
Design Document

Media Server

Version 1.0

Media Server Design Document

Objective

Document Scope

The information provided in this document specifies the design details of Media Server. For complete scope of Media Server please see the Project Proposal.

1. Abbreviations

Following are the abbreviations that have been used in the document:

- PEG** Parsing Expression Grammar
- API** Application Programming interface

Table of Contents

1.	Abbreviations	iii
2.	Introduction	1
3.	Media Server Architecture	1
3.1.	Overall Media Server Architecture Diagram	1
3.2.	Sip Call Processing State Machine	1
4.	Media Handling	2
4.1.	Media Processing Architecture	2
4.2.	Component Chaining	3
4.3.	Components Signals	3
5.	State Machines	4
6.	Design of MSCML Play State Machine	4
6.1.	MSCML Play State Machine Inputs	5
6.2.	MSCML Play State Machine Input Parameters	5
6.3.	MSCML Play State Machine Outputs Parameters.....	7
6.4.	MSCML Play Output Parameters	7
6.5.	States of Play State Machine	8
6.6.	State Machine MSCMLPlayContext	9
6.7.	Play State Machine Diagram.....	10
6.8.	Pseudo-Code.....	10
6.9.	Functionalities of States of the Play State Machine	11
6.10.	Play State Machine Diagram PsuedoCode.....	15
7.	MSCML Play Collect State Machine	18
7.1.	MSCML Play Collect State Machine Inputs 1. SM_STOP.....	18
7.2.	MSCML Play Collect State Machine Outputs 1. SM_STOPPED	19
7.3.	Play Collect State Machine Diagram.....	21
7.4.	MSCML Play Collect State Machine Context Elements.....	21
7.5.	Play Collect State Machine States.....	24
7.6.	MSCML Play Collect State Machine pseudo Code	24
7.7.	Digit Regular expression	29
8.	Play Record	35
8.1.	Inputs	35
8.2.	Outputs	35
8.3.	Play Record State Machine Diagram.....	36
8.4.	Context	36
8.5.	Transformation	39
9.	Reference	45

2. Introduction

Media Server is an important component in IMS architecture. Media Server is responsible for handling and delivery of media services to end user. It provides all media related functions i.e. media mixing, Transcoding, and playing of tones and announcements. Media Server is further decomposed into two separate *functional* elements i.e. MRFC and MRFP.

MRFC is responsible for controlling all session related signaling, it establishes multi party sessions, and deals with all kinds of announcement services and transcoding services. MRFP on the other hand, ensures media processing activities

It further controls all routing information interrogations from CSF and AS. MRFP is responsible of mixing/processing of all media streams and ensures that data is transferred in a correct format.

The following deployment diagram explains the context of Media Server communication with other IMS components in IMS architecture.

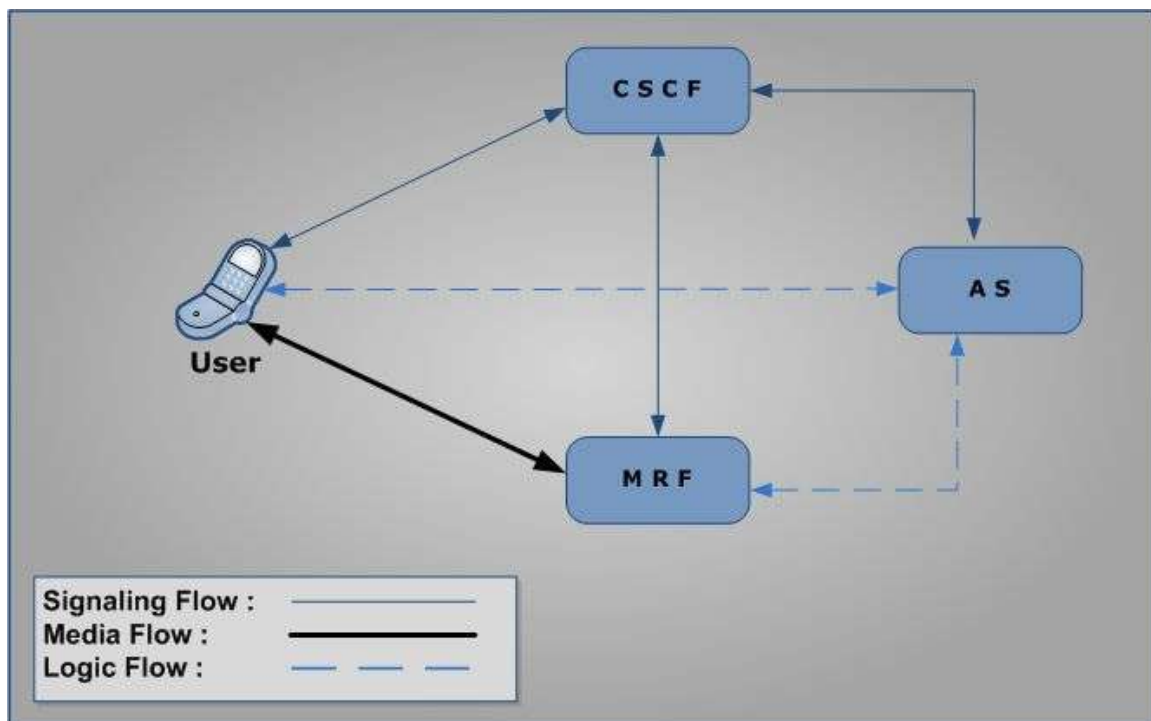


Figure 1 –Media Server in IMS Architecture

3. Media Server Architecture

3.1. Overall Media Server Architecture Diagram

3.2. Sip Call Processing State Machine

Media Server Design Document

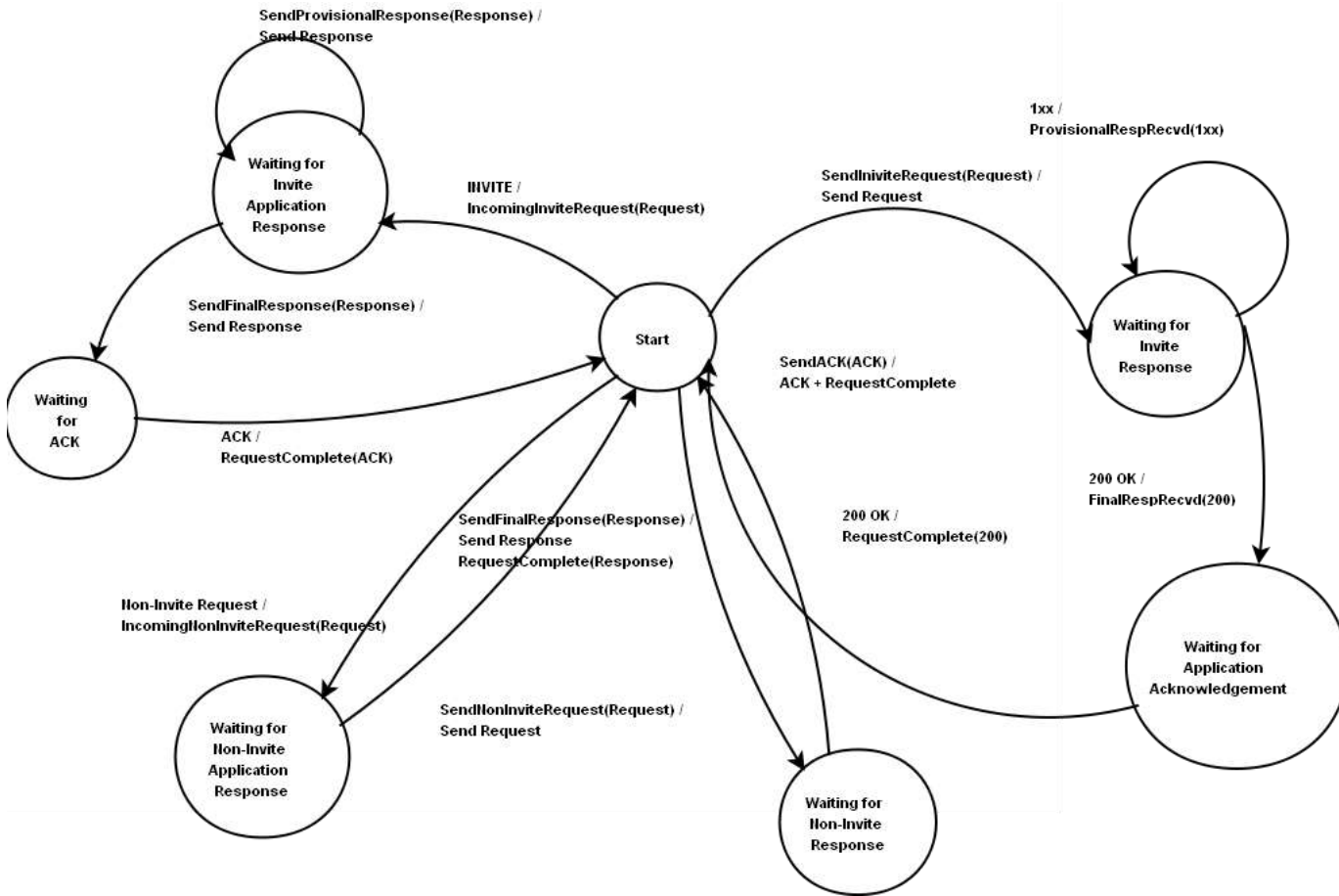


Figure 3 - Call Leg

4. Media Handling

One of the main tasks of the media server is to process media. Media processing may include recording, playback, mixing, encoding, decoding etc. Media Server requires an efficient and flexible framework to process audio packets.

4.1. Media Processing Architecture

Individual tasks are divided into distinct block called components. The components are linked together in a chain and configured once. These components can then work independently according to the configured options.

Each component has a defined input and output formats. The task of the component is to convert the data from the input format to its output format. The output data then becomes the source of another component. There are three types of components.

1. Source Component:
This type of component only generates an output, and do not have input. Thus they act as a source of data for a chain e.g. File-Reader and UDP-Receiver.
2. Filter Component:
This type of component converts the data from one format to another. Most of the components are filter components e.g. Encoder and Decoder.

3. Sink Components: This type of components only have an input, but no output. Thus they act as a sink of data in a chain e.g. File-Writer and UDP-Transmitter.

4.2. Component Chaining

Components are linked with each other to form a chain. The components in the chain depend on the functionality to be provided by the chain. Any type of chain is possible, as long as the output of one component is compatible with the input of the next. The chain represents a unidirectional flow of data. Following are typical chains

1. Transmitter Chain: This chain reads a file and eventually sends it on the network. Thus it provides audio playback functionality. (media transmitter chain diagram)
2. Receiver Chain: This chain receives an audio stream and eventually writes it in the file. Thus it provides an audio recording functionality. (media receiver chain diagram)

4.3. Components Signals

In addition to process data, each component generates certain signals in case of certain occasional events. Application can provide signal handling functions, which process these signals. Example of these signal include, receiving of DTMF packet in a media stream.

Following components are needed by Media Server

Receivers:

1. UDP-Receiver: It receives packets on a UDP port.
 - Input: None
 - Output: UDP Packet
2. RTP-Depacketizer: It de-encapsulates RTP header.
 - Input: RTP Packet
 - Output: Codec Data
3. Decoder: It decodes encoded audio data according to a certain codec. Each codec will have its own decoder
 - Input: Codec Data
 - Output: Raw Data
4. File-Writer: it writes data in a file
 - Input: Generic Data
 - Output: None

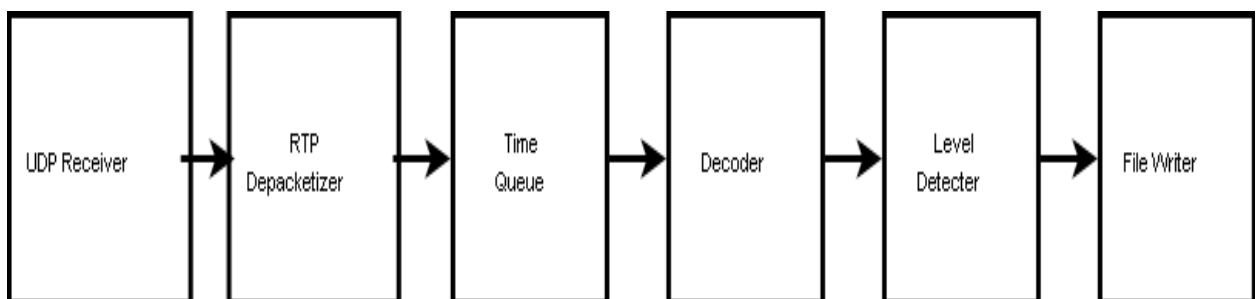


Figure 4 - Media Receiver Chain

Transmitters:

Media Server Design Document

1. UDP-Transmitter: It sends packets from a UDP port
 - Input: Generic Data
 - Output: None
2. RTP-Packetizer: It encapsulates RTP header
 - Input: Codec Data
 - Output: RTP Packet
3. Encoder: It encodes raw audio data according to a certain codec
 - Input: Raw Data
 - Output: Codec Data
4. File-Reader: It reads data from a file
 - Input: None
 - Output: Generic Data

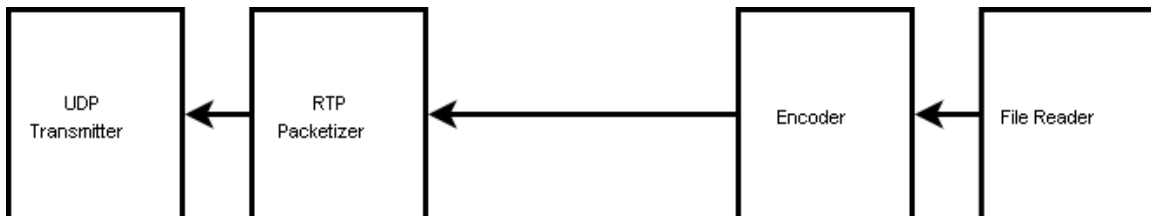


Figure 5 - Media Transmitter Chain

Others:

1. Buffer: It provides a break in a continuous sequence of processing of components.
 - Input: Generic Data
 - Output: Generic Data
2. TimeQueue: It queues the packets that are given it and passes packets to next component after a defined time interval.
 - Input: Generic Data
 - Output: Generic Data
3. Mixer: it mixes audio from two or more sources
 - Input: Raw Data
 - Output: Raw Data
4. Audio Level Detector: It detect audio volume in raw audio signal
 - Input: Raw Data
 - Output: Raw Data

5. State Machines

The following state machines are discussed:

6. Design of MSCML Play State Machine

6.1. MSCML Play State Machine Inputs

- **PLAY_STARTED**
File has started playing.
- **PLAY_STOPPED**
File has stopped playing.
- **TIMER_EXPIRED**
Timer has expired.
- **SM_STOP**
It's a request from the Network to stop the State Machine.
- **START_ERROR**
There was an error in playing the file.
- **STOP_ERROR**
The file that is playing can not be stopped.
- **MSCML_REQUEST**
This input is a request from the network, an MSCML file.

6.2. MSCML Play State Machine Input Parameters

PLAY_STARTED Parameters

```
struct PlayStartedParams
{
    int iStartTime; //time stamp
}
```

iStartTime:

This is the starting time of the file that is to be played. This field contains the integer value of the starting time of the media file.

PLAY_STOPPED Parameters

```
struct PlayStoppedParams
{
    int iStopTime; //time stamp
}
```

iStopTime:

This is the stop time of the file that is being played. This field contains the integer value of the stopping time of the media file.

TIMER_EXPIRED Parameters

Media Server Design Document

```
struct TimerExpiredParams
{
    int iTimerId;
}
```

iTimerId:

State machine receives this message from the Timer and the duration for the file being played expires.

SM_STOP Parameters

```
struct SMStopParams
{
}
```

PLAY_ERROR Parameters

```
struct PlayErrorParams
{
    int    iErrorCode;
}
```

iErrorCode:

This input message is sent by the Player component in case it experiences an error in playing the appropriate media file.

STOP_ERROR Parameters

```
struct StopErrorParams
{
    int    iErrorCode;
}
```

iErrorCode:

This input message is sent by the Player component in case it experiences an error in stopping the appropriate media file.

MSCML_REQUEST Parameters

```
struct MSCMLRequestParams
{
XMLDocPtr    xdPMSCMLPlayRequest;
}
```

xdPMSCMLPlayRequest:

This is a document pointer sent by the network element to state machine. It contains details like file name, duration of play etc.

6.3. MSCML Play State Machine Outputs Parameters

- **START_PLAY**
Start playing a file
- **MSCML_RESPONSE**
Generate an MSCML Response
- **STOP_PLAY**
Stop playing a file
- **NONE**
Some inputs may be mapped to NONE
- **SM_STOPPED**
Its shows that the State machine is stopped
- **START_TIMER**
It starts and timer

6.4. MSCML Play Output Parameters

START_PLAY Parameters

```
struct StartPlayParams
{
int iOffset;
char *cPFilePath;
}
```

iOffset:

State machine sends this parameter to Player. This field contains the starting point for the requested media file. The time value is by default sent in milliseconds.

***cPFilePath:**

State machine sends this parameter to Player. This field contains the pointer to the file that is to be played.

Media Server Design Document

MSCML_RESPONSE Parameters

```
struct MSCMLResponseParams
{
    xml    DocPtr  xdpMSCMLResponse;
}
```

xdpMSCMLResponse:

State machine sends this response to the network element. This parameter contains pointer to the document that contains report related to the requested media file. Report contains details like duration for which the file was played, errors encountered if any etc.

STOP_PLAY Parameters

```
struct Stop_Play_Params
{
}
```

NONE Parameters

```
struct NoneParams
{
}
```

SM_STOPPED Parameters

```
struct SMStoppedParams
{
}
```

START_TIMER Parameters

```
struct StartTimerParams
{
    int iTimerId;
    int iTimerDurationMilliSec;
}
```

iTimerId:

This field contains the ID for the type of timer we want the Timer component to start. There can be different types of timers like duration timer or delay timer.

iTimerDurationMilliSec:

State machine outputs this parameter to the Timer component. This field contains the duration of time for which the requested media file is to be played.

6.5. States of Play State Machine

PSM_Init

It's the initial state of the MSCML play collect state machine, where MSCML request is received and processed.

PSM_PlayingSeq

In this state of state machine, all files of the sequence that are to be played are listed.

Waiting4InterSeqDelay

State machine waits in this state for inter sequence delay to be expired.

PSM_SM_Halt

State machine goes into this stage on receiving the stop message from the network element.

6.6. State Machine MSCMLPlayContext

```
{  
  
    e_PlaySMInputEventTypes eCurrentState;    //stores the current state  
    long int nPlayDurationStartTime;        //Stores the start time of the duration of play  
    long int nPlayOffsetStartTime;        //Stores the start time of the offset of play  
    int nRepeat ;                          //It is used to keep track of the number of times a file has been  
                                           played.  
    nRepeat;                               //value changes with time  
    int nLeftNoOfAudioFilesInSeq;          // It is used to keep track of the number of  
                                           audio files that have to be played. Its value also  
                                           changes with time  
  
    //New attributes  
    int nDelayAttr ;                       //Stores the time duration for delay  
    int nDurationAttr;                    //Stores the time duration for playing the audio sequence.  
    int nOffsetAttr;                      //Stores the offset value for the audio sequence  
    int nRepeatAttr ;                     //Number of times a sequence is to be repeated  
    int nStuponerrorAttr;                 //Boolean value that indicates whether to stop on error or not.  
    int nTotalNoOfFilesInSeqAttr;         //total number of files in a sequence  
    MS_TIMER_ID pvDelayTimerID;           //ID for delay timer  
    MS_TIMER_ID pvDurationTimerID;        //ID for duration timer  
    int nMscmlPlayTraceID;                //stores error severity level  
    void* pvAMPSContext;                  //framework context  
  
    //Response Structure  
    t_MscmlPlayResponse oMscmlPlayResponse;  
    xmlChar* pxcUri;                      XML character string used by state machine  
};
```

6.7. Play State Machine Diagram

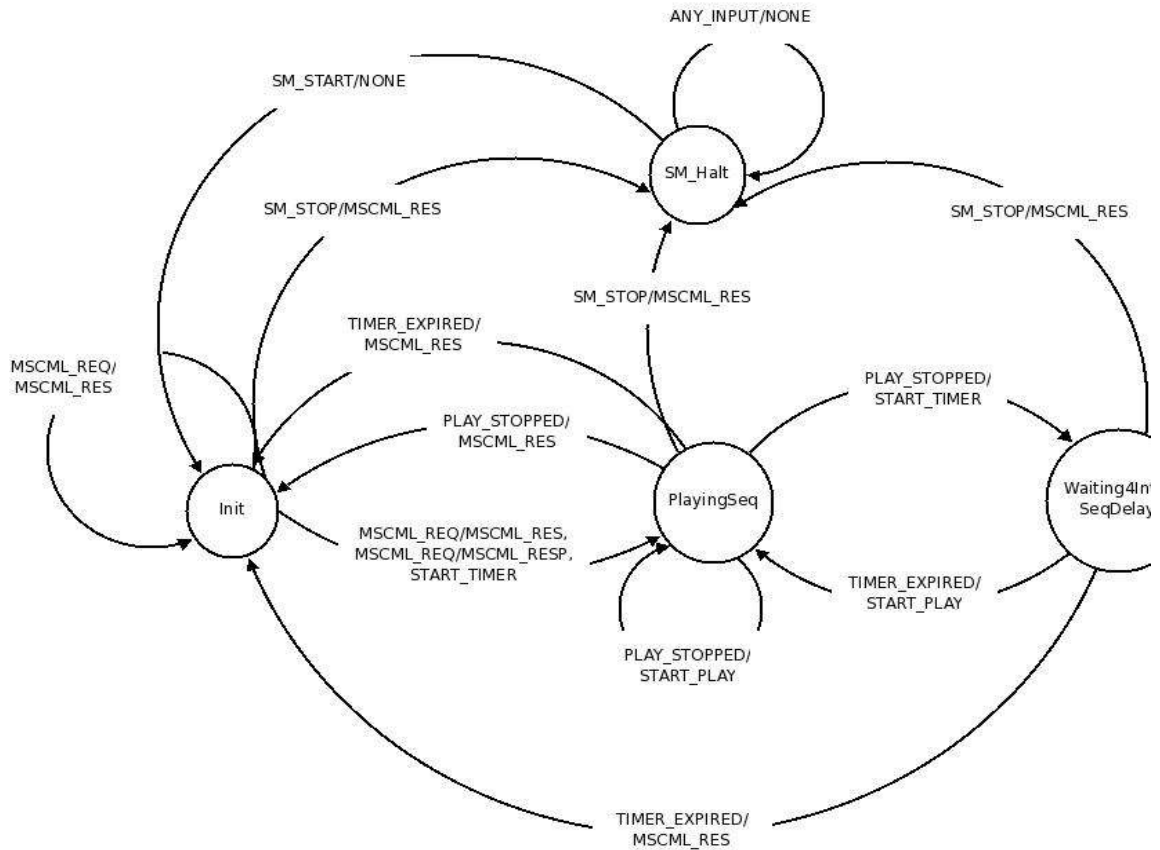


Figure 6 – Play State Machine

6.8. Pseudo-Code

PSM_Init

```

SM_STOP
{
    output=NONE;
    PlayMStates =PSM_SM_Halt;
    splayContext .iCurrentState=PSM_SM_Halt;
}
    
```

MSCML_REQ

```

{
    If (attributes.Repeat==0)
    {
        output =MSCML_RES;
        iPlayDuration=0;
        Generate_MSCML_RESPONSE(pvMSContext_i, poPSMInputEvent);
        PlayMStates =PSM_Init;
        splayContext .iCurrentState=PSM_Init;
    }
}
    
```

```
Else {
if(Repeat>0 )
{

output[0]=START_PLAY;
splayContext .iPlayDurationStartTime==gettimeofday(&tp, &tzp);
splayContext .iPlayOffsetStartTime=gettimeofday(&tp, &tzp);
Repeat=Repeat-1;

}
if( Duration!=Infinite)
{
output[1]=START_TIMER{Duration};
}
PlayMStates =PSM_PlayingSeq;
splayContext .iCurrentState=PSM_PlayingSeq;

}
}
```

SM_STOP

```
output=NONE;
PlayMStates =PSM_SM_Halt;
splayContext.iCurrentState=PSM_SM_Halt;
```

STOP_ERROR

```
output=MSCML_RES;
StopPlayTimeStamp=gettimeofday(&tp, &tzp);
iPlayDuration=StopPlayTimeStamp-StartPlayTimeStamp;
PlayMStates =PSM_Init;
splayContext .iCurrentState=PSM_Init;
Generate_MSCML_RESPONSE(pvMSContext_i, poPSMInputEvent);
```

MSCML_REQ

```
output=MSCML_RES;
iPlayDuration=0;
Generate_MSCML_RESPONSE(pvMSContext_i, poPSMInputEvent);
PlayMStates =PSM_Init;
splayContext .iCurrentState=PSM_Init;
```

6.9. Functionalities of States of the Play State Machine

The following are the different states of the State Machine

PSM_PlayingSeq

- **PLAY_STOPPED**
if(NoOfAudiofilesInSeq>=1) //there exists an audio file in the given sequence
{

Media Server Design Document

```
output=START_PLAY;
PlayMStates =PSM_PlayingSeq;
splayContext .iCurrentState=PSM_PlayingSeq;
NoOfAudiofilesInSeq= NoOfAudiofilesInSeq-1;
}

else if( NoOfAudiofilesInSeq==0 )
{
if(Repeat>0)
{
f(getDelayAttrValue(>0))// if sequence has ended and Delay >0 then
{
output=START_TIMER{Delay};
PlayMStates =Waiting4InterSeqDelay;
splayContext .iCurrentState=Waiting4InterSeqDelay;
}
}

else if(getDelayAttrValue()==0)// if sequence has ended and Delay >0 then
{

output=START_PLAY;
NoOfAudiofilesInSeq=ciNoOfAudiofilesInSeq;
Repeat= Repeat-1;
NoOfAudiofilesInSeq= NoOfAudiofilesInSeq-1;
PlayMStates =PSM_PlayingSeq;
splayContext .iCurrentState=PSM_PlayingSeq;
}
}
else if(Repeat==0)
{
output=MSCML_RES;
StopPlayTimeStamp=gettimeofday(&tp, &tzp);
iPlayDuration=StopPlayTimeStamp-splayContext
.iPlayDurationStartTime;

Generate_MSCML_RESPONSE(pvMSContext_i,
poPSMInputEvent);

PlayMStates =PSM_Init;
}
}

}
```

▪ TIMER_EXPIRED

```
{

output=MSCML_RES;
StopPlayTimeStamp=gettimeofday(&tp, &tzp);
```

Media Server Design Document

```
        iPlayDuration=StopPlayTimeStamp-StartPlayTimeStamp;
        Generate_MSCML_RESPONSE(pvMSContext_i, poPSMInputEvent);
PlayMStates =PSM_Init;
splayContext .iCurrentState=PSM_Init;
}
    else if input==SM_STOP
    {
output=MSCML_RES;
StopPlayTimeStamp=gettimeofday(&tp, &tzp);
iPlayDuration=StopPlayTimeStamp-StartPlayTimeStamp;
Generate_MSCML_RESPONSE(pvMSContext_i, poPSMInputEvent);
    PlayMStates =PSM_SM_Halt;
    splayContext .iCurrentState=PSM_SM_Halt;
}

    ▪ START_ERROR
    {
        if (getStoponerrorAttrValue()==0)
        {
output=START_PLAY;
                Repeat= Repeat-1;
                NoOfAudiofilesInSeq= NoOfAudiofilesInSeq-1;
                PlayMStates =PSM_PlayingSeq;
splayContext .iCurrentState=PSM_PlayingSeq;

        }
        else if (getStoponerrorAttrValue()==1)
        {
output=MSCML_RES;
StopPlayTimeStamp=gettimeofday(&tp, &tzp);
iPlayDuration=StopPlayTimeStamp-StartPlayTimeStamp;

Generate_MSCML_RESPONSE(pvMSContext_i, poPSMInputEvent);

PlayMStates =PSM_Init;
splayContext .iCurrentState=PSM_Init;
        }
    }

    ▪ STOP_ERROR
    {
output=MSCML_RES;
StopPlayTimeStamp=gettimeofday(&tp, &tzp);
iPlayDuration=StopPlayTimeStamp-StartPlayTimeStamp;

Generate_MSCML_RESPONSE(pvMSContext_i, poPSMInputEvent);

PlayMStates =PSM_Init;
        splayContext .iCurrentState=PSM_Init;
```

```
}
```

PSM_Waiting4InterSequenceDelay

- **TIMER_EXPIRED**

```
{  
if(Repeat>1)//Sequence is needed to be repeated after a delay  
{  
output=START_PLAY;  
PlayMStates =PSM_PlayingSeq;  
splayContext .iCurrentState=PSM_Waiting4InterSequenceDelay ;  
}  
else if(Repeat<=1)//No repetition is needed )then  
{  
output=MSCML_RES;  
StopPlayTimeStamp=gettimeofday(&tp, &tzp);  
iPlayDuration=StopPlayTimeStamp-StartPlayTimeStamp;  
Generate_MSCML_RESPONSE(pvMSContext_i,  
poPSMInputEvent);  
  
PlayMStates =PSM_Init;  
splayContext .iCurrentState=PSM_Init;  
}  
}
```

- **SM_STOP**

```
{  
output=MSCML_RES;  
StopPlayTimeStamp=gettimeofday(&tp, &tzp);  
iPlayDuration=StopPlayTimeStamp-StartPlayTimeStamp;  
Generate_MSCML_RESPONSE(pvMSContext_i, poPSMInputEvent);  
PlayMStates =PSM_SM_Halt;  
splayContext .iCurrentState=PSM_Init;  
}  
}
```

- **START_ERROR**

```
{  
output=MSCML_RES;  
StopPlayTimeStamp=gettimeofday(&tp, &tzp);  
iPlayDuration=StopPlayTimeStamp-StartPlayTimeStamp;  
Generate_MSCML_RESPONSE(pvMSContext_i, poPSMInputEvent);  
PlayMStates =PSM_Init;  
splayContext .iCurrentState=PSM_Init;  
}  
}
```

Media Server Design Document

```
▪ STOP_ERROR
{
    output=MSCML_RES;
    StopPlayTimeStamp=gettimeofday(&tp, &tzp);
    iPlayDuration=StopPlayTimeStamp-StartPlayTimeStamp;
    Generate_MSCML_RESPONSE(pvMSContext_i, poPSMInputEvent);
    PlayMStates =PSM_Init;
    splayContext .iCurrentState=PSM_Init;
}

▪ MSCML_REQ
{
    output=NONE;
    PlayMStates =PSM_Waiting4InterSequenceDelay;
    splayContext .iCurrentState=PSM_Waiting4InterSequenceDelay;
}

▪ SM_START
{
    output=NONE;
    PlayMStates =PSM_Waiting4InterSequenceDelay;
    splayContext .iCurrentState=PSM_Waiting4InterSequenceDelay;
}
```

PSM_SM_Halt

```
▪ SM_START
{
    output=NONE;
    PlayMStates =PSM_Init;
    splayContext .iCurrentState=PSM_Init;
}
```

6.10. Play State Machine Diagram PsuedoCode

```
/*-----*/
xmlDocPtr MSCMLPlayReqFile;
int iPlayDuration;
int iPlayOffset;

int CalcNoOfAudioFilesInSeq(sturct MSCMLPlayContext, xmlDocPtr MSCMLPlayReqFile);

LIST GetListOfFilesInSeq(sturct MSCMLPlayContext, xmlDocPtr MSCMLPlayReqFile);

int CalcDurationOfSeqOfFiles ( sturct MSCMLPlayContext, LISTOfFILES);

int CalculateDurationOfAnAudioFile(sturct MSCMLPlayContext, char *fileURI);
```

Media Server Design Document

```
int getRepeatAttrValue();
int getDelayAttrValue();
int getDurationAttrValue();
int getOffsetAttrValue();
int getRepeatAttrValue();
int getStoponerror AttrValue();
```

```
/**/
```

```
int CalcNoOfAudioFilesInSeq(sturct MSCMLPlayContext, xmlDocPtr MSCMLPlayReqFile)
{
while(!EOF)
{
if(Node's attribute Name=="AudioURI"&& the Attribute Value!=NULL )
{
seqCounter=seqCounter+1;
}
}
return seqCounter;
}
```

```
/**/
```

```
LIST GetListOfFilesInSeq(sturct MSCMLPlayContext, xmlDocPtr MSCMLPlayReqFile)
{
while(!EOF)
{
if(Node's attribute Name=="AudioURI"&& the Attribute Value!=NULL )
{
put the value of the attribute in LISTOfFILES
}
}
return LISTOfFILES;
}
```

```
/**/
```

```
int CalcDurationOfSeqOfFiles ( sturct MSCMLPlayContext, LISTOfFILES)
{
int sumOfSeqOfFilesDurations=0;
While(seqCounter)
{
sumOfSeqOfFilesDurations +=CalculateAudioFileDuration(char *fileURI)
seqCounter=seqCounter-1;
}
return sumOfSeqOfFilesDurations;
}
```

```
/**/
```

Media Server Design Document

```
int CalculateAudioFileDuration(sturct MSCMLPlayContext, char *fileURI)

{
Calculate the Audio file Duration and

return Duration;
}
/*****/
int getRepeatAttrValue(){
return attributes.Repeat;
}
int getDelayAttrValue(){
return attributes.Delay ;
}
int getDurationAttrValue(){
return attributes.Duration ;
}
int getOffsetAttrValue(){
return attributes.Offset ;
}
int getRepeatAttrValue(){
return attributes.Repeat ;
}
int getStoponerror AttrValue(){
return attributes.Stoponerror ;
}

/*****/

iNoOfAudiofilesInSeq= CalcNoOfAudioFilesInSeq(MSCMLPlay, MSCMLPlayReqFile);
ciNoOfAudiofilesInSeq= CalcNoOfAudioFilesInSeq(MSCMLPlay, MSCMLPlayReqFile);
Repeat=getRepeatAttrValue();
Delay=getDelayAttrValue();
Duration=getDurationAttrValue();
Offset=getOffsetAttrValue();
Repeat=getRepeatAttrValue();
Stoponerror=getStoponerrorAttrValue();

/*****/
void MSCMLPlayStateMachineModule( sturct MSCMLPlayContext splayContext, sturct
InputEvent *spInputEvent )
{
Switch(PlayMStates)
{

/*****/
```

7. MSCML Play Collect State Machine

7.1. MSCML Play Collect State Machine Inputs

1. SM_STOP

Request to stop MSCML Play Collect State Machine

Parameter:

NONE

2. MSCML_REQUEST

MSCML play collect request received by they state machine

Parameter:

xmlDocPtr xdPMSCMLPlayCollectRequest
Pointer to libxml parsed message(play collect)

3. PLAY_ERROR

System encountered error while trying to play the requested audio file

Parameter:

int nErrorCode

4. STOP_ERROR

System encountered error while trying to stop the requested audio file

Parameter:

int nErrorCode

5. PLAY_ENDED

The file being played has ended

Parameter:

NONE

6. FF_KEY_PRESSED

Forward key is pressed to forward the current media file/sequence

Parameter:

NONE

7. RW_KEY_PRESSED

Rewind key is pressed to rewind the current media file/sequence

Parameter:

NONE

8. RETURN_KEY_PRESSED

Return key, indicates that user has completed input and all the digits entered are forwarded to the client.

Parameter:

NONE

9. ESCAPE_KEY_PRESSED

Escape key, indicates that the user wants to terminate the current operation.

Parameter:

NONE

10. DTMF_KEY_PRESSED

DTMF key is pressed

Parameter:

NONE

11. PLAY_DURATION_TIMER_EXPIRED

Duration time for playing the sequence has expired

Parameter:

NONE

12. INTER_SEQ_TIMER_EXPIRED

Delay time between two sequences has expired

Parameter:

NONE

13.

COLLECTION_TIMER_EXPIRED

One of the collection timers has expired.

Parameter:

NONE

7.2. MSCML Play Collect State Machine Outputs

1. SM_STOPPED

play-collect state machine has stopped

Parameter:

NONE

2. MSCML_RESPONSE

MSCML response is generated for a particular request

Parameter:

xmlDocPtr xdPMSMMLPlayCollectResponse;
Pointer to libxml parsed message(play collect response)

3. START_PLAY

Start playing an audio file at a given offset.

Parameter:

int iOffset(ms);
Number of milliseconds from the start of the audio file

char *pcFilePath;
Path and name of the audio file to be played

4. STOP_PLAY

Stop playing currently playing file.

Parameter:

NONE

5. START_PLAY_DURATION_TIMER

Start a play duration timer for a given duration (in milliseconds)

Parameter:

int nPlayDurationMs

6. STOP_PLAY_DURATION_TIMER

Stop play duration timer

Parameter:

NONE

7. START_INTER_SEQ_TIMER

Start inter sequence timer before a new sequence is played

Parameter:

int nInterSeqTimerMs

8. STOP_INTER_SEQ_TIMER

Stop the inter sequence timer

Parameter:

NONE

9. START_COLLECTION_TIMER

Start one of the collection timers. Depending on the state of the state machine, relevant timer is started

Parameter:

NONE

10. STOP_COLLECTION_TIMER

Stop one of the collection timers. Depending on the state of the state machine, relevant timer is stopped

Parameter:

NONE

7.3. Play Collect State Machine Diagram

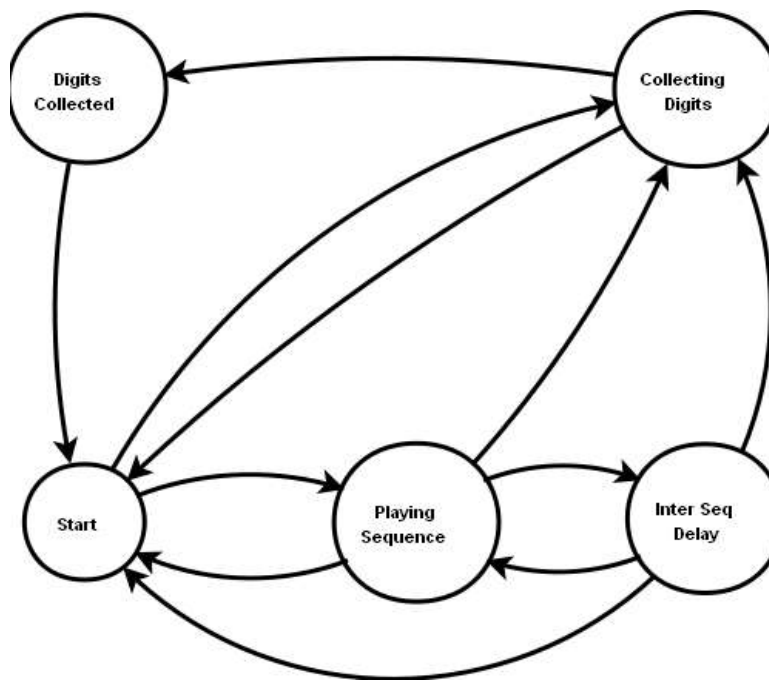


Figure 7 - Play Collect State Machine

7.4. MSCML Play Collect State Machine Context Elements

```
Context
typedef struct t_PlayCollectContext
{
    bool        bClearDigits
    bool        bBarge
    bool        bStopOnError
bool bRegularExpressionMatched
    int         nExtraDigitTimer
    int         nInterDigitTimer
    int         nCurrentRepeatCount
    int         nRepeat
```

Media Server Design Document

```
int          nCurrentAudioFileInSeq
char*       pcAudioFileList[]
int         nNumDigitsCollected
int         nMaxDigits
int         nPlayStartTime
int         nSeqStartTime
int         nPlayStartOffset
int         nSeqOffset
int         nDelayTimerId;
int         nDurationTimerId;
int         nMscmlPlayTraceId;
int         nCollectionTimerId;
int         nTotalDurOfAudioFiles
void*       pvAMPSContext;
eMPCSMState ePcCurrState;
t_MscmlPlayResponse oMscmlPlayResponse;

}oPlayCollectContext;
```

1. bool bClearDigits

The value stored in this field is converted into Boolean first and then is stored

2. bool bBarge

The value of “barge” attribute in boolean

3. bool bStopOnErrors

The value of “stoponerror” attribute in boolean

4. int nExtraDigitTimer

Value of extradigittimer in integer (milliseconds)

5. int nInterDigitTimer

The value of “Delay” attribute in integer (millisecond)

6. bool bRegularExpressionMatched

The value of “regex” attribute in boolean

7. int nCurrentRepeatCount

The Modified value of the “Repeat” after playing a sequence

8. int nRepeat

The number of times a sequence is to be repeated

9. int nCurrentAudioFileInSeq

The value(index) of the current audio files

10. char* pcAudioFileList[]

Media Server Design Document

Names of the audio file in the request

11. int nNumDigitsCollected

The value of the total number of DTMF digits collected

12. int nMaxDigits

The maximum number of digits that can be sent in a request

13. int nPlayStartTime

System time at the start of the play. When the audio file is played, at that point the system time is stored in this variable

14. int nPlayStartOffset

Value of the Play offset to start with

15. int nSeqStartTime

System time at the point at which a particular sequence is being played

16. int nSeqOffset

Offset value that is to be applied on all files within a particular sequence

17. int nPCCurrState

Value of the current state of the play-collect state machine

18. int nDelayTimerId;

Value of the Delay Timer Id

19. int DurationTimerId;

Value of the duration timer Id

20. int nMscmlPlayTraceId;

Value of the TRACE Id

21. int nCollectionTimerId;

Value of one of the collection timer Ids

22. void* pvAMPSContext;

AMPS's Context

23. MscmlPlayResponse oMscmlPCResponse;

The value of generated MSCML Response for the request

24. int nTotalDurOfAudioFiles

The value for duration of audio files in a sequence

7.5. Play Collect State Machine States

PCSM_Start

It's the initial state of the MSCML play collect state machine, where MSCML request is received and processed.

PCSM_PlayingSequence

In this state of state machine, all files of the sequence that are to be played are listed

PCSM_InterSeqDelay

State machine waits in this state for inter sequence delay to be expired

PCSM_CollectingDigits

In this state all the digits are collected

PCSM_DigitsCollected

In this state, state machine waits for the extra digits timer to expire

7.6. MSCML Play Collect State Machine pseudo Code

1. PCSM_Start

▪ MSCML_REQUEST

1. if <prompt> is present
 1. if either max_digits or regexp present
 1. if Repeat is equal to zero
 1. Go to the collection part of the request
 2. else if Repeat is greater than zero
 1. if Duration is greater than zero
 1. Move to PCSM_PlayingSeq
 2. Calculate offset
 3. Generate START_PLAY with the the Offset.
 4. Generate START_TIMER(Duration)
 2. else if Duration is equal to infinite
 1. Move to PCSM_PlayingSeq
 2. Calculate offset
 3. Generate START_PLAY with the Offset.
 2. else //max_digits or regexp not present
 - return error of "Invalid request"
 2. else //<prompt> not present
 - Move to PCSM_CollectingDigits state
 - Generate START_COLLECTION_TIMER(first digit timer)

2. PCSM_PlaySequence

1. SM_STOP

1. Move to PCSM_Start
2. Generate MSCML_RESPONSE

2. PLAY_ERROR

1. if StopOnError is true
 1. Move to PCSM_Start and
 2. Generate MSCML_RESPONSE with error
1. else //StopOnError is false
 1. if sequence is repeated the required number of times
 1. Move to PCSM_CollectingDigits state
 2. Generate START_COLLECTION_TIMER(firstdigittimer)
 2. else // Sequence is to be repeated
 1. if delay is to be given between sequences
 1. Generate START_INTER_SEQ_TIMER(delay)
 2. Move to PCSM_InterSeqDelay
 3. Reinitialize the currently played file counter by zero
 4. Decrement the repeat value by one
 5. Generate START_PLAY
 2. else //delay is not needed
 1. Reinitialize the currently played file counter
 2. Increment the repeat value by one
 3. Generate START_PLAY

1. STOP_ERROR

1. Move to PCSM_Start
2. Generate MSCML_RESPONSE with error

1. PLAY_ENDED

1. if all the files of a sequence are played
 2. if sequence is repeated the required number of times
 1. Move to PCSM_CollectingDigits state
 2. Generate START_COLLECTION_TIMER(firstdigittimer)
 3. else // Sequence is to be repeated
 1. if delay is to be given between sequences
 1. Generate START_INTER_SEQ_TIMER(delay)
 2. Move to PCSM_InterSeqDelay
 3. Reinitialize the currently played file counter
 4. Decrement the repeat value by one
 2. else //delay is not needed
 1. Reinitialize the currently played file counter by zero
 2. Decrement the repeat value by one
 3. Generate START_PLAY
2. else //if all the files are not played
 1. Increment the currently played file counter by one
 2. Generate START_PLAY

5. FF_KEY_PRESSED

1. Get current time
2. Get the Start time of the sequence
3. add the sum of Start Timer and Current time with Key forward time
4. Get file from offset (Start Timer+Current time+ Key forward time)
5. Generate START_PLAY with the above calculated value and file.

6. RW_KEY_PRESSED

1. Get current time
2. Get the Start time of the sequence
3. Subtract Key Rewind Timer form the sum of Start Timer and Current time
4. Get file from offset ((Start Timer+Current time)- Key Rewind time)
5. Generate START_PLAY with the above calculated value and file.

7. DTMF_KEY_PRESSED

1. if Barge is true
 - ◆ Move to PCSM_CollectingDigits
 - ◆ Generate STOP_PLAY
 - ◆ Stop duration timer(if started)
 2. else //Barge is false
2. Do nothing

8. PLAY_DURATION_TIMER_EXPIRED

1. Move to PCSM_CollectingDigits to start collecting digits
2. STOP_PLAY

3. PCSM_InterSeqDelay :

1. SM_STOP

1. Move to PCSM_Start
2. Generate MSCML_RESPONSE

2. FF_KEY_PRESSED

1. Get current time
2. Get the Start time of the sequence
3. add the sum of Start Timer and Current time with Key forward time
4. Get file from offset (Start Timer+Current time+ Key forward time)
5. Generate START_PLAY with the above calculated value and file.

3. RW_KEY_PRESSED

1. Get current time.
2. Get the Start time of the sequence
3. Subtract Key Rewind Timer form the sum of Start Timer and Current time
4. Get file from offset ((Start Timer+Current time)- Key Rewind time)
5. Generate START_PLAY with the above calculated value and file.

4. DTMF_KEY_PRESSED

1. if Barge is true
 1. Move to PCSM_CollectingDigits
 2. On Entry, Start one of the Collection Timers
 3. Stop the Delay timer and Duration timer(if started)
 2. else //Barge is false
1. Do nothing

5. INTER_SEQ_TIMER_EXPIRED

1. Move to PCSM_PlayingSequence and
2. Initialize currently played file counter by zero
3. Generate START_PLAY

4. PCSM_CollectingDigits:

1. SM_STOP

1. Move to PCSM_Start and
2. Generate MSCML_RESPONSE

2. RETURN_KEY_PRESSED

1. Generate MSCML_RESPONSE with reason equal to return-key and digits collected
2. Move to PCSM_Start State

3. ESCAPE_KEY_PRESSED

1. Generate MSCML_RESPONSE reason equal to escape-key
2. Move to PCSM_Start State

4. DTMF_KEY_PRESSED

1. Generate STOP_COLLECTION_TIMER
2. if maxdigit present
 1. Store the digit in a **buffer**
 2. increment current digit counter
 3. if current digit counter is equal to maxdigit
 1. Move to PCSM_DigitsCollected
 2. Generate START_COLLECTION_TIMER(extradigit timer)
 4. else
 1. Generate START_COLLECTION_TIMER(interdigit timer)
3. else if regular expressions is specified
 1. if regular expression matched
 1. Move to PCSM_DigitsCollected
 2. Generate START_COLLECTION_TIMER(interdigitcritical timer)
 2. else
 1. Generate START_COLLECTION_TIMER(interdigit timer)

5. COLLECTION_TIMER_EXPIRED

1. **Generate MSCML_RESPONSE with reason timeout and currently collected digits.**
2. **Generate STOP_COLLECTION_TIMER**
3. **Move to PCSM_Start**

5. PCSM_DigitsCollected:

1. SM_STOP

- Move to PCSM_Start
- Generate MSCML_RESPONSE
- Generate STOP_COLLECTION_TIMER

2. DTMF_KEY_PRESSED

1. if maxdigit present
 1. add digit to the buffer
 2. else regular expression present
 1. Call match function for the regular expression if true
Generation MSCML_RESPONSE with reason digits

3. COLLECTION_TIMER_EXPIRED

- Generate MSCML_RESPONSE with reason timeout and currently collected digits.
- Generate STOP_COLLECTION_TIMER
- Move to PCSM_Start.

7.7. Digit Regular expression

Digit Regular expression is used to collect the DTMF keys pressed by user according to a certain format. This is a regular expression like format as defined in Appendix A of RFC-5022. The client of Media Server can specify the regular expression in <playcollect> tag. An example of the MSMCL message is below:

```
<?xml version="1.0" encoding="UTF-8"?>
<MediaServerControl version="1.0">
<request>
<playcollect>
<prompt baseurl="http://www.example.com/prompts/">
<audio url="generic/en_US/enter_pin.wav"/>
</prompt>
<pattern>
<regex value="*6[179#]" name="a"/>
<regex value="[179#]" name="b"/>
</pattern>
</playcollect>
</request>
</MediaServerControl>
```

In this example the user has specified two regular expressions, "*6[179#]" and "[179#]".

The first part is to parse the regular expression itself. This will ensure that the regular expression is according to the specified syntax. The parser generates a data-structure that is suitable for collecting user inputs in an efficient way.

Parsing Expression Grammar (PEG) is used to parse the regular expression. PEG is a standard notation that can be used to parse a string that has defined format. Once the syntax of the input

Media Server Design Document

is specified in the PEG notation, standard parser-generators are available. These generators take the PEG notation as input and generates code of a basic parser. More code is added to generate the data-structures that are required.

Digit Regular Expression PEG Notation:

```
dreg_expr:      (repetition_expr / range_expr / long_expr / x / star / hash / dot / digit )+
repetition_expr: ( range_expr / x / star / hash / dot / digit ) repetition
range_expr:     '[' range_value+ ']'
long_expr:      long ( x / star / hash / dot / digit )
range_value:    range / digit / extra
range:          (digit '-' digit) / (extra '-' extra)
repetition:     '{' num? ','? num? '}'
digit:          '0' / '1' / '2' / '3' / '4' / '5' / '6' / '7' / '8' / '9'
extra:          'a' / 'b' / 'c' / 'd' / 'A' / 'B' / 'C' / 'D'
num:           digit+
x:             'x'
star:          '*'
hash:          '#'
dot:           '.'
long:          'L'
```

The parse generates the following structures

- t_DRegDigit
 - char cValue : Value of the digit

- t_DRegRangeExpr
 - t_List* poNodeList : list of nodes in range expression

- t_DRegX
 - none

- t_DRegDot
 - none
- t_DRegStar
 - none

- t_DRegHash
 - none

- t_DRegLongExpr
 - t_DRegNode* poNode : The node that is to be detected for long duration

- t_DRegRepetitionExpr
 - t_DRegNode* poNode : The node that is to be repeated
 - int nCurrentRepCount : Number of times the node is repeated
 - int nMinRep : Minimum number of times the node is to be repeated
 - int nMaxRep : Maximum number of times the node is to be repeated

- t_DRegRange

Media Server Design Document

- char cMinValue : Minimum value of range
- char cMaxValue : Maximum value of range

- t_DRegExpr
 - t_List* poNodeList : list of nodes in regular expression
 - char pcMatchBuffer[MAX_DREG_EX_MATCH_BUFFER_LENGTH] : Collected digits
 - t_DRegNode* poCurrentNode : Current node that is to be matched

- t_DRegNode
 - e_DRegNodeType : Type of node
 - union
 - t_DRegDigit
 - t_DRegRangeExpr
 - t_DRegX
 - t_DRegDot
 - t_DRegStar
 - t_DRegHash
 - t_DRegLongExpr
 - t_DRegRepetitionExpr
 - t_DRegRange

The Regular expression functionality is available in the form of an API. The API is listed below

In addition to these structures, following structures are used by the API

1. t_DRegExConfig

char pcName[MAX_DREG_EX_NAME_LENGTH] :User-defined name of expression
char pc Expression[MAX_DREG_EX_EXPRESSION_LENGTH] :D-Regex to match
unsigned int unLongMatchDurationMs : Duration of long match in milliseconds

e_DRegExRetCode: Status codes returned by API

DRE_Ok : Operation completed successfully
DRE_Matched : Digit was matched by the regular expression
DRE_NotMatched : Digit was not matched by the regular expression
DRE_MatchComplete : Digit caused regular expression to match completely
DRE_InsufficientMemory : Memory Allocation error
DRE_InvalidExpression : Expression passed is not a valid Dreg expression

e_DRegExRetCode re_create(hDRegEx* phDRegEx_o, tDRegExConfig* poDRegExConfig_i)
Create a digit-regular expression. The regular expression is parsed and internal data structures are generated.

phDRegEx_o: Place holder to return the handle for the regular expression
poDRegExConfig_i: configuration parameters of regular expression

Media Server Design Document

e_DRegExRetCode re_destroy(hDRegEx* pHRegEx_io)
Destroy a digit-regular expression.

pHRegEx_io: Pointer to handle of regular expression

e_DRegExRetCode re_match(hDRegEx hDRegEx_i, char cDigit_i)
Match a digit with the regular expression.

hDRegEx_i: Handle of regular expression
cDigit_i: Ascii value of the digit pressed by the user

e_DRegExRetCode re_get_current_match(hDRegEx hDRegEx_i, char* pcMatch_o, unsigned int*
punNumDigits_io)
Get the digits that are currently matched

hDRegEx_i: Handle of regular expression
pcMatch_o: Pointer to the buffer to store the matched digits
punNumDigits_io: (On input) Maximum length of buffer to store digits. (On output) Actual
number of digits stored

e_DRegExRetCode re_get_name(hDRegEx hDRegEx_i, char* pcName_o, unsigned int*
punNumDigits_io)
Get the name of the regular expression.

hDRegEx_i: Handle of regular expression
pcMatch_o: Pointer to the buffer to store the name
punNumDigits_io: (On input) Maximum length of buffer to store name. (On output) Actual
length of name

Pseudo-code

The pseudo-code of the functions that match the input digit with a particular node of regular expression is as follows

Explanation: Match Digit to “.”

```
e_DRegExRetCode re_match_dot(hDRegEx hDRegEx_i, char cDigit_i)
1. ret_code = DRE_NotMatched
2. if cDigit_i is equal to 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, A, B, C, D, #, *
   1. Add cDigit_i in buffer
   2. ret_code = DRE_Matched
3. return ret_code
```

Explanation: Match Digit to “*”

e_DRegExRetCode re_match_star(hDRegEx hDRegEx_i, char cDigit_i)

1. ret_code = DRE_NotMatched
2. if cDigit_i is equal to “*”
 1. Add cDigit_i in buffer
 2. ret_code = DRE_Matched
3. return ret_code

Explanation: Match Digit to “#”

e_DRegExRetCode re_match_hash(hDRegEx hDRegEx_i, char cDigit_i)

1. ret_code = DRE_NotMatched
2. if cDigit_i is equal to “#”
 1. Add cDigit_i in buffer
 2. ret_code = DRE_Matched
3. return ret_code

Explanation: Match Digit to range expression

e_DRegExRetCode re_match_range_expr(hDRegEx hDRegEx_i, char cDigit_i)

1. ret_code = DRE_NotMatched
2. Loop through the list of child nodes
 1. ret_code = Call match function
 2. If ret_code = DRE_Matched
 1. break
3. return ret_code

Explanation: Match Digit to range

e_DRegExRetCode re_match_range(hDRegEx hDRegEx_i, char cDigit_i)

1. ret_code = DRE_NotMatched
2. if cDigit_i >= RangeMin and cDigit_i <= RangeMax
 1. Add cDigit_i in buffer
 2. ret_code = DRE_Matched
1. return ret_code

Explanation: Match Digit to “x”

e_DRegExRetCode re_match_x(hDRegEx hDRegEx_i, char cDigit_i)

1. ret_code = DRE_NotMatched
2. if cDigit_i is equal to 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
 1. Add cDigit_i in buffer
 2. ret_code = DRE_Matched
3. return ret_code

Explanation: Match Digit to repetition expression

e_DRegExRetCode re_match_repetition_expr(hDRegEx hDRegEx_i, char cDigit_i)

1. ret_code = DRE_NotMatched
2. ret_code = Call match function of “value”
3. if ret_code = DRE_Matched
 1. ret_code = DRE_NotMatched

Media Server Design Document

2. Increment current_rep_count
 3. if min_rep == max_rep // {m}
 1. if current_rep_count == min_rep
 1. Add cDigit_i in buffer
 2. ret_code = DRE_Matched
 4. else if min_rep == -1 // {,n}
 1. if current_rep_count <= max_repd
 1. Add cDigit_i in buffer
 2. ret_code = DRE_Matched
 5. else if max_rep == -1 // {m,}
 1. if curr_rep_count >= min_rep
 1. Add cDigit_i in buffer
 2. ret_code = DRE_Matched
4. return ret_code

Explanation: Match Digit to regular expression

e_DRegExRetCode re_match_dreg_expr(hDRegEx hDRegEx_i, char cDigit_i)

1. ret_code = DRE_NotMatched
1. ret_code = Call match function of current_node
2. if ret_code = DRE_Matched
 4. if more nodes available after current_node
 1. current_node = next node
 5. else
 1. ret_code = DRE_MatchComplete
3. return ret_code

8. Play Record

8.1. Inputs

1. SILENCE_DETECTED: Generated when silence is detected for the first time after sound
2. SOUND_DETECTED: Generated when sound is detected for the first time after silence or at the start of the recording.
3. START_RECORDING_ERROR: An error occurred while starting a recording.
4. STOP_RECORDING_ERROR: An error occurred while stopping a recording
5. RECORDING_DURATION_TIMER_EXPIRED: Duration for which recording was being done has expired.
6. RECORDING_SILENCE_TIMER_EXPIRED: This input signifies that the timer that starts on detection of silence in the audio file has expired.
7. RECORDING_STOPPED: This event is generated in response to STOP_RECORDING. It inputs the number of bytes recorded.
8. PLAY_ERROR: System encountered error in trying to play the relevant audio file
9. STOP_ERROR: System encountered error in trying to stop the relevant audio file
10. PLAY_ENDED: The file being played has ended

8.2. Outputs

1. START_RECORDING
Start recording
2. STOP_RECORDING
Stop the current recording process
3. START_RECORDING_DURATION_TIMER
Start the duration timer of recording
4. STOP_RECORDING_DURATION_TIMER
Stop the already started "recording duration timer"
5. START_RECORDING_SILENCE_TIMER
If silence is detected, start the recording silence timer
6. STOP_RECORDING_SILENCE_TIMER
If sound is detected, stop the recording silence timer
7. START_PLAY

Media Server Design Document

Start playing an audio file at a given offset.

8. STOP_PLAY
Stop playing currently playing file.
9. START_PLAY_DURATION_TIMER
Start a play duration timer for a given duration (in milliseconds)
10. STOP_PLAY_DURATION_TIMER
Stop play duration timer.
11. START_INTER_SEQ_TIMER
Start inter sequence timer before a new sequence is played
12. STOP_INTER_SEQ_TIMER
Stop the inter sequence timer

8.3. Play Record State Machine Diagram

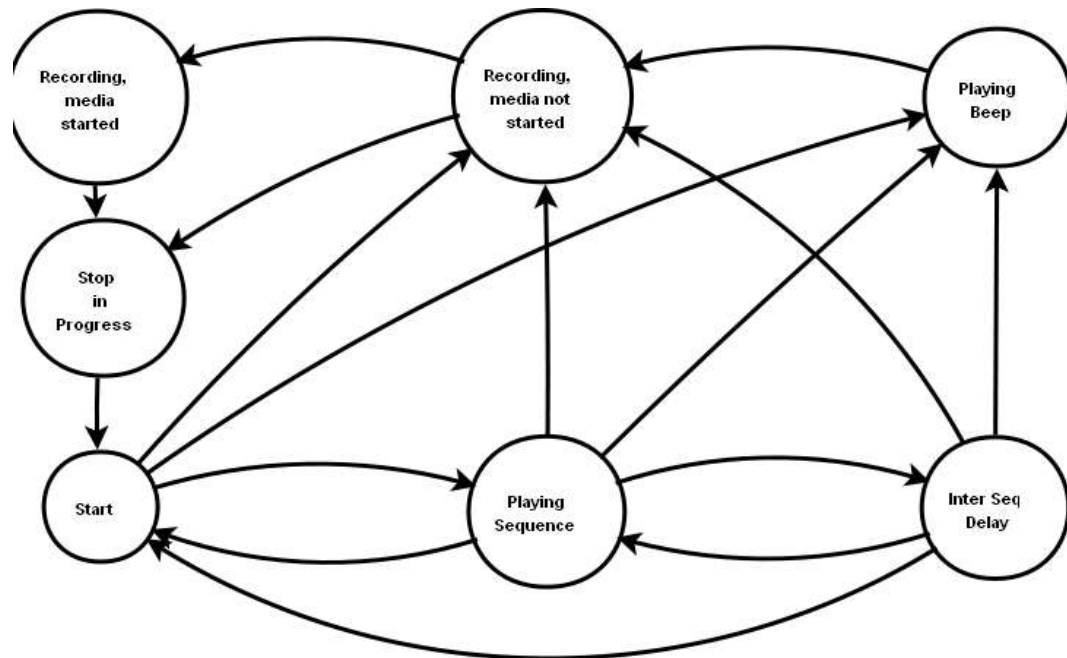


Figure 8 - Play Record State Machine

8.4. Context

1. int nRecordingDurationTimerId;
2. int nRecordingSilenceTimerId;
3. int nRecordStartTime;
4. bool bStopOnError;
5. int nCurrentRepeatCount;
6. int nRepeat;
7. int nCurrentAudioFileIneq;
8. char* pcAudioFileList[];

Media Server Design Document

9. int nPlayStartTime;
10. int nSeqStartTime;
11. int nPlayStartOffset;
12. int nSeqOffset;
13. int nDelayTimerId;
14. int nDurationTimerId;
15. int nMscmlPlayRecordTraceId;
16. int nTotalDurOfAudioFiles;
17. void* pvAMPSContext;
18. eMPRSMState ePRCurrState;
19. t_MscmlPlayRecordResponse oMscmlPlayRecordResponse;

- nRecordingDurationTimerId
ID for value duration timer
- nRecordingSilenceTimerId
ID for value Silence timer
- nRecordStartTime
The system time at the start of a recording session
- bool bStopOnErrors
The Boolean value of “stoponerror” attribute. Value is converted into Boolean before storing in this field
- int nCurrentRepeatCount
The Modified value of the “Repeat” after playing a sequence. After a sequence this value is reduced by 1
- int nRepeat
Number of times a sequence is to be repeated
- int nCurrentAudioFileInSeq
The value(index)of the current audio file
- char* pcAudioFileList[]
Names of the audio file in the request
- int nPlayStartTime
System time at the start of audio file play
- int nPlayStartOffset
Value of the Play offset to start with
- int nSeqStartTime
System time at the start of Sequence play
- int nSeqOffset
Offset value for a sequence. This value is applied on all the files within a sequence

Media Server Design Document

- int nPRCurrState
Value of the current state of the play-collect state machine
- int nDelayTimerId;
ID for Delay Timer
- int DurationTimerId;
ID for duration timer
- int nMscmlPlayRecordTraceId;
Value of the TRACE Id
- void* pvAMPSContext;
AMPS's Context
- MscmlPlayRecordResponse oMscmlPRecordResponse;
MSCML Response for the request
- int nTotalDurOfAudioFiles
The value of the total file duration that is calculated

8.5. Transformation

1. Start

1. MSCML_REQUEST
 1. if <prompt> present
 1. If duration in request is greater than ZERO
 1. if repeat_count is equal to repeat in request
 1. Change state to START
 2. Else// need to repeat more
 1. Calculate offset and filename to play
 2. Change state to PLAYING_SEQUENCE
 3. Generate
START_PLAY_DURATION_TIMER(request.prompt.duration)
 2. Else // duration in request is ZERO
 1. Change state to START
 2. else // prompt not present
 1. if beep is to be played
 1. Change state to PLAYING_BEEP
 2. else // beep is not to be played
 1. If recording duration in request is not "immediate"
 1. Change state to RECORDING_MEDIA_NOT_STARTED
 2. Else // recording duration in request is "immediate"
 1. Set reason to "stopped"
 2. Change state to START

2. Playing Sequence

1. SM_STOP
 1. set reason to "stopped"
 2. Change state to START
2. PLAY_STOPPED
 1. If more file to play in sequence
 1. Generate START_PLAY
 2. Else// sequence ended
 1. If inter-sequence delay in request is greater than 0
 1. Change state to INTER_SEQ_DELAY
 2. Else // no delay between sequence
 1. if repeat_count is equal to repeat in request
 1. Change state to START
 2. Else// need to repeat more
 1. Increment repeat_count
 2. Generate START_PLAY
3. PLAY_ERROR
 1. set reason to "error"
 2. Change state to START
4. ESCAPE_KEY_PRESSED
 1. set reason to "escapekey"
 2. If beep is to be played
 1. Change state to PLAYING_BEEP
 3. Else
 1. Change state to RECORDING_MEDIA_NOT_STARTED
5. DTMF_KEY_PRESSED
 1. if key one on rec_stop_mask
 1. set reason to "stopped"
 2. Change state to START
6. PLAY_DURATION_TIMER_EXPIRED
 1. set reason to "stopped"
 2. Change state to START

3. Inter-Sequence Delay

1. SM_STOP
 1. set reason to “stopped”
 2. Change state to START
2. PLAY_INTER_SEQ_TIMER_EXPIRED
 - 1.
3. DTMF_KEY_PRESSED
4. ESCAPE_KEY_PRESSED
5. PLAY_DURATION_TIMER_EXPIRED

4. Playing Beep

1. SM_STOP
 1. Change state to START
 2. Generate MSCML_RESPONSE(“stopped”)
2. PLAY_STOPPED
 1. Change state to RECORDING_MEDIA
 2. Generate START_RECORDING_SILENCE_TIMER(init_silence)
 3. Generate START_RECORDING_DURATION_TIMER
 4. Generate START_RECORDING
3. PLAY_ERROR
4. ESCAPE_KEY_PRESSED
 1. Set reason to “stopped”
 2. Change state to START

5. Recording, Media not started

1. SM_STOP
 1. Set reason to “stopped”
 2. Change state to STOP_IN_PROGRESS
2. SILENCE_DETECTED
 1. Pseudo code
 1. Generate START_RECORDING_SILENCE_TIMER(endsilence)
3. SOUND_DETECTED
 1. Pseudo code
 1. Change state to “Recording, media started”
 2. Generate STOP_RECORDING_SILENCE_TIMER
4. START_RECORDING_ERROR
 1. Set reason to “error”
 2. Change state to STOP_IN_PROGRESS
5. STOP_RECORDING_ERROR
 1. Set reason to “error”
 2. Change state to STOP_IN_PROGRESS
6. RECORDING_DURATION_TIMER_EXPIRED
 1. Set reason to “max_duration”
 2. Change state to STOP_IN_PROGRESS
 3. Generate MSCML_RESPONSE
7. RECORDING_SILENCE_TIMER_EXPIRED
 1. Set reason to “init_silence”
 2. Change state to STOP_IN_PROGRESS
8. ESCAPE_KEY_PRESSED
 1. Set reason to “escapekey”

Media Server Design Document

2. Change state to STOP_IN_PROGRESS
9. DTMF_KEY_PRESSED
 1. If DTMF_KEY present in "rec_stop_mask"
 1. Set reason to "digit"
 2. Change state to STOP_IN_PROGRESS
 2. Else
 1. //Ignore

6. Recording, media started

1. SM_STOP
 1. Set reason to "stopped"
 2. Change state to STOP_IN_PROGRESS
2. SILENCE_DETECTED
 1. Pseudo code
 1. Generate START_RECORDING_SILENCE_TIMER(endsilence)
3. SOUND_DETECTED
 1. Pseudo code
 1. Generate STOP_RECORDING_SILENCE_TIMER
4. START_RECORDING_ERROR
 1. Set reason to "error"
 2. Change state to STOP_IN_PROGRESS
5. STOP_RECORDING_ERROR
 1. Set reason to "error"
 2. Change state to STOP_IN_PROGRESS
6. RECORDING_DURATION_TIMER_EXPIRED
 1. Set reason to "max_duration"
 2. Change state to STOP_IN_PROGRESS
 3. Generate MSCML_RESPONSE
7. RECORDING_SILENCE_TIMER_EXPIRED
 1. Set reason to "end_silence"
 2. Change state to STOP_IN_PROGRESS
8. ESCAPE_KEY_PRESSED
 1. Set reason to "escapekey"
 2. Change state to STOP_IN_PROGRESS
9. DTMF_KEY_PRESSED
 1. If DTMF_KEY present in "rec_stop_mask"
 1. Set reason to "digit"
 2. Change state to STOP_IN_PROGRESS
 2. Else
 1. //Ignore

7. Stop in Progress

1. RECORDING_STOPPED
 1. set reclength
 2. Change state to "Start"

Onentry_START

1. Generate MSCML_RESPONSE

Onentry_RECORDING_MEDIA_NOT_STARTED

1. if request.recording.duration is not infinite
 1. Generate START_RECORDING_DURATION_TIMER (request.recording.duration)
2. Generate START_RECORDING_SILENCE_TIMER (request.recoding.initsilence)
3. Get current time and store in record_start_time

Onexit_RECORDING_MEDIA_NOT_STARTED

1. Generate STOP_RECORDING_SILENCE_TIMER(if not stopped already)

Onexit_RECORDING_MEDIA_STARTED

1. Generate STOP_RECORDING_DURATION_TIMER(if not stopped already)

Onexit_STOP_IN_PROGRESS

Onentry_PLAYING_BEEP

1. Generate START_PLAY("beep.wav")

Onentry_STOP_IN_PROGRESS

1. Generate STOP_RECORDING

Onentry_INTER_SEQ_DELAY

1. Generate START_INTER_SEQ_TIMER

Onexit_INTER_SEQ_DELAY

1. Generate STOP_INTER_SEQ_TIMER(if not already stopped)

Onentry_PLAYING_SEQUECNE

1. Generate START_PLAY(calculated offset and filename)
2. Increment current_repeat_counter

Onexit_PLAYING_SEQUENCE

1. Generate STOP_PLAY_DURATION_TIMER(if not already stopped)

ChangeState(new_state)

1. if current state is different from new state
 1. Call current state onexit function.
 2. Change state variable
 3. Call onentry on current (new) state.

9. Reference

http://en.wikipedia.org/wiki/Parsing_expression_grammar