JavaアプリケーションからUSBデバイスにアクセスする

USB、jUSB、JSR-80の入門

Qingye Jiang
Research Scientist
HappyFox Engineering Solutions

2003年 9月 02日

Javaプラットフォームは元来、そのプラットフォーム非依存性を誇りにしています。その非依存性は多くの利点を持つ一方で、ハードウェアとやりとりする非常に手の込んだJavaアプリケーションを記述するという工程をもたらします。この記事では、研究者Qingye Jiangが2つのプロジェクトについて考察します。それらのプロジェクトは、JavaアプリケーションからUSBデバイスを利用可能にするAPIを用いることで、処理をより簡単にします。両プロジェクトはまだ初期形態ではありますが、将来性を発揮し、既に実在のアプリケーションの基礎として役立っています。

1996年1月に、Universal Serial Bus (USB)規格の最初のバージョンがリリースされました。コストの低さ、高いデータ転送速度、使いやすさ、および柔軟性から、USBはコンピューター業界で広く受け入れられています。今日、多くの周辺機器やデバイスはUSBインターフェースを介してコンピューターに接続します。現在では、ほとんどの汎用のオペレーティングシステムは、USBデバイスをサポートしており、またそのような周辺機器にアクセスするアプリケーションをCやC++で開発することは比較的容易です。しかし、Javaプログラミング言語はハードウェアアクセスをほとんどサポートしていません。したがって、USBデバイスとやりとりするJavaアプリケーションの記述は、非常に難しいということが分かります。

Java言語でUSBデバイスにアクセスするという試みは、1999年にIBMのDan Streetmanによって開始されました。彼のプロジェクトは2001年に、Java Specification Request(JSR)のプロセスを経てJava言語の拡張標準の候補として承認されました。現在プロジェクトはJSR-80と呼ばれ、Javaパッケージ`javax.usb`に公式に割り当てられています。その一方で2000年6月に、SourceForge のMojo JojoとDavid BrownellはjUSBプロジェクトを開始しました。どちらも完成してはいませんが、両プロジェクトはそれ以来Linux開発者のための有用なパッケージを作成しています。まだどちらからも有用なパッケージは現れていませんが、両プロジェクトは他のオペレーティングシステム上のJavaアプリケーションからUSBデバイスにアクセスする試みも始めました。(この記事で示されたプロジェクトや他のプロジェクトに関しては、参考文献を参照して下さい。)

この記事では、jUSBとJSR-80プロジェクトを簡単に紹介します。ただし、両プロジェクトがどのようにUSBデバイスとやりとりするのかを理解するために、最初にUSBプロトコルの基本を学びま

す。また、USBデバイスにアクセスするために、どのように両プロジェクトのAPIを使用するのかを示すため、部分的なコードを提示します。

USB入門

1994年、4社 (Compaq、Intel、Microsoft、NEC)が協力し、USBプロトコルを規格化しました。プロトコルの当初の目的は、電話とPCの接続と、拡張や再設定が容易なI/Oインターフェースを用意することでした。1996年1月に、USB規格の最初のバージョンがリリースされ、改訂版(バージョン1.1)が1998年9月にリリースされました。仕様では、12Mbps以下の総通信帯域で、同時に127個のデバイスが接続可能としました。後に、もう3社(Hewlett Packard、Lucent、Philips)が連合に参加し、2000年4月に、480Mbpsまでの転送速度をサポートする、USB規格のバージョン2.0がリリースされました。今日、USBはハイスピード (ビデオ、映像周辺機器、記憶装置)、フルスピード (オーディオ、ブロードバンド、マイクロホン)のデータ転送アプリケーションにおいて重要な役割を果たしています。また、PCに様々なロースピードのデバイス(キーボード、マウス、ゲーム周辺機器、バーチャルリアリティ周辺機器)も接続可能です。

USBプロトコルは完全な階層型です。どんなUSBシステムでも、ホストは1つだけであり、ホストコンピューターへのUSBインターフェースは、ホストコントローラーと呼ばれます。ホストコントローラーには2種類の規格があります。-- Open Host Controller Interface (OHCI、Compaqが策定)とUniversal Host Controller Interface (UHCI、インテルが策定)。両規格は同じ機能を提供し、すべてのUSBデバイスで動作します。UHCIのハードウェアインプリメンテーションのほうが単純ですが、より複雑なデバイスドライバが必要となります(したがって、CPUにより負荷がかかります)。

USBの物理的な接続形態は、7階層までの階層的なスター型トポロジーです。ハブは各星型の中心に位置し、USBホストはルートハブと見なされます。各配線セグメントは、ハブとUSBデバイスの2地点間接続です。後者は、システムに追加接続するための別のハブか、機能性のある何らかのデバイスのどちらかです。ホストは主従関係のプロトコルを使用して、USBデバイスと通信を行います。この方法はパケット衝突の問題を解決しますが、接続されたデバイス同士の直通通信も防ぎます。

すべてのデータ転送はホストコントローラーによって開始されます。ホストからデバイスへのデータは、ダウンストリームやアウトトランス（OUT転送）と呼ばれ、デバイスからホストへのデータは、アップストリームやイントランス（IN転送）と呼ばれます。データ転送は、ホストとUSBデバイス上の特定のエンドポイント間で行われ、ホストとエンドポイント間のデータリンクはパイプと呼ばれます。接続されたUSBデバイスは多くのエンドポイントを持っている可能性があり、ホストとデバイス間のデータパイプ数は、デバイス上のエンドポイント数と同じです。パイプは単方向あるいは双方向であり、1つのパイプのデータフローは他のパイプのデータフローからは独立しています。

USBネットワーク上の通信には、4種類の異なるデータ転送形式のうちのどれか1つが使用可能です。

- コントロール転送: アタッチ時（USBケーブル接続時）の、デバイス管理と設定用の短いデータパケットです。
- バルク転送: 比較的大量中のデータパケットです。スキャナやSCSIアダプタのようなデバイスはこの転送形式を使用します。

- **インタラプト転送**: 一定周期で転送されるデータパケットです。ホストコントローラーは、自動的に指定された間隔で割込みを送信します。
- **アイソクロナス転送**: 信頼性より、帯域幅の高い必要条件を備えたリアルタイムのデータストリームです。オーディオとビデオデバイスは一般にこの転送形式を使用します。

シリアルポートのように、コンピューター上の各USBポートはUSBコントローラーによってユニークな識別番号(ポートID)が割り当てられます。USBデバイスがUSBポートにアタッチされるとき、このユニークなポートIDがデバイスに割り当てられ、USBコントローラーによってデバイスディスクリプタが読込まれます。デバイスディスクリプタは、デバイスのコンフィギュレーションの情報と同時にデバイス全般の情報も含んでいます。コンフィギュレーションでは、USBデバイスの機能性およびI/Oの振る舞いを定義します。USBデバイスは1つ以上のコンフィギュレーションを持つ可能性があり、それらは対応するコンフィギュレーションディスクリプタによって記述されます。各コンフィギュレーションは1つ以上のインターフェースを持ち、それらは物理的な通信チャネルと見なすことができます。各インターフェースは0かそれ以上のエンドポイントを持ち、それらはデータ送信側かデータ受信側のどちらかか、あるいは両方です。インターフェースはインターフェースディスクリプタによって記述され、またエンドポイントはエンドポイントディスクリプタによって記述されます。更にUSBデバイスは、ベンダー名、デバイス名やシリアル番号のような補足情報を提供するためにストリングディスクリプタを持つ可能性があります。

ご存知のとおりUSBのようなプロトコルは、Java言語を使用する開発者に、プラットフォームおよびハードウェア非依存性ゆえに大変な努力を強いります。それでは、その問題の解決を試みる2つのプロジェクトを見てみましょう。

jUSB API

jUSBプロジェクトは、2000年6月にMojo JojoとDavidドBrownellによって作成されました。その目的はLinuxプラットフォーム上のUSBデバイスにアクセスするために、1セットのフリーのJava APIを提供することでした。このAPIは、Lesser GPL(LGPL)の下で配布されており、フリーソフトウェアプロジェクトと同様に独自のプロジェクトで使用可能です。APIは、多数の物理的なUSBデバイスへのマルチスレッドのアクセスを可能にし、ローカルとリモートのデバイスをサポートします。多数のインターフェースをもつデバイスは、異なるインターフェースを使用するアプリケーション(またはデバイスドライバ)を使用して、同時に多数のアプリケーション(またはデバイス・ドライバ)によってアクセス可能です。APIはコントロール転送、バルク転送、およびインタラプト転送をサポートします。アイソクロナス転送はサポートされません (オーディオやビデオのようなメディアデータ用に使用され、既に他の標準化されたデバイスドライバ上のJMF API(参考文献を参照)によって十分にサポートされているため) 。現在、APIはLinux 2.4カーネルか2.2.18カーネルのバックポートをもつGNU/Linuxディストリビューションで動作します。したがって、最も新しいディストリビューションはサポートされています。例えば、Red Hat 7.2および9.0上ではパッチや他のアップグレードなしで動作します。

jUSB APIは次のパッケージを含んでいます。

- `usb.core`: このパッケージは、jUSB APIのコア部分です。Javaアプリケーションが USBホストからUSBデバイスにアクセスすることを可能にします。
- `usb.linux`: このパッケージは、`usb.core.Host`オブジェクトのLinuxインプリメンテーション、ブートストラップのサポート、Linux USBのサポートを利用するほかのクラスを含んでい

ます。このインプリメンテーションは、仮想のUSBデバイスファイルシステム(usbdevfs)を介してUSBデバイスにアクセスします。
- usb.windows: このパッケージは、usb.core.HostオブジェクトのWindowsインプリメンテーション、ブートストラップのサポート、Windows USBのサポートを利用するほかのクラスを含んでいます。このインプリメンテーションは、まだ非常に初期の段階にあります。
- usb.remote: このパッケージは、usb.core APIのリモートバージョンです。RMIプロキシおよびデーモンアプリケーションを含み、Javaアプリケーションがリモートコンピューター上のUSBデバイスにアクセスすることを可能にします。
- usb.util: このパッケージは、いくつかの便利なユーティリティを提供します。それらはUSBデバイスのためのファームウェアをダウンロードし、XMLへUSBシステムの内容をダンプし、バルクI/Oのみを使用してUSBデバイスをソケットに変換します。
- usb.devices: このオプションのパッケージは、KodakデジタルカメラやRio 500 MP3プレーヤーを含む、jUSB APIを使用した様々なUSBデバイスにアクセスするためのJavaコードを集めたものです。これらのAPIは指定のUSBデバイスにアクセスする処理を簡易化するために特別に記述されており、他のデバイスにアクセスするために使用することはできません。APIはusb.core APIに基づいており、jUSBがサポートされているすべてのオペレーティングシステムで動作します。
- usb.view: このオプションのパッケージは、Swingに基づいた簡単なツリー式のUSBブラウザーを提供します。それはjUSB APIの使い方を例証する非常によいサンプルプログラムです。

usb.core.Hostオブジェクトのインプリメンテーションはオペレーティングシステムによって異なりますが、Javaプログラマはj USB APIを使用したアプリケーションの開発を始めるために、usb.coreパッケージだけを理解すればいいのです。表1は、Javaプログラマが理解しておくべきusb.coreのインターフェースとクラスの概要です。

表1. jUSBのインターフェースとクラス

利点	説明
Bus	Hostに1組のUSBデバイスを接続します。
Host	1つ以上のBusを持つUSBコントローラーを表します。

クラス	説明
Configuration	デバイスのUSBコンフィギュレーションや、そのコンフィギュレーションに関連するインターフェースへのアクセスを実現します。
Descriptor	USB形式のディスクリプタを使用する実体の基底クラス。
Device	USBデバイスへのアクセスを実現します。
DeviceDescriptor	USBデバイスディスクリプタへのアクセスを実現します。
EndPoint	特定のデバイスコンフィギュレーションのデバイスデータ入出力を構造化し、USBエンドポイントディスクリプタへのアクセスを実現します。
HostFactory	ブートストラップメソッドを含んでいます。
Hub	USBハブディスクリプタと、いくつかのハブオペレーションへのアクセスを実現します。
Interface	エンドポイントのセットを記述し、特定のデバイスコンフィギュレーションに関連しています。

PortIdentifier	USBデバイス用の不変の文字列識別子を作成します(オペレーションやトラブルシューティング用に使用します)。

jUSB APIを使用してUSBデバイスにアクセスするための通常の手順は、以下のとおりです。

1. HostFactoryからUSBホストを取得して、ブートストラップします。
2. ホストからUSBバスにアクセスし、次に、バスからUSBルートハブ(USBデバイスのこと)にアクセスします。
3. ハブで利用可能なUSBポート数を取得し、すべてのポートをトラバースし適切なデバイスを検出します。
4. 特定のポートにアタッチされたUSBデバイスにアクセスします。デバイスは、そのPortIdentifierを持つホストから直接アクセスされるか、ルートハブから順にUSBバスをトラバースし検出されます。
5. ControlMessageを使用してデバイスと直接通信するか、あるいはデバイスの現在のコンフィギュレーションからインターフェースを求め、インターフェース上で利用可能なエンドポイントを使用してI/Oを実行します。

リスト1は、jUSB APIを使用してUSBシステムの内容を取得する方法を示します。このプログラムは、利用可能なUSBデバイスのためにルートハブを調べるという簡単なものですが、容易にUSBツリー全体を調べるように改良することができます。以下のロジックは、1～4のステップに相当します。

リスト1. jUSB APIを使用してUSBシステムの内容を取得する

```
import usb.core.*;

public class ListUSB
{
    public static void main(String[] args)
    {
        try
        {
            // Bootstrap by getting the USB Host from the HostFactory.
            Host   host = HostFactory.getHost();

            // Obtain a list of the USB buses available on the Host.
            Bus[]  bus  = host.getBusses();
            int    total_bus = bus.length;

            // Traverse through all the USB buses.
            for (int i=0; i<total_bus; i++)
            {
                // Access the root hub on the USB bus and obtain the
                // number of USB ports available on the root hub.
                Device root = bus[i].getRootHub();
                int total_port = root.getNumPorts();

                // Traverse through all the USB ports available on the
                // root hub. It should be mentioned that the numbering
                // starts from 1, not 0.
                for (int j=1; j<=total_port; j++)
                {
                    // Obtain the Device connected to the port.
                    Device device = root.getChild(j);
                    if (device != null)
                    {
                        // USB device available, do something here.
```

```
                    }
                }
            }
        } catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
```

リスト2は、アプリケーションがデバイスを検出したと仮定して、`Interface`と`EndPoint`を使用してバルクI/Oを実行する方法を示します。この部分的なコードも、コントロールや割り込みI/Oを実行するように修正することができます。以下は、5のステップに相当します。

リスト2. jUSB APIを使用したバルクI/Oの実行

```
  if (device != null)
  {
      // Obtain the current Configuration of the device and the number of
      // Interfaces available under the current Configuration.
      Configuration config = device.getConfiguration();
      int total_interface = config.getNumInterfaces();

      // Traverse through the Interfaces
      for (int k=0; k<total_interface; k++)
      {
          // Access the currently Interface and obtain the number of
          // endpoints available on the Interface.
          Interface itf = config.getInterface(k, 0);
          int total_ep  = itf.getNumEndpoints();

          // Traverse through all the endpoints.
          for (int l=0; l<total_ep; l++)
          {
              // Access the endpoint, and obtain its I/O type.
              Endpoint ep = itf.getEndpoint(l);
              String io_type = ep.getType();
              boolean input  = ep.isInput();

              // If the endpoint is an input endpoint, obtain its
              // InputStream and read in data.
              if (input)
              {
                  InputStream in;
                  in = ep.getInputStream();
                  // Read in data here
                  in.close();
              }
              // If the Endpoint is and output Endpoint, obtain its
              // OutputStream and write out data.
              else
              {
                  OutputStream out;
                  out = ep.getOutputStream();
                  // Write out data here.
                  out.close();
              }
          }
      }
  }
```

jUSBプロジェクトは、2000年6月から2001年2月まで非常に活動的でした。APIの最も新しいリリース(バージョン0.4.4)は、2001年2月14日に利用可能になりました。それ以来、いくつかの小さい進展しか報告されていません。おそらくIBMグループはJava言語の拡張標準候補となり成功したため

でしょう。しかしながら、JPhotoプロジェクト(デジタルカメラとの接続にjUSBを使用するアプリケーション)やjSyncManagerプロジェクト(Palm OSベースのPDAの同期化にjUSBを使用するアプリケーション)を含むいくつかのサードパーティーアプリケーションは、jUSBに基づいて開発されています。

JSR-80 API(javax.usb)

前記のように、JSR-80プロジェクトは1999年にIBMのDan Streetmanによって作成されました。2001年に、プロジェクトはJava Specification Request (JSR)のプロセスを経てJava言語の拡張標準の候補として承認されました。現在プロジェクトはJSR-80と呼ばれ、Javaパッケージ`javax.usb`に公式に割り当てられています。プロジェクトはCommon Public Licenseの下で認可され、Java Community Processを使用して開発されています。このプロジェクトの目的は、どんなJavaアプリケーションやミドルウェアコンポーネントからでもUSBシステムに十分にアクセスできる、Javaプラットフォーム用のUSBインターフェースを開発することです。JSR-80 APIは、USB規格によって定義された4種類の転送形式すべてを全面的にサポートしています。現在、このAPIのLinuxインプリメンテーションは、Red Hat 7.2や9.0のような2.4カーネルをサポートする、最も新しいGNU/Linuxディストリビューションにて動作します。

JSR-80プロジェクトは3つのパッケージを含んでいます：`javax-usb`(`javax.usb`API)、`javax-usb-ri`(OS-independent Reference Implementationの共通部分)、`javax-usb-ri-linux`(Linux USBスタックに共通のリファレンスインプリメンテーションを接続する、Linuxプラットフォーム用のリファレンスインプリメンテーション)。3つはすべて、Linuxプラットフォーム上で完全に機能する`java.usb`APIを形成するために必要です。他のオペレーティングシステム(主にMicrosoft Windows)へのAPIの移植を目指す自主的な試みがプロジェクトのメールリスト上で報告されましたが、まだ機能するパッケージはリリースされていません。

JSR-80 APIのOS依存のインプリメンテーションはオペレーティングシステムによって異なりますが、Javaプログラマはアプリケーションの開発を始めるために、`javax.usb`パッケージだけを理解すればいいのです。表2は、Javaプログラマが理解しておくべき`javax.usb`のインターフェースとクラスの一覧です。

表2. JSR-80 APIのインターフェースとクラス

インターフェース	説明
`UsbConfiguration`	USBデバイスのコンフィギュレーションを表わします。
`UsbConfigurationDescriptor`	USBコンフィギュレーションディスクリプタのためのインターフェース
`UsbDevice`	USBデバイスのためのインターフェース
`UsbDeviceDescriptor`	USBデバイスディスクリプタのためのインターフェース
`UsbEndpoint`	USBエンドポイントのためのインターフェース
`UsbEndpointDescriptor`	USBエンドポイントディスクリプタのためのインターフェース
`UsbHub`	USBハブのためのインターフェース
`UsbInterface`	USBインターフェースのためのインターフェース
`UsbInterfaceDescriptor`	USBインターフェースディスクリプタのためのインターフェース
`UsbPipe`	USBパイプのためのインターフェース

UsbPort	USBポートのためのインターフェース
UsbServices	javax.usbインプリメンテーションのためのインターフェース

表2. JSR-80 APIのインターフェースとクラス

クラス	説明
UsbHostManager	javax.usbのためのエントリポイント

JSR-80 APIを使用してUSBデバイスにアクセスするための通常の手順は、以下のとおりです。

1. UsbHostManagerから適切なUsbServicesを取得して、ブートストラップします。
2. UsbServicesによってルートハブにアクセスします。ルートハブはアプリケーション中でのUsbHubと見なされます。
3. ルートハブに接続されているUsbDeviceのリストを取得します。適切なUsbDeviceを検出するために、全ての下位層のハブを調べます。
4. コントロールメッセージ(UsbControlIrp)を使用してUsbDeviceと直接通信するか、あるいはUsbDeviceの適切なUsbConfigurationからUsbInterfaceを求め、UsbInterfaceで利用可能なUsbEndpointを使用してI/Oを実行します。
5. UsbEndpointがI/Oを実行するために使用される場合は、それに関連したUsbPipeを開きます。アップストリームデータ(USBデバイスからホストコンピューターへ)およびダウンストリームデータ(ホストコンピューターからUSBデバイスへ)は、UsbPipeによって同期あるいは非同期でサブミットされます。
6. UsbPipeを閉じ、アプリケーションがもはやUsbDeviceへのアクセスを必要としない場合は適切なUsbInterfaceを解放します。

リスト3では、JSR-80 APIを使用してUSBシステムの内容を取得します。プログラムは、再帰的にUSBシステム上のすべてのUSBハブを通してトラバースし、ホストコンピューターに接続されたすべてのUSBデバイスを検出します。以下のコードは、1～3のステップに相当します。

リスト3. JSR-80 APIを使用してUSBシステムの内容を取得する

```
import javax.usb.*;
import java.util.List;

public class TraverseUSB
{
        public static void main(String argv[])
        {
          try
          {
              // Access the system USB services, and access to the root
              // hub. Then traverse through the root hub.
              UsbServices services = UsbHostManager.getUsbServices();
              UsbHub rootHub = services.getRootUsbHub();
              traverse(rootHub);
          } catch (Exception e) {}
        }

        public static void traverse(UsbDevice device)
        {
          if (device.isUsbHub())
          {
             // This is a USB Hub, traverse through the hub.
             List attachedDevices =
```

```
                ((UsbHub) device).getAttachedUsbDevices();
            for (int i=0; i<attachedDevices.size(); i++)
            {
              traverse((UsbDevice) attachedDevices.get(i));
            }
         }
         else
         {
            // This is a USB function, not a hub.
            // Do something.
         }
       }
}
```

リスト4は、アプリケーションがデバイスを検出したと仮定して、`Interface`と`EndPoint`を使用してI/Oを実行する方法を示します。この部分的なコードも、4種類のデータ転送形式すべてのI/Oを実行するように修正することができます。以下は、4～6のステップに相当します。

リスト4. JSR-80 APIを使用したI/Oの実行

```
public static void testIO(UsbDevice device)
{
    try
    {
       // Access to the active configuration of the USB device, obtain
       // all the interfaces available in that configuration.
       UsbConfiguration config = device.getActiveUsbConfiguration();
       List totalInterfaces = config.getUsbInterfaces();

       // Traverse through all the interfaces, and access the endpoints
       // available to that interface for I/O.
       for (int i=0; i<totalInterfaces.size(); i++)
       {
          UsbInterface interf = (UsbInterface) totalInterfaces.get(i);
          interf.claim();
          List totalEndpoints = interf.getUsbEndpoints();
          for (int j=0; j<totalEndpoints.size(); j++)
          {
             // Access the particular endpoint, determine the direction
             // of its data flow, and type of data transfer, and open the
             // data pipe for I/O.
             UsbEndpoint ep = (UsbEndpoint) totalEndpoints.get(i);
             int direction = ep.getDirection();
             int type = ep.getType();
             UsbPipe pipe = ep.getUsbPipe();
             pipe.open();
             // Perform I/O through the USB pipe here.
             pipe.close();
          }
          interf.release();
       }
    } catch (Exception e) {}
}
```

JSR-80プロジェクトは当初から非常に活動的でした。バージョン0.10.0の`javax.usb` API、RIおよびLinux 用RIは、2003年2月にリリースされました。このバージョンは、最終承認のためにJSR-80委員会に提出されるようです。JSR-80が形式的にJava言語の拡張標準になった後に、他のオペレーティングシステム用のインプリメンテーションがすぐに利用可能になることが期待されます。Linuxディベロッパーコミュニティーは、jUSBプロジェクトよりJSR-80プロジェクトにより関心を示すように思われますが、Linuxプラットフォーム上で`javax.usb` APIを使用するプロジェクトはますます増えています。

結論

jUSB APIとJSR-80 APIは、Linuxオペレーティングシステムを実行するマシンからUSBデバイスにアクセスする機能を持つJavaアプリケーションを実現します。JSR-80 APIはjUSB APIより高い機能性をもち、Java言語の拡張標準になる可能性があります。現在は、Linux開発者しかjUSBとJSR-80 API の利益を受けることができませんが、jUSBとJSR-80 API APIを他のオペレーティングシステムに移植する積極的な取組みが報告されています。Java開発者は、近い将来には他のオペレーティングシステム上のUSBデバイスにアクセス可能になるはずです。今これらのAPIに習熟しておけば、これらのプロジェクトが多数のプラットフォーム上で使用可能になったときには、アプリケーションにUSBの機能を追加する準備は万端です。

著者について

Qingye Jiang

Qingye JiangはHappyFox Engineering Solutionsの研究者であり、1999年に清華大学を修了し、2000年にイリノイ大学アーバナ・シャンペーン校から工学士号を受けました。彼の研究範囲はJ2ME、分散コンピューティング、モバイルコンピューティング、組み込みシステム、ワイヤレスコミュニケーションに渡ります。