BlackBerry Architecture

Anthony Scian
Principal Developer
Research in Motion

Background

- CS/EEE BMath 1987, MMath 1990
- Co-op
  - FORTRAN, C compilers
  - Pascal, BASIC interpreters
- After graduation
  - Watcom, Powersoft, Sybase
    - C++ compiler (ISO C++ committee)
    - SQL compiler
  - Research in Motion
    - Java optimization and standards
    - Java virtual machine
    - Cryptography and Security

Overview

- BlackBerry History
- Java on the BlackBerry
- API Evolution
- Quality Assurance
- Reality Check
- Review

BlackBerry History

- RIM is an engineering company started in early 1980's
- Awarded a technical Oscar in 1998 for a digital file bar code reader
- Early innovator in pager wireless networks
- 1999: first BlackBerry device, the 950 for Mobitex networks
- 2001: first Java-based BlackBerry

Evolution of BlackBerry architecture

- 850/950 Hardware
  - Intel-based
  - 512k RAM
  - 4MB NOR flash
- Very limited hardware running on one(!) AA battery
- Programmed in C++
- Applications very difficult to code due to limited runtime environment

Evolution of BlackBerry architecture

- 5810: first Java-based device
  - ARM-based
  - 1MB RAM
  - 8MB NOR flash
  - GSM phone
- Programmed in Java
  - Java is a standardized language
  - Sun wanted to maintain “write once; run anywhere”
  - Established JCP: Java Community Process

Why convert BB to Java?

- BlackBerry was a low power device, as such, it shared one CPU to manage the radio and the applications
- C++ is a difficult and error-prone language for writing apps
- Security was important; must protect radio code from app errors and hackers
- Mobile CPUs didn’t have MMUs to isolate code

Benefits of Java on a mobile device

- JVM provides isolation of application code from radio code on a single CPU device
- Errors in apps are less spectacular; controlled exceptions rather than crashing device
- Lower barrier to entry for programmers; Java is easier to learn than C++
- Security is provided by the JVM; Java cannot access raw memory; all accesses are checked
- Java code can be smaller than C++ code with dense bytecode representation
- Java bytecode can be verified (all opcodes can be checked to ensure operands are the correct type); important for security

How was the transition handled?

- Legacy C++ devices were supported for many years
  - Improvements made to the OS
  - Phased out as the Mobitex network gave way to GPRS/CDMA cellular networks
- Large team was built to develop brand new Java apps and environment
  - Were able to leverage network knowledge developed for C++ device; same protocols were used
  - Productivity higher in Java; most BB functions (email, phone) could be written with adequate performance in Java

Java bootstrap

- JVM developed
- Java debugger and tools developed
- Early JVM could run on the C++ device allowing Java development to proceed with available device hardware
- Based Java environment on emerging J2ME (Java 2 Micro Edition) standard

Evolution of BlackBerry Java architecture

- CLDC: Connected Limited Device Configuration
  - Define base Java language runtime support
  - Innovation: new fast bytecode verifier
  - GCF: Generalized Connection Framework
  - No UI
  - CLDC 1.0: basic Java (no floating point)
  - CLDC 1.1: add floating point support

Evolution of BlackBerry Java architecture

- MIDP: Mobile Information Device Profile
  - UI library appropriate for small screen devices
  - Standardized application format (MIDlet) and download
  - MIDP 1.0: monochrome UI, tone generator audio
  - MIDP 2.0: colour UI, richer audio, more fonts, richer UI, signed MIDlets
  - MIDP 3.0: richer security framework, richer UI, audio, video

Java Community Process

- RIM has been a founding member of the JCP Micro Edition Executive Committee
- JSR (Java Specification Request)
  - Used to evolve the Java language and API
  - Community of Java experts, domain experts
  - Unique in that standards come with TCK (Technology Compatibility Kit)
  - Licensing of intellectual property rights

Java Specification Requests

- JSR 30: CLDC 1.0
- JSR 37: MIDP 1.0
- JSR 82: Bluetooth
- JSR 118: MIDP 2.0
- JSR 120: Wireless Messaging
- JSR 139: CLDC 1.1
- JSR 177: Security and Trust Services
- Etc.

RIMlets

- CLDC/MIDP was very limited
- JSRs provided good standardized APIs
  - Some of these APIs were still very limited in scope
- BlackBerry network was an asset
  - Connectivity to corporate networks through BES
- Richer framework for UI, containers, crypto
- JDE (Java Development Environment)
  - Code editing
  - Desktop simulator
  - On-device debugging
  - Performance analysis
  - Code signing

Characteristics of a good API

- Easy to learn
- Easy to use
- Hard to misuse
- Readable code
- Easy to extend (API)
- Easy to extend (user)
- Appropriate to user's domain knowledge

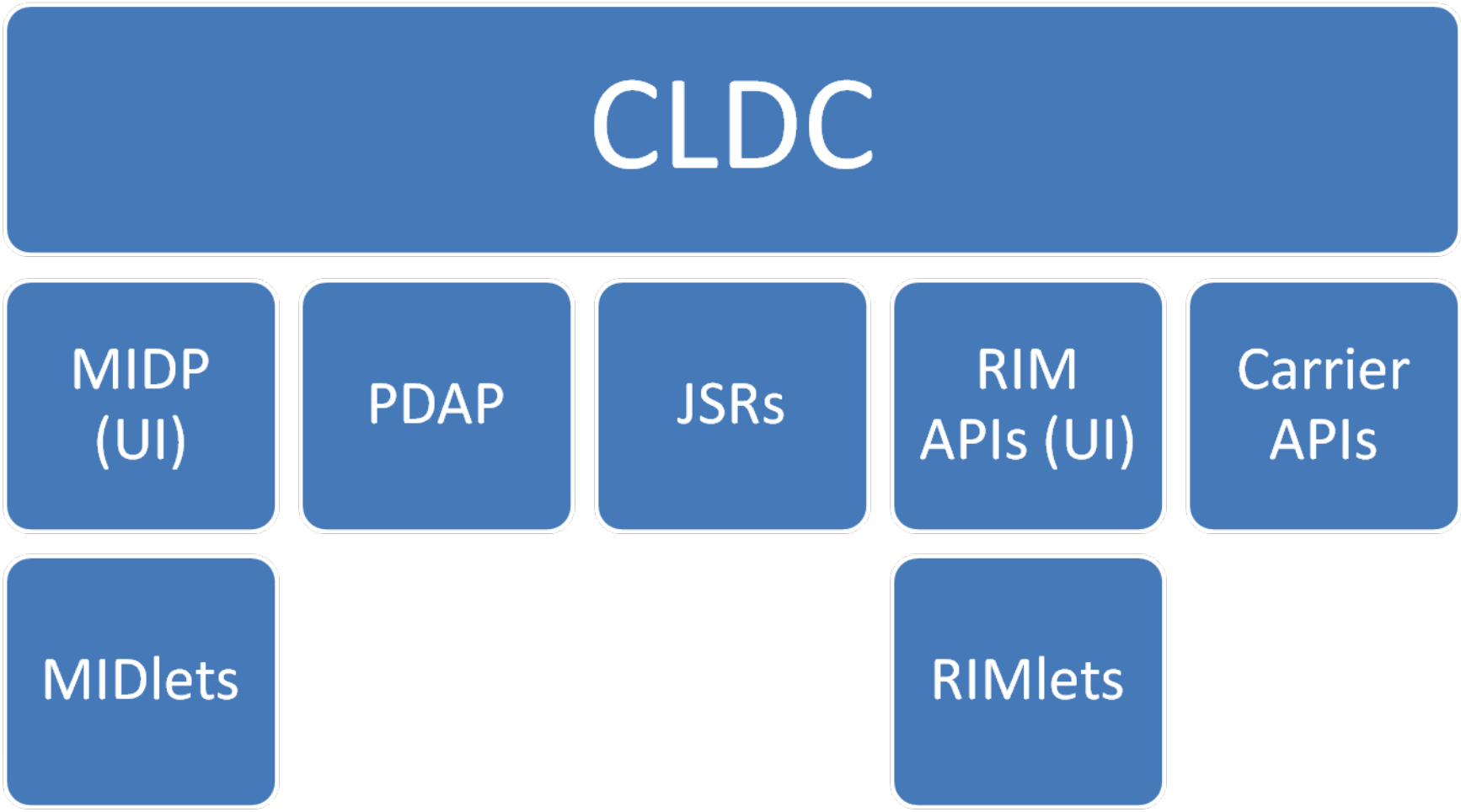
API Evolution

- Public APIs are forever
  - Can be an asset if they are done well
  - Can be a liability if they are done poorly
- Very difficult
  - Marketplace changes rapidly
  - New security concerns
  - Old code needs to continue to work
  - Mistakes happen

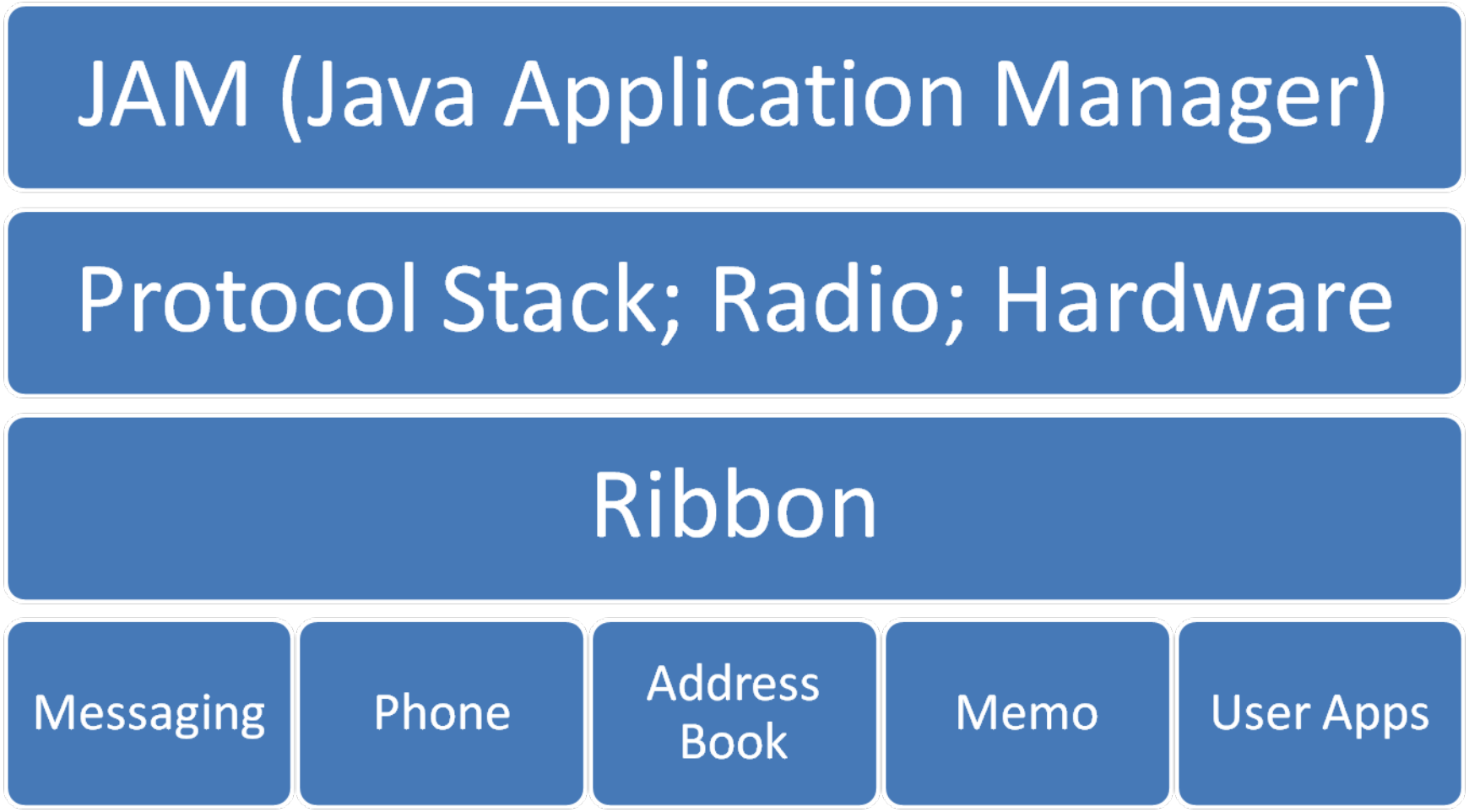
Evolution of BlackBerry APIs

- Designed to highlight cornerstones of BB
  - Security
  - Network
  - Battery Life
- Provide developers with access to hardware
  - Backlight
  - Accelerometer
  - Touch pad/screen
  - GPS
  - WiFi
  - Audio

BB API Architecture



BB Process Architecture



Evolution by Carriers

- Carriers can provide access to their networks and websites to applications
- Location-based services (cell towers)
- Payment APIs
- Push APIs
- Product Differentiation
  - Additional APIs can be provided
- Carrier acceptance can be a lengthy process

Quality Assurance

- API Reviews
- Code Reviews
- Beta Testing
  - Early access to new APIs for partners
  - Feedback can change APIs
- Test Streams
- Test Streams
- Test Streams

Quality is Job 1.1

- What do you do if an API has a flaw?
  - Failure to satisfy contract
    - Easy, fix the problem?
    - Depends, what if people worked around the problem already?
  - Can you change the API?
    - How old is the API?
    - How many users?
    - Impact on users?
  - Can you add to the API?
    - Does the API allow growth in features?

Quality by Process

- As companies get larger, the impact of API flaws grows
  - Lots of customer impact
  - Lots of devices
    - Lots of different software versions
    - Upgrade infrastructure must be built
- How do you manage API development?
  - Formal design documents
  - Formal reviews
  - Build on existing API designs
    - Use “big” Java APIs for inspiration

Quality Branches

- Device software evolves as hardware evolves
- More RAM, more Flash, new devices
- A common problem in any long term software development effort
- Standard Branch Model
  - Sub-branches for specific carrier releases
  - Carriers like to have control over their code when acceptance is close

Quality is Social

- Customer meetings
  - What APIs do they need?
  - What problems do they have with existing APIs?
- Promotion of internal APIs
  - APIs benefit from real-world use
  - Hard to get right the first time
- Last ingredient
  - Hire quality people

Reality Check

- Mistakes happen
  - Incompatibility can be necessary
  - “Break with the past” Principle: more code will be written in the future
- Use Java to deprecate APIs
  - Moves developers away from poor APIs

Review

- Good API design is difficult
- Prepare to fail
  - Design for extension and repair of APIs
- Quality is like real estate: Test, test, test
  - Writing tests results in experience using the API
  - Tests may influence API design early on
  - Tests help prevent regressions
- Questions?