Overview of
UHAPI Architecture and Specifications

John Vugts
June 14th, 2005
Yokohama, Japan
Outline

- UHAPI scope
- UHAPI characteristics
- Logical components, roles, interfaces
- Use cases
- UHAPI and variation
- API specification walk-through
- PVR preview
UH Scope: Control AV streams

Applications

Middleware services

Core AV

Customer AV

Native OS Services

MIDDLEWARE API

UNIVERSAL HOME API

OS & CONNECTIVITY API
UHAPI is an API to the Middleware

- Focus on runtime control by the Middleware (ISV).
- It does not specify e.g. streaming interfaces.
- Specification structure deals with diversity.
Example UHAPI 1.0 Logical Components

- Front-end components (11)
  - Channel Decoding
  - Tuning
  - Hdmiln
  - ...

- Decoders/encoders (5)
  - ATSC Decoder
  - Transport Stream Demultiplexing
  - ...

- Video processing components (16)
  - Basic Video Featuring
  - Color Transient Improvement
  - Sharpness Measurement
  - Video Mixing
  - ....

- Audio processing components (10)
  - Audio Bass Enhancements
  - Audio Dynamic Range Control
  - Audio Volume Control
  - ...

- Generic (8)
  - Connection Management
  - Fatal Error Handling
  - Unknown
  - ...

Universal Home Application Programming Interface

© UHAPI Forum
## UHAPI 1.0 contents

<table>
<thead>
<tr>
<th>General documents (7)</th>
<th>Decoders/Encoders (5)</th>
<th>Audio Processing Components (10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Specification Reader's Guide</td>
<td>ATSC Decoder</td>
<td>Audio Automatic Volume Leveling</td>
</tr>
<tr>
<td>API Naming Conventions</td>
<td>Image Decoding</td>
<td>Audio Bass Enhancements</td>
</tr>
<tr>
<td>Error Handling</td>
<td>SPDIF Decoding</td>
<td>Audio Dynamic Range Control</td>
</tr>
<tr>
<td>Execution Architecture</td>
<td>STC Decoding</td>
<td>Audio Mixing</td>
</tr>
<tr>
<td>Notification</td>
<td>Transport Stream Demultiplexing</td>
<td>Audio Noise Generation</td>
</tr>
<tr>
<td>Qualifiers Quick Reference</td>
<td></td>
<td>Audio Program Selection</td>
</tr>
<tr>
<td>API Evolution Rules</td>
<td></td>
<td>Audio Volume Control</td>
</tr>
</tbody>
</table>

### Type specifications (2) :
- Basic Types
- Global Types

### API specifications (50) :
- **Front End Components (11)**
  - Analog Audio & Video Demodulation
  - Analog AV Input
  - Analog Audio Decoding
  - Channel Decoding
  - RF Amplification
  - Out Of Band Tuning & Demodulation
  - Signal Strength
  - Tuning
  - Hdmiln
  - SPDIF-in
  - VBI Slicing

- **Decoders/Encoders (5)**
  - ATSC Decoder
  - Image Decoding
  - SPDIF Decoding
  - STC Decoding
  - Transport Stream Demultiplexing

- **Video Processing Components (16)**
  - Ambient Level
  - Analog Video Decoding
  - Analog Video Encoding
  - Analog Video Encryption
  - Anti Aging
  - Basic Video Featuring
  - Black Bar Detection
  - Color Transient Improvement
  - Dynamic Noise Reduction
  - Histogram Modification
  - Noise Measurement
  - Scan Rate Conversion
  - Sharpness Enhancement
  - Sharpness Measurement
  - Video Color Enhancement
  - Video Mixing

- **Audio Processing Components (10)**
  - Audio Automatic Volume Leveling
  - Audio Bass Enhancements
  - Audio Dynamic Range Control
  - Audio Mixing
  - Audio Noise Generation
  - Audio Program Selection
  - Audio Volume Control
  - Equalizing
  - Speaker Set /Headphones
  - Output Configuration

- **Generic (8)**
  - Analog AV Output
  - SPDIF-out
  - Connection Management
  - Fatal Error Handling
  - I am Alive
  - Pin Objects
  - Unknown
  - URL Source
Outline

- UHAPI scope
- UHAPI characteristics
  - Logical components, roles, interfaces
  - Use cases
  - UHAPI and variation
  - API specification walk-through
  - PVR preview
Universal Home Application Programming Interface

UHAPI Characteristics (1)

- Designed with a middleware view in mind
  - To support a large set of middlewares
- API family for AV functionality (analog, digital)
- Interface-based programming
  - Provides a consistent, orthogonal, coherent set of interfaces
  - Well defined (syntax and semantics)
- Binary interface
UHAPI Characteristics (2)

- Hardware and implementation independent interface
  - Allows freedom in implementation and evolution
  - Supports both HW and SW streaming
  - Supports both on and off chip peripherals
  - Does not expose the implementation software component architecture

- Processor independent

- Used processor not visible to client
  - Support efficient RPC implementation

- Operating System independent
UHAPI Characteristics (3)

- **Uses standard mechanisms for**
  - Notifications (runtime binding)
  - Error handling
  - Connection management (simple to program)

- **Well-defined execution architecture**

- **Uses standard COM like mechanisms (a small subset)**
  - IUnknown
    - QueryInterface
    - AddRef & Release
  - v-tables
  - GUIDs
Outline

- UHAPI scope
- UHAPI characteristics
- Logical components, roles, interfaces
- Use cases
- UHAPI and variation
- API specification walk-through
- PVR preview
 Logical components, roles and interfaces

- Each logical component has its own specification document
- A logical component can have one or more roles
- A role is an abstract class in UML terms
  - It describes behavior/interface interaction without referencing a particular implementation
- A role typically provides one or more control interfaces and “requires” one or more notification interfaces
- An interface is a coherent set of functions
Logical components, roles and interfaces

VideoMixingClient

GfxLayer

Layer

Mixer

VideoLayer

Digital Component Video

Digital Composite Video
Outline

- UHAPI scope
- UHAPI characteristics
- Logical components, roles, interfaces
- Use cases
- UHAPI and variation
- API specification walk-through
- PVR preview
Use cases

- **Use case:** one or more streaming graphs of connected logical component (LC) instances
- LC instances in a use case are active simultaneously
- UHAPI does not define use cases
- Vendor/implementer chooses use cases for its solution (platform instance)
  - A platform instance may support any number of use cases, including just a single use case.
- **UHAPI implementation realizes the use cases**
  - This relieves the client from the difficult and HW-specific task of setting up and connecting components (priorities, buffer sizes, …)
Example use case: digital input single window

- Tuning 1
- Channel decoding
- ATSC decoder
- Audio processing 1
- Speaker set
- Video processing 1
- Video mixing
- Video processing 2

Actual UHAPI Logical Component instance

Group of UHAPI Logical Component instances
Outline

- UHAPI scope
- UHAPI characteristics
- Logical components, roles, interfaces
- Use cases
  - UHAPI and variation
- API specification walk-through
- PVR preview
Logical component diversity

- Optional interfaces
- Parameter ranges
  - e.g. min/max volume
- Available resources
  - e.g. number of section filters
- Other
  - e.g. which standards are supported

See the “Diversity” section of a Logical Component specification
UHAPI is an API framework to the MW

Universal Home Application Programming Interface
What documentation does a vendor need to provide with a platform instance?

- **Which logical component instances are supported**
  - Some (simple) platform instances may only need a few LC instances
  - No need to include all UHAPI LCs in a platform instance

- **Diversity choices for each logical component instance**
  - For example, availability of optional interfaces (= optional features).

- **What use cases are supported**

- **Optional: Resource usage figures (CPU cycles, memory, memory bandwidth, algorithms used, …)**
Writing MW code: two choices

Platform instance
dependent MW component

Platform instance
independent MW component
(can explicitly handle the
specified diversity)

depends on
(uses also
knowledge of)
Outline

- UHAPI scope
- UHAPI characteristics
- Logical components, roles, interfaces
- Use cases
- UHAPI and variation
- API specification walk-through
- PVR preview
## API Specification

**Content of each Logical Component**

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Logical Component</th>
<th>Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Concepts</td>
<td></td>
<td>5 Interfaces</td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>3.1 Interface-Role Model</td>
<td>5.1 uhlVmfix</td>
</tr>
<tr>
<td>1.2 Blanking</td>
<td>3.2 Diversity</td>
<td>5.1.1 Subscribe</td>
</tr>
<tr>
<td>1.3 Smooth zooming</td>
<td>3.2.1 Provided Interfaces</td>
<td>5.1.2 Unsubscribe</td>
</tr>
<tr>
<td>1.4 Panoramic scaling</td>
<td>3.2.2 Configurable Items</td>
<td>5.1.3 GetSuppNrLayers</td>
</tr>
<tr>
<td>1.5 Color-keying</td>
<td>3.2.3 Constraints</td>
<td>5.1.4 GetLayerConfig</td>
</tr>
<tr>
<td>1.6 Alpha blending</td>
<td>3.3 Instantiation</td>
<td>5.1.5 GetLayer</td>
</tr>
<tr>
<td>2 Types &amp; Constants</td>
<td>3.3.1 Objects Created</td>
<td>5.1.6 GetOutputProperties</td>
</tr>
<tr>
<td>2.1 Public Types &amp; Constants</td>
<td>3.3.2 Initial State</td>
<td>5.1.7 BlankOutput</td>
</tr>
<tr>
<td>2.1.1 ErrorCodes</td>
<td>3.4 Execution Constraints</td>
<td>5.1.8 GetOutputBlanked</td>
</tr>
<tr>
<td>2.1.2 uhvMix_LayerType_t</td>
<td></td>
<td>5.1.9 EnableAutoBlank</td>
</tr>
<tr>
<td>2.1.3 uhvMix_Notification_t</td>
<td>5.2 Roles</td>
<td></td>
</tr>
<tr>
<td>2.1.4 uhvMix_Notifications_t</td>
<td>4.1 Mixer</td>
<td>5.3.1 Subscribe</td>
</tr>
<tr>
<td>2.1.5 uhvMix_AllNotifications</td>
<td>4.2 Layer</td>
<td>5.3.2 Unsubscribe</td>
</tr>
<tr>
<td>2.1.6 uhvMix_LayerNotification_t</td>
<td>4.3 VideoLayer</td>
<td>5.3.3 GetBgSupp</td>
</tr>
<tr>
<td>2.2 ModelTypes &amp; Constants</td>
<td>4.4 GfxLayer</td>
<td>5.3.4 SetBgColor</td>
</tr>
<tr>
<td>2.2.1 Borders</td>
<td>4.5 VideoMixingClient</td>
<td>5.3.5 GetBgColor</td>
</tr>
<tr>
<td>2.2.2 LayerSettings</td>
<td></td>
<td>5.3.6 Hide</td>
</tr>
<tr>
<td>2.2.3 GfxBuffer</td>
<td></td>
<td>5.3.7 GetHidden</td>
</tr>
<tr>
<td>5.3 uhlVmfixLayer</td>
<td></td>
<td>5.3.8 GetBlendFactorRange</td>
</tr>
</tbody>
</table>
Specification views

Interfaces Chapter

function behavior:
the effect of calls to functions in the provides interfaces

Logical Component Chapter

instantiation behavior:
the effect of creating a logical component instance

Roles Chapter

active behavior:
observable actions triggered by internal events

streaming behavior:
the transformation of input stream(s) to output stream(s)

input

output

provides interface

notification interface

transform

call

event

instantiate
Outline

- UHAPI scope
- UHAPI characteristics
- Logical components, roles, interfaces
- Use cases
- UHAPI and variation
- API specification walk-through
- PVR preview
Personal Video Recorder (PVR) extension

- New UHAPI PVR workgroup started in April this year
- Engineers currently working on PVR:
  - Philips (2), Samsung (2), NVIDIA (1), HP (1)

**Process:**
- Step 1: Outline/architecture total PVR solution
- Step 2: Outline/design of individual logical components
- Step 3: Detailed specification of individual logical components

**Current status: in step 2**
- 11-page outline/architecture document available
- Initial Logical Components being written

- Review of results step 1 are now being reviewed by selected ISVs
- PVR specification expected by August 2005
PVR workgroup charter (some key points)

- **To enable Personal Video Recording on Digital reception**
  - Record live broadcast to HDD
  - Play recorded programs from HDD
  - Play program from HDD while recording program to HDD
  - Various trick play modes
  - Support for storing SPTS (TS→SPTS transmuxing for efficient use of storage space)

- **To enable Networked PVR**
  - Align this with DLNA requirements on e.g. SPTS

- **Middleware responsibility**
  - PSI/SI parsing, PVR EPG database build up, maintain index-files with I-frame positions, execution of trick modes, store and retrieve content from HDD

- **Security and PVR on analog reception are second priority**
PVR (preview)

- New logical components for recording an SPTS

Middleware writes SPTS to HDD

- TsDmx
- TsMux
- IFrame Detector

SPTS Transmuxing LC

Extractor

Data extracting LC

TS → SPTS
PVR (preview, cont’d)

- Play back of an SPTS from HDD

Middleware reads SPTS from HDD and injects it.

Data injecting LC

SPTS

New LC

Will be extended to support trickplay (while being backwards compatible)

ATSC Decoder LC (simplified)
PVR (preview, cont’d)

- Example use case for live/delayed viewing

Middleware either passes content directly from extractor to inject (live viewing) or it first stores it on HDD and later reads it back and injects it (delayed viewing)
Summary

- UHAPI is complete, consistent and well documented
- UHAPI is hardware independent
- UHAPI has support for variation and scalability “built into its genes”
- UHAPI allows for vendor extensions in a structured way
- PVR specification under development and expected July 2005
- Alignment with CELF via OSS and DirectFB
- Download UHAPI 1.0 at www.uhapi.org