International Telecommunication Union

ITU-T J.1203

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(05/2020)

SERIES J: CABLE NETWORKS AND TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS

Smart TV operating system

The specification of a smart TV operating system

Recommendation ITU-T J.1203

Recommendation ITU-T J.1203

The specification of a smart TV operating system

Summary

Recommendation ITU-T J.1203 defines the detailed specification of a smart television operating system (TVOS) to enable integrated broadcast and broadband (IBB)-capable cable set-top box (STB) and TV to apply to broadcasting services and IP-based interactive services provided by cable television operators and third-party providers. By running the smart TV operating system, the IBB capable STB and TV will be able to provide subscribers with advanced and personalized services by downloading and installing advanced and personalized apps from cable operators' platforms and third-party platforms, which are interconnected with the related cable operators' platforms.

Recommendation ITU-T J.1203 is developed in accordance with the requirements defined in Recommendation ITU-T J.1201 and based on the architecture defined in Recommendation ITU-T J.1202. This Recommendation provides a specification for administrations and entities who intend to implement a smart TV operating system.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T J.1203	2020-05-29	9	11.1002/1000/14281

Keywords

Broadband, broadcast, cable television, smart TV, smart television operating system, TVOS.

* To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

Table of Contents

Recommendation ITU-T J.1203

The specification of a smart TV operating system

1 Scope

In accordance with the requirements defined in [ITU-T J.1201] and based on the viewpoint of [ITU-T J.1202], this Recommendation provides specification for administrations and entities who intend to implement a smart TV operating system.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T H.770] Recommendation ITU-T H.770 (2015), *Mechanisms for service discovery and selection for IPTV services*.

[ITU-T J.205] Recommendation ITU-T J.205 (2012), *Requirements for an application control framework using integrated broadcast and broadband digital television*.

[ITU-T J.295] Recommendation ITU-T J.295 (2012), *Functional requirements for a hybrid cable set-top box*.

[ITU-T J.1026] Recommendation ITU-T J.1026 (2019), *Downloadable conditional access system for unidirectional networks – Requirements*.

[ITU-T J.1201] Recommendation ITU-T J.1201 (2019), *Functional requirements of a smart TV operating system*.

[ITU-T J.1202] Recommendation ITU-T J.1202 (2019), *The architecture of a smart TV operating system*.

[ECMA 262] Recommendation ECMA-262 (2015), *ECMAScript 2015 Language Specification*.

[W3C CSS2.1] Recommendation W3C CSS2.1 (2011), *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*.

[W3C DOM2 Core] Recommendation W3C DOM2 Core (2000), *Document Object Model (DOM) Level 2 Core Specification*.

[W3C DOM2 Events] Recommendation W3C DOM2 Events (2000), *Document Object Model (DOM) Level 2 Events Specification*.

[W3C DOM2 HTML] Recommendation W3C DOM2 HTML (2003), *Document Object Model (DOM) Level 2 HTML Specification*.

[W3C DOM2 Style] Recommendation W3C DOM2 Style (2000), *Document Object Model (DOM) Level 2 Style Specification*.

[W3C DOM2 Traversal and Range] Recommendation W3C DOM2 *Traversal and Range (2000), Document Object Model (DOM) Level 2 Traversal and Range Specification*.

[W3C DOM2 Views] Recommendation W3C DOM2 Views (2000), *Document Object Model (DOM) Level 2 Views Specification*.

[W3C DOM3 Core] Recommendation W3C DOM3 Core (2004), *Document Object Model (DOM) Level 3 Core Specification*.

[W3C HTML5] Recommendation W3C HTML5.2 (2017), *A vocabulary and associated APIs for HTML and XHTML*.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 Integrated broadcast and broadband (IBB) DTV service [ITU-T J.205]: A service that simultaneously provides an integrated experience of broadcasting and interactivity relating to media content, data and applications from multiple sources, where the interactivity is sometimes associated with broadcasting programmes.

3.1.2 second screen [ITU-T J.295]: This refers to a display screen of mobile phones or other network-enabled devices that show services associated with the television screen.

3.1.3 television operating system (TVOS) [ITU-T J.1201]: A system software running on the IBB-capable cable STB and TV which is capable of managing hardware, software and data resources of IBB-capable cable STB and TV, supporting and controlling the application software execution.

3.1.4 rich execution environment (REE) [ITU-T J.1201]: A hugely extensible and versatile operating environment which brings flexibility and capability.

3.1.5 trusted execution environment (TEE) [ITU-T J.1201]: A secure area of the main processor in an IBB-capable cable STB and TV to ensure that sensitive data is stored, processed and protected in an isolated and trusted environment. It offers isolated safe execution of authorized security software providing end-to-end security by enforcement of protected execution of authenticated code, confidentiality, authenticity, privacy, system integrity and data access rights.

3.1.6 secure OS [ITU-T J.1202]: An operating system running in a trusted execution environment (TEE) which is used to trigger secure execution of applications within the TEE.

3.1.7 downloadable conditional access system (DCAS) [ITU-T J.1026]: A conditional access (CA) system that supports all the features of legacy conditional access and provides a CA-neutral mechanism to securely download CA client image and switch CA terminals without changing hardware through either a broadcasting or a two-way network.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

3.2.1 DRM App: An application running in a television operating system (TVOS) that executes none-secure sensitive digital rights management (DRM) functionalities such as communication with DRM head end and retrieving content authorization. TVOS can manage multiple DRM applications to support different DRM services from different service providers.

3.2.2 DRM TApp: A trusted application running in a television operating system (TVOS) trusted execution environment that executes secure digital rights management (DRM) functionalities such as content decryption, secure video path and trust chain verification.

3.2.3 DCAS App: An application running in a television operating system (TVOS) that executes none-secure sensitive downloadable conditional access system (DCAS) functionalities such as setting a filter to get an entitlement control message (ECM)/entitlement management message (EMM) packet, known as an ECM/EMM packet from a transport stream, and sending the ECM/EMM to the DCAS TApp. TVOS can manage multiple DCAS applications to support different DCAS services from different service providers.

3.2.4 DCAS TApp: A trusted application running in a television operating system (TVOS) trusted execution environment that executes secure downloadable conditional access system (DCAS) functionalities such as ECM/EMM packet decryption and signature verification.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

API	Application Programming Interface
App	Application
ATV	Analogue Television
AV	Audio Video
AVSource	Audio and Video stream Source
CA	Certification Authority
CSS	Cascading Style Sheets
DASH	Dynamic Adaptive Streaming over HTTP
DB	Database
DCAS	Downloadable Conditional Access System
DLNA	Digital Living Network Alliance
DOM	Document Object Model
DRM	Digital Rights Management
DT	Device Tree
DTV	Digital Television
DVB	Digital Video Broadcasting
ECM	Entitlement Control Message
EIT	Event Information Table
EMM	Entitlement Management Message
EPG	Electronic Programme Guide
ES	Elementary Stream
HAL	Hardware Abstraction Layer
HCI	Human-Computer Interaction
HLS	HTTP Live Streaming

HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IPQAM	Internet Protocol Quadrature Amplitude Modulation
JNI	Java Native Interface
JS	Java Script
MPEG	Moving Picture Experts Group
NIT	Network Information Table
OS	Operating System
OSD	On-Screen Display
OTT	Over-The-Top
PAT	Programme Association Table
PCM	Pulse Code Modulation
PID	Packet Identifier
PMT	Programme Map Table
PPV	Pay-Per-View
PSI	Programme Specific Information
REE	Rich Execution Environment
SDT	Service Descriptor Table
SI	Service Information
TApp	Trusted Application
TDT	Time Date Table
TEE	Trusted Execution Environment
TOT	Time of Transmission
TS	Transport Stream
TVM	TV Virtual Machine
TVOS	Television Operating System
UPnP	Universal Plug and Play
VOD	Video On Demand

5 Conventions

In this Recommendation:

The phrase "**is required to**" indicates a requirement which must be strictly followed and from which no deviation is permitted if conformity with this Recommendation is to be claimed.

The phrase "**is recommended**" indicates a requirement which is recommended but which is not absolutely required. Thus this requirement need not be present to claim conformity.

The phrase "**is prohibited from**" indicates a requirement which must be strictly followed and from which no deviation is permitted if conformity with this Recommendation is to be claimed.

The phrase "**can optionally**" indicates an optional requirement which is permissible, without implying any sense of being recommended. This term is not intended to imply that the vendor's implementation must provide the option and the feature can be optionally enabled by the network operator/service provider. Rather, it means the vendor may optionally provide the feature and still claim conformity with this Recommendation.

In the body of this Recommendation and its annexes, the words *shall*, *shall not*, *should*, and *may* sometimes appear, in which case they are to be interpreted, respectively, as *is required to*, *is prohibited from*, *is recommended*, and *can optionally*. The appearance of such phrases or keywords in an appendix or in material explicitly marked as *informative* are to be interpreted as having no normative intent.

6 Reference architecture of the TVOS software

According to the architecture defined in clause 6 of [ITU-T J.1202], TVOS software is recommended to be implemented with the following architecture as defined in [b-GY/T 303.1] and shown in Figure 1.

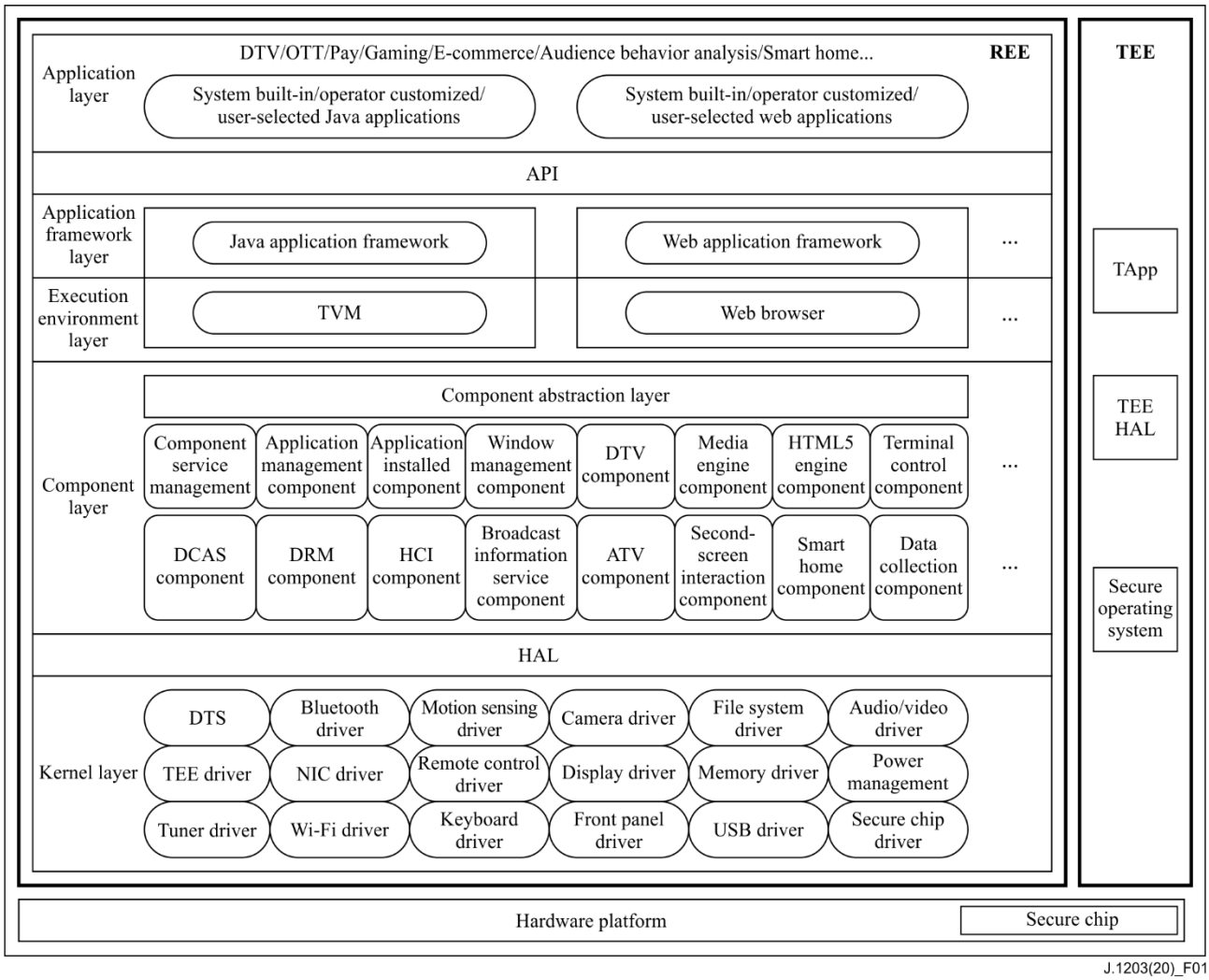

Figure 1 – Reference architecture of the TVOS software

The television operating system (TVOS) kernel layer and the TVOS HAL should be implemented as defined in [ITU-T J.1202].

The TVOS functional component layer is recommended to provide service component modules such as media processing, digital television (DTV), digital rights management (DRM), downloadable conditional access system (DCAS), smart home, human-computer interaction (HCI), terminal control, etc.

The TVOS execution environment layer is recommended to implement the interpretative execution environment of the application software and application adaptation software and support the loading and running of the Java and web applications. The execution environment of Java applications is a TV virtual machine (TVM) and that of web applications is Web Runtime.

The TVOS application framework layer is recommended to provide application programming interfaces (APIs) to Java and web applications, so that Java and web applications can be adapted to the interface encapsulation of functional component modules. The Java application framework contains the TVOS Java application programming interface unit and is compatible with the interface units of third-party Java applications. The web application framework contains the TVOS Web application programming interface units.

7 TV related service components

7.1 DTV component

7.1.1 Functions

The DTV component should implement the following functions:

– Implement the searching, filtering, obtaining, parsing, storage, and management of the programme specific information/service information (PSI/SI) data of various DTV and cable protocols including those used for interactive services.

– Control the tuning and demodulation of demodulation devices and provide functional interfaces and capability support for applications related to DTV live programmes.

– Collaborate with the media engine, DCAS, and DRM components to assist related applications in implementing DTV services such as live programme, programme guide, TV teletext advertisement, video on demand (VOD), programme recording and time-shifting, interactive services, and channel preview.

The DTV component should support parsing of the protocols and tables, supporting the cable, terrestrial wireless, and satellite multi-mode DTV terminals.

7.1.2 Component implementation and invocation mode

The DTV component should be implemented according to the component model. Figure 2 shows the component invocation mode.

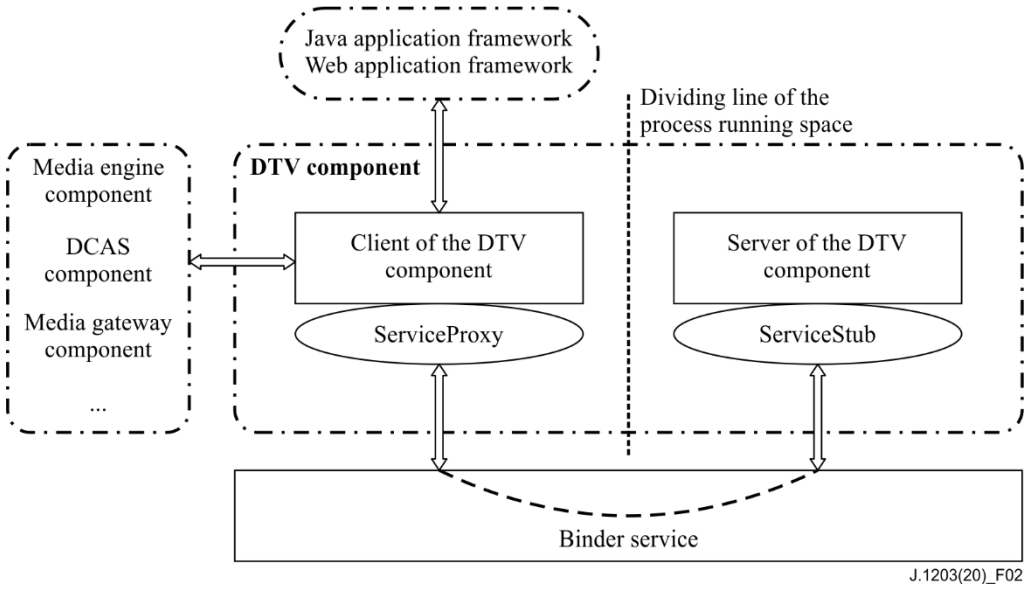

Figure 2 – Component implementation and invocation mode of the DTV component

7.1.3 Functional architecture and modules

The server of the DTV component consists of the tuner, DataEngine, scan, DB, electronic programme guide (EPG), and device tree (DT) modules. Figure 3 shows the functional architecture of the DTV component.

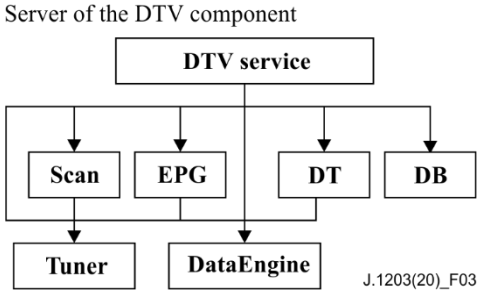

Figure 3 – Functional architecture of the DTV component

The tuner module controls the channel tuning and demodulation of the cable, terrestrial and satellite broadcast demodulators.

The DataEngine module filters and buffers the section data in the currently receiving transport stream and manages lower-layer filters and packet identifier (PID) conflict of the transport stream (TS) packets.

The scan module parses the section data in the basic tables related to the audio and video broadcast programmes and obtains the table information related to the PSI/SI programmes such as network information table (NIT), programme association table (PAT), programme map table (PMT), and service descriptor table SDT. The scan module supports multiple searching modes including automatic search, single-frequency manual search, and frequency-based manual search.

The DB module is a database module and saves the table information related to the PSI/SI programmes such as NIT, PAT, PMT, and SDT and provides the query service for other software modules.

The EPG module parses the section data in the EIT programme event information table, generates the EPG information data, and provides the query service for other software modules.

The DT module parses the section data in the tables related to the time date table (TDT) and time of transmission (TOT) time, generates the time data, and provides the query service for other software modules.

7.1.4 Interfaces

The DTV component provides interfaces for other software modules including tuner control, programme search, PSI/SI data obtaining, EPG data obtaining, and programme information query by using the client. These interfaces are functional component interfaces.

7.1.5 Collaboration with other software modules

The DTV component interacts with the media processing component, DCAS component, and functional units related to different TVOS interpretative application environments. Figure 4 shows the relationship between the DTV component and other software modules.

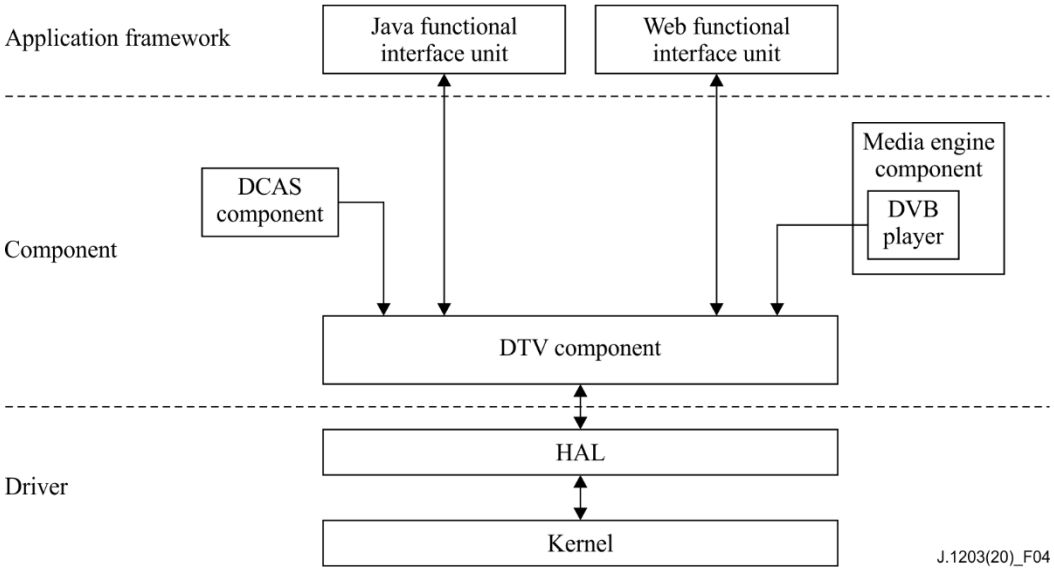

Figure 4 – Relationship between the DTV component and other software modules

7.2 Media engine component

7.2.1 Functions

The media engine functional component module processes various media formats and protocols and collaborates with the lower-layer hardware to implement the playing, recording, and dispatching of various media data of cable DTV and video on demand, over-the-top (OTT), gaming audio and video, and local media files.

The cable DTV function supports playing of the unscrambled and scrambled DTV live streams and provides the playing control functions such as channel selection as well as volume and display size.

The cable video on demand function supports playing of the VOD streams and provides the playing control functions such as programme playing, pause, resume, stop, fast forward, fast rewind, time selection, and volume.

The OTT function supports playing of the non-encrypted and encrypted OTT streams and provides the playing control functions such as programme playing, pause, resume, stop, fast forward, fast rewind, time selection, and volume.

The local media file playing function supports playing of local media files in various audio and video formats and provides the playing control functions such as programme playing, pause, resume, stop, fast forward, fast rewind, time selection and volume.

7.2.2 Component implementation and invocation mode

The media engine component should be implemented according to the component model. Figure 5 shows the component invocation mode.

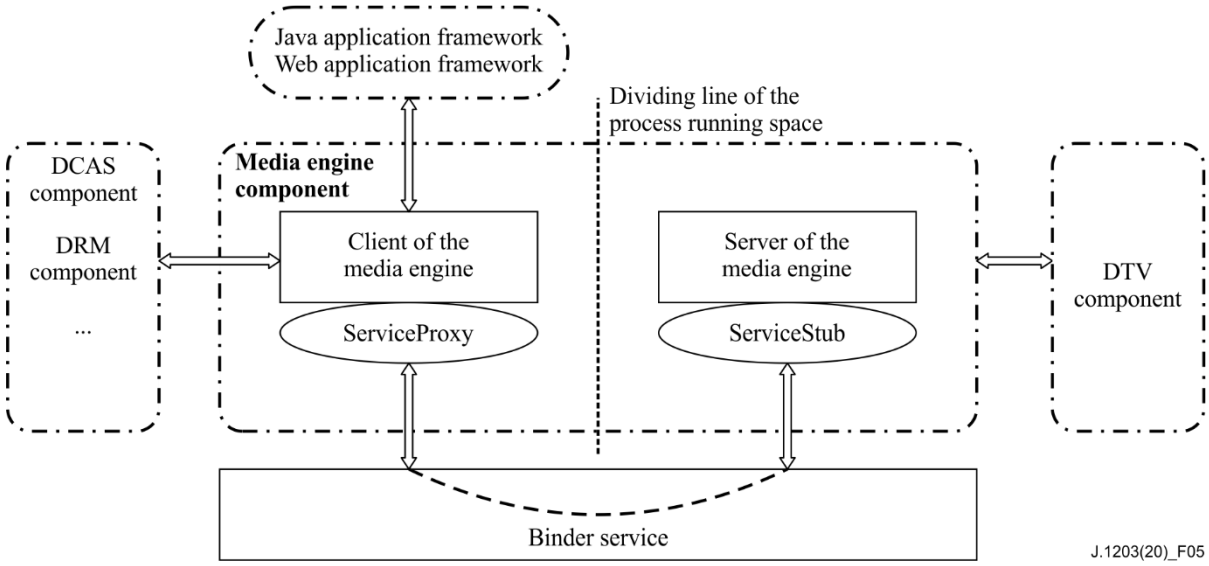

Figure 5 – Implementation and invocation mode of the media engine component

7.2.3 Basic architecture and mechanism

The server of the media engine component should be implemented by using the plug-in-based pipeline full media processing architecture, including the player manager, media players such as the DVBPlayer, VODPlayer, OTTPlayer, and LocalPlayer, media playing pipeline manager, and player pipeline modules such as the DVBPipeline, VODPipeline, OTTPipeline, and LocalPipeline. Figure 6 shows the functional architecture of the media engine component.

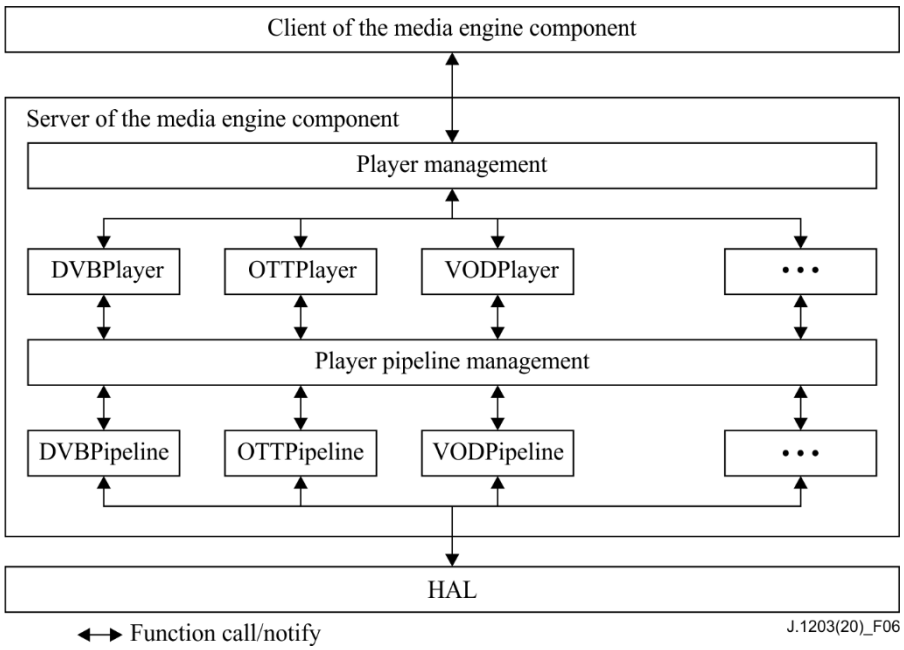

Figure 6 – Functional architecture of the media engine component

The player manager implements player management functions such as player registration, creation, destruction, type selection, and status management, and provides media playing services for other software modules.

The DVBPlayer, VODPlayer, OTTPlayer, and LocalPlayer implements the playing control logic of the corresponding media according to the related media playing and control protocols.

The media playing pipeline manager manages the plug-in elements as well as pipeline setup and running. A plug-in element is a plug-in unit responsible for a single media processing function. The plug-in element types include Source, Typefind, Demux, ProtocolParse, VideoDecode, VideoSink, AudioDecode, and AudioSink.

The DVBPipeline, VODPipeline, OTTPipeline, and LocalPipeline implements the media playing functions of the corresponding media player by combining the plug-in elements according to instructions of the media playing pipeline manager.

The pipeline selects appropriate plug-in elements from the Source, Typefind, Demux, ProtocolParse, VideoDecode, VideoSink, AudioDecode, and AudioSink elements and places them on the corresponding pipeline nodes according to the requirements of the media playing pipeline manager. Figure 7 shows the structure of the player pipeline.

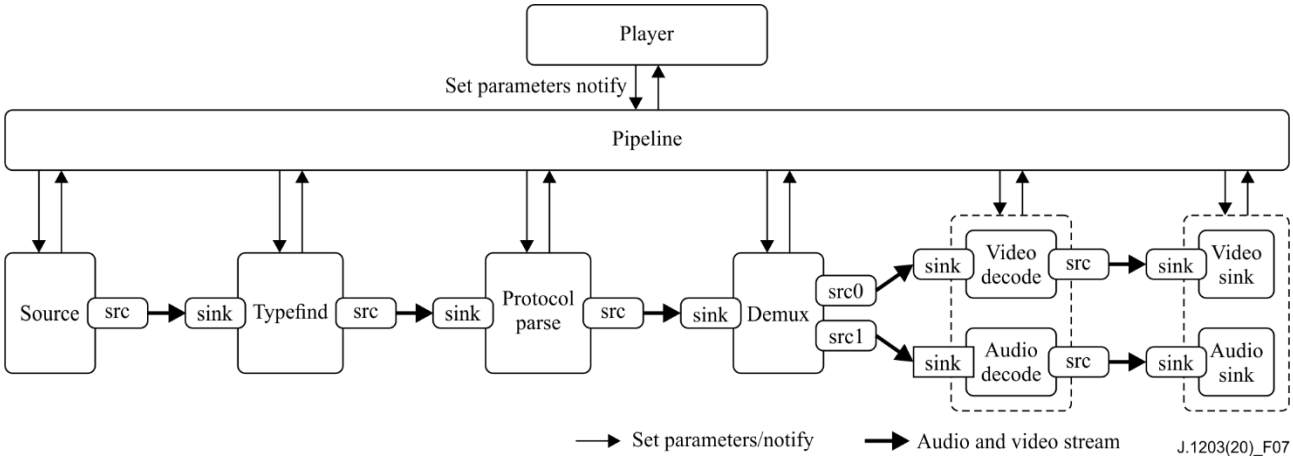

Figure 7 – Structure of the player pipeline

In the media engine component, the player manager controls the media playing pipeline manager through the corresponding media player to select related plug-in elements, sets up the corresponding media playing pipeline, and provides media playing services for the corresponding software modules that request media playing.

The media engine component has a media player corresponding to each type of media playing request. The media playing functions are implemented through the media player pipeline according to the playing and control logic of the corresponding media player.

The media players are extensible and the plug-in elements used to build the media player pipeline also are extensible to support full media parsing and playing.

The media engine component schedules and manages resources such as players according to resource management strategies.

7.2.4 Core functional modules

7.2.4.1 Basic plug-in elements

The basic plug-in element types include Source, Typefind, ProtocolParse, Demux, VideoDecode, VideoSink, AudioDecode, and AudioSink.

a) The Source element obtains media streams, including local storage, HTTP server and so on.

b) The Typefind element identifies the encapsulation format by MIME type of the Source element and determines the appropriate Demux type.

c) The ProtocolParse element parses various media protocols and contains the HLS Parse, DASH Parse, and other elements.

d) The Demux element demultiplexes the media data, extracts the audio, video, and embedded subtitle ESs, and separately transfers them to the decoding plug-in element for decoding.

e) The VideoDecode element implements hardware decoding on the video ES by invoking the HAL interfaces.

f) The VideoSink element outputs decoded video frames through the hardware abstraction layer (HAL) interface.

g) The AudioDecode element implements hardware decoding on the audio ES by invoking the HAL interfaces.

h) The AudioSink element outputs decoded pulse code modulation (PCM) audio frames through the HAL interface.

7.2.4.2 Player pipeline manager

The player pipeline manager manages the plug-in elements as well as pipeline setup and running.

The player pipeline manager is capable of registering and identifying type of plug-in elements and allows selection of appropriate plug-in elements to media data.

The player pipeline manager supports automatic progressive pipeline setup. That is, the pipelines setup starts from creating the Source element. Then an appropriate plug-in element is selected as the node element based on the created Source element and the player capability. The element of the next-level node is determined based on the previous-level node element and the player capability. In this way, the entire player pipeline is constructed progressively.

The player pipeline manager manages the execution of the player pipeline through the operations and control of the control flow, data flow, and state machine as well as clock priority selection.

7.2.4.3 DVBPlayer functional module and DVBPipeline

As a functional module that implements DTV programmes in the media engine component, the DVBPlayer supports playing of the unscrambled and scrambled live streams and provides the playing control functions such as channel selection as well as volume and display size.

The DVBPlayer functional module contains sub-modules such as tuning, programme information obtaining, programme descrambling and programme playing. Figure 8 shows the relationship between the framework of the functional module and other modules.

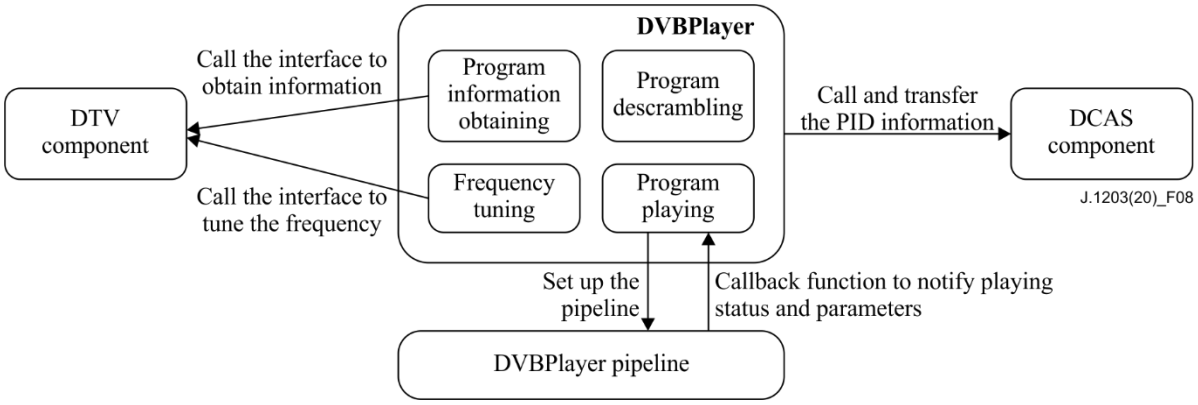

Figure 8 – The DVBPlayer functional module framework and its relationship with other functional modules

The tuning submodule calls interfaces related to the tuner module of the DTV component and tunes to the designated live TV programme channel through the DTV component.

The programme information obtaining submodule calls interfaces related to the DB module of the DTV component and obtains the programme information about the live TV programme through the DTV component.

The programme descrambling submodule calls interfaces related to the DCAS component and transfers the scrambling information of the live TV programme to the DCAS component to descramble the live TV programme.

The programme playing submodule collaborates with the media playing pipeline manager to build the DVBPipeline through combination of the plug-in elements related to the media engine, realizing the function of playing live TV programmes.

The DVBPipeline implements media playing functions under the control of the media playing pipeline manager in response to the control of the DVBPlayer. Figure 9 shows the structure of the DVBPipeline.

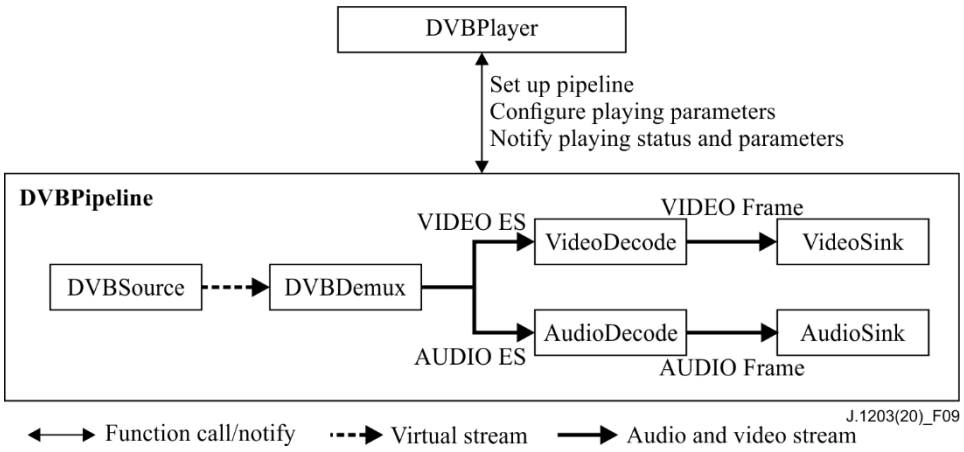

Figure 9 – Structure of the DVBPipeline

The DVBPipeline includes DVBSource, DVBDemux, VideoDecode, AudioDecode, VideoSink, AudioSink elememts. The DVB Source is a virtual element to make the pipeline complete and does not generate data. DVBDemux gets element streams and sends to VideoDecode and AudioDecode. VideoDecode and AudioDecode elements decode the element stream to video frame and audio frame. VideoSink and AudioSink elements get and render the video and audio frames.

7.2.4.4 VODPlayer functional module and VODPipeline

As a functional module that implements DTV on demand and playing in the media engine component, the VODPlayer supports the functions of playing of VOD streams and providing the related control functions such as programme playing, pause, resume, stop, fast forward, fast rewind, time selection and volume setting.

The VODPlayer functional module contains sub-modules such as frequency tuning, programme information obtaining, signalling interaction protocol processing, and programme playing. Figure 10 shows the relationship between the framework of the functional module and other modules.

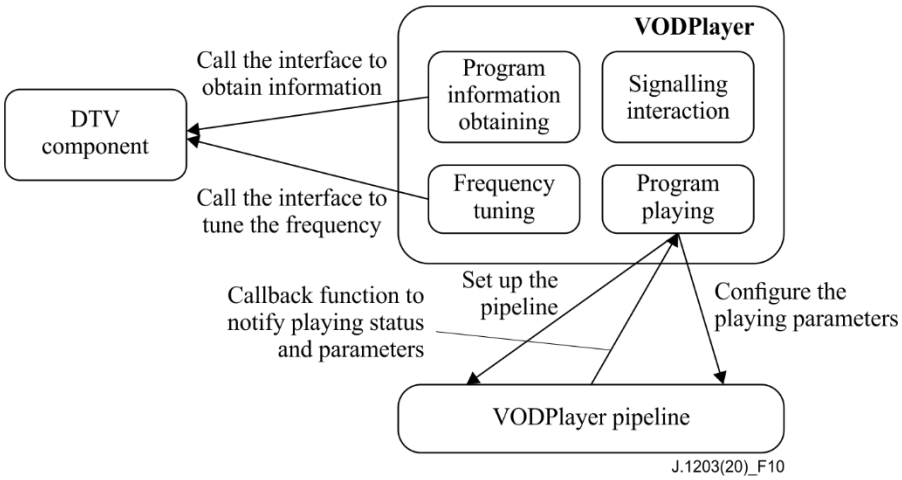

Figure 10 – The VODPlayer functional module framework and its relationship with other functional modules

The frequency tuning submodule invokes interfaces related to the tuner module of the DTV component and tunes the frequency of the on-demand programme channel through the tuner module.

The programme information obtaining submodule invokes interfaces related to the DB module of the DTV component and obtain the programme information about the on-demand programme through the DB module.

The signalling interaction protocol processing submodule implements signalling interaction with the front-end DTV VOD system and control the programme playing submodule to play audio on demand streams and VOD streams based on the Internet protocol quadrature amplitude modulation (IPQAM) channel or IP broadband channel.

The programme playing submodule collaborates with the media playing pipeline manager to build the VODPipeline through combination of the plug-in elements related to the media engine, implementing the function of playing on demand TV programmes.

VODPipeline supports IPQAM channel and IP channel.

Figure 11 shows the structure of the VODPipeline.

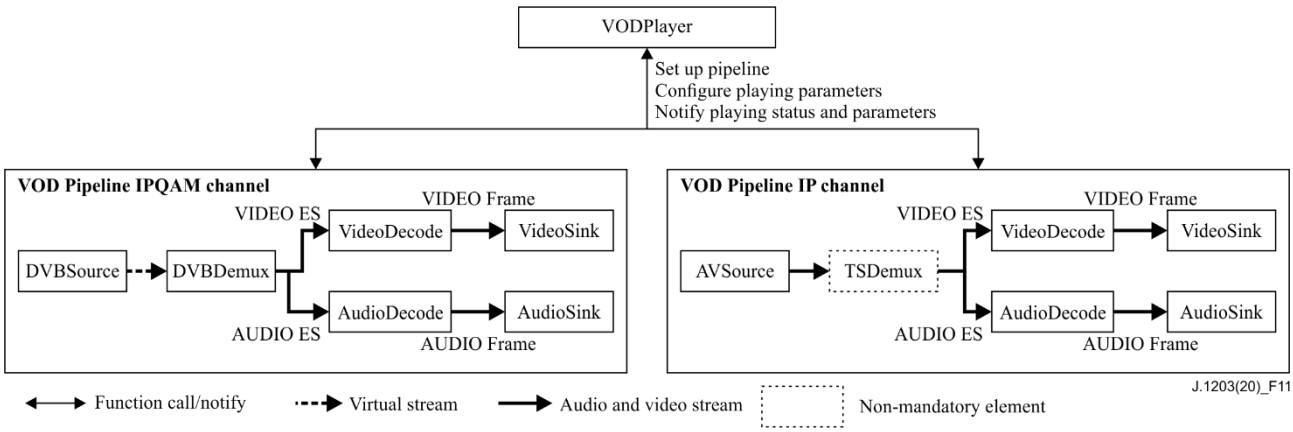

Figure 11 – Structure of the VODPipeline

The IPQAM channel pipeline implements the the function of playing VOD streams using IPQAM channel and includes DVBSource, DVBDemux, VideoDecode, AudioDecode, VideoSink,

AudioSink elememts. The DVBSource is a virtual element to make the pipeline complete and does not generate data. DVBDemux gets element streams and sends to VideoDecode and AudioDecode. VideoDecode and AudioDecode elements decode the element stream to video frame and audio frame. VideoSink and AudioSink elements get and render the video and audio frames.

The IP channel pipeline implements the the function of playing VOD streams using IP channel and includes AVSource, TSDemux, VideoDecode, AudioDecode, VideoSink, AudioSink elememts. The AVSource gets the VOD stream in TS or ES format. TSDemux is loaded in case of TS format VOD stream. TSDemux demultiplexes the TS stream into element streams and sends element streams to VideoDecode and AudioDecode. VideoDecode, AudioDecode, VideoSink and AudioSink elements implements the same as in IPQAM channel mode pipeline.

7.2.4.5 OTTPlayer functional module and OTTPipeline

OTTPlayer is a functional module that implements the Internet TV playing function in the media engine component and supports the function of playing on demand streams based on the streaming media protocols such as HTTP live streaming (HLS) and MPEG DASH and provides the related control functions such as programme playing, pause, resume, stop, fast forward, fast rewind, time selection and audio volume control.

OTTPlayer functional module consists of programme decryption and playing sub-module, Figure 12 shows the OTTPlayer functional module framework and its relationship with other functional modules.

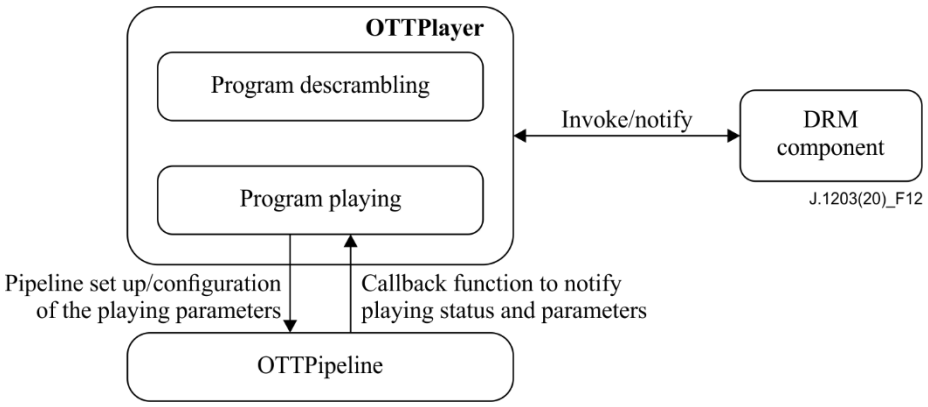

Figure 12 – The OTTPlayer functional module framework and its relationship with other functional modules

The programme decryption sub-module implements programme decryption by invoking related interfaces of the DRM functional module, and sending the encrypted information of the encrypted on demand programme to the DRM functional module and getting the respective decrypted information from the DRM functional module.

The programme playing sub-module implements on demand programme playing functions by establishing OTTPipeline through coordinating with player pipeline manager and combining with related plugin elements of the media engine.

OTTPipeline supports playing unencrypted stream and encrypted stream.

OTTPipeline for playing unencrypted stream includes Source, TypeFind, ProtocolParse, Demux, AudioDecode, VideoDecode, AudioSink and VideoSink elements.

Figure 13 shows the structure of the OTTPipeline for unencrypted stream.

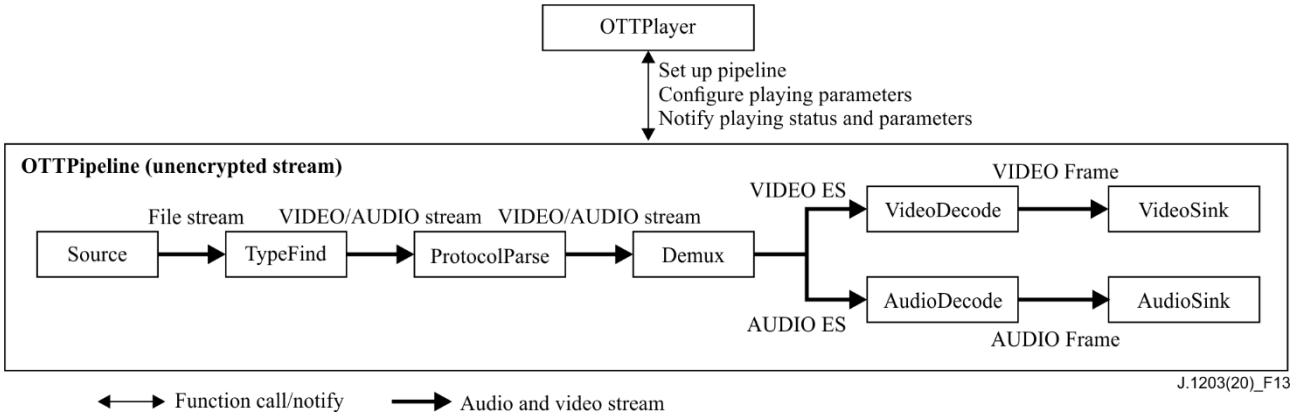

Figure 13 – The structure of the OTTPipeline for unencrypted stream

Source element gets media streaming files from the Internet, and passes them to TypeFind. TypeFind element analyses the received streaming files from the Source element, and selects the right ProtocolParse element based on the result of the analysis in order to establish the appropriate OTTPipeline. ProtocolParse element can analyse streaming media protocols such as HLS and DASH, and parses the received streaming file from the TypeFind element, and sends the parsed streaming file to Demux element. Demux element processes the demultiplexing of the received streaming file from the ProtocolParse element, and sends the processed result to AudioDecode and VideoDecode elements. AudioDecode and VideoDecode elements perform audio and video decoding respectively, and generate audio and video data frames. AudioSink and VideoSink elements get audio and video data frames from AudioDecode and VideoDecode elements, and render them.

OTTPipeline for playing encrypted stream includes Source, TypeFind, ProtocolParse, Demux, AudioDecode, VideoDecode, AudioSink and VideoSink elements. The AudioDecode and VideoDecode send encrypted ES to the respective audio decoder and video decoder in the secure channel.

Figure 14 shows the structure of the OTTPipeline for encrypted stream.

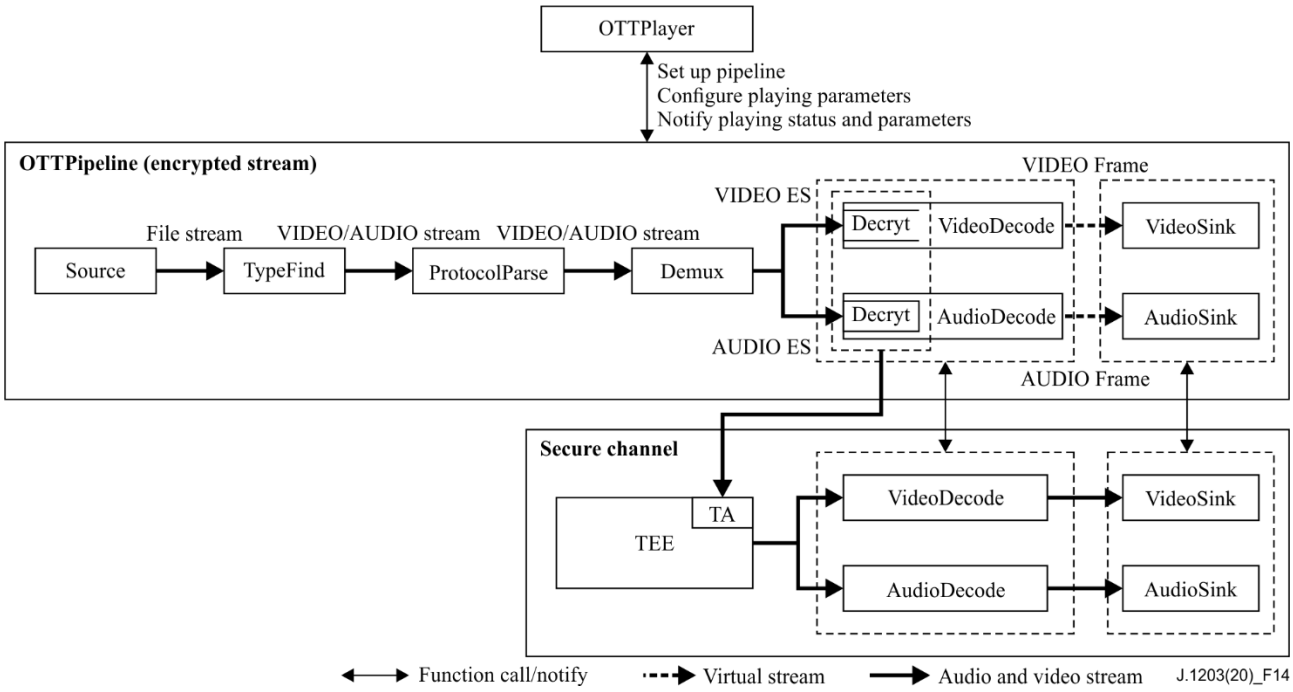

Figure 14 – The structure of the OTTPipeline for encrypted stream

The Source element gets media streaming files from Internet, and passes them to the TypeFind element. The TypeFind element analyses the encrypted streaming files received from the Source element, and selects the right ProtocolParse element based on the result of the analysis in order to establish the appropriate OTTPipeline. The ProtocolParse element can analyse streaming media protocols such as HLS and DASH, and parses the encrypted streaming file received from the TypeFind element, and sends the parsed streaming file to the Demux element. The Demux element processes the demultiplexing of the encrypted streaming data received from the ProtocolParse element, and sends the processed result to AudioDecode and VideoDecode elements. AudioDecode and VideoDecode elements send the encrypted streaming data received from the Demux element to the secure channel, and invoke the DRM functional module to assist the decryption and decoding of the encrypted streaming data by the related secure hardware and generate audio and video data frames in the secure channel. AudioSink and VideoSink elements control the rendering of audio and video data frames in the secure channel.

7.2.4.6 LocalPlayer functional module and LocalPipeline

LocalPlayer functional module implements playing of the local media files in the media engine component and supports parsing of media files in different formats and provides the related control functions such as programme playing, pause, resume, stop, fast forward, fast rewind, time selection and audio volume setting.

Figure 15 shows the LocalPlayer functional module framework and its relationship with other functional modules.

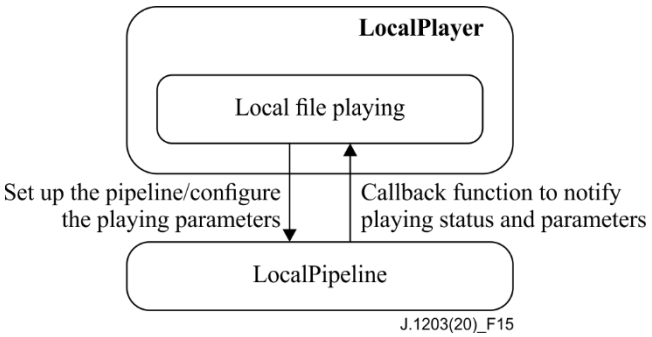

Figure 15 – The LocalPlayer functional module framework and its relationship with other functional modules

The Local file playing sub-module implements local file playing functions by establishing LocalPipeline through coordinating with player pipeline manager and combining with related plugin elements of the media engine.

LocalPipeline includes FileSource, TypeFind, Demux, AudioDecode, VideoDecode, AudioSink and VideoSink elements. Figure 16 shows the structure of the LocalPipeline.

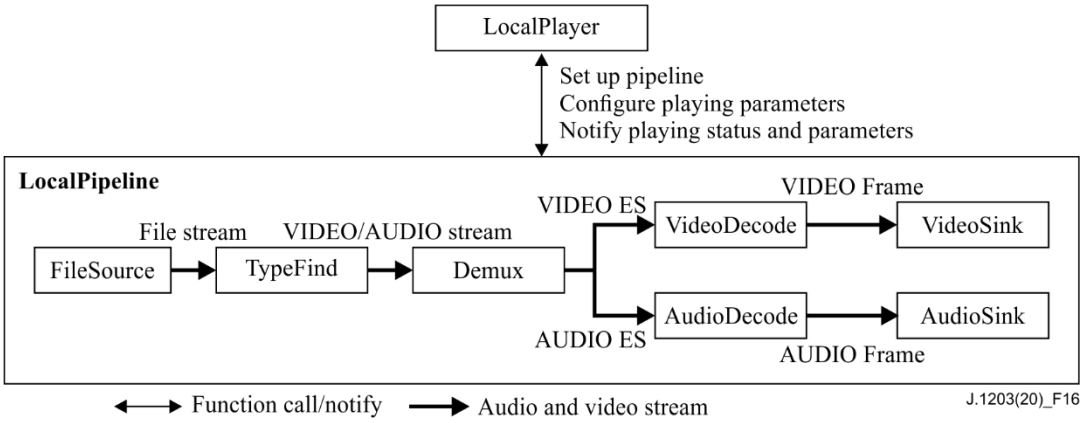

Figure 16 – The structure of the LocalPipeline

The FileSource element gets local files, and passes them to the TypeFind element. The TypeFind element analyses the received files from the FileSource element, and selects the right Demux element based on the result of the analysis in order to establish the appropriate LocalPipeline. The Demux element processes the demultiplexing of the received file from the TypeFind element, and sends the processed result to AudioDecode and VideoDecode elements. AudioDecode and VideoDecode elements perform audio and video decoding respectively, and generate audio and video data frames. AudioSink and VideoSink elements get audio and video data frames from AudioDecode and VideoDecode elements and render them.

7.2.5 Interfaces

The media engine component provides interfaces for other software modules in a unified manner through the component client. The invocation interfaces provided by the component client should be extended according to the extension requirements of the media player functions. These interfaces are functional component interfaces.

7.2.6 Collaboration with other software modules

Figure 17 shows the relationship between the media engine component and other software modules.

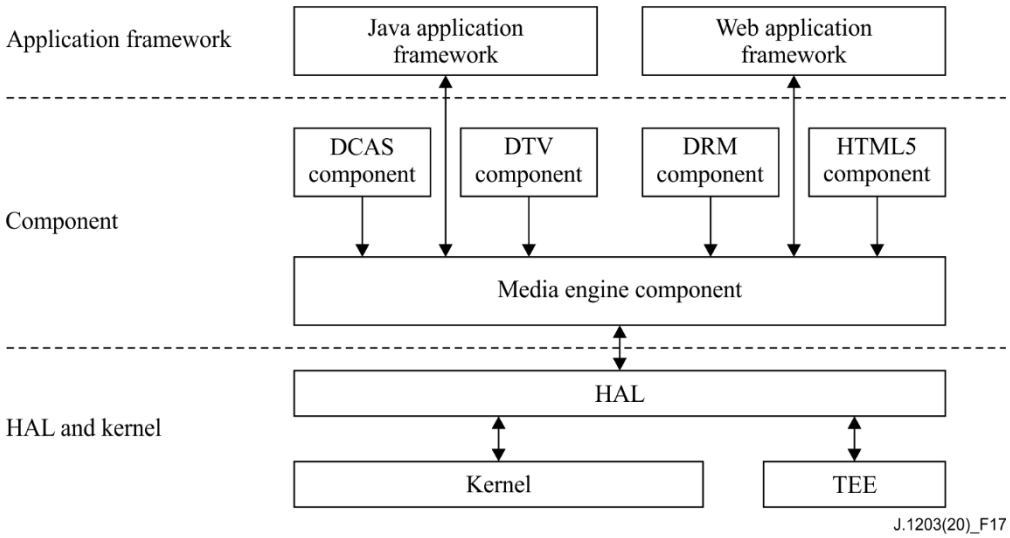

Figure 17 – Relationship between the media engine component and other software modules

7.3 HTML5 engine component

7.3.1 Functions

The HTML5 engine component implements the following functions:

– Downloading, parsing, rendering, typesetting, presentation, and script execution of HTML5 web pages.

– Cooperating with the application management component to suspend, abort, and destroy web applications.

– Cooperating with the window management component to implement the event forwarding function, such as forwarding the input event.

– Adapting to and invoking other components such as the DTV, DCAS, DRM, media engine, and HCI components.

– Providing unified HTML5 engine interfaces for Web Runtime and other upper-layer modules; have the capability of invoking HAL interfaces and other lower-layer modules.

The HTML5 engine component should support HTML5 interface [W3C HTML5], CSS interface [W3C CSS2.1], JS interface [ECMA 262] and DOM interface [W3C DOM3 Core] [W3C DOM2 HTML], [W3C DOM2 Core], [W3C DOM2 Events], [W3C DOM2 Style], [W3C DOM2 Traversal and Range], an [W3C DOM2 Views] and adapt the JS APIs related to the web application framework, including the TVOS Web application programming interface, DCAS APIs, and broadcast information service APIs.

7.3.2 Component mplementation and invocation mode

The HTML5 engine component uses the client-server model. The component server implements page downloading, parsing, rendering, typesetting, presentation, and script execution. The component client cooperates with the application management component and Web Runtime to suspend, abort, and destroy applications and manage instances that run on the component server. The component client also cooperates with the window management component to implement event forwarding functions such as the input event. Both the server and client can run multiple instances and these instances run in different process space. Other software modules such as the application management component and Web Runtime can create different instances that run on the client. Web Runtime can create multiple different instances that run on the component server. Generally a web

application corresponds to an instance that runs on the component server. Figure 18 shows the implementation and invoke mode of the HTML5 engine component.

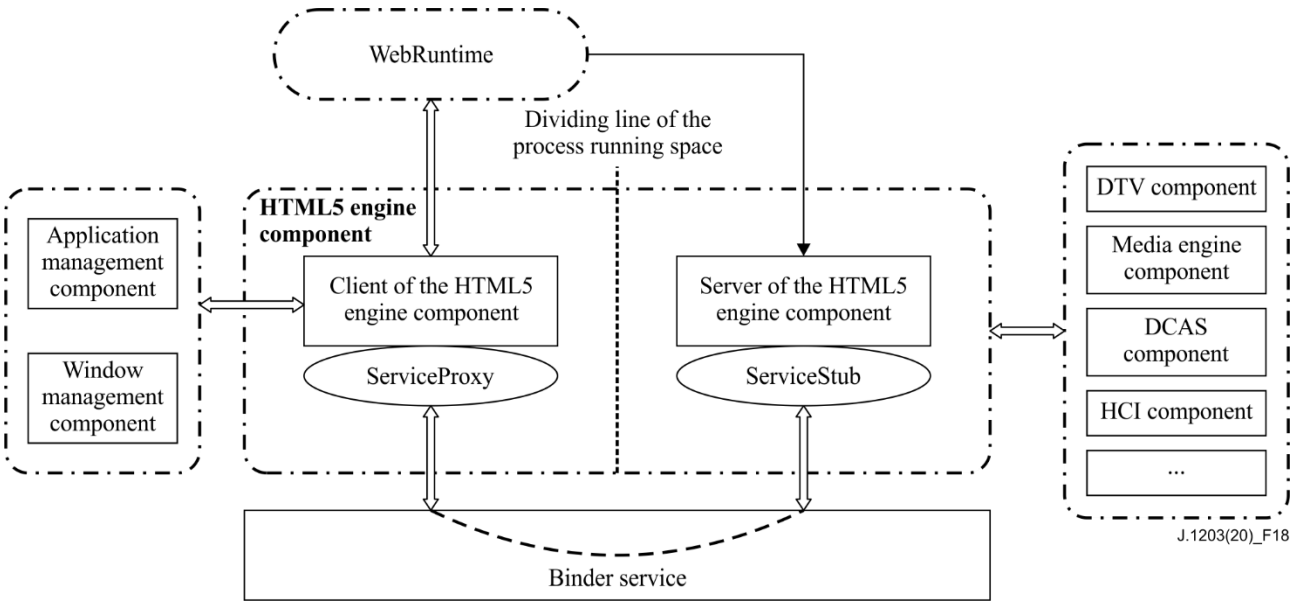

Figure 18 – Implementation and invocation mode of the HTML5 engine component

7.3.3 Functional architecture and modules

The server of the HTML5 engine component contains multiple functional modules such as the ContentShell, ContentBrowser, graphics, network, window, ContentRender, HTML rendering, JS execution, plug-in, media, font, diagnosis, and database. Figure 19 shows the functional architecture of the component server.

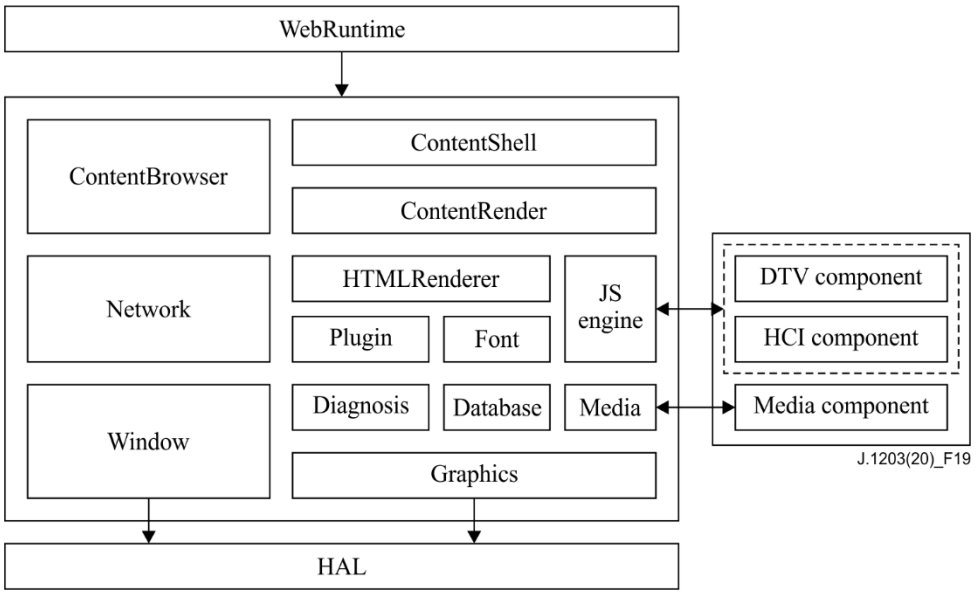

Figure 19 – Functional architecture of the component server

The ContentShell module provides the HTML5 engine interfaces, which are invoked by Web Runtime.

The ContentBrowser module receives ContentShell commands, manages message scheduling of the graphics, network, and window modules, and transfers messages for the ContentRender module.

The ContentRender module manages message scheduling of the HTML rendering, JS execution, plug-in, media, font, diagnosis, and database modules.

The HTML rendering module loads page resources, parses the content and format, calculates the layout, and displays the rendering effect.

The JS execution module parses and runs the JS as well as controls and accesses the DOM tree through the script, implementing dynamic interaction.

The plug-in module manages the plug-ins of the HTML5 engine so that the browser can execute external programmes in plug-in mode.

The graphics module implements drawing and image compositing. It is recommended to support hardware acceleration.

The network module downloads various types of resources.

The media module connects to the media engine component to implement the media playing function of the web page.

The diagnosis module implements the programme crash capture mechanism, supports cross-platform crash dump, and has capability of performing unified exception analysis.

The window module implements window management, UI event interaction mechanism, and element display.

The database module manages the database.

7.3.4 Interfaces

Both the server and client of the HTML5 engine component provide external interfaces. The component server provides the fork, ContentShell, and ContentBrowser interfaces. The component client provides the input event, window drawing event, and application management interfaces.

The fork interface allows the component server to create running instances.

The ContentShell interface allows the component server to start auto-running services.

The ContentBrowser interface provides the application entry loading function, management services such as browsing history recording, and user agent configuration service.

The input event interface receives messages of input devices such as the remote control.

The window drawing event interface distributes the drawing, re-drawing, and invalid area events.

The application management interface provides the functionalities to run, pause, interrupt, and destroy applications.

These interfaces are functional component interfaces.

7.3.5 Collaboration with other software modules

Figure 20 shows the position of the HTML5 engine in the TVOS and its surrounding associated components.

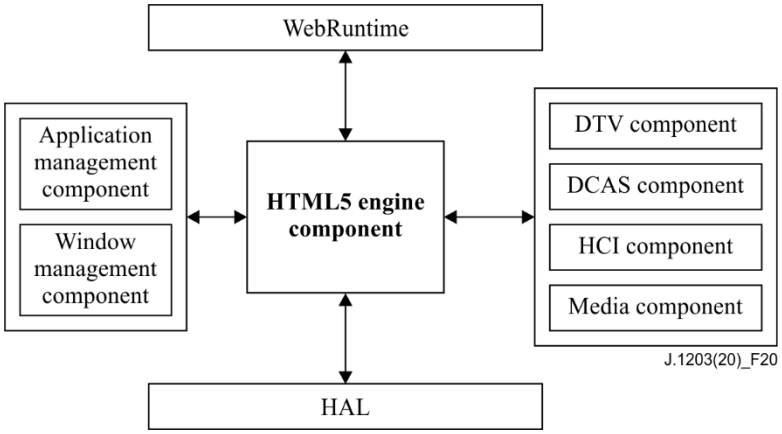

Figure 20 – Collaboration of the HTML5 engine with other software modules

7.4 DRM component

7.4.1 Functions

The DRM component should implement the following functions:

– Manage the registration, deregistration, and running status of DRM App.

– Transfer messages between the media engine component and DRM App and DRM TApp.

– Allow the DRM App, DRM TApp, and media component to collaboratively decrypt encrypted media streams.

– Provide DRM invocation interfaces for the functional interface units at the application framework layer and other components at the functional component layer.

7.4.2 Component implementation and invocation mode

The DRM component should be implemented according to the component model. Figure 21 shows the structure of the components.

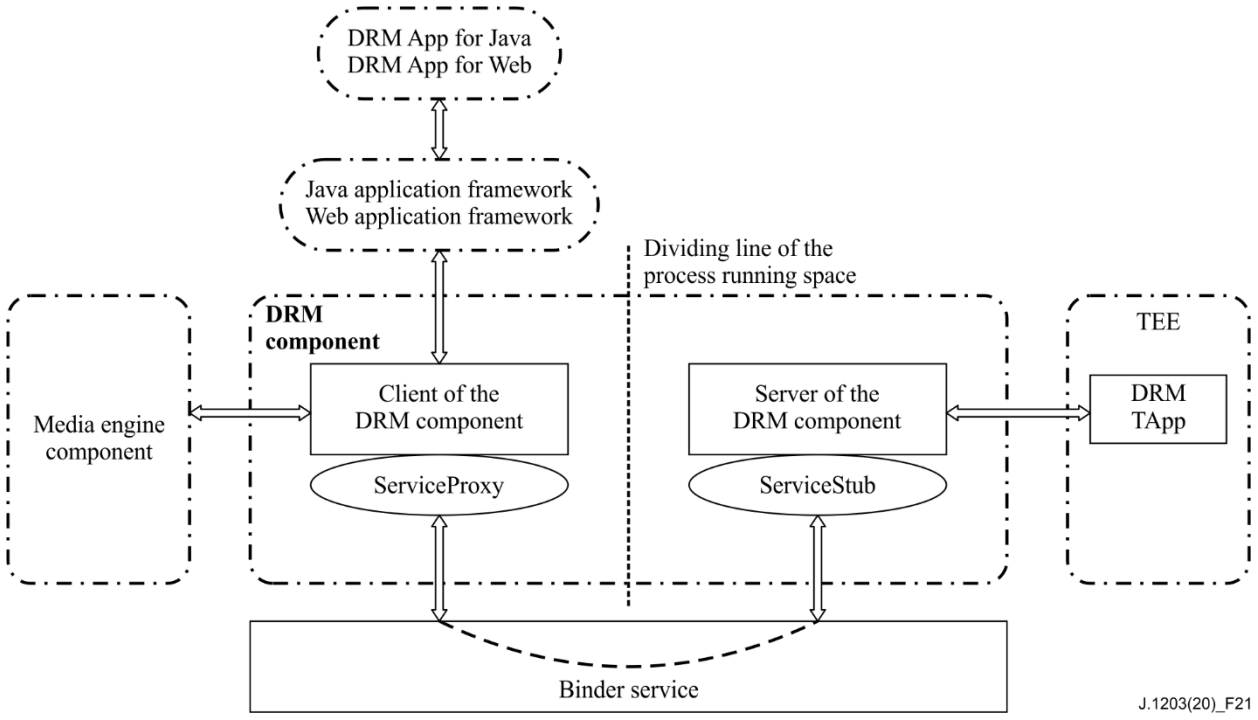

Figure 21 – The structure of the DRM components

7.4.3 Functional architecture and modules

The DRM component server consists of the DRM application manager and message manager. Figure 22 shows the functional architecture.

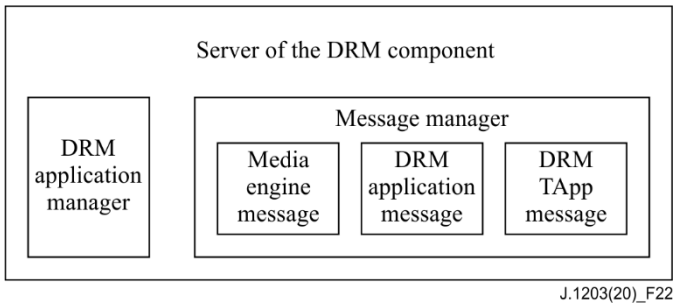

Figure 22 – Functional architecture of the DRM component

The DRM application manager should manage the DRM App, maintain the DRM App identifier as well as the mapping between the DRM App and DRM TApp, receive requests from other service modules in the DRM Server, and execute signature verification, loading, invocation and unloading of the DRM App.

The message manager should transfer messages between the media engine component and the DRM App, and the DRM App should transfer the encrypted session key and encrypted content key which are downloaded from the server to the DRM TApp.

The message manager transfers messages on stream decryption from the media engine component and/or the DRM App to the DRM TApp. The message manager receives the information about the encrypted content handling from the media engine component and the content id from the DRM App, and passes the received information to the DRM TApp. As responses, the message manager receives information of decryption status and handling of the decrypted data from the DRM TApp, and passes the received information to the media engine component, and to the DRM App.

7.4.4 Interfaces

The DRM component should provide the following two types of interfaces:

– Decrypting content interface:

  This interface receives the DRM decryption information including content information, product key information, and encrypted content and is invoked by the media engine component.

– DRM application interface:

  This interface is used for data interaction between the DRM App and the DRM component.

These interfaces are functional component interfaces.

7.4.5 Collaboration with other software modules

The DRM component should collaborate with relevant functional interface units such as the application framework layer, component layer, DRM TApp, and media engine component. Figure 23 shows the collaborative relationship.

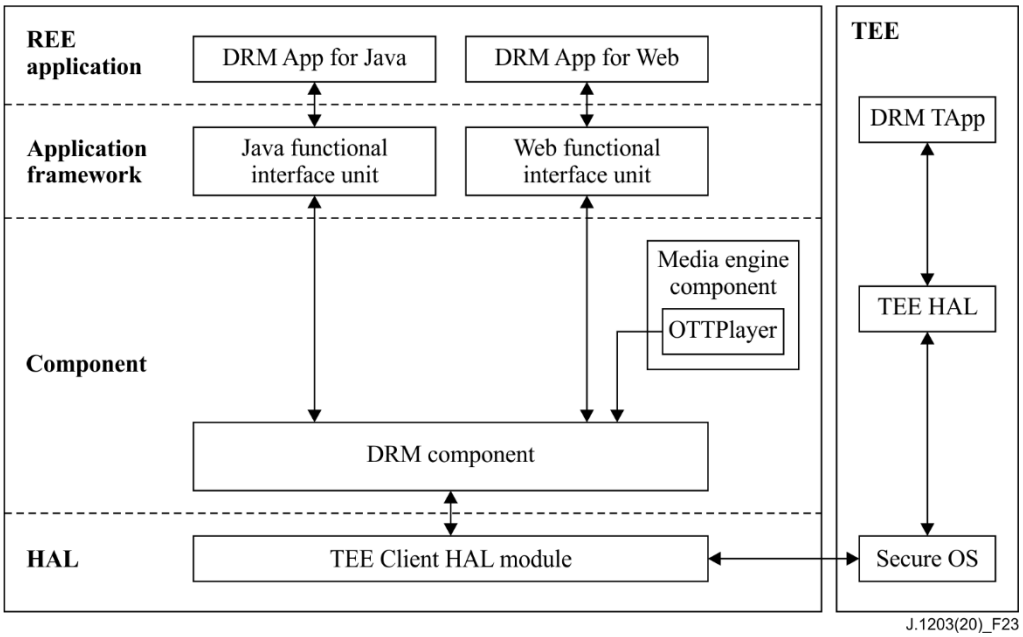

Figure 23 – Relationship between the DRM component and other software modules

7.5 DCAS component

7.5.1 Functions

The DCAS component should implement the following functions:

– Collaborate with the DTV component to receive and forward the in-band transmission CA ECM and EMM.

– Collaborate with the network protocol stack module to receive and forward the out-band transmission CA EMM.

– Provide a message exchange channel for the DCAS App and DCAS TApp.

– Assist the media engine component in implementing message interaction between the DCAS App and DCAS TApp.

– Register and manage the DCAS App.

– Support query of the CA-related information such as the CA version, chip ID, and entitlement status, and receive and forward the OSD and fingerprint display information.

7.5.2 Component implementation and invocation mode

The DCAS component should be implemented according to the component model. Figure 24 shows the component invocation mode.

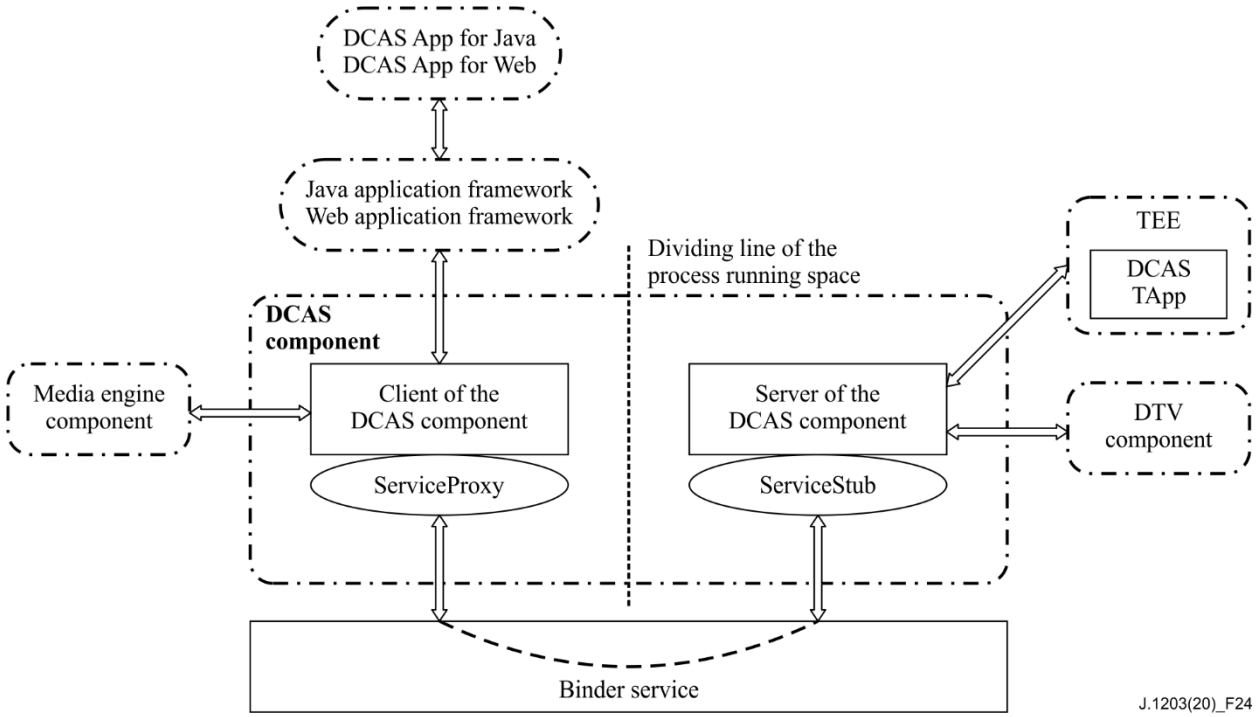

Figure 24 – Component implementation and invocation mode of the DCAS component

7.5.3 Functional architecture and modules

The DCAS component should contain three functional modules: CA data reception and forwarding, CA application management, and TApp interface. Figure 25 shows the component architecture.

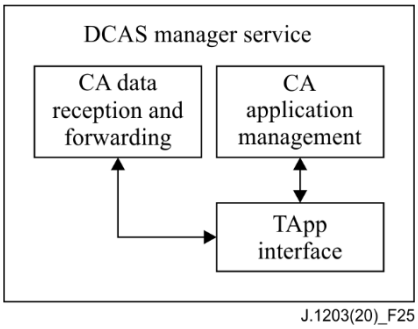

Figure 25 – Logical architecture of the DCAS component

The CA data reception and forwarding module obtains the ECM and EMM data from the DTV component or network protocol stack module according to the received DTV descrambling parameter information, and then forwards the obtained data to the DCAS App.

The TApp interface module interfaces with the DCAS TApp and provides a message exchange channel between the DCAS App and DCAS TApp.

The CA application management module implements the following functions:

– Manage the registration and deregistration of the DCAS application.

– Match the encrypted media flow with the DCAS application according to the CAS identifier.

– Receive the DTV descrambling parameters from the media engine and forward them to the matched DCAS application. The parameters include the DTV programme video stream

identity, audio stream identity, CA application identity, ECM identity, EMM identity, and stream path.

– Query the CA-related information such as the CA version, chip ID, and entitlement status, and receive and forward the OSD and fingerprint display information.

7.5.4 Interfaces

The DCAS component should provide the following two types of interfaces:

– Descrambling operation notification interface:

  This interface receives the DCAS descrambling notification information (including starting descrambling and stopping descrambling notification) and is invoked by the media engine component.

– CA application interface:

  This interface is used for data interaction between the DCAS App and the DCAS component.

These interfaces are functional component interfaces.

7.5.5 Collaboration with other software modules

Figure 26 shows the relationship between the DCAS component and other components.

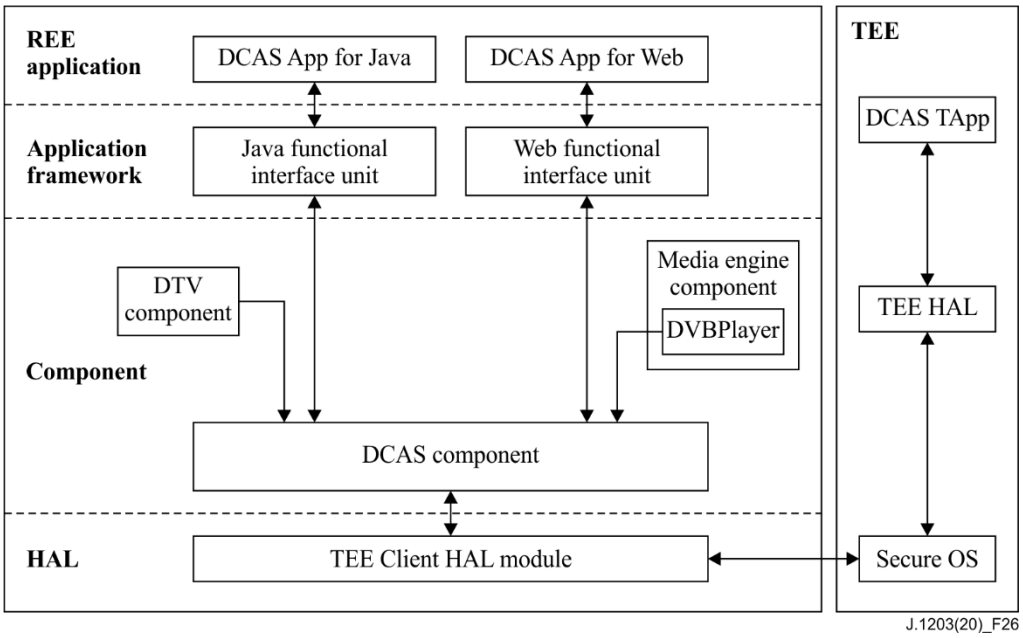

Figure 26 – Relationship between the DCAS component and other components

The media engine component needs to invoke the descrambling operation notification interface of the DCAS component.

The DCAS App needs to invoke the CA application interface of the DCAS component through the functional interface units related to TVOS Java application programming interface or TVOS Web application programming interface.

The DCAS component needs to rely on the interfaces related to DTV component data obtaining and interfaces of the TEE client HAL module.

7.6 Smart home component

7.6.1 Functions

The smart home component should implement the following functions:

– Identify smart home devices, establish connections, and control the devices.

– Provide interfaces to convert and adapt to third-party smart home interconnection protocols. These interfaces support the extension of third-party smart home interconnection protocols and manage smart home devices.

– Configure Wi-Fi network parameters for smart home devices.

– Support third-party smart home interconnection protocols, network parameter configuration protocols, and standard Bluetooth protocols.

7.6.2 Component implementation and invocation mode

The smart home component should be implemented according to the component model. Figure 27 shows the component invocation mode.

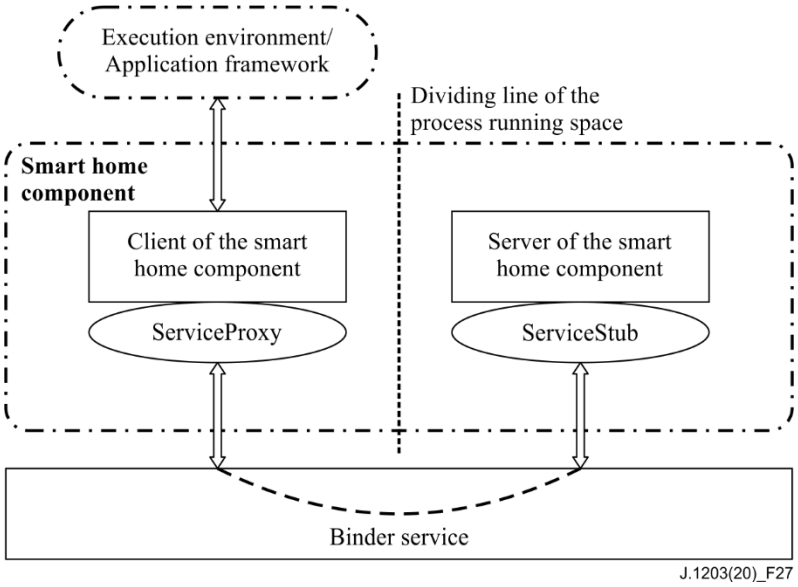

Figure 27 – Component implementation and invocation mode of the smart home component

7.6.3 Functional architecture and modules

The smart home component consists of the device configuration module, smart home management module, and protocol conversion and adaptation module. Figure 28 shows the structure.

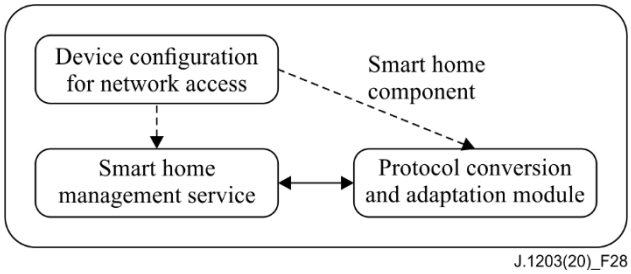

Figure 28 – Architecture of the smart home component

The device configuration module should configure the network access parameters for smart home devices based on network parameter configuration protocols.

The smart home management module should identify, connect, and control smart home devices. This module supports third-party smart home interconnection protocols and related secure interconnection functions. This module also provides other software modules with invocation interfaces to control smart home devices.

The protocol conversion and adaption module should provide interfaces used to convert and adapt third-party smart home interconnection protocols. These interfaces support the extension of third-party smart home interconnection protocols.

7.6.4 Interfaces

Each module in the smart home component provides its own interface to work together with other software modules or components as shown in Figure 28:

a) Device network access interface of the smart home component: This interface configures network parameters for the Wi-Fi or Bluetooth smart home devices.

b) Device management interface of the smart home component: This interface manages smart home devices.

c) Protocol conversion and adaptation interface of the smart home component: This interface converts and adapts the third-party smart home interconnection protocols for other software modules.

These interfaces are functional component interfaces.

7.7 HCI component

7.7.1 Functions

The main objective of HCI component is to manager TVOS input devices such as remote controller, mouse, keyboard, game handle, and smartphone. It provides unified management and adaptation for different human-computer interaction devices to access TVOS with customized API, and thus to enhance user experience of human-computer interaction by making the application more convenient to access all kinds of input devices.

The first function of HCI component is to realize the adaptation of all kinds of input devices. The second function is to realize information encapsulation and transportation for various input information including key message, voice message, sensor message, game handle message, virtual device control message, etc.

HCI component should support following scenarios:

a) The TVOS terminal is controlled by universal input devices such as remote controller, mouse and keyboard.

b) The TVOS terminal is controlled by voice input devices such as voice remote control and microphone.

c) The TVOS terminal is controlled by game handles.

d) The TVOS terminal is controlled by smart terminals such as smart phone, smart speaker.

Figure 29 shows HCI component scenarios.

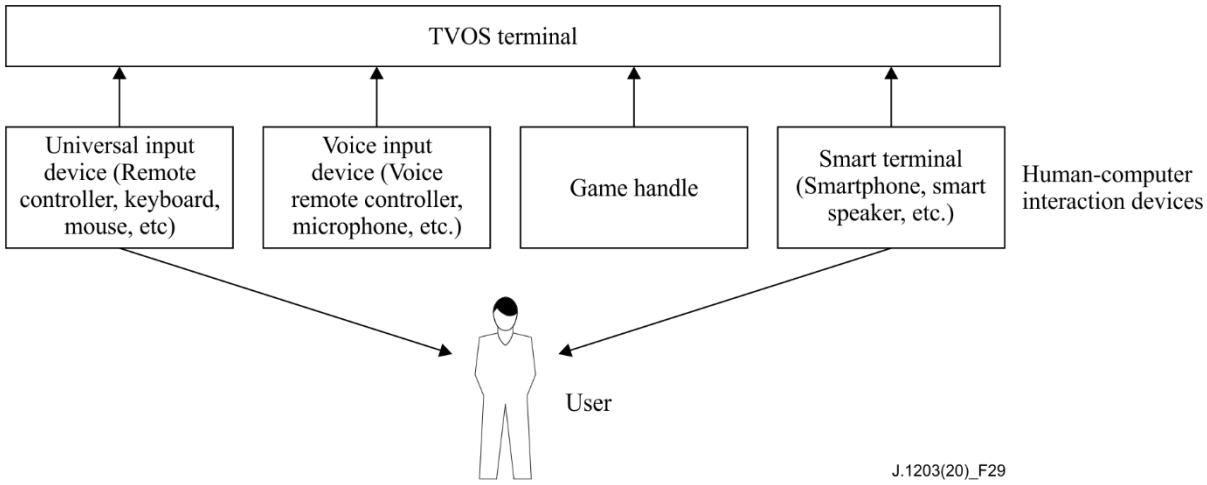

Figure 29 – Scenarios of HCI component

7.7.2 Component Implementation and Invocation Mode

The HCI component should be implemented according to the component model. Figure 30 shows the component invocation mode.

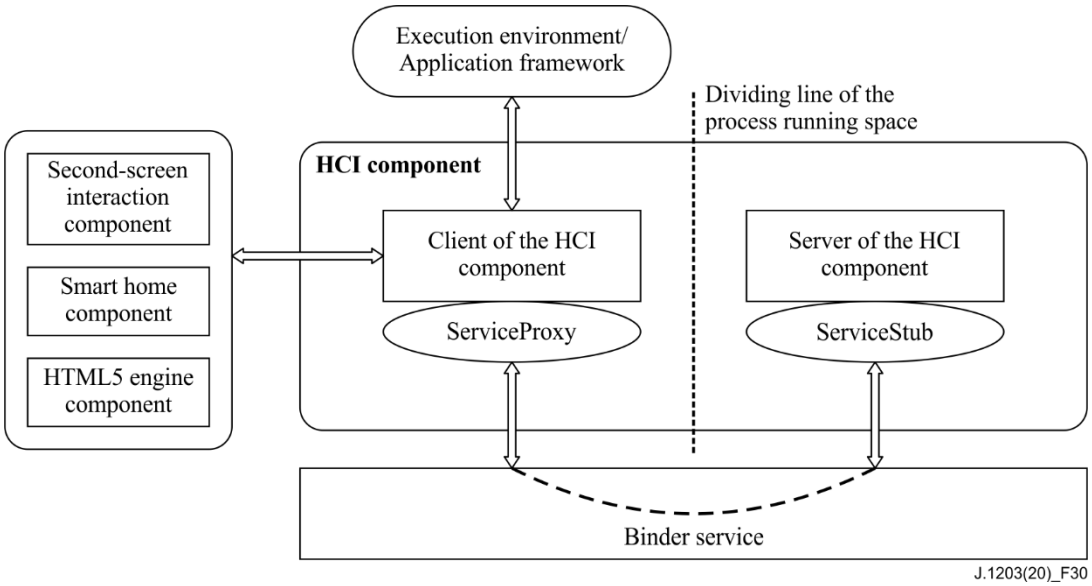

Figure 30 – Component implementation and invocation mode of the HCI component

7.7.3 Functional architecture and modules

The HCI component is a system service component and activated when the TVOS system is started. HCI server consists of key message processing, voice message processing, sensor message processing, game handle message processing and virtual input message processing sub-modules. Figure 31 shows the structure.

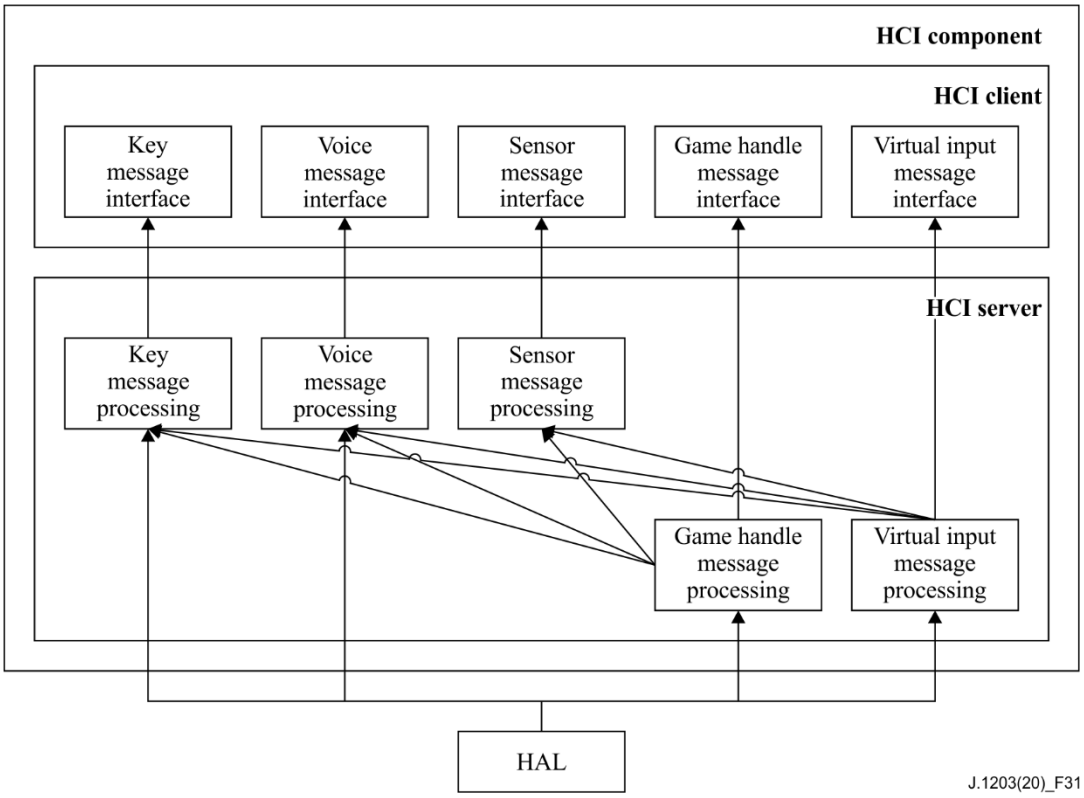

Figure 31 – HCI server module

The game handle message processing module is responsible for processing data from game handles. The messages from game handles includes key messages, voice messages and sensor messages.

The virtual input message processing module is responsible for processing data from remote smart terminals such as smart phone, smart speaker. Remote smart terminals can send different kinds of messages such as key message and voice message to TVOS terminal.

The key message processing module is responsible for processing remote controller, keyboard and mouse data from universal input devices, and also processing virtual keyboard and mouse data from the game handle processing module and the virtual input message processing module.

The voice message processing module is responsible for processing the voice data from voice input devices such as voice remote control and microphone. It supports the integration of different vendors' voice message recognition engines.

The sensor message processing module is responsible for processing virtual sensor data from the game handle processing module and the virtual input message module.

7.7.4 Interfaces

The HCI component provides key message interface, voice message interface, sensor message interface, game handle message interface and virtual input message interface for other software modules.

a) key message interface:

The HCI component provides the interface to the application framework layer to send key messages to the application.

b) voice message interface:

The HCI component provides the interface to the application framework layer to send voice messages to the application.

c) sensor message interface:

The HCI component provides the interface to the application framework layer to send sensor messages to the application.

d) game handle message interface:

The HCI component provides the interface to the application framework layer to send game handle messages to the application.

e) virtual input message interface:

The HCI component provides the interface to the application framework layer to send virtual input messages to the application, and also provides the interface to other components to inject virtual input messages.

7.7.5 Collaboration with other software modules

The HCI component supports collaboration with the second screen interaction component, smart home component, and HTML5 engine component, and provides invocation interfaces for different applications through the functional unit software modules at the application framework layer. Figure 32 shows the collaboration between the HCI component and other software modules.

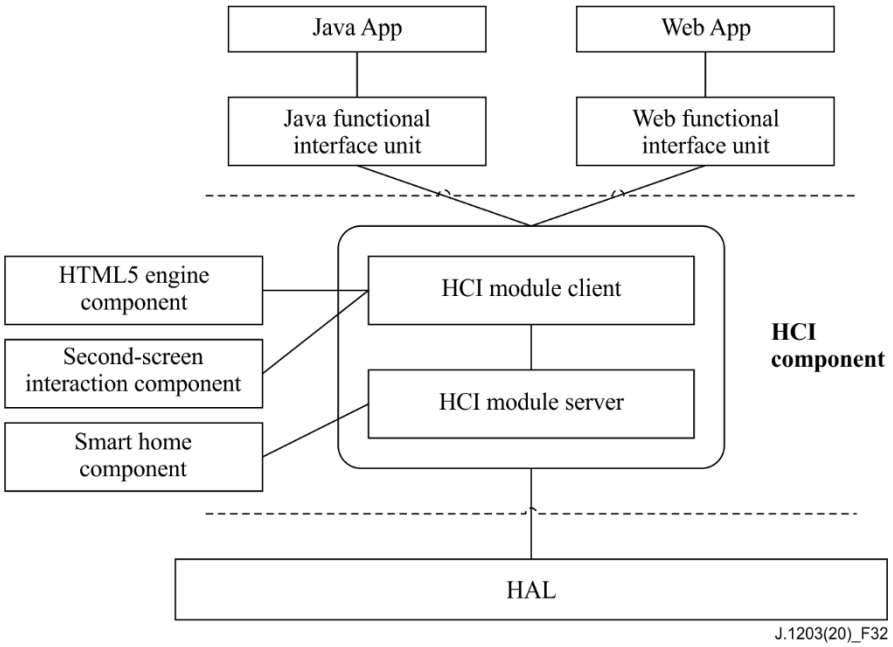

Figure 32 – Collaboration between the HCI component and other software modules

a) HCI component and Application:

HCI component interacts with Java APP through Java functional interface unit and interacts with Web APP through Web functional interface unit to provide all kinds of input messages.

b) HCI component and HTML5 engine component:

The HCI component provides keyboard and mouse events to HTML5 engine component.

c) HCI component and second-screen interaction component:

HCI component receives messages from second-screen interaction component, converts the messages to virtual input messages and sends them to the application.

d) HCI component and smart home component:

HCI component interconnects and interoperates with various smart terminals through interfaces provided by smart home component.

7.8 Second screen interaction component

7.8.1 Functions

The second screen interaction component should control content delivery, play, and present multimedia content including graphics, video, and audio among devices such as the mobile phone, tablet, and TV and support protocols for cross-device communication such as universal plug and play (UPnP) and digital living network alliance (DLNA), implementing cross-device screen control.

7.8.2 Component implementation and invocation mode

The second-screen interaction component should be implemented according to the component model. Figure 33 shows the component invocation mode.

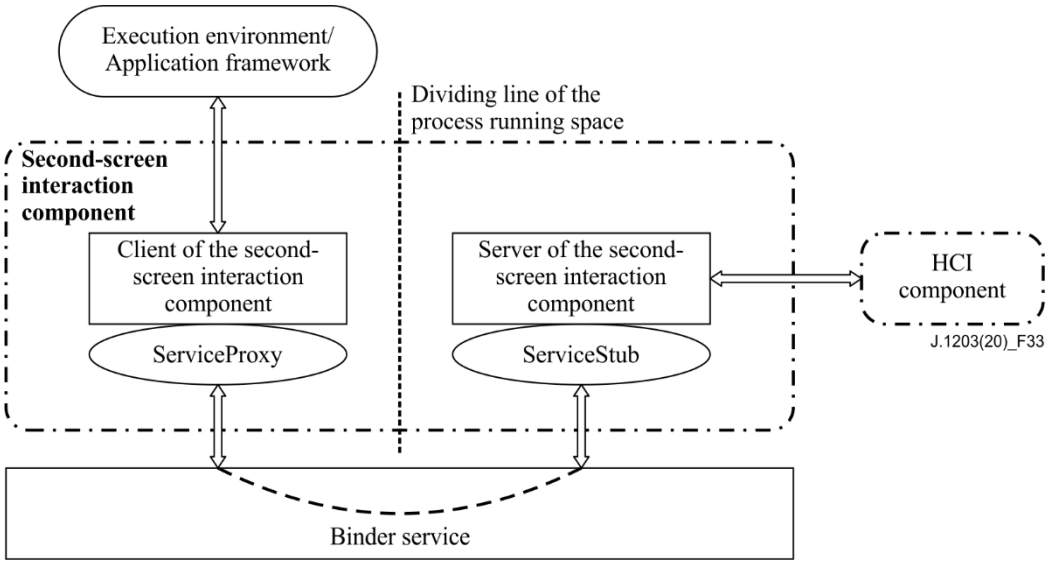

Figure 33 – Model of the second screen interaction component

7.8.3 Functional architecture and modules

The second-screen interaction component contains the device detection module, device connection module, device control module, and cross-screen playing and control module, as shown in Figure 34.

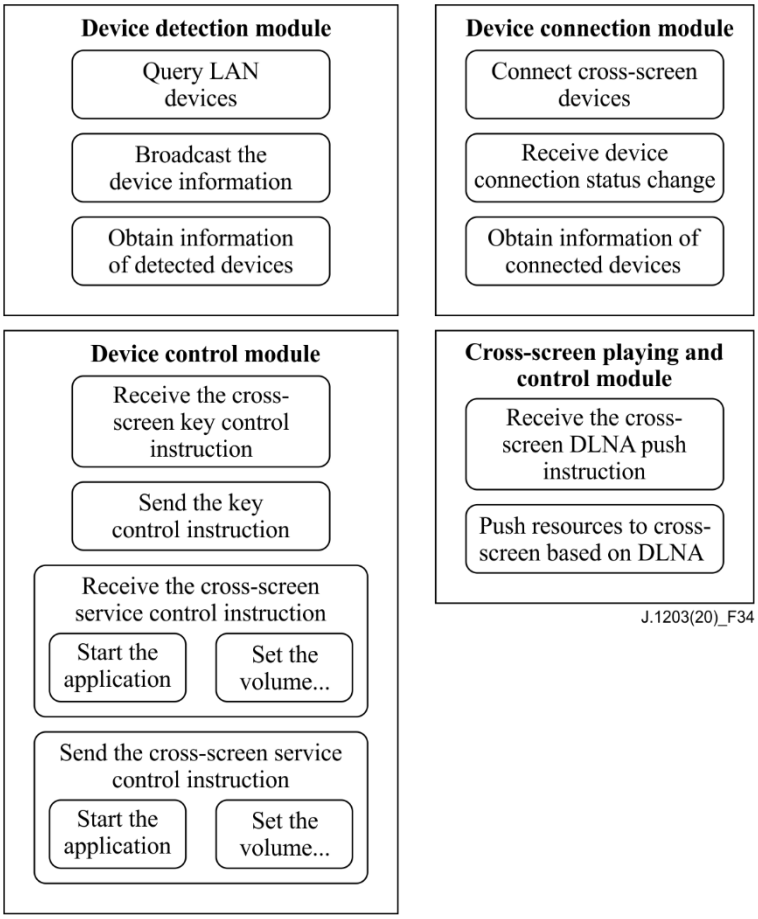

Figure 34 – Functional architecture of the second-screen interaction component

The device discovery module proactively discovers other devices or is discovered by other devices. This module stores information of the discovered devices, provides query interfaces for other software modules, and supports protocols such as UPnP.

The device connection module connects devices, receives connection status change messages of cross-screen devices, obtains capabilities of connected devices, and supports protocols such as UPnP.

The device control module implements cross-screen control functions, including receiving and sending cross-device key control instructions, and service control instructions to/from other devices. These service instructions can include application startup and audio volume configuration. Footnote: When presenting a TV programme and an application initiated by a second-screen device, a proper combination between the TV programme and the application should be respected.

The cross-screen playing and control module implements remote playing of local media files and locally playing of remote media files such as images, audio, and video. The cross-screen playing and control module should support protocols such as DLNA.

7.8.4 Interfaces

The second-screen interaction component provides the device discovery interface, device connection interface, cross-screen control interface, and cross-screen media playing interface. These interfaces are functional component interfaces.

7.8.5 Collaboration with other software modules

The device control module of the second-screen interaction component is capable of receiving cross-device key control instructions. This function requires invocation of the HCI component to

inject virtual keys. Add media component, application management component and system setting component.

7.9 Terminal control component

7.9.1 Functions

The terminal control component should parse and encapsulate packets for query, count, configure, monitor, and report the information and parameters of the IBB capable cable STB and TV, including restoring factory settings, configuring terminal restart, triggering software upgrade, and performing network diagnosis, such as TR069 as defined in Amendment 2 of [ITU-T H.770].

The terminal control component should implement the following functions:

a) Parse and encapsulate packets for query, count, configure, monitor, and report the information and parameters of the IBB capable cable STB and TV.

b) Query the software, hardware, network, application, and CA card information of terminals, and report the query result of related information in a timely manner.

c) Configure the control and management parameters of terminal functions, including the parameters that control the enabling, disabling, scheduled startup, and scheduled shutdown of related functions as well as parameters that configure network diagnosis triggering, terminal status alarm, and network.

d) Control terminal operations such as software update. In some cases, factory setting restoration, application cache clearing, and system restart can be supported.

e) Proactively report the system event information according to preset conditions and interval.

f) Register and deregister terminal control App and control its running state.

g) Verify the digital signature of information sent from the terminal control App. Only the App that passes the digital signature verification can be executed.

h) Digitally sign the report information.

7.9.2 Component implementation and invocation mode

The terminal control component should be implemented according to the component model. Figure 35 shows the component invocation mode.

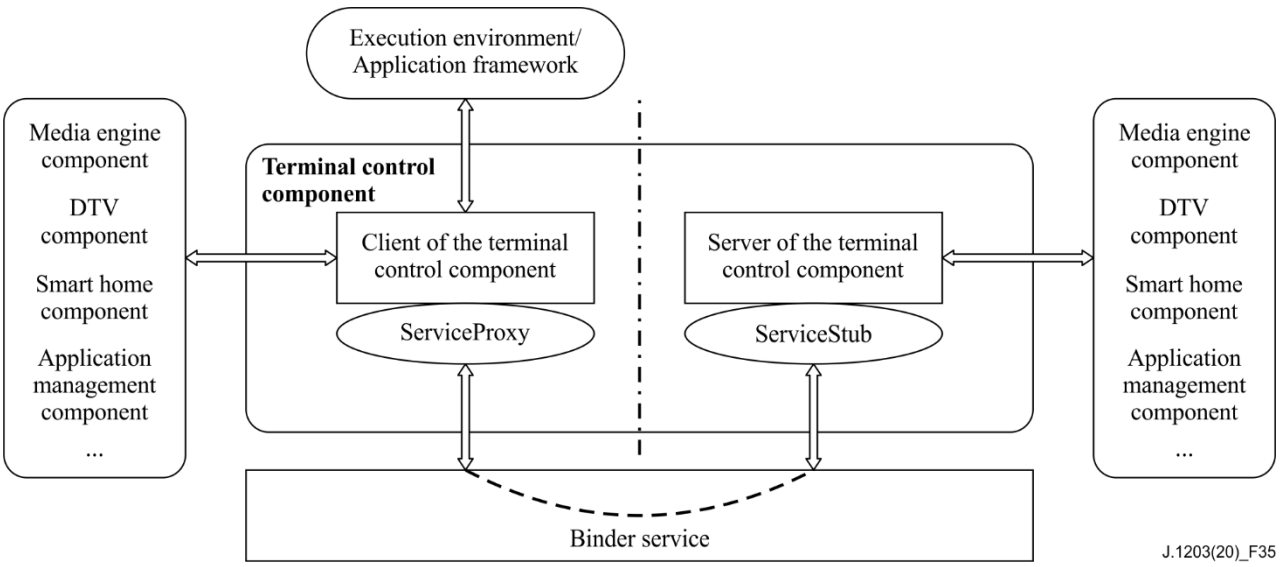

Figure 35 – Implementation and invocation mode of the terminal control component

7.9.3 Functional architecture and modules

The server of the terminal control component consists of the modules for reception and reporting, digital signature, terminal control protocols, collection and configuration, and control App management. Figure 36 shows the architecture of the server of the terminal control component.

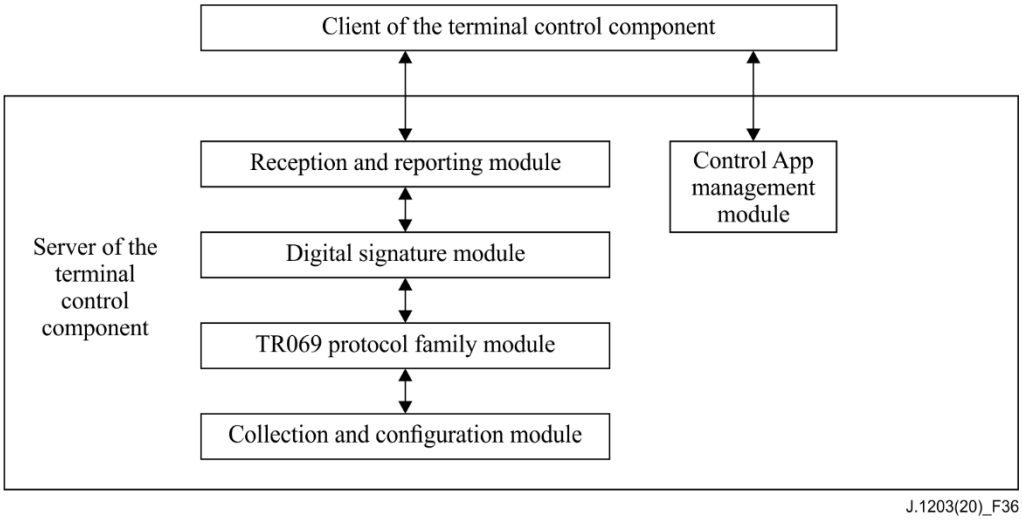

Figure 36 – Architecture of the terminal control component

The reception and reporting module cooperates with the terminal control App to receive the terminal control information and report the terminal status information to the headend.

The digital signature module verifies the digital signature of the information sent from the terminal control App and digitally signs the information to be reported to the headend.

The terminal control protocol module handles various kinds of communications for terminal management.

The collection and configuration module cooperates with other software modules to query and configure terminal status parameters.

The control App management module implements the registration and deregistration mechanism of the terminal control App.

7.9.4 Interfaces

The terminal control component should provide the interfaces for the terminal control App to transfer the control commands and reported information as well as the registration and deregistration interfaces through the client. These interfaces are functional component interfaces.

7.9.5 Collaboration with other software modules

The terminal control component should collaborate with all other components to collect data such as the media engine component, DTV component, smart home component, and application management component. Figure 37 shows the collaborative relationship between the terminal control component and other components.

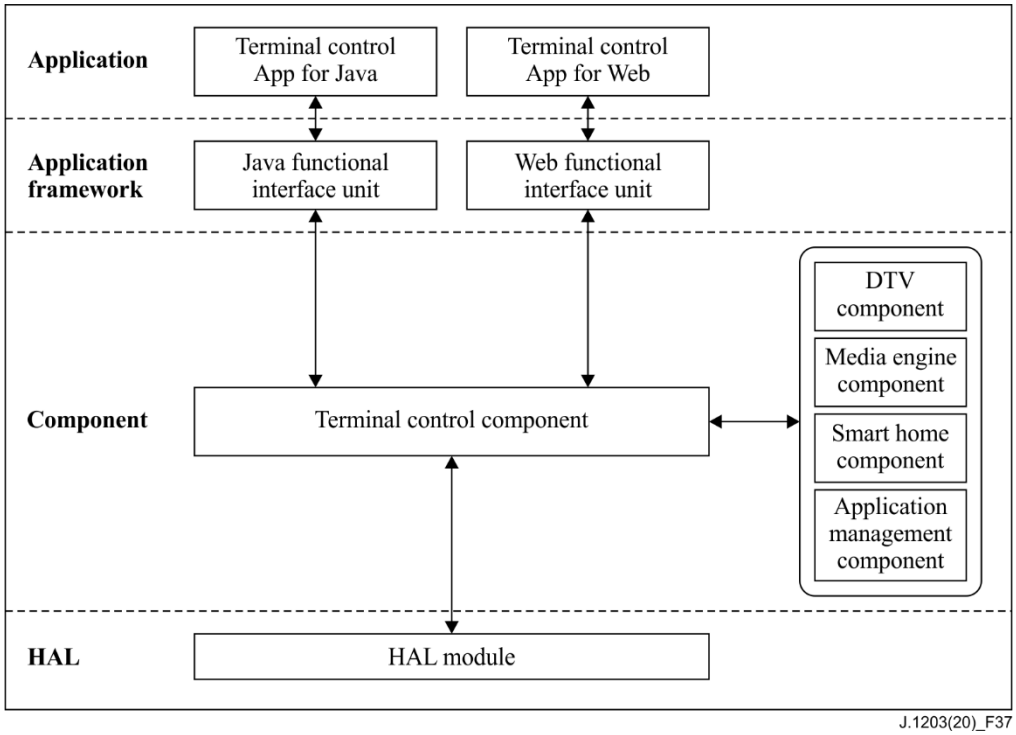

Figure 37 – Collaborative relationship between the terminal control component and other components

7.10 Data collection component

7.10.1 Functions

The data collection component collects and reports information and data related to the advanced TVOS terminal capability and user behaviour.

NOTE – Collection of user's private data must be compliant to local regulations with consideration of the operator's need.

7.10.2 Component implementation and invocation mode

The data collection component should be implemented according to the component model. Figure 38 shows the component invocation mode.

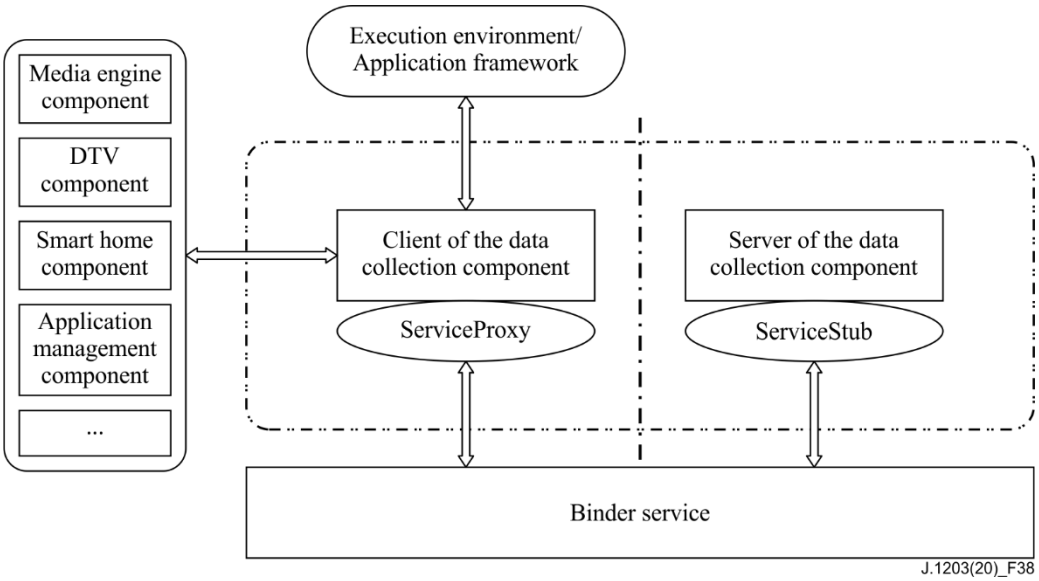

Figure 38 – Implementation and invocation mode of the data collection module

7.10.3 Functional architecture and modules

The data collection server consists of the reception and reporting, data collection, and collection App management modules. Figure 39 shows the architecture of the data collection component.

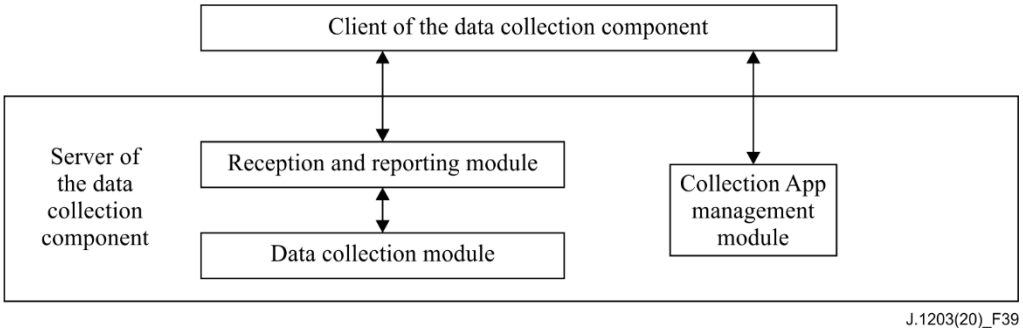

Figure 39 – Architecture of the data collection component

The reception and reporting module cooperates with the data collection App to receive data collection instructions and report the collected data and information.

The data collection module cooperates with other software modules to collect terminal service data.

The collection App control module implements the registration and deregistration mechanism on the data collection App.

7.10.4 Interfaces

The data collection component should provide information delivery interfaces as well as registration and deregistration interfaces through the client to the data collection App. These interfaces are functional component interfaces.

7.10.5 Collaboration with other software modules

The data collection component should collaborate with all other components such as the media engine component, DTV component, smart home component, and application management component. Figure 40 shows the collaborative relationship between the data collection component and other components.

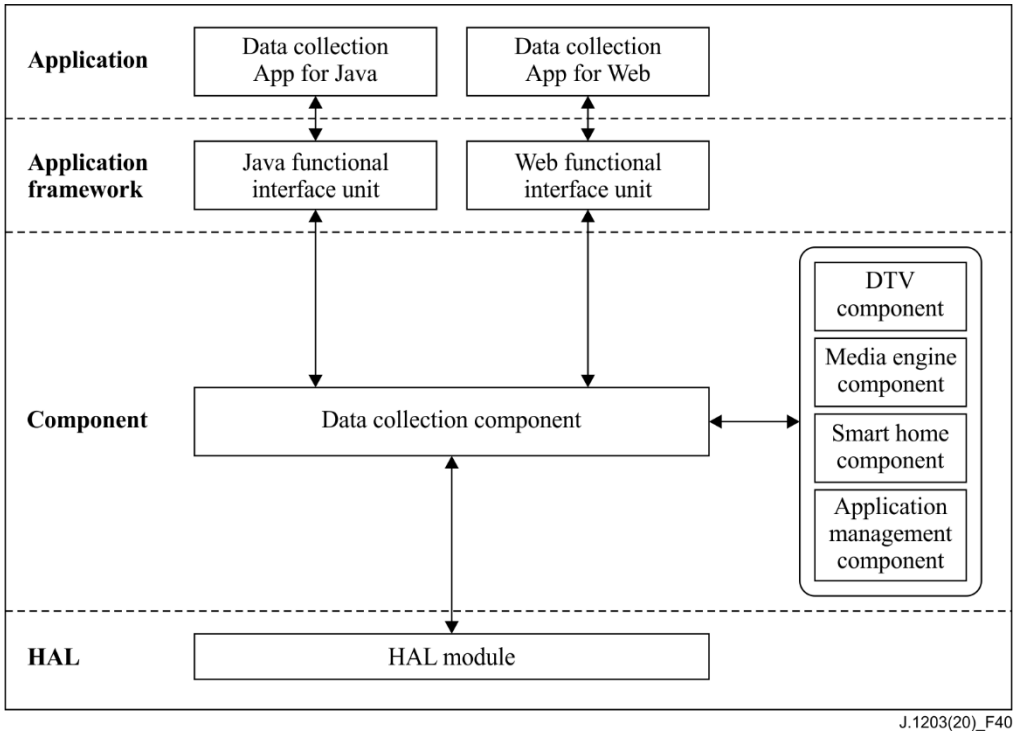

Figure 40 – Collaborative relationship between the data collection component and other components

7.11 Broadcast information service component

7.11.1 Functions

The broadcast information service component should implement the following functions:

– Cooperate with the DTV component to monitor, receive, and process broadcast information services and support services such as emergency broadcast, information service, and notification from an STB.

– Monitor the relevant tables and filter the section data in the broadcast information service through the DTV component.

– Obtain the CA user identifier corresponding to the terminal through the DCAS component and implement precise reception of information such as the terminal emergency broadcast and advertisements on the basis of the identifier.

7.11.2 Component implementation and invocation mode

The broadcast information service component should be implemented according to the component model. Figure 41 shows a model of the broadcast information service component.

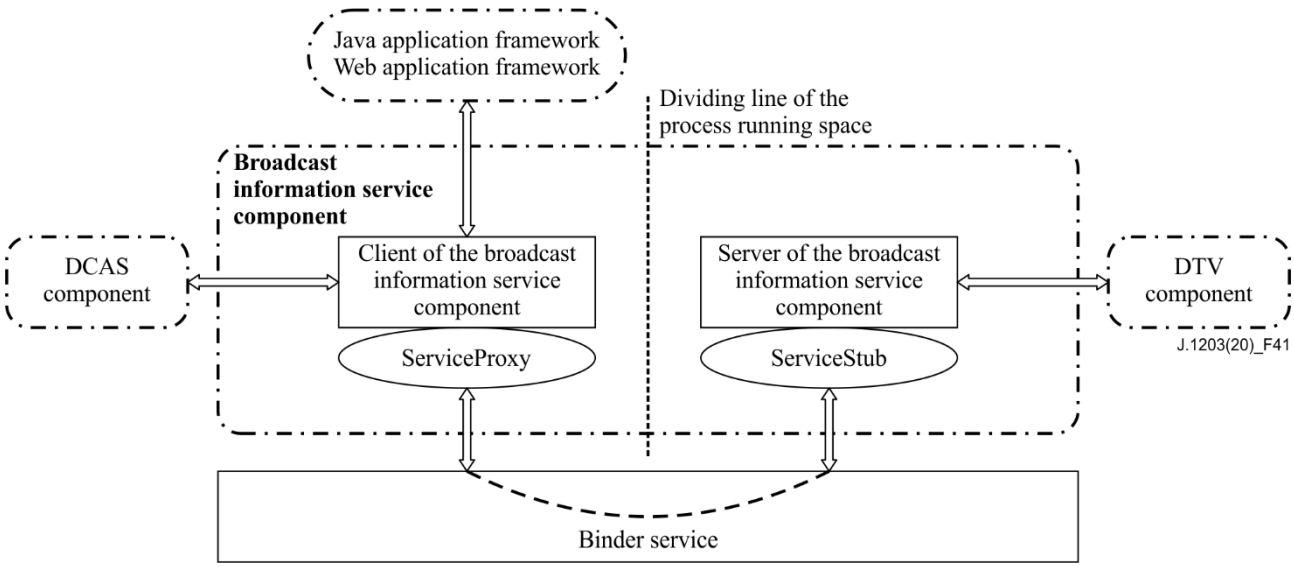

Figure 41 – Model of the broadcast information service component

7.11.3 Functional architecture and modules

The broadcast information service component should include the emergency broadcast signalling information monitoring and forwarding, information service data reception, advertisement content update and notification from an STB update functional modules. Figure 42 shows the structure.

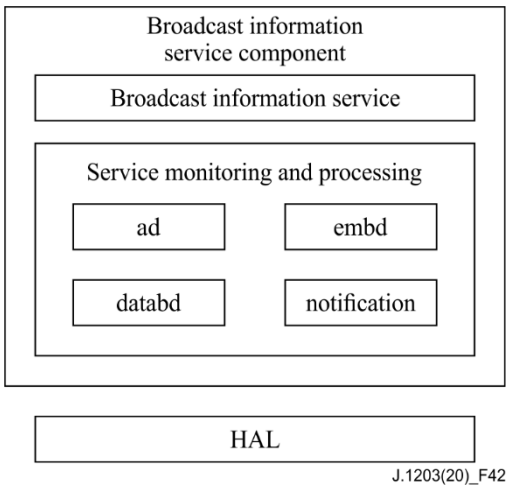

Figure 42 – Internal logical architecture of the broadcast information service component

The emergency broadcast signalling information monitoring and forwarding functional module (embd) monitors, receives, and processes emergency broadcast signalling information and cooperates with other software modules according to requirements of the received signalling information such as switching the terminal to the required emergency broadcast channel.

The information service data reception functional module (databd) receives and stores the information service data transferred in the dedicated data broadcast channel.

The advertisement content functional module (ad) updates the advertisement pictures and data, such as the startup screen, startup advertisements on the main menu advertising space and programme browsing advertising space, and real-time advertisement update of programme bar information and volume bar images.

The notification functional module (notification) monitors, receives and parses the notification signalling information and assists other software modules in displaying and updating the notificaton information on the terminal UI, such as the basic CA module and pay-per-view (PPV) prompt information.

7.11.4 Interfaces

The broadcast information service component should be accessed through the client of the broadcast information service component. Table 1 lists the access interfaces provided by the client of the broadcast information service component.

Table 1 – Interfaces of the broadcast information service component

No.	Functional Unit	Description
1	Emergency broadcast	Emergency broadcast interface for notifying emergency broadcast event to the relevant application and emergency broadcast information acquisition.
2	Information service	General information service interface for information data acquisition.
3	Advertisement	Advertisement data reception, status query and advertisement data acquisition interface
4	Notification	Notification status and data acquisition interface

7.11.5 Relationship with other components

The broadcast information service component needs to interact with the DTV component and DCAS component. Figure 43 shows the relationship between the broadcast information service component and other components.

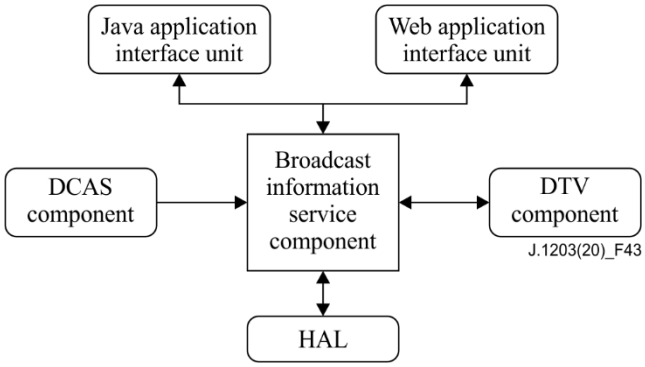

Figure 43 – Relationship between the broadcast information service component and other components

7.12 ATV component

7.12.1 Functions

The ATV component should implement the programme search, frequency channel management, channel management and TV-related configuration parameter management functions, and provide interfaces and capability support for related applications.

NOTE – Analogue cable services may not be needed in some countries.

– Programme search includes automatic search and manual search.
– Frequency channel management includes frequency channel switching and storage.
– Channel management includes channel switching and information obtaining.
– TV input source management includes AV input, component input, VGA input and HDMI input ports management.
– TV-related configuration parameter management includes configuration and storage of basic image and sound parameters.

7.12.2 Component implementation and invocation mode

The ATV component should be implemented according to the component model. Figure 44 shows the component invocation mode of the ATV component.

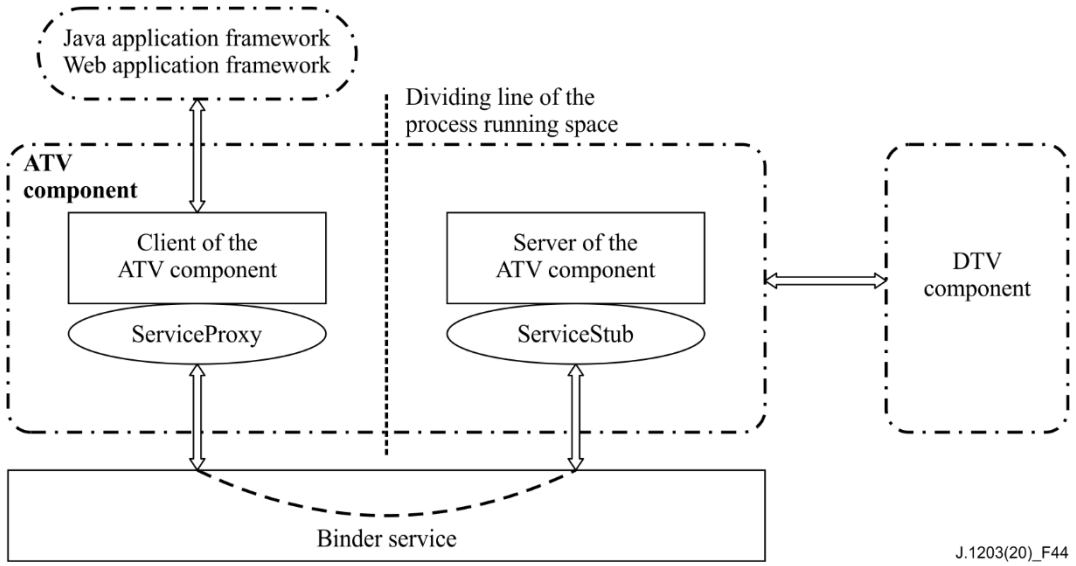

Figure 44 – Component implementation and invocation mode of the ATV component

7.12.3 Functional architecture and modules

The ATV component consists of the channel manager, source manager, TV setting, and data manager modules. Figure 45 shows the architecture of the ATV component.

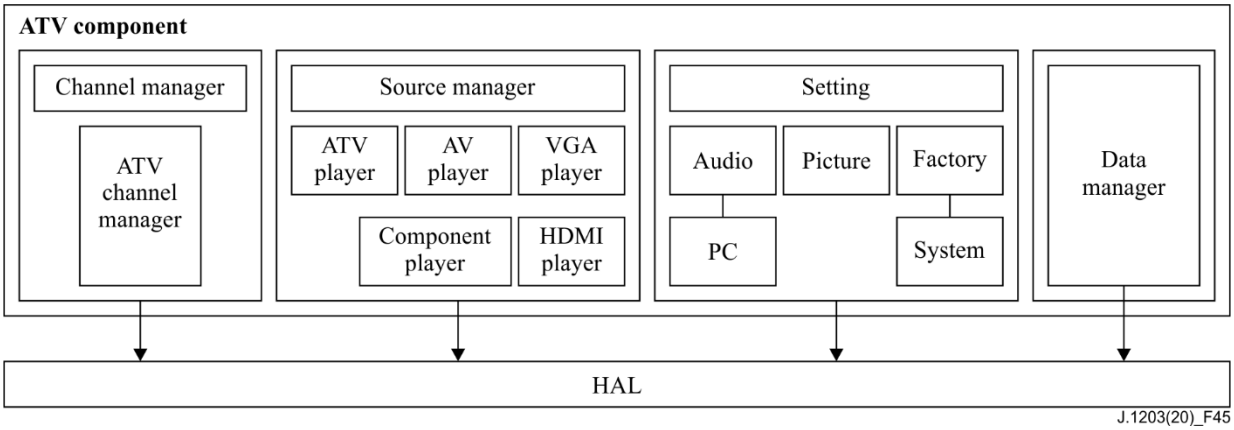

Figure 45 – Logical architecture of the ATV component

The channel manager implements ATV automatic or manual channel search and management.

The source manager switches sources such as the ATV channel, AV, VGA, component, and HDMI sources as requested by other components or application and responds source state query.

The TV setting module configures and obtains TV-related parameters such as the audio, picture, factory and system parameters. Audio includes sound-related settings such as the volume and sound mode. Picture includes picture-related settings such as the brightness, contrast, definition and saturation. Factory includes settings related to the factory mode. System includes system-related settings such as screen settings and source settings.

The data manager stores information under the control of this component. The stored data includes audio, picture, factory, system, source and ATV related parameters.

7.12.4 Interfaces

The ATV component provides interfaces through the client of the ATV component. The provided interfaces are as follows:

a) The source control interface provides channel switching and status query functions.

b) The channel control interface provides frequency channel search and switching functions.

c) The TV setting interface provides picture, audio, system and factory configuration functions.

7.12.5 Collaboration with other software modules

The ATV component also cooperates with the DTV component to implement TV-related services. Figure 46 shows the collaborative relationship between the ATV component and other software modules.

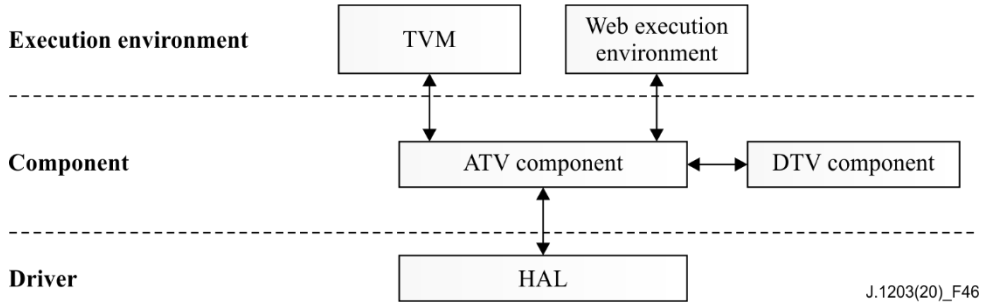

Figure 46 – Relationship between the ATV component and other software modules

The following describes the relationship between the ATV component and other software modules:

The ATV component invokes the player interfaces of the DTV component to switch between the DTV channels and other channels.

8 Application execution environment

8.1 TVM

8.1.1 Functions

The TVM execution environment should implement the following functions:

– The interpretation and running environment for Java applications and the application framework-layer functional interface instances invoked by Java applications.

– Loading and running of Java applications.

– Process isolation for Java applications by using of different TVM instance for each Java application.
– Cooperation with the application management component to manage the life cycle of Java applications, including starting, pausing, resuming, and restarting Java applications.
– Managing the access rights of Java application resources.

The TVM should be built with Java virtual machine and application model converter. The application model converter is used to support applications designed to run on third party virtual machines.

8.1.2 Architecture and implementation mechanism

TVM Runtime consists of the application model converter, byte code converter, Java ME support module, and Java virtual machine (VM). Figure 47 shows the architecture and implementation mechanism of TVM Runtime.

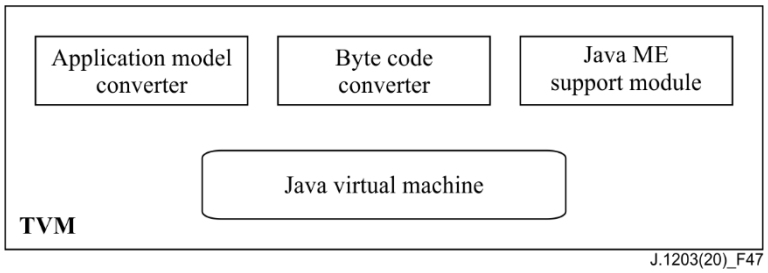

Figure 47 – TVM architecture

The application model converter converts third-party Java application package into Java VM application package. The byte code converter converts the third-party Java byte code into Java VM byte code.

The Java ME support module configures the related running environment based on the Java Specification Requests (JSR) specification, provides support for the running of Java ME Xlet/MIDlet applications, and supports the CDC 1.1.2 [b-CDC], FP 1.1.2 [b-FP], PBP 1.1.2 [b-PBP], and MIDP 2.0 [b-MIDP].

8.2 Web Runtime

8.2.1 Functions

Web Runtime should implement the following functions:

– Cooperation with the HTML5 engine to manage the life cycle of web applications, including loading, starting, suspending, and destroying web applications.
– Managing the access rights of web application resources, including checking and requesting the rights of web applications.
– Application security management functions such as web application isolation.
– Managing web application policies, including application resource allocation and its lifecycle.

8.2.2 Architecture and implementation mechanism

Web Runtime closely cooperates with the HTML5 engine component to create and provide the running execution environment for web applications and manage the running, rights, security, and policy of web applications.

Figure 48 shows the architecture and implementation of Web Runtime and the HTML5 engine component.

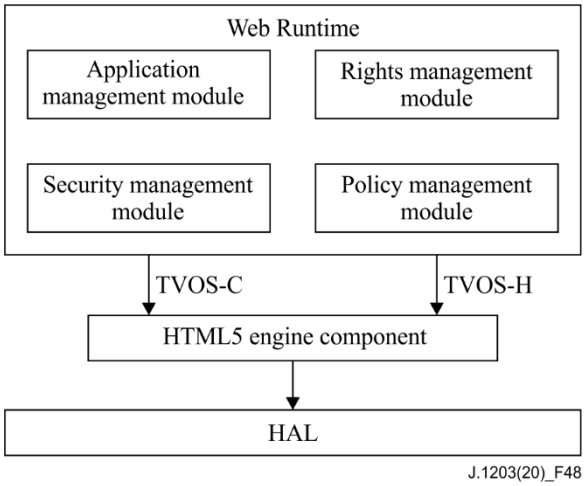

Figure 48 – Architecture and implementation of Web Runtime and the HTML5 engine component

Web Runtime consists of the application running management module, rights management module, security management module, and policy management module.

The application management module creates instances of the server of the HTML5 engine component which is shown in Figure 18, forms the basic environment of web applications, loads web applications to the basic environment, starts web applications, and manages the life cycle of web applications.

The rights management module manages the access rights of web application resources when web applications are running.

The security management module implements security management including process isolation and data isolation.

The policy management module implements policy management for the running modes of web applications, including exclusively occupying processes for different application configurations or sharing the HTML5 engine component of the process.

9 Application framework

9.1 Architecture of the Java application framework

The Java application framework consists of the TVOS Java application programming interface units and extended functional interface units.

9.1.2 TVOS Java application programming interface units

9.1.2.1 Functions

The TVOS Java application programming interface units implement interfaces of various functional components and modules, provide Java application programming interfaces, and assist applications in implementing DTV services such as EPG, channel list, and TV programme playing.

The TVOS Java application programming interface units include:

– DAVIC (Digital Audio Video Council) unit;

– Unidirectional broadcast network access unit;
– Bidirectional broadband access unit;
– HCI unit;
– Media processing unit;
– Message management unit;
– DCAS functional interface unit;
– Terminal control data collection unit;
– Broadcast information service unit.

9.1.2.2 Main functional interface units

Main functional interface units are as follows:

a) DAVIC functional interface unit:

This unit cooperates with the DTV component to process basic objects and exceptions related to DTV services to support the DTV-related applications. This unit complies with DAVIC 1.4.1 [b-DAVIC], including MPEG, DVB.

b) Unidirectional broadcast functional interface unit:

This unit cooperates with the DTV component to access the unidirectional cable DTV broadcast network, including controlling the frequency, modulation mode, and symbol rate as well as obtaining information such as the signal strength and quality. This unit also cooperates with the DTV component to implement basic DTV functions such as programme search, programme guide, and information search, supporting the running of DTV-related applications.

c) Bidirectional broadband network functional interface unit:

This unit cooperates with the network service to implement bidirectional broadband network access, including connection management.

d) HCI functional interface unit:

This unit cooperates with the HCI component to implement the user interface. The user instructions can be obtained by input devices such as the remote control, mouse, keyboard, and front-panel key as a form of key messages. The information to the user is shown to the front panel or display screen.

e) Media playing functional interface unit:

This unit cooperates with the media component to implement media playing functions, including playing, control, language selection, event processing, and exception processing.

f) Message management functional interface unit:

This unit is the TVOS message repository for the applications dealing with message event, message event listener, and message manager. This unit manages and distributes messages and assists applications in message acquisition.

g) DCAS functional interface unit:

This unit cooperates with the DCAS component to implement DCAS application management, ECM/EMM data processing, and DCAS data interaction, assisting DTV applications in playing encrypted DCAS programmes. Some interfaces may have limitation for access by the application.

h) Terminal control and data collection unit:

This unit cooperates with the terminal control component and other components for some applications to interact with the terminal control and data collection components.

i) Broadcast message service unit:

This unit provides the monitoring, reception, and processing functional interfaces related to broadcast information services.

9.1.2.3 Relationship of the interfaces and functional components

The framework-layer TVOS Java application programming interface depends on function support from components at the component layer. The JNI is used to work with appropriate functional components at the component layer to implement the TVOS Java application programming interface. Figure 49 shows the interface relationship.

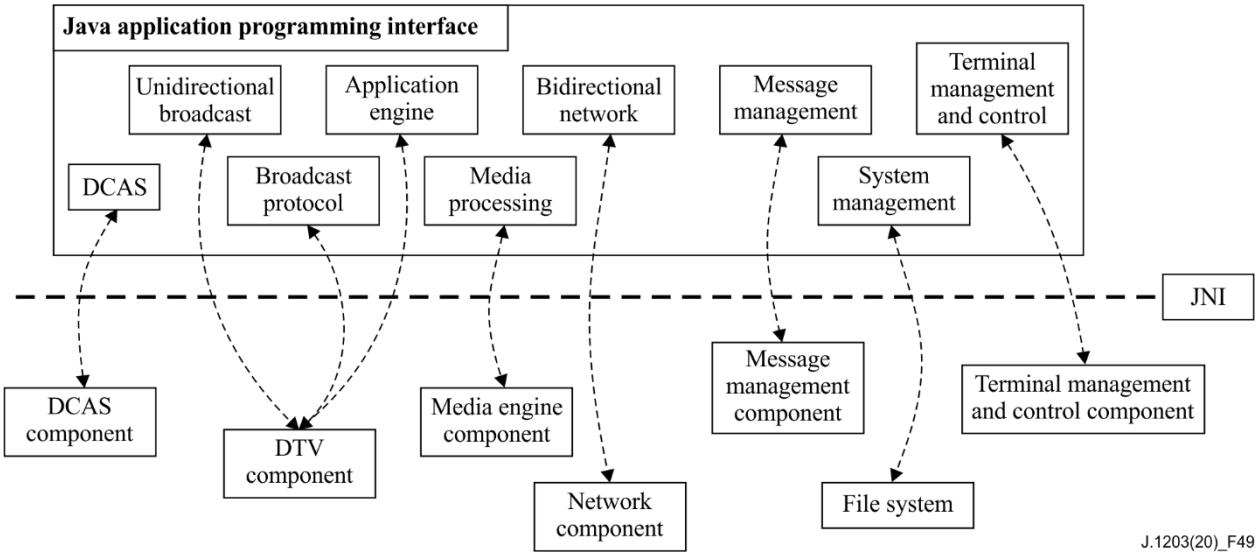

Figure 49 – TVOS Java application programming interface of the framework layer

9.1.3 Extended functional interface units

The TVOS Java application framework layer interface may support third-party Java APIs by extended functional interface units.

9.2 Web application framework

9.2.1 Architecture of the Web application framework

The web application framework consists of the HTML5 functional interface units and TVOS Web application programming interface units.

9.2.2 HTML5 functional interface unit

The HTML5 functional interface units support different types of interfaces including HTML5, CSS, ECMA Script, and DOM.

The HTML5 interface unit should support interfaces defined in the HTML5 standard.

The CSS interface unit should support interfaces defined in the CSS3 standard.

The ECMA Script interface unit should support interfaces defined in the ECMA-262 standard.

The DOM interface unit should support interfaces defined in the DOM2 standard.

9.2.3 TVOS Web application programming interface units

9.2.3.1 TVOS Web application programming interface functions

The TVOS Web application programming interface units implement ECMA Script interface encapsulation for interfaces of various functional components and modules, provide web applications with functions as ECMA Script object, and assist applications in implementing DTV services such as EPG, channel list, and TV programme playing.

The TVOS Web application programming interface units include:

– Unidirectional broadcast network access unit;

– Bidirectional broadband access unit;

– HCI unit;

– Media processing unit;

– Message management unit;

– Application management unit;

– DCAS unit;

– Broadcast information service unit.

9.2.3.2 Functional interface units

The TVOS Web application programming interfaces include the following units:

a) Unidirectional broadcast network access unit, which implements functional modules related to unidirectional broadcast network access such as DVB tuning and demodulation and provides the channel management, information search, and programme guide management functions.

b) Broadcast protocol processing unit, which implements broadcast protocol processing.

c) Bidirectional broadband network access unit, which defines functional modules related to bidirectional broadband network access control.

d) HCI unit, which implements display and control functions such as device input control and panel.

e) Media processing unit, which processes multimedia files such as audio, video, and images.

f) Message management unit, which captures and processes system messages and application-layer messages.

g) Application management unit, which manages and controls application download and running.

h) DCAS unit, which provides functions related to DCAS applications, such as registration, filter configuration, and communication with the TApp.

i) Broadcast message service unit, which provides the monitoring, reception, and processing functional interfaces related to broadcast information services.

9.2.3.3 Relationship of the interfaces and functional components

The TVOS Web application programming interface units implement the client to work with servers of appropriate functional components.

For example, the DvbTune object needs to interact with the DTV component and the media player object needs to interact with the media engine component. Figure 50 shows the relationship between TVOS Web application programming interface and other component interfaces.

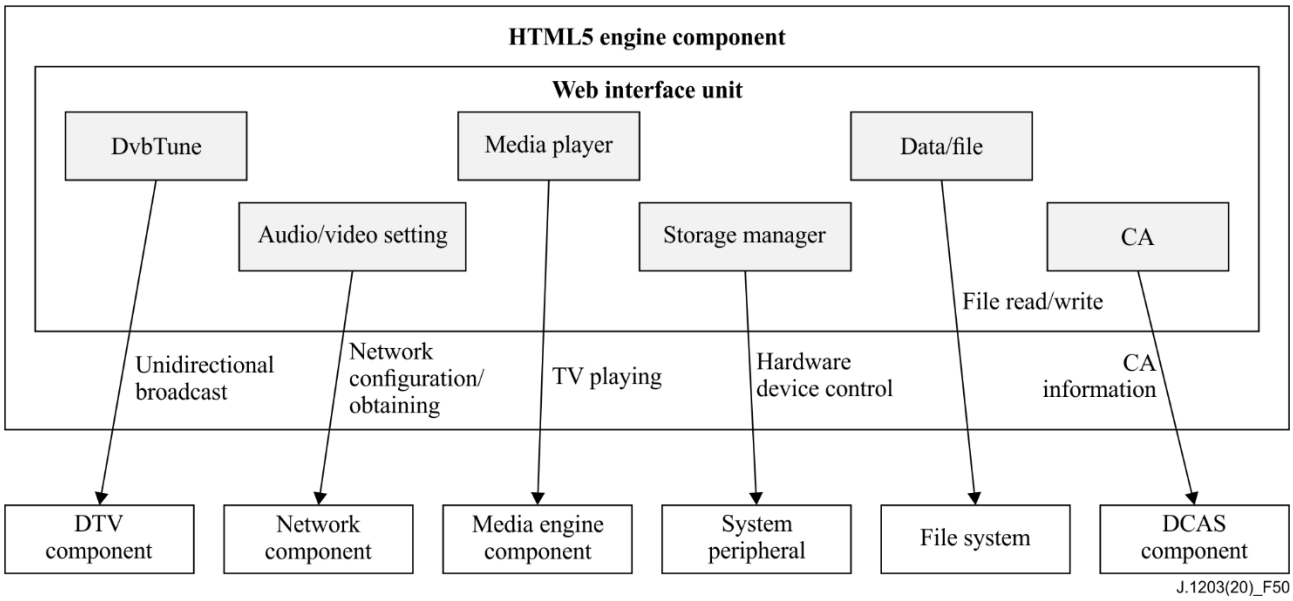

Figure 50 – Relationship between TVOS Web application programming interface and other component interfaces

Bibliography

[b-CDC] Java Community Process (2006), JSR218, *Connected Device Configuration (CDC), version 1.1.2.*

[b-DAVIC] DAVIC, *Digital Audio Video Council 1.4.1.*

[b-FP] Java Community Process (2006), JSR219, *Foundation Profile 1.1.2.*

[b-GY/T 303.1] GY/T 303.1-2016, *SmartTV operating system – Part 1: Function and architecture.*

[b-MIDP] Java Community Process (2006), JSR118, *Mobile Information Device Profile 2.0.*

[b-PBP] Java Community Process (2006), JSR217, *Personal Basis Profile 1.1.2.*

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	**Cable networks and transmission of television, sound programme and other multimedia signals**
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2020