****

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

**IntelliRoom: Domótica**

Realizado por:

**Rafael Gómez García**

Para la obtención del título de

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Dirigido por:

**Daniel Cagigas Muñiz**

**Pablo Íñigo Blasco**

Realizado en el departamento de Arquitectura y Tecnología de Computadores (ATC)

Convocatoria de Junio, Curso 2010-2011

Tabla de contenido

[1 IntelliRoom como asignatura 7](#_Toc297501792)

[1.1 ¿Qué es IntelliRoom? 7](#_Toc297501793)

[1.2 ¿Qué se pretende con IntelliRoom? 7](#_Toc297501794)

[1.3 Objetivos que pretendía con el proyecto 7](#_Toc297501795)

[1.4 Habilidades o conocimientos adquiridos con el proyecto 8](#_Toc297501796)

[1.5 Planificación 9](#_Toc297501797)

[1.6 Autoría 10](#_Toc297501798)

[1.7 Licencia 10](#_Toc297501799)

[1.8 Agradecimientos 10](#_Toc297501800)

[2 Arquitectura Hardware 11](#_Toc297501801)

[2.1 Diagrama de despliegue 11](#_Toc297501802)

[2.2 El porqué de esta distribución 11](#_Toc297501803)

[3 Arquitectura Lógica Básica 12](#_Toc297501804)

[3.1 Requisitos funcionales 12](#_Toc297501805)

[3.2 División en Capas 12](#_Toc297501806)

[3.2.1 Presentación 12](#_Toc297501807)

[3.2.2 Lógica de negocio 12](#_Toc297501808)

[3.2.3 Datos 13](#_Toc297501809)

[4 Electrónica 15](#_Toc297501810)

[4.1 Introducción 15](#_Toc297501811)

[4.2 Conceptos de electrónica necesarios 15](#_Toc297501812)

[4.2.1 Duty Cycle 15](#_Toc297501813)

[4.2.2 PWM 16](#_Toc297501814)

[4.3 Elección del microcontrolador 16](#_Toc297501815)

[4.3.1 Requisitos del microcontrolador 16](#_Toc297501816)

[4.3.2 PIC 16F1828 / 16F1824 17](#_Toc297501817)

[4.3.3 Arduino 19](#_Toc297501818)

[4.4 Control de iluminación 20](#_Toc297501819)

[4.4.1 Circuito eléctrico 20](#_Toc297501820)

[4.4.2 Aislador de tensiones 21](#_Toc297501821)

[4.5 Control de Dispositivos 25](#_Toc297501822)

[4.5.1 Circuito eléctrico 25](#_Toc297501823)

[4.6 Código Arduino 27](#_Toc297501824)

[4.6.1 Pseudocódigo 27](#_Toc297501825)

[4.6.2 Código 28](#_Toc297501826)

[4.7 Circuito completo 29](#_Toc297501827)

[5 Capa de presentación 31](#_Toc297501828)

[5.1 Requisitos de la capa de presentación 31](#_Toc297501829)

[5.2 Consola 31](#_Toc297501830)

[5.3 GUI 32](#_Toc297501831)

[6 Lógica de negocio: IntelliRoom 34](#_Toc297501832)

[6.1 Lista de funciones 34](#_Toc297501833)

[6.2 Sistema de eventos 38](#_Toc297501834)

[6.3 Programador 39](#_Toc297501835)

[7 Módulo Arduino 40](#_Toc297501836)

[7.1 Requisitos funcionales 40](#_Toc297501837)

[7.2 Estructuración “Arduino” 40](#_Toc297501838)

[7.3 protocolo de mensajes 41](#_Toc297501839)

[7.3.1 Funciones de control de color 43](#_Toc297501840)

[7.3.2 Funciones de control de dispositivos 43](#_Toc297501841)

[7.3.3 Configuración puerto serie 44](#_Toc297501842)

[7.4 Funciones relevantes 44](#_Toc297501843)

[7.4.1 Creación de conexión Serial 44](#_Toc297501844)

[7.4.2 Envío de un mensaje 45](#_Toc297501845)

[8 Módulo Voice 47](#_Toc297501846)

[8.1 Introducción 47](#_Toc297501847)

[8.2 Requisitos funcionales 47](#_Toc297501848)

[8.3 Estructura del módulo 48](#_Toc297501849)

[8.4 SAPI 49](#_Toc297501850)

[8.5 Configuración de SAPI 51](#_Toc297501851)

[8.5.1 Reconocedor del habla 51](#_Toc297501852)

[8.5.2 Sintetizador de voz 53](#_Toc297501853)

[8.6 Gramáticas en SAPI 55](#_Toc297501854)

[8.7 Como generar una nueva gramática en IntelliRoom 56](#_Toc297501855)

[8.8 Casos de uso relevantes 58](#_Toc297501856)

[8.8.1 Añadir gramática por documento XML 58](#_Toc297501857)

[8.8.2 Línea de proceso de reconocimiento de un comando 59](#_Toc297501858)

[9 Módulo Media 61](#_Toc297501859)

[9.1 Introducción 61](#_Toc297501860)

[9.2 WMP SDK 61](#_Toc297501861)

[9.3 Requisitos funcionales 61](#_Toc297501862)

[9.4 Diagrama de diseño 61](#_Toc297501863)

[9.5 Casos de usos relevantes 62](#_Toc297501864)

[9.5.1 Cargar librería de música 63](#_Toc297501865)

[10 Módulo Camera 64](#_Toc297501866)

[10.1 OpenCV / EmguCV 64](#_Toc297501867)

[10.2 Requisitos funcionales 64](#_Toc297501868)

[10.3 Estructura del módulo. 64](#_Toc297501869)

[10.4 Métodos Relevantes. 67](#_Toc297501870)

[10.4.1 Calculo de movimiento 67](#_Toc297501871)

[10.4.2 Reconocimiento facial 68](#_Toc297501872)

[10.4.3 Cálculo de iluminación 69](#_Toc297501873)

[11 Módulo Utils 70](#_Toc297501874)

[11.1 Requisitos funcionales 70](#_Toc297501875)

[11.2 Estructura de diseño 70](#_Toc297501876)

[11.3 Weather API 71](#_Toc297501877)

[11.3.1 Funcionamiento de Google Weather 71](#_Toc297501878)

[11.3.2 La alternativa: Yahoo! Weather 73](#_Toc297501879)

[11.4 Métodos relevantes. 73](#_Toc297501880)

[12 Capa de Datos 74](#_Toc297501881)

[13 Anexo: Instalación de IntelliRoom 76](#_Toc297501882)

[13.1 Requisitos Software 76](#_Toc297501883)

[13.2 Requisitos Hardware 76](#_Toc297501884)

[13.3 Organización del directorio principal del programa 76](#_Toc297501885)

[14 Anexo: Referencias 79](#_Toc297501886)

Tabla de ilustraciones

[1 Tabla de planificacion de proyecto 10](#_Toc297400533)

[2 Esquema básico de Arquitectura lógica 14](#_Toc297400534)

[3 Ejemplos de Duty cycle 15](#_Toc297400535)

[4 Ecuación Duty cycle 16](#_Toc297400536)

[5 Comportamiento PWM 16](#_Toc297400537)

[6 Encapsulado del modelo PIC16F1824 17](#_Toc297400538)

[7 Encapsulado del modelo PIC16F1828 18](#_Toc297400539)

[8 Configuración del 16F1828 para el proyecto 18](file:///C:\Users\Rafgomgar\Desktop\PFC\intelli-room\Documentacion\Documentacion.docx#_Toc297400540)

[9 Prototipo de programador universal 19](#_Toc297400541)

[10 Modelo RGB Elegido para el proyecto 20](#_Toc297400542)

[11 encapsulado del transistor y transformador de corriente 21](#_Toc297400543)

[12 Circuito Aislador de tensión 22](#_Toc297400544)

[13 Circuito y encapsulado del rectificador 22](#_Toc297400545)

[14 diseño del circuito en placa de prototipo 23](#_Toc297400546)

[15 Esquemático del control de iluminación 24](#_Toc297400547)

[16 Fotografía del circuito en la placa de prototipo 24](#_Toc297400548)

[17 Una ilustración y una fotografía de Relés 25](#_Toc297400549)

[18 Captura de pantalla de la versión de consola 32](#_Toc297400550)

[19 Captura de pantalla de la versión con formulario windows forms 33](#_Toc297400551)

[20 Estructura del paquete arduino 41](#_Toc297400552)

[21 Traza de ejemplo de envío de mensaje 42](file:///C:\Users\Rafgomgar\Desktop\PFC\intelli-room\Documentacion\Documentacion.docx#_Toc297400553)

[22 Estructura del paquete voice 49](#_Toc297400554)

[23 Estructura de un objeto Grammar 55](#_Toc297400555)

[24 Ejemplo de estructura gramatical 56](#_Toc297400556)

[25 Estructura gramatical del ejemplo anterior 57](#_Toc297400557)

[26 Propiedades y métodos del objeto *SpeechRecognizerEventArgs* 59](#_Toc297400558)

[27 Estructura del paquete media 62](#_Toc297400559)

[28 Ejemplo de haarcascade con XML de rostros frontales 66](#_Toc297400560)

[29 Estructura del paquete camera 67](#_Toc297400561)

[30 Estructura del paquete utils 71](#_Toc297400562)

[31 Respuesta dada por google weather a una petición 72](#_Toc297400563)

[32 Estructura del paquete de datos 74](#_Toc297400564)

[33 Directorios de Intelli-Room 75](file:///C:\Users\Rafgomgar\Desktop\PFC\intelli-room\Documentacion\Documentacion.docx#_Toc297400565)

[34 Raiz principal de Intelli-Room 77](#_Toc297400566)

# Introducción

En este capítulo no abordaremos ningún aspecto técnico de IntelliRoom como proyecto software/hardware sino que lo veremos desde la perspectiva de la asignatura *proyecto fin de carrera* contestando a la pregunta obvia “¿qué es IntelliRoom?” y a otras más docentes del tipo ¿qué me aportó como futuro ingeniero?, ¿qué conocimientos adquirí? así como que motivaciones fueron sustento de este proyecto.

## ¿Qué es IntelliRoom?

IntelliRoom es un proyecto de domótica. Según Wikipedia la [domótica](http://es.wikipedia.org/wiki/Domótica) es un conjunto de sistemas capaces de automatizar una vivienda, aportando servicios de gestión energética, seguridad, bienestar y comunicación.

En nuestro caso nos hemos centrado en las funciones de bienestar o confort y algún aspecto de seguridad. Vamos a poner algunos ejemplos de usos (integrados dentro del artículo de la Wikipedia y que extenderemos ampliamente a lo largo este documento).

* Iluminación:
  + Apagado general de todas las luces de la vivienda
  + Automatización del apagado y encendido en cada punto de luz.
  + Regulación de la iluminación según el nivel de luminosidad ambiente
* Automatización de distintos sistemas, instalaciones o equipos dotándolos de control eficiente y de fácil manejo
* Control vía Internet
* Gestión multimedia y del ocio electrónico.
* Sensor de presencia.

En general la idea principal es la de disponer de ciertas características que sean administradas por un ordenador principal (de uso general) que procese la información recibida por los sensores (micrófonos, cámara, sensores de temperatura, etc.) y que la redistribuya a los periféricos (control de luz, dispositivos, altavoces, etc.).

## Objetivos del proyecto

Lo que pretende IntelliRoom es, además de satisfacer la necesidad de completar un proyecto fin de carrera para obtener el título, centrarme en un campo que permita mejorar mis competencias profesionales, estos son los objetivos:

* Profundización en la [plataforma .NET](http://es.wikipedia.org/wiki/.NET), en concreto su lenguaje principal [C#](http://es.wikipedia.org/wiki/C_Almohadilla). Aplicando conceptos del paradigma de la programación orientada a objeto en otras tecnologías.
* Documentarme sobre el tratamiento de imágenes y utilización de sus conceptos con la implementación de OpenCV.
* Conocer conceptos sobre tratamiento de reconocimiento del habla y en concreto utilizar SAPI.
* Adquirir conocimientos de microcontroladores y electrónica.

## Habilidades o conocimientos adquiridos con el proyecto

Con una proyección bastante final del proyecto he confeccionado esta lista de habilidades, conocimiento en áreas científicas y tecnologías concretas que he ido adquiriendo a lo largo del desarrollo del proyecto.

* Tecnologías:
  + Framework .NET y C#
    - Reflexión.
    - Linq.
    - Hilos o Threads.
    - Peticiones HTTP.
    - Lenguaje XML.
    - Serialización.
    - Comunicación por puerto serie.
    - Suscripción y creación de eventos.
    - Manejo de monitores.
    - Internacionalización del sistema.
    - Utilización de librerías externas/Wrappers.
  + Visual Studio 2010
    - IntelliTrace.
    - Funciones de arquitectura y modelado.
  + Arduino
    - Utilización de la plataforma
    - Profundización del hardware.
    - Desarrollo con su librería llamada Wiring: Sintaxis, funciones y librerías externas.
  + OpenCV
  + SAPI
  + XML
* Conocimiento de áreas:
  + Lenguaje funcional
  + Tratamiento de imagen y sonido
  + Electrónica:
    - Uso de componentes básicos en la práctica.
      * Resistencias.
      * Transistores.
      * Condensadores
      * Relés.
      * Diodos.
      * Reguladores de tensión.
      * LEDs.
      * Fuentes de alimentación.
      * Placas de prototipo.
    - Circuitos
      * Aisladores de tensión.
      * Rectificadores.
  + Microcontroladores
* Habilidades
  + Uso de debugger.
  + Busqueda en diversas fuentes de información como Google, datasheets o ayuda de MSDN.
  + Generar documentación completa de un proyecto.

## Planificación

A continuación, expondré una tabla con las duraciones de las tareas estimadas, y cuando tiempo le fue aplicada a cada una de esas tareas.

|  |  |  |
| --- | --- | --- |
| Tarea | Tiempo estimado | Tiempo empleado |
| Definir Arquitectura del proyecto | 10 | 25 |
| Definir detalles a nivel de casos de uso | 15 | 25 |
| Investigar sobre framework .NET 4.0 | 25 | 25 |
| Investigar sobre tratamiento de sonido | 20 | 5 |
| Investigar sobre tratamiento de imagen | 30 | 15 |
| Investigar sobre SAPI | 20 | 10 |
| Investigar sobre OpenCV | 20 | 15 |
| Investigar sobre EmguCV | 5 | 10 |
| Investigar sobre Arduino | 10 | 10 |
| Adquirir conocimientos de electrónica | 20 | 20 |
| Confección de circuitos | 25 | 30 |
| Montaje de circuitos | 20 | 25 |
| Testeo de circuitos | 10 | 10 |
| Puertos serie | 5 | 5 |
| Implementación de IntelliRoom | 150 | 200-250 |
| Testeo de IntelliRoom | 50 | 75 |
| Generación de la documentación | 100 | 75 |

1 Tabla de planificacion de proyecto

## Organización del documento

El documento está dividido en 12 capítulos y 2 anexos.

[TODO]

## Autoría

El autor de este proyecto es Rafael Gómez García, el cual posee los derechos de autor, alumno de Ingeniería Técnica en informática de gestión en la Escuela Técnica Superior de Ingeniería Informática (ETSII) de la Universidad de Sevilla. Este proyecto conforma el proyecto Fin de Carrera del autor y tiene como tutores a los profesores Daniel Cagigas Muñiz y Pablo Íñigo Blasco, del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Sevilla.

Puede contactar con el autor del proyecto por correo electrónico en la siguiente dirección:

[Rafgomgar@gmail.com](mailto:Rafgomgar@gmail.com)

## Licencia

El proyecto IntelliRoom se rige por la licencia *GNU Lesser General Public License 3.0*. El contenido completo de la licencia (en inglés) está disponible en la siguiente dirección:

<http://www.gnu.org/licenses/lgpl-3.0-standalone.html>

## Agradecimientos

Concluir con que mi labor se habría visto retrasada sin la ayuda de Manuel Caballero, Pablo Íñigo Blasco, Daniel Cáceres Maña y, en general, mis compañeros del club .NET.

# Arquitectura Hardware

## Diagrama de despliegue

2 Diagrama de despliegue

<<device>>

: Ordenador Personal

{OS= Windows7/Vista}

IntelliRoom.exe

Ratón

Teclado

Tarjeta de red

Altavoces

Micrófono

Cámara

IntelliRoom.dll

Voice.dll

Camera.dll

MediaPlayer.dll

...



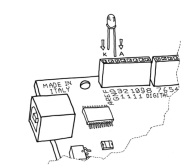
<<device>>

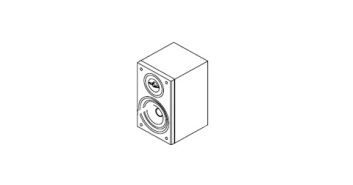
: Arduino

Codigo Arduino.pde

Control de dispositivos

Control de iluminación





<<puerto serie>>

## El porqué de esta distribución

# Arquitectura Lógica Básica

Para comprender la arquitectura lógica de la aplicación desde un punto de vista muy abstracto, empezaremos definiendo los requisitos funcionales y a lo largo del capítulo definiremos cada requisito funcional con una sección en su capa.

## Requisitos funcionales

* Interactuación máquina-usuario por reconocimiento de voz y por ejecución de comandos.
* Medición de luz, sensor de movimiento y detector de presencia.
* Control de color e iluminación del espacio.
* Administrador de dispositivos eléctricos.
* Control ambiental de música o sonidos.
* Funciones varias como: alarmas, situación climatológica…

## División en Capas

He dividido la arquitectura lógica en tres niveles básicos siguiendo la estructura básica de [programación por capas](http://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas), quedando, tal y como podemos ver en la figura de la arquitectura de debajo: capa de presentación, capa de lógica de negocio y capa de datos, vamos a hacer una breve descripción de las capas y sus usos en el proyecto.

### Presentación

Es la más próxima al usuario (también se la denomina "capa de usuario"), presenta el sistema al usuario, le comunica y captura la información en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). También es conocida como interfaz gráfica y debería ser "amigable" (entendible y fácil de usar). Esta capa se comunica únicamente con la capa de negocio y tiene:

* Console: Es un terminal de consola que interpreta comandos y muestra resultados de ejecución de esos comandos e información adicional. Veremos más adelante que tipos de comandos pueden ser ejecutados y como se ejecutan.
* GUI: Una implementación mejorada de la consola de comandos, programada con la librería Windows Forms. En funcionalidad es igual que la consola, pero la información está más organizada.
* Además de esta alternativa podríamos publicar un servicio web para que consumiera también información del sistema u otra GUI en Windows Forms o en [Windows Presentation Foundation](http://es.wikipedia.org/wiki/WPF).

### Lógica de negocio

Es el punto central de la aplicación, que consta de dos grandes partes: nuestro motor de la aplicación, IntelliRoom, y sus módulos pensados para incrementar su funcionalidad.

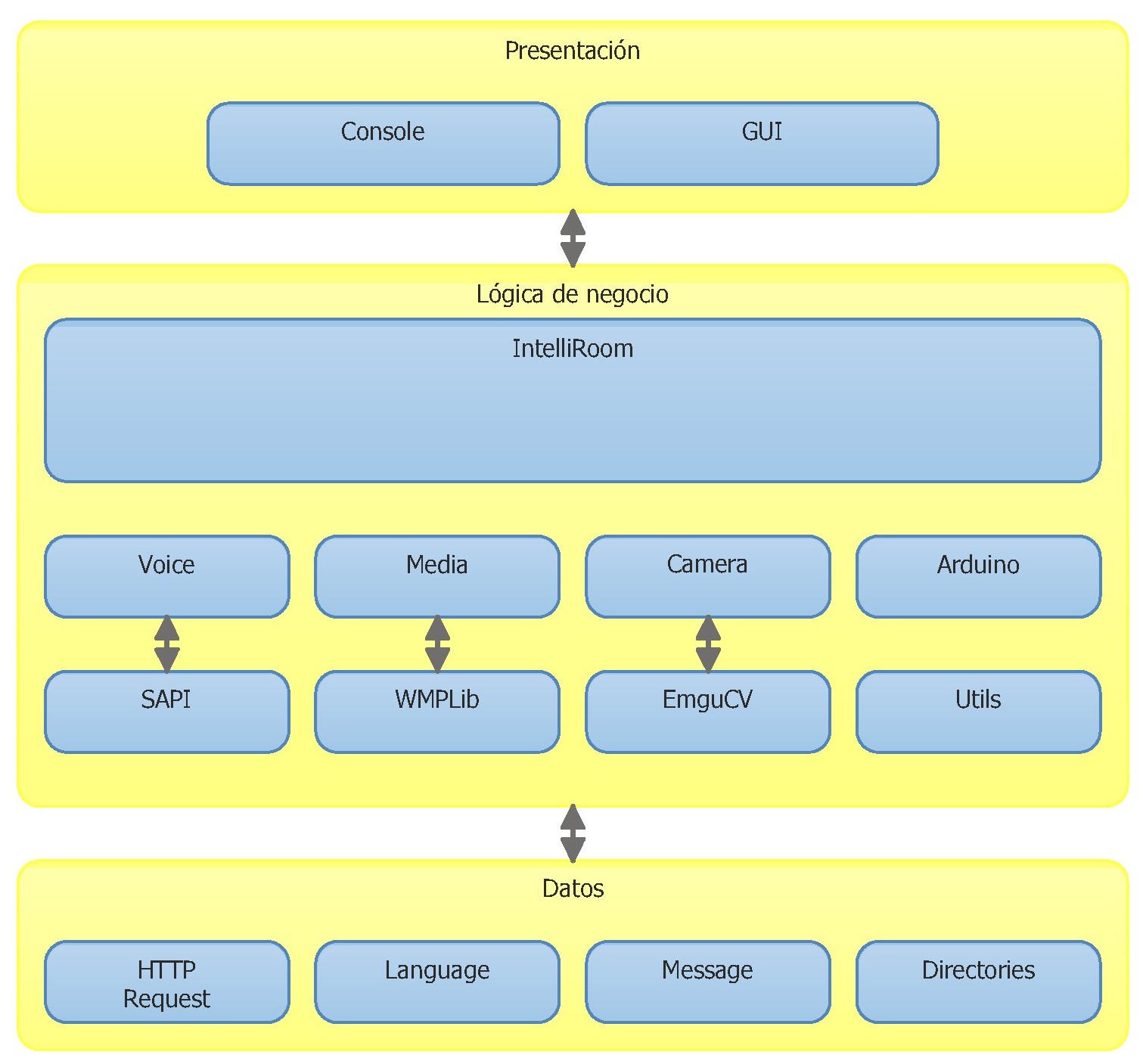
IntelliRoom ofrece una interfaz, y su implementación, usada por la capa de presentación. Esta interfaz (*IIntelliRoom*) tiene la funcionalidad de la aplicación que es usable por la capa de presentación. Además de ello gestiona la interpretación de voz, reflexión y otros temas que abordaremos con más tranquilidad en el capítulo dedicado a él. Además de ello la lógica de negocio contiene los siguientes módulos:

* Voice: Es, en grandes rasgos, una abstracción de la API de Microsoft SAPI (Speech API) su principal objetivo es la interpretación de ordenes usuario-maquina mediante comandos de voz y sintetización de texto plano.
* Media: Modulo encargado de reproducción de la biblioteca de música que tenga el usuario en la máquina, la librería que usamos por debajo WMPLib, permite sincronización con la biblioteca contenida en la aplicación Windows Media Player, reproducción y control de archivos locales, enlaces remotos, carpetas en red, etc.
* Camera: Es el módulo de procesamiento de imágenes para detección de movimiento, luminosidad de la habitación y reconocimiento facial o de personas. Por debajo usa Emgu CV, que es un wrapper de OpenCV.
* Arduino: Módulo de conexión y envío de órdenes a la plataforma Arduino, gestiona la funcionalidad de ambientación de la habitación por colores y encendido/apagado de dispositivos del hogar.
* Utils: Es un módulo que contiene funcionalidad variada como puede ser captura de información de clima, alarmas…

### Datos

La tercera capa de la aplicación es la encargada de gestionar la información persistente o de hacer consultas al exterior de la aplicación.

* HTTP Request: Realiza peticiones y gestiona respuestas HTTP.
* Directories: Gestiona la creación de carpetas, devuelve las rutas a directorios y archivos.
* Language: proporciona las bases para la internacionalización de la aplicación.
* Messages: Gestiona los mensajes de información y error dentro del sistema.



3 Esquema básico de Arquitectura lógica

# Electrónica

## Introducción

Para solucionar el objetivo de control de dispositivos y de iluminación se barajaron diversas alternativas de componentes electrónicos, sin embargo muchas fueron descartadas porque no se ajustaban de manera adecuada a los requisitos o a las pretensiones del proyecto. Sin embargo su estudio fue importante detectar carencias y mejoras en el sistema. [Se describe brevemente estas alternativas].

Este capítulo se divide 2 grandes partes: la dedicada al control de iluminación y la de control de dispositivos, aunque antes de describir esas partes entraremos en conceptos que son necesarios para poder seguir el documento.

## Conceptos de electrónica necesarios

### Duty Cycle

El ciclo de trabajo o duty cycle es la relación de tiempo que permanece la onda periódica (supongamos cuadrada) en valores positivos, veamos unos ejemplos de valores de ciclo de trabajo:



4 Ejemplos de Duty cycle

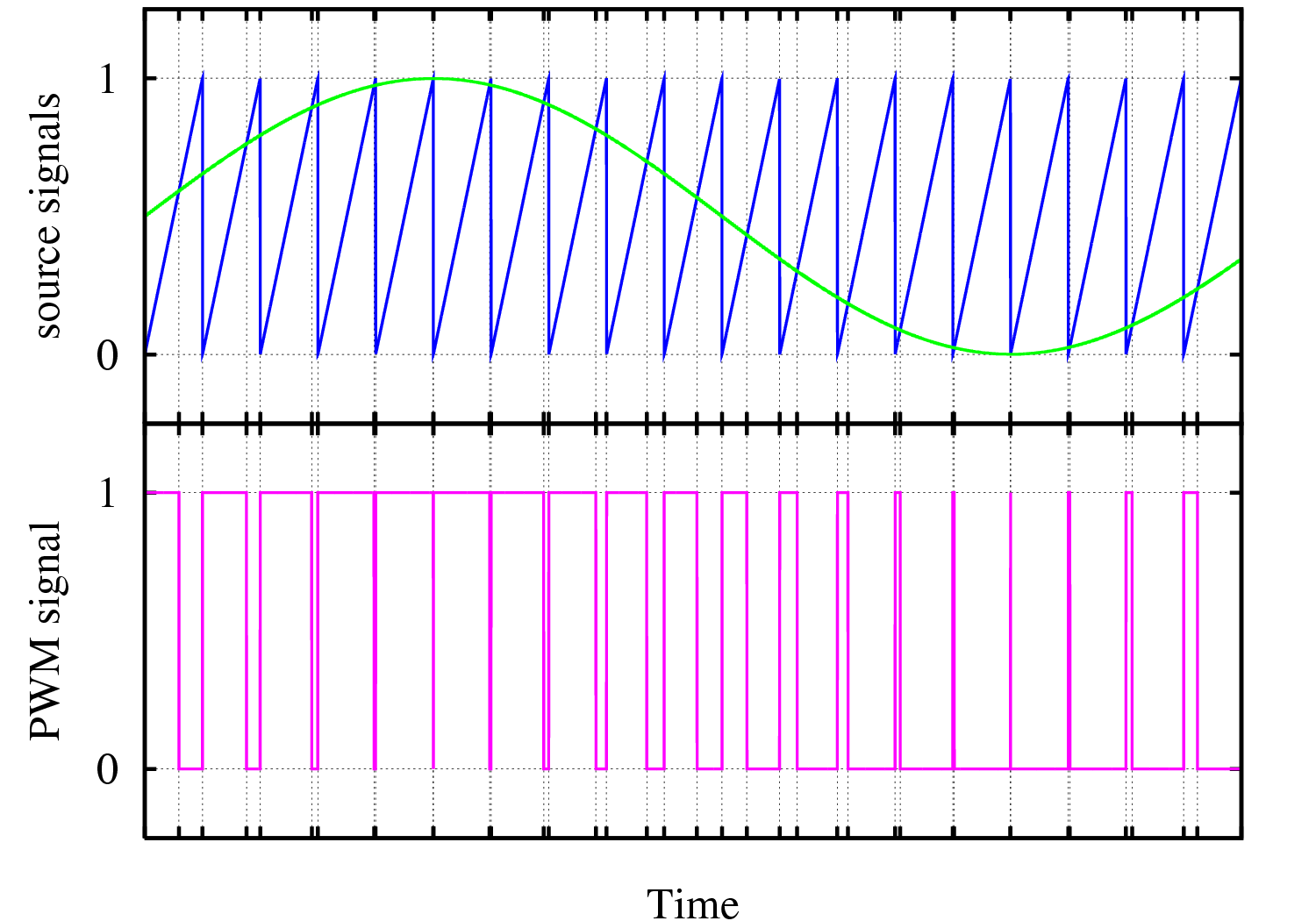
De esta manera concluimos, muy intuitivamente, que el cálculo del duty cycle es:

5 Ecuación Duty cycle

### PWM

PWM (pulse-width modulation) o modulación por ancho de pulsos es una técnica en la que modificamos el ciclo de trabajo, o duty cycle, de una señal periódica para controlar la cantidad de energía que se envía a una carga.

Las aplicaciones del PWM son variadas: comunicaciones, efectos de sonido, amplificación, regulación de voltaje, etc.



6 Comportamiento PWM

## Elección del microcontrolador

Una vez visto los conceptos que iba a necesitar para conseguir el control de luz y control de dispositivos ya solo quedaba preguntarme ¿Qué microcontrolador es el idóneo para nuestro trabajo?

### Requisitos del microcontrolador

El punto de partida para comenzar la búsqueda de un microcontrolador que cumpliera con las características mínimas, fue confeccionar una lista con todo que necesitaría para afrontar el proyecto.

Aclarar, además, que los PWM los usaremos como regulador de voltaje que le llega al LED. Usaremos tres salidas PWM puesto que necesitamos de controlar intensidad de las tres componentes (roja, verde y azul).

A continuación, se deja la lista que utilice a modo de filtro para la búsqueda de un microcontrolador:

* 3 PWM: para las componentes R G B del LED.
* 8 bits: Puesto que 28 valores para la componente roja, 28 para la verde y 28 para la azul hacían un total de más de 16 millones de colores posibles, en concreto 16.777.216 colores, no es necesario más precisión que la dada con 8 bits de resolución.
* Al menos 5 ó 6 salidas lógicas para la conexión de los dispositivos (1 salida lógica por cada uno de ellos).
* Comunicación serie y otros patillajes.
* Económico.

### PIC 16F1828 / 16F1824

Tras realizar una búsqueda por diferentes fabricantes, tras la consulta de diversas fuentes, se planteó el uso de dos microcontroladores relativamente nuevos de la empresa Microchips: el PIC 16F1828 y el PIC 16F1824.

Viendo su patillaje, las salidas de PWM están menos replicadas de las salidas lógicas, bajo esta premisa usaremos el 1828 para el control de iluminación y el 1824 para el control de dispositivos. Ambos microcontroladores son iguales en características y su diferencia es determinada por el hecho de que el encapsulado de 16F1828 está menos multiplexado que el del 16F1824, como podemos ver en las figuras a continuación:



7 Encapsulado del modelo PIC16F1824



8 Encapsulado del modelo PIC16F1828

Ambos microcontroladores cumplen y superan los requisitos definidos en el apartado 4.3.1: tenían 4 + 2 timers para hacer PWM (4 timers y 2 de ellos con salidas clonadas), 8 bits, 12 salidas lógicas en el modelo 1824, comunicaciones y un precio no superior a los 80 céntimos.

Tras analizar su encapsulado, y trabajar con diferentes configuraciones de patillado. Buscaba una configuración que nos permitiera usar toda la funcionalidad que necesitábamos usando cada patilla para una finalidad. La configuración del microcontrolador 16F1828 quedaría de la siguiente manera para el sistema de control de luz:

* VDD y VSS: Alimentación.

9 Configuración del 16F1828 para el proyecto

* OSC1, OSC2: Cristal de cuarzo.
* MCLR/VPP, ICSPDAT, ICSPCLK: Programación.
* CCP1 al CCP4: PWMs.
* TX, RX: Interfaz con puerto serie.
* SDI, SDO, SCK, INT y uno de los terminales libres: futura interfaz con un transceptor por radio. El INT se conectaría al IRQ del transceptor, de forma que cuando haya datos para procesar se avise al microcontrolador y estos puedan ser leídos.
* DACOUT: Salida del DAC, podría servir para emitir sonidos, por ejemplo si perdemos la conexión por radio.

Ahora solo nos quedaba entrar en la programación del microcontrolador. Instalé el IDE de Microchip y, dado que hablamos de un modelo relativamente nuevo, el IDE ofrecía soporte parcial a los microcontroladores y, en su momento, no encontré la creación de un compilador en C para estos microcontroladores.

A raíz de este problema se utilizó durante un tiempo un simulador para PICs, para desarrollar código. Tras escribir el código ensamblador para la comunicación con el microcontrolador y ver que las pruebas en el simulador funcionaban llegamos al siguiente problema *¿Cómo programamos el microcontrolador?*

Los programadores para PIC son caros y no quería hacer una inversión en estos programadores para los muy limitados usos que les iba a dar. Por estas razones Manuel Caballero me ayudo a buscar información y diseñó de un posible programador universal, dejo el último esquemático a modo de curiosidad de un trabajo que quedó inconcluso:



10 Prototipo de programador universal

Por los problemas mencionados decidí que era mucho más costoso utilizar un microcontrolador que requería de programadores que eran caros o difíciles de fabricar. Además, mi formación en electrónica ralentizaría la evolución del proyecto sin poder asegurar que este programador funcionara tras dedicarle muchas horas de trabajo futuras, por lo que cambie la trayectoria del proyecto dirigiéndome a Arduino.

### Arduino

Arduino es una plataforma de hardware libre basada en una placa con un microcontrolador, en mi caso el ATMega328 para el modelo Arduino UNO, y un entorno de desarrollo que implementa el lenguaje de programación C y utiliza una librería llamada Wiring. Wiring tiene un conjunto de funciones muy simples e intuitivas que permite, en pocas horas, desarrollar proyectos completos para Arduino.

Su placa tiene una entrada/salida por puerto serie (USB) para una fácil programación. Además tiene las siguientes características:

* 14 pines digitales configurables a entrada o salida.
* Puertos series Rx y Tx.
* 2 Pines para interrupciones externas.
* 6 PWM con 8 bits de resolución.
* Pines de comunicación.
* 6 entradas analógicas.
* Otras características

Aunque uno de nuestros pilares en los requisitos del microcontrolador era hacerlo económico y aunque el precio de Arduino asciende a unos 25 euros, sale bastante rentable puesto que ya tenemos a nuestra disposición una placa con funcionalidad para que el microcontrolador pueda ser programado y un microcontrolador con un bootloader ya instalado y un sistema bastante completo amparado por una comunidad bastante amplia de usuarios que aportan contenido constantemente.

En un futuro sería posible adquirir otros ATMega328. Programarlos en la placa para, posteriormente, ponerlo en otra placa diseñada por nosotros. En SparkFun se puede adquirir el ATMega328 con el bootloader instalado por menos de 4 euros.

## Control de iluminación

Para este apartado comenzaremos por una descripción básica y rápida de todos los materiales, algunos problemas que fueron encontrados y como fueron solucionados.

### Circuito eléctrico

Tras documentarme sobre los diodos LEDs que abundan en nuestro mercado (tipo de luminancia, colores, potencia, ángulo de radiación) concluí que necesitaba un LED RBG o tres LEDs emparejado (rojo, azul y verde). Tras analizar varios modelos me decante por el siguiente modelo:



11 Modelo RGB Elegido para el proyecto

La cinta contiene 54 LEDs (18 por cada color) con el consumo siguiente:

Componente R (Roja): 100 mA a 12 V

Componente G (Verde): 87 mA a 12 V

Componente B (Azul): 80 mA a 12 V

Haciendo un **total de 240 mA a 12 V**

Estos diodos leds requieren ser alimentados por alguna fuente alternativa puesto que Arduino ofrece 40 mA por PWM a 5 V y como hemos visto, necesitamos 240mA a 12 V. No podríamos conectarlo directamente, ya que:

1. Podría terminar quemándose el microcontrolador, la placa del PC o los LEDs.
2. En caso de no hacerlo los LEDs no se iluminarían demasiado.

Así que necesitamos un transformador de corriente. En mi caso utilice uno que generaba (según su identificación) 9V a 300mA.

### Aislador de tensiones

Para aislar Arduino de la nueva fuente utilizamos unos transistores NPN, en concreto 3 transistores modelo 2n2222, uno por cada componente.



12 encapsulado del transistor y transformador de corriente

Quedando el circuito aislador de la siguiente manera:



13 Circuito Aislador de tensión

La resistencia “R” de 1KΩ es para que no pase intensidad muy elevada mientras el transistor esté en estado de saturación.

El circuito sería montado 3 veces para el apartado de iluminación, una por cada una de sus componentes. Y puesto que la componente azul y la verde consumen menos intensidad (80-87mA frente a 100mA de la roja) vamos a añadir una resistencia de 220Ω en el colector del 2n2222 para regular la salida de estas dos.

Por ultimo comentar que los diodos leds tenían un positivo común y las tres componentes van conectadas a los transistores (como veremos en el diagrama incluido un poco más adelante).

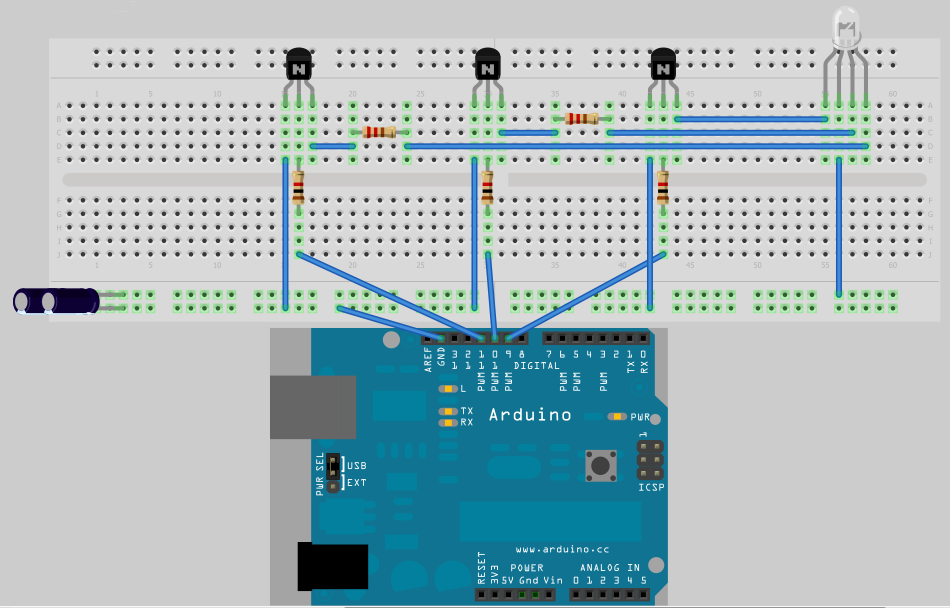
Una vez conectado todos los componentes nos dimos cuenta que los leds sufrían un parpadeo bastante molesto, la razón (aunque tardo en llegar) era que el transformador transformaba de 230V a 50Hz de alterna a 9V a 300mA en ALTERNA**.** Para solucionar esto, pusimos entre la placa de prototipo y el transformador un circuito rectificador como el de la figura, aunque posteriormente compramos un encapsulado.



14 Circuito y encapsulado del rectificador

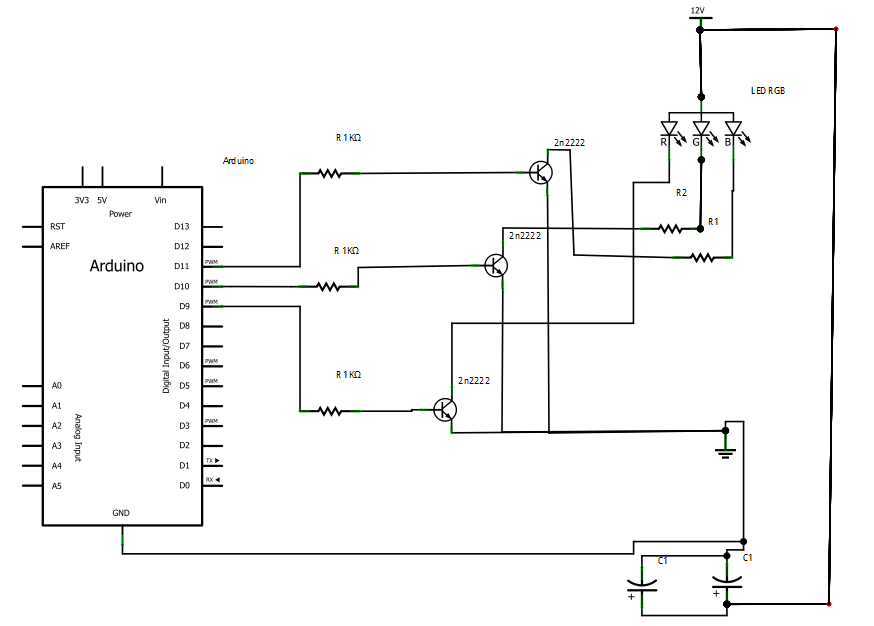
Además añadimos dos condensadores en paralelo de 100µF al final del rectificador, obteniendo así una señal continua de casi 17 voltios, algo más de lo esperado en voltaje, pero sin problemas de funcionamiento.

El circuito en placa de prototipo, obviando el circuito rectificador que quedaría conectado a la 2ª patilla de la ilustración del LED RGB, y enfatizando en el aislamiento y conexionado de leds.

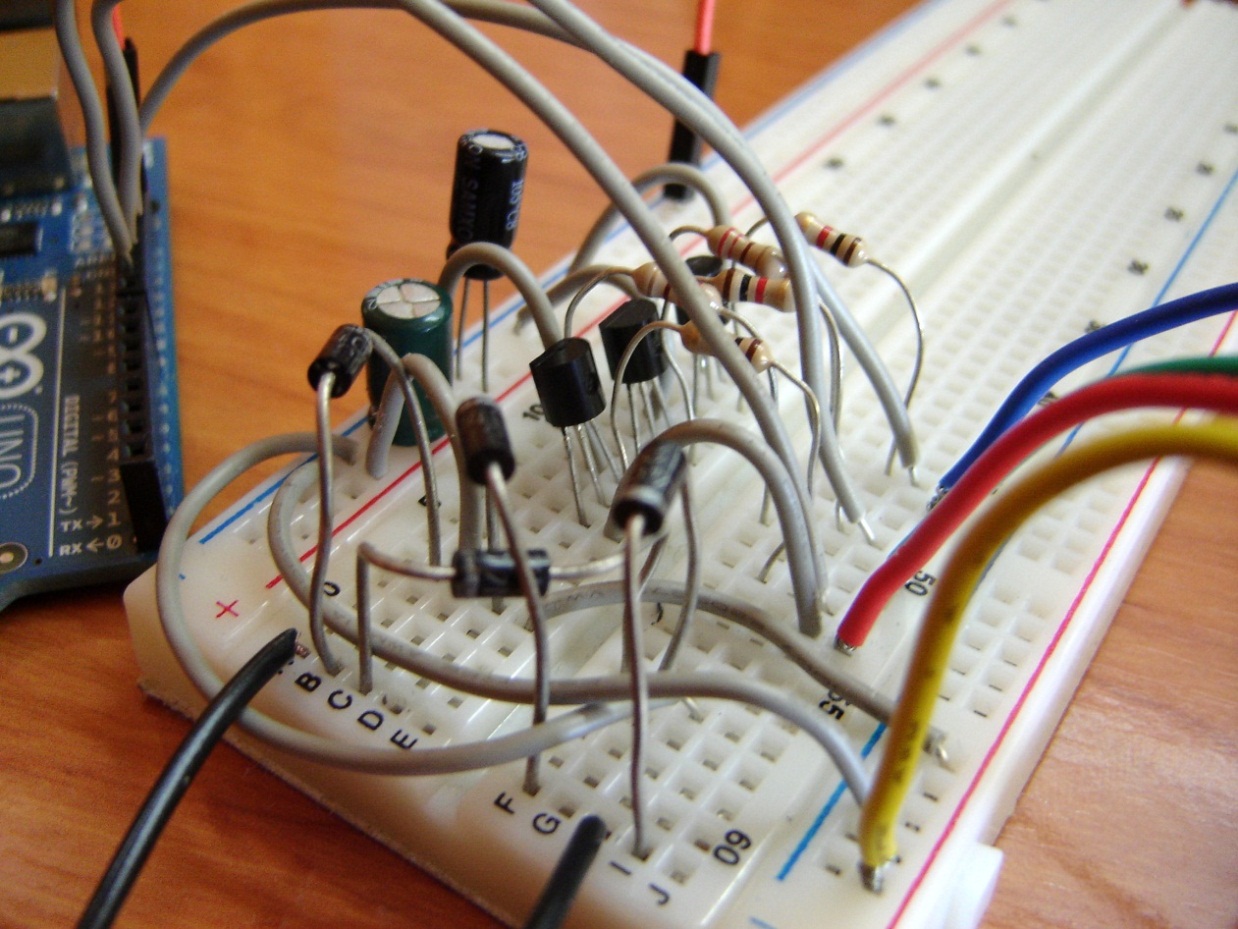


15 diseño del circuito en placa de prototipo

Por ultimo añadimos el circuito en esquemático y una fotografía del resultado final del control de iluminación.



16 Esquemático del control de iluminación



17 Fotografía del circuito en la placa de prototipo

## Control de Dispositivos

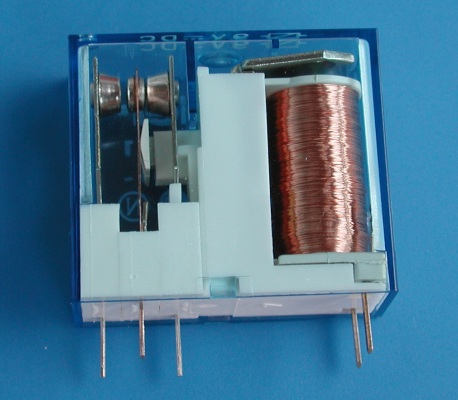
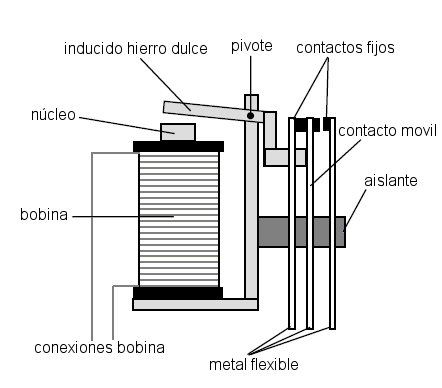
El control de dispositivos eléctricos del hogar es otro de los apartados de IntelliRoom. Su finalidad es la de tener control del encendido y apagado de dispositivos conectados en una regleta de enchufes, previamente preparada para su fin. Además, este concepto puede ser trasportado a la instalación casera.

En un comienzo íbamos a usar un PIC de Microchip. Pero finalmente, al utilizar el ATMega328 y comprobar que teníamos muchas salidas lógicas a nuestra disposición que iban a ser desaprovechadas se decidió por que fuera una placa Arduino la encargada de solucionar también el control de dispositivos.

### Circuito eléctrico

Para la circuitería del control de dispositivo usaremos técnicas muy parecidas a las del control de luz pero algunos componentes distintos. Para este circuito usaremos componentes usados anteriormente (transistores NPN, resistencias, rectificador, etc.). Pero para el aislamiento entre la corriente generada y rectificada del transformador y los dispositivos usaremos relés.

Un relé es un dispositivo electromecánico. Funciona como un interruptor controlado por un circuito eléctrico en el que, por medio de una bobina y un electroimán, se acciona un juego de uno o varios contactos que permiten abrir o cerrar otros circuitos eléctricos independientes.



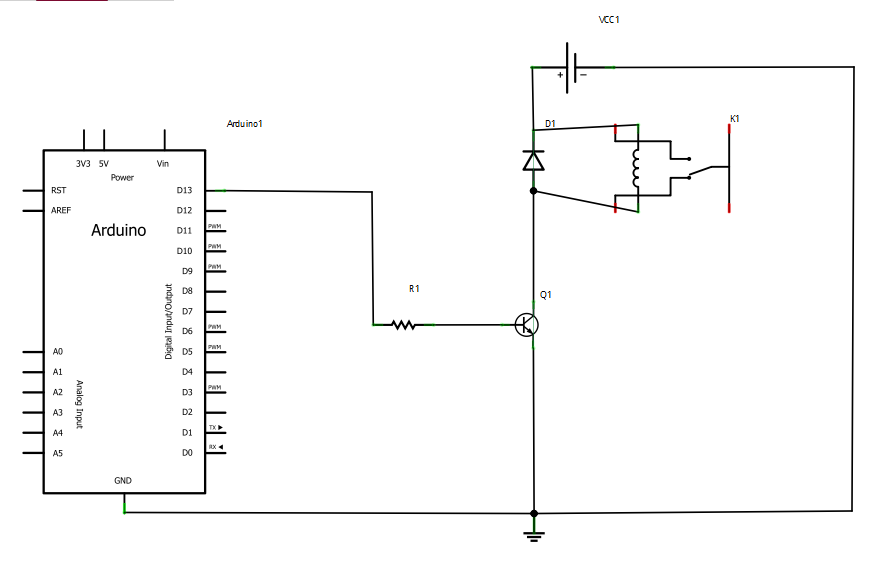
18 Una ilustración y una fotografía de Relés

En el caso del control eléctrico usaremos las salidas lógicas de Arduino (encendidas o apagadas) para la conexión del relé. También sería posible utilizar las patillas de PWM, con duty cycle al 100%.

En resumen, y antes de ver el proceso al detalle, para activar un dispositivo eléctrico del hogar (como puede ser una lámpara, ventilador, calentador…) IntelliRoom envía un comando de control de dispositivos a Arduino que sería interpretado por él. Arduino activara la salida que corresponda dependiendo del comando enviado. Esa salida colocara un transistor en modo polarización directa dando paso a la tensión del transformador activando el relé y dejando este, a su vez, activada la toma de corriente que esté controlada por ese relé.

En la siguiente ilustración se ve como hemos montado el circuito para que desde la salida de Arduino hasta la conexión del relé funcionando correctamente.

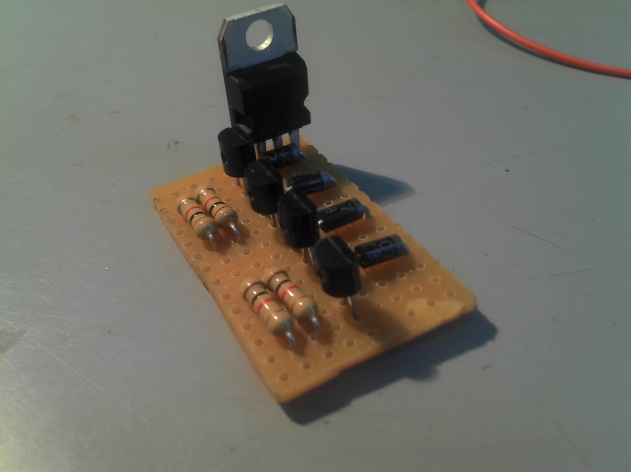
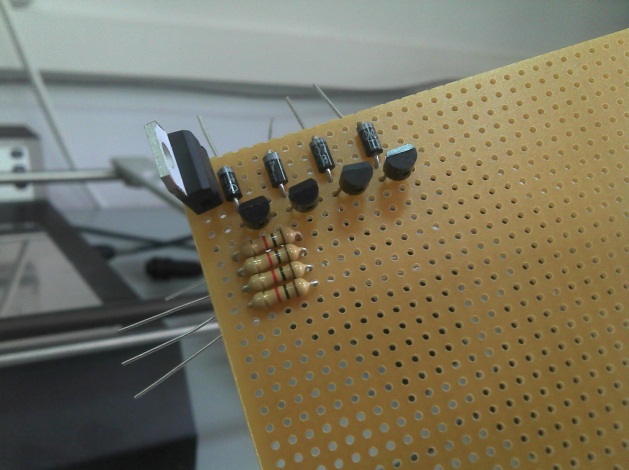
[PONER ILUSTRACION]



Como se puede apreciar, hemos añadido un diodo antes de entrar en cada uno de los relés para que, tras la desconexión no se produzcan inductancias que puedan dañar el diodo.

Esquemático

Fotografía final



19 Placa final para control de dispositivos

## Código Arduino

El código de Arduino fue desarrollado con el IDE de Arduino y sufrió bastantes cambios antes de llegar a su versión definitiva. Estos cambios se basan en que en un principio solo teníamos el procesamiento de coloreado de leds, que, en ocasiones, ocupaba la capacidad de proceso de Arduino haciendo cálculos de los degradados de colores. Y posteriormente, con el agregado del control de dispositivos en el mismo código, me vi obligado a hacer cambios en su estructura.

No voy a pasar por toda la evolución del código, si es de interés puede ser consultada en la [forja](http://code.google.com/p/intelli-room/source/browse/#svn%2Ftrunk%2FCodigo%20Arduino%2FCodigo_Arduino_1).

A continuación veremos un pseudocódigo acompañado de una explicación y finalmente veremos partes del código final.

### Pseudocódigo

Iniciamos las variables

Iniciamos la conexión con puerto serie

Configuramos la Entrada/Salida

mientras

si (HayUnMensajeDisponible) -> ConfiguraMensaje

si no Haz otros tratamientos si están disponibles

fmientras

configuraMensaje()

si existe el mensaje

configuramosArduinoParaNuevoMensaje()

El código en resumidas cuentas, obviando las variables y configuración del puerto serie, lo que hace es continuamente estar preguntando por ¿hay un nuevo mensaje?:

* Si no lo hay, actualiza las salidas del Arduino si está activado el modo “aleatorio” o en “degradado”.
* Si hay un nuevo mensaje, comprueba que el mensaje que ha llegado es interpretable por uno de los mensajes que tiene a disposición, y de ser así configura Arduino para reaccionar a partir de ese momento con la configuración establecida.

### Código

Para el entendimiento completo del código se necesitara tener la [librería *Messenger*](http://www.arduino.cc/playground/Code/Messenger) y el código en su [versión completa](http://code.google.com/p/intelli-room/source/browse/trunk/Codigo%20Arduino/Codigo_Arduino_6/Codigo_Arduino_6.pde). Ya que solo vamos a limitarnos a partes generales del código, suprimiendo cosas complejas.

Definimos los pines de entrada y salida para la conexión de la lámpara y del controlador de dispositivos

//Configuración de leds

#define PINLEDR 9 //Red LED

#define PINLEDG 10 //Green LED

#define PINLEDB 11 //Blue LED

//Configuramos los dispositivos

#define arrayLength 10 //Numero de dispositivos

uint8\_t devices[] = {4, 5, 6, 7}; //Salidas de cada uno de los dispositivos

Iniciamos los valores con el *setup* de Android.

void setup()

{

Serial.begin(9600);//configuro el puerto serie a 9600 baudios

message.attach(messageReady);//pongo la funcion callback de message

//iniciamos los pines digitales

for(int i = 0; i < arrayLength; i++)

pinMode(devices[i], OUTPUT);

}

El *loop* del programa principal

void loop()

{

while(Serial.available()){

message.process(Serial.read());

}

timeNow = millis();

if(timeEnd>timeNow) //si tiempo actual es menor que tiempo final entonces encontramos en una situación de degradado

{

UpdateValues(); //Calculamos las componentes

SetColor(rNow,gNow,bNow); //la imprimimos en los LEDs

}

else

{

if(timeRandom!=0)

{

timeInit = millis();

timeEnd = timeInit + timeRandom;

ConfigRandomColor();

}

else // llegados aquí es que todo proceso ha terminado, así que asignamos al valor final.

{

SetColor(rEnd,gEnd,bEnd);

}

}

}

Función que asigna los valores a los LEDs

void SetColor(int r, int g, int b)

{

analogWrite(PINLEDR, r); // PWM asignado al valor r

analogWrite(PINLEDG, g); // PWM asignado al valor g

analogWrite(PINLEDB, b); // PWM asignado al valor b

}

Función simple, que dado un conjunto de 3 valores por puerto serie, estos se pinten por los leds de manera inmediata.

//Modo directo

if (message.checkString("DIRECT"))

{

rEnd = message.readInt();

gEnd = message.readInt();

bEnd = message.readInt();

timeEnd = millis();

timeRandom = 0;

SetColor(rEnd,gEnd,bEnd);

}

Función simple de encendido de dispositivos

//Encender dispositivo

else if (message.checkString("SWITCHON"))

{

int device = message.readInt();

if(device >= 0 && device < arrayLength)

digitalWrite(devices[device], HIGH); //activa la señal del device

}

## Circuito completo

Para hacer los 3 circuitos necesarios (tensión, control de dispositivos y control de luz) usé los siguientes componentes:

* 7 Transistores NPN
* X Resistencias
* 4 Diodos
* 1 Rectificador
* 2 Condensadores
* 4 Relés
* …

Esquemático final

Explicación extendida (ayuda Minolo)

Fotografías final

# Capa de presentación

Nuestra capa de presentación cumple con las características definidas según el desarrollo de software en n capas, es decir: presenta el sistema al usuario, le comunica, captura la información en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato), debería ser "amigable" (entendible y fácil de usar).

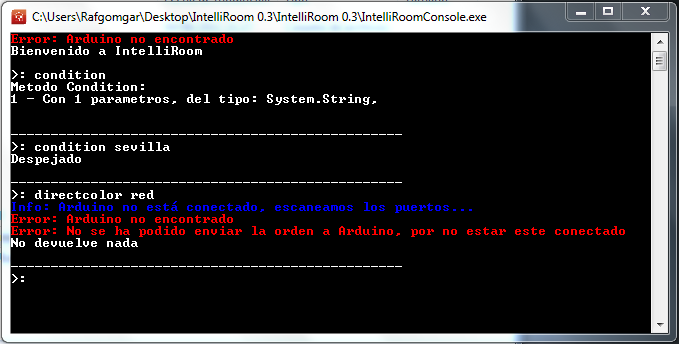
Nuestra capa de presentación tiene dos implementaciones: *Console* y *GUI* y ambas tienen la misma estructura interna. Las dos se componen de la interfaz propiamente dicha (una consola o un formulario en Windows Forms) y un intérprete de comandos sencillo. Este intérprete interactúa con una clase, que veremos más adelante, llamada *Refletion*, que ejecuta métodos y extrae información.

## Requisitos de la capa de presentación

* [TODO]

## Consola

Es un terminal de consola que interpreta comandos y muestra resultados de ejecución de esos comandos e información adicional. Veremos más adelante que tipos de comandos pueden ser ejecutados y como se ejecutan.



20 Captura de pantalla de la versión de consola

Sus funciones son:

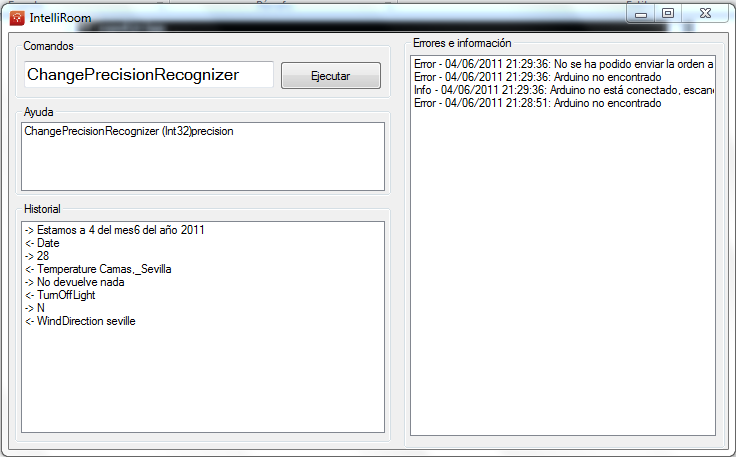
* *SearchCommand nombre\_comando*: busca parentescos de comandos según el “nombre\_comando” escrito
* *Help*: devuelve información de cómo usar en general el programa
* *?nombre\_comando*: devuelve su contrato y sus posibles sobrecargas.
* *AllCommands*: devuelve la lista con todos los comandos del sistema.
* Si no escribes bien un comando te devuelve información de cómo escribirlos o de sus posibles comandos cercanos a él.

## GUI

Una implementación mejorada de la consola de comandos, programada con la librería Windows Forms. En funcionalidad es igual que la consola, pero la información está más organizada y ayuda al usuario con ciertas características no presentes en la versión de consola:

* No permite ejecutar comandos si no se encuentra disponible en el sistema
* Devuelve en tiempo real una lista con los comandos que pueden entrar en el patrón escrito hasta el momento.

Veamos una fotografía



21 Captura de pantalla de la versión con formulario windows forms

Las motivaciones que me llevaron a crear esta interfaz fueron creadas en el proceso de realización de pruebas. Me parecía un poco complejo utilizar la consola y sobre todo lento. Lento puesto que el sistema empezaba a tener muchas funciones que, debidos a continuos cambios en el código habían cambiado su nombre repetidas veces y tenía que ejecutar muchas veces el comando *searchcommand* en la interfaz de consola. Por ello creé un sistema de ayuda simple pero efectivo que acercará el programa a más personas.

[Motivaciones para el cambio a WF]

[Cambios respecto de la consola]

[Fotografias]

[Tutorial con partes y fotos]

# Lógica de negocio: IntelliRoom

[Introducción explicando la lógica de negocio]

[Explicación de la estructura]

[El porqué de la división en módulos]

En un principio todo estaba junto para para mejorar cohesión y reducir acoplamiento en el sistema

Futuro sistema de carga de módulos dinámicos para añadir funcionalidad.

## Lista de funciones

A continuación vamos a listar todas las funciones que tiene el sistema. Todas estas funciones están en la fachada “Command” que hace las llamadas oportunas a cada uno de los módulos.

Las funciones las definiremos como:

En el paquete de **comunicación hombre-máquina** tenemos las siguientes funciones:

* Speak (string)speakText: Sintetiza el texto pasado por parámetros.
* AddGrammar (Grammar)grammar: Añade al reconocedor de voz la gramática pasada por parámetros.
* LoadGrammar: Carga la gramática por defecto, dependiendo del idioma donde se encuentre. Por defecto el idioma es español, por lo que leerá la gramática del archivo “Grammar/es.xml”.
* DeleteGrammar: Elimina toda la gramática del reconocedor de voz.
* AddGrammarList (List<string>)list (string)context: Añade a la gramática los elementos de la lista *list* bajo el contexto *context.*
* LoadGrammarList (List<string>)list (string)context: Carga en gramática únicamente los elementos incluidos en la lista *list* bajo el contexto *context.*
* DictationMode: Activa el sistema de dictado. La ejecución de cualquier otro método relacionado con carga o borrado de gramática desactiva el modo dictado.
* ChangePrecisionRecognizer (int)precision: Cambia la precisión del reconocedor de voz, por defecto el reconocimiento está al 70% de precisión.

Funciones de **iluminación**:

* DirectColor (string)colorName: Pinta el color en la lámpara, pasándole como argumento el nombre del color.
* DirectColor (byte)red (byte)green (byte)blue: Pinta el color en la lámpara, pasándole como argumento la intensidad de color de cada una de las componentes primarias de color.
* DirectColor (Color)color: Pinta el color en la lámpara, pasándole como argumento un objeto de tipo Color.
* GradientColor (string)colorName (int)timeMillis: Pinta con una degradación de duración *timeMillis* milisegundos al color con nombre *colorName.*
* GradientColor (byte)red (byte)green (byte)blue (int)timeMillis: Pinta con una degradación el color pasado en codificación RGB
* GradientColor (Color)color (int)timeMillis: Pinta con una degradación el objeto de tipo Color pasado por parámetros.
* TurnOffLight: Apaga la lámpara.
* TurnOnLight: Enciende la lámpara en color blanco.
* RandomColor (int)timeMillis: Hace degradaciones de colores aleatorios en el tiempo timeMillis milisegundos
* DesactiveRandomColor: desactiva la función de degradación.
* RandomColor (bool) active (int)timeMillis: gestiona la función de colores aleatorios.

Funciones de **dispositivos eléctricos**:

* SwitchOnDevice (int)device: Activa el dispositivo conectado al enchufe número *device.*
* SwitchOffDevice (int)device: Desactiva el dispositivo conectado al enchufe número *device.*

Funciones de **reproductor multimedia**:

* Play: reproduce la canción pausada del reproductor.
* Pause: pausa el reproductor.
* Stop: Para la canción en reproducción.
* Forward: Continúa reproduciendo la siguiente canción.
* Rewind: Reproduce la canción anterior.
* DecreaseVolume: Decrementa el volumen en un 20%
* IncreaseVolume: Aumenta el volumen en un 20%
* GetVolume: devuelve un valor entero de 0 a 100 con el porcentaje de volumen total.
* MinimumVolume: Pone el volumen al 1%.
* MaximumVolume: Pone el volumen al 100%.
* ChangeVolume (int)volumen: Asigna el volumen pasado por parámetros
* Mute: Deja el volumen al 0%.
* LoadAllMedia: Carga todos los archivos disponibles en la medioteca de WindowsMediaPlayer, en el orden en el que lo haría el mismo reproductor.
* LoadAlbum (string)nameAlbum: Carga un Álbum, si existe, correspondiente con el nombre nameAlbum
* LoadAuthor (string)nameAuthor: Carga las canciones del autor con nombre *nameAuthor.*
* LoadGenre (string)nameGenre: Carga las canciones cuyo genero corresponda con nameGenre.
* LoadTitle (string)nameSong: Carga las canciones que tengan como título nameSong.
* LoadMediaUrl (string)url: Carga el archivo o una lista compatible de la url determinada.
* InfoAuthor: Devuelve nombre del autor de la canción actual. Utilizando para ellos la información contenida en su ID3.
* InfoAlbum: Devuelve el título del álbum de la canción actual. Utilizando para ellos la información contenida en su ID3.
* InfoDuration: Devuelve la duración de la canción.
* InfoTitle: Devuelve el título de la canción actual. Utilizando para ellos la informacion contenida en su ID3.
* InfoPlayList: Devuelve la lista de títulos, géneros, álbumes y autores de la playList actual en reproducción.
* InfoMedia: Devuelve una lista con toda la información de la biblioteca de reproducción.
* GetAllAuthors: Devuelve una lista con todos los autores de la biblioteca de reproducción.
* GetAllSongs: Devuelve la lista con todos los títulos de canciones de la biblioteca de reproducción.
* GetAllGenres: Devuelve todos los géneros de la biblioteca de reproducción.
* GetAllAlbums: Devuelve todos los títulos de los álbumes de la biblioteca de reproducción.

Las funciones de **tratamiento de imagen**:

* StartProcessImage: Activa el procesamiento de imagen con la configuración establecida
* StopProcessImage: Desactiva el procesamiento de imagen.
* GetRoomIluminance: Devuelve, en porcentaje, el grado de iluminación de la habitación.
* FaceDetect: Devuelve un *FaceResult* con la información de las caras de una fotografía tomada en el momento de la ejecución del comando.
* NumberFacesDetect: Devuelve el número de caras encontradas en la captura relacada en el momento de la ejecución del comando.
* GetLastResults: Devuelve, en caso de estar en ejecución el motor de tratamiento de imagen, el resultado del último análisis.
* ConfigCameraSetXX: Configura el XX al valor pasado por parámetros. XX son muchas variables, explicadas todas en el capítulo dedicado a ello.
* ConfigCameraGetXX: Devuelve la configuración de XX. XX puede ser alguna de las variables del sistema Camera.
* GetConfigCamera: Devuelve un resumen de la configuración de todas las variables del sistema Camera.

La función del **programador** es la siguiente, con 3 variantes:

* AddTask (string)command (DateTime) date: Ejecuta el comando *command* en la fecha indicada.
* AddTask (string)command (int)day (int)month (int)year (int)hour (int)minute: ejecuta el comando *command* en el día mes año hora y minuto indicado.
* AddTask (string)command (int)hour (int)minute: Ejecuta el comando en la hora y minutos especificados del mismo día.

La funcion del **gestor de acciones** del sistema.

* AddAction (string)nameEvent (string)command: Añade la ejecución del comando command cuando el evento nameEvent se invoque en el sistema.

Funciones de **información climatológica**:

* Temperature (string)city: Devuelve la temperatura en grados centígrados de la ciudad *city*.
* TemperatureFahrenheit (string)city: Devuelve la temperatura en grados Fahrenheit de la ciudad *city*.
* Condition (string)city: Devuelve la condicion de una ciudad determinada en el momento actual. La condición puede ser: despejado, lluvioso, nuboso, tormenta…
* Humidity (string)city: Devuelve la cantidad de humedad, en porcentaje, de la ciudad *city*.
* WindDirection (string)city: Devuelve la dirección del viento actual.
* WindSpeed (string)city: Devuelve la velocidad del viento actual.
* ChangeRangeInTemperatureEvent (int)max (int)min: Indica el rango máximo y mínimo de temperatura para que sean invocados los eventos.

Otras funciones:

* Exit: Sale de IntelliRoom.
* GetMessages: Devuelve los mensajes del sistema hasta el momento.
* Date: Devuelve la fecha actual en texto.
* Time: Devuelve la hora actual en texto.
* DateAndTime: Devuelve un objeto *DateTime* con la fecha y hora actual.

Además de estas funciones, incluidas en la clase *IntelliRoom.Command*, tenemos otra clase *IntelliRoom.SpeakCommand*, que contiene el prototipo para incluir funciones por voz que requieran la entrada de parámetros decidibles por el usuario en ese mismo momento. Veamos un ejemplo para ver más claro esto.

Si por ejemplo tuviéramos un comando en gramática que fuera “color de la lámpara a rojo” es evidente que esta frase estará ligada a un comando del sistema que va a pintar en el control de iluminación el color rojo. Su llamada podría ser la siguiente “directColor red” o “gradientColor red 1000” entre otras. Pero si queremos un comando de voz que fuera por ejemplo “color de lámpara” y el sistema diera un tiempo para que el usuario diera el parámetro de entrada necesario, en este caso un color, estaríamos ante un ejemplo de un comando de voz que requiere parámetros.

Este mismo ejemplo podemos llevarlo a la carga de música “cargar música” seguido del nombre del artista por ejemplo. “grados centígrados” más el nombre de la ciudad. Podríamos continuar hasta que nuestra imaginación se agotase.

De esta manera el intérprete de comandos de voz permite la carga, de todos los comandos anteriores y de los incluidos en la clase *IntelliRoom.SpeakCommand*.

## Sistema de eventos

El sistema de eventos y el programador, el cual veremos en el siguiente apartado, es lo que proporciona al proyecto de cierto interés.

Hasta el momento IntelliRoom es un proyecto de domótica que tiene una conjunto de funciones ejecutables y esas funciones interactúan con el proyecto en forma de acción (activa el sistema X, reproduce Y canción) o de simple información (cuanta iluminación tengo en la habitación, que temperatura hace en nuestra ciudad).

Lo que vamos a hacer es enriquecer el sistema y transformar una condición (una información) en una acción u otra información. Veamos algunos ejemplos del sistema de eventos:

* Cuando detectes movimiento en la habitación guarda una captura de la cámara.
* Cuando la temperatura sea superior a 24 grados, enciende un ventilador.
* Si la habitación tiene una iluminación menor al 30% enciende una luz.

Cada paquete de IntelliRoom tiene sus propios eventos y todos los eventos de todos los paquetes se encuentran centralizados en la clase *Events.* Para el proyecto he creado un prototipo funcional con los eventos del sistema implementados hasta el momento. Estos son los siguientes:

* finishImageProcess: Invocado cuando el procesamiento de imagen termina, incluye además el resultado complete de su análisis.
* iluminanceEvent: Evento configurable que devuelve la iluminación de la habitación a partir de un valor. Además devuelve la iluminación de la misma.
* movementDetected: Invocado cuando se ha detectado movimiento en la habitación.
* peopleDetected: Invocado cuando se ha detectado la presencia de una persona.
* newMessage: Invocado cuando hay un nuevo mensaje interno en el sistema, incluye el mensaje.
* speechRecognizer: Invocado cuando el reconocedor de voz reconoce algo en gramática.
* temperatureMaxEvent: Evento configurable que es invocado cuando la temperatura es superior a la configurada, incluye la temperatura actual en centígrados.
* temperatureMinEvent: Evento configurable que es invocado cuando la temperatura es inferior a la configurada, incluye la temperatura actual en centígrados.

El sistema no sustituye una acción relacionada con un evento ya configurado, sino que lo sobrecarga. Esto quiere decir que si tenemos un proceso para un evento este también se ejecutara en cadena con el nuevo insertado. [Encadenamiento|de|comandos]

## Programador

El programador, a diferencia del sistema de eventos, gestiona tareas. Una tarea es la ejecución de una o varias funciones determinadas en un momento determinado.

Por ejemplo, si conociéramos la hora a la que volvemos a casa después de un frio día de invierno podríamos decirle al sistema que encienda un calefactor 30 minutos antes de nuestra llegada. O si queremos mantener una bombilla encendida de diez de la noche a dos de la madrugada…

Para hacer esto, he creado el concepto de tarea, que no es más que la relación de una función o varias de ellas en un momento determinado.

Así, de manera sencilla, podemos dotar al sistema de funciones mucho más enfocadas a la domótica. [LO DEJAMOS AQUI]

# Módulo Arduino

El módulo Arduino es el encargado de controlar todo lo referente a la comunicación con Arduino, en dos aspectos:

* Configuración de la comunicación.
* Sistema de mensajería para el envío de comandos.

## Requisitos funcionales

Los objetivos que cumple este módulo son:

* Sistema de comunicación por puerto serie optimizado para la comunicación con Arduino. Nos abstrae de detalles de configuración como pueden ser: COM al que va dirigido, cantidad de baudios por segundo, tratamiento de fallos en envío de mensajes o manera en la que se envían los datos.
* Modelar el conjunto de funciones que hemos implementado en Arduino para que puedan ser tratadas desde IntelliRoom con funciones básicas.
* Búsqueda y detección de Arduino automática.

## Estructuración “Arduino”

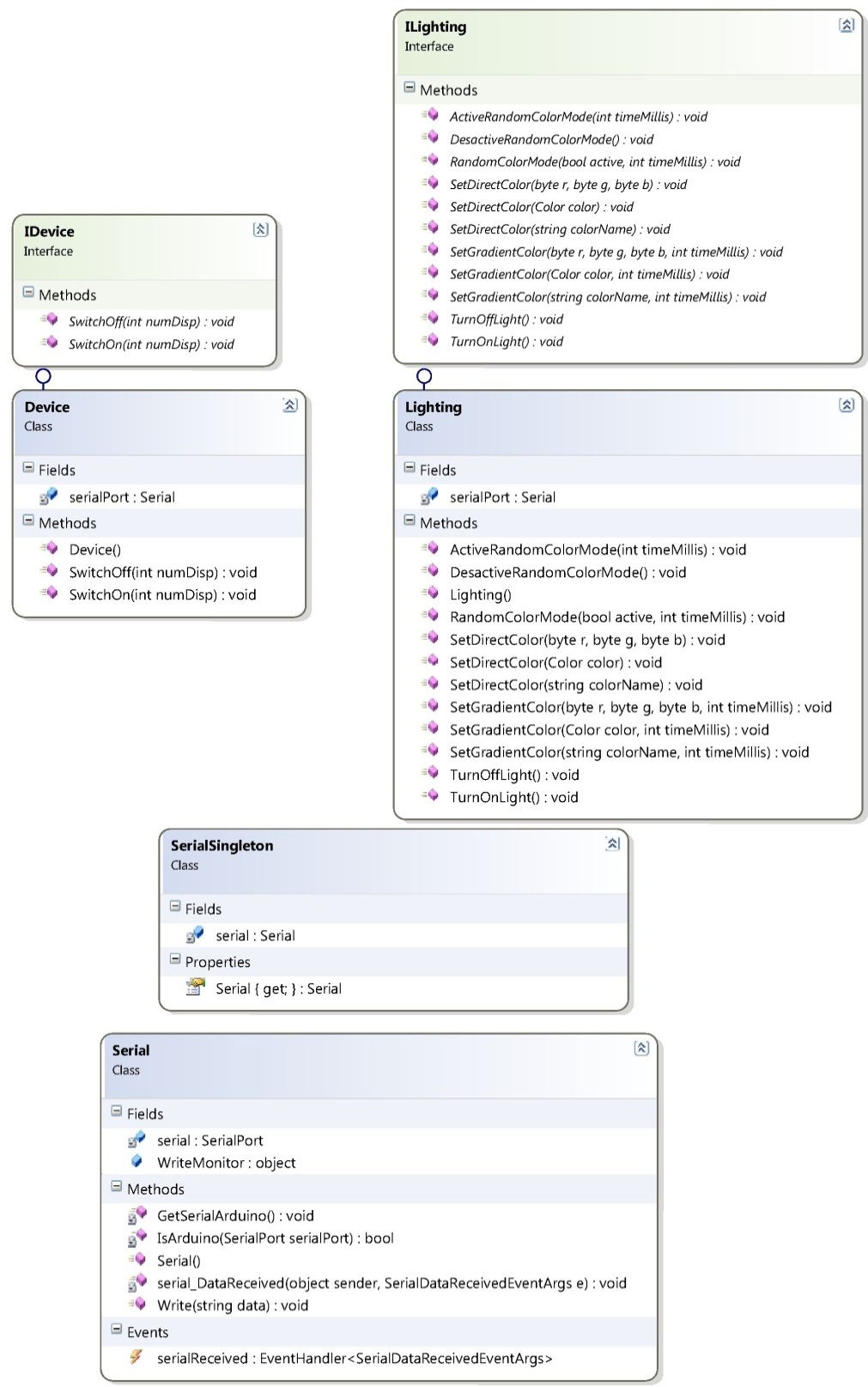
Como podemos ver en la [ilustración que tenemos un poco más abajo], el módulo Arduino se conforma de 4 clases y 2 interfaces:

Las dos interfaces, *ILigting* y *IDevices*, solo especifican el contrato de funciones que ejecutara IntelliRoom. En nuestro caso tenemos 2 apartados a tratar: el control de iluminación y el control de dispositivos.

De esas dos interfaces, tenemos sus dos implementaciones: *Device* y *Ligting* que implementa los mensajes que son reconocidos por Arduino, estas dos clases contienen una instancia de la clase *Serial* que es la encargada de las comunicaciones y su finalidad es:

* Localizar todos los puertos COM
* En caso de encontrar puertos COM, detectar si alguno es Arduino.
* Envió y recepción de mensajes configurados para la plataforma.
* Control de errores de conexión.

Por último *SerialSingleton* encargada de mantener una instancia de Serial (que abre puertos COM produciendo muchos errores) devolviendo solo referencias del mismo al sistema.



22 Estructura del paquete arduino

## protocolo de mensajes

En este apartado vamos a definir de manera rápida las funciones que delega IntelliRoom a Arduino, y cómo hemos definido la comunicación de cada uno de los mensajes para cada una de sus funciones.

Cuando me refiero a “delegar” funciones, quiero decir que las funciones no son ejecutadas en IntelliRoom, sino que IntelliRoom ordena hacer un proceso y este es procesado íntegramente por el código que tiene Arduino. Un ejemplo de proceso podría ser “degradaciones de color” podríamos trabajarlas de dos maneras diferentes:

1. Dotar a Arduino de una función que dado un color enviado por puerto serie, este se pinte en los leds y, continuamente, desde el sistema enviamos una y otra vez la actualización del color.
2. Enviamos un comando más complejo a Arduino y que sea este el que procese interiormente un sistema de cambio de color por si solo.

Se pueden apreciar las ventajas de la opción dos, que permite que el canal de comunicación este desocupado y que no tengamos hilos que envíen constantemente comandos a Arduino. Nuestra manera de trabajar en el proyecto será encaminándonos a la opción 2. Por lo que el peso del desarrollo de las funciones estará en el código de Arduino.

Para el envío de información hemos usado algo cómodo pero en cierto modo ineficiente, le enviamos al Arduino por puerto serie una cadena de caracteres, por lo que por cada carácter le llega 8 bits. Arduino tiene un buffer virtual de 128 bytes, pero la librería que hemos usado para la comunicación no permite más de 64 bytes así que, a menos que enviemos cadenas mayores de 64 caracteres, no vamos a tener ningún problema.

Otra manera, más eficiente seria con envíos de bytes en funciones. Podríamos enviar 1 byte para las funciones y otros bytes para los argumentos. Pero esta opción fue descartada por que era más visual mandar cadenas de texto y tampoco iba a afectar demasiado al sistema, puesto que el envío de mensajes es muy escaso y no forma parte de una parte critica del proyecto optimizar estos envíos.

23 Traza de ejemplo de envío de mensaje

Máquina con Windows

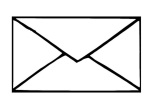
IntelliRoom

Serial

Arduino programado



Enciende dispositivo 1



Vamos a definir un comando general de la siguiente manera:

Donde función es una de las funciones que posteriormente vamos a definir y Arg1...ArgN cada uno de sus argumentos. Enviando entonces una cadena que contenga esa información, con espacios entre función y argumentos y entre argumentos y retorno de carro y de línea al final.

### Funciones de control de color

A continuación describiremos las funciones de control de color o iluminación.

**Función directa**: Cambia de color instantáneamente al color establecido por valores RGB:

Nombre función: DIRECT

R: Valor color Rojo

G: Valor color Verde

B: Valor color Blue

Estructura: DIRECT [R] [G] [B]

Ejemplo: DIRECT 123 220 1 -> R=123, G=220, B=1

**Función degradado:** Cambia de color gradualmente en un tiempo de [Time] milisegundos.

Nombre función: GRADIENT

R: Valor color Rojo

G: Valor color Verde

B: Valor color Azul

Time: Número de milisegundos que estará degradando

Estructura: GRADIENT [R] [G] [B] [Time]

Ejemplo: GRADIENT 255 0 0 10000 -> Tarda 10000 milisegundos (10 segundos) en cambiar a color rojo

**Función aleatoria:** Cambia de color gradual y aleatoriamente en un tiempo fijado de milisegundos.

Nombre función: RANDOM

TimeRandom: Tiempo, en milisegundos que hay entre el paso de un color a otro.

Estructura: RANDOM [TimeRandom]

Ejemplo: RANDOM 1000 -> Activamos la función aleatoria para que cambie de color cada 1000 milisegundos.

Observación: para desactivar el modo aleatorio introducir en el campo TimeRandom 0 milisegundos.

### Funciones de control de dispositivos

Los mensajes enviados para el control de dispositivos tienen la misma estructura de los de control de color o iluminación, a continuación se definen los dos tipos de mensajes utilizados para la gestión de dispositivos eléctricos.

**Función encender dispositivo**: Enciende un dispositivo conectado en la salida “Dispositivo”.

Nombre función: SWITCHON

Dispositivo: Número de referencia del dispositivo que quiere encenderse.

Estructura: SWITCHON [Dispositivo]

Ejemplo: SWITCHON 3 -> Enciende dispositivo 4 (los valores que se toman son de 0-9)

**Función apagar dispositivo**: Apaga un dispositivo conectado en la salida “Dispositivo”.

Nombre función: SWITCHOFF

Dispositivo: Número de referencia del dispositivo que quiere apagarse.

Estructura: SWITCHOFF [Dispositivo]

Ejemplo: SWITCHON 0 -> Apaga dispositivo 1 (los valores que se toman son de 0-9)

### Configuración puerto serie

La configuración entre Arduino y el ordenador principal está configurada de la siguiente manera:

* La velocidad elegida es la estándar, 9600 baudios.
* Y la marca para el fin de maneje (el retorno de línea es): “\r\n”

## Funciones relevantes

### Creación de conexión Serial

La conexión por puerto serie se inicia con este método que es llamado desde el constructor. Este método pregunta por los puertos series activos y hace un chequeo de cada uno de ellos, comprobando si alguno es Arduino. Si lo encuentra inicia la conexión con una instancia de la clase *SerialPort* perteneciente al paquete *System.IO.Ports.*  Si no, deja a “*null”* ese objeto para hacer comprobaciones en cada envió de mensaje.

private void GetSerialArduino()

{

SerialPort arduino = null;

string[] serialPortsName = SerialPort.GetPortNames();

foreach (var PortCom in serialPortsName)

{

SerialPort serialPort = new SerialPort(PortCom, 9600) { NewLine = "\r\n" };

serialPort.ReadTimeout = 500;

bool found = false;

if (!serialPort.IsOpen)

{

//a veces detecta COM que no existen.

try

{

serialPort.Open();

found = IsArduino(serialPort);

serialPort.Close();

}

catch (Exception)

{

continue;

}

}

else //puerto está abierto

{

found = IsArduino(serialPort);

}

if (found)

{

arduino = serialPort;

break;

}

}

if (arduino != null) //hemos encontrado arduino

{

//abrimos si es posible

if (!arduino.IsOpen)

arduino.Open();

serial = arduino;

serial.DataReceived += new System.IO.Ports.SerialDataReceivedEventHandler(serial\_DataReceived);

Message.InformationMessage("Arduino encontrado en puerto " + arduino.PortName);

}

else

{

Message.ErrorMessage("Arduino no encontrado");

}

}

El método *IsArduino* devuelve un booleano si el dispositivo es Arduino, puesto que SerialPort está configurado para esperar 500ms, en caso de no recibir el mensaje devuelve falso y en caso de recibir “ACK” al enviarle el mensaje “CHECK” podemos decir que el dispositivo conectado en ese puerto COM es Arduino.

private bool IsArduino(SerialPort serialPort)

{

serialPort.WriteLine("CHECK");

try

{

return serialPort.ReadLine() == "ACK";

}

catch (Exception)

{

return false;

}

}

### Envío de un mensaje

Para el envío de un mensaje, lo primero que hacemos es entrar en un monitor para que no podamos llamar a este método más de una vez en al mismo tiempo y enviar cadenas corruptas.

Antes del envío de cada mensaje hacemos un chequeo de la disponibilidad de Arduino, en caso de no estar disponible, buscaría de nuevo al dispositivo por medio del caso de uso descrito en el apartado 7.4.1

Por último si hay un último fallo al enviar, después de localizar el dispositivo se envía un mensaje de error a la consola.

public void Write(string data)

{

Monitor.Enter(WriteMonitor);

if (serial == null)

{

Message.InformationMessage("Arduino no está conectado, escaneamos si se encuentra activo");

GetSerialArduino();

}

try

{

if (serial != null)

{

serial.WriteLine(data);

}

else

{

Message.ErrorMessage("No se ha podido enviar la orden a Arduino, por no estar esté conectado");

}

}

catch (Exception)

{

Message.ErrorMessage("Error en el envío de datos");

GetSerialArduino();

}

finally

{

Monitor.Exit(WriteMonitor);

}

}

# Módulo Voice

## Introducción

El módulo Voice es el encargado de satisfacer el objetivo de interfaz usuario-maquina por medio de la voz. Por un lado utilizaremos reconocimiento del habla para interpretar las funciones que el usuario le da al sistema y por otro, en caso de que sea necesario, un sintetizador para que el sistema informe al usuario.

Antes de continuar en materia vamos a formalizar los conceptos de reconocimiento y sintetización:

[**Reconocimiento de automático del habla (RAH)**](http://es.wikipedia.org/wiki/Reconocimiento_del_habla) es una rama de la inteligencia artificial cuya finalidad es permitir la comunicación hablada entre seres humanos y computadoras electrónicas. El problema principal de esta rama es encontrar una interpretación aceptable del mensaje partiendo del conocimiento que tenemos del habla humana (como pueden ser las áreas de acústica, fonética, fonológica, léxica, sintáctica…) en presencia de posibles errores (los cuales son inevitable como por ejemplo ruido de fondo…)

En estos sistemas tienen diversas características como pueden ser:

* La entrenabilidad: si necesitan ser entrenados previamente.
* La dependencia del hablante: si es necesario un entrenamiento personal por cada uno de los hablantes.
* Continuidad: si requiere pararse o no entre palabras.
* Robustez: determina si es vulnerable o no ante espacios ruidosos y otros factores que generen posibles errores.
* Tamaño del dominio: determina si el sistema está diseñado para dominio reducido de palabras o extenso.

[**Sintetización del habla**](http://es.wikipedia.org/wiki/Sintetizaci%C3%B3n_del_habla) o síntesis del habla es la producción artificial de habla humana sin necesidad de que necesitemos que este pregrabada. Este proceso puede llevarse a cabo a través de software o hardware. Es posible que a lo largo del documento mencione text-to-speech (TTS) para referirme a esta técnica.

La calidad de una voz sintética vendrá dada por su inteligibilidad o con qué facilidad es entendida y con la medida en que esta se asemeja a la voz real de un humano.

## Requisitos funcionales

* Reconocimiento, por medio de la voz, de estructuras gramaticales
* Sintetización de texto.
* Gestión de un sistema de gramática que permita: añadir nueva gramática por XML o listas, borrar, activar el modo de dictado libre.
* Permita el cambio de sensibilidad en detección de estructuras gramaticales.
* Sistema para evitar errores generados por la retroalimentación que se puede generar cuando el reconocimiento de voz está activo mientras se encuentra sintetizando texto.

## Estructura del módulo

En el módulo de reconocimiento y sintetización tenemos 3 clases y 1 interfaz.

La interfaz *IVoiceEngine* es el contrato que debe cumplir el modulo y contiene los métodos de los que nos vamos a abstraer gracias al resto clases.

Las clases tienen una estructura en árbol, podríamos decir que *VoiceEngine* es la clase padre de todas. Su misión es la comunicación entre las otras dos clases (*Recognizer* y *Sinthesizer*) y además la de generar, a partir de archivos XML o listas gramática entendible por el reconocedor.



24 Estructura del paquete voice

## SAPI

Una vez visto lo necesario que hay que saber a nivel conceptual de los sintetizadores y los reconocedores de voz hablaremos de la tecnología que he usado en el proyecto, más concretamente hablaremos de SAPI.

SAPI o *Speech Application Programming Interface* es una API para el desarrollo de técnicas de sintetización y reconocimiento de voz para aplicaciones desarrolladas en Windows. Históricamente podemos diferenciar etapas del producto, la que nosotros usaremos en el desarrollo es la 5.x la cual supuso un salto en la API en relación con las versiones de la 1 a la 4. Entre sus novedades una de las que más nos interesa es el soporte para la plataforma .NET desde el framework 3.0. En concreto, para el desarrollo, se usó la versión 5.4 (su salida fue producida por la llegada de Windows7) pero para el uso que le vamos a dar es completamente compatible con la versión 5.3 (Windows Vista).

SAPI se divide en 2 grandes partes:

* El paquete System.Speech.Recognition que satisface nuestro objetivo en el ámbito del reconocimiento automático del habla o speech recognizer.
* El paquete System.Speech.Synthesis que resuelve el problema de la sintetización del habla o text-to-speech.

Además de estos dos paquetes incluye uno adicional (System.Speech.AudioFormat) que omitiremos en este proyecto.

Una vez que hemos visto las características cuantitativas que posee, vamos a ver como SAPI las cumple cualitativamente desde el punto de vista de las pruebas que he estado realizando.

El reconocimiento funciona realmente bien, tiene mucho soporte para una gran cantidad de idiomas y, siguiendo el esquema de introducción del capítulo en el apartado de características del reconocimiento automático del habla, vamos a verlas una por una su comportamiento en SAPI:

* La entrenabilidad: En el caso de IntelliRoom el juego de semántica y sintaxis están muy acotados. No es un problema diferenciar entre “pausar canción” y “reproducir música”, además los paquetes de idiomas solucionan muchos problemas.
* La dependencia del hablante: SAPI va entrenándose conforme vas haciendo uso de ella (incluso si la usas desde el programa), además Windows tiene un apartado de entrenamiento que puedes acceder fácilmente desde el panel de control. Según mis pruebas es posible la independencia del hablante, pero funciona mucho mejor con entrenamiento personal.
* Continuidad: La continuidad es muy buena, incluye funcionalidad de dictado para redactar tus escritos a tiempo real (usable en la interfaz de Windows Vista/7)
* Robustez: En este aspecto he conseguido trasmitirle comandos de voz con música en reproducción bastante alta e incluso con difícil acceso al micrófono, dando resultados bastante buenos, mejores si lo tienes entrenado.
* Tamaño del dominio: en este aspecto no es necesario que entremos demasiado, no vamos a tener que elegir si una muestra x se asemeja más a y, siendo y un elemento dentro de un conjunto de miles de posibilidades. Los comandos de voz estarán acotados y no hay previsión que supere los 300. En el caso de que sí, también funciona con bastante acierto.

Con relación a la sintetización, Microsoft no tiene muchos agentes (que es como son llamados los paquetes de sintetización para los diferentes idiomas) y el español, por ejemplo, no está incluido entre ellos. Una posible solución a este problema podría ser utilizar un agente de Loquendo (que son compatibles con SAPI) pero no son gratuitos o utilizar Google que desde hace poco tiempo incluye una gran variedad de idiomas.

Dejo por si es de interés incluyo un ejemplo de sintetización del texto “hola mundo” en español:

<http://translate.google.com/translate_tts?ie=UTF-8&q=hola%20mundo&tl=es&prev=input>

Se puede ver claramente que **q=** contiene el texto a sintetizar y **tl=** el idioma a ser sintetizado. Además, si pincháis en el enlace, podéis comprobar que es un TTS con bastante buena calidad.

Aun con este problema usaremos SAPI que se encuentra muy bien documentada y tiene mucha potencia gramatical y, si es necesario, utilizar en un futuro un gestor de sintetización internacional basado Google.

## Configuración de SAPI

Vamos a ver como configure SAPI para utilizarlo en IntelliRoom.

En el proyecto hay 3 clases principales:

* Recognizer: Encargada de encapsular y gestionar la configuración de un objeto de tipo SpeechRecognitionEngine perteneciente al paquete, anteriormente comentado, System.Speech.Recognition.
* Synthesizer: Gestiona la sintetización, utilizando el objeto SpeechSynthesizer.
* VoiceEngine: Aglutina dos objetos, uno de la clase Recognizer y otro de Synthesizer y los sincroniza, además de ello incluye el sistema de carga de gramática por medio de XML descrito en el apartado **¡Error! No se encuentra el origen de la referencia.**

### Reconocedor del habla

Para ver la configuración del objeto SpeechRecognitionEngine sería interesante copiar ciertas partes del código y verlo en detalle.

using System;

using System.Speech.Recognition;

namespace Voice

{

class Recognizer

{

private SpeechRecognitionEngine speechRecognition;

private bool isAvailable;

private bool dictationMode;

private int precision;

public event EventHandler<SpeechRecognizedEventArgs> speechRecognized;

internal Recognizer()

{

speechRecognition = new SpeechRecognitionEngine();

DictationMode();

this.precision = 70;

speechRecognition.SetInputToDefaultAudioDevice();

isAvailable = false;

speechRecognition.RecognizeAsync(RecognizeMode.Multiple);

ActiveRecognizer();

}

Lo importante se puede ver en el constructor, podemos ver que por defecto inicia el modo de dictado, (para incluir todas las palabras del idioma que tengamos instalado en el PC. Por defecto hay muchísimas. Teóricamente, todas las necesarias para poder reconocer una conversación o un texto.

Se especifica nuestra línea de entrada como la estándar por defecto del PC, (en caso de que el sistema no reconozca comandos de voz, el problema puede verse solucionado configurando en panel de control la línea de entrada por defecto con la línea que queramos usar). También sería posible aplicar algún algoritmo de eliminación de ruido antes de que se procese por la entrada del reconocedor o enviar los datos de voz por otros canales como puede ser a través de un servicio web destinado a esta finalidad.

Se configura para que reconozca de manera asíncrona, de esta manera reconocerá independientemente de la tarea que se esté realizando, permitiéndonos, una escucha continua en el tiempo.

Para el tipo de reconocimiento asíncrono, usaremos el enumerado *RecognizeMode* con valor *Multiple.* Esto permite a la aplicación lanzar una instancia independiente a la usada por el SO, si esta es usada por el usuario, y poder tener un intérprete propio dentro de nuestro programa. Permitiéndonos diferenciar la configuración del reconocedor en IntelliRoom con la que un usuario pueda tener en Windows7 pudiéndose ejecutar ambas al mismo tiempo.

También es necesario suscribirse al evento *speechRecognized* que se enviará cada vez que reconozca un patrón definido por la entrada de audio configurada.

internal void AddGrammar(Grammar grammar)

{

if (dictationMode)

{

DeleteAllGrammars();

dictationMode = false;

}

speechRecognition.LoadGrammar(grammar);

}

internal void DictationMode()

{

DeleteAllGrammars();

speechRecognition.LoadGrammar(new DictationGrammar());

dictationMode = true;

}

internal void DeleteAllGrammars()

{

dictationMode = false;

speechRecognition.UnloadAllGrammars();

}

Los 3 métodos anteriores para gestionar gramática y el modo dictado.

internal void InactiveRecognizer()

{

if (isAvailable)

{

isAvailable = false;

speechRecognition.SpeechRecognized -= new EventHandler<SpeechRecognizedEventArgs>(speechRecognition\_SpeechRecognized);

}

}

internal void ActiveRecognizer()

{

if (!isAvailable)

{

isAvailable = true;

speechRecognition.SpeechRecognized += new EventHandler<SpeechRecognizedEventArgs>(speechRecognition\_SpeechRecognized);

}

}

void speechRecognition\_SpeechRecognized(object sender, SpeechRecognizedEventArgs e)

{

if (e.Result.Confidence \* 100 >= precision)

{

speechRecognized(sender, e);

}

}

Y los dos primeros métodos que gestionan la suscripción y [[desuscripcion]] del evento de reconocimiento del sistema y el último método es el tratamiento que se le da al evento cada vez que es enviado por SAPI. Este proceso *speechRecognition\_SpeechRecognized*, cada vez que envía el evento de reconocimiento comprueba si la confianza de esa muestra en relación con las posibilidades gramaticales que tenemos formadas supera una precisión establecida por nosotros, por defecto el 70% de exactitud, si lo supera envía el evento a la clase *SpeechInterpreter* que se encarga de lanzar comandos por voz.

### Sintetizador de voz

La configuración de la sintetización de voz es bastante más sencilla que la configuración de reconocimiento. Procedamos a su explicación.

using System;

using System.Collections.Generic;

using System.Speech.Synthesis;

namespace Voice

{

class Synthesizer

{

private SpeechSynthesizer speechSynthesizer;

private Boolean inUse;

private Queue<String> queueSpeech;

internal event EventHandler<SpeakCompletedEventArgs> finishSpeechEvent;

public Synthesizer()

{

speechSynthesizer = new SpeechSynthesizer();

inUse = false;

queueSpeech = new Queue<string>();

speechSynthesizer.SpeakCompleted += new EventHandler<SpeakCompletedEventArgs>(finishSpeaking);

}

Importamos el paquete *System.Speech.Synthesis* necesario.El constructor de la clase inicializa el objeto de tipo SpeechSysthesizer y nos suscribimos a un evento que salta cada vez que ha terminado de reproducir una sintetización. Esto último lo utilizaremos para gestionar colas de texto para el caso de que sintetizar texto en el momento de que el sintetizador está ocupado en otra sintetización.

internal Boolean SpeakText(String text)

{

Boolean res = false;

if(!InUse)

{

InUse = true;

speechSynthesizer.SpeakAsync(text);

res = true;

}

return res;

}

internal void SpeakTextQueue(String text)

{

if (InUse)

{

queueSpeech.Enqueue(text);

}

else

{

InUse = true;

speechSynthesizer.SpeakAsync(text);

}

}

Dos métodos para escribir en el sintetizador.

void finishSpeaking(object sender, SpeakCompletedEventArgs e)

{

InUse=false;

if (!EmptyQueue())

{

SpeakNextTextInQueue();

}

else

{

finishSpeechEvent(sender, e);

}

}

Cada vez que salta el evento “finishSpeaking” comprobamos si hay más mensajes en cola, si los hay continuamos con su reproducción, si no lanzamos un evento a la próxima clase para que active el reconocedor de nuevo (ya que cuando sintetizamos el reconocedor se para).

## Gramáticas en SAPI

Ya sabemos cómo configurar SAPI para que reconozca y sintetice, ahora vamos a ver cómo funciona SAPI para introducir tus propias gramáticas.

SAPI tiene 3 clases principales para la gestión de gramática: Choice, GrammarBuilder y Grammar.

El objetivo es generar un *Grammar* que es lo que se le añade al objeto *SpeechRecognizerEngine*. Para generar un *Grammar* necesitamos al menos un objeto de tipo *GrammarBuilder* y para un objeto *GrammarBuilder* al menos uno de tipo *Choice*. Por lo que podemos definirlos de la siguiente manera.

25 Estructura de un objeto Grammar

* *Grammar*: Es un conjunto de gramáticas (*GrammarBuilder*). Un objeto de este tipo incluye un atributo *Name* que engloba. El concepto o *Name* es un atributo que aparecerá cuando salte el evento en el reconocedor (junto con la información, la frase literal reconocida, además de información fonética, conjuntos de palabras, etc.).
* *GrammarBuilder*: Es un conjunto de opciones (*Choices*) que guardan un sentido común. Además un objeto de tipo *GrammarBuilder* tiene un *Choices*.
* *Choices*: Es un conjunto de elecciones.

Veamos un ejemplo sencillo para entender estos conceptos, sobre cómo crear una orden de encender una luz:

Para encender una luz podríamos decir bastantes comandos, por ejemplo: Encender la luz, encender la bombilla, encender la iluminación, encender una lámpara… o por ejemplo en vez de utilizar el verbo “encender” podríamos haber usado el verbo “activar”. De esta manera vemos que para encender una luz podríamos tener este posible árbol de posibilidades:

26 Ejemplo de estructura gramatical

De esta forma podríamos crear dos conjuntos de elecciones (Choices) “los verbos” y “lo que queremos encender”, siendo su código.

Choices verbs = new Choices();

verbs.Add("encender");

verbs.Add("activar");

Choices complements = new Choices();

complements.Add("la iluminación");

complements.Add("la luz");

complements.Add("la lámpara");

complements.Add("la bombilla");

GrammarBuilder grammarBuilder = new GrammarBuilder();

grammarBuilder.Append(verbs);

grammarBuilder.Append(complements);

Grammar grammar = new Grammar(grammarBuilder);

grammar.Name = "encendido de luz";

Como podemos ver en este ejemplo SAPI es realmente interesante y con relativamente poco código podemos generar e introducir gramática en el sistema.

## Como generar una nueva gramática en IntelliRoom

Ya hemos visto cómo generar gramática para SAPI ahora veremos cómo crear gramática para IntelliRoom y que esta pueda ser traducida por comandos interpretables por el sistema.

En el directorio *Grammar* de IntelliRoom podemos encontrar varios XML (tantos como distintas gramáticas para distintos idiomas tengamos) en ellos podemos encontrar por ejemplo:

<command name="pause">

<choice>

<element>pausar</element>

</choice>

<choice>

<element>la</element>

<element></element>

</choice>

<choice>

<element>canción</element>

<element>reproducción</element>

<element>música</element>

<element></element>

</choice>

</command>

Este XML engloba una gramática para definir el comando “pause”, si seguimos el diagrama podemos ver algunos ejemplos de frase:

27 Estructura gramatical del ejemplo anterior

Decir al micrófono (con esta gramática cargada): Pausar la canción, Pausar música o Pausar la reproducción llevarían a la ejecución del comando “pause” dentro del sistema.

Ahora pues… ¿Cómo generamos gramática para nosotros? Pues visto este ejemplo anterior se puede deducir claramente que:

<command name=" nombre\_del\_comando1|nombre\_del\_comando2|… ">

//elección 1

<choice>

<element> elemento 1 </element>

<element> elemento 2 </element>

<element> … </element>

<element> elemento n </element>

</choice>

//eleccion 2

<choice>

<element></element>

<element></element>

</choice>

//…

<choice>

<element></element>

<element></element>

</choice>

//eleccion n

<choice>

<element></element>

<element></element>

</choice>

</command>

También es posible la ejecución de varios comandos consecutivos con un solo comando de voz, si dividimos los comandos entre sí por el carácter “|”.

Ejemplo: “pause | loadAuthor Chopin | play”

Con esto hemos creado un sistema rápido de generación gramáticas, más cercano al usuario y sin necesidad de que este *hardcodeado* en el código.

## Casos de uso relevantes

### Añadir gramática por documento XML

El método de cargar gramática por medio de XML es el más complejo, en cuanto a carga de gramática se respecta, por lo que es uno de los más interesantes a mostrar.

Básicamente se trata de 2 bucles for, un bucle for para el método *LoadCommand* y otro por el *LoadChoices*.

El método *LoadGrammar* carga el documento XML (en caso de no hacerlo genera un mensaje de error) y una vez creado, la añade al reconocedor.

public void LoadGrammar()

{

//cargamos el documento XML

XmlDocument xml = new XmlDocument();

try

{

xml.Load(Directories.GetGrammarXML());

foreach (XmlNode command in xml.ChildNodes[1].ChildNodes)

{

AddGrammar(LoadCommand(command));

}

}

catch (Exception)

{

Message.ErrorMessage("No se ha encontrado el archivo de gramática: "+Directories.GetGrammarXML()+", no se cargará la gramática de voz");

}

}

*LoadCommmand* recorre por cada hijo del XML cada uno de los comandos y genera una *Grammar* de cada Nodo de XML valido. Un *Grammar* a su vez necesita el *GrammarBuilder* que a su vez requiere los *Choices*.

private Grammar LoadCommand(XmlNode commandNode)

{

GrammarBuilder grammarBuilder = new GrammarBuilder();

foreach (XmlNode choice in commandNode.ChildNodes)

{

grammarBuilder.Append(LoadChoices(choice));

}

Grammar command = new Grammar(grammarBuilder);

//introduzco nombre de la gramatica

command.Name = commandNode.Attributes[0].Value.ToString();

return command;

}

*LoadChoices* devuelve el objeto *Choices* para un nodo del árbol dado.

private Choices LoadChoices(XmlNode choiceNode)

{

Choices choices = new Choices();

foreach (XmlNode element in choiceNode.ChildNodes)

{

if (element.FirstChild == null)

{

choices.Add(" ");

}

else

{

choices.Add(element.FirstChild.InnerText);

}

}

return choices;

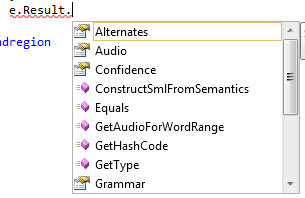
}

### Línea de proceso de reconocimiento de un comando

En este caso de uso vamos a seguir una traza estándar de lo que ocurre en el sistema desde que el usuario cita un comando hasta que este es procesado.

Supuesto el sistema iniciado y la gramática cargada en el reconocedor:

1. El usuario cita al micrófono un comando de voz.
2. Desde SAPI salta un evento llamado *speechRecognizer* que tiene como argumento un objeto de tipo *SpeechRecognizerEventArgs* que contiene una propiedad llamada *Result* que contiene información de lo citado por el usuario.



28 Propiedades y métodos del objeto *SpeechRecognizerEventArgs*

1. Capturamos ese evento en un método que, en caso de pasar un umbral determinado del 70% de confianza (por defecto) es enviado de nuevo el evento a la próxima clase en jerarquía, la clase VoiceEngine.

void SpeechRecognized(object sender, SpeechRecognizedEventArgs e)

{

if (e.Result.Confidence \* 100 >= precision)

{

speechRecognized(sender, e);

}

}

1. En el módulo IntelliRoom tenemos una clase llamada *InterpreterSpeech* que está suscrita al evento público que está en *VoiceEngine* y especificado en *IVoiceEngine*. Cada vez que el evento salta, se ejecuta en *InterpreterSpeech* el siguiente tratamiento:

void speechRecognition(object sender, RecognitionEventArgs e)

{

String result = "";

string[] commands = SeparateCommands(e.Result.Grammar.Name);

foreach (string command in commands)

{

result += Command(e.Result.Grammar.Name) +", ";

}

IntelliRoomSystem.voiceEngine.Speak(result);

}

1. Por ultimo llama al método *Command* que ejecutará el comando que este escrito en el *e.Result.Grammar.Name* que, recordando conceptos anteriores, es el que está escrito en el XML con el atributo “Name” de la etiqueta <command>.
2. Por último el comando o comandos son traducidos y ejecutados, si están disponibles, por reflexión en la clase *Reflection*.

# Módulo Media

## Introducción

El módulo Media es el encargado de satisfacer todo el conjunto de problemas relacionados con reproducción de sonidos/música y control de los mismos, para ello utilizaremos el SDK de Windows Media Player, que se integra perfectamente con el reproductor que tiene el mismo nombre, dándonos por defecto funcionalidades tan interesantes como *“reproducción de equipos remotos”, “búsquedas por su biblioteca de medios”, “soporte para un montón de formatos de audio”*.

## WMP SDK

WMP SDK o Windows Media Player Software Developer Kit es un kit de desarrollo que permite interactuación con el reproductor de Windows media.

Sobre este SDK tengo que añadir que hay poca documentación y para descubrir su funcionamiento me he basado en pruebas

## Requisitos funcionales

* Análisis de la biblioteca de música del usuario.
* Reproducción de canciones de la biblioteca definidas por filtros como pueden ser: “canciones del artista Dire Straits”.
* Control de la reproducción.
* Información de archivos en reproducción o biblioteca de música.

## Diagrama de diseño

Media se compone de 2 clases (*MusicMedia* y *MediaPlayer*) y una interface (*IMediaPlayer*).

La entidad *MediaPlayer* es la encargada de gestionar la reproducción. Listemos sus funciones por categorías:

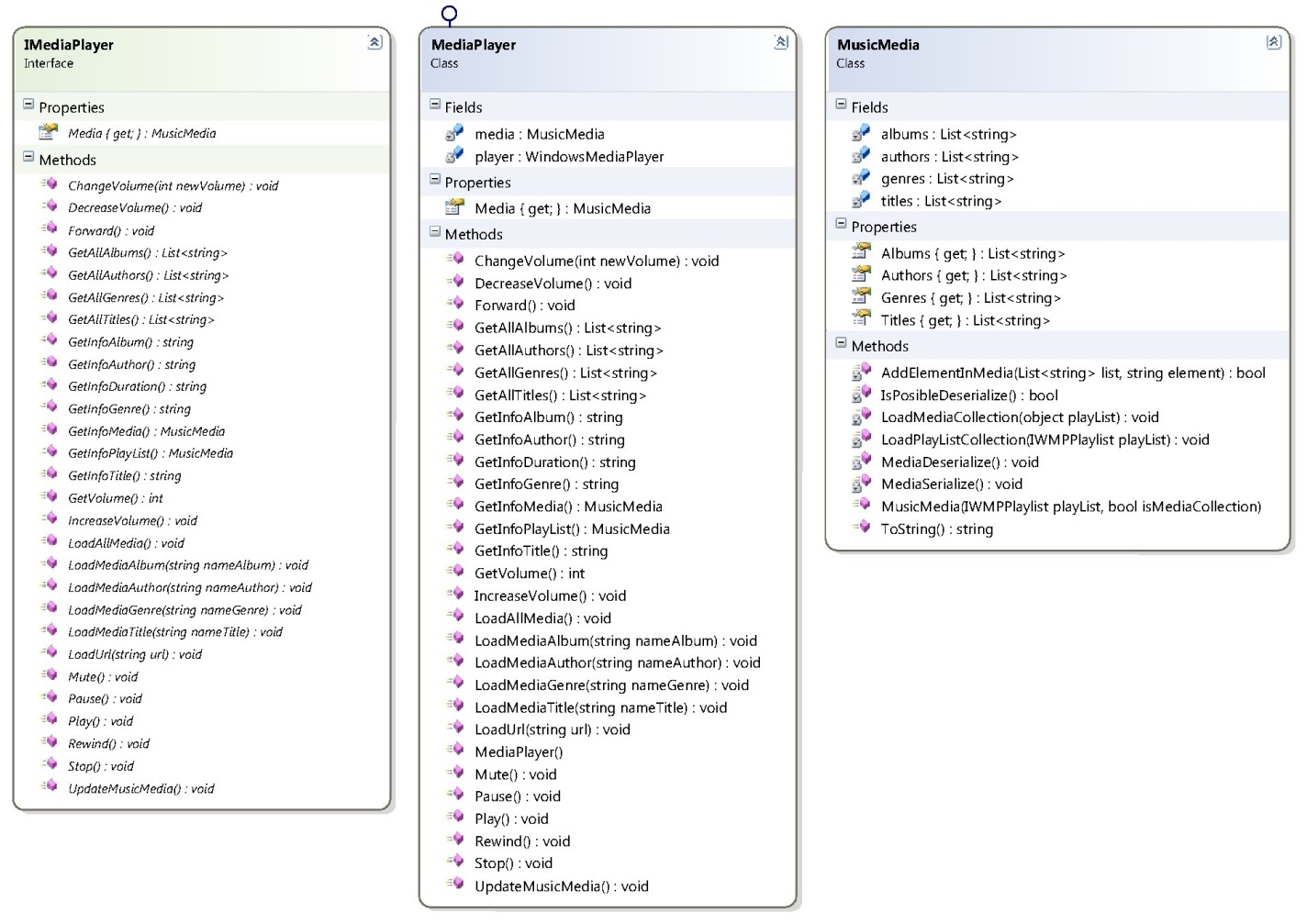
* Navegar por una lista de reproducción: Reproducir, pausar, parar, siguiente canción, canción anterior.
* Gestión de volumen: Aumentar, disminuir, silencio, definir volumen o consultar su valor.
* Información de la pista actual, de la playList o de su medioteca.
* Carga por contenido disponible en el PC: Por artista, álbum, género o título.

La clase *MusicMedia* es la encargada de extraer información de la playList actual en reproducción o de toda la medioteca de canciones. Su creación fue pensada por la razón de que la carga de esta información es lenta de conseguir (en torno a 30 segundos para 10.000 canciones) y es útil obtenerlo rápidamente para dar casos de uso como el del siguiente ejemplo:

1. “Usuario indica con su voz el comando <<cargar artista>>”
2. IntelliRoom carga en la gramática todas las posibilidades de artistas.
3. Usuario dice un artista que está contenido en ese conjunto.
4. IntelliRoom carga las canciones de ese artista en el reproductor.

[CONCLUSIONES]

Sin la existencia de *MusicMedia*, el paso entre el paso 1 y 2 en tiempo no solo sería percibible por el usuario. Por lo que se concibió el análisis de artistas, álbumes, títulos y géneros de canciones en hilos aparte que van actualizando progresivamente esas listas y estas puedan ser extraídas en tiempo casi instantáneo para el usuario.



29 Estructura del paquete media

## Casos de usos relevantes

### Cargar librería de música

La carga, como es costosa en tiempo, se hace en hilo aparte (si se trata únicamente de cargar la lista completa de todos los archivos de música indexados por WindowsMediaPlayer, en los demás casos se carga en el hilo principal.

Crea 4 listas, y recorre la lista “playList” completa preguntando en cada elemento, si ya está contenido en la lista correspondiente, si no es así, lo añade.

private void LoadPlayListCollection(IWMPPlaylist playList)

{

List<string> authors = new List<string>();

List<string> genres = new List<string>();

List<string> albums = new List<string>();

List<string> titles = new List<string>();

for (int i = 0; i < playList.count; i++)

{

if (!ExistElementInMedia(authors,playList.Item[i].getItemInfo("Author")))

{

authors.Add(playList.Item[i].getItemInfo("Author"));

}

if (!ExistElementInMedia(genres,playList.Item[i].getItemInfo("Genre")))

{

genres.Add(playList.Item[i].getItemInfo("Genre"));

}

if (!ExistElementInMedia(albums, playList.Item[i].getItemInfo("Album")))

{

albums.Add(playList.Item[i].getItemInfo("Album"));

}

if (!ExistElementInMedia(titles, playList.Item[i].getItemInfo("Title")))

{

titles.Add(playList.Item[i].getItemInfo("Title"));

}

}

//asignar a la clase

this.albums = albums;

this.authors = authors;

this.genres = genres;

this.titles = titles;

}

El método *LoadPlayListCollection* es participante directo, junto con otros métodos, en la aceleración del programa, añadiendo funcionalidad de serialización, deserialización, si se trata de la clasificación de la medioteca.

# Módulo Camera

El módulo de tratamiento de imágenes o “camera” es la única que toca el apartado de domótica enfocada a la seguridad ya que contiene una función de detección de movimiento y otra búsqueda de rostros (así como cualquier otro elemento del que se tenga el XML de entrenamiento) y un método de iluminación para, por ejemplo, encender una lámpara en caso de que la habitación se encuentre deficientemente iluminada.

## OpenCV / EmguCV

Para el tratamiento de imágenes hemos usado OpenCV o (Open Computer Vision) que es una biblioteca libre de tratamiento de imagen, desarrollada originalmente por Intel en 1999. OpenCV es multiplataforma (Linux, Mac y Windows), muy eficiente (desarrollada en C y C++) y contiene más de 500 funciones de tratamiento de imagen. Por lo que OpenCV satisface todas nuestras posibles necesidades.

Para solucionar muchos problemas de implementación provenientes del hecho de que OpenCV está escrito en C/C++ hemos usado en el proyecto usaremos EmguCV que es una implementación para C# de OpenCV.

EmguCV es un wrapper para .NET de la librería OpenCV. Este wrapper es compatible con lenguajes como C#, VisualBasic, VisualC++, IronPython… incluso puede ser compilado en mono para ser ejecutado en entornos Linux o Mac OSX.

EmguCV aun siendo un wrapper tiene a sus espaldas OpenCV que es una librería con una extensísima bibliografías.

## Requisitos funcionales

Los objetivos de este módulo son:

* Reconocimiento facial
* Detector de movimiento
* Detector de iluminación
* Guardado de imágenes en local o en servidores
* Gestor y captura de cámaras web instalas en el ordenador

## Estructura del módulo.

La estructura está compuesta de 1 interface (*IImageEngine*) y 5 clases (*ImageEngine, ImageUtils, FaceResult, Camera, Config* y *LastResults*), comencemos describiendo las clases, puesto que la interface solo es el contrato para la clase principal ImageEngine:

*Camera* representa una cámara, encapsula en objeto Capture del paquete Emgu.CV, nos abstrae de ese objeto y nos devuelve imágenes de él y trata su conexión.

*ImageUtils* contiene métodos estáticos para el tratamiento de imágenes. Estos métodos, serán llamados desde *ImageEngine*. Los tratamientos de imágenes disponibles son:

* Calculo de iluminación: Se basa para hacerlo en los valores dados de las componentes, por ejemplo si tenemos muchos valores altos de píxeles se deducirá que tenemos mucha iluminación en la habitación.
* Calculo de movimiento: Hace operaciones entre la imagen actual y la anterior, para devolverte un valor de 0 a 100 con decimales de la diferencia existente entre las dos fotografías.
* Calculo facial: se basa en hacer búsquedas en cascada por la imagen de un XML entrenado para reconocer rostros.
* Guardado de imágenes: tiene dos métodos, uno enfocado al guardado de imágenes completas y otro para guardar una zona específica de la imagen.

*FaceResults* es el objeto que devuelve el método *FaceDetect* de la clase *ImageUtils,* en esencia se trata de un encapsulado de una imagen y de un conjunto de rectángulos que contienen el área de la que está compuesta cada una de las caras.

public class FaceResult

{

Image<Bgr, Byte> image;

List<Rectangle> faces;

public FaceResult(Image<Bgr, Byte> image)

{

faces = new List<Rectangle>();

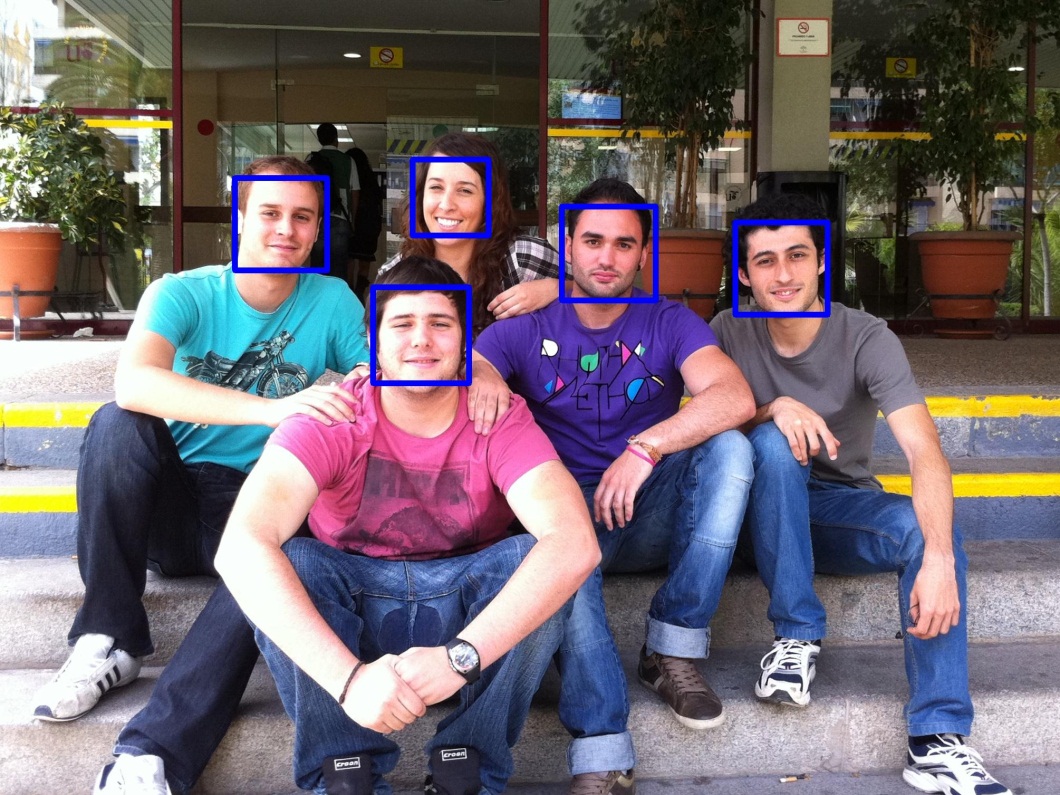
image.ROI = Rectangle.Empty;

image = image.Copy();

}

…

Veamos un ejemplo gráfico de como representa el resultado del reconocedor facial usando un *FaceResult*: por un lado tendríamos la imagen y por el otro, pintado en azul, los rectángulos que tenemos guardados en la lista *faces* correspondiendo a cada una de las caras detectadas.

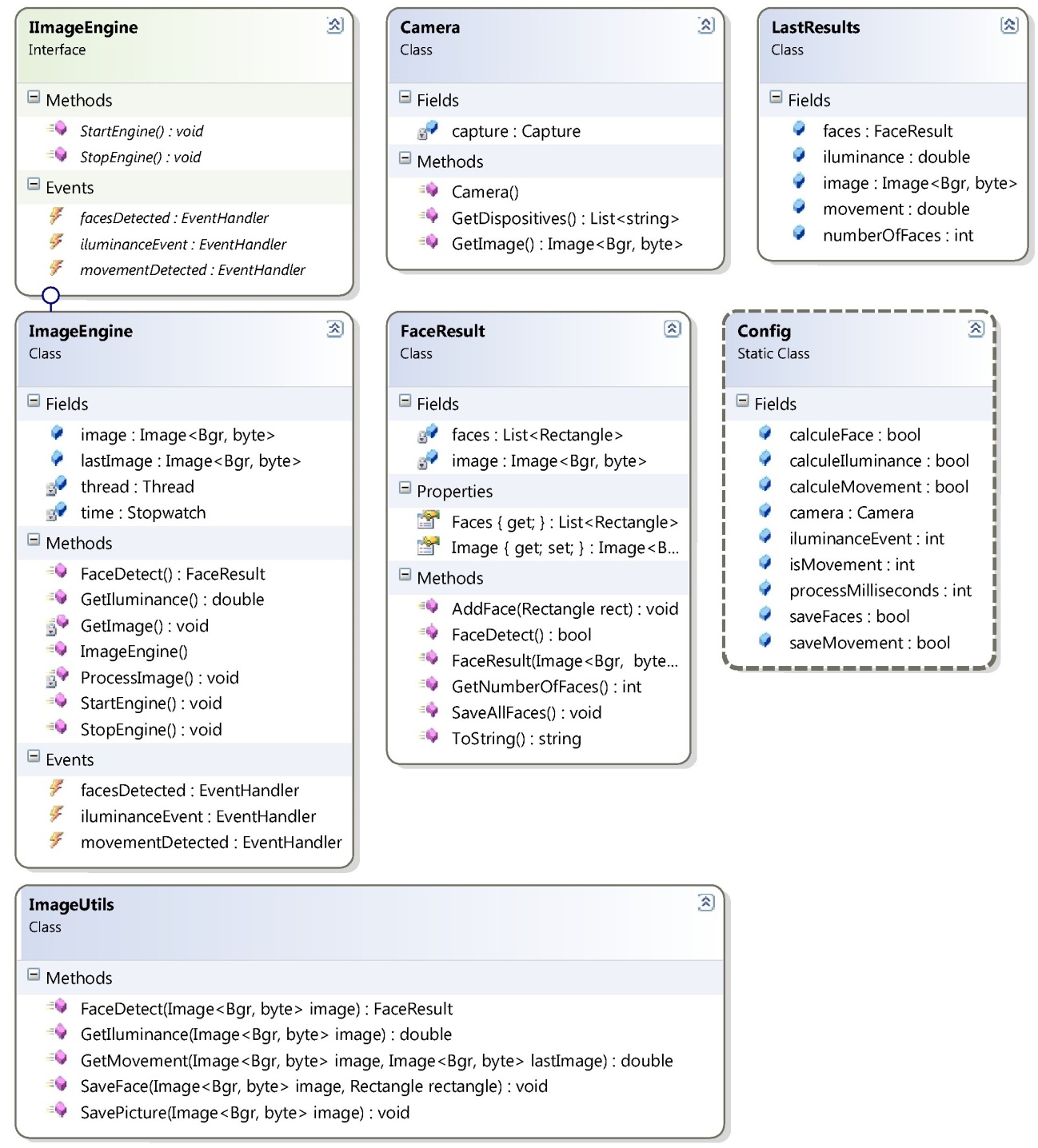


30 Ejemplo de haarcascade con XML de rostros frontales

*Config* contiene una serie de objetos estáticos de configuración y la instancia de la clase Camera, si fuera necesaria una instantánea de cámara en ese momento. La configuración disponible es sobre todo usada por el motor de imágenes y sus posibilidades son:

* Tiempo entre dos procesamientos de imagenes: el procesamiento de imagen es un proceso costoso y si no ponemos un valor alto en milisegundos podríamos mantener al ordenador con una alta carga de trabajo constante y eso se traduce en consumo y calor. Un valor a 1000 milisegundos podría ser un buen valor por defecto (el método que más consume en tiempo es *FaceDetect* que puede tardar hasta 300 ms para resoluciones de 640x480 píxeles)
* Que procesamientos vamos a llevar a cabo por cada intervalo de *processMilliseconds* milisegundos: podemos desactivar y activar cualquiera de los 3 (iluminación, reconocimiento facial, detector de movimiento).
* La precisión con la que definimos los saltos de eventos: Ejemplo ¿30% de movimiento entre imágenes es considerado movimiento?, de ser así ponemos *isMovement* debe valer 30.
* El guardado de imágenes: podemos definir si queremos guardar las imágenes o los rostros de las personas que aparezcan en la captura.

Por último *ImageEngine,* tiene 5 métodos StartEngine y StopEngine que activan y desactivan el gestor y otros 3 métodos de tratamiento de imágenes. Start/Stop Engine Inicia el cálculo de procesamiento de imágenes en un hilo aparte. La manera de ejecutarse dependerá de cómo este configurado en *Config*, así como la gestión de eventos es realizada por ella misma. Al final de cada procesado completo guarda el resultado en una instancia de la clase *LastResults* que tiene un resumen de todo el procesado.



31 Estructura del paquete camera

## Métodos Relevantes.

### Calculo de movimiento

Para calcular el movimiento, la función con prototipo:

*static double GetMovement(Image<Bgr, Byte> image, Image<Bgr, Byte> lastImage)* coge ambas imágenes y seguir la siguiente secuencia paso a paso:

1. Aplicamos a ambas imágenes un filtro de media con tamaño de la matriz de convolución 9, es un filtro alto para eliminar posibles defectos que pueda tener la cámara.
2. Restamos entre si ambas fotografías obteniendo las diferencias entre ellas, con la resta “lastImage-image” obtendríamos una imagen que tiene en negro todos los pixeles que fueran iguales o más altos (con más valor en cada componente). Y con “image-lastImage” nos quedamos con los iguales o más bajos. Cuando me refiero a valores más altos o bajos me refiero al caso más sencillo, una fotografía en blanco y negro, cuando hablamos de fotografías a color, se restan los valores de cada componente de un pixel y serian estos los que son más claros o más oscuros.
3. Sumamos las dos restas obteniendo las diferencias de valores más claros y más oscuros en una sola imagen.
4. Dependiendo de los valores de cada pixel podemos deducir el movimiento total.

Vamos a ver este proceso con un ejemplo ilustrativo:

[EJEMPLO CON FOTOGRAFIA EN EL PASILLO]

Por último veamos el código en C# para EmguCV.

public static double GetMovement(Image<Bgr, Byte> image, Image<Bgr, Byte> lastImage)

{

//aplicamos una media para "solucionar" posibles errores que puedan ser cometidos por la cámara

Image<Bgr, byte> imageA = image.SmoothMedian(9);

Image<Bgr, byte> imageL = lastImage.SmoothMedian(9);

//hacemos sus diferencias

Image<Bgr, byte> imageSub1 = imageA.Sub(imageL);

Image<Bgr, byte> imageSub2 = imageL.Sub(imageA);

//sumamos sus diferencias

Image<Bgr, byte> imageOr = imageSub1.Or(imageSub2);

//vemos cuanto valor tiene esa imagen reutilizando la función GetIluminance

return GetIluminance(imageOr);

}

### Reconocimiento facial

Para el reconocimiento facial usaremos el método

*public static FaceResult FaceDetect(Image<Bgr, Byte> image),* que, dada una imagen, genera un *FaceResult* (que es la misma imagen y una lista con rectángulos de imágenes). Para el procesamiento de este método requeriremos un objeto de la clase *HaarCascade*, que a su vez requiere de un XML de entrenamiento, (el XML usado por defecto en el proyecto es de reconocimiento facial frontal) que es el que necesita la imagen para el método *DetectHaarCascade.*

[Breve Explicación de HaarCascade]

Por ultimo podríamos ver código utilizado para la función.

public static FaceResult FaceDetect(Image<Bgr, Byte> image)

{

FaceResult result = new FaceResult(image);

//convierto a escala de grises

Image<Gray, Byte> gray = image.Convert<Gray, Byte>();

//normalizamos el brillo y mejoramos el contraste

gray.\_EqualizeHist();

//leemos el XML con el entrenamiento (en nuestros caso usamos uno de caras frontales)

HaarCascade face = new HaarCascade("HaarCascade\\haarcascade\_frontalface.xml");

//Detectamos las caras de la imagen en blanco y negro

//El primer dimensional contiene el canal (solo nos centraremos en el canal 0, porque estamos trabajando en blanco y negro)

//El segundo dimensional es el indice del rectangulo

MCvAvgComp[][] facesDetected = gray.DetectHaarCascade(face, 1.5, 10, Emgu.CV.CvEnum.HAAR\_DETECTION\_TYPE.DO\_CANNY\_PRUNING, new Size(20, 20));

//Por cada rectangulo detectado, lo incluimos en el resultado

foreach (MCvAvgComp f in facesDetected[0])

{

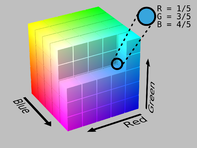
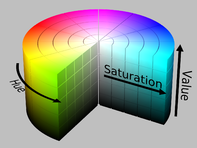
result.AddFace(f.rect);

}

return result;

}

### Cálculo de iluminación

En el cálculo de iluminación lo que hacemos es cambiar de modelo de color de RGB (Red, Green, Blue), que representa cada uno de los colores primarios en intensidad, a HSV (Hue, Saturation, Value) que es un modelo de color que codifica las imágenes en 3 componentes: en una componente la tonalidad, en otra la saturación y por último el brillo en cada uno de los pixeles de la imagen. A esta última componente, *Value*, le calculamos la media y leemos un valor de cualquier pixel de la imagen obteniendo así la iluminación media de la captura o imagen.

Para tener más claro el funcionamiento, vamos a ver un ejemplo práctico con una imagen cualquiera.

[Ejemplo ilustrativo]

Y el código para su cálculo en C#.

public static double GetIluminance(Image<Bgr, Byte> image)

{

//transformamos la imagen de RGB to HSV

Image<Hsv, Byte> imageHSV = image.Convert<Hsv, Byte>();

//calculo la media del componente "V"

double value = imageHSV.GetAverage().Value;

//para darlo en funcion de porcentaje (0 a 100)

return value/255\*100;

}

# Módulo Utils

El módulo *utils* está pensado para añadir funcionalidad extra que no ha sido encajada en ninguno de los módulos anteriores y que no puedan estar contenida en el módulo principal del proyecto (IntelliRoom).

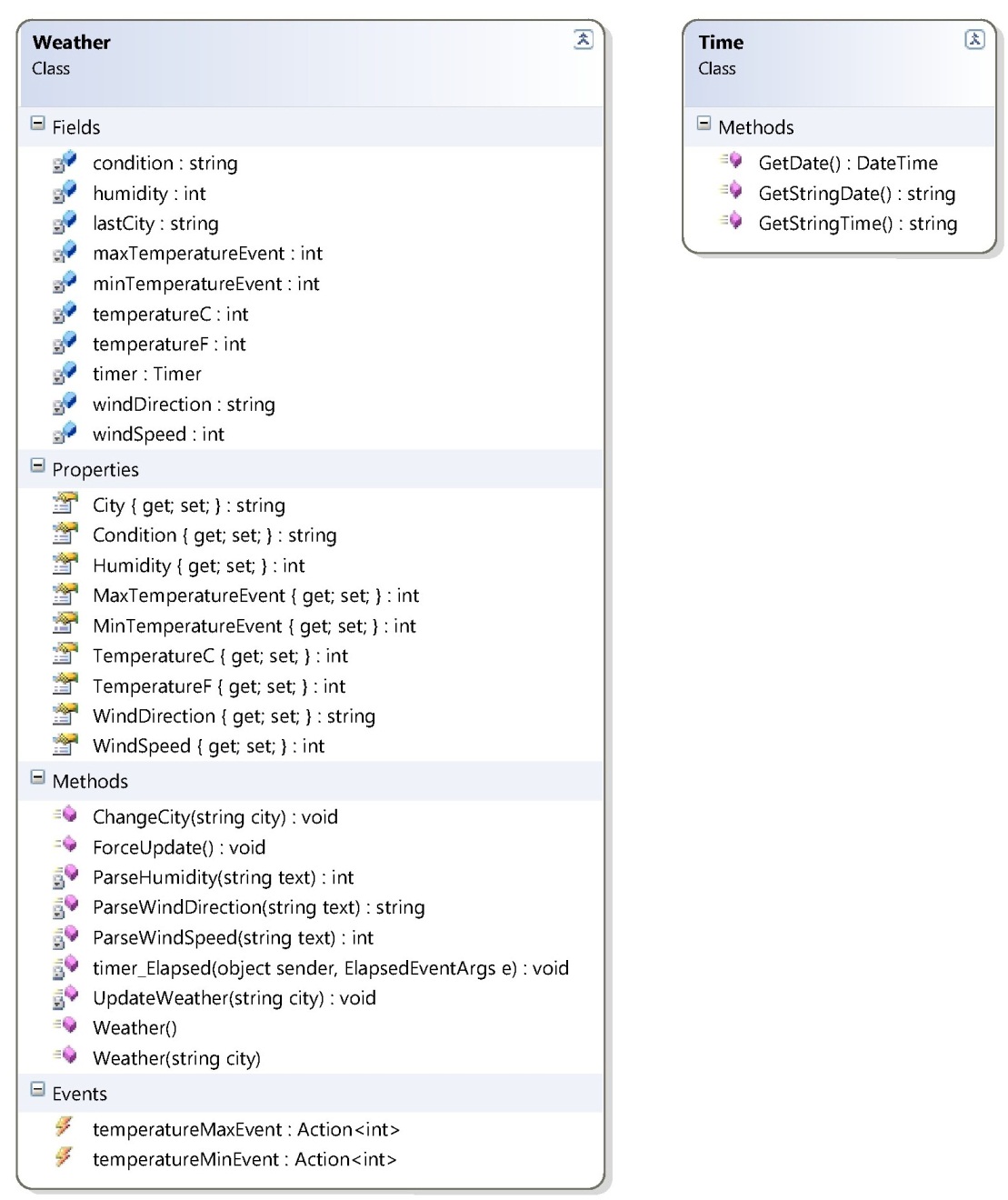
Como funcionalidades incluidas por defecto tenemos utilidades que nos permiten preguntar sobre la fecha, climatología o gestión de un simple sistema de alarmas.

## Requisitos funcionales

* Proporcionar al sistema un lugar donde añadir funcionalidad variada que no está contenida en ningún otro paquete del proyecto.
* Obtener información sobre fecha y hora.
* Obtener información climatológica.
* Administración de alarmas.

## Estructura de diseño

Se compone de dos clases que gestionan un sistema de alarmas, *Alarm* y *Alarms*. Una clase para información climatológica *Wheather* y por ultimo *Time* utilizada para devolver fechas y horas.



32 Estructura del paquete utils

## Weather API

Para el objetivo funcional de información climatológica hemos usado una API externa, la API de Google Weather, que nos da muchísima información referente al tiempo de cada ciudad del mundo, pasándole como parámetros el nombre de la ciudad o su código postal.

### Funcionamiento de Google Weather

Google Weather no tiene documentación oficial, pero es tan sencilla que puede ser usada a ciegas con bastante buenos resultados.

La URL principal de la API es: <http://www.google.com/ig/api?>

Esta página nos proporciona un XML sin información, puesto que si queremos información al respecto tenemos que pasarle por get algunos de los parámetros que soporta, en concreto buscando por webs y probando he encontrado dos:

* weather=: permite definir el nombre de la ciudad o código postal de la que se desea obtener información climatológica. Veamos un ejemplo, en mi caso si quisiéramos información del clima de mi pueblo, “Camas” con código postal 41900, la dirección quedaría de la siguiente manera:

<http://www.google.com/ig/api?weather=Camas,Sevilla>

<http://www.google.com/ig/api?weather=41900>

* hl=: también es posible definir el idioma en el que quieres que te devuelva el XML. De esta manera si queremos que nos la devuelva en español (es) haríamos la petición de la siguiente manera:

<http://www.google.com/ig/api?weather=Camas,Sevilla&hl=es>

La respuesta a esta petición sería:



33 Respuesta dada por google weather a una petición

Hemos visto que podemos adquirir de manera sencilla este XML con información no solo actual sino de predicciones futuras. Ahora solo queda parsearlo para obtener la información deseada. Toda la información es tratada en la clase “Weather” que veremos más adelante los métodos que tiene.

### La alternativa: Yahoo! Weather

Como alternativa muy atractiva encontramos la API de Yahoo! Que cuenta con documentación y que proporciona información que no podemos obtener con la API de Google.

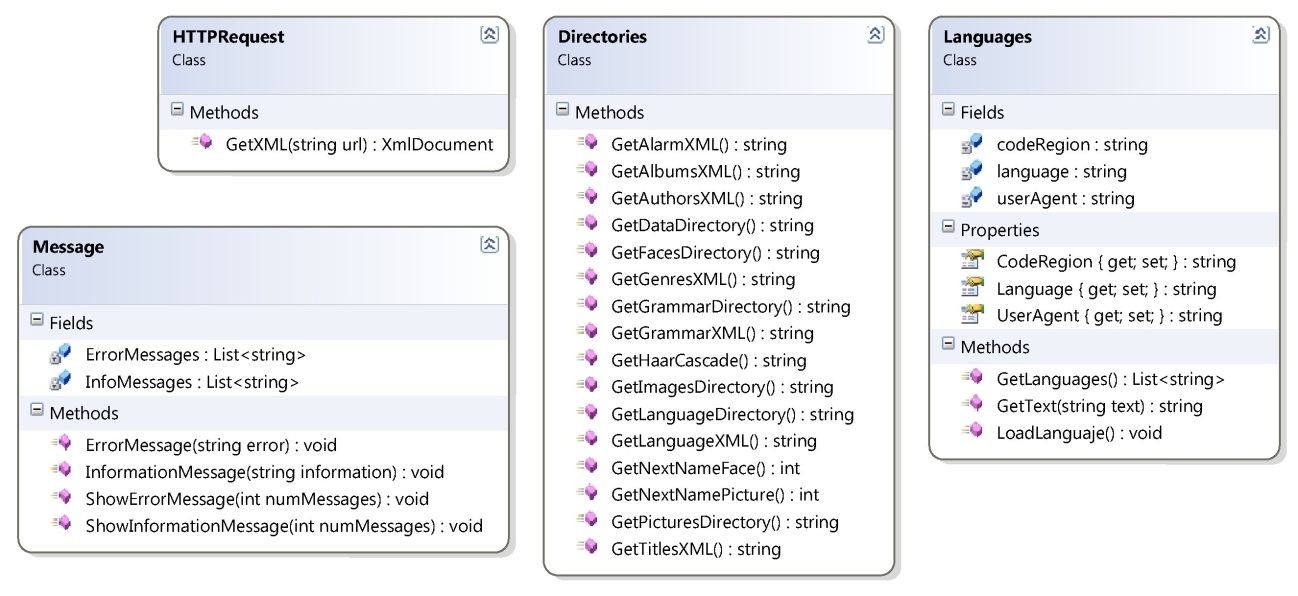
Para más información sobre Yahoo! Weather: <http://developer.yahoo.com/weather/>

## Métodos relevantes.

# Capa de Datos

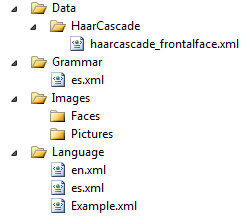
La capa de datos es la tercera y última en nuestra jerarquía, recordar que nuestra arquitectura es en 3 capas (presentación, lógica de negocio y datos). Es la encargada de gestión de ruta en el disco duro.

La capa de dato tiene la siguiente estructura:



34 Estructura del paquete de datos

Como se puede apreciar tenemos 4 clases y la mayoría de los métodos son estáticos y públicos. Todos los módulos descritos en capítulos anteriores contienen referencias a este para poder hacer consultas en cualquier momento. Navegaremos un poco por la utilidad de cada una de las clases:

* *HTTPRequest* hace consultas HTTP, en nuestro caso de momento solo tenemos una petición que dada un url nos devuelve el documento XML que este contenido en esa url, si no se conecta envía un error al usuario.
* *Message*: envía mensajes de información y error al usuario. Tiene 3 métodos: *ErrorMessage*, *InfoMessage* y *GetMessajes*. Los dos primeros muestran por consola los errores y la información y, además, guarda esos mensajes en una lista. Por el contrario, *GetMessage* obtiene todos los mensajes por orden de llegada y es utilizada para la versión con formulario en Windows Forms.
* *Directories* devuelve cada una de las rutas de cada directorio disponible en el programa. A continuación dejamos la jerarquía de directorios que tiene y una explicación de para qué sirve cada carpeta:
  + Data: Es el encargado de guardar información de serializaciones como puede ser la configuración del sistema, los artistas, discos, géneros y títulos de canciones de la biblioteca de canciones o las alarmas que tenemos guardadas en el sistema y de almacenar otros archivos de interés como el HaarCascade necesario para la detección de rostros.
  + Grammar: Contiene la gramática de cada uno de los idiomas.

35 Directorios de Intelli-Room

* + Language: Tiene los XML para la traducción del programa, y también los códigos de cada una de las regiones disponibles.
  + Images: Guardará las capturas de la cámara. Contiene 2 subdirectorios (Faces y Pictures)
* *Languages* gestiona idiomas alternativos para el sistema. Con esta clase podemos separar las gramáticas pertenecientes a cada uno de los idiomas añadidos, modificar su userAgent de las peticiones HTTP, así como otros aspectos.

# Anexo: Instalación de IntelliRoom

## Requisitos Software

Para la ejecución de este proyecto se requiere tener instalado:

* Sistema operativo Windows Vista o superior
* [Windows Media Player SDK](http://www.microsoft.com/downloads/en/details.aspx?FamilyID=e43cbe59-678a-458a-86a7-ff1716fad02f%23Instructions)
* Wrapper [Emgu CV](http://sourceforge.net/projects/emgucv/)
* [Microsoft .NET Framework 4](http://www.microsoft.com/downloads/en/details.aspx?displaylang=en&FamilyID=9cfb2d51-5ff4-4491-b0e5-b386f32c0992)

Para instalar el código en Arduino necesitaras compilar el código escrito en Processing/Wiring y traducirlo a uno entendible por ATMega328 y programarlo, esta tarea es muy sencilla si usamos el IDE de Arduino que podrá ser fácilmente descargarlo/instalado en la siguiente dirección: <http://arduino.cc/en/>

Por supuesto es necesario descargarse la última versión disponible en la forja del proyecto: <http://code.google.com/p/intelli-room/>

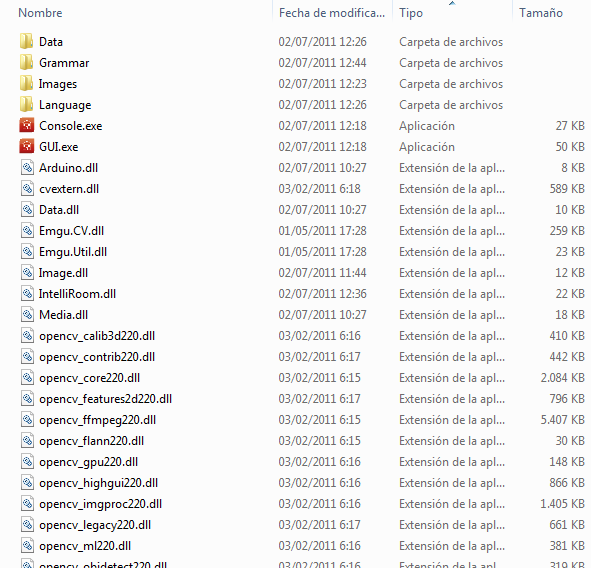
## Requisitos Hardware

Los requisitos Hardware y sus dispositivos son:

* Ordenador capaz de soportar Windows Vista
* Placa Arduino
* Tener montado el circuito del proyecto disponible en el apartado ¿¿?¿?¿
* Cámara web
* Altavoces y micrófono

## Organización del directorio principal del programa

La organización del directorio principal de IntelliRoom es la siguiente:



36 Raiz principal de Intelli-Room

Como se puede ver está compuesta de un directorio principal (IntelliRoom) del que parten otros subdirectorios, un conjunto de dlls y los ejecutables.

Los subdirectorios son, como ya explicamos anteriormente:

* Data: Es el encargado de guardar información de serializaciones como puede ser la configuración del sistema, los artistas, discos, géneros y títulos de canciones de la biblioteca de canciones o las alarmas que tenemos guardadas en el sistema y de almacenar otros archivos de interés como el HaarCascade necesario para la detección de rostros.
* Grammar: Contiene la gramática de cada uno de los idiomas.
* Language: Tiene los XML para la traducción del programa, y también los códigos de cada una de las regiones disponibles.
* Images: Guardará las capturas de la cámara. Contiene 2 subdirectorios (Faces y Pictures)

Los DLLs son la compilación de los módulos explicados durante los capítulos de este documento y archivos de la librería OpenCV y EmguCV.

Por último tenemos dos puntos de entrada en programa:

* Para usar la versión de consola ejecutaremos: *Console.exe*
* Para la versión con windowsForms: *GUI.exe*

# Anexo: Referencias

[TODO]

Para los conceptos básicos los apuntes de la asignatura procesamientos de imagen documental.

Para la ampliación de conceptos utilicé el libro: “Tratamiento digital de imágenes” escrito por Rafael C. González, editiorial “Addison-Wesley”

Para el uso de la librería OpenCV el libro: “Learning OpenCV” de Gary Bradski y Adrian Kaehler, editorial “O’Reilly”

Y por último los ejemplos que incluyen tras su instalación EmguCV y búsquedas en internet.