Bluetooth-Anwendungen selbst gemacht

Die letzten beiden Ausgaben beschreiben einige Grundlagen über den Kurzstreckenfunk Bluetooth und ein Modul, welches eigene Anwendungen um eine drahtlose serielle Schnittstelle mit dem Funkprotokoll Bluetooth erweitert. Die spannende Frage, wie Bluetooth selbst programmiert wird, klärt dieser Artikel.

Drahtlose Alleskönner

Die Java-Plattform Micro Edition (Java ME) und der Kurzstreckenfunk Bluetooth sind zwei herausragende Technologien in der drahtlosen Welt. Das „write once run anywhere" (zu Deutsch: einmal schreiben, überall ausführen) -Konzept der Java Micro Edition verschafft dem Programmierer eine sehr hohe Produktivität. Bluetooth lässt sowohl mobile Endgeräte wie Mobilfunktelefone als auch Computer über einheitliche Protokolle weltweit miteinander kommunizieren. Was liegt näher, als diese erfolgreichen Konzepte zu vereinen und auf bequeme Art über ein Mobilfunktelefon ein Garagentor, den PC oder die Außenbeleuchtung fernzubedienen. Auch der Sicherheitsaspekt spielt bei diesem Unterfangen eine nicht unerhebliche Rolle, welcher bei einer Bluetooth-Verbindung schon eingebaut ist. Die mittlerweile sehr große Vielfalt mobiler Endgeräte würde den Entwickler vor die unlösbare Aufgabe stellen, eine speziell angepasste Version für jedes einzelne Gerät zu erstellen – doch hier setzt das Konzept von Java ME an und stellt eine einheitliche Umgebung mit standardisiertem Zugriff auf die Hardware zur Verfügung. Um dieses Vorhaben in die Tat umsetzen zu können, erklärt Ihnen dieser Artikel den Kommunikationsablauf einer Bluetooth-Verbindung anhand von Ausschnitten aus Quelltexten. Dazu stellen wir eine passende Entwicklungsumgebung vor, die vollkommen kostenfrei aus dem Internet zu beziehen ist.

Entwicklungsumgebung

Bluetooth wird standardmäßig nicht direkt von Java ME oder Java SE unterstützt. Einige Firmen erkannten bereits früh das Potenzial mobiler Anwendungen mit Bluetooth und formierten schon im Jahre 2000 im so genannten JCP (Java Community Process) die JSR 82 (Java Specification Request): „Java APIs for Bluetooth". Viele Hersteller von Mobilfunktelefonen haben heutzutage diese Spezifikation bereits umgesetzt, genaue Informationen sind beim jeweiligen Hersteller abzurufen.

Als Entwicklungsumgebung wird hier NetBeans 5.0 der Firma Sun verwendet, die hier genannten Beschreibungen beziehen sich auf diese Version, wobei viele andere Entwicklungsumgebungen wie z. B. Eclipse oder der JBuilder von Borland ebenso verwendet werden können. NetBeans 5.0 wird zusammen mit dem JDK 5.0 auf der Internetseite http://java.sun.com zum Download angeboten. Dort ist ebenfalls das NetBeans Mobility Pack 5.0 erhältlich, welches noch nachzuinstallieren ist, damit NetBeans auch Anwendungen für Java ME erstellt. Eine Installation des Sun Java Wireless Toolkit ist ebenso empfehlenswert, da dieses einige Beispiele enthält, welche für

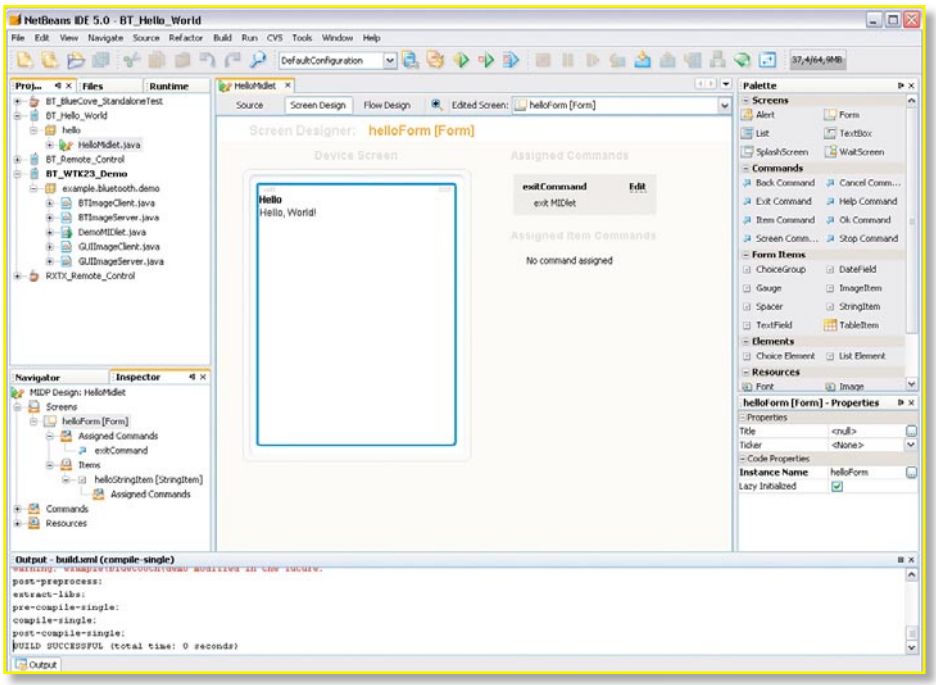

Bild 1: Hello-World-MIDlet in der Entwicklungsumgebung NetBeans 5.0

Java ME geschrieben sind. Abbildung 1 zeigt ein Bildschirmfoto der Entwicklungsumgebung mit einer Hello-World-Anwendung für ein so genanntes MIDlet. MIDlets sind Anwendungsprogramme für mobile Geräte, die in der Programmiersprache Java geschrieben sind.

Kommunikationswege

Die Aufgabe, einen PC von einem Mobiltelefon aus fernzusteuern, hört sich zunächst einfach an. Auf dem Mobiltelefon läuft ein MIDlet, welches Eingaben von der Tastatur entgegennimmt und diese über eine Bluetooth-Verbindung an den PC sendet. Der PC ist dazu mit einem Bluetooth-USB-Adapter ausgestattet und empfängt die Tastendrücke – allerdings wird die PC-seitige Anwendung in Java SE geschrieben, welche ohne eine Bluetooth-Unterstützung ausgeliefert wird. Hier gibt es zwei Auswege, zum einen kann man vom Mobiltelefon aus eine serielle Datenverbindung über Bluetooth anlegen. Der Treiber des Bluetooth-USB-Adapters stellt auf dem PC eine virtuelle serielle Schnittstelle bereit. Mit dem Open-Source-Projekt RxTx (Internetseite: http://www.rxtx.org) steht eine plattformübergreifende Erweiterung von Java SE bereit, welche den Zugriff auf serielle Schnittstellen erlaubt. Falls man sich auf das Betriebssystem Windows XP mit Service-Pack 2 beschränkt, gibt es auch eine Erweiterung namens BlueCove, welche eine JSR-82-Umsetzung auf Java SE vornimmt. Das Projekt wurde ursprünglich von Intel begonnen, ist mittlerweile Open Source und auf der Projektseite http://bluecove.sourceforge.net verfügbar. Anleitungen zum Einbinden der Erweiterungen in die Java SE finden sich auf den jeweiligen Projektseiten.

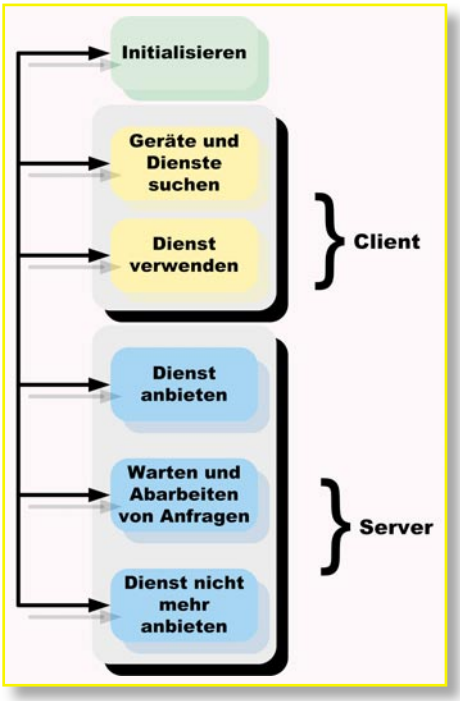

Bild 2: Eigenschaften der Bluetooth-Geräte

Ablauf eines Verbindungsaufbaus

Eine Bluetooth-Anwendung verhält sich entweder wie ein Client oder wie ein Server,

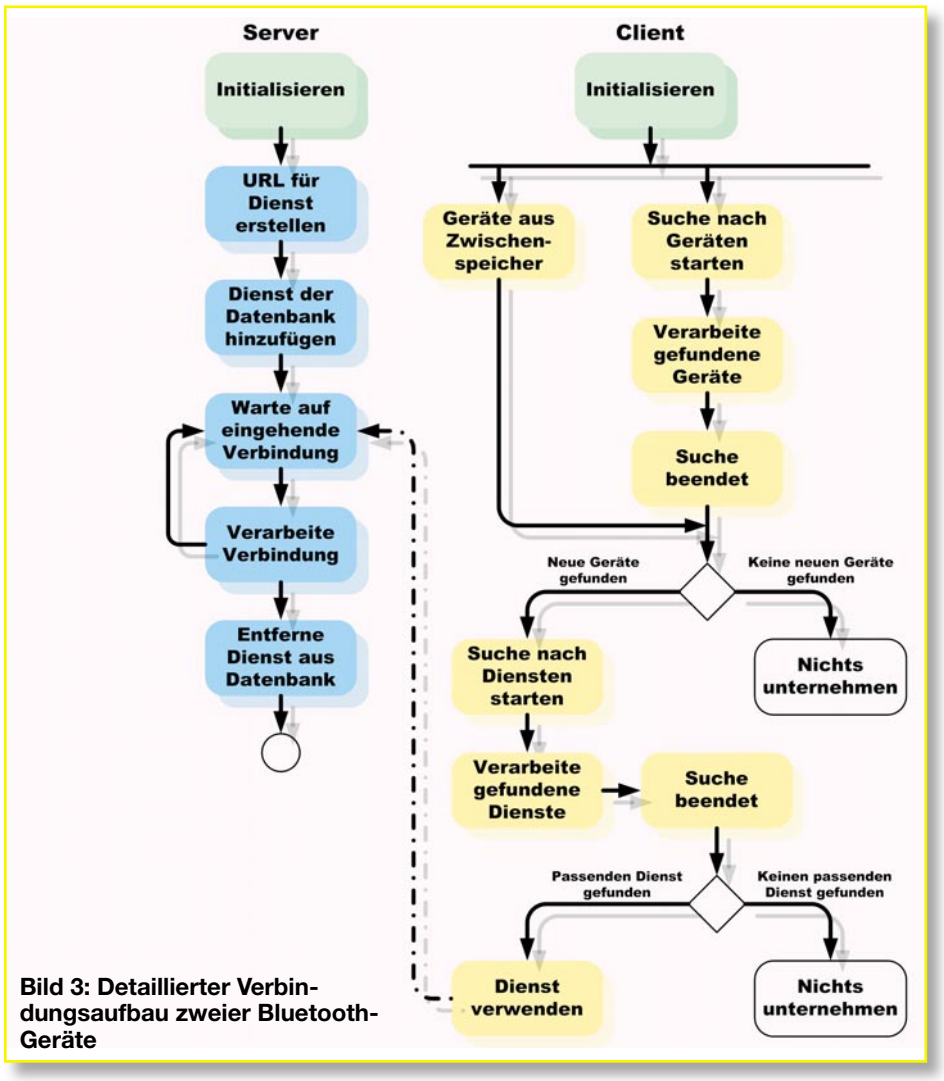

Bild 3: Detaillierter Verbindungsaufbau zweier Bluetooth-Geräte

```
(...)
LocalDevice localDevice;       // lokaler Bluetooth Manager
DiscoveryAgent discoveryAgent; // Suchagent
(...)

// Bluetooth initialisieren
public void blueInit() throws BluetoothStateException {
   // Referenz auf lokalen Bluetooth Manager holen
   localDevice = LocalDevice.getLocalDevice();
   // Server werden hier fuer die Umgebung sichtbar gemacht
   // Hinweis: nicht alle Implementationen liefern TRUE
   // zurueck, daher wird der Rueckgabewert nicht ausgewertet
   localDevice.setDiscoverable(DiscoveryAgent.GIAC);
   // Clients verschaffen sich eine Referenz auf den Suchautomaten
   discoveryAgent = localDevice.getDiscoveryAgent();
}
```

Bild 4: Die Initialisierung des Bluetooth-Stacks.

bietet also einen Dienst an oder nimmt einen Dienst wahr. In der Abbildung 2 sind die verschiedenen Rollen, die ein Gerät wahrnimmt, in einer Übersicht dargestellt.

Initialisierung: Jede Bluetooth-Anwendung, ob Client oder Server, muss als ersten Schritt den Bluetooth-Stack initialisieren.

Client: Ein Client nimmt einen Dienst eines Servers wahr. Dazu sucht er zunächst nach nahen Bluetooth-Geräten (Device Discovery). Anschließend ruft er die verfügbaren Dienste der einzelnen Geräte ab. Falls einer der Dienste gewünscht ist, kann eine Verbindung aufgebaut werden.

Server: Der Server stellt dem Client Dienste zur Verfügung. Die Dienste werden in der Datenbank SDDB (Service Discovery Database) lokal auf dem Gerät abgelegt. Der Server ist sichtbar für andere Bluetooth-Geräte und wartet auf Verbindungsanfragen. Werden angebotene Dienste nicht mehr gebraucht, entfernt sie der Server aus der Datenbank.

Ein Ablaufschema, welches mehr Detailinformationen enthält, zeigt die Abbildung 3. Beide Geräte bereiten zunächst den Bluetooth-Stack vor. Die Anwendung auf dem Server stellt einen Dienst bereit und wartet auf eingehende Verbindungsanfragen. Der Client sucht nach nahen Bluetooth-Geräten und ermittelt die angebotenen Dienste. Wenn ein passender Dienst gefunden wird, verbindet sich der Client mit dem Server.

Initialisierung

Jede Bluetooth-Anwendung verschafft sich zu Beginn eine Referenz auf den Bluetooth-Manager über die Klasse „Local Device“. Während eine Server-Anwendung sich nur „discoverable“ (auffindbar) markiert, besorgt sich der Client eine Referenz auf den „DiscoveryAgent“. Dieser stellt die Dienste zur Verfügung, um andere Geräte zu suchen und dessen Dienste zu erkunden. Der Quelltext in Abbildung 4 illustriert dieses Beispiel.

Verbindungen erstellen

Damit eine Server-Anwendung einen Dienst anbieten kann, lauscht sie auf spezielle Verbindungsanfragen. Serielle Datenverbindungen werden über das SPP (Serial Port Profile) abgewickelt. Die allgemeine Form einer URL (Verbindungen werden als URL realisiert) lautet:

`btspp://hostname:[CN|UUID];parameters`

Die Parameter bedeuten im Einzelnen:

- **btspp** ist das URL-Schema für eine serielle Datenverbindung.

```
// Name des Dienstes
static final String serviceName = „serialVCP“;
// UUID
static final String serviceUUID = „3f564545634eb5433a5b48ff04234bc2“;
UUID SERVICEUUID_UUID = new UUID(serviceUUID, false);
// URL vorbereiten
String btURL =  „btspp://localhost:“+SERVICEUUID_UUID.toString()+“;“+“name=“+serviceName;
// Erstellen der Server Verbindung
StreamConnectionNotifier streamConnN = (StreamConnectionNotifier) Connector.open(btURL);
// Warten und Akzeptieren von Client Verbindungen
StreamConnection streamConn = streamConnN.acceptAndOpen();
```

Bild 5: Erstellen einer Server-Verbindung

```
// gewuenschter Dienst ist gefunden
ServiceRecord serviceRecord
= (ServiceRecord)discoveredServices.elementAt(index);
// URL der Server Verbindung holen
String serverConnURL =
serviceRecord.getConnectionURL(ServiceRecord.AUTHENTICATE_ENCRYPT, false);
// Verbindung oeffnen
StreamConnection streamConnection = (StreamConnection)
Connector.open(serverConnURL);
```

Bild 6: Ein Client verbindet sich mit einem Server.

- **hostname** ist entweder „localhost“ für eine Server-Verbindung oder eine Bluetooth-Adresse für eine Client-Verbindung.
- **CN** ist die Kanalnummer und wird vom Client beim Verbindungsaufbau verwendet.
- **UUID** ist eine universelle eindeutige Kennung, deren Eindeutigkeit im Rahmen der Wahrscheinlichkeit garantiert wird.
- **parameters** beschreiben den Dienst, enthalten Sicherheitseinstellungen.

Ein konkretes Beispiel für eine solche URL wäre damit:

```
btspp://localhost:3f564545
634eb5433a5b48ff04234bc2;
authenticate=true;encrypt=
true;name=serialVCP
```

Falls eine Verbindung von einem Client zu einem Server aufgebaut wird, verwendet man anstelle von „localhost“ eine Verbindung, die im „ServiceRecord“ enthalten ist. Im Quelltext in Abbildung 5 wird das Erstellen der Verbindung gezeigt.

Wenn ein Client eine passende Gegenstelle gefunden hat, verbindet er sich mit dem Server wie in Abbildung 6 gezeigt.

Geräte und Dienste finden

Ein eigener Bereich der Bluetooth API beschäftigt sich ausschließlich mit dem Auffinden anderer Bluetooth-Geräte und dem Ermitteln der angebotenen Dienste. Die Klasse „DiscoveryAgent“ enthält hierfür drei Methoden:

- retrieveDevices() – ruft bereits bekannte Geräte ab.
- startInquiry() – beginnt eine aktive Suche nach Bluetooth-Geräten in Reichweite.
- cancelInquiry() – beendet den Suchlauf.

Beim Auffinden von Geräten werden automatisch folgende Methoden im Interface „DiscoveryListener“ aufgerufen:

- deviceDiscovered() – wird aufgerufen, sobald ein neues Gerät gefunden wird.
- inquiryCompleted() – wird aufgerufen, sobald die Suche beendet ist.

Analog zum Auffinden von Geräten funktionieren die Methoden der Klasse „DiscoveryAgent“ zum Suchen nach bereitgestellten Diensten.

- searchServices() – startet die Suche nach Diensten.
- cancelServiceSearch() – beendet den Suchlauf.

Auch hier gibt es ein Interface, dessen Methoden jeweils beim Auffinden eines Dienstes gestartet werden. Folgende Methoden des Interfaces „DiscoveryListener“ sind dabei zu implementieren:

- serviceDiscovered() – wird aufgerufen, wenn ein Dienst gefunden wird.
- serviceSearchCompleted() – wird aufgerufen, sobald die Suche beendet ist.

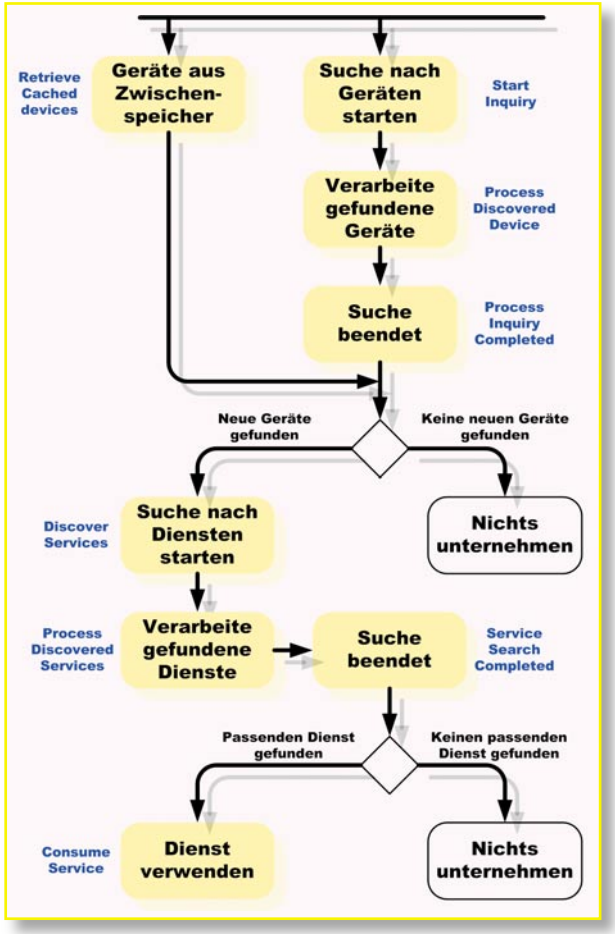

Bild 7: Auffinden von Geräten und Diensten

Bild 8: Simulieren von Mobiltelefonen im Sun Java Wireless Toolkit

Komfortabel testen

Im Sun Java Wireless Toolkit sind Emulatoren für Mobiltelefone enthalten, damit die Anwendungen sehr komfortabel getestet werden können, ohne sie jedes Mal umständlich auf ein reales Mobiltelefon zu kopieren. Der Emulator kann sogar über Bluetooth (simuliert über TCP) zwischen mehreren Instanzen Daten austauschen.

Das Bildschirmfoto in der Abbildung 8 zeigt ein mit dem Wireless Toolkit ausgeliefertes Beispiel. Das Mobiltelefon links dient als Server und stellt Bilder zum Herunterladen zur Verfügung, während die beiden Mobiltelefone auf der rechten Seite als Client konfiguriert sind und jeweils ein Bild des Servers geladen haben und anzeigen.

Ausblick

Ein hilfreicher Tipp für eigene Entwicklungen ist die Klasse Robot im Paket „java.awt.Robot“. Diese Klasse kann Tastendrücke simulieren und damit verschiedene Aktionen ausführen. Der Beispielcode in Abbildung 9 sperrt unter Windows XP den Bildschirm.

Obwohl der Quellcode zur Kommunikation mit Bluetooth sehr kompakt und übersichtlich ausfällt, ist zum Verständnis Hintergrundwissen zu Bluetooth erforderlich. Mit den vorgestellten Konzepten sollten Sie jetzt in der Lage sein, eine Anwendung zu schreiben, welche Ihren PC vom Mobiltelefon aus fernsteuert. ELV

```
import java.awt.Robot;
(...)
private Robot robot;
robot = new Robot();
robot.keyPress(KeyEvent.VK_WINDOWS);
robot.keyPress(KeyEvent.VK_L);
robot.keyRelease(KeyEvent.VK_L);
robot.keyRelease(KeyEvent.VK_WINDOWS);
```

Bild 9: Bildschirm unter Windows XP unter Java sperren