

JavaPhoneAPI

SeminarJavaAktuell
14.Mai2001
GeorgWaldispühl

Inhaltsverzeichnis

1.Vorwort	2
2.Übersicht	2
3.DieKomponentenderJavaPhoneAPI	3
3.1AddressBook	3
3.2UserProfile	3
3.3Calendar	4
3.4NetworkDatagram	4
3.5Install	4
3.6PowerMonitor	4
3.7PowerManagement	4
3.8JavaTelephonyAPI(JTAPI)	5
3.8.1JTAPICorePackage	5
3.8.2JTAPICallControlPackage	6
3.8.3JTAPIPhonePackage	6
3.8.4JTAPIMobilePackage	6
3.8.5JTAPIPrivateDataPackage	6
3.8.6JTAPIMediaPackage	7
3.8.7JTAPICallCenterPackage	7
4.OptionalePersonalJavaInterfaces	7
4.1Communication	7
4.2SecureSocketsLayer(SSL)	7
5.DasJavaTelephonyCallModel	8
6.Schlusswort	9

1.Vorwort

DerKommunikationsmarktistamboomenundverzahntsichimmermehrmitder Computerwelt,dasnormaleTelefonistvomAussterbenbedrohtunddieZukunftscheintder rechnergestütztenKommunikationzugehören. IndiesemUmfeldsolltensichdieHerstellervonHandy'sundvonTelefonapplikationendie Händereiben,wennSundieJavaPhoneAPIlanciert,dennmannimmtan,dasssichvieles vereinfachenundverbilligenlässtmitihr.SimpleApplikationensollensehneinfacherstellt werdenkönnen,währenddiefürkomplexereTelefonapplikationennötigenFunktionen trotzdemzurVerfügungstehen.Zudem sollendiemitHilfevonJTAPIgeschriebenen ApplikationenportabelüberverschiedenenComputerplattformenundTelefonsystemesein. DiefolgendenAbschnittesollendieMöglichkeitenderJavaPhoneAPIaufzeigen,undzwar so,wieSUNespropagiert.NurdasVor-undSchlusswortsindnichtaufSUN's Dokumentationengestützt.

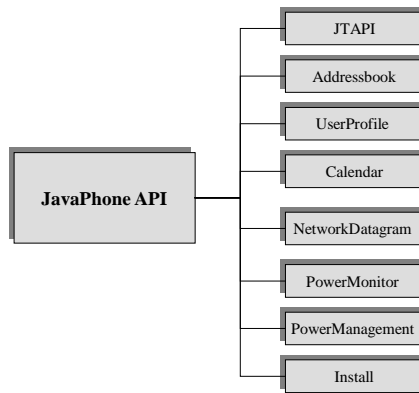
2.Übersicht

DieJavaPhoneAPIisteineErweiterungderPersonalJavaPlattformundwurdeineinem offenenProzessvonSunMicrosystemsinZusammenarbeitmitandernExpertender Telekommunikationsbrancheentwickelt.InKombinationmitderPersonalJavaAPIbietetsie eineidealeEntwicklungsplattformfürTelefonapplikationenund-diensteimBereichder mobilenundinternetbasiertenKommunikation.

SosollenunterAnderemfolgendeLeuteangesprochenwerden:

- HerstellermobilerGerätekönnenschnellerTelefonprodukteaufdenMarktbringen durchdasNutzenderVorteile,dieplattformunabhängigeundstandardisierte Software-Komponentenmitsichbringen.
- Netzwerk-OperatorenkönnenTelefondiensteinBezugaufSicherheitund Erweiterbarkeitverbessern,zudemwirdderenVerteilungdurchdie Plattformunabhängigkeiteinfacherundtransparenter.
- Software-EntwicklerkönnenschnellerkostengünstigeundportableApplikationen entwickeln.SieprofitierenvonZeit-undKostenersparnissendank wiederverwendbarerSoftware.

Die JavaPhone API besteht gemäss der aktuellen Spezifikation 1.0 aus folgenden Komponenten:



Die JavaPhone API besteht hauptsächlich aus Interfaces, die dann je nach Bedarf kombiniert werden können (es gibt keine Referenzimplementation). Man spricht in diesem Zusammenhang von Profilen, die in Bündeln einer oder mehrerer JavaAPI's sind. Es gibt z.B. das „WirelessProfile“ und das „ScreenphoneProfile“, die gewisse Komponenten der JavaPhone API zwingend benötigen oder dann optional brauchen können.

3. Die Komponente der JavaPhone API

3.1 AddressBook

Die AddressBook API erlaubt Applikationen die Speicherung und den Zugriff auf Kontaktinformationen wie z.B. Telefonnummern, Adressen, E-Mail-Adressen und andere Informationen über Individuen, Gruppen und Organisationen. In Handy's wäre dies im konkreten Fall der Zugriff auf die Namen und Adressen, die auf der SIM-Karte gespeichert sind.

Das Hauptziel beim Entwurf der AddressBook API war, etwas Einfaches und Schlankes zu entwickeln, damit es in ressourcenlimitierte Geräte passen würde. Eine Speicherklasse ist für alle Arten von zu speichernden Daten wie Kontakte, Kalender- und Todo einträge zuständig.

3.2 UserProfile

Die UserProfile API versorgt eine Applikation mit Informationen über den momentanen Benutzer oder Besitzer des Gerätes. Wenn wieder ein Handy als Beispiel genommen wird, erlaubt die UserProfile API den Zugriff auf die SIM-Karte, wo Informationen über den Besitzer wie Name und andere Angaben zu finden sein werden. Diese Information wird im gleichen Format wie in der AddressBook API übermittelt.

3.3 Calendar

Die Calendar API gibt Informationen über den Zeitplan. Es geht also um die Speicherung und das Zugreifen auf Informationen wie z.B. Daten und Einträge zu Aufgaben, die Beschreibungen und Wiederholungsinformationen beinhalten können. Todo-Einträge geben Zugriff auf Notizen und Aufgaben, die noch gemacht werden sollen. Auch hier wurde beim Entwurf darauf geachtet, alle einfach und schlank zu halten, damit der Einsatz auch in ressourcenlimitierten Geräten gewährleistet ist, und die Speicherung erfolgt ebenfalls über eine Speicherklasse für alle Arten von Einträgen.

3.4 NetworkDatagram

Die NetworkDatagram API ermöglicht die Entwicklung von Nachrichtendiensten zur Nachrichtenübermittlung, unabhängig vom physikalischen Netzwerk oder dem Träger. Erst wenn eine Nachricht gesendet wird, wird das Netzwerk oder der Träger ausgewählt. Beim Handy könnte der Short Message Service (SMS) gebraucht werden. So müssen Entwickler nur das Adressierungs- und Datenprotokoll kennen, während die Details des Transports versteckt werden. Deshalb müssen dann Applikationen nicht neu geschrieben werden, um auf verschiedenen Telefonen oder Netzwerken zu laufen. Beispiele für schnurlose Nachrichtendienste sind neben dem bereits genannten SMS auch WAP oder auch ein bestimmter Empfänger (ein Server oder Service).

3.5 Install

Dieses Interface ermöglicht das Installieren und Entfernen von Applikationen, wobei das Bilden von Packages und Jar-Files sowie auch Versionskontrolle der vorhandenen Applikation unterstützt wird.

3.6 PowerMonitor

Die PowerMonitor API überwacht die vorhandene Energie eines Gerätes. Die Applikation kann mit Hilfe dieser API den Batterie-Status und die geschätzte Restlebensdauer angeben, sie kann feststellen, ob eine externe Energiequelle benutzt wird und sie benachrichtigt, wenn das Gerät in einen anderen Energie-Modus wechselt oder wenn ein Service aufgrund zu tiefer Batterieleistung nicht mehr ausgeführt werden kann. So kann die Applikation ihre Aktivitäten der zur Verfügung stehenden Energie anpassen.

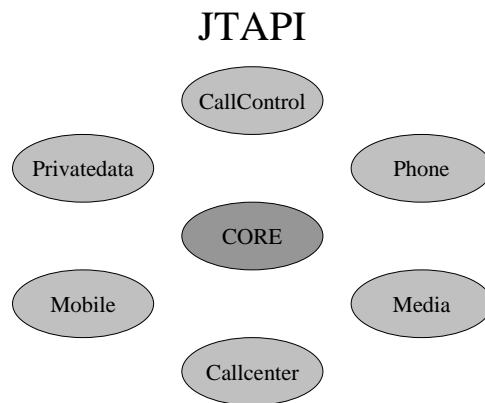
3.7 PowerManagement

Applikationen können die PowerManagement API benutzen, um den Betriebszustand eines Gerätes festzustellen, z.B. ob ein Gerät eingeschaltet ist und die Reaktionen auf verschiedene Betriebsarten wie volle Leistung, effiziente Betriebsweise, Warte- und Schlafmodus bestimmen.

3.8 Java Telephony API (JTAPI)

Die Java Telephony API (im Folgenden JTAPI genannt) ist entworfen worden, um verschiedensten Telefonapplikationen passende Funktionalität zu verleihen. Dies wird erreicht, indem ein Core Package mit einfachen Telefonfunktionen und eine Reihe von Erweiterungen (optional packages) mit komplexeren Funktionen gebildet wurden. Die Entwicklung der JTAPI Spezifikationen im kurzen Überblick:

- Version 1.0 November 1996
- Version 1.1 Februar 1997
- Version 1.2 Dezember 1997
- Version 1.3 April 1999



3.8.1 JTAPI Core Package

Dieses Package wird bei jeder Anwendung von JTAPI zwingend benötigt und übernimmt die grundlegenden Funktionen wie Anrufemachen, antworten und aufhängen. Für einfache Anwendungen reicht bereits das Core Package alleine, sodass man sich nicht mit den Details der Erweiterungs-Packages herum schlagen muss.

3.8.2 JTAPI CallControl Package

Dieses Package bietet weiterführende, detailliertere Funktionen als Erweiterung zum Core Package an, unter anderem das Weiterleiten, Umleiten, Parken, Halten von Anrufen, Übernehmen von Anrufen an eine andere Nummer sowie Konferenzfunktionen. Ausserdem können präzisere Informationen über einen Anruf gewonnen werden, da viele neue Statistiken bereits im Core Package vorhandenen Statistiken hinzukommen, die Zustände von Verbindungen und Endgeräten beschreiben.

3.8.3 JTAPI Phone Package

Dieses optionale Package stellt Methoden zur Verfügung, um die aktuellen Einstellungen von Komponenten eines Telefons wie Klingeltöneinstellungen, Knöpfe, Mikrophone, Anzeige, Beleuchtung und Lautsprecher etc. von einer Applikation aus abzufragen und zu manipulieren. Man kann z.B. die Lautstärke des Klingels steuern oder das Drücken von Telefonknöpfen simulieren.

Die einzelnen Komponenten können in Gruppen zusammengefasst werden, und ein Terminal kann mehrere solche Gruppen besitzen, was dann Sinn macht, wenn die eine Gruppe für das Telefongerät auf dem Tisch gebildet wurde, und die andere für das „virtuelle“ Telefon auf dem Computer. Es kann aber nur eine Gruppe aktiv sein.

3.8.4 JTAPI Mobile Package

Im Mobile Package werden die Eigenschaften eines Telefons (siehe z.B. das JTAPI Phone Package) wie Klingeln, Knöpfe und Aufhängen an die speziellen Bedürfnisse in der mobilen Telefonie angepasst. Dabei wurden nur die nötigsten Funktionen mitgenommen, um möglichst wenig Ressourcen zu verbrauchen. Weiter werden Funktionen zum Auswählen eines Netzes und zum Steuern eines eventuellen eingebauten Radios angeboten.

3.8.5 JTAPI PrivateData Package

Damit kann eine Applikation direkt mit der darunterliegenden Hardware kommunizieren und diese steuern. Wenn dieses Package genutzt wird, dann ist die Applikation allerdings an die darunterliegende Schicht gebunden.

3.8.6 JTAPIMediaPackage

Dieses Package gibt Applikationen die Möglichkeit, auf verschiedene Medien in Zusammenhang mit Telefonapplikationen zuzugreifen (z. B. Fax etc.). Es gibt eine Basis-API, welche alle Arten von Medien unterstützt. Darauf abgestützt gibt es noch eine Voice API, die gängige sprachbasierte Telefonapplikationen unterstützt. Die Voice API braucht dabei nur bei Bedarf eingesetzt zu werden und ist aufgrund der Beschränkung auf die grundlegenden Funktionen einfach zu handhaben. Einsatzgebiete für sie sind das Weiterleiten von Sprache aus dem Telefon in die Audio-Hardware des Computers, sodass dieser als Telefon benutzt werden kann. Auf diese Weise kann der Computer Nachrichten automatisch beantworten, aufzeichnen und bei bestimmten Signalen die gewünschten Aktionen ausführen.

3.8.7 JTAPICallCenterPackage

Dieses Package ermöglicht das Weiterleiten von Anrufen zu ihren Zieldestinationen. Weiter können Anrufe automatisch an Agenten (Telefonoperatoren) verteilt werden; falls kein Agent frei ist, wird eine Warteschlange gesetzt. Eine andere Funktion ist das automatische Wählen von Nummern durch die Applikation: Wenn sich niemand meldet, wird die nächste Nummer in der Liste gewählt, falls jemand abnimmt, wird der Anruf an einen Agenten übergeben (Einsatz: Eine Reihe von Kundenanrufen).

4. Optionale Personal Java Interfaces

4.1 Communication

Die Communication API kann von Applikationen benutzt werden, um auf serielle oder parallele Ports auf dem Gerät zuzugreifen. Falls das Gerät aber gar keine der Applikation zugänglichen Ports besitzt, dann braucht diese API natürlich nicht vorhanden zu sein. Wenn das Gerät z. B. ein zugängliches Modem oder Drucker hat, dann wird darauf über das `javax.comm` Package zugegriffen, sprich diese API hier wird nicht gebraucht.

4.2 SecureSocketsLayer (SSL)

Dies ist ein optionales Package, welches sichere Kommunikation über TCP/IP ermöglicht. Es enthält Unterstützung für die client- und serverseitige Benutzung der `SecureSockets` und Zertifizierung.

5. Das Java Telephony Call Model

Das JTAPICallModel besteht aus 6 Java Objekten, die in den Interfaces des JTAPICore packages definiert werden. Jedes Objekt repräsentiert entweder eine reale oder eine logische Einheit aus der Telefonwelt.

- **Provider Objekt:**
Ein Provider versteckt die servicespezifischen Aspekte des Telefonsystems und ermöglichtes Java Applikationen und Applets, geräteunabhängig in dem Telefonsystem miteinander zu interagieren.
- **Call Objekt**
Dieses Objekt repräsentiert einen Telefonanruf, die Information, die zwischen dem Serviceprovider und den Anrufteilnehmern fließt. Ein Telefonanruf enthält ein Call Objekt und keine bis mehrere Verbindungen (üblicherweise zwei für einen normalen Anruf, drei oder mehr Verbindungen ergeben eine Konferenzverbindung).
- **Address Objekt**
Dieses Objekt repräsentiert eine Telefonnummer, es ist eine Abstraktion für das logische Ende eines Anrufes.
- **Connection Objekt**
Mit diesem Objekt wird die Kommunikationsverbindung zwischen einem Call und einem Address Objekt gebildet.
- **Terminal Objekt**
Das Terminal Objekt steht für ein Gerät (z. B. ein Telefon) und seine Eigenschaften. Es kann mehrere Address Objekte (Telefonnummern) haben, und man kann es als den physikalischen Endpunkt eines Anrufes betrachten.
- **TerminalConnection Objekt**
Dieses Objekt steuert die Beziehung zwischen einer Verbindung und dem physikalischen Endpunkt eines Anrufes (Terminal Objekt), indem es verschiedene Zustände annehmen kann.

6. Schlusswort

Das Schlusswort stützt sich auf Erfahrungsberichte der Siemens Schweiz AG mit JTAPI 1.2. Es ist denkbar, dass ein Teil der vorgebrachten Kritik mit der neuen Version 1.3 behoben wurde. Das neue der Version 1.3 ist aber vor allem der weitere Ausbau der API, das Vorgehenskonzept selbst, das sich wohl kaum geändert.

Für die JavaPhone API hat Sundie Werbetrömmel ziemlich laut geschlagen, wer jetzt aber glaubt, schnell und einfach zu einem fertigen Produkt zu kommen, hat sich getäuscht. Wie schon gesagt, besteht die JavaPhone API hauptsächlich aus Interfaces, und deren Implementation, eine aufwendige Arbeit, ist den Entwicklern überlassen. Obwohl Sun gerne von Wiederverwendbarkeit und vom Zurückgreifen auf vorhandenen Code spricht, beschränkt sich das in diesem Fall auf das Benutzende Interface-Sammlung. Aber das ist trotz dem schon ein Malet was, denn wenn nicht viel von der Telefonie versteht, ist dankbar dafür, auf einen Schnittstellenvorschlag zurückgreifen zu können. Nach dem man sich dann einige Zeit mit der Thematik befasst hat und vielleicht eine erste Implementation hinter sich hat, findet man aber die noch vorhandenen Lücken und weiss schon genau, was man das nächste Mal realisieren möchte. Zu diesem Zeitpunkt kann man sich dann ernsthaft überlegen, ob nicht alle aus der JavaPhone API benutzten Element weg geworfen und die entsprechenden Teile selbst, den eigenen Bedürfnissen entsprechend, neu geschrieben werden sollen. Denn der Aufwand, der damit einherkommt, steht in keinem Verhältnis zu all den selbst geschriebenen Klassen, wäre also nicht sehr gross, und danach hätte man dafür eine Lösung nach Mass.

Es ist erstaunlich, dass SUN in derart grosses und umfangreiches Paket anbietet, ohne dabei eine Referenzimplementation gemacht zu haben. Wie wollen sie denn sodie kritischen Punkte finden? Die Erfahrung aus einer Implementation könnte die Qualität der Interfaces sicher einigeanheben.

Schade ist es auch, dass die Packages nicht immer die gewünschte Funktionalität bieten. Man möchte also z.B. eine Call-Center Applikation entwickeln und dafür auf das Call Center-Package zurückgreifen, muss dann aber feststellen, dass dieses nur einen Teil der gewünschten Funktionen bietet. Dann macht man sich halt doch selbst an's Entwickeln von besser passenden Interfaces.

Für einfache Anwendungen kann man die JavaPhone API gut einsetzen, wenn aber komplexere Anwendungen entstehen sollen, dann kann schnell ein Malet was fehlen. Es ist halt schwierig, in einem Bereich mit sehr unterschiedlichen Bedürfnissen etwas schaffen zu wollen, das allen passt. Schlussendlich muss doch jeder wieder seine eigenen Wege gehen, um seine Wunschlösung zu erhalten. Vielleicht ist es Sun auch deshalb noch nicht gelungen, ein Beispiel einer erfolgreichen Implementation oder einen Hersteller, der eine JTAPI-Implementation als Produkt anbietet, auf ihrer Homepage zu zeigen, es werden bloss einige Firmen zitiert, die geloben, die JavaPhone API so einzusetzen, wie es Sun gerne hört: Günstig und schnell Applikationen erstellen. Doch das wird die Zukunft erst noch beweisen müssen, im Moment ist es nicht der Fall.

Java TV API

Luc Benninger
luc@evolootion.ch

Inhaltsverzeichnis

1. Einleitung	13
2. Einführung	13
2.1. Die Heimkehr von Java.....	13
2.2. Grundlegendes zum Java TV API.....	13
2.3. Die Softwareumgebung	14
2.4. Hardwareumgebung.....	15
2.5. Typische Anwendungen.....	16
3. Technische Informationen	18
3.1. Packages im Java TV API	18
3.2. Services und Service Information (SI).....	19
3.3. Xlets	20
3.4. Broadcast Data API.....	21
4. Schlusswort und Ausblick	22

1. Einleitung

Bereits Anfangs Neunziger wollte man die Entwicklung der Fernsehtechnologie vorantreiben. Die Schaffung interaktiven Fernsehens war damals die Wunschvorstellung vieler. „Video-on-demand“ – also die freie Wahl des Fernsehprogramms – galt als nahe Zukunft. Bis heute wurde diese Ziele aber nicht erreicht. Als Hauptgründe dafür können vor allem die beinahe unerschwinglichen Entwicklungskosten angeführt werden, die sich schliesslich auch in den Endgeräten zu negativ bemerkbar machten.

Auf jeden Fall hatten aber auch diese Projekte ihr gutes. So floss viel der gewonnenen Erfahrung in die Entwicklung des Internets ein. Ein Erfahrungsaustausch, der jetzt in umgekehrter Richtung wieder wirkt - diesmal unterstützt durch Internet Standards wie Java. Fernsehgeräte mit java-fähigen Fernsehempfängern ermöglichen nicht nur passives Zuschauen, sondern erlaubt dem Fernsehzuschauer auch, interaktiv ins Geschehen einzugreifen.

2. Einführung

2.1. Die Heimkehr von Java

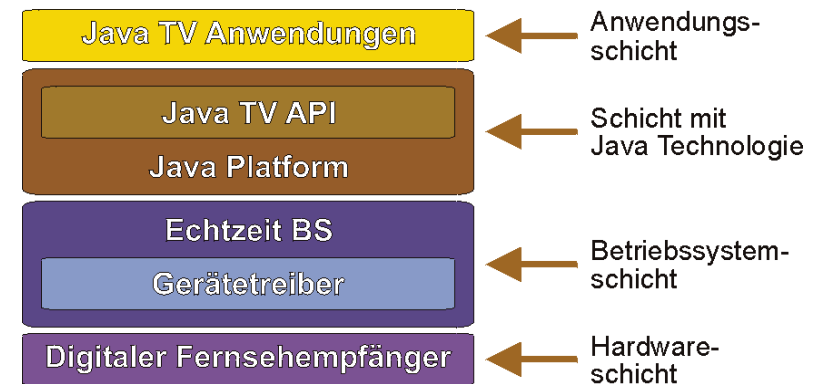
Bevor das Internet in der Medien-Landschaft auftauchte, war die Entwicklung der Sprache Java darauf ausgerichtet, eine möglichst einfache Programmierung unterschiedlichster Haushaltsgeräte, wie zum Beispiel Videorecorder, Fernseher oder Kaffemaschinen, zu ermöglichen. Nach einem längeren Umweg ins Internet ist Java nun wieder bei seinen eigentlichen Wurzeln gelandet und Sun Microsystems hat diese Rückkehr mit der Veröffentlichung des Java TV API noch gefestigt

2.2. Grundlegendes zum Java TV API

Das Java TV API ist eine Erweiterung zu der Standard Java Plattform. Die Entwicklung daran begann anfangs 1997. Federführend war Sun Microsystems und einige der wichtigsten Firmen der digitalen Fernseh-Industrie. Ziel war es, Softwareentwicklern Zugang zu den neuartigen Funktionen von digitalen Fernsehempfängern zu gewährleisten. In der digitalen Fernsehwelt werden solche Empfänger zu spezialisierten Computern, die digitale Inhalte von Fernseh Anbietern oder auch dem Internet verwalten und dem Benutzer in gewünschter Form präsentieren. Die fertige Java TV API Spezifikation 1.0 wurde Mitte 1999 veröffentlicht. Seit November 2000 lässt sich zudem eine Referenz-Implementation von der Sun-Homepage herunterladen.

2.3. Die Softwareumgebung

Die Softwareumgebung in einem digitalen Fernsehempfänger beinhaltet typischerweise die folgenden Komponenten:



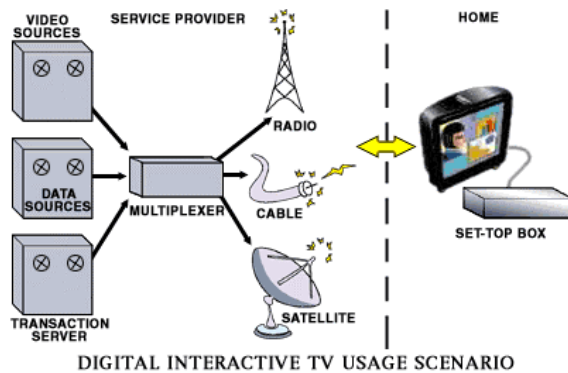
Anwendungen, die auf einem digitalen Fernsehempfänger laufen, benutzen das Java TV API und Java Klassen, die in der restlichen Java Plattform enthalten sind. In unserem Fall besteht die Java Plattform typischerweise aus der Java 2 Micro Edition (J2ME). Diese zeichnet sich insbesondere durch ihren geringen Speicherbedarf aus. Das Java TV API abstrahiert die Steuerung über die Hardware des Fernsehgerätes und benutzt das Java Media Framework (JMF) um die empfangenen Daten zu verarbeiten und anzuzeigen.

Wann immer möglich vertraut das Java TV API auf die restliche Softwareumgebung und die dort zur Verfügung gestellten allgemeinen APIs. So zum Beispiel bei Klassen zum Abspeichern von Daten oder zum Kommunizieren über ein Netzwerk. In einigen Fällen kann gewisse Funktionalität auch anderen Java-Erweiterungen anvertraut werden, so kann zum Beispiel bei speziellen Fernsehgeräten mit Telefonfunktionalität das Java Phone API zum Einsatz kommen.

Die Eigenschaft, dass Java Byte-Code durch die virtuellen Maschine auf allen Plattformen lauffähig ist, kommt der Fernsehindustrie entgegen. Die Anzahl verschiedener Hersteller mit zumeist proprietären Betriebssystemen ist hier bedeutend grösser als beispielsweise in der Computerindustrie. Die Hersteller ihrerseits können auf ein grosses Heer an Softwareentwickler zählen, die Java bereits vom Internet her kennen und so ohne grosse Einarbeitungszeit Programme für die neuen Fernsehgeräte schreiben können.

2.4. Hardwareumgebung

Das Java TV API wurde entworfen, um eine Abstraktion auf die Hardware von Fernsehempfängern zu bieten. Fernsehempfänger können entweder analoge, digitale oder beide Arten von Signalen verarbeiten. Ein digitales Signal lässt eine grössere Vielfalt an zu übertragenden Daten zu als ein analoges. Es kann zusätzlichem zu den Bild- und Tondaten Informationen gänzlich anderer Art beinhalten, zum Beispiel eine Java Anwendung. Digitale Empfänger kann man entweder gleich in einen neuen (digitalen) Fernseher eingebaut kaufen, oder aber als separate sogenannte Set-Top Box, die an den herkömmlichen Fernseher angeschlossen wird. Untenstehend sieht man den Weg, den ein digitales Signal geht, bevor es z.B. als Bild auf dem Fernseher landet:



Da es viele Fernsehempfänger mit sehr unterschiedlichen Fähigkeiten gibt, macht es Sinn, diese in drei Kategorien einzuteilen:

- **Erweiterter Empfänger**
Dieser Typ beherrscht sämtliche Funktionalitäten, die unsere heute modernen Fernseher auch anbieten. Zusätzlich ist er aber auch noch in der Lage, lokal Java Anwendungen abzuarbeiten.
- **Interaktiver Empfänger**
Wie der Name schon impliziert kann ein interaktiver Empfänger auch Daten versenden und so mit einem Server kommunizieren. Er unterstützt also Anwendungen wie zum Beispiel Video-on-demand, E-Mail oder E-Commerce.
- **Multi-Netzwerk Empfänger**
Diese Art von Empfänger bietet nicht nur einen Antwortkanal, sondern die volle Netzwerkfunktionalität, wie wir sie vom Internet her kennen.

2.5. Typische Anwendungen

Das Java TV API betrachtet Fernsehprogramme als **Services**. Durch diese Abstraktion kann sich eine Anwendung auf eine Vielzahl an unterschiedlichen Datensignalen beziehen, ohne dass sie für jeden Signaltyp umgeschrieben werden müsste. So kann ein Dienst genauso gut ein altes analoges Fernsehsignal oder ein digitales Signal begleitet von einer Java Anwendung darstellen. Das Java TV API stellt die Mittel zur Verfügung um solche Dienste auszuwählen, um auf eine Datenbank mit Informationen zu den jeweils verfügbaren Diensten zuzugreifen, um Datenströme aufzubereiten und abzuspielen und um andere Daten, wie zum Beispiel Java Anwendungen abzuspeichern und auszuführen.

Entwickler können mit Hilfe des hier vorgestellten APIs viele verschiedene Anwendungen für ihre Fernsehgeräte schreiben. Nachfolgend sind ein paar typische Arten mit dazugehörigen Beispielen aufgeführt:

- **Elektronische Programmführer**

Nicht wegzudenken von den Fernsehgeräten sind die elektronischen Programmführer. Sie zeigen eine Übersicht über die momentan aktuellen Programme und lassen den Zuschauer auf einfache Weise den Kanal wechseln. Wichtig sind eine kurze Aufstartzeit und gute Interaktivität.



- **Anwendung zum aktuellen Fernsehprogramm**

Eine Anwendung dieses Typs läuft nur im Zusammenspiel mit einer speziellen Sendung. Zum Beispiel kann der Fernsehzuschauer während einer Quiz-Sendung aktiv am Spiel teilnehmen oder während einem Sportanlass sein Geld auf den Sieger verwetten.

TV Sport-Wettbüro		Optionen Abmelden Hilfe	
Statistik	Euroliga, 12. Spieltag, Dienstag 02.05.2007 43:23		
<u>Ballbesitz</u>	FC Basel - Lausanne Sport 1 : 1 (- : -)		
Torschüsse			
3 (-) 4 (-)			
Eckbälle			
0 (-) 2 (-)			
G./R. Karten			
G:0 R:0			
G:1 R:0			
Bisherige Partien			
Aktuelle Teaminfos			
Ihre Wette			
Res. 3:1			
Eins. 150\$			
Quote 1:178			
<input type="button" value="Ändern"/>	 Klick hier! alle Suchmaschinen		

- **Stand-alone Anwendungen**

Anwendungen, die sich nicht aufs aktuelle Fernsehprogramm beziehen und so jederzeit vom Benutzer gestartet und beendet werden können, nennt man Stand-alone Anwendungen. Sie sind in der Regel sehr nah mit den uns bestens bekannten Internet Anwendungen verwandt. Als Beispiel kann ein Echtzeit-Börsenkurssystem angeführt werden.

- **Werbungen**

Diese Anwendungen sollen den Inhalt von Werbungen anreichern und verschönern. Sie haben normalerweise eine sehr kurze Lebensdauer. Leider ist sie nur allzu oft immer noch zu lang...

3. Technische Informationen

3.1. Packages im Java TV API

Das Java TV API besteht aus Klassen und Interfaces, die in folgende Packages aufgeteilt sind:

- **javax.tv.carousel** bietet Zugang zu Daten in Form von Verzeichnissen und Dateien, die mit dem Fernsehsignal mitgeschickt werden. Siehe dazu auch Punkt 3.4.
 - **javax.tv.graphics** ermöglicht einfaches Erstellen von Bildern (z.B. durch Übereinanderlegen von zwei Bildern) und bietet einen Container um aus AWT-Komponenten eine graphische Benutzerschnittstelle zu bilden.
 - **javax.tv.locator** bietet Zugang zu allen Daten, auf die von einem Fernsehempfänger aus zugegriffen werden kann.
 - **javax.tv.media** definiert Erweiterungen zum Java Media Framework (JMF), um eingehende Audio- und Video-Datenströme zu verarbeiten.
 - **javax.tv.net** bietet Zugang zu Daten, die über das Internet Protokoll (IP) gesendet werden.
 - **javax.tv.service** bietet Zugang zu der Datenbank, in der Informationen über die zur Verfügung stehenden Dienste abgelegt sind. Dieses Package umfasst folgende Subpackages:
 - **javax.tv.service.guide**
 - **javax.tv.service.navigation**
 - **javax.tv.service.selection**
 - **javax.tv.service.transport**
- Meh zum Thema Service und Service Information gibt es im nächsten Kapitel (3.2.) zu erfahren.
- **javax.tv.util** unterstützt das Erzeugen und Verwalten von Timer-Ereignissen.
 - **javax.tv.xlet** bietet Methoden, die die verschiedenen Lebenszyklen und die Eigenschaften von Java TV Anwendungen kontrollieren

3.2. Services und Service Information (SI)

Service im Java TV API ist ein Container, der Service Unterkomponenten aufnimmt und verwaltet. Solche Unterkomponenten sind zum Beispiel Audio- oder Video-Ströme. Damit die Synchronisierung dieser Komponenten einfacher fällt, werden sie als eine Einheit angesehen. Ein Service braucht aber nicht unbedingt Ton- oder Sprach-Daten. Genausogut kann es sich um einen Service zur Abwicklung von Bankgeschäften handeln. Der Benutzer interagiert dabei mit einer Java-Anwendung, Ton und Sprach braucht es dazu nicht.

Jeder Service besitzt einen dazugehörigen Service Information Eintrag. Dieser ist in einer Datenbank, der sogenannten **SI Database**, abgelegt. Darin werden die speziellen Eigenschaften eines Service offengelegt. Der Empfänger, und damit also auch die Java Anwendung, muss Zugriff auf diese Einträge haben, damit der Service korrekt für die Anwendung verwendet werden kann.

Eine **Service Component** ist ein einzelnes Element eines Service. Wie gesagt, kann es einen Ton- oder Bilddatenstrom darstellen. Auch denkbar sind Java TV Anwendungen oder andere Daten von nicht weiter definierten Typen, die von einer laufenden Anwendung eingelesen werden. Ein Service wird immer ein oder mehrere Service Components beinhalten; auf der anderen Seite kann eine solche Komponente von mehreren Services geteilt werden.

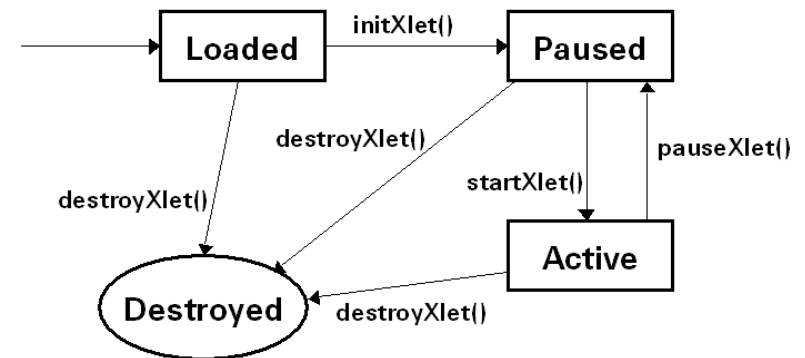
Ein **Service Locator** ist etwas wie eine URL. Er ist die benötigte Information, um einen gewünschten Kanal einzustellen. Diese Information besteht typischerweise aus einer Adresse, die den Sender darstellt, sowie aus zusätzlichen Informationen, die benötigt werden, um das Signal richtig zu demultiplexen.

3.3. Xlets

Ein anderer Name für Java TV Anwendungen ist Xlets. Wie Applets werden Xlets von der Software kontrolliert, die sie aufgerufen haben. Bei Applets ist das meist ein Browser oder der Appletviewer von Sun, bei Xlets ist es der digitale Fernsehempfänger, der die Java Plattform unterstützt.

Xlets haben keine **main**-Methode und implementieren immer das Interface **Xlet**. Wie Applets haben Xlets einen Lebenszyklus, der durch verschiedene im Interface Xlet Methoden von einem Zustand in den nächsten gebracht werden kann. Folgende Zustandsübergänge sind vorhanden:

- Create
- Initialize
- Start
- Pause
- Destroy



XletStateMachineDiagram

Jede Java TV Implementation hat einen **Application Manager**, der diese Methoden aufrufen kann und so die installierten Xlets durch ihre verschiedenen Zustände führt. Er merkt selbstständig, wann er ein Xlet pausieren muss, damit ein anderes mit höherer Priorität genügend Ressourcen frei hat.

3.4. Broadcast Data API

Weil man nicht bei jedem Fernsehempfänger davon ausgehen kann, dass tatsächlich ein Kanal zum Senden von Daten vorhanden ist, hat man beim Java TV API die Klasse `javax.tv.carousel` eingeführt. Damit ist es möglich, aus Java Code auf Dateien zuzugreifen, die mit dem digitalen Fernsehsignal mitgeschickt werden. Schlussendlich kann der Programmierer auf die Java Standardfunktionen zum Lesen (**FileInputStream**) von Dateien zurückgreifen.

Damit dies funktioniert, schickt der Service-Provider in einem bestimmten Sekundentakt immer wieder die gewünschte Datei mit (daher auch der Name). Die Java-Anwendung auf der Kundenseite hat soviel „Geduld“, dass sie beim Zugriff auf eben diese Datei solange wartet, bis sie tatsächlich wieder im Datenstrom auftaucht.

Als Codefragment sieht dieser Vorgang folgendermassen aus:

1. Erzeuge ein `CarouselFile` des Hauptverzeichnisses eines bestimmten Dienstproviders (durch Adresse im locator bestimmt):

```
CarouselFile serviceGateway = new CarouselFile(locator);
```
2. Lese dieses Verzeichnis in einen Array ein:

```
String files[] = serviceGateway.list();
```
3. Erzeuge ein File Objekt:

```
CarouselFile myFile=new CarouselFile(serviceGateway,  
                                     files[0]);
```
4. Erzeuge einen `InputStream` aus dem File Objekt:

```
FileInputStream fis = new FileInputStream(myFile);
```
5. Lese ein Byte aus dem `CarouselFile` aus:

```
byte data = fis.read;
```
6. Schliesse die Datei:

```
fis.close;
```

4. Schlusswort und Ausblick

Ob sich das Java TV API tatsächlich durchsetzen und als Standard auf zukünftigen Fernsehern etablieren kann, bleibt meiner Meinung nach zu bezweifeln. Zuallererst muss der heutige Fernsehschauer sein Verhalten dahingehend ändern, dass er tatsächlich komplexere Anwendungen auf seinem Fernseher laufen lassen will. Heutzutage dürfte es kein grosses Verlangen sein, Excel Tabellen oder Programmiercode auf dem Fernseher anzuzeigen.

Ein weiteres Kriterium wird die Unterstützung durch die Hardware-Hersteller sein. Obwohl doch einige Sun beim Entwicklungsprozess unterstützt haben, dürfte es wenn es dann ums tatsächliche Realisieren geht anders aussehen. Jeder Hersteller möchte ja die Kunden damit anlocken, dass nur gerade auf seinem Gerät die besten Funktionen zur Verfügung stehen. Ausserdem steht mit Microsoft und dessen WebTV, das zusammen mit Windows CE ebenfalls für die gleiche Produktparte vorgesehen ist, ein grosser Konkurrenz vor der Sonne.

Zuletzt nicht unerwähnt bleiben dürfen die Nachteile der Java Technologie. Vorweg die mangelnde Geschwindigkeit. Damit eine Java TV Anwendung eine vergleichbare Geschwindigkeit zu bisherigen proprietären Systemen erreicht, sind schnellere und modernere Hardwarekomponenten erforderlich. Diese drücken den schlussendlichen Marktpreis in die Höhe und halten somit viele potentielle Kunden von einem Kauf ab.

Aber wir werden ja sehen...