Universidad de las Ciencias Informáticas

Facultad 6

Título

Herramienta para la generación de clasificadores haar

Trabajo de diploma para optar por el Título de Ingeniero en Ciencias Informáticas

Autor: Orisel Caridad Leiva Scull

Tutor: Ing. Reynier Pupo Gómez

La Habana, Cuba

Curso 2012 - 2013

Libre, y para mi sagrado, es el derecho de pensar… La educación es fundamental para la felicidad social; es el principio en que descansan la libertad y el engrandecimiento de los pueblos.

Benito Juárez

Declaración de Autoría

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 6 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Orisel Caridad Leiva Scull

_____

Firma del autor

Ing. Reynier Pupo Gómez

_____

Firma del tutor

Datos de contacto

Tutor:

Nombre y apellidos: Reynier Pupo Gómez

Título: Ingeniero en Ciencias Informáticas.

Universidad de las Ciencias Informáticas, Habana, Cuba

e-mail: rgomez@uci.cu

Agradecimientos

A mi tutor Reynier Pupo Gómez por apoyarme y defenderme en los momentos en los que lo necesité.

A mi oponente Rayner Pupo Gómez por la ayuda brindada y por lo consejos dados.

A Maikel, Yunier, Javier y el Dany porque dejaron de atender sus propias tesis para ayudarme con los problemas que yo tenía con la mía.

A los profesores Reinier Pupo Ruiz, Olga Maria Rivera y Dainovis por su ayuda con el documento de tesis y las dudas con la aplicación.

A mi amigo Frank Alberto que siempre estuvo apoyándome y prestándome su ayuda incondicional y sin interés.

A todos mis compañeros de aula, los que están y los que hoy no están entre nosotros Yudelis, Sadys, Moya, Yandy, Anyelique, Linares, porque todos de alguna forma me ayudaron a que yo esté aquí hoy.

A Doina, Tamara, Andris, Vivi, Ulises, Lisandra, Elisa, Daniel, Breissy, Yunet, Tahimi, Leidy, Maralys, Deimis, las chicas de la farándula, Heriberto, Eddy Ernesto, Darcy, Elaine, Yeilen, Nathalie y todos aquellos que estuvieron conmigo en estos 5 años.

A todos mis profesores Frank Alain, Anay, Yaima, Albretch, Yuya, Alleyne, Yinet, Basulto, Nápoles, Lisbeth Olinda, Pedro Puig, Adrian Misael, Adrian, Elianis y todos aquellos que contribuyeron a mi formación como profesional.

A mi Hurshel que nunca dejó que yo me diera por vencida y que siempre me alentaba a que me secara las lágrimas y siguiera adelante yo lo iba a lograr.

Dedicatoria

Quiero dedicarle este trabajo de diploma a todas estas personas que son muy especiales para mí:

A mi mamá por haberme dado la vida y por estar junto a mí en todo momento apoyándome incluso cuando sabe que no he tomado la mejor decisión.

A mi hermano por ser el motor impulsor de mi carrera y por incitarme a dar siempre lo mejor de mí.

A mi abuela por su apoyo y su amor incondicional.

A mi papá por darme su cariño aunque este no sea de la forma convencional a la que todos estamos acostumbrados.

A mi tíos Maura, Francis, Tuti y Willy por apoyarme en estos cinco años de mi carrera en los que he pasado por momentos difíciles.

A mi familia entera por el cariño que me han brindado.

Resumen

En algunos de los productos que se desarrollan en el departamento de Señales Digitales se utiliza OpenCV que es una biblioteca de tratamiento de imágenes, destinada principalmente a aplicaciones de visión por computador en tiempo real. Esta biblioteca contiene una serie de herramientas para el procesamiento de imágenes y dentro de ellas se encuentran los clasificadores haar.

El problema con las cascadas de clasificadores haar es que resulta muy engorroso su proceso de instauración para los desarrolladores ya que cada uno de los pasos que se utilizan para ello, como crear la colección de muestras positivas y negativas, guardar cada tipo de muestra en un archivo y entrenar el clasificador, se hacen mediante comandos entrados por consola y después de pasado todo este proceso es que el desarrollador puede utilizarlas.

El presente trabajo de diploma tiene como objetivo principal implementar una herramienta que le permita al desarrollador a realizar el proceso de creación de clasificadores haar de una forma amena mediante una interfaz amigable que evite el proceso engorroso de creación por consola.

El presente documento está estructurado en cuatro capítulos que van desde las primeras etapas de la investigación hasta las pruebas que se le realizan a la herramienta para comprobar su funcionamiento.

Palabras claves: cascadas de clasificadores de haar, muestras negativas, muestras positivas, openCV, procesamiento.

Abstract

In some of the products developed in the Digital Signals Department it is used OpenCV, which is a library of image processing, intended primarily for real time computer vision applications. This library contains a set of tools for image processing and within them are the haar classifiers.

The problem with haar cascade classifiers is that the installing process is very cumbersome for developers onset since each of the steps that is used for this, such as creating the collection of positive and negative samples, save each type of sample in a file and train the classifier, these are made through console commands and after this process is that the developer can use the haar cascade classifiers.

This diploma work has as main objective to implement a tool that allows the developer to make the process of creating haar cascade classifiers in a pleasant way by a friendly interface that avoids the cumbersome process.
This document is divided into four chapters ranging from the early stages of the investigation until the tests performed to verify the tool functioning.

Key words: haar cascade classifiers, negative samples, OpenCV, positive samples, processing

Tabla de contenido

Índice de figuras

Índice de tablas

Introducción

El reconocimiento de patrones en imágenes digitales ha despertado gran interés en las últimas décadas permitiendo abordar problemas de inspección visual, de reconocimiento de números, letras manuscritas, el reconocimiento de huellas dactilares, de rostro, etc. Este reconocimiento puede ir desde ejemplos muy simples como reconocer en una imagen el único objeto de color rojo, hasta posibilidades muy complejas y útiles que aún hoy son prácticamente imposibles, como cámaras de aeropuerto que detecten terroristas automáticamente reconociendo su cara.

Este último ejemplo se puede lograr mediante la video vigilancia que consiste en la vigilancia a través de un sistema de cámaras ip, fijas o móviles (RAE, 2013). El proyecto Video Vigilancia Suria, que se encuentra ubicado en el centro de Geoinformática y Señales Digitales (GEYSED) en la facultad 6 de la Universidad de las Ciencias Informáticas centra su atención en el proceso de vídeo vigilancia que permita fortalecer la seguridad en cualquier institución donde sea desplegado. La ejecución de este proyecto incluye la infraestructura tecnológica para soportar un conjunto de cámaras de seguridad, que se puedan gestionar dentro de una red de datos asegurando la visualización, almacenamiento y trasmisión de los flujos de vídeos generados en cada uno de los dispositivos de adquisición. (UCI, 2010)

Para realizar esta tarea el proyecto Video Vigilancia Suria utiliza la biblioteca OpenCV que es una biblioteca para el tratamiento de imágenes, en aplicaciones de visión por computador en tiempo real. La biblioteca OpenCV contiene una serie de herramientas, entre ellas están los clasificadores haar que se utilizan en la detección de objetos.

Un clasificador haar (llamado cascade of boosted classifiers working with haar-like features) consiste en varios clasificadores simples que son aplicados subsecuentemente a una región de interés dentro de una imagen, hasta que en alguna etapa la imagen a procesar es rechazada, o todas las etapas son pasadas, es decir, que se obtiene el objeto que se está buscando. (Shen, 2006)

Para realizar el proceso de generación de un clasificador primero se deben de guardar una serie de imágenes del objeto que se quiera identificar. Estas imágenes se denominan muestras de entrenamiento. Existen dos tipos de muestras: positivas y negativas. Las muestras positivas corresponden a imágenes en las que el objeto se evidencia claramente y las negativas no contienen el objeto que se quiere identificar. Con estas muestras de entrenamiento se debe crear una colección que se utilizará luego en el entrenamiento del clasificador. Esta colección

se crea mediante comandos entrados por consola con la herramienta Haar Training. Después de esto se pasa a entrenar el clasificador mediante una serie de comandos específicos que se deben entrar por consola, como por ejemplo el directorio específico donde va a ser almacenado el clasificador y los nombres de los archivos correspondientes a las muestras positivas y negativas utilizadas, todo esto es realizado con la herramienta Train Cascade.

Después de explicado cómo es que se realiza el proceso de generación de los clasificadores de haar actualmente, surge como **situación problemática** que los productos que se desarrollan en el departamento de Señales Digitales no cuentan con una herramienta que automatice el proceso de generación de los clasificadores haar y como todo se realiza manualmente y mediante comandos entrados por consola se hace muy engorroso realizar todo este proceso.

Por todo lo antes planteado se define como **problema a resolver** el siguiente: ¿Cómo contribuir a la generación de clasificadores haar para los productos que se desarrollan en el departamento de Señales Digitales?

El **Objeto de estudio** se enmarca en el proceso de generación de los clasificadores haar, mientras que el **Campo de acción** queda delimitado en la generación de los clasificadores haar en el departamento de Señales Digitales. Del problema planteado se deriva como **objetivo general** desarrollar una herramienta que permita la generación de clasificadores haar para los proyectos del departamento de Señales Digitales.

Como **idea a defender** se tiene: Con el desarrollo de la herramienta para la generación de clasificadores haar, se obtendrá una forma más rápida de crear y entrenar cascadas de clasificadores para ser utilizada en el departamento de Señales Digitales.

Para dar cumplimiento al objetivo general planteado se trazaron las siguientes **tareas de la investigación:**

- ✓ Caracterizar los procesos relacionados con la generación de clasificadores haar.
- ✓ Identificar las herramientas y tecnologías que soporten la implementación de la herramienta para la generación de clasificadores haar.
- ✓ Realizar el análisis y diseño de la herramienta para la generación de clasificadores haar.
- ✓ Implementar la herramienta para la generación de clasificadores haar.

- ✓ Realizar pruebas a la herramienta desarrollada.

Posibles resultados:

- ✓ Documentación generada en el proceso de desarrollo de la herramienta para la generación de clasificadores haar.
- ✓ Herramienta para la generación de clasificadores haar.

Para la realización de este trabajo se tuvieron en cuenta los siguientes métodos científicos:

- ➢ **Analítico - Sintético:** este método permite el análisis de los elementos teóricos para la creación de la herramienta que se desea implementar, como por ejemplo la creación de la documentación que se genera, la realización de un estudio del proceso de generación de los clasificadores, permitiendo así la extracción de los elementos más importantes que se relacionan con el objeto de estudio.
- ➢ **Histórico-Lógico:** este método se utiliza para conocer los antecedentes y los elementos de la investigación que se asocian a la generación de clasificadores haar.
- ➢ **Modelación:** este método se utiliza fundamentalmente para modelar los artefactos que se generan a lo largo de todo el proceso de desarrollo de software de la herramienta, ejemplos los diagramas de casos de uso y los diagramas de despliegue.
- ➢ **Observación:** la utilización de este método permite hacer un registro visual de lo que ocurre realmente en el proceso de generación manual de los clasificadores de haar, logrando recoger la información necesaria para llevar a cabo satisfactoriamente la investigación.
- ➢ **Análisis documental:** este método permite realizar una búsqueda de información sobre el objeto de estudio y el campo de acción en documentos normativos propios de la institución como manuales y actas.

El presente trabajo consta de cuatro capítulos:

- ✓ **Capítulo 1 Fundamentación teórica del estudio de la generación de los clasificadores haar:** Incluye un estudio sobre la generación de los clasificadores haar, a nivel internacional, nacional y de universidad, además de las tendencias, técnicas, tecnologías, metodologías y software usados en la actualidad y en las que se utilizan para la solución del problema que se enfrenta.

- ✓ **Capítulo 2 Características de la herramienta para la generación de clasificadores haar:** Se describe cómo se ejecutan actualmente los procesos, las causas que originan la situación problémica y las consecuencias. Además se describen los procesos que serán objeto de automatización, la descripción general de la propuesta de sistema y todo lo referente al modelo de negocio.
- ✓ **Capítulo 3 Análisis y diseño de la herramienta para la generación de clasificadores haar:** Este capítulo se definen los elementos del modelo de análisis, los diagramas de interacción, la descripción de las clases y las definiciones de diseño que se apliquen. Además de definir la interfaz gráfica de usuario que posee el sistema.
- ✓ **Capítulo 4 Implementación y prueba de la herramienta para la generación de clasificadores haar:** Incluye los diagramas de despliegue y de componentes correspondientes a la fase de implementación de la herramienta, además de la descripción de los casos de prueba para cada uno de los casos de uso.

Capítulo 1 Fundamentación teórica del estudio de la generación de los clasificadores haar

Introducción

En este capítulo se definen los conceptos asociados a la generación de clasificadores haar, se explica detalladamente cómo es que se realiza el proceso actual de generación de los clasificadores haar, se hace un estudio del tema a nivel nacional e internacional, además de una caracterización de las tecnologías y herramientas usadas en la solución.

1.1 Conceptos asociados

Un clasificador haar (llamado cascade[1] of boosted[2] classifiers working with haar-like features) consiste en varios clasificadores simples que son aplicados subsecuentemente a una región de interés hasta que en alguna etapa el candidato es rechazado o todas las etapas son pasadas (Shen, 2006). El clasificador es diseñado para que los objetos puedan ser encontrados a diferentes escalas, o sea, en orden de ser capaz de encontrar objetos de interés a diferentes tamaños, lo cual es más eficiente que redimensionar la imagen por sí misma. Por lo tanto, para encontrar un objeto de tamaño desconocido en la imagen, la búsqueda se realiza varias veces a diferentes escalas (Mungia-Grau-Mungia, 2010). Los clasificadores haar son entrenados y utilizados en los productos que se desarrollan en el departamento de Señales Digitales para identificar objetos específicos, como por ejemplo identificar una maleta perdida en el aeropuerto. El **entrenamiento** es realizado por la herramienta Train Cascade o entrenador de cascada y consiste en pasarle una serie de comandos entre los que se encuentran: el nombre del directorio donde se va a almacenar el clasificador, el número de etapas del clasificador para ser entrenado, la memoria disponible en MB para realizar la operación y el número de ejemplos de muestras positivas y negativas usadas. Para la realización del entrenamiento es necesario crear una **colección de muestras de entrenamiento**, esta colección es creada mediante la herramienta Haar Training y está compuesta por dos tipos de muestras: las muestras positivas

[1] La palabra “cascade” en el nombre del clasificador significa que el clasificador resultante consiste en varios clasificadores simples que son aplicados subsecuentemente a una región de interés hasta que en alguna etapa el candidato es rechazado o todas las etapas son pasadas.

[2] La palabra “boosted” significa que los clasificadores correspondientes a cada etapa son a la vez complejos y están construidos de clasificadores simples usando una de las diferentes técnicas de boosting o peso por voto.

y las negativas. Las **muestras positivas** no son más que imágenes que contienen el objeto que se quiere obtener al final del proceso. Estos objetos deben de estar bien marcados. Además de eso, las imágenes deben estar escaladas al mismo tamaño. En el proyecto Video Vigilancia se escogen como muestras positivas aquellas imágenes que se encuentren en un entorno controlado, es decir, imágenes en las que no exista ningún tipo de obstáculo que impida que el objeto se evidencie claramente. Las **muestras negativas** no son más que imágenes que no contienen representaciones del objeto que se quiere obtener al final del proceso. Los ejemplos negativos son tomados de imágenes arbitrarias. En el proyecto Video Vigilancia se toman como ejemplos negativos una serie de imágenes arbitrarias, es decir, imágenes que no tienen absolutamente nada que ver con el objeto que se desea encontrar.

1.2 Proceso actual de generación de los clasificadores haar

Para realizar el proceso de generación de los clasificadores haar lo primero que se debe hacer es crear una colección de imágenes del objeto que se desea obtener al final del proceso. Ésta colección de imágenes es llamada colección de muestras de entrenamiento. Esta colección está compuesta por dos tipos de muestras: muestras positivas y muestras negativas. Después que se tienen las muestras positivas y negativas identificadas se pasa a crear un archivo de descripción para cada tipo de muestra. El archivo con las imágenes positivas contiene la dirección donde se encuentran guardadas las imágenes, la cantidad de objetos que contiene la imagen y las coordenadas de ese objeto. El archivo con las muestras negativas contendrá solamente la dirección donde se encuentran guardadas las imágenes. A continuación se pasa a crear la colección en la herramienta Haar Training, mediante comandos entrados por consola como el nombre de los archivos creados con las muestras positivas y negativas, el número de muestras positivas a crear y el largo y el ancho de las muestras de salida. Inmediatamente después se pasa a entrenar el clasificador mediante la herramienta Train Cascade. Los comandos utilizados para esto son: el directorio donde se va a guardar el clasificador, los archivos creados con las muestras positivas y negativas, el número de muestras positivas y negativas creadas y el número de etapas para ser entrenado. La herramienta devuelve un archivo de tipo xml con el clasificador entrenado.

1.3 Estudio de la generación de clasificadores haar a nivel nacional e internacional

Se realizó una búsqueda a nivel nacional e internacional en la que se encontraron las herramientas que traen predefinidas la librería OpenCV para realizar los procesos de generación de las cascadas de clasificadores haar. Estas herramientas son la Haar Training y la Train Cascade.

1.3.1 Haar Training

Esta es una aplicación que se ejecuta por consola y su función es crear la colección de muestras de entrenamiento para entrenar el clasificador de haar. Entre los parámetros que se le deben pasar para crear la colección se encuentran: el nombre con el archivo de salida que contiene las muestras positivas para el entrenamiento, el archivo de descripción de las muestras negativas, el número de muestras positivas que se van a generar, el ancho y el largo de las muestras de salida, entre otros.

1.3.2 Train Cascade

Esta es una aplicación que se ejecuta por consola y su función es entrenar la cascada de clasificadores haar. Entre los parámetros que se le deben pasar para entrenar al clasificador están: el nombre del directorio donde se va a guardar el clasificador que se va a entrenar, el nombre con el archivo de salida que contiene las muestras positivas para el entrenamiento, el archivo de descripción de las muestras negativas, el número de ejemplos positivos generados y el número de ejemplos negativos utilizados, el número de etapas que tendrá el clasificador, entre otros.

Estas herramientas realizan el proceso de generación de los clasificadores haar manualmente y con comandos pasados por consola. Ninguna de las herramientas tiene las mismas características de la herramienta que se desea implementar, es decir, que automatice el proceso de generación de los clasificadores haar todo en una sola herramienta.

1.4 Tecnologías y herramientas utilizadas en la solución

En este epígrafe se define lo que es un proceso de desarrollo de software y lo que son las metodologías de desarrollo de software. También se definen y caracterizan las tecnologías, herramientas y metodología de desarrollo de software utilizadas para la realización de este trabajo de diploma. Cabe resaltar, que todas estas tecnologías, herramientas y metodología que se señalan en este trabajo son las mismas definidas por la dirección del proyecto productivo para garantizar así la compatibilidad del sistema con los productos desarrollados en el departamento de Señales Digitales.

1.4.1 Proceso de desarrollo de software

Un proceso de desarrollo de software tiene como propósito la producción eficaz y eficiente de un producto software que reúna los requisitos del cliente. Es válido destacar que el proceso de desarrollo de software no es único. No existe un proceso de software universal que sea efectivo para todos los contextos de proyectos de desarrollo. (Letelier, 2011)

Un proceso de software está compuesto de una serie de pasos a seguir para la realización de un producto o sistema de forma eficaz y eficiente que contribuya a que el producto que se va a obtener al final de proceso tenga la calidad requerida.

1.4.2 Metodología de software

Las metodologías de desarrollo de software son el marco de trabajo que colecciona un conjunto de pasos y procedimientos que se deben seguir para organizar, controlar y planear el proceso de desarrollo de un software. Hoy en día existen significativas metodologías que brindan soluciones a los desarrolladores. Su uso posee una enorme importancia a la hora de gestionar de forma eficiente un proyecto. (Pressman, 2010)

Definir una metodología de desarrollo de software contribuye a mantener de una forma ordenada el proceso de creación del producto. Hoy en día esto resulta de gran importancia ya que permite ahorrar tiempo y dinero.

1.4.2.1 RUP

Su nombre proviene de las siglas en inglés Rational Unified Process. Se escoge RUP al ser una metodología de desarrollo de software robusta que tiene como objetivo asegurar la producción de software de alta calidad que satisfaga los requerimientos de los usuarios finales y porque genera una gran cantidad de documentación a lo largo de todo el proceso que ayuda a un mejor entendimiento del producto que se está desarrollando. Esta metodología ofrece una alta capacidad organizativa, permitiendo realizar grandes proyectos, pero a la vez es flexible por lo que permite que se ajuste a cualquier entorno de desarrollo.

El proceso de software propuesto por la metodología de desarrollo de software RUP tiene tres características esenciales: (Jacobson-Booch-Rumbaugh, 2000)

I. **Está dirigido por los Casos de Uso:** En RUP los Casos de Uso no son sólo una herramienta para especificar los requisitos del sistema. También guían su diseño, implementación y prueba. Los Casos de Uso constituyen un elemento integrador y una guía.

II. **Está centrado en la arquitectura:** La arquitectura de un sistema es la organización o estructura de sus partes más relevantes, lo que permite tener una visión común entre todos los involucrados (desarrolladores y usuarios) y una perspectiva clara del sistema completo, necesaria para controlar el desarrollo.

III. **Es iterativo e incremental:** el equilibrio correcto entre los Casos de Uso y la arquitectura es algo muy parecido al equilibrio de la forma y la función en el desarrollo del producto, lo cual se consigue con el tiempo. Para esto, la estrategia que se propone en RUP es tener un proceso iterativo e incremental en donde el trabajo se divide en partes más pequeñas o mini proyectos. Permitiendo que el equilibrio entre Casos de Uso y arquitectura se vaya logrando durante cada mini proyecto, así durante todo el proceso de desarrollo.

RUP fragmenta el proceso de desarrollo de software en cuatro etapas: Inicio, Elaboración, Construcción y Transición. Durante la fase de inicio las iteraciones hacen mayor énfasis en las actividades de modelado del negocio y de requisitos.

En la fase de elaboración, las iteraciones se orientan al desarrollo de la línea base de la arquitectura, abarcan más los flujos de trabajo de requerimientos, modelo de negocios, análisis, diseño y una parte de implementación orientado a la línea base de la arquitectura.

En la fase de construcción, se lleva a cabo la construcción del producto por medio de una serie de iteraciones.

En la fase de transición se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios. Como se puede observar en la imagen que aparece a continuación en cada fase participan todas las disciplinas, pero dependiendo de la fase el esfuerzo dedicado a una disciplina varía.

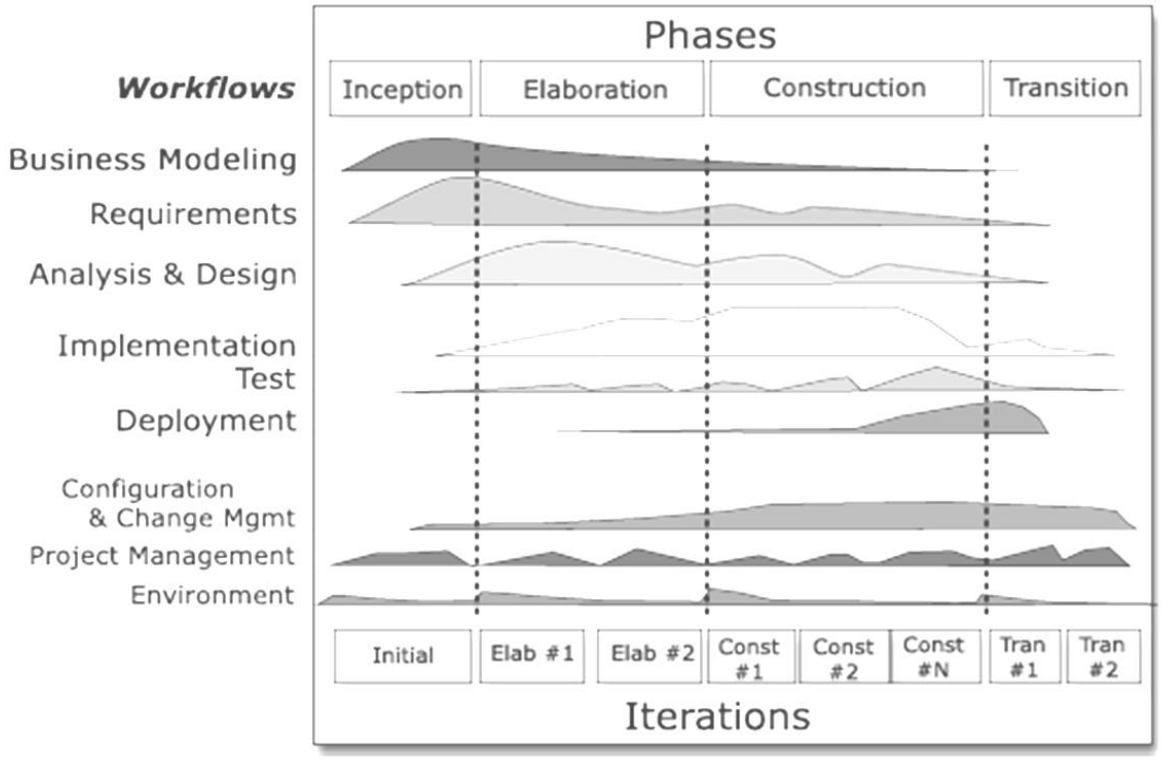


Figura 1 Flujos de trabajo y disciplinas de la metodología de desarrollo RUP

1.4.3 Lenguaje de modelado

El modelado del negocio es la técnica por excelencia para alinear el desarrollo con las metas y objetivos de las empresas e instituciones. Si se realiza de tal forma en que el modelo quede consensuado entre los grupos interesados, las posibilidades de éxito del proyecto aumentarán. El modelado de negocios, y específicamente el modelado de procesos de negocio, es la forma idónea para comunicarnos con los usuarios de todos los niveles. El modelado de procesos constituye la base para el análisis, a partir del cual se identificarán los aspectos que tienen problemas y por tanto deben ser mejorados. (Andrés, 2008)

Utilizar herramientas de modelado visual ayuda a mantener la consistencia entre los artefactos del sistema. En la actualidad existen un gran número de lenguajes de modelado como el BPMN y el UML.

1.4.3.1 UML

UML de sus siglas en inglés Unified Modeling Language o Lenguaje Unificado de Modelado, como su nombre lo indica es un lenguaje que se centra en la representación gráfica de un sistema (Orallo, 2002). Se escoge UML en su versión 2.0 como lenguaje de modelado además

de por las múltiples ventajas que este ofrece, porque es el que define la metodología de desarrollo de software RUP.

Funciones de UML:

- **Visualizar:** UML permite expresar un sistema de una forma gráfica de forma que otra persona lo pueda entender.
- **Especificar:** UML permite especificar cuáles son las características de un sistema antes de su construcción.
- **Construir:** A partir de los modelos especificados se pueden construir los sistemas diseñados.
- **Documentar:** Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

Aunque UML está pensado para modelar sistemas complejos con gran cantidad de software, el lenguaje es lo suficientemente expresivo como para modelar sistemas que no son informáticos, como flujos de trabajo en una empresa, diseño de la estructura de una organización y por supuesto, en el diseño de hardware.

Un modelo UML está compuesto por tres clases de bloques de construcción:

- **Elementos:** Los elementos son abstracciones de cosas reales o ficticias (objetos, acciones, etc.)
- **Relaciones:** relacionan los elementos entre sí.
- **Diagramas:** Son colecciones de elementos con sus relaciones.

UML es además un método formal de modelado. Esto aporta las siguientes ventajas:

- Mayor rigor en la especificación.
- Permite realizar una verificación y validación del modelo realizado.
- Se pueden automatizar determinados procesos y permite generar código a partir de los modelos y a la inversa. Esto permite que el modelo y el código estén actualizados, con lo que siempre se puede mantener la visión en el diseño de más alto nivel de la estructura de un proyecto.

1.4.4 Herramienta CASE

CASE es un acrónimo para Computer-Aided Software Engineering, aunque existen algunas variaciones para lo que actualmente se entiende por CASE como por ejemplo: Computer

Assisted Software Engineering. Esencialmente, un CASE es una herramienta que ayuda al ingeniero de software a desarrollar y mantener software (Valencia, 2002).

En palabras simples se puede decir que las herramientas CASE dan asistencia a todos los involucrados durante los pasos del ciclo de vida de desarrollo de un software. Existen varias herramientas CASE como el Visual Paradigm y el Rational Rose Enterprise.

1.4.4.1 Visual Paradigm

Visual Paradigm está compuesto por un conjunto de herramientas que permiten crear soluciones a la medida para cada programador. Estas herramientas facilitan y permiten a las organizaciones visualizar, diseñar, integrar y distribuir sus aplicaciones empresariales de misión crítica. Además de soporte de modelado, ofrece generación de informes, incluye la generación de código y puede crear ingeniería inversa (del código hacia los diagramas). (Andrés, 2008)

Se escoge Visual Paradigm en su versión 8.0 como herramienta CASE para el modelado de la solución ya que es compatible con el lenguaje de modelado UML que propone RUP y porque es una herramienta multiplataforma que cuenta con una versión Community que es gratis, lo que se ajusta con la política de migración del país y de la universidad hacia software libre.

Visual Paradigm se conforma de tres partes para ser utilizada:
Captura de requisitos: capturar los requerimientos del sistema con el diagrama de casos de uso, diagramas SysML, requerimientos y el análisis textual.

Software de diseño: diseño de la estructura del sistema con el diagrama de clases, diagrama de estructura compuesta. Modelo de interacciones con diagrama de secuencia.

Base de datos y generación de código: diseño de base de datos con el diagrama entidad relación. Generar diagrama UML de clases.

Entre las principales características de Visual Paradigm destacan:

- Creación de modelos UML (Compatible con UML 2.1 y anteriores).
- Modelado de base de datos: proporciona una mayor documentación de la base de datos y diagramas de mapeo de relación de objetos.
- Mapa de relación de objetos.
- Interoperabilidad: intercambio de diagramas UML y modelos con otras herramientas, usando representaciones industriales comunes.
- Modelado de requerimientos.

- Colaboración en equipo: compatible con servidores de equipo como VP Teamwork Server, CVS, etc.
- Generador de código.
- Generador de documentación.

Visual Paradigm ofrece los siguientes beneficios:

- Navegación intuitiva entre código y modelo.
- Generador de documentación y reportes UML/PDF/HTML/MS Word.
- Soporte completo de notaciones UML.
- Diagramas de diseño automático sofisticado.

1.4.5 Lenguaje de programación

Un lenguaje de programación es un conjunto de caracteres, reglas para combinarlos y reglas que especifican sus efectos cuando son ejecutados por una computadora, que tienen las características siguientes: no es necesario que el usuario conozca el código máquina, es independiente de la máquina y es traducido a lenguaje máquina. (Ralston-Reilly-Hemmendinger, 2000)

En otra palabras un lenguaje de programación es el que traduce las instrucciones que el usuario le da a la máquina.

1.4.5.1 C++

C + + es un descendiente directo de C. Apoya directamente una gama más amplia de estilos de programación que C y admite la programación orientada a objetos. El nombre C++ lo obtiene en el año 1983 y hace referencia al carácter del operador incremento de C (++). (Bustamante-Aginaga-Aybar-Olaizola-Lazacano, 2004)

Se escoge C++ como lenguaje de programación ya que este es el lenguaje nativo del IDE de desarrollo QT y porque la librería OpenCV que se utiliza para el tratamiento de imágenes lo utiliza.

Ventajas de C++: (Strouptrup, 2010)

- Lenguaje de programación orientado a objetos.
- Es un lenguaje que permite programar de forma estructurada, modular y orientado a objetos.
- Facilita la reutilización del código.

1.4.6 Entorno de desarrollo

Un Entorno Integrado de Desarrollo o Integrated Development Environment (IDE) es un sistema que facilita el trabajo del desarrollador de software, integrando sólidamente la edición orientada al lenguaje, la compilación o interpretación, la depuración y las medidas de rendimiento de forma modular. (Barahona-Gonzalez, 2008)

En otras palabras un IDE de desarrollo es un marco de trabajo diseñado para simplificarle el trabajo al desarrollador en la creación de aplicaciones.

1.4.6.1 Qt Creator

Qt Creator es un IDE multiplataforma. Este se centra en proporcionar características que ayudan a los nuevos usuarios de Qt a aprender rápido y también aumenta la productividad de los desarrolladores de Qt con experiencia. (Nokia, 2012)

Características

- Resaltado de sintaxis y completamiento de código.
- Herramientas de navegación rápidas.
- Código plegable.

Ventajas de QT Creator

- **Multi-plataforma:** Desarrollar aplicaciones que funcionen tanto en Mac, Windows y Linux es bastante complicado sin Qt. Por eso algunas aplicaciones populares como Google Earth o Skype lo utilizan.
- **Estabilidad y calidad:** Qt existe desde 1992, usado profesionalmente y mejorado muchas veces, resultando en muy buena fiabilidad y facilidad de uso.
- **Amplia documentación:** Luego de 18 años, se acumuló una gran cantidad de documentación acerca de QT que va desde tutoriales hasta fragmentos de código.

Se escoge QT Creator en su versión 4.8 como IDE de desarrollo porque soporta el lenguaje C++ que es el lenguaje escogido, además porque soporta la librería OpenCV.

1.4.7 Biblioteca

Una biblioteca (del inglés library) es un conjunto de subprogramas utilizados para desarrollar software. Las bibliotecas contienen código y datos, que proporcionan servicios a programas independientes, es decir, pasan a formar parte de estos. Esto permite que el código y los datos se compartan y puedan modificarse de forma modular. (Igual-Medrano, 2010)

Una biblioteca no es más que una herramienta que contiene una serie de subprogramas que contienen códigos y datos que se pueden compartir y modificar de forma modular.

1.4.7.1 OpenCV

Las siglas OpenCV provienen de los términos anglosajones “Open Source Computer Vision Library”. Por lo tanto, OpenCV es una biblioteca de tratamiento de imágenes, destinada principalmente a aplicaciones de visión por computador en tiempo real. (Igual-Medrano, 2010)

OpenCV tiene una estructura modular, lo que significa que el paquete incluye varias bibliotecas compartidas o estáticas. Los siguientes módulos están disponibles:

- **Core:** módulo compacto que define las estructuras de datos básicas, incluyendo un arreglo multi-dimensional.
- **Imgproc:** módulo de procesamiento de imagen que incluye la imagen lineal y no lineal de filtrado, transformaciones geométricas de imagen (cambio de tamaño, deformación afín y de perspectiva), conversión de espacio de color, histogramas, entre otros.
- **Video:** módulo de análisis de video que incluye estimación del movimiento, sustracción de fondo, y los algoritmos de seguimiento de objetos.
- **calib3d:** este módulo incluye la calibración simple de una cámara, y elementos de reconstrucción 3D.
- **features2d:** detector de características sobresalientes, descriptores y descriptores comparadores.
- **Objdetect:** detección de objetos e instancias de las clases predefinidas (por ejemplo, caras, ojos, tazas, gente, coches, etc.).
- **Highgui:** una interfaz de video fácil de usar para la captura, códec de imagen y vídeo, así como las capacidades de interfaz de usuario sencilla.
- **Gpu:** Algoritmos GPU acelerados por diferentes módulos de OpenCV.

Se escoge la biblioteca OpenCV en su versión 2.4.3 porque permite el procesamiento de imágenes y además porque contiene las herramientas Haar Training y Train Cascade que son fundamentales para la creación y entrenamiento de las cascadas de clasificadores de haar.

Conclusiones parciales

En este capítulo se analizaron los conceptos relacionados con los clasificadores de haar los cuales favorecieron en el entendimiento del proceso de generación de los mismos contribuyendo a que la herramienta que se desea implementar sea como la requiere el cliente. Luego de realizado un estudio del tema, no se encontró ninguna herramienta que automatice el

proceso de generación de los clasificadores de haar a nivel nacional o internacional. Luego de realizar un estudio de las herramientas y tecnologías usadas en el departamento se determinó escoger las mismas herramientas y tecnologías escogidas por la dirección del proyecto para que contribuya a la compatibilidad de la herramienta que se desea implementar. Todo esto da paso a que se puedan definir las características que tendrá la herramienta para la generación de clasificadores haar.

Capítulo 2. Características de la herramienta para la generación de clasificadores haar

Introducción

En este capítulo se abordarán temas como el modelo del dominio del sistema, la descripción de la propuesta del sistema, el diagrama del modelo de dominio, los requisitos funcionales y no funcionales que tiene el sistema y la descripción del sistema que incluye la definición de los actores del sistema, el listado de los casos de uso, Diagrama de Casos de Uso del sistema y Descripción textual de los casos de uso del sistema.

2.1 Modelo de dominio

El modelo del dominio muestra a los modeladores clases conceptuales significativas en un dominio del problema; es el artefacto más importante que se crea durante el análisis orientado a objetos. El modelo del dominio es una representación de las cosas del mundo real del dominio de interés, no de componentes software, como una clase Java o C+ +, u objetos software con responsabilidades. El modelo del dominio podría considerarse como un diccionario visual de las abstracciones relevantes, vocabulario del dominio e información del dominio. (Larman, 2003)

Utilizando la notación UML, un modelo del dominio se representa con un conjunto de diagramas de clases en los que no se define ninguna operación. Pueden mostrar (Larman, 2003):

- ✓ Objetos del dominio o clases conceptuales.
- ✓ Asociaciones entre las clases conceptuales.
- ✓ Atributos de las clases conceptuales.

Clases conceptuales:

El modelo del dominio muestra las clases conceptuales o vocabulario del dominio. Informalmente, una clase conceptual es una idea, cosa u objeto. Más formalmente, una clase conceptual podría considerarse en términos de su símbolo, intensión, y extensión. (Larman, 2003)

- **Símbolo:** palabras o imágenes que representan una clase conceptual.
- **Intensión:** la definición de una clase conceptual.
- **Extensión:** el conjunto de ejemplos a los que se aplica la clase conceptual.

2.1.1 Descripción del modelo de dominio

El diagrama del modelo de dominio que se muestra a continuación muestra como el Módulo de Análisis del proyecto Video Vigilancia Suria procesa videos digitales que pueden ser provenientes de las cámaras IP de vigilancia asociadas al proyecto o pueden ser videos arbitrarios que se toman para probar las diferentes herramientas que se implementan en el proyecto; el Módulo de Análisis utiliza el reconocimiento de objetos para procesar estos videos digitales y dentro del proceso de reconocimiento de objetos están las cascadas de clasificadores de haar las que se entrenan para identificar los objetos.

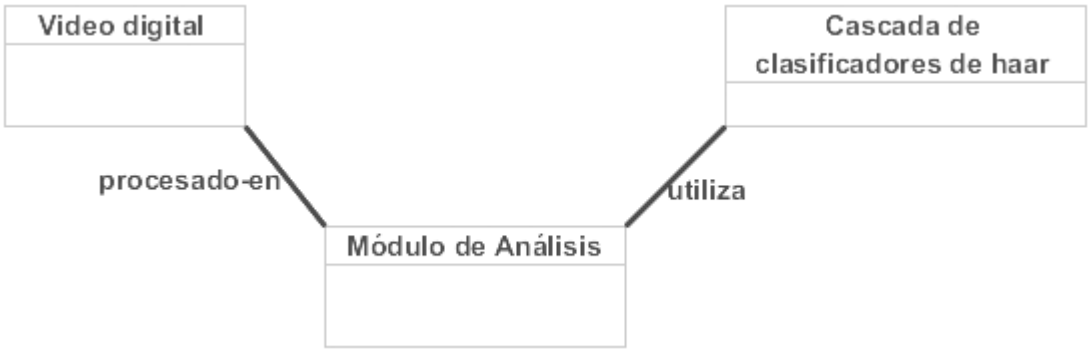


Figura 2 Diagrama de Modelo de Dominio de la herramienta para la generación de clasificadores haar

Se desea desarrollar una herramienta que automatice el proceso de generación de las cascadas de clasificadores de haar para facilitar el trabajo que realizan los desarrolladores, ya que este proceso, ya que este proceso se realiza manualmente en la actualidad. Esta herramienta que se desea implementar debe automatizar las etapas de generación de las cascadas de clasificadores de haar, y devolverá un archivo XML que es lo que manejará el desarrollador a la hora de utilizar la cascada de clasificadores de haar en el proceso de reconocimiento de objetos.

2.2 Requisitos funcionales del sistema

Los requisitos funcionales no son más que declaraciones de como el sistema debe reaccionar a entradas particulares y cómo el sistema debe comportarse en determinadas situaciones. En algunos casos, los requisitos funcionales también pueden declarar explícitamente lo que el sistema no debe hacer. Los requisitos funcionales del sistema describen la función del sistema en detalle, sus entradas y salidas, excepciones, etc. En principio, la especificación de los requisitos funcionales de un sistema debe ser completa y consistente. La completitud significa que todos los servicios requeridos por el usuario deben ser definidos. La consistencia significa que los requisitos no deberían tener definiciones contradictorias. En la práctica, para sistemas

grandes y complejos, es prácticamente imposible de lograr la coherencia y la integridad de los requisitos. (Sommerville, 2007) En resumen los requisitos funcionales son capacidades que debe cumplir el sistema.

- **RF#1 Crear colección de muestras de entrenamiento**

El sistema debe ser capaz de crear una colección de muestras de entrenamiento que luego serán utilizadas para entrenar la cascada de clasificadores de haar, la colección de muestras de entrenamiento se dividirá en muestras de entrenamiento positivas y muestras de entrenamiento negativas.

- **RF#2 Crear archivos de descripción**

El sistema deberá ser capaz de generar archivos de descripción para cada tipo de muestra. El archivo con las muestras positivas contendrá la dirección y el nombre de cada una de las muestras positivas, la cantidad de objetos que están presentes en la imagen y las coordenadas donde está ubicado el objeto. El archivo de descripción de las muestras negativas solo contendrá la dirección y el nombre de las muestras.

- **RF#3 Entrenar la cascada de clasificadores de haar**

El sistema deberá hacer las configuraciones necesarias para que la cascada de clasificadores de haar sea capaz de identificar el objeto en cuestión, es decir, se encargará de los comandos que se entran por consola y que forman parte del entrenamiento de la cascada de clasificadores de haar.

2.3 Requisitos no funcionales del sistema

Los requisitos no funcionales como su nombre indica, son requisitos que no están directamente relacionados con las funciones específicas entregadas por el sistema. Estas son restricciones sobre los servicios o funciones ofrecidas por el sistema. Estos incluyen las limitaciones de tiempo, las limitaciones en el proceso de desarrollo y las normas. Los requisitos no funcionales a menudo se aplican al sistema como un todo. Por lo general no se aplica solamente a funciones del sistema o servicios individuales. (Sommerville, 2007) En otras palabras los requisitos no funcionales son cualidades que debe tener el sistema.

Los tipos de requisitos no funcionales son: (Sommerville, 2007)

- **Requisitos del producto**: Estos requisitos especifican el comportamiento del producto. Los ejemplos incluyen requisitos de desempeño a la rapidez con que el sistema debe ejecutar y la cantidad de memoria que requiere, los requisitos de fiabilidad que fijan la tasa de falla aceptable, requisitos de portabilidad y los requisitos de usabilidad.

- **Requisitos de organización**: Estos requisitos se derivan de las políticas de formularios y procedimientos en los clientes y la organización de los desarrolladores. Los ejemplos incluyen las normas de procesos que se deben utilizar, los requisitos de implementación, tales como el método de programación o lenguaje de diseño utilizado, y los requisitos de entrega que especifican cuándo el producto y su documentación deben ser entregados.

- **Requisitos externos**: Este título amplio cubre todas las necesidades que se derivan de factores para el sistema y su proceso de desarrollo. Estos pueden incluir requisitos de interoperabilidad que definen la forma en que el sistema interactúa con los sistemas de otras organizaciones, los requisitos legales que deben seguirse para asegurar que el sistema funciona dentro de la ley, y los requisitos éticos. Requisitos éticos son requisitos que debe cumplir un sistema para asegurar que sea aceptable para sus usuarios y el público en general.

Requisitos no funcionales:

- **Requisito de Apariencia o Interfaz Externa**

El sistema debe tener una interfaz amigable y sencilla donde el usuario pueda sentirse cómodo. Además de que las interfaces deben ser uniformes y con los mismos colores y diseños. Es un requisito relacionado con el producto

- **Requisito de Usabilidad**

Podrá ser usado por personas que tengan conocimientos básicos en la creación de los clasificadores haar. Es un requisito relacionado con el producto.

2.4 Descripción del sistema

En este epígrafe se dará una descripción detallada del sistema como la definición de lo que es un actor y la definición de los actores del sistema, el listado de casos de uso y los requisitos a

los que están asociados, el diagrama de casos de uso del sistema y la descripción textual de los casos de uso.

2.4.1 Definición de los actores del sistema

Un actor es un rol que cumple un usuario, puede intercambiar información o puede ser un recipiente pasivo de información y representa a un ser humano, a un software o a una máquina que interactúa con el sistema. (Jacobson-Booch-Rumbaugh, 2000)

Actores	**Justificación**
Desarrollador	El desarrollador es el actor principal del sistema ya que los CU que se definen son para satisfacer sus objetivos de usuario. Esto significa que el programador es quien inicia y para el sistema.

Tabla 1 Descripción de los actores del sistema

2.4.2 Listado de Casos de Uso del Sistema

Casos de Uso	**Requisitos que engloba**
Crear colección de muestras de entrenamiento.	Crear colección de muestras de entrenamiento.
	Crear archivos de descripción.
Entrenar cascada de clasificadores haar.	Entrenar cascada de clasificadores haar.

Tabla 2 Listado de CUS y RF asociados

2.4.3 Diagrama de Casos de Uso del Sistema

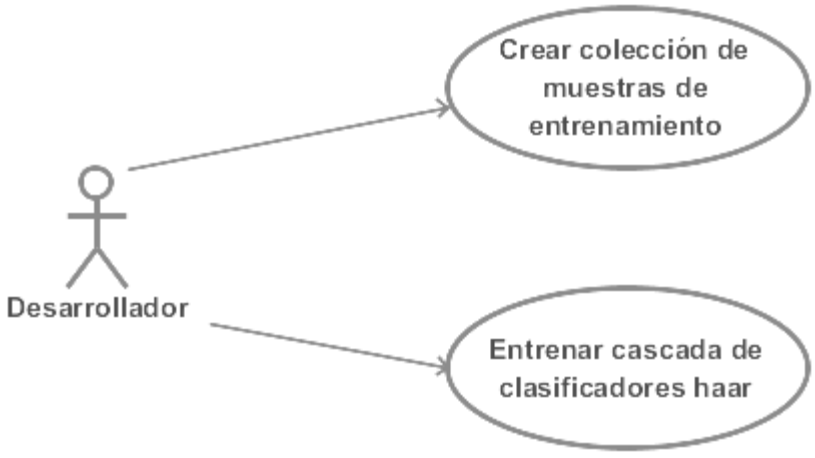


Figura 3 Diagrama de Casos de Uso del Sistema de la herramienta para la generación de clasificadores haar

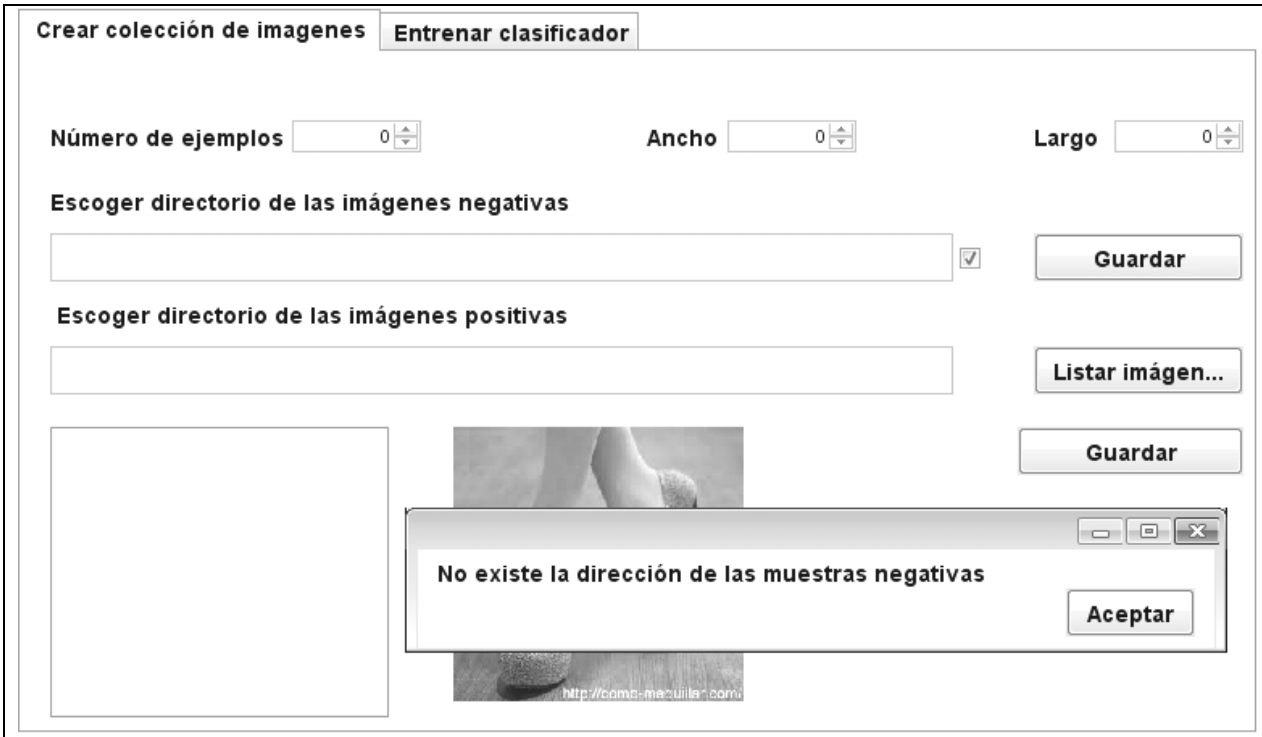
2.4.4 Descripción textual de los principales Casos de Uso del sistema

Objetivo	Crear la colección de muestras de entrenamiento.

Actores	Desarrollador.	
Resumen	Este caso de uso se inicia cuando el sistema es ejecutado y se muestra la interfaz de Crear la colección de muestras de entrenamiento. El caso de uso termina cuando el desarrollador haya creado la colección de imágenes.	
Complejidad	Media	
Prioridad	Alta	
Precondiciones	No procede.	
Poscondiciones	Se obtiene la colección de imágenes creada.	
Flujo de eventos		
Flujo básico: Crear colección de muestras de entrenamiento.		
	Actor	**Sistema**
1.	El desarrollador ejecuta la aplicación.	
2.		El sistema muestra una interfaz con las opciones necesarias para crear la colección de muestras de entrenamiento.
3.	El desarrollador debe entrar el número de ejemplos positivos que quiere que se generen, además del ancho y del alto que tendrán los ejemplos de salida.	
4.	Después el desarrollador presiona el botón examinar y escoge las imágenes que serán utilizadas como imágenes negativas de muestra y presiona el botón guardar.	
5.		El sistema obtiene la información necesaria para crear el archivo de descripción de fondo para las imágenes negativas de muestra.
6.	El desarrollador presiona el botón examinar y escoge las imágenes que	

	serán utilizadas como imágenes positivas de muestra y luego pulsa el botón listar imágenes.	
7.		El sistema muestra una lista con cada una de las imágenes seleccionadas por el desarrollador.
8.	Luego de esto el desarrollador selecciona una imagen y esta se mostrará en la interfaz, luego el desarrollador debe de dar clic derecho encima de la imagen y le saldrá la opción definir coordenadas, el desarrollador pulsará la opción y deberá hacer una diagonal para definir la región donde se encuentra el objeto que se quiere identificar.	
9.		El sistema pinta un rectángulo con la diagonal definida por el usuario en la región donde se encuentra el objeto.
10.	El desarrollador debe pulsar el botón guardar que aparece al lado de la imagen cargada.	
11.		El sistema guarda el nombre de la imagen y las coordenadas donde se encuentra el objeto.
12.	El desarrollador pulsar el botón crear colección.	
13.		El sistema crea la colección de imágenes de muestra.
Prototipo de interfaz		

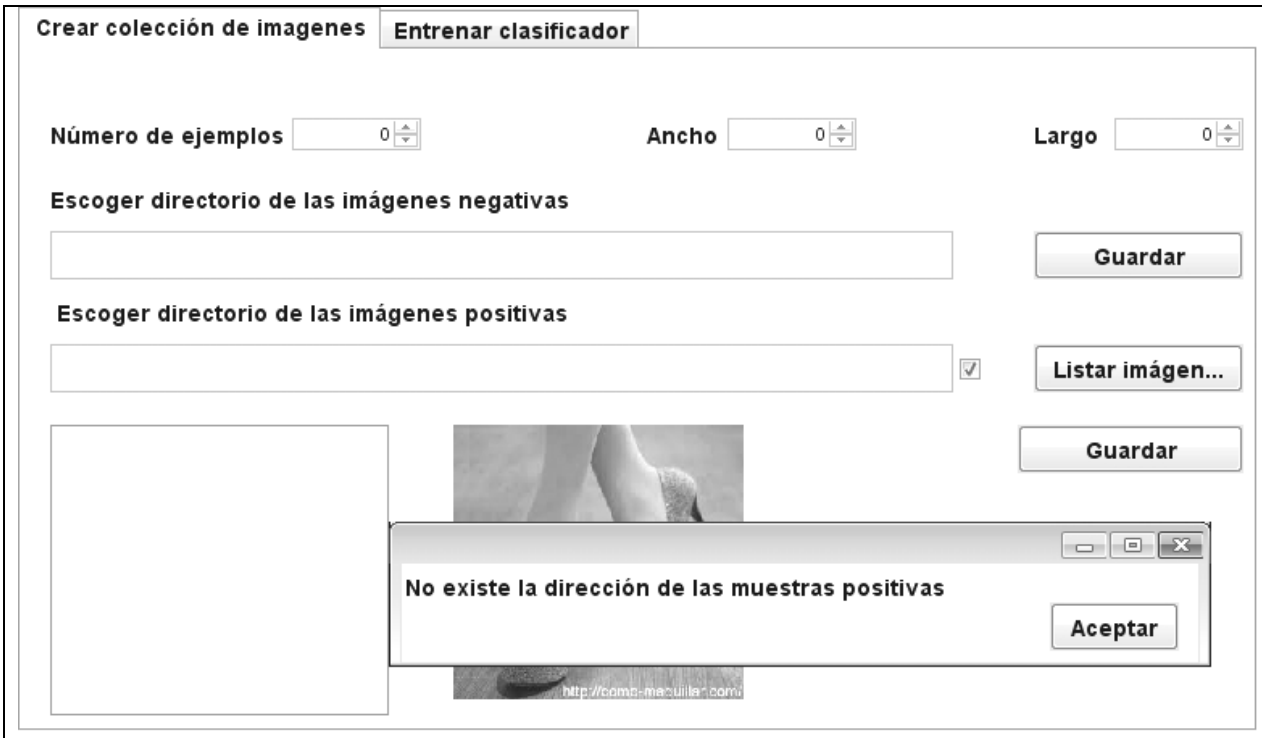
Flujos alternos		
4a. El sistema emite un mensaje indicando que el desarrollador no escogió la dirección de las muestras negativas.		
	Actor	**Sistema**
4a.	El desarrollador después de ejecutar la aplicación, debe escoger la dirección donde se encuentran las muestras negativas.	
		El sistema muestra un mensaje en el que le indica al usuario que no existe la dirección donde se encuentran las muestras negativas.
Prototipo de interfaz		



6a. El sistema emite un mensaje indicando que el desarrollador no escogió la dirección de las muestras positivas.

6a.	El desarrollador después de ejecutar la aplicación, debe escoger la dirección donde se encuentran las muestras positivas.	
		El sistema muestra un mensaje en el que le indica al usuario que no existe la dirección donde se encuentran las muestras positivas.

Prototipo de interfaz



12a. El sistema emite un mensaje indicando que el desarrollador aún no ha llenado todos los campos.

12a.	El desarrollador pulsar el botón crear colección.	
		El sistema muestra un mensaje en el que le indica al usuario que aún tiene campos sin llenar.

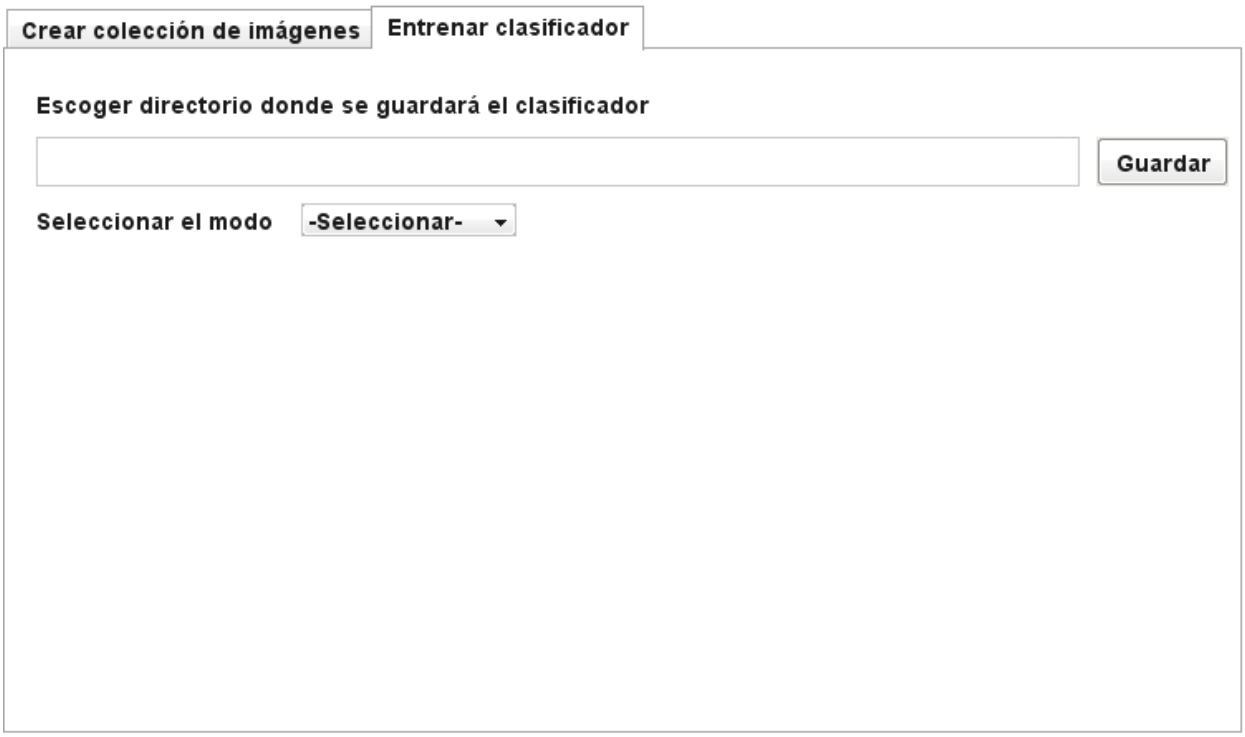
Prototipo de interfaz

Relaciones	**CU Incluidos**	No procede.
	CU Extendidos	No procede.
Requisitos no funcionales	RF 1	
Asuntos pendientes	No procede.	

Tabla 3 Descripción del CUS Crear colección de muestras de entrenamiento

Objetivo	Entrenar cascada de clasificadores haar.
Actores	Desarrollador
Resumen	El caso de uso comienza cuando el desarrollador seleccione la opción de entrenar la cascada de clasificadores haar, luego se muestra una interfaz donde el desarrollador puede configurar las opciones para entrenar la cascada de clasificadores haar. El caso de uso termina cuando la cascada de clasificadores haar haya sido entrenada.
Complejidad	Media
Prioridad	Alta
Precondiciones	Se debe de haber creado la colección de muestras de entrenamiento.

Poscondiciones	Se obtiene el clasificador entrenado.	
Flujo de eventos		
Flujo básico: Entrenar la cascada de clasificadores haar.		
	Actor	**Sistema**
1.	El desarrollador selecciona la opción de entrenar la cascada de clasificadores haar.	
2.		El sistema muestra una interfaz con la cual el desarrollador podrá entrenar el clasificador.
3.	El desarrollador escoge el lugar donde quiere que se guarde la cascada de clasificadores haar, además de seleccionar el tipo de característica del clasificador, el modo y el tipo de boosting y pulsa el botón entrenar.	
4.		El sistema guarda todos los datos que el desarrollador acaba de entrar y genera el clasificador.
Prototipo de interfaz		

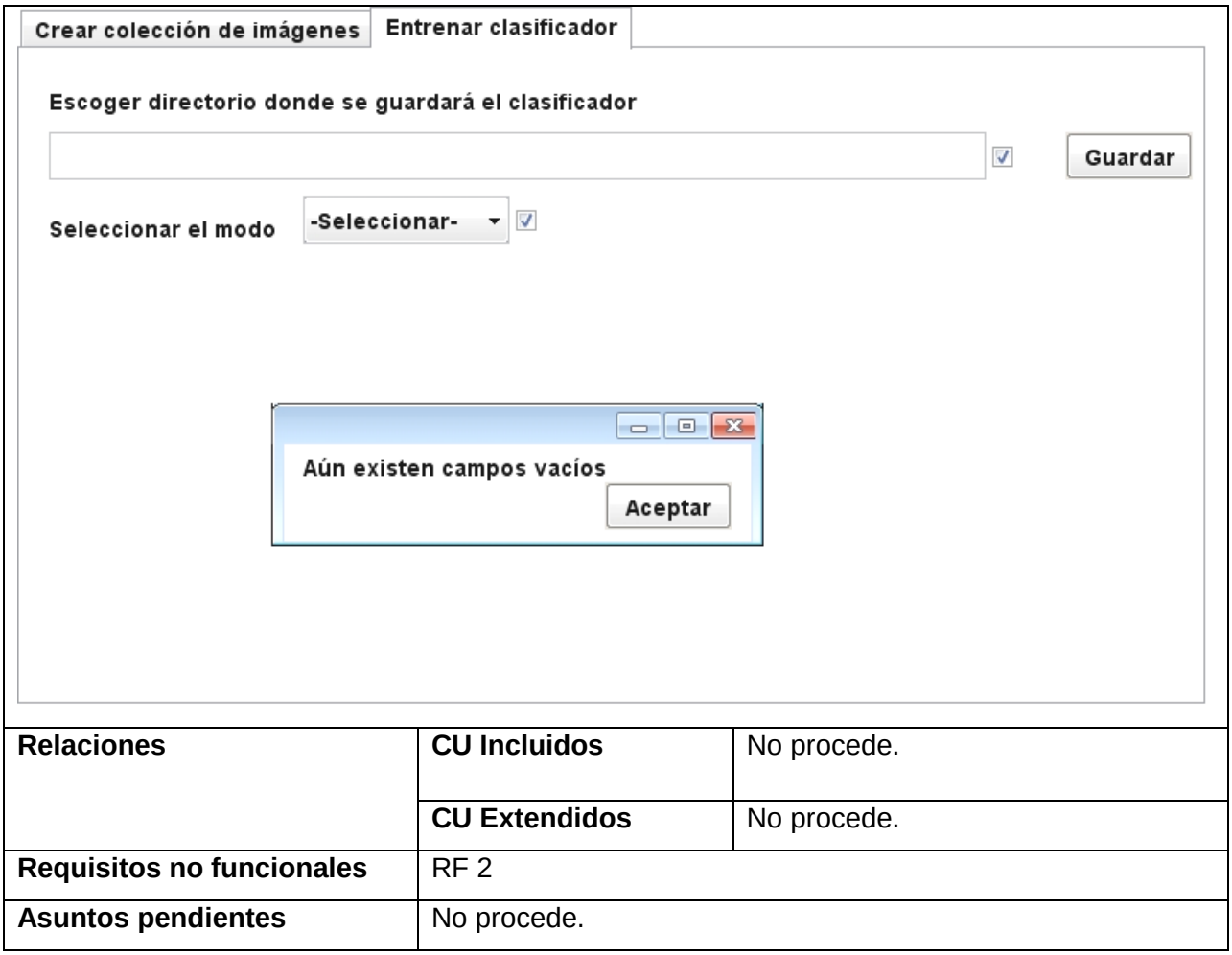


Flujos alternos

3a. El sistema emite un mensaje indicando que el desarrollador dejó campos vacíos y otras opciones por escoger.

	Actor	**Sistema**
3a.	El desarrollador seleccionó la opción de entrenar el clasificador, llenó los campos necesarios y luego pulsó el botón entrenar.	
		El sistema muestra un mensaje donde le informa al usuario que dejó vacía alguna de las opciones necesarias para entrenar el clasificador.

Prototipo de interfaz



Relaciones	**CU Incluidos**	No procede.
	CU Extendidos	No procede.
Requisitos no funcionales	RF 2	
Asuntos pendientes	No procede.	

Tabla 4 Descripción del CUS Entrenar cascada de clasificadores haar

Conclusiones parciales

En este capítulo se hizo un estudio de las características de la herramienta para la generación de clasificadores de haar donde se definió el modelo de dominio, el cual sirvió para representar de forma visual el entorno real del proyecto. Se especificaron los requisitos de software de la herramienta, los cuales ayudaron a la obtención de un mayor entendimiento de lo que hay que tener en cuenta para el desarrollo de la herramienta. Se definieron los casos de uso del sistema y se describieron cada uno de ellos para llegar a una mejor comprensión de lo que se desea que haga la herramienta propuesta. Con la especificación de los requisitos de software y la definición de los casos de uso se logrará definir correctamente los artefactos correspondientes al análisis y diseño de la herramienta.

Capítulo 3. Análisis y diseño de la herramienta para la generación de clasificadores haar.

Introducción

Este capítulo está dirigido a definir los objetivos de los flujos de trabajo de análisis y diseño en el proceso de desarrollo de software. En él se definen los diagramas colaboración y cada uno de los estereotipos que están presentes en el diagrama de clases del análisis, se definen además el diagrama de clases del diseño y el diagrama de secuencia correspondientes al diseño. En este capítulo se define además lo que es la arquitectura y los patrones arquitectónicos y GRASP que se utilizarán.

3.1 Análisis

Llevando a cabo el análisis conseguimos una separación de intereses que prepara y simplifica las subsiguientes actividades de diseño e implementación, delimitando los temas que deben resolverse y las decisiones que deben tomarse en esas actividades. (Jacobson-Booch-Rumbaugh, 2000)

El análisis es la disciplina que sienta las bases para las disciplinas posteriores, simplificando las actividades que se desarrollan en el diseño y la implementación.

3.1.1 Modelo de análisis

El lenguaje que utilizamos en el análisis se basa en un modelo de objetos conceptual: el modelo de análisis. El modelo de análisis ofrece un mayor poder expresivo y una mayor formalización, como por ejemplo, la que proporciona el diagrama de interacción que se utiliza para describir los aspectos dinámicos del sistema. El modelo de análisis se caracteriza también por proporcionar una vista interna del sistema. Es descrito con el lenguaje del desarrollador, proporciona la estructura de las vistas internas y está estructurado por clases y paquetes estereotipados. Es utilizado fundamentalmente por los desarrolladores para comprender como debería darse forma al sistema, es decir, como debería ser diseñado e implementado. No debería contener redundancias, inconsistencias, etc. Esboza como llevar a cabo la funcionalidad dentro del sistema, incluida la funcionalidad significativa para la arquitectura; sirve como una primera aproximación al diseño. Define realizaciones de casos de uso, y cada una de ellas representa el análisis de un caso de uso del modelo de casos de uso. (Jacobson-Booch-Rumbaugh, 2005)

El modelo del análisis se puede definir como un lenguaje descrito por el desarrollador que ofrece una mayor formalización, se caracteriza por proporcionar una vista interna del sistema y que define realizaciones de casos de uso.

3.1.1.1 Diagrama de clases del análisis

Una clase de análisis representa una abstracción de una o varias clases y/o subsistemas del diseño del sistema. Esta abstracción posee las siguientes características: (Jacobson-Booch-Rumbaugh, 2005)

- Una clase del análisis se centra en el tratamiento de los requisitos funcionales y pospone los no funcionales, denominándolos requisitos especiales, hasta llegar a las actividades de diseño e implementación subsiguientes.
- Una clase del análisis raramente define u ofrece una interfaz en términos de operaciones y de sus signaturas. En cambio, su comportamiento se define mediante responsabilidades en un nivel más alto y menos formal. Una responsabilidad es una descripción textual de un conjunto cohesivo del comportamiento de una clase.
- Una clase de análisis define atributos, aunque esos atributos también son de nivel bastante alto. Normalmente los tipos de esos atributos son conceptuales y reconocibles en el dominio del problema, mientras que los tipos de los atributos en las clases de diseño e implementación suelen ser tipos de lenguajes de programación. Además, los atributos identificados durante el análisis con frecuencia pasan a ser clases en el diseño y la implementación.
- Las clases de análisis siempre encajan en uno de tres estereotipos básicos: de interfaz, de control o de entidad. Cada estereotipo implica una semántica específica, lo cual contribuye un método potente y consistente de identificar y describir las clases de análisis y contribuye a la creación de un modelo de objetos y una arquitectura robustos.

Resumiendo una clase del análisis se centra en el tratamiento de los requisitos funcionales y siempre encajan en uno de tres estereotipos básico: de interfaz, de control o de entidad.

Clase interfaz

Las clases de interfaz se utilizan para modelar la interacción entre el sistema y sus actores, es decir, usuarios y sistemas externos. Esta interacción a menudo implica recibir y presentar información y peticiones de y hacia los usuarios y los sistemas externos. Las clases de interfaz modelan las partes del sistema que dependen de sus actores, lo cual implica que clarifican y reúnen los requisitos en los límites del sistema. Por tanto, un cambio en una interfaz de usuario

o en una interfaz de comunicaciones queda normalmente aislado en una o más clases de interfaz. (Jacobson-Booch-Rumbaugh, 2005)

Sintetizando las clases de interfaz se utilizan para formar la relación que existe entre el sistema y el usuario.

Clase entidad

Las clases de entidad se utilizan para modelar información que posee una larga vida y que es a menudo persistente. Las clases de entidad modelan la información y el comportamiento asociado de algún fenómeno o concepto, como una persona, un objeto o un suceso del mundo real. Las clases de entidad suelen mostrar una estructura de datos lógica y contribuyen a comprender de qué información depende el sistema. (Jacobson-Booch-Rumbaugh, 2005)

Reduciendo el concepto de lo que es una clase entidad podemos decir que son las clases que se utilizan para modelar la información que es persistente en el sistema.

Clase control

Las clases de control se usan con frecuencia para encapsular el control de un caso de uso en concreto. Las clases de control también se utilizan para representar derivaciones y cálculos complejos, como la lógica del negocio, que no pueden asociarse con ninguna información concreta, de larga duración, almacenada por el sistema, es decir, una clase de entidad concreta. Los aspectos dinámicos del sistema se modelan con clases de control, debido a que ellas manejan y coordinan las acciones y los flujos de control principales, y delegan trabajo a otros objetos, es decir, objetos de interfaz y de entidad. (Jacobson-Booch-Rumbaugh, 2005)

Las clases de control no son más que las clases que se usan para modelar los aspectos dinámicos del sistema ya que estas manejan las acciones de coordinación, secuencia, transacciones y control de otros objetos.

A continuación se muestra el diagrama de clases del análisis del CUS Crear colección de muestras de entrenamiento.

Figura 4 Diagrama de Clases del Diseño del CUS Crear colección de muestras de entrenamiento

Descripción del diagrama de clases del análisis del CUS Crear colección de muestras de entrenamiento.

En el diagrama está presente la clase CI_Generator que es la clase interfaz del sistema, clase con la que interactúa el usuario que en este caso es el desarrollador, esta clase interfaz se conecta con la clase controladora CC_TrainCascade que es la que contiene los datos necesarios para crear la colección de muestras y por último la clase entidad CE_Label que es la que se encarga de mostrarle las imágenes al usuario.

A continuación se muestra el diagrama de clases del análisis del CUS Entrenar cascada de clasificadores haar.

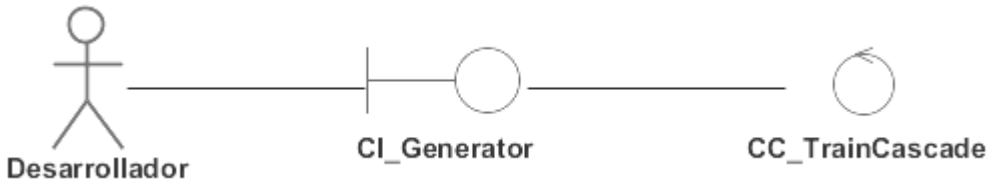


Figura 5 Diagrama de Clases del Análisis del CUS Entrenar cascada de clasificadores haar

Descripción del diagrama de clases del análisis del CUS Entrenar cascada de clasificadores haar.

En el diagrama está presente la clase CI_Generator que es la clase interfaz del sistema, clase con la que interactúa el usuario, la clase interfaz se conecta con la clase controladora CC_TrainCascade que es la que contiene los datos necesarios para entrenar al clasificador.

3.1.1.2 Diagrama de colaboración

En los diagramas de colaboración se muestran las interacciones entre objetos creando enlaces entre ellos y añadiendo mensajes a esos enlaces. El nombre de un mensaje debería denotar el propósito del objeto invocante en la interacción con el objeto invocado. (Jacobson-Booch-Rumbaugh, 2005)

Los diagramas de colaboración son los encargados de describir la interacción que existe entre los objetos que intervienen en el sistema.

A continuación se muestran el diagrama de colaboración del CUS Crear colección de muestras de entrenamiento.

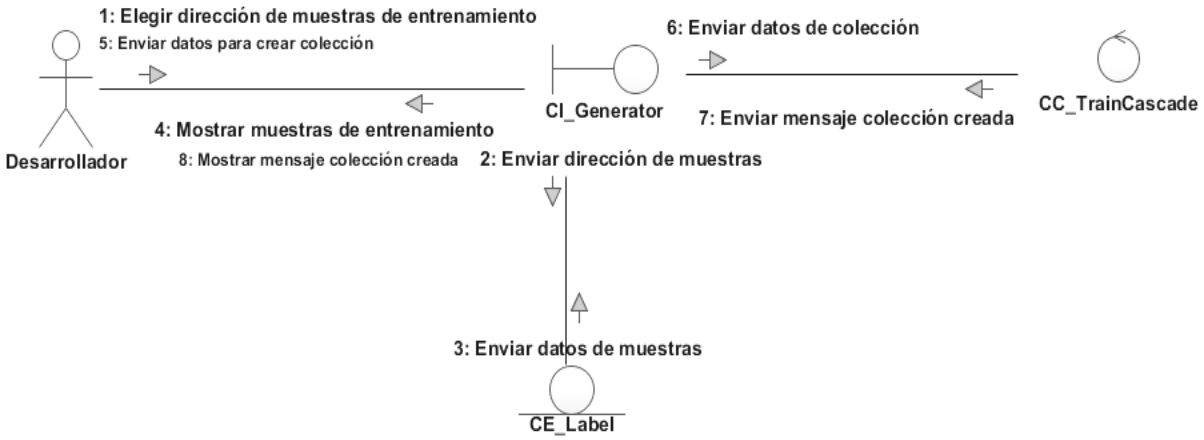


Figura 6 Diagrama de colaboración del CUS Crear colección de muestras de entrenamiento

Descripción del diagrama de colaboración del CUS Crear colección de muestras de entrenamiento.

Se muestra una interfaz donde el usuario debe escoger la dirección donde se encuentran las muestras de entrenamiento. La clase interfaz envía las direcciones de las muestras a la clase entidad marco que es la encargada de mostrar las imágenes en la clase interfaz. Luego de esto el usuario llena los demás datos que se necesitan para crear la colección. La clase interfaz le envía los datos a la clase controladora y esta se encarga de crear una nueva colección de muestras de entrenamiento. Después de esto la clase interfaz el envía un mensaje al desarrollador indicando que ya se creó la colección de muestras de entrenamiento.

A continuación se muestra el diagrama de colaboración del CUS Entrenar cascada de clasificadores haar.

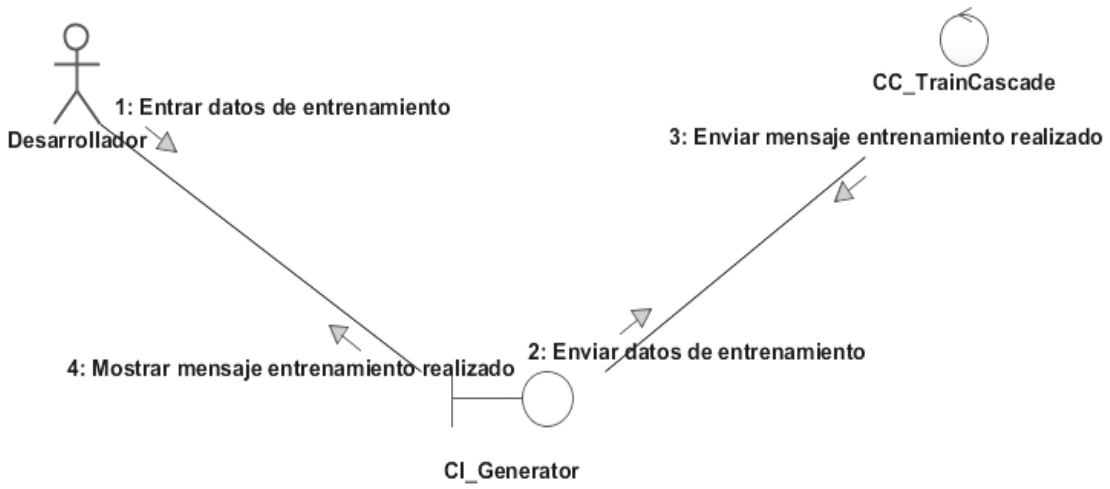


Figura 7 Diagrama de clases del análisis de CUS Entrenar cascada de clasificadores haar

3.2 Diseño

En el diseño se modela el sistema y se encuentra su forma incluida la arquitectura para que soporte todos los requisitos, incluyendo los requisitos no funcionales y otras restricciones que se le suponen. Los propósitos del diseño son: (Jacobson-Booch-Rumbaugh, 2005)

- Adquirir una comprensión en profundidad de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación y concurrencia, tecnologías de interfaz de usuario, tecnologías de gestión de transacciones, etc.
- Crear una entrada apropiada y un punto de partida para actividades de implementación subsiguientes capturando los requisitos o subsistemas individuales, interfaces y clases.
- Ser capaces de descomponer los trabajos de implementación en partes más manejables que pueden ser llevadas a cabo por diferentes equipos de desarrollo, teniendo en cuenta la posible concurrencia. Esto resulta útil en los casos en los que la descomposición no puede ser hecha basándose en los resultados de la captura de requisitos o análisis.

En la disciplina del diseño es donde se define la forma que va a tener el sistema, esto se logra mediante la definición de la arquitectura del sistema, además crea una entrada y un punto de partida para las actividades de la disciplina de implementación.

3.2.1 Arquitectura

La arquitectura no es más que la estructura organizativa de un sistema que incluye su descomposición en partes, conectividad, mecanismos de interacción y principios de guía que proporcionan información sobre el diseño del mismo. Es el conjunto de decisiones significativas sobre la organización de un sistema software. Incluye la selección de elementos estructurales y las interfaces mediante las que se conectan, la organización a gran escala de los elementos estructurales y la topología de su conexión, su comportamiento en las colaboraciones entre dichos elementos, los mecanismos importantes de que se dispone en el sistema y el estilo arquitectónico que guía su organización. (Jacobson-Booch-Rumbaugh, 2005)

La arquitectura de un sistema no es más que la distribución de sus partes más relevantes, esto permite tener una visión común entre todos los involucrados y una perspectiva clara del sistema completo.

Se necesita una arquitectura para:

- Comprender el sistema.
- Organizar el desarrollo.
- Fomentar la reutilización.
- Hacer evolucionar el sistema.

3.2.2 Patrón arquitectónico n-capas

Un patrón es una descripción de un problema y su solución, que recibe un nombre y que puede emplearse en otros contextos; en teoría, indica la manera de utilizarlo en circunstancias diversas. (Larman, 2003)

El patrón n-capas es aplicable a muchos tipos de sistemas y define cómo organizar el modelo de diseño en capas. (Jacobson-Booch-Rumbaugh, 2005)

Principios comunes para los diseños que utilizan la arquitectura en capas

- **Abstracción.** La arquitectura en capas abstrae la vista del sistema en su conjunto, al tiempo que proporciona suficiente detalle para entender los roles y responsabilidades de las capas individuales y la relación entre ellos.
- **Encapsulación.** No se necesita hacer suposiciones sobre los tipos de datos, los métodos y las propiedades de aplicación, o durante el diseño, ya que estas funciones no están expuestas en los límites de la capa.
- **Capas funcionales claramente definidas.** La separación entre la funcionalidad en cada capa es clara. Las capas superiores envían comandos a las capas inferiores y

pueden reaccionar a eventos en estas capas, lo que permite que los datos fluyan hacia arriba y hacia abajo entre las capas.

- **Alta cohesión.** Fronteras bien definidas de responsabilidad para cada capa, y asegurando que cada capa contiene la funcionalidad directamente relacionada con las funciones de esa capa, ayudarán a maximizar la cohesión dentro de la capa.
- **Reutilizable.** Las capas inferiores no tienen dependencias en las capas más altas, potencialmente permitiendo que sean reutilizables en otros escenarios.
- **Bajo acoplamiento.** La comunicación entre las capas se basa en la abstracción y eventos para proporcionar bajo acoplamiento entre las capas.

Específicamente dentro del patrón n capas se utilizará el patrón dos capas, el mismo estará compuesto por las capas de presentación y negocio. En la capa de presentación se encontrarán los objetos relacionados con las interfaces del sistema y en la capa del negocio se encontrarán los objetos relacionados con los datos del sistema y la clase controladora. Se le aplica este patrón a la solución porque simplifica la comprensión y la organización del sistema, reduce las dependencias entre sus componentes y permite la reutilización de las capas que se definan.

3.2.3 Patrones GRASP

GRASP es un acrónimo que significa General Responsibility Asignment Software Patterns (patrones generales de software para asignar responsabilidades). El nombre se eligió para indicar la importancia de captar (grasping) estos principios, si se quiere diseñar eficazmente el software orientado a objetos. Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Los patrones GRASP que serán utilizados son: (Larman, 2003)

- Bajo acoplamiento.
- Alta cohesion.
- Experto.
- Creador.

3.2.3.1 Bajo acoplamiento

Solución: Asignar una responsabilidad para mantener bajo acoplamiento.

Problema: ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización?

El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases. Una clase con bajo (o débil) acoplamiento no depende de muchas otras. Una clase con alto (o

fuerte) acoplamiento recurre a muchas otras. Este tipo de clases no es conveniente ya que presentan los siguientes problemas:

- Los cambios de las clases afines ocasionan cambios locales.
- Son más difíciles de entender cuando están aisladas.
- Son más difíciles de reutilizar porque se requiere la presencia de las otras clases de las que dependen.

El Bajo Acoplamiento es un principio que debemos recordar durante las decisiones de diseño, es la meta principal que es preciso tener presente siempre. Es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño. (Larman, 2003)

Este patrón se le aplica a todas las clases del sistema ya que aporta los siguientes beneficios: (Larman, 2003)

- Las clases no se afectan por cambios de otras clases.
- Las clases son fáciles de entender por separado.
- Las clases son más fáciles de reutilizar.

3.2.3.2 Alta cohesión

Solución: Asignar una responsabilidad de modo que la cohesión siga siendo alta.

Problema: ¿Cómo mantener la complejidad dentro de límites manejables?

En la perspectiva del diseño orientado a objetos, la cohesión (o, más exactamente, la cohesión funcional) es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Una clase con baja cohesión hace muchas cosas no afines o un trabajo excesivo. No conviene este tipo de clases pues presentan los siguientes problemas:(Larman, 2003)

- Son difíciles de comprender.
- Son difíciles de reutilizar.
- Son difíciles de conservar.
- Son delicadas: las afectan constantemente los cambios.

Las clases con baja cohesión a menudo representan un alto grado de abstracción o han asumido responsabilidades que deberían haber delegado a otros objetos. Alta Cohesión es un principio que debemos tener presente en todas las decisiones de diseño, es la meta principal que ha de buscarse en todo momento. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño. Este patrón -como tantas otras cosas en la tecnología de objeto- presenta semejanzas con el mundo real. Todos sabemos que, si alguien asume

demasiadas responsabilidades -sobre todo las que debería delegar-, no sería eficiente. (Larman, 2003)

Este patrón es aplicado a todas las clases del sistema ya que ofrece los siguientes beneficios: (Larman, 2003)

- Mejoran la claridad y la facilidad con que se entiende el diseño.
- Se simplifican el mantenimiento y las mejoras en funcionalidad.
- A menudo se genera un bajo acoplamiento.
- La ventaja de una gran funcionalidad soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico.

3.2.3.3 Experto

Solución: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.

Problema: ¿Cuál es el principio fundamental en virtud del cual se asignan las responsabilidades en el diseño orientado a objetos?

Durante el diseño orientado a objetos, cuando se definen las interacciones entre los objetos, tomamos decisiones sobre la asignación de responsabilidades a las clases. Si se hacen en forma adecuada, los sistemas tienden a ser más difíciles de entender, mantener y ampliar, y se nos presenta la oportunidad de reutilizar los componentes en futuras aplicaciones. Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos y expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen. (Larman, 2003)

Beneficios: (Larman, 2003)

- Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento.
- El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases sencillas y más cohesivas que son más fáciles de comprender y de mantener. Así se brinda soporte a una alta cohesión.

Este patrón se aplica a todas las clases del sistema ya que como se explicaba anteriormente expresa que los objetos solo deben de hacer cosas relacionadas con la información que poseen.

3.2.3.4 Creador

Asignar a la clase B la responsabilidad de crear una instancia de clase A si se cumple uno o más de los casos siguientes:

- B agrega objetos de A.
- B contiene objetos de A.
- B registra instancias de objetos de A.
- B utiliza más estrechamente objetos de A.
- B tiene los datos de inicialización que se pasarán a un objeto de A cuando sea creado. B es un creador de los objetos A.

Problema ¿Quién debería ser el responsable de la creación de una nueva instancia de alguna clase?

La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos. En consecuencia, es útil contar con un principio general para la asignación de las responsabilidades de creación. Si se asignan bien, el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización.

Beneficios

Se soporta bajo acoplamiento, lo que implica menos dependencias de mantenimiento y mayores oportunidades para reutilizar. Probablemente no se incrementa el acoplamiento porque la clase creada es presumible que ya sea visible a la clase creadora, debido a las asociaciones existentes que motivaron su elección como creador.

Este patrón se aplica a la clase Generator que es la clase que tiene la necesidad de crear una instancia de la clase controladora TrainCascade para poder utilizar los métodos que en ella se definen y de la clase entidad Label para poder mostrar las imágenes que el usuario necesita.

3.3.4 Patrones GoF

Sus siglas significan Gang of Four y se les denomina así por sus creadores que fueron 4: Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides.

Estos son patrones de diseño se dividen en tres categorías: (Gamma-Helm-Johnson-Vlissides, 2003)

- **Patrones de creación:** estos patrones abstraen el proceso de creación de objetos y ayudan a crear sistemas independientes de cómo los objetos son creados, compuestos y representados.
- **Patrones estructurales:** definen cómo clases y objetos se combinan para formar estructuras más complejas. Son patrones basados en la herencia.

- **Patrones de comportamiento:** estos patrones están relacionados con la asignación de responsabilidades entre clases. Enfatizan la colaboración entre objetos.

Observador

Este patrón entra en la clasificación de patrón de comportamiento. Su propósito es definir una dependencia uno-a-muchos entre objetos, de modo que cuando cambia el estado de un objeto, todos sus dependientes automáticamente son notificados y actualizados. (Gamma-Helm-Johnson-Vlissides, 2003)

3.3.5 Modelo de diseño

Un modelo de diseño es un modelo de objetos físicos que describen la realización física de los casos de usos centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema. El modelo de diseño perdurará durante todo el ciclo de vida del software y constituye la entrada fundamental utilizada para el correcto desarrollo de la implementación. (Figueras, 2011)

En otras palabras un modelo de diseño es aquel artefacto del diseño que describe la realización física de los casos de uso y de cómo estos relacionados con otros artefactos y restricciones tienen impacto en el sistema.

3.3.5.1 Diagrama de clases del diseño

El diagrama de clases del diseño muestra un conjunto de clases, interfaces y sus relaciones. Éste es el diagrama más común a la hora de describir el diseño de los sistemas orientados a objetos. (Orallo, 2002)

Las clases del diseño sirven para obtener una representación del modelo del diseño estático de un sistema. Incluye los elementos de UML típicos de los diagramas de clase que se utilizan durante el análisis de requisitos y añade otros elementos específicos del diseño como interfaces, visibilidad y navegación. (informáticos, 2005)

Resumiendo el diagrama de clases del diseño muestra un conjunto de clases del diseño y sus relaciones, estas clases sirven para obtener una representación estática del sistema.

Descripción de las clases del diagrama de clases del diseño de la herramienta para la generación de clasificadores haar

uiGenerator: es la clase interfaz del sistema y contiene todos los elementos visuales que intervienen en la solución del problema.

generator: es la clase controladora de la interfaz del sistema en donde se encuentran las acciones visuales que el usuario ejecuta.

label: esta clase hereda de QLabel y contiene los eventos del mouse que se utilizan en la aplicación además de ser la clase encargada de mostrar las imágenes que el usuario debe cargar.

trainCascade: es la clase controladora del sistema, en ella están presentes los principales métodos de la aplicación.

A continuación se muestra el diagrama de clases del diseño de la herramienta para la generación de clasificadores haar.

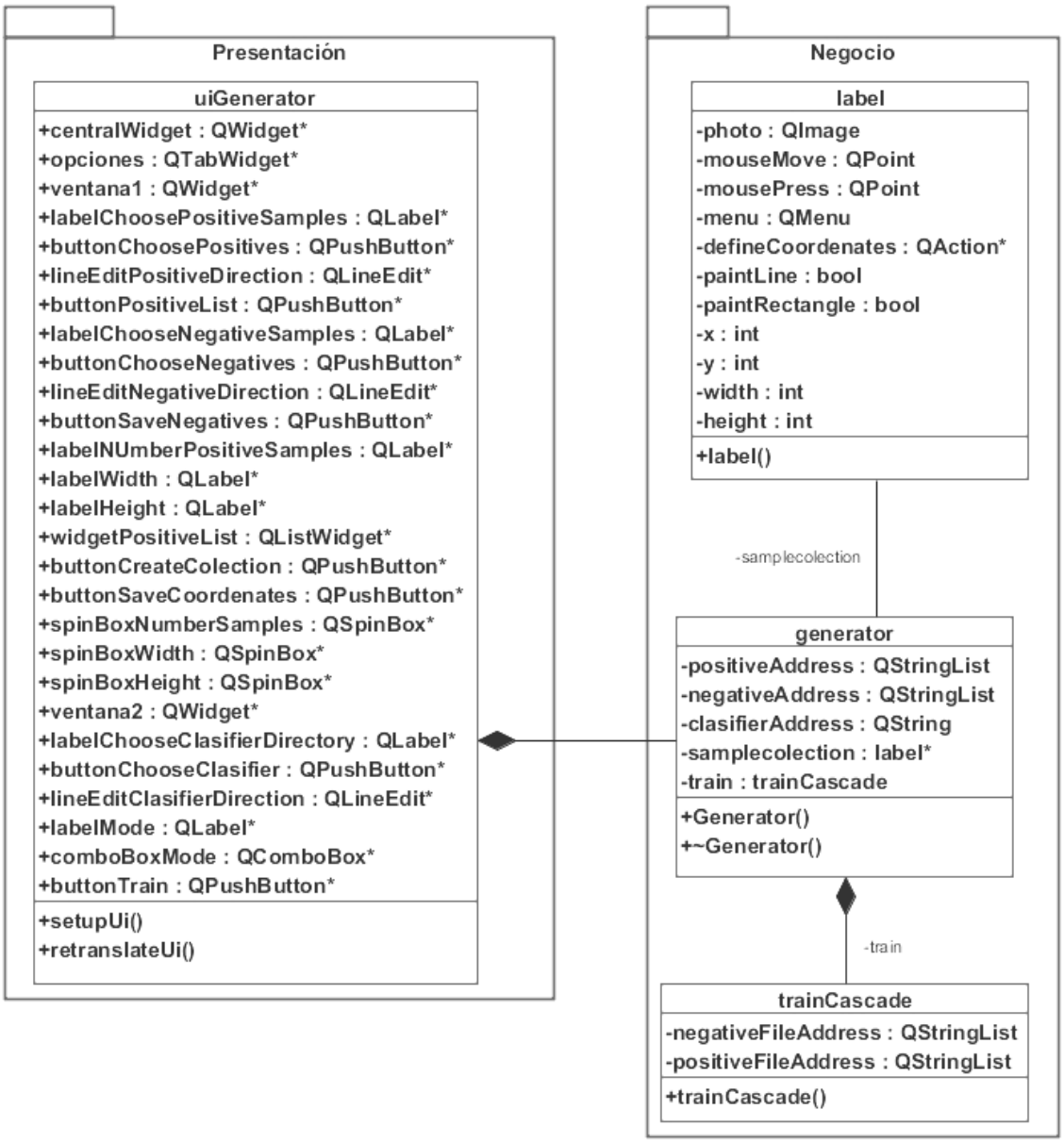


Figura 8 Diagrama de clases del diseño de la herramienta para la generación de clasificadores haar

3.3.5.2 Diagrama de secuencia

Muestran la interacción entre elementos, mostrando de forma explícita la secuencia de estímulos ordenada temporalmente. Se utilizan para describir los distintos escenarios derivados de los casos de uso. Un escenario es una secuencia específica de acciones que ilustra un comportamiento. Básicamente es una instancia de un caso de uso. Un caso de uso puede tener muchos escenarios. (Mora, 2002)

Los diagramas de secuencia son los encargados de mostrar la interacción que existe entre las acciones que realiza el usuario y las clases que interviene en el sistema.

A continuación se muestra el diagrama de secuencia del CUS Crear colección de muestras de entrenamiento.

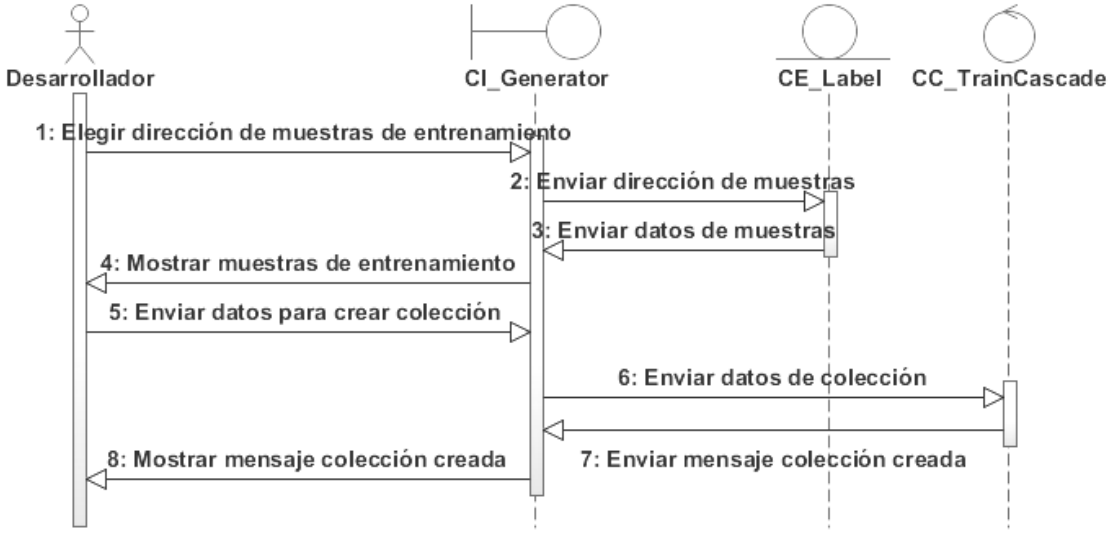


Figura 9 Diagrama de secuencia del CUS Crear colección de muestras de entrenamiento

A continuación se muestra el diagrama de secuencia del CUS Entrenar cascada de clasificadores haar.

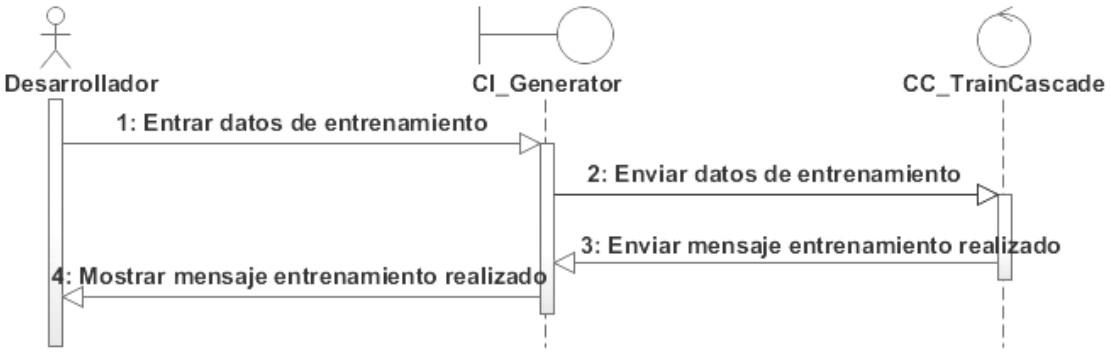


Figura 10 Diagrama de secuencia del CUS Entrenar cascada de clasificadores haar

Conclusiones parciales

En este capítulo se definieron los flujos de análisis y diseño de la herramienta para la generación de clasificadores haar, y luego de ello se llegaron a las siguientes conclusiones parciales:

Se definió el modelo de análisis y diseño de la herramienta que permitió profundizar en los casos de usos, ya que son la base fundamental para la implementación de la herramienta. Se expuso la línea base de la arquitectura que se utilizará en el desarrollo de la herramienta, la cual posibilita una mayor comprensión de la misma. Se definieron los patrones de diseño alta cohesión, bajo acoplamiento, experto y creador, los cuales junto a al patrón arquitectónico dos

capas guiarán todo el proceso de desarrollo de software y servirá como guía durante la disciplina de implementación.

Capítulo 4. Implementación y prueba de la herramienta para la generación de clasificadores haar

Introducción

En este capítulo se describirán los flujos de implementación y prueba de la herramienta para la generación de clasificadores haar, así como los artefactos que se generan en cada uno de ellos siendo estos el diagrama de despliegue, el diagrama de componentes, el estándar de codificación que se utilizó para estandarizar el código y las pruebas realizadas a la herramienta.

4.1 Implementación

La disciplina de implementación se empieza con el resultado del diseño y se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables y similares. Los propósitos de la implementación son: (Jacobson-Booch-Rumbaugh, 2005)

- Planificar las integraciones de sistema necesarias en cada iteración. Se sigue para ello un enfoque incremental, lo que da lugar a un sistema que se implementa en una sucesión de pasos pequeños y manejables.
- Distribuir el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue. Esto se basa fundamentalmente en las clases activas encontradas durante el diseño.
- Implementar las clases y subsistemas encontrados durante el diseño. En particular, las clases se implementan como componentes de fichero que contienen código fuente.
- Probar los componentes individualmente, y a continuación integrarlos compilándolos y enlazándolos en uno o más ejecutables, antes de ser enviados para ser integrados y llevar a cabo las comprobaciones de sistema.

4.1.1 Diagrama de despliegue

Un diagrama de despliegue muestra cómo se configuran las instancias de los componentes y los procesos para la ejecución run-time en las instancias de los nodos de proceso. (Larman, 2003)

A continuación se muestra el diagrama de despliegue de la Herramienta para la generación de clasificadores haar

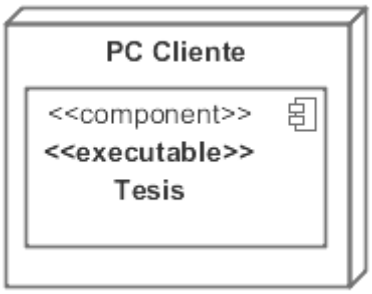


Figura 11 Diagrama de despliegue de la herramienta para la generación de clasificadores haar

4.1.2 Diagrama de componentes

Un **componente** representa una parte de un sistema modular, desplegable, y reemplazable, que encapsula la implementación y expone un conjunto de interfaces. Podría ser, por ejemplo, código fuente, binario o ejecutable. Entre los ejemplos encontramos navegadores o servidores HTTP, una base de datos, una DLL, o un fichero JAR (como para un Enterprise Java Bean). Los componentes en UML se representan en los diagramas de despliegue, en lugar de independientemente. (Larman, 2003)

A continuación se muestra el diagrama de componentes de la herramienta para la generación de clasificadores haar

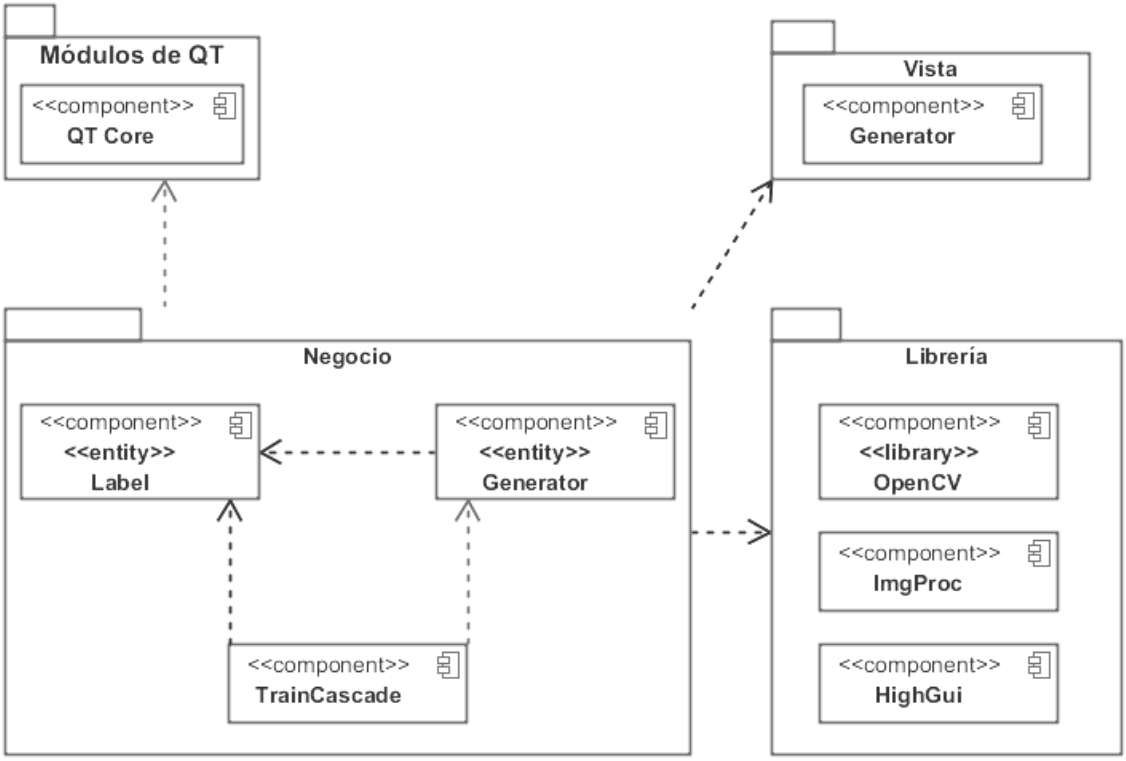


Figura 12 Diagrama de componentes de la herramienta para la generación de clasificadores haar

Descripción del diagrama de componentes de la herramienta para la generación de clasificadores haar

El diagrama de componentes está formado por el módulo de Qt que se utilizó en la confección de la herramienta, este módulo es el QT Core. Está compuesto además por el componente de la capa de la vista generator y por los componentes de la capa del negocio label, generator, trainCascade y además por la librería OpenCV y los módulos de la misma que se usaron en el tratamiento de las imágenes el ImgProc y el HighGui.

4.1.3 Estándares de codificación

Se definen estándares de codificación porque un estilo de programación homogéneo en un proyecto permite que todos los participantes lo puedan entender en menos tiempo y que el código en consecuencia sea mantenible. (Calleja, 2008)

El uso de estos estándares tiene innumerables ventajas, entre ellas: (Electrónico, 2006)

- Asegurar la legibilidad del código entre distintos programadores, facilitando el debugging del mismo.

- Proveer una guía para el encargado de mantenimiento/actualización del sistema con código claro y bien documentado.
- Facilitar la portabilidad entre plataformas y aplicaciones.

A continuación se describe el estándar de codificación que se utilizó en la realización de la herramienta, que es el mismo que utilizan todos las demás aplicaciones que se desarrollan en el proyecto Video Vigilancia.

Indentación

Se utilizarán cuatro espacios para la indentación, no se utilizará tabulación.

Declaración de variables y métodos

Las variables se declararán en líneas separadas. Se evitará utilizar abreviaturas en la declaración de una variable a no ser que sea para declarar una variable temporal o un contador. Las variables y los métodos empezarán con letra minúscula y si la variable o el método están compuestos por más de una palabra se escribirán las palabras una a continuación de la otra, sin guiones o guiones bajos y a partir de la segunda palabra se empieza a escribir con mayúscula haciendo uso del estándar camel case.

Espacios en blanco

Se usarán líneas en blanco para agrupar declaraciones que estén relacionadas. Se usará solamente una línea en blanco. Se usará siempre un espacio después de una palabra clave del lenguaje y antes de la llave de inicio un bloque de sentencias. Para punteros o referencias, usar siempre un espacio entre el tipo y los símbolos (*, &) y no entre los símbolos y las variables. No poner espacios en blanco después de un casteo.

Llaves

Como regla básica la llave de inicio se pondrá en la misma línea que el inicio de la sentencia. No poner llaves cuando las sentencias están constituidas por una sola línea.

Excepciones: la implementación de las funciones y la declaración de las clases siempre tendrán la llave de inicio se pondrán debajo del inicio de la sentencia. Se pondrán llaves a pesar de que sea una sola sentencia cuando los las condicionales que están dentro de los paréntesis se llevan más de una línea.

Paréntesis

Se usarán los paréntesis para agrupar expresiones.

Saltos de línea

Se insertarán saltos de línea si la línea de la sentencia tiene más de 100 caracteres. Las comas van al final de una línea quebrada y los operadores van al inicio de la nueva línea.

4.2 Prueba

En el flujo de trabajo de prueba se verifica el resultado de la implementación probando cada construcción, incluyendo tantas construcciones internas intermedias, así como las versiones finales del sistema a ser entregadas a terceros. Los objetivos de este flujo de trabajo son: (Jacobson-Booch-Rumbaugh, 2005)

- Planificar las pruebas necesarias en cada iteración, incluyendo las pruebas de integración y las pruebas de sistema. Las pruebas de integración son necesarias para cada construcción dentro de la iteración, mientras que las pruebas de sistema son necesarias sólo al final de la iteración.
- Diseñar e implementar las pruebas creando los casos de prueba que especifican qué probar, creando los procedimientos de prueba que especifican como realizar las pruebas y creando, si es posible, componentes de prueba ejecutables para automatizar las pruebas.
- Realizar las diferentes pruebas y manejar los resultados de cada prueba sistemáticamente. Las construcciones en las que se detectan defectos son probadas de nuevo y posiblemente devueltas a otro flujo de trabajo, como diseño o implementación, de forma que los defectos importantes puedan ser arreglados.

Existen diferentes tipos de técnicas de diseño de casos de pruebas que se pueden aplicar a un software: (Pressman, 2005)

La prueba de caja blanca: denominada a veces prueba de caja de cristal es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca, el ingeniero del software puede obtener casos de prueba que:

- garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo;
- ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa;
- ejecuten todos los bucles en sus límites y con sus límites operacionales;

- ejerciten las estructuras internas de datos para asegurar su validez.

La prueba de caja negra: también denominada *prueba de comportamiento,* se centran en los requisitos funcionales del software. O sea, la prueba de caja negra permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. La prueba de caja negra no es una alternativa a las técnicas de prueba de caja blanca.

Para la realización de las pruebas a la Herramienta para la generación de clasificadores haar se utilizará la prueba de caja negra, ya que esta permite encontrar errores funcionales que puede existir en el software y que no son detectados por los desarrolladores.

Caso de Prueba #1 CUS Crear colección de muestras de entrenamiento.

Sesión	**Escenario**	**Acción Realizada**	**Reacción del Sistema**	**Resultado**
SC 1. Crear colección de muestras de entrenamiento.	EC 1.1. Muestras negativas creadas satisfactoriamente.	El desarrollador introduce la dirección de las muestras negativas de entrenamiento y presiona el botón Guardar.	El sistema verifica que el campo no esté vacío y guarda las muestras negativas en un archivo.	Satisfactorio
	SC 1.2. Muestras positivas creadas satisfactoriamente.	El desarrollador introduce la dirección de las muestras positivas y presiona el botón Guardar.	El sistema verifica que el campo no esté vacío y guarda las muestras negativas en un archivo.	Satisfactorio.
	SC 1.3. Llenar demás datos para la colección.	El desarrollador introduce los datos que faltan para crear la colección y presiona el botón	El sistema verifica que los restantes campos no estén vacíos y que fueron llenados correctamente.	Satisfactorio.

		Crear colección.		
	SC 1.4. No se encuentra la dirección de las muestras negativas.	El desarrollador no seleccionó la dirección donde estaban las muestras negativas.	El sistema envía un mensaje donde le informa al desarrollador que ese campo está vacío.	Satisfactorio.
	SC 1.5. No se encuentra la dirección de las muestras positivas.	El desarrollador no seleccionó la dirección donde estaban las muestras negativas.	El sistema envía un mensaje donde le informa al desarrollador que ese campo está vacío.	Satisfactorio.
	SC 1.6. Los restantes campos aún están vacíos.	El desarrollador no ha llenado aún los restantes campos que se necesitan para crear la colección.	El sistema envía un mensaje donde le informa al desarrollador que aún quedan campos vacíos.	Satisfactorio.

Tabla 5 Caso de Prueba #1 CUS Crear colección de muestras de entrenamiento

Caso de Prueba #2 CUS Entrenar cascada de clasificadores haar.

Sesión	Escenario	Acción Realizada	Reacción del Sistema	Resultado
SC 1. Entrenar cascada de clasificadores haar.	EC 1.1. Guardar clasificador satisfactoriamente.	El desarrollador escoge la dirección donde quiere que se guarde el clasificador que se va a entrenar y presiona el botón guardar.	El sistema comprueba que el campo no esté vacío y guarda el clasificador.	Satisfactorio

	SC 1.2. Llenar todos los datos satisfactoriamente.	El desarrollador llena los demás datos que se necesitan para entrenar la colección y presiona el botón entrenar.	El sistema comprueba que no haya quedado ningún campo vacío y entrena el clasificador.	Satisfactorio.
	SC 1.3. Dirección para guardar el clasificador vacía.	El desarrollador ha presionado el botón guardar y no ha escogido donde va a guardar el clasificador.	El sistema envía un mensaje indicando al desarrollador que no ha escogido un directorio para guardar el clasificador.	Satisfactorio.
	SC 1.4. Campos vacíos.	El desarrollador no ha llenado todos los datos que se necesitan para entrenar el clasificador.	El sistema envía un mensaje indicando al desarrollador que ha dejado campos vacíos.	Satisfactorio.

Tabla 6 Caso de Prueba #2 CUS Entrenar cascada de clasificadores haar

Durante el transcurso de la etapa de prueba a la herramienta se diseñaron dos casos de pruebas correspondientes a los casos de usos que tiene la herramienta. En la primera iteración se detectó que la aplicación no ejecutaba los procesos externos que necesitaba para generar la colección de muestras y para entrenar el clasificador. Estos errores fueron corregidos en la segunda iteración, en esta iteración también se detectaron que existían campos sin validar, los errores encontrados fueron que los campos que son solamente de números se podían escribir letras, el usuario podía dejar campos en blanco y el sistema no lo notificaba. Estos errores fueron corregidos en la tercera iteración.

Conclusiones del capítulo

En este capítulo se abordaron las fases de implementación y prueba de la herramienta para la generación de clasificadores de haar y luego de haber generado los artefactos correspondientes en cada fase se llegaron a las siguientes conclusiones parciales: la realización del diagrama de despliegue permitió que se determinara donde se desplegaría la herramienta una vez terminada. Generar el diagrama de componentes de la herramienta permitió una mejor comprensión de la interacción que existe entre estos. Luego de haberle realizado las pruebas de caja negra con el método de partición equivalente a la herramienta se determinaron los problemas con los que aún contaba la herramienta y se le dieron solución realizando otra iteración

Conclusiones generales

Al culminar la presente investigación se llegaron a las siguientes conclusiones generales:

Fue estudiado el proceso actual de generación de los clasificadores haar permitiendo obtener los pasos que se llevan a cabo actualmente para la generación de los clasificadores haar, estos pasos sirvieron de base para la realización de la herramienta desarrollada.

Con la implementación de la Herramienta para la generación de clasificadores haar los desarrolladores pueden generar cascadas de clasificadores haar de una forma sencilla.

Se definieron herramientas y tecnologías actuales que cumplen con la política del proyecto de utilizar software libre y que contribuyeron a un correcto proceso de modelación e implementación de la herramienta.

Se le realizaron pruebas a la herramienta desarrollada y con esta validación se demuestra que la herramienta está apta para usarse por los clientes.

Recomendaciones

Se recomienda que:

- La propuesta resultante de dicha investigación sea aplicada en todos los productos del departamento que la necesiten.
- Se le agregue a la propuesta resultante una nueva funcionalidad: Analizar el rendimiento del clasificador entrenado

Referencias bibliográficas

Andrés, Ing Rolando Rodríguez. 2008. ***Herramientas para el modelado y análisis de procesos.* 2008.**

Barahona-Gonzalez. 2008. *Introducción al software libre.* 2008.

Bustamante-Aginaga-Aybar-Olaizola-Lazacano. 2004. *Aprenda C++ básico.* 2004.

Calleja, Manuel Arias. 2008. *Estándares de codificación.* 2008.

Electrónico, Dirección General de Gobierno. 2006. *Estándares de codificación de sistemas.* 2006.

Figueras, Alejandro Alvares. 2011. *Sistema para el control autónomo de espacio en disco en servidores de media.* 2011.

Gamma-Helm-Johnson-Vlissides. 2003. *Design Patterns.* 2003.

Igual-Medrano. 2010. *Tutorial de OpenCV.* 2010.

informáticos, Dpto de lenguajes y sistemas. 2005. *UML y Diseño: diagramas de clases e interacción.* 2005.

Jacobson-Booch-Rumbaugh. 2005. *El lenguaje unificado de modelado, manual de referencia.* 2005.

—. 2000. *El proceso unificado de desarrollo de software.* 2000.

Larman, Craig. 2003. *UML y patrones.* 2003.

Letelier, Patricio. 2011. *Proceso de desarrollo de software.* 2011.

Mora, Francisco. 2002. *UML: Lenguaje Unificado de Modelado.* 2002.

Nokia. 2012. *Qt Project.* 2012.

Orallo, Enrique Hernandez. 2002. *El lenguaje unificado de modelado.* 2002.

Pressman, Roger S. 2005. *Ingeniería del software. Un enfoque práctico.* 2005.

—. 2010. *Software engineering.* 2010.

RAE, Real Academia de la Lengua Española. 2013. *Video vigilancia.* 2013.

Ralston-Reilly-Hemmendinger. 2000. *Encyclopedia of Computer Science - programming language.* 2000.

Sommerville, Ian. 2007. *Ingeniería de Software.* 2007.

Strouptrup, Bjarne. 2010. *Everything about C++.* 2010.

UCI. 2010. Centro de Geoinformática y Señales Digitales. [Online] 2010.

Valencia, Universidad Politécnica de. 2002. *Introducción a Herramientas CASE.* 2002.

Bibliografía

Andrés, Ing Rolando Rodríguez. 2008. ***Herramientas para el modelado y análisis de procesos.* 2008.**

Barahona-Gonzalez. 2008. *Introducción al software libre.* 2008.

Barreto-Dias-Menezes. 2004. *Human-robot interaction based on Haar-like features and eigenfaces.* 2004.

Bartlett, Marian Stewart. 2004. *Real time face detection and facial expression recognition: development and applications to human computer interaction.* California : s.n., 2004.

Belaroussi-Milgram. 2007. *Face detecting and skin color based tracking: a comparative study.* Paris : s.n., 2007.

Bustamante-Aginaga-Aybar-Olaizola-Lazacano. 2004. *Aprenda C++ básico.* 2004.

Calleja, Manuel Arias. 2008. *Estándares de codificación.* 2008.

Castrillón-Déniz-Guerra. 2007. *ENCARA2: Real-time detection of multiple faces at different resolutions in video streams.* 2007.

Electrónico, Dirección General de Gobierno. 2006. *Estándares de codificación de sistemas.* 2006.

Figueras, Alejandro Alvares. 2011. *Sistema para el control autónomo de espacio en disco en servidores de media.* 2011.

Gamma-Helm-Johnson-Vlissides. 2003. *Design Patterns.* 2003.

Habili, Nariman. 2001. *Automatic segmentation of the face and hands in sign language video sequences.* Australia : s.n., 2001.

Igual-Medrano. 2010. *Tutorial de OpenCV.* 2010.

informáticos, Dpto de lenguajes y sistemas. 2005. *UML y Diseño: diagramas de clases e interacción.* 2005.

Jacobson-Booch-Rumbaugh. 2005. *El lenguaje unificado de modelado, manual de referencia.* 2005.

—. 2000. *El proceso unificado de desarrollo de software.* 2000.

Jones, Viola-Paul-Michael. 2001. *Rapid object detection using boosted cascade of simple features.* 2001.

Larman, Craig. 2003. *UML y patrones.* 2003.

Letelier, Patricio. 2011. *Proceso de desarrollo de software.* 2011.

Lienhart-Kuranov-Pisarevsky. 2002. *Empirical analysis of detection cascades of boosted classifiers for rapid object detection.* 2002.

Lienhart-Maydt. 2002. *An extended set of haar-like features for rapid object detection.* 2002.

Meneses-Barreto-Diaz. 2005. *Face tracking based on haar-like features and eigenfaces.* 2005.

Mohan-Papageorgiou-Poggio. 2001. *Example-based object detection in images by components.* 2001.

Mora, Francisco. 2002. *UML: Lenguaje Unificado de Modelado.* 2002.

Mungia-Grau-Mungia. 2010. *Visca: seguimiento de caras servo-controlado.* 2010.

Nokia. 2012. *Qt Project.* 2012.

OpenCV, Sitio oficial de. 2012. *La librería OpenCV.* 2012.

Orallo, Enrique Hernandez. 2002. *El lenguaje unificado de modelado.* 2002.

Pressman, Roger S. 2005. *Ingeniería del software. Un enfoque práctico.* 2005.

—. 2010. *Software engineering.* 2010.

RAE, Real Academia de la Lengua Española. 2013. *Video vigilancia.* 2013.

Ralston-Reilly-Hemmendinger. 2000. *Encyclopedia of Computer Science - programming language.* 2000.

Sommerville, Ian. 2007. *Ingeniería de Software.* 2007.

Strouptrup, Bjarne. 2010. *Everything about C++.* 2010.

UCI. 2010. Centro de Geoinformática y Señales Digitales. [Online] 2010.

Valencia, Universidad Politécnica de. 2002. *Introducción a Herramientas CASE.* 2002.

Yang, Ming-Hsuan. 2002. *Detecting faces images: A survey.* 2002.

Glosario de términos

Boosted: significa que los clasificadores correspondientes a cada etapa son a la vez complejos y están construidos de clasificadores simples usando una de las diferentes técnicas de boosting o peso por voto.

cámara IP: es una cámara que emite las imágenes directamente a la red sin necesidad de un ordenador.

Cascadas de clasificadores de haar: llamado cascade of boosted classifiers working with haar-like features consiste en varios clasificadores simples que son aplicados subsecuentemente a una región de interés dentro de una imagen, hasta que en alguna etapa la imagen a procesar es rechazada, o todas las etapas son pasadas, es decir, que se obtiene el objeto que se está buscando.

Cascade: significa que el clasificador resultante consiste en varios clasificadores simples que son aplicados subsecuentemente a una región de interés hasta que en alguna etapa el candidato es rechazado o todas las etapas son pasadas.

Haartraining: significa entrenador haar y es la herramienta que se utiliza actualmente para crear la colección de muestras de entrenamiento necesarias para erigir un clasificador.

Muestras negativas: no son más que imágenes que no contienen el objeto que se quiere obtener al final del proceso.

Muestras positivas: no son más que imágenes que contienen el objeto que se quiere obtener al final del proceso.

OpenCV: librería de tratamiento de imágenes, destinada principalmente a aplicaciones de visión por computador en tiempo real.

Reconocimiento de patrones: El reconocimiento de patrones es la ciencia que se encarga de la descripción y clasificación (reconocimiento) de objetos, personas, señales, representaciones, etc. Esta ciencia trabaja con base en un conjunto previamente establecido de todos los posibles objetos (patrones) individuales a reconocer.

Traincascade: significa entrenar cascada y es la herramienta que se usa actualmente para entrenar un clasificador.