Publizist, Trainer und Berater für verschiedene Firmen aus den Gebieten Telekommunikation und angewandte Informatik tätig.*

Summary

Java for Mobile Terminal Equipment

Nowadays the Internet would be unthinkable without the programming language Java. Many home pages have been made significantly more attractive thanks to Java animations. Java has opened up completely new horizons for mobile Internet applications (such as i-Mode which has been in use for some time in Japan). Is Java now set to penetrate the mobile communications world, for instance with GPRS-compatible terminal equipment?
Since the Internet evolved into a mass medium in the mid-1990s, browsers on PCs and Macs have become increasingly sophisticated. In particular, animated processes for accessing an Internet address in the form of movements or video sequences have become a standard feature. These animations are almost exclusively based on Java technology from Sun Microsystems and play a key role in enhancing the attractiveness of well-designed home pages. Java has now become the de facto standard for interaction between the browser in the terminal device and the application on the Internet server.