Embedded Middleware

National Chiao Tung University
Chun-Jen Tsai
3/21/2011
Embedded Middleware

- “Middleware” is a very popular terminology in embedded system software design
  - Just like “cloud computing,” it is merely a new name for old technologies

- We can further divide embedded middleware into two layers
  - System middleware – an API library that provides system abstraction to application software to increase application portability
  - Hardware abstraction middleware – an API specification that acts as a hardware abstraction layer for chip-level integration
Why System Middleware

- The purpose of system middleware is to provide a homogeneous environment for applications.
  - Operating system does the same thing, but due to the variety of embedded hardware and competitions among OS companies, we need something beyond OS.
System Middleware Components

- A system middleware must provide the following abstractions for (multimedia) applications
  - Application model
  - GUI subsystem
    - Graphics, video, audio rendering
    - User event handling
  - Timing control
  - File system

e.g. DVB-MHP application
DVB-MHP Architecture Revisited

Multimedia Home Platform functional blocks:

- Native Application
- Data
- Xlets
- Xlets based on GEM

- Video Codec (H-264, MPEG-2)
- Audio Codec (HE-AAC)
- Graphic library (e.g., Microwin)
- Transport Demux (MPEG-2 TS, RTSP/RTP)
- Sun Java-TV (Application Model)
- DVB (Event, Services Abstraction)
- HAVI (UI Abstraction)
- DAVIC (FS Abstraction)
- JMF 1.1
- Application Manager (Navigator)

- Operating Systems
- Java Standard Classes (CDC/PBP)

- I/O Devices
- RISC Processor (< 300MHz)
- Graphics accelerator
- Audio accelerator
- Video accelerator
- Java Processor

SoC
Java-based Middleware

- DVB-MHP 1.0 (aka DVB-J) system abstraction is based on Java RE, just like many other middleware
  - In particular, CDC/PBP is adopted for MHP
- Google Android is also based on Java Model
  - Google defined their own “profile” and VM (not JVM byte-code compatible)
- Rationale: Java is “virtually a computer plus an OS!”

![Diagram showing the relationship between Real Computer Model and Java Model](image-url)
Native vs. Java-based Middleware

- We can use native code to design system middleware. For example, BREW from Qualcomm is a native middleware for mobile handsets:
  - A C/C++ development environment for ARM-based platforms
  - Application: BREW “Applet” model; cooperative multitasking (IThread API)
  - Graphics API: OpenGL/ES
  - Multimedia API: IMedia

- Supposedly, BREW is faster than Java, but with a properly designed system, this may not be true.
MHP Application Model Abstraction

- Application implements the javax.tv.xlet.Xlet interface
  - Similar to java.applet.Applet, but simpler
  - Methods are called in the following sequence:
    - initXlet()
    - startXlet(): resource allocation begins here!
    - destroyXlet()

- Xlet life cycle (initiated by the MHP application manager, which sets up an XletContext for the Xlet)
Middleware Architecture for DVB-J

- MHP middleware layers†:

  - Application Manager and Xlet API
  - Service Selection
  - Tuning
  - DSM-CC
  - SI
  - Section Filtering
  - MPEG
  - JMF
  - Conditional Access
  - Return Channel
  - DVB UI
  - HAVi
  - AWT
  - Inter-Xlet Comm.
  - UI Events
  - Java

†S. Morris and A. Smith-Chaigneau, Interactive TV Standards, Elsevier, 2005
DVB-J APIs

- Java Language API
- Application Model (Life Cycle) API
  - Based on Java TV xlets
- Graphical API
  - Based on AWT and HAVi UI level 2
- Data Access API
- Multimedia API
  - Based on JMF
- Service and User Settings API
MHP Application UI Construction

- An MHP application can allocate screen resource, called HScene, using HAVi API
  - An HScene represents a region on the TV display

- UI components are mainly drawn using
  - java.awt.Graphics primitives
  - HAVi widget classes
The MHP display model is defined in Sun Java-TV middleware:

- **Background Data**
  - Background Decoder
  - HBackgroundDevice

- **Video Data**
  - Video Decoder
  - Decoder Format Conversion
  - scaling
  - JMF Manipulation
  - HVideoDevice
  - HScreen
  - TV Behavior Control
  - Application Behavior Control
  - Format signaling to display
  - Broadcast metadata
  - User preference display info.

- **Graphics Data**
  - Graphics Composition
  - HGraphicsDevice
  - Graphics primitives created using java.awt and HAVi level 2 UI

Implementation?
MHP Display Model Implementation

- If MHP adopts Sun’s PBP reference implementation:

```
org.havi.ui.HScene
java.awt.Container
java.awt.Component

add()
```

```
widgets
```

```
Java PBP Class Libraries
  Portable Graphics Library (MicroWindows)
  Native Graphics Library (X11)
  OS (Linux)
  fbdev driver

HW video frame buffer
  VGA/DVI Controller

HScreen
```

```
HVideoDevice
HGraphicsDevice
HBackgroundDevice

HScreenDevice
```

```
Should be merged for efficiency!
```
Building a GUI Application

- **During initXlet()**
  - Configure the required HScene and retrieve the handle to the HScene object

- **During startXlet()**
  - Create widget components under the HScene
  - Display the HScene using setVisible(true)
  - Request the focus

- **During destroyXlet**
  - Hide the HScene
  - Release the allocated resources (components, images, etc.)
Controlling Media from Xlets

- Xlets can control audio/video media using Java Media Framework
  - Based on JMF 1.0
  - Use org.davic.net.dvb.DVBLocator to select what to show
- MHP provides extra JMF controls for
  - Video scaling
  - Subtitle control and service components monitoring
  - Audio playback fine control (for local playback)
  - Notification of changes in incoming TV signals
JMF Class Architecture

- JMF is used as media playback interface in MHP
- Only JMF 1.1 API are used:
  - Player
  - DataSource
  - Control

![Diagram showing JMF class architecture with relationships between classes like Clock, TimeBase, Duration, Controller, MediaHandler, Player, and DataSource.]
Video/Graphics Integration† for MHP

†iTV Handbook, Prentice Hall 2003
DVB File System Model†

DVB File System Abstraction

- DVB applications (Xlets, etc.) and data can be transmitted to STBs using DSM-CC Object Carousel†

Extraction of Xlets from DSM-CC

- One possible implementation of Xlets extractor:

```
Object Cache

Version Manager

Section Filter

DSM-CC Parser

DII Cache

DII Cache

Monitor Threads

Download Threads

Real-Time Thread Pool

Xlet Cache

Xlet Running Space

DSM-CC Cache Space

Application Manager

Incoming data stream

User Input
```
Synchronization

- Synchronization of application with media are facilitated by the following API functions:
  - DSM-CC stream events
    - org.dvb.dsmcc.DSMCCStreamEvent.subscribe()
    - org.davic.media.StreamEventControl.subscribeStreamEvent()
  - Media Time (NPT)
    - org.davic.media.MediaTimeEventControl
  - Private section events
    - org.davic.mpeg.sections
Java for 2.5/3G Mobile Services

- 3GPP adopts the concept of middleware for CPE
  - For application middleware, Java RE has been selected (CLDC/MIDP 2.0)
  - For multimedia device integration middleware, OpenMAX has been proposed (by ARM, Motorola, Nokia, TI, etc)
- Google also selects Java R.E. for G-phone, but use their own virtual machine and profile libraries
  - This avoids Sun’s J2ME licensing fee
Google Android Architecture

Applications
- Home
- Contacts
- Phone
- Browser
- ...

Application Framework
- Activity Manager
- Window Manager
- Content Providers
- View System
- Notification Manager
- Package Manager
- Telephony Manager
- Resource Manager
- Location Manager
- XMPP Service

System Libraries
- Surface Manager
- Media Framework
- SQLite
- OpenGL/ES
- FreeType
- WebKit
- SGL
- SSL
- C Library

Android Runtime
- Core Libraries
- Dalvik Virtual Machine

Linux Kernel
Linux’s Role in Android

- Java model does not have an I/O subsystem, Linux kernel is only (hopefully) used to provide
  - Hardware abstraction layer (for different drivers)
  - Multi-threading management
  - Shared library interface
- The following “infamous” Linux components are not included
  - Native windowing system (e.g. X11)
  - glibc
Android Applications

- An Android application package (APK) is a collection of components which usually run in one thread
  - Services or ContentProviders sometimes need another thread (process)
Android Application Model

- An android application has four building blocks
  - Activity → an operation that optionally associated with a UI component (window), similar to execution context in MHP
  - Service → background task
  - Content Provider → file system abstraction
  - Intent Receiver → event handler
Activities Life Cycle

- Application activities (e.g. UI objects) have following entry points
  - Life cycle begins with `onCreate()`, ends with `onDestroy()`
  - Visible life cycle begins with `onStart()`, ends with `onStop()`
  - Foreground (in-focus) life cycle begins with `onResume()`, ends with `onPause()`
Booting an Android Platform

- Init process starts the zygote process:
  - Zygote spawns and initializes Dalvik VM instances based on requests; it also loads classes

![Diagram of Android Managed Services](image-url)
Invoking Media Accelerators

- Invocation of audio device from a Java application

- Application
  - Application Framework (Java)
  - MediaPlayer
    - JNI
  - C library
    - MediaPlayer
      - Binder IPC
      - Media Framework
      - Audio Flinger
        - Dynamic loading
        - libaudio.so
        - ALSA
        - Device driver

- Linux Kernel
System Middleware Issues

- Although system middleware can ideally solve the application portability problem, it does not (yet)
  - Mobile phone Java games may not work on all Java handsets
  - MHP xlets may not run on all MHP set-top boxes
- However, as a system designer, the holy goal of middleware should still be honored
HW/SW Integration Issues

- In the past, IC/IP designers tend to define their own HW interface, including IC pinout specs, IC control registers, and subsystem behaviors.
- For complex systems, the partition of a system into sub-behaviors is not standardized.
Example Code to Control ICs/ IPs

When a software function needs to interact with the hardware circuit, how do we invoke† the circuit from a program?

```c
/* HW accelerator interface */
short *input = (short *) 0x0F0000;
short *output = (short *) 0x0F0200;
short *call_idct = (short *) 0x0F0400;

void idct(short *block)
{
    int unit = sizeof(short);
    memcpy(input, block, 64*unit);
    *call_idct = 1;
    memcpy(block, output, 64*unit);
}
```

† Here, memory-mapped I/O is used.
Chip vs. System Designers

- For large IC design houses, they want to sell chips to as many system manufacturers as possible.
- For small IC design houses, they want to compete with large IC design houses without stepping on patents/copyrights.
- For system manufacturers, they want to use the lowest-priced chip (given same specification) available without changing their application software.

![Diagram](image.png)

- Applications
- Middleware
- HW (chips)

An open standard here can enables third party chip market!
OpenMax Middleware

- The OpenMAX† standard was originally conceived as a method of enabling portability of components and media applications throughout the mobile device landscape.
- The proposal of OpenMax was brought to the Khronos Group in mid-2004 by a handful of key mobile hardware companies.
- Khronos Group is a member-funded industry consortium focused on the creation of open standard APIs for multimedia applications.

† Check [http://www.khronos.org/openmax](http://www.khronos.org/openmax) for the spec.
Khronos Multimedia Middlewares

<table>
<thead>
<tr>
<th>Application or middleware libraries (JSR 184 engines, Flash players, media players etc.)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OpenGL ES</strong></td>
</tr>
<tr>
<td>3D Small footprint 3D for embedded systems</td>
</tr>
<tr>
<td><strong>OpenMAX IL</strong></td>
</tr>
<tr>
<td>Component interfaces for codec integration</td>
</tr>
<tr>
<td><strong>OpenMAX DL</strong></td>
</tr>
</tbody>
</table>

Media Engines – CPUs, DSP, Hardware Accelerators etc.

†Jim van Welzan, “OpenMax Overview”, 2007
Why OpenMax?

- **Goal:**
  - Solve the problem of supporting expanding multimedia standards and the increasing complexity of solutions for system and chip manufacturers

- **Solution:**
  - Standardize acceleration API for multimedia kernels
    - Focused on key ‘hotspot’ functions and algorithms
  - Standardize acceleration API for multimedia codecs
    - Abstract system architecture for increased OS portability
  - Provide reference designs across a wide variety of architectures and OS’s

- **Question:**
  - *Who shall write (port) the middleware?*
OpenMax Layers†

- **OpenMAX DL – Development Layer**
  - For codec authors that need to leverage low-level hardware accelerated primitives

- **OpenMAX IL – Integration Layer**
  - For system integrators that need to build solutions from mid-level component pieces (e.g. codecs) potentially from multiple sources

- **OpenMAX AL – Application Layer**
  - For application writers that need high-level access to multimedia functionality (e.g. player, recorder, input, and output object)

†Jim van Welzan, “OpenMax Overview”, 2007
Integration and Development Layers†

†Neil Trevett, “Introduction to OpenMax”, SIGGRAPH presentation, 2004
OpenMax DL Domains

- Audio Coding
  - MP3, AAC
- Image Coding
  - JPEG (encode and decode)
- Image Processing
  - Color space conversion
  - Pixel packing/unpacking
  - De-blocking / de-ringing
  - Rotation, scaling, compositing, etc.
- Signal Processing
  - FIR, IIR, FFT, Dot Product
- Video Coding
  - MPEG-4 SP/H.263 BL (encode and decode)
  - H.264 (encode and decode)
Example: Video DL

- Provide common building block API to handle:
  - MPEG-4 SP/H.263 BL (encode & decode)
  - H.264 (encode/decode)
  - Java Video DL (under discussion)
  - Hardware accelerators (under discussion)

- The building blocks includes:
  - 8x8 SAD and 16X16 SAD
  - 8x8 DCT+Q and 8x8 IDCT+Q
  - MPEG-4 Variable Length Decode
  - Multiple 8x8 SADs in one function
  - Multiple 8x8 DCTs in one function
  - . . .
Example: Image DL

- Provide common building block API to handle
  - JPEG / JPEG 2000 (encode and decode)
  - Color space conversion and packing/unpacking
  - De-blocking / de-ringing
  - Simple rotation and scaling

- The building blocks includes:
  - JPEG 8x8 DCT and 8x8 IDCT
  - JPEG quantization
  - JPEG Huffman encoding and decoding
  - Color conversion
  - Multiple 8x8 DCT and quantization in one function
  - Multiple 8x8 DCTs in one function
  - ...
OpenMax Integration Layer (IL)

- Abstracts Hardware/Software Architecture
  - Provides a uniform interface for framework integration across many architectures
- Integrates with Major Frameworks (OS’s, Systems):
  - Microsoft DirectShow, Symbian MDF, GStreamer, Java MMAPI
OpenMax IL Architecture

Multimedia Framework

IL client

IL core

OMX Component

Port

data tunnel

Platform

IL client

IL core API

commands

callbacks

commands

callbacks
OpenMax IL Concept

- OpenMax IL is based on the visual programming model (similar to DirectShow)
- Example: A DVD player can use the OpenMax IL core API to set up a component network:

  - The system behavior must specify†
    - Data flow
    - Timing (scheduling of processing)

† More on this topic will be discussed in “system formalism” in future classes
OpenMax Core API

- Core API is used to build a “component graph”
  - OMX_init(), OMX_Deinit()
  - OMX_ComponentNameEnum(), OMX_GetComponentsOfRole(), OMX_GetRolesOfComponent()
  - OMX_GetHandle(), OMX_FreeHandle()
  - OMX_SetupTunnel()

- There are many types of components:
  - Source, filter, sink, mixer, scheduler, …, etc.

Component enumeration
Component invocation
Component connection
Relationship Between IL and DL

Application

Multimedia Framework/Middleware

System Media Driver

OpenMax Integration Layer

Component Interface

Component Interface

Component Interface

codec

OpenMax Development Layer

DL Primitives

DL Primitives

Hardware
OpenMAX AL Concept

- OpenMax IL is too complex for many developers
  - AL is designed for simplified streaming media applications
  - AL adopts an object-oriented media approach (similar to the concept in MPEG-21)
- Media Objects enable PLAY and RECORD of media
OpenMax is not the only portable framework for multimedia applications
- MPEG has MPEG-21, MPEG-B/C Reconfigurable Video Coding, and MPEG Multimedia Middleware (M3W)
- Microsoft has Direct X/DirectShow framework
- Many, many other stuff …

If there are too many “standard” frameworks for the developers to support, these “standards” are as bad as no standard at all