TEC | Tecnológico de Costa Rica

Aseguramiento de la Calidad Software
IC-6831

Proof of Concept: Segmentación Temporal

Profesor: M. Sc. Saúl Calderón Ramírez

Estudiantes:

Carlos M. Girón Alas - 2014113159

Julián J. Méndez Oconitrillo - 2014121700

Daniel A. Troyo Garro - 2014073396

20 de agosto 2016.

Índice

1. Diagrama de componentes

La herramienta elegida para todo diagrama UML fue *Enterprise Architect*. Esto debido a la facilidad con la que es realizar diagramas de componentes con ella y lo completa que es al incluir soporte para cualquier tipo de diagrama UML, así como importación y exportación de proyectos en distintos entornos y lenguajes. A continuación se muestra un borrador del diagrama general de componentes del sistema:

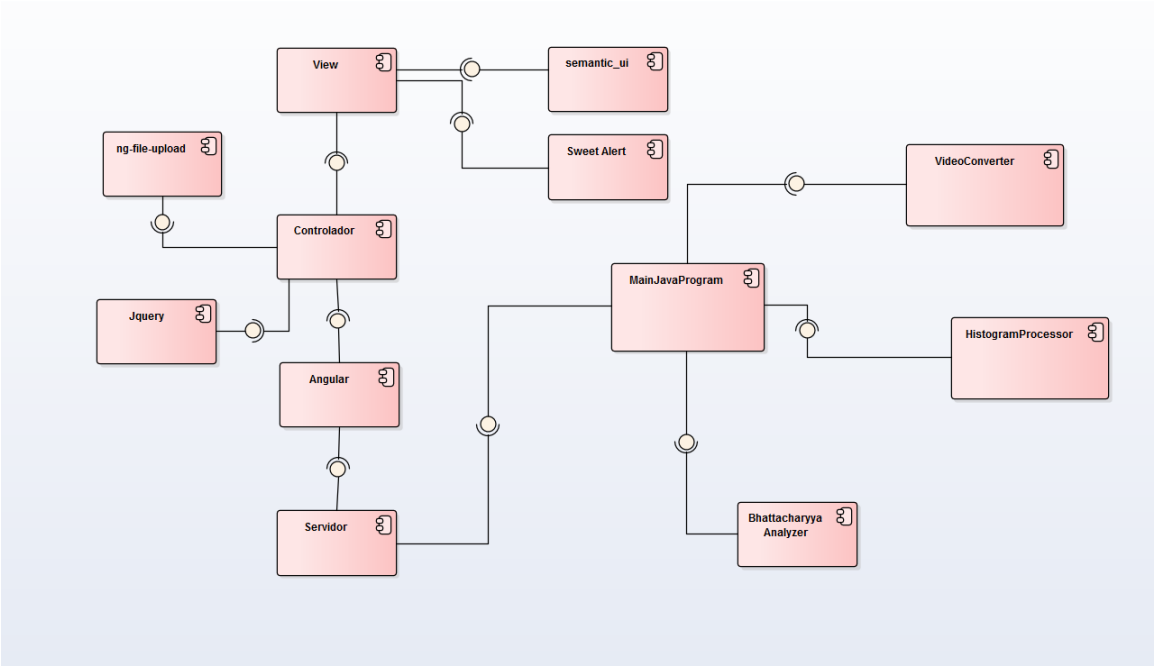

Figura 1. Diagrama de componentes general del sistema.

El front end está hecho en AngularJS, HTML5 y conectado mediante un web service que levanta WAMP server, una aplicación externa de fácil uso para las primeras etapas de desarrollo de una aplicación web. En el diagrama se ve claramente cuáles son las bibliotecas que se usan como complemento a las funciones básicas que tiene Angular, como la directiva ng-file-upload, las vistas de html se hacen de forma elegante con la ayuda de Semantic UI y los Sweet Alerts de Bootstrap, en conjunto con las animaciones de JQuery que se desarrollan en los archivos de lógica de control. En general, AngularJS y su separación de responsabilidades funciona como un interconector para las tres capas de un diseño Modelo Vista Controlador, en la que sus route providers funcionan como controladores, las vistas se asignan a cada subcontrolador, y la lógica se programa localmente en cada página o en un servicio provisto por un servidor. En este caso nos valemos de ambas posibilidades para acomodar los elementos completamente.

La parte back-end del programa consiste en la aplicación de java ejecutándose en la terminal del computador servidor, la cual recibirá el video de la aplicación web y realizará las operaciones de lectura, conversión a HSV y segmentado en frames a través del componente de **VideoConverter.** De aquí pasa a generar los histogramas a través del **HistogramProcessor** para luego operar dicho arreglo de histogramas con el **Bhattacharyya Analyzer**, que calculará la medida de disimilud entre cada histograma y la almacenará en un arreglo para que el VideoConverter pueda así generar los videos con base a los cortes.

2. Procesos implementados

En esta sección del documento se explican los procesos o funciones implementadas y la manera en la que estas fueron logradas a través de la utilización de la biblioteca OpenCV en el lenguaje JAVA. La versión utilizada de dicha biblioteca fue la 2.4.13, ya que es la que contaba con la mayor documentación así como tutoriales en la web, además de ser la ideal para la resolución de un error que se explicará más adelante. Tras esta prueba de concepto (*Proof of Concept*), se escuchará a las recomendaciones dadas por el profesor y se corregirá lo necesario para continuar con los componentes faltantes en el sistema. Los procedimientos implementados en esta prueba son:

Lectura de video: El paso inicial de este sistema consiste en la lectura de un archivo de video por el sistema. En esta prueba de concepto, únicamente se leyó archivos de formato *.mp4*, los cuales fueron proporcionados por el profesor. La lectura del video se logró mediante la utilización de la clase **VideoCapture**, que pasa a cumplir funciones de captura de una videocámara o de un archivo de video en la biblioteca. Inicialmente, ocurrieron muchos problemas con esta clase. El primero se debió a la instalación de versiones distintas de la biblioteca, ya que esta clase cambia de módulos entre versiones y al trabajar sobre el mismo proyecto en un repositorio, los comandos de *import* daban error. Por este motivo, se decidió definir como versión a utilizar la 2.4.13 puesto que posteriormente sería la recomendada para resolver el error que presentaba el comando de VideoCapture. Al leer el archivo de video con **VideoCapture.open()**, siempre devolvía valor *false*, lo que indicaba que no había ocurrido la lectura del archivo de video exitosamente. Esto fue algo que atrasó el desarrollo del proyecto por un tiempo considerable de un día y que se resolvió mediante la instalación de un codec faltante en el sistema. Este codec era el **ffmpeg**, el cual se instaló mediante la añadidura de la variable de ambiente *PATH* a su localización y también a la localización de los archivos de la biblioteca OpenCV, especificamente a la carpeta *opencv/build/java/x64/vc12/bin*. Tras realizar estos cambios, la lectura del video resultó exitosa.

División del video en frames: Al tener el video en la memoria del programa y albergado en el objeto de clase VideoCapture, se procedió a la separación de este en los cuadros o *frames* que lo componen. Dicha separación fue simple y se logró mediante un ciclo *while* que se mantiene hasta que el comando **VideoCapture.read()** de *false*, lo que indica que ya no hay más frames que extraer. Los frames extraídos son guardados en objetos de clase **Mat** o Matrices, los cuales son utilizados por OpenCV para representar cualquier clase de arreglo de n-dimensiones. Se recuerda que una imagen es un arreglo bidimensional de pixeles. Dichos objetos son albergados en una estructura de datos *LinkedList*, provista por el API de Java, y son devueltos una vez extraídos todos.

La función de lectura y separación del video en frames fueron agrupadas en una función de la clase **VideoSegmenter**, esto debido a la simpleza de su estructura y que la separación de estas en dos funciones separadas añadiría complejidad innecsaria. Si en el avance y mantenimiento del proyecto se considera conveniente separar estas dos funciones por razones justificadas, se procederá a separarlas según convenga.

Conversión de los videos al modelo de color HSV: Los frames obtenidos del video vienen en el modelo de color **BGR**, que es el modelo de color inverso del **RGB** al venir cada imagen en los canales *blue-green-red*. El modelo de color HSV (*Hue-Saturation-Value* o Tono-Saturación-Valor) es ideal para el contexto del sistema ya que es más fácil detectar cambios bruscos de un frame a otro si se evalúa el valor del histograma del valor de **Tono** de cada frame. Por este motivo, se convierte a este modelo de color utilizando la clase **Imgproc**, que cumple las funciones de procesamiento de metadatos de objetos Mat, entre los que se incluyen imágenes. La conversión se realiza obteniendo el valor de conversión de BGR a HSV, incluído en la clase, y llamando a la función **Imgproc.cvtcolor()** de la clase, la cual recibe una imagen de origen, una imagen de destino y el valor de conversión obtenido previamente. Tras lograr esta conversión, se devuelve el arreglo de imágenes ya modificadas con el modelo de color HSV.

Esta función fue implementada en la clase **VideoSegmenter**, ya que se consideró estar relacionado al procesamiento del video y presentar alta cohesión con la función de lectura y separación en frames del video.

Generación de lista de histogramas del valor H de cada frame: La generación de histogramas por cada frame del video se realiza de una manera simple al ya existir una función que lo realiza en OpenCV. Primero que todo, se debe entender que los histogramas generados son de una única dimensión (el valor H de cada pixel del frame) y de 256 *bins*, que son las agrupaciones del valor del histograma. Según documentación de OpenCV 2.4.13, el canal 0 del objeto de una imagen HSV, el cual viene a representar el valor del *Hue* ya está normalizado en el rango [0,255] por lo que no se normalizó cada objeto de imagen sino hasta el cálculo del histograma mediante la función **Imgproc.calcHist()**. A esta función se le introduce un arreglo de objetos de clase Mat, del cual calculará el histograma, el número de dimensiones que presentará y el número de bins, el cual aquí se agrupa de [0,255], indicando que en el histograma cada pixel será categorizado en 256 grupos dado el valor de su canal 0, que corresponde al valor del tono. Se realiza un ciclo iterativo *for* a través del arreglo de frames, generando un histograma similar por cada frame y almacenándolo en una estructura de datos *LinkedList*.

El arreglo de histogramas generado será posteriormente evaluado por el componente que implementará el algoritmo de disimilitud entre histogramas, lo que permitirá determinar mediante la distancia de Bhattacharyya la diferencia entre un frame a otro a través de los histogramas de su valor de *hue*. Por supuesto que este proceso será posteriormente realizado en el proyecto y no se encuentra en esta prueba de concepto.

La conexión con la aplicación web en esta prueba de concepto consiste en la carga y envío del archivo del video al servidor ejecutando el programa de java que realiza la segmentación temporal.

3. Problemas presentados

Como se ya se había explicado anteriormente, los problemas principales presentados en este proyecto se dieron con la lectura del video y el envío del mismo.

En cuanto al trabajo en Java, Eclipse nos dió un par de problemas por la instalación de OpenCV, ya que era posible poner los caminos para las librerías en el proyecto, en la pestaña Window y en el path de ejecución de los proyectos generales. Una vez que se consiguió setear correctamente los elementos de OpenCV, los ejemplos no funcionaban con videos debido a un codec de ejecución que necesitaba windows, *ffmpeg*, sin el que los videos eran ilegibles para la librería. Después de invertir un par de horas en encontrar el motivo de este problema y de encontrar que debían agregarse tanto OpenCV como ffmpeg al PATH (variable de entorno que tiene varias rutas), las pruebas de ejecución funcionaron.

Respecto a la interfaz de usuario, en AngularJS, también hubo un par de retrasos por notar. Anteriormente habíamos desarrollado proyectos con esta tecnología, pero también se combinaba con algunas otras que actualmente ya no están disponibles o son mucho menos estables hoy en día que lo que fueron cuando se usaron por primera vez, por lo que tuvimos que encontrar sustitutos a esas tecnologías. La última vez que tuvimos que programar con AngularJS fue hace más de un año, por lo que tuvimos que reencontrarnos con ello y eso nos produjo una pequeña curva de aprendizaje o de repaso si se quiere. Una vez completo el cascarón de la interfaz de usuario, comenzamos a trabajar de forma paralela dos elementos: la conexión con la aplicación de java que trabaja el video y lo convierte a histogramas, y la subida del video al servidor. Este segundo elemento tuvo un peso particular porque el código general para subir archivos está restringido por PHP.ini a cierto tamaño de subida, pero los errores relacionados a estas restricciones son ambiguos por lo que provocaron confusión. Por ejemplo, PHP informa que el error proviene de un índice mal definido en la variable global FILES, pero el error real proviene de una variable que estaba limitada a 10MB, mientras que el video de prueba era de mayor tamaño. Otros errores molestos durante el desarrollo son de los esperados por el trabajo web, como el hecho de que usar un formulario con un submit diseñado para cargar la información a un PHP redirige la página innecesariamente hacia el php mismo, cuando lo requerido era sólo que hiciera el análisis. Ese problema se resolvió usando un producto externo que se adjunta a angular para trabajar la directiva ng-file-upload.

En cuanto a JUnit, se presentaron problemas comparables con el trabajo de las bibliotecas que deben importarse para el uso de esta herramienta, pues en algunas de las computadoras que lo usamos sirve mientras que en otras no. Una vez correctamente instalados, lo problemático fue acostumbrarse al uso de esta tecnología que para todos los miembros del equipo es desconocida.

Para la conexión entre Java y la página web utilizamos un server de Java conectado mediante un Socket a una instancia de Web Socket de JavaScript, la cual también da bastantes problemas porque los puertos que estaban abriéndose a la vez, la ejecución de los programas se mantenía por lo que aparecían errores que en su momento eran confusos y en general dio problemas porque el servidor en algún momento recibe el cliente pero el cliente no recibe al servidor, por la falta del manejo adecuado del proceso de handshake que requieren los servidores con sus clientes según los estándares de seguridad.

4. Tiempos de desarrollo

Duración de desarrollo de cada componente	
Componente	**Duración en días**
VideoConverter	4 días
HistogramProcessor	3 días
Bhattacharyya Analyzer	4 días
Servidor	6 días
Controlador	5 días
View	7 días

5. Repositorio en Git

El acceso al código fuente del programa puede ser encontrado en el siguiente repositorio de Github: `https://github.com/menocsk27/ACS_SegmentTemp.git`