Capítulo 3 Sistema de Gestión de Archivos de Video

Para el desarrollo de este sistema utilicé el modelo cascada retroalimentado. Opté por este modelo de ciclo de vida, ya que el sistema es para uso interno y la mayoría de los requerimientos ya se tenían. Los requisitos iniciales son los que se muestran a continuación como parte del objetivo.

Durante su desarrollo este sistema ha experimentado varios cambios, principalmente en la herramienta que genera las pizarras, que han derivado en un mejor funcionamiento y mayor familiaridad con el usuario. Estos cambios se mencionaran en este capítulo y en el siguiente.

3.1 Objetivos

Se desarrollará un Sistema de Gestión de Archivos de Video el cual será utilizado para archivar, catalogar y recuperar de una manera rápida y eficiente todo los vídeos creados dentro de la empresa.

Este sistema servirá para reducir el tiempo que se necesita para procesar, archivar y consultar este material. Con esta reducción de tiempo también se reduce el personal necesario para realizar estas tareas y también se requiere menos equipo especializado.

El sistema tiene que ser funcional en todos los sistemas operativos utilizados dentro del estudio los cuales son: Fedora, Mac OS X y Windows, y tienen que tener conexión con *Shotgun*.

Se busca usar la mayor cantidad de software libre y de código abierto posible, buscando reducir costos por un lado y por el otro, es más sencillo adaptarlo a las necesidades que se tienen en la empresa.

Implementar *qrcode* como código de identificación de material para control interno, con esta herramienta se busca conocer la viabilidad de usar este tipo de codificación, para en el futuro poder usarlo en la empresa.

3.2 Análisis

3.2.1 El sistema

El sistema tendrá que ser dividido en dos programas o herramientas, estos términos serán usados como sinónimos, que trabajaran de manera independiente entre sí, pero del resultado de la primera dependerá el buen funcionamiento de la segunda. Estas herramientas son:

- **csSlateCreator:**

Esta herramienta tendrá que ser capaz de realizar pizarras con los datos existentes en *Shotgun* y datos propios del proyecto. Las pizarras se generaran a partir de una serie de imágenes (machotes) que servirán como fondo y sobre los cuales se pondrá otra imagen que es la que contendrá los datos particulares de la *pizarra*. En algún lado de la *pizarra* se le colocara una imagen que representara algún dato de identificación de la *pizarra*, para poder identificarla en un futuro para su procesamiento. Los datos de la *pizarra* creada se guardaran en *Shotgun*, así como un thumbnail de la *pizarra*, que será el último cuadro.

Los datos que debe de contener la *pizarra* son los siguientes:

- Cliente – Es quien recibirá el producto final, en algunas ocasiones pueden haber dos clientes. Este dato es obligatorio.
- Agencia – Es quien se encarga de la campaña publicitaria y en algunas ocasiones puede trabajar en conjunto con otra. Este campo no es obligatorio, ya que pueden existir proyectos sin agencia.
- Casa Productora – En caso de que el proyecto requiera alguna filmación en acción real. Puede darse el caso de que haya dos productoras.
- Marca o Producto – Nombre del proyecto o artículo publicitado. Este campo es obligatorio.
- Versión – Nombre para diferenciar diferentes proyectos de un mismo producto. Por esta razón este dato es obligatorio.
- Duración del video – Es el tiempo en minutos y segundos que dura el video. En *Shotgun* se convierte

y almacena en segundos.

- Fecha – Refleja el día en que el video fue masterizado.
- Tipo – Indica en que etapa del proyecto o la versión que se masteriza. Puede ser:
  - *Master*
  - *Genérico*
  - *Intergenérico*
  - *Agencia*
  - *Director*
  - *WIP* (Trabajo de Proceso)
- Categoría – Representa el tipo de trabajo realizado, se puede seleccionar más de una. Las categorías que se utilizan son:
  - *3D*
  - *Motion Graphics*
  - *VFX* (Efectos Visuales)
  - *Adaptación*
  - *Online*
  - *Master*

Adicionalmente a estos datos en *Shotgun* también se tendrán que registrar lo siguiente:

- Sala de creación – Es el nombre de la sala de edición en la que se hizo la *pizarra*. Se cuenta con tres salas que son: *Flint* 1, *Flint* 2 y Smoke.
- Usuario – Es el nombre de la persona que creo la *pizarra*.

- Fecha de creación – Es la fecha en que a *pizarra* fue creada.
- Nombre de la *pizarra* – Es la forma de identificar a la *pizarra*, se formara de la siguiente manera: id_producto_versión_duraciónensegTipo_formato. Esto asegura que cada nombre sea único.
- Id – Es un número consecutivo, que servirá como identificador de la *pizarra* dentro de *Shotgun*, y es el dato que se codificara usando *qrcode*.
- Formato – Se refiere a la resolución de la *pizarra*, que puede ser SD o HD.

Después que se creo la *pizarra*, esta se colocara al inició de cada video y se presentará al cliente. Para almacenar el video, se quita toda la *pizarra* exceptuando los últimos dos cuadros, los cuales contienen el *qrcode*, y son almacenados en el servidor.

- **csConsecutivo:**

Este programa estará a cargo de filtrar y procesar los archivos de video que cumplan con ciertas especificaciones de formato y cuya *pizarra* haya sido creada con csSlateCreator. Su tarea consistirá en descodificar el *qrcode* y obtener los datos de la *pizarra* que se encuentran en *Shotgun* y se tomará el nombre de la *pizarra* para renombrar el archivo, para posteriormente identificar si esta bien codificado, para que el archivo pueda ser procesado y almacenado.

La codificación que tiene que contener el video es la siguiente:

- Video:
  - Formato sin compresión ya sea *rgb*24 o *YUV* 4:2:2.
  - La resolución de 720x486 pixeles para SD o 1920x1080 pixeles para HD.
- En caso de que contenga audio deberá de cumplir lo siguiente:
  - 1 sola pista de audio.
  - Codificación en *pcm*_alaw.
  - Frecuencia de muestreo de 48.0 o 44.1 kHz.

- 16 bits de profundidad
- 2 canales de audio.

Si el video en cuestión logra superar todas estas pruebas, se transcodificara a 3 versiones diferentes utilizando presets de *x.264*. Estas versiones son:

- Large – Se utilizara "*lossless*" compression, que se encuentra definido en el *preset* libx264-csLossless. Este tipo de compresión esta diseñado para obtener la máxima compresión posible sin perder calidad, aunque en este caso si existe una pequeña reducción de calidad debido al cambio de *espacio de color*. Las características técnicas de este formato son las siguientes:
  - Resolución: 720x486 pixeles.
  - *Espacio de color*: *YUV* 4:2:0.
  - Audio: Misma codificación que el original.
- Medium – Para esta versión se utiliza una compresión en alta calidad, que se encuentra definida por el *preset* libx264-hq. EL tamaño del archivo es más pequeño por lo que su manejo es más sencillo y mantiene una buena calidad.
  - Resolución: 640x480 pixeles.
  - *Espacio de color*: *YUV* 4:2:0.
  - Audio: *AAC* (Advanced Audio *Codec*).
  - Frecuencia de muestreo: 44.1 kHz
  - Bit rate: 128 kbps.
- Small – Esta versión se utiliza una compresión en calidad standard que se encuentra definida en el *preset* libx264-csSQ. Esta versión es ideal para ser enviada por correo electrónico y para vistas previas.
  - Resolución: 320x240.

- *Espacio de color*: *YUV* 4:2:0.
- Audio: *AAC* (Advanced Audio *Codec*).
- Frecuencia de muestreo: 22.05 kHz
- Bit rate: 96 kbps.

Una vez terminada la transcodficación, se copiaran los vídeos resultantes en otro servidor, con una estructura de subdirectorios ya establecida. Estos subdirectorios están organizados por números los cuales representan el número de cinta de video en el que hayan sido archivados. Esto permitirá a un programa en PHP, que ya existía, mostrar los vídeos en *Shotgun* en la página del proyecto correspondiente.

3.2.2 Herramientas de desarrollo

Como columna vertebral del estudio y del sistema se tiene *Shotgun*, por lo que cada proyecto que se realice tiene que estar conectado y ser administrado por este sistema. *Shotgun* cuenta con API en dos lenguajes C++ y Python, siendo la segunda la oficial. El Api de Python se usa para lo que llaman CRUD(create, read, update. delete).

Para el procesamiento de imágenes se tienen las opciones de ImageMagick, GraphicsMagick y el módulo Image de Python. Se opto por ImageMagick 6.7.1-1 y GraphicsMagick 1.3.12, que su primera versión se deriva de Imagemagick 5.5.2, ya que ambos sistemas cuentan con más opciones para procesar imágenes que el módulo de Python. La razón por utilizar dos sistemas en lugar de uno, es porque después de realzar pruebas de eficiencia, en algunos comandos ImageMagick es más rápido que GraphicsMagick y en algunos otros el resultado era el contrario, por lo que para algunos comandos se utilizara ImageMagick y para otros GraphicsMagick.

En el estudio se utiliza *ffmpeg* para la codificación de imágenes, audio y video. No se pensó en otra opción ya que es una herramienta libre y bastante poderosa que puede ser fácilmente modificada para realizar las tareas que necesitamos. Esta herramienta nos da la posibilidad de convertir secuencias de imágenes en video, el cual será útil para generar las pizarras, y también se puede convertir un video

en secuencia de cuadros, puede ser completo o un número n de cuadros, y se usará al momento de verificar que sea un archivo de vídeo válido.

Para llevar un control interno y sea más fácil y rápido el procesamiento de los archivos se utilizará el módulo externo de QR code para Python (pyQR Code). Con este módulo se busca tener una forma de identificar las pizarras que fueron realizadas con el sistema y así poder procesarlas si logran superar los filtros posteriores.

Se utilizará Python como lenguaje ya que el API oficial de *Shotgun* y todas las herramientas que se utilizan en el estudio cuentan con API oficial en el mismo lenguaje, lo cual nos dará más experiencia en el uso del lenguaje y los proyectos subsecuentes serán desarrollados de una manera más rápida. Python cuenta con un módulo que nos permite la utilización de comandos de sistema, esto se puede hacer de dos maneras: utilizando Bash o código de Python. Se usarán ambas opciones, ya que en Python no se encuentran todas las opciones que en Bash si existen, pero para los comandos que si existen en ambas opciones es más eficiente utilizar la implementación de Python.

Es necesario que cada máquina en la que se generaran las pizarras se cree una llave de confianza, lo cual agilizará el envío de los archivos de vídeo entre las estaciones de trabajo y el servidor, además de que fue necesario realizar pruebas de procesamiento remoto, dando como resultado que la eficiencia de hacerlo de manera local es superior a realizarlo de forma remota y más si se tiene en cuenta que el servidor tiene una carga de trabajo variable. De la misma manera, se montarán por *NFS* las carpetas del servidor donde se guardarán los machotes para las pizarras, así como los vídeos que se generen con la herramienta.

El programa que se encargará de la realización de las pizarras tendrá que llevar una interfaz gráfica, para la cual se utilizará y su implementación para Python, PyQt. De esta forma el desarrollo de la interfaz gráfica y la herramienta se podrán hacer de manera independiente, lo que quiere decir, que si la herramienta o la interfaz necesitan algún cambio o ajuste, se pueden realizar sin afectar al otro. Además la implementación de la interfaz con la herramienta se da de manera sencilla.

La herramienta de csSlateCreator será utilizada en Mac OS X, por lo que para hacer más fácil su

distribución y su uso, se creara su .app. Para poder hacer esto se cuenta con py2app y pyinstaller. En un principio se opto por py2app, pero debido a que cuenta con algunas limitaciones al momento de empaquetar los archivos, se decidió utilizar pyinstaller, con el cual no se tiene ningún problema al empaquetar y ejecutar la aplicación.

En cuanto al de csConsecutivo, se ejecutará diariamente a cierta hora, por lo que no es necesario crear una interfaz gráfica ni una forma especial de distribución. Solo será necesario que en el servidor donde se ejecutará estén instalados los módulos que son necesarios para su ejecución.

El control de versiones se llevará con un repositorio en *GitHub* para cada aplicación. De esta forma se garantiza que se tendrán almacenadas todas la revisiones que se han hecho de los programas y en caso de ser necesario regresar a alguna de ellas si algún cambio provoco un error. Esto también ayudará a llevar en *Shotgun* un registro de las revisiones y releases que se han hecho de cada herramienta para que tanto los desarrolladores como los usuarios sepan en que se ha estado trabajando y mejorando cada herramienta.

3.3 Diseño y codificación

3.3.1 csSlateCreator

Basándose en el análisis previo y el objetivo del sistema, diseñe el diagrama de flujo que se muestra en la figura 7.

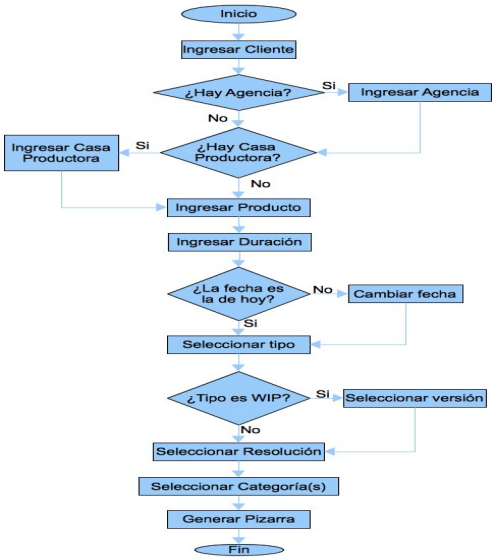

Figura 7: Diagrama de Flujo csSlateCreator

Teniendo el diagrama, lo siguiente que hice fue diseñar la interfaz gráfica. En la figura 8 se muestra la interfaz se uso para la versión 1.0.0 de la herramienta.

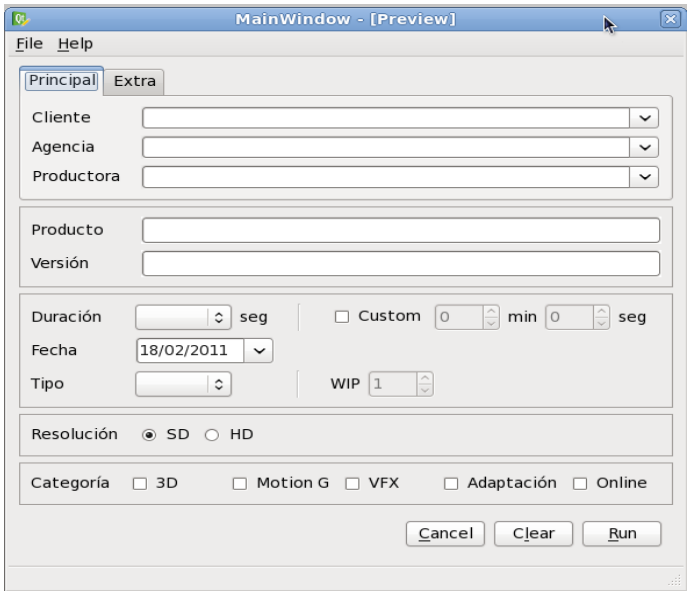

Figura 8: GUI csSlateCreator

Con base en esta interfaz gráfica y el diagrama de flujo, comencé a codificar la herramienta. Para empezar, creé un registro del script en *Shotgun*, de esta forma la herramienta puede conectarse con el sistema para obtener y registrar datos, además de poder utilizar la funcionalidad extra que nos da el API de *Shotgun*. Esto se realiza de la siguiente forma:

```
sg = Shotgun( "https://mysite.shotgunstudio.com", "csSlates","##############")
sgScript = { 'type': 'ApiUser', 'id': 101 }
```

El último argumento que recibe *Shotgun* es una clave que da el programa, la cual por cuestiones de privacidad, no es incluida. Ahora siempre que quiera utilizar algún método de *Shotgun* simplemente utilizo sg seguido por un punto y el método que necesite.

Posteriormente, cuando se ejecuta la herramienta hice que se mostrara una barra de progreso para que el usuario sepa que la herramienta esta cargando, ya que el tiempo de espera entre que se solicita su ejecución y que aparezca la interfaz puede ser largo, ya que depende de la disponibilidad de *Shotgun* y de la red interna. Durante este intervalo el sistema registra el tiempo en el que inicio la

ejecución, servirá para llevar un registro del tiempo que tardan en usar la herramienta, de *Shotgun* se obtienen todas las compañías y se organizan para aparecer en la interfaz dependiendo de su tipo y se obtiene y se muestra en la interfaz el nombre completo del usuario, para esto obtengo el usuario conectado en la computadora donde ese esta ejecutando el programa y en *Shotgun* busco su nombre, este dato lo utilizare para llevar una estadística por usuario del uso de la herramienta y también para saber quien genero la *pizarra*. A continuación se muestra la clase que obtiene al usuario y el método para obtener las compañías.

```
class checkUserSG( QtCore.QThread ):
    #Class used to check username before display it on Gui
    #Qt Thread using, not python threads!

    def __init__( self, parent = None ):
        QtCore.QThread.__init__( self, parent )
        self.user =  getpass.getuser()

    def run( self ):
        global sgUser, sCord, aUser
        sgUser = sg.find_one( "HumanUser", [ [ "login", "is", self.user ] ], [ "name" ] )
        if sCord == 1:
            self.user = aUser
        else:
            if sgUser is not None and  sgUser.has_key( "name" ):
                self.user = sgUser[ "name" ]
            else:
                self.user = ""
        self.emit( QtCore.SIGNAL( "Activated( Qstring )" ), self.user )

    #Method to get the options for the 3 first fields
    def getDictionary( self ):
        #Connection to Shotgun to get the data needed for the comboboxes
        try:
            filters = [ [ 'sg_type', 'is', 'Brand (Marca)' ], [ 'sg_type','is','Agency' ], [ 'sg_type','is','Production House' ] ]
```

```
        sgDict = sg.find( "CustomNonProjectEntity01", filters, [ 'code', 'sg_type' ], [ { 'field_name':'code',
'direction':'asc' } ], 'in' )
        for clients in sgDict:
            if clients[ 'sg_type' ] == 'Brand (Marca)':
                self.brands.append( clients[ 'code' ] )

            elif clients[ 'sg_type' ] == 'Agency':
                self.agencies.append( clients[ 'code' ] )

            elif clients[ 'sg_type' ] == 'Production House':
                self.houses.append( clients[ 'code' ] )

        self.companies = self.agencies + self.brands + self.houses
        self.companies.sort( cmp = None, key = None, reverse = False)

    except IOError:
        print "dictionary not in anticipated location"
```

Teniendo todo esto, la barra de progreso desaparecerá para dar lugar a que se muestre la interfaz gráfica. Aquí el usuario ingresara los datos que aparecerán en la *pizarra*. Para evitar que el usuario ingrese algún dato que no este registrado en *Shotgun* o que no ingrese algún dato obligatorio, en momento que le da crear, se verifica que todos los campos obligatorios hayan sido llenados, en caso contrario aparece un mensaje mencionando que faltan datos. Si todo esta bien, se procede a la creación de la *pizarra*, se revisa si los datos con respecto a los *safeties* y en caso de ser necesarios son recorridos hacia el centro los pixeles necesarios para que se encuentren dentro de los límites. Teniendo estos datos se pegan los datos con las imágenes de las pizarras y se agrega el *QrCode* en la última imagen. El siguiente método es el que se encarga de la creación de las pizarras.

```
#Method to create the images and the video for the slate
    def ivCreation( self, selType ):

        #Changeable options for imagemagick
        fill = "black"
        font = "Helvetica"
```

```
pointSizeSD = "20"
pointSizeHD = "35"
fillType = "red4"
pointSizeSDType = "43"
pointSizeHDType = "60"
size = [ "", "", "", "", "", "" ]

product1 = self.getName( self.product )
version1 = self.getName( self.version )

self.version = "%s" % self.version
self.product = "%s" % self.product

sgBrand = [ "", "" ]
sgAgency = [ "", "" ]
sgProdHouse = [ "", "" ]

#filling of slates shotgun page
proy = sg.find_one( "Project", [ [ "name", "is", "Consecutivo" ] ] )

sgBrand[ 0 ] = sg.find_one( "CustomNonProjectEntity01", [ [ 'code', 'is', self.auxBrand[ 0 ] ] ], [ 'code' ] )

if self.auxBrand[ 1 ] != "":
   sgBrand[ 1 ] = sg.find_one( "CustomNonProjectEntity01", [ [ 'code', 'is', self.auxBrand[ 1 ] ] ], [ 'code' ] )

if self.auxAgency[ 0 ] != "" :
   sgAgency[ 0 ] = sg.find_one( "CustomNonProjectEntity01", [ [ 'code', 'is', self.auxAgency[ 0 ] ]  ], [ 'code' ] )

   if self.auxAgency[ 1 ] != "":
      sgAgency[ 1 ] = sg.find_one( "CustomNonProjectEntity01", [ [ 'code', 'is', self.auxAgency[ 1 ] ]  ], [ 'code' ] )

if self.auxProdHouse[ 0 ] != "":
   sgProdHouse[ 0 ] = sg.find_one( "CustomNonProjectEntity01", [ [ 'code', 'is', self.auxProdHouse[ 0 ] ] ],
```

```
[ 'code' ] )
        if self.auxProdHouse[ 1 ] != "" and self.auxProdHouse[ 1 ] != 'Cluster Studio':
            sgProdHouse[ 1 ] = sg.find_one( "CustomNonProjectEntity01", [ [ 'code', 'is', self.auxProdHouse[ 1 ] ] ],
[ 'code' ] )

    type = selType[ 0 : 3 ]

    name = product1 + "_" + version1 + "_" + str( self.totTime ) + type + "_" + self.resolution

    if selType != ( "WIP " + str( self.wip ) ):
        type = selType
        self.wip = ""

    elif selType == ( "WIP " + str( self.wip ) ):
        type = "Work In Progress"
        name = product1 + "_" + version1 + "_" + str( self.totTime ) + "Wip" + str( self.wip ) + "_" + self.resolution

    host = commands.getoutput( 'hostname' )

    if host == 'rumble.clusterstudio.com':
        host = 'Flint 2'
    elif host == 'gears.clusterstudio.com':
        host = 'Flint 1'
    elif host == 'finalcut.local':
        host = 'Smoke'
    else:
        host = 'Dev'
    data = { 'code':name, 'sg_client':sgBrand[ 0 ], 'sg_product':self.product, 'sg_version':self.version,
'sg_duration':self.totTime, 'sg_date':self.ct, 'sg_format':self.resolution, 'sg_type':type, 'sg_wip':str( self.wip ),
'sg_3d':bool( self.category[ 0 ] ), 'sg_motion_graphics':bool( self.category[ 1 ] ), 'sg_vfx':bool( self.category[ 2 ] ),
'sg_adaptation':bool( self.category[ 3 ] ), 'sg_online':bool( self.category[ 4 ] ), 'sg_master':bool( self.category[ 5 ] ),
'sg_online_suite':host, 'created_by':sgUser, 'project':proy }
    execute = sg.create( "Slate", data )
    id = execute[ "id" ]
    name = str( id ) + "_" + name
```

```
self.ui.slateNumSpinBox.setValue( id )

if self.minutes > 0 and self.seconds > 0:
    time = unicode( self.minutes ) + " min " + unicode( self.seconds ) + " seg"
elif self.minutes > 0 and self.seconds == 0:
    time = unicode(self.minutes ) + " min"
elif self.minutes == 0:
    time = unicode( self.seconds ) + " seg"

#Imagemagick commands
if selType != ( "WIP " + str( self.wip ) ):
    os.mkdir( '/.tmp/Pizarras/' + name )

    if selType == 'Generico':
        selType = u'Genérico'
    elif selType == 'Intergenerico':
        selType = u'Intergenérico'

    if self.resolution == "SD":
        size = self.getSize( fill, font, pointSizeSD, self.resolution, id, product, version )
        self.slateData( font, fill, pointSizeSD, size, product, version, time, fillType, pointSizeSDType, selType, id, 'white' )

        if int( self.fr ) == 29:
                self.createSlate( 'SD', 30, 243 )
        elif int( self.fr ) == 23:
                self.creatSlate( 'SD', 24, 194 )

      elif self.resolution == 'HD':
          size = self.getSize( fill, font, pointSizeHD, self.resolution, id, product, version )
        self.slateData( font, fill, pointSizeHD, size, product, version, time, fillType, pointSizeHDType, selType, id, 'white' )

        if int( self.fr ) == 29:
                self.createSlate( 'HD', 30, 181 )
```

```
        elif int( self.fr ) == 23:
                self.creatSlate( 'SD', 24, 146 )
    else:
        fill = "white"
        pointSizeSDType = "60"
        pointSizeHDType = "150"

        if self.resolution == "SD":
            size = self.getSize( fill, font, pointSizeSD, self.resolution, id, product, version )
            self.slateData( font, fill, pointSizeSD, size, product, version, time, fill, pointSizeSDType, selType, id, 'black' )

        elif self.resolution == "HD":
            size = self.getSize( fill, font, pointSizeHD, self.resolution, id, product, version )
            self.slateData( font, fill, pointSizeHD, size, product, version, time, fill, pointSizeHDType, selType, id, 'black' )
        cmd = "/opt/local/bin/gm convert /.tmp/" + str( id ) + ".tga /.tmp/Pizarras/" + name + ".tga"
        subprocess.call( cmd, shell = True )
        cmd = "/opt/local/bin/gm convert -resize 640x360 -quality 60 /.tmp/" + str( id ) + ".tga /.tmp/" + name + ".jpg"

        subprocess.call( cmd, shell = True )
    cmd = "rsync -av /.tmp/Pizarras/ server:/Pizarras/"
    subprocess.call( cmd, shell = True )

    thum = "/.tmp/" + name + ".jpg"
    sg.upload_thumbnail( "Slate", id, thum )

    cmd = "rm -rf /.tmp/Pizarras/*"
    subprocess.call( cmd, shell = True )

    cmd = "rm /.tmp/" + name + ".jpg"
    subprocess.call( cmd, shell = True )
```

Con este método la *pizarra* ha sido creada y registrada en *Shotgun*, por lo que al usuario le aparece un mensaje de que la *Pizarra* fue creada con éxito y se registra el momento en que se termino

la creación, al cual se le resta el tiempo de inicio y así se obtiene el tiempo total de ejecución de la herramienta, el que es registrado en *Shotgun*. Aquí el usuario puede cerrar la aplicación o generar una nueva *pizarra*. Si elige la segunda opción se repite el proceso antes mencionado.

3.3.2 csConsecutivo

Nuevamente para esta herramienta diseñe el diagrama de flujo que tenía que seguir. Este diagrama se muestra en la figura 9. Como mencione en el análisis, para esta herramienta no es necesario crear una interfaz gráfica, por lo que se desarrollara como un script que se ejecutará en consola.

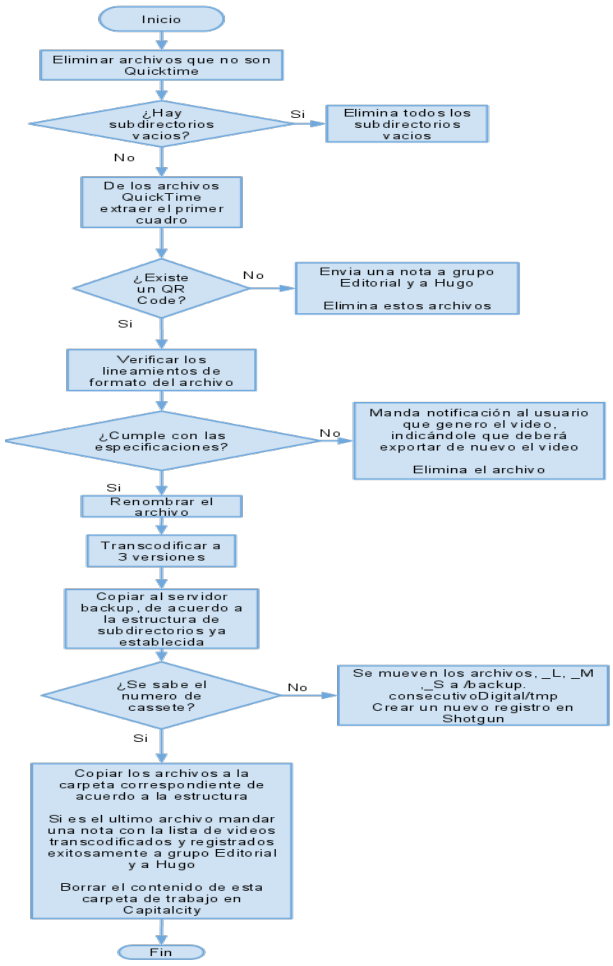

Figura 9: Diagrama de flujo csConsecutivo

Para mayor facilidad al momento de darle mantenimiento y hacerle cambios a la herramienta, decidí dividirlo en dos módulos. El primero, csConsecutivo.py, será el encargado de verificar que los archivos almacenados en el servidor cumplan con todos los requisitos para ser procesados. Mientras que el segundo, csDigitalArchive.py, se encargara de realizar la *transcodificación* y este será una "traducción" del código que ya se tenía en Bash a Python.

3.3.2.1 csConsecutivo.py

Como en csSlateCreator, este módulo se inicia con la conexión a *Shotgun*. Simplemente se necesita una línea para hacer esto que es la siguiente:

```
sg = Shotgun( "https://mysite.shotgunstudio.com",  "csConsecutivo", "########" )
```

Posteriormente se borran todos los archivos ocultos y se comienza a verificar todo el contenido de la carpeta elegida. En el momento que encuentre un directorio, verifico que no este vacío y toma su ruta absoluta para verificar su contenido, y si encuentra un directorio se repite el proceso y si es un archivo verifica que sea mov y que cumpla las especificaciones, y si el directorio esta vacío, es eliminado. El método encargado de ese proceso se muestra a continuación.

```
def __init__( self ):

    self.path = '/consecutivo'
    cmds.getoutput( 'rm -rf ' + self.path + '/.*' )
    files = cmds.getoutput( 'ls -R ' + self.path )
    self.newPath = ''

    for item in files.split( "\n" ):
        if str( item ).endswith( ":" ):
            item = str( item ).replace( ':', '/' )
            if len( os.listdir( item ) ) == 0:
                os.rmdir( item )
            else:
                self.newPath = item
```

```
    elif os.path.isfile( self.newPath + item ):
        file = str( item ).split( '.' )
        if file[ len( file ) - 1 ] == 'mov':
            item = self.newPath + item
            self.checkMov( item )
        else:
            os.remove( self.newPath + item )
```

En caso de que encuentre un mov, llama un nuevo método que es el que se encarga de verificar que cumpla con todos los requerimientos de codificación. Lo primero es obtener los primeros dos cuadros del video con el objetivo de verificar que cuente con un *QrCode* que nos indica que la *pizarra* haya sido realizada con csSlateCreator y en caso de no encontrar el código se elimina el archivo y manda un mensaje a todos los usuarios de que ese archivo no cumple con ese requisito. Si pasa esa prueba, sigue verificar su codificación y en caso de que cumpla con todas las especificaciones el video es movido a la raìz de la carpeta y transcodificado, en caso de que no cumpla algún requerimiento, el archivo es eliminado y se envía un mensaje a su creador para que lo arregle. El código que se muestra a continuación es el encargado de hacer lo antes mencionado.

```
def checkMov( self, fileMov ):
    self.id = '0'
    cmds.getoutput( 'ffmpeg -vframes 2 -ss 0.001 -i ' + fileMov + ' -f image2 ' + self.path + '/mov%02d.png' )
    cmds.getoutput( 'convert -resize 640x480\! -quality 60 ' +  self.path + '/mov01.png ' + self.path + '/mov01.png' )
    cmds.getoutput( 'convert -resize 640x480\! -quality 60 ' +  self.path + '/mov02.png ' + self.path + '/mov02.png' )
    qrCode = self.qrDecoder( self.path + '/mov01.png' )
    qrCode = self.qrDecoder( self.path + '/mov02.png' )

    if not qrCode:
        os.remove( fileMov )
        self.badMov.append( fileMov )

    elif qrCode:
        video = resolution = widthHD = heightHD = widthSD = heightSD = audioChan = audioCodec = audioRate =
```

```
audioDepth = channels = audio = False
        metadata = cmds.getoutput( 'mediainfo ' + fileMov )

        if 'raw' in metadata or 'YUV' in metadta or '2vuy' in metadata:
            video = True
        if '720 pixels' in metadata:
            widthSD = True

        if '486 pixels' in metadata:
            heightSD = True
        if '1920 pixels' in metadata:
            widthHD = True
        if '1080 pixels' in metadata:
            heightHD = True
        if 'Audio' in metadata:
            audioChan = True
        if 'Audio #1' in metadata:
            audioChan = 'error'
        if 'A-Law' in metadata or 'alaw' in metadata or 'PCM' in metadata and audioChan == True:
            audioCodec = True
        if '48.0 KHz' in metadata and '44.1 KHz' in metadata and audioChan == True:
            audioRate = True
        if '16 bits' in metadata and audioChan == True:
            audioDepth = True
        if '2 channels' in metadata and audioChan == True:
            channels = True
        if audioChan == True and audioCodec == True and audioRate == True and audioDepth == True and channels
== True:
            audio = True
        if heightHD == True and widthHD == True or heightSD == True and widthSD == True:
            resolution = True

        fields = [ 'code', 'created_by' ]
```

```
sgSlate = sg.find_one( "Slate", [ [ 'id','is', int( self.id ) ] ], fields )
if video == True and audio != 'error' and resolution == True:
    name = str( sgSlate[ 'code' ] ).split( '_' )
    name = name[ 0 ] + '_' + name[ 1 ] + '_' + name[ 2 ] + '.mov'

    os.rename( fileMov, self.newPath + name )
    cmds.getoutput( 'mv ' + self.newPath + name + " " + self.path + '/' + name )
    csDigitalArchive.compress( self.id )

else:
    os.remove( fileMov )
    proy = sg.find_one( "Project", [ [ "name", "is", "consecutivo" ] ] )
    sg.create( 'Note', { "subject":'Mala codificacion del archivo ' + fileMov,'content': "El archivo " + fileMov + ' fue eliminado por no cumplir con los requerimientos. Favor de volverlo a exportar.','addressings_to':[ sgSlate[ 'created_by' ] ], "project":proy } )
```

3.3.2.2 csDigitalArchive.py

Si el archivo de video logra superar todos los filtros, esté se transcodificara y este módulo es el encargado de esta tarea. Aquí se obtiene nuevamente la ruta absoluta del video y se comienza con la *transcodificación*, se verifica si el video tiene una pista de audio y en caso afirmativo se selecciona su codificación para cada versión que se realizará y en caso de que no existan sus directorios correspondientes se crean. Para finalmente realizar la *transcodificación* y cuando termina son movidos a otra carpeta y envía un mensaje de que la *transcodificación* fue realizada con éxito. De esta manera termina el proceso y regresa a csConsecutivo.py, para que siga procesando los directorios y archivos que faltan. El código de este método se muestra a continuación.

```
def compress( id ):
  sg = Shotgun( "https://mysite.shotgunstudio.com", "csConsecutivo","##############" )
   proj = sg.find_one( "Project", [ [ "name", "is", 'consecutivo'] ] ) # Diccionario para poder ligar al proyecto de consecutivo
   # Get start hour and date
   sdate = strftime( "%Y-%m-%d %H:%M:%S" , gmtime())
   actualPath = os.getcwd() + '/consecutivo'
```

```
for fileMov in filesMov:
    messageErrorSC = 'Transcoding of "{mov}" videos finished... \n'.format( mov = fileMov )
    messageErrorSC += 'Videos in {pwd} /compressed/ \n'.format( pwd = actualPath )

    # get the filename without extension
    bn = fileMov.split( '.' )[0]

    typePath = bn
    while not typePath[-1].isdigit():
        typePath = typePath[ 0: len(typePath)-3]

    # get the file date
    #fd = time.ctime(os.path.getmtime(fileMov))
    st = os.stat(actualPath + '/' + fileMov)
    atime = st[7] #access time
    mtime = st[8] #modification time
    if 'Audio' in commands.getoutput('mediainfo {fileM}'.format( fileM = fileMov )):
        an=""
        aparmsL="-acodec pcm_alaw -ar 48000"
        aparmsM="-acodec aac -strict experimental -ar 44100 -ab 128k"
        aparmsS="-acodec aac -strict experimental -ar 22050 -ab 96k"
    else:
        an="-an"
        aparmsL=""
        aparmsM=""
        aparmsS=""

    # If subdirectory with the file name doesn't exist, create it
    if not os.access( '{directory}/compressed/{tp}'.format( directory = actualPath, tp = typePath ) , os.F_OK):
        os.mkdir( '{directory}/compressed/{tp}'.format( directory = actualPath , tp = typePath ) )

    # If subdirectory with the file name doesn't exist, create it
```

```
    if not os.access( '{directory}/compressed/{tp}/{basename}'.format( directory = actualPath , tp = typePath , basename = bn)  , os.F_OK):
        os.mkdir( '{directory}/compressed/{tp}/{basename}'.format( directory = actualPath, tp = typePath , basename = bn) )

    # Transcode to x.264 lossless
    subprocess.call('ffmpeg -y -benchmark -i {directory}/{inputmov} {noaudio} -vpre libx264-csLossless -vcodec libx264 {apL} -threads 0  \
    -me_method umh \
    -metadata title="{basename}_L" \
    -metadata author="{cs}" \
    -metadata album="{a}" \
    {directory}/compressed/{tp}/{basename}/{basename}_L.mov \
    1>{directory}/bench 2>{directory}/ffout'.format( inputmov = fileMov , noaudio = an , basename = bn , apL = aparmsL ,cs = cluster ,
                                a = album , directory = actualPath , tp = typePath) , shell=True )

    # Check if file log.txt has size greater than zero
    # If not, transcode failed, then write failed video and error to errors.txt
    errorsLog('{basename}_L'.format(basename = bn))
    deleteLogs()

    # Transcode to x.264 two pass high quality 640x480
    subprocess.call('ffmpeg -y -benchmark -i {directory}/{inputmov} {noaudio} -vpre libx264-hq -vcodec libx264 -crf 17 {apM} -s vga -threads 0  \
    -metadata title="{basename}_M" \
    -metadata author="{cs}" \
    -metadata album="{a}" \
    {directory}/compressed/{tp}/{basename}/{basename}_M.mov \
    1>{directory}/bench 2>{directory}/ffout'.format( inputmov = fileMov , basename = bn, noaudio = an ,apM = aparmsM , cs = cluster , a = album ,
                                directory = actualPath , tp = typePath) , shell=True )

    # Check if file log.txt has size greater than zero
    # If not, transcode failed, then write failed video and error to errors.txt
```

```
        errorsLog('{basename}_M'.format(basename = bn))
        deleteLogs()

        # Transcode to x.264 two pass standard quality 320x240
        subprocess.call('ffmpeg -y -benchmark -i {directory}/{inputmov} {noaudio} -vpre libx264-csSQ -vcodec libx264 -crf 17
{apS} -s qvga -threads 0  \
        -metadata title="{basename}_S" \
        -metadata author="{cs}" \
        -metadata album="{a}" \
        {directory}/compressed/{tp}/{basename}/{basename}_S.mov \
        1>{directory}/bench 2>{directory}/ffout'.format( inputmov = fileMov , basename = bn, noaudio = an ,apS =
aparmsS , cs = cluster , a = album ,
                                directory = actualPath , tp = typePath) , shell=True )

        # Check if file log.txt has size greater than zero
        # If not, transcode failed, then write failed video and error to errors.txt
        errorsLog( '{basename}_S'.format( basename = bn ) )
        deleteLogs()

        ### Mueve los archivos comprimidos a la ruta establecida
        output = subprocess.call('rsync -av {directory}/compressed/{tp}
server:/backup/consecutivoDigital/tmp'.format( directory = actualPath , tp = typePath ) , shell =True)

        # Get end hour and date
        edate= strftime("%Y-%m-%d %H:%M:%S", gmtime())

        messageErrorSC += '---------------------------------- \n\n'
        messageErrorSC += "Script started: {startdate} ".format( startdate = sdate) + '\n'
        messageErrorSC += "Script finished: {startdate} ".format( startdate = edate) + '\n\n'

        if output is not 0:
           messageErrorSC += 'Error: rsync could not send the compressed files \n\n'

        messageErrorH += messageErrorSC
```

```
    sgSlateCreator = sg.find_one("Slate",[['id','is', int( id )]],["created_by"])['created_by']

    sg.create( 'Note', { "subject":'Transcoding finished','addressings_to': [sgSlateCreator],
"project":proj ,'content':messageErrorSC } )


  sgCC= sg.find_one( "HumanUser",[ [ "login","is", user ] ],[ "name" ] )

  sg.create( 'Note', { "subject":'Transcoding finished','addressings_to': [sgCC], "project": proj ,'content':
messageErrorH })
```