****

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

Índice

[1 IntelliRoom como asignatura 5](#_Toc291173390)

[1.1 ¿Qué se pretende con IntelliRoom? 5](#_Toc291173391)

[1.2 Objetivos que pretendía con el proyecto 5](#_Toc291173392)

[1.3 Habilidades o conocimientos adquiridos con el proyecto 5](#_Toc291173393)

[1.4 Agradecimientos 7](#_Toc291173394)

[2 Introducción 8](#_Toc291173395)

[2.1 ¿Qué es IntelliRoom? 8](#_Toc291173396)

[2.2 Autoría 8](#_Toc291173397)

[2.3 Licencia 8](#_Toc291173398)

[3 Requisitos de IntelliRoom 10](#_Toc291173399)

[3.1 Requisitos Software 10](#_Toc291173400)

[3.2 Requisitos Hardware 10](#_Toc291173401)

[4 Objetivos del proyecto 11](#_Toc291173402)

[5 Arquitectura Lógica Básica 12](#_Toc291173403)

[5.1 Presentación 12](#_Toc291173404)

[5.2 Lógica de negocio 12](#_Toc291173405)

[5.3 Datos 12](#_Toc291173406)

[6 Arquitectura Hardware 14](#_Toc291173407)

[6.1 Diagrama de despliegue 14](#_Toc291173408)

[6.2 El porqué de esta distribución 14](#_Toc291173409)

[7 Electrónica 15](#_Toc291173410)

[7.1 Introducción 15](#_Toc291173411)

[7.2 Conceptos de electrónica necesarios 15](#_Toc291173412)

[7.2.1 PWM 15](#_Toc291173413)

[7.2.2 Duty Cycle 15](#_Toc291173414)

[7.3 Elección del microcontrolador 16](#_Toc291173415)

[7.3.1 Requisitos microcontrolador 16](#_Toc291173416)

[7.3.2 PIC 16F1828 / 16D1824 16](#_Toc291173417)

[7.3.3 Arduino 18](#_Toc291173418)

[7.4 Coloreado 19](#_Toc291173419)

[7.4.1 Circuito eléctrico 19](#_Toc291173420)

[7.4.2 Funciones y protocolo de mensajes 21](#_Toc291173421)

[7.5 Dispositivos 22](#_Toc291173422)

[7.5.1 Circuito eléctrico 23](#_Toc291173423)

[7.5.2 Funciones y protocolo de mensajes 24](#_Toc291173424)

[7.6 Código Arduino 24](#_Toc291173425)

[7.6.1 Pseudocódigo 24](#_Toc291173426)

[7.6.2 Explicación 25](#_Toc291173427)

[7.6.3 Código final 25](#_Toc291173428)

[7.7 Circuito completo 25](#_Toc291173429)

[7.8 Puerto serie 25](#_Toc291173430)

[7.9 Agradecimientos 25](#_Toc291173431)

[8 Módulo Arduino 27](#_Toc291173432)

[8.1 Objetivos 27](#_Toc291173433)

[8.2 Casos de usos relevantes 27](#_Toc291173434)

[8.2.1 Creación de conexión Serial 27](#_Toc291173435)

[8.2.2 Procesamiento de un mensaje 27](#_Toc291173436)

[8.3 Diagrama de diseño 27](#_Toc291173437)

[9 Módulo Voice 29](#_Toc291173438)

[9.1 Introducción 29](#_Toc291173439)

[9.2 SAPI 30](#_Toc291173440)

[9.3 Objetivos 31](#_Toc291173441)

[9.4 Casos de uso relevantes 31](#_Toc291173442)

[9.4.1 Sintetizar 31](#_Toc291173443)

[9.4.2 Añadir gramática 31](#_Toc291173444)

[9.4.3 Línea de proceso de Reconocimiento 31](#_Toc291173445)

[9.5 Diagrama Diseño 32](#_Toc291173446)

[10 Módulo Media 33](#_Toc291173447)

[10.1 Introducción 33](#_Toc291173448)

[10.2 WMP SDK 33](#_Toc291173449)

[10.3 Objetivos 33](#_Toc291173450)

[10.4 Casos de usos relevantes 33](#_Toc291173451)

[10.4.1 Cargar librería de música 33](#_Toc291173452)

[10.5 Diagrama de diseño 33](#_Toc291173453)

[11 Módulo Camera 34](#_Toc291173454)

[11.1 OpenCV / EmguCV 34](#_Toc291173455)

[11.2 Objetivos 34](#_Toc291173456)

[11.3 Casos de usos relevantes 35](#_Toc291173457)

[11.4 Diagrama de diseño 35](#_Toc291173458)

[12 Módulo Utils 35](#_Toc291173459)

[12.1 Introducción 35](#_Toc291173460)

[12.2 Objetivos 35](#_Toc291173461)

[12.3 Weather API 35](#_Toc291173462)

[12.3.1 Funcionamiento de Google Weather 35](#_Toc291173463)

[12.3.2 La alternativa: Yahoo! Weather 37](#_Toc291173464)

[12.4 Casos de uso relevantes 37](#_Toc291173465)

[12.5 Diagrama de diseño 37](#_Toc291173466)

[13 Capa de presentación 37](#_Toc291173467)

[14 Capa de Datos 37](#_Toc291173468)

[15 Anexos 37](#_Toc291173469)

# IntelliRoom como asignatura

En este apartado no abordaremos ningún aspecto técnico o conceptual de IntelliRoom como proyecto software sino que lo veremos desde la perspectiva de la asignatura proyecto fin de carrera contestando a preguntas del tipo ¿qué me aportó como ingeniero?, ¿qué conocimientos adquirí? así como el ¿por qué este proyecto y no otro?....

## ¿Qué se pretende con IntelliRoom?

Lo cierto es que incluso antes de entrar en la carrera todo el mundo habla del proyecto de fin de carrera y aún más cuando somos alumnos destinados a pasar por un proyecto. Cuando me pregunté *que quería hacer como proyecto* tenía claro que tenía que ser algo que me permitiera poner el pie en un amplio abanico de términos y después de bastantes ideas rechazadas pensé que la domótica podía ofrecerme un amplio, casi infinito me atrevería a decir, de conceptos.

Así que IntelliRoom lo que pretende, además de satisfacer la necesidad de completar un proyecto fin de carrera para obtener el título, es centrarme en un campo que me permita saciar la curiosidad, aunque sea de manera superficial, de campos como el tratamiento de imagen, procesamiento multimedia, reconocimiento del habla, sintetización de la voz, peticiones web… y la domótica permite jugar con todos estos conceptos.

## Objetivos que pretendía con el proyecto

Aunque han sido anteriormente mencionados en el párrafo anterior, mis objetivos son:

* Conocer la plataforma .NET, en concreto su lenguaje principal C#. Saliéndonos un poco de la dinámica de la carrera, quería conocer otros lenguajes de orientación a objeto con cierta profundidad.
* Conocer conceptos sobre tatamiento de reconocimiento del habla y en concreto utilizar SAPI
* La utilización de tratamiento de imágenes y profundizar en la librería OpenCV
* Adquirir conocimientos básicos de microcontroladores y electrónica.

## Habilidades o conocimientos adquiridos con el proyecto

Con una proyección bastante final del proyecto he confeccionado esta lista de habilidades que he ido adquiriendo a lo largo de mi desarrollo.

* En relación con la plataforma .NET
  + El conocimiento con cierta profundad en lenguaje C#
    - Reflexión
    - Lenguaje funcional o Linq
    - Hilos o Threads
    - Peticiones HTML
    - Parseado de XML
    - Serialización
    - Comunicación por puerto serie
    - Suscripción y creación de eventos
    - Manejo de monitores
    - Utilización de librerías externas/Wrappers
    - Internacionalización del sistema
  + Visual Studio 2010
    - Uso de su potente debugger y nuevas características como puede ser IntelliTrace
    - Generación de diagramas de UML con .NET
* Tratamiento de imagen y sonido
* Arduino
  + Utilización de la plataforma
  + Conocimientos básicos de cómo está construida su placa
  + Desarrollo en su lenguaje Processing/Wiring:
    - Sintaxis básica
    - Utilización de interrupciones
    - Librerías externas
* Microcontroladores:
  + He entrado de manera discreta en el mundo de los microcontroladores de la empresa Microchip, conociendo un poco su estructura interna, características y funcionamiento.
  + Me he adentrado en el mundo de los grandes datasheets y he conseguido encontrar respuestas a las preguntas que se me formulaban.
* Electrónica:
  + Uso de componentes básicos en la práctica.
    - Resistencias.
    - Transistores.
    - Condensadores???
    - Relés.
    - Diodos.
    - Reguladores de tensión.
    - LEDs.
    - Fuentes de alimentación.
    - Placas de prototipo.
* Otros conocimientos:
  + XML
  + Habilidad de encontrarte ante un problema y solucionarlo haciendo búsquedas por Google o por la documentación de .NET, OpenCV, SAPI…
  + Enfrentarme a la documentación de un proyecto.

## Agradecimientos

Por ultimo decir que mi labor no podría haber sido concluida sin la ayuda de Manuel Caballero, Pablo Íñigo Blasco, Victor…. y otros compañeros del club .NET

# Introducción

## ¿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 que extenderemos ampliamente en este documento).

* Iluminación:
  + Apagado general de todas las luces de la vivienda
  + Automatización del apagado/ encendido en cada punto de luz.
  + Regulación de la iluminación según el nivel de luminosidad ambiente
* Automatización de todos los distintos sistemas/ instalaciones / 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, cámaras…) y que la redistribuya a los periféricos (control de luz, altavoces…).

## 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

# Requisitos 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/>

## Requisitos Hardware

Ordenador capaz de soportar Windows Vista

Periféricos: (Hay que ser muy específico???????)

* Placa Arduino
* Micrófono
* Cámara web

Componentes:

* LED RGB
* 3 Transistores NPN 2n2222
* 3 Resistencias 1KOhm

# Objetivos del proyecto

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

# Arquitectura Lógica Básica

He dividido la arquitectura lógica en tres niveles básicos siguiendo la estructura básica de programación por capas, quedando así: capa de presentación, capa de lógica de negocio y capa de datos, vamos a hacer una breve descripción de los apartados del proyecto.

## Presentación

Es el encargado de presentar el sistema al usuario por medio de la interfaz que ofrece la interfaz de la lógica de negocio. Tiene dos módulos: GUI y Console:

* Console: Es un intérprete de consola que interpreta comandos, veremos más adelante que tipos de comandos pueden ser ejecutados y como se ejecutan.
* GUI: Una versión grafica que haga lo mismo que la versión de Console pero más amena para los usuarios. Podría hacerse en WindowsForm o Windows Presentation Fundation (WPF)
* Además de estas dos alternativas podríamos publicar un servicio web para que consumiera también información del sistema.

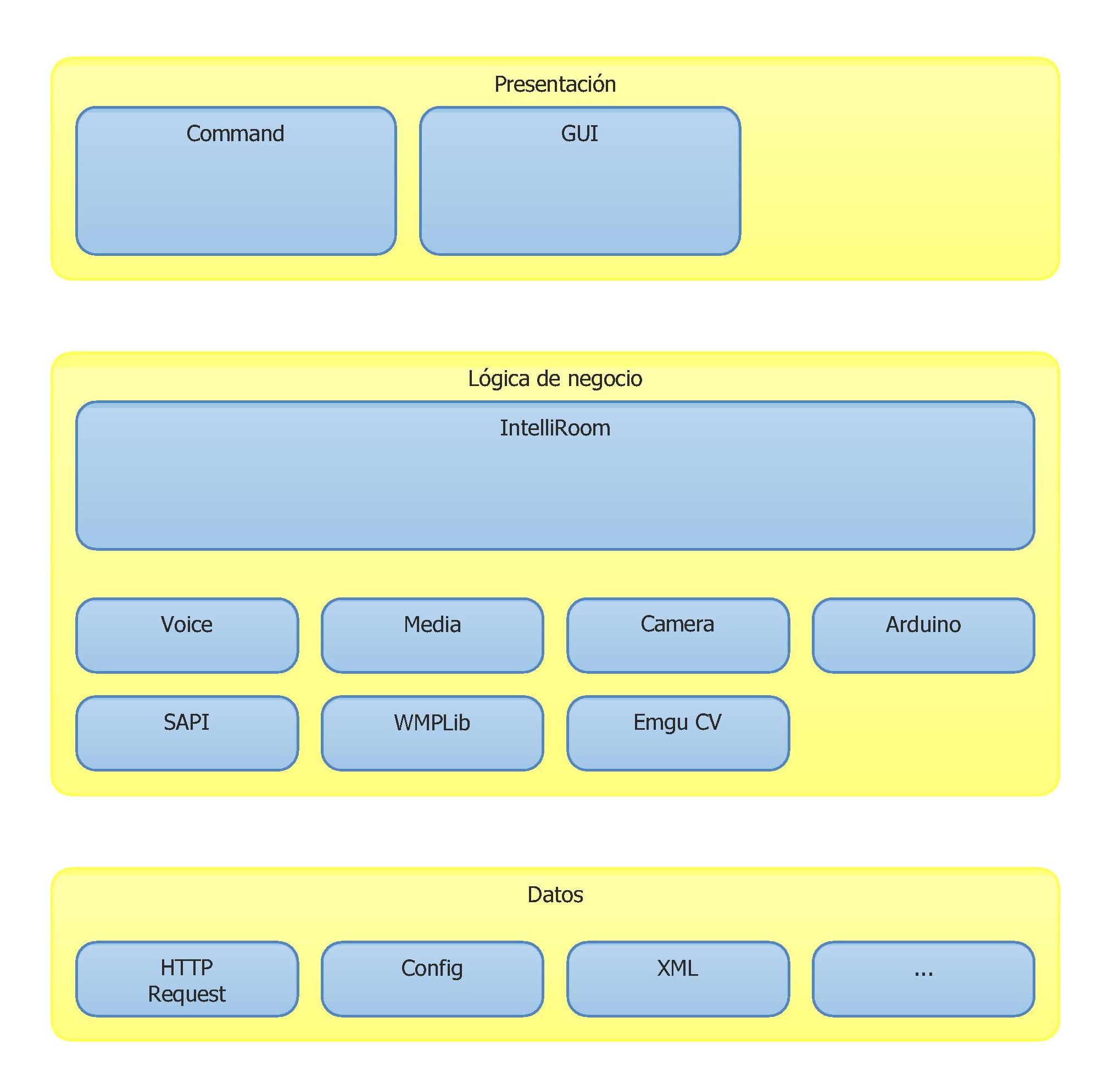
## Lógica de negocio

Es el punto central de la aplicación, que consta de dos grandes partes: nuestro motor de la aplicación (IntelliRoom) y módulos ampliables que incrementan la funcionalidad (Voice, Media, Camera…)

* IntelliRoom:
* 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 la sintonización de texto plano.
* Media: Modulo encargado de reproducción de la biblioteca de música que tenga el usuario en la maquina principal, la librería que usamos por debajo WMPLib permite, sincronización con la biblioteca contenida en la aplicación Windows Media Player, enlaces remotos, carpetas compartidas en red…
* Camera: Es el módulo de procesamiento de imágenes, por debajo usa Emgu CV, un wrapper de OpenCV.
* Arduino: Modulo de envío de órdenes a la plataforma Arduino, contiene la funcionalidad de ambientación de la habitación por colores y encendido/apagado de electrodomésticos.
* Util: 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 (configuración y tratamiento de XML).



# Arquitectura Hardware

## Diagrama de despliegue

## El porqué de esta distribución

# Electrónica

## Introducción

En este apartado contaré el proceso que experimenté y todo lo necesario que hay que saber para llegar a las conclusiones que posteriormente se darán.

La parte de electrónica del proyecto se divide 2 grandes partes: control de iluminación y control de dispositivos

## Conceptos de electrónica necesarios

### 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.

### 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:



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

[¿¿?¿?¿]Las aplicaciones del PWM son variadas: comunicaciones, efectos de sonido, amplificación, regulación de voltaje. En nuestro caso lo usaremos como regulador de voltaje que le llega al LED. Con esto ya hemos conseguido la manera de regular la potencia con la que los diodos emitirán su luminosidad. [¿¿?¿]

## Elección del microcontrolador

Una vez visto los conceptos de electrónica que tuve que tener en cuenta antes de plantearme que buscaba aho

Además, al ser RGB necesitare que el microcontrolador que seleccionemos para el proyecto tenga como mínimo 3 canales PWM para cada uno de los canales de colores.

Después de tener más o menos claro lo que necesitábamos para la parte de electrónica, era el momento de decidirse por un microcontrolador del mercado. Para encontrarlo primero definí lo que quería para no partir de búsquedas muy complejas. Aquí dejo la lista de las características mínimas que buscaba:

### Requisitos 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 necesitaba tantísima precisión así que nos quedamos con 8 bits.
* Al menos 5 o 6 salidas lógicas para la conexión de al menos 5 enchufes
* Comunicación serie y otros patillajes.
* Económico

### PIC 16F1828 / 16D1824

Después de una semanas de búsquedas y por recomendación de un compañero me decante por dos microcontroladores relativamente nuevos de la empresa Microchips, el PIC 16F1828 para el apartado de iluminación y el PIC 16F1824 para el apartado de control eléctrico.

Ambos microcontroladores son iguales en características, solo se diferencia en el hecho de que el patillado de 16F1828 está menos multiplexado que el del 16F1824, como podemos ver en las figuras a continuación:



Encapsulado del modelo PIC16F1824



ENCAPSULADO DEL MODELO PIC16F1828

Estos microcontroladores cumplían y superaban las características de la lista, tenían 4 + 2 timers para hacer PWM: 4 timers y 2 de ellos con salidas clonadas, 8 bits, salidas lógicas de sobra y un precio no superior a los 80 céntimos.

Tan pronto como me decanté por este modelo me puse a trabajar en la configuración del patillaje, llegando a la siguiente configuración:

* VDD y VSS: Alimentación.
* OSC1, OSC2: Cristal de cuarzo.
* MCLR/VPP, ICSPDAT, ICSPCLK: Programación.
* CCP1 al 4: PWMs.
* TX, RX: Interfaz con puerto serie.
* SDI, SDO, SCK, INT y uno de los terminales libres: Interfaz con el transceptor por radio. El INT se conectaría al IRQ del transceptor, de forma que cuando haya datos para procesar se avise al microcontrolador y los pueda leer.
* 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, me instale el IDE de Microchip que ¡daba soporte parcial a los micros!, y ya empezaron los problemas. Nadie había creado un compilador en C para estos micros, puesto que eran bastante nuevos, y puesto que el IDE no daba casi soporte me vi en la necesidad de buscar alternativas, utilizamos durante un tiempo un simulador para PICs, y nos pusimos manos a la obra. Escribimos el código en ensamblador necesario para comunicarnos con el microcontrolador y en el momento de querer hacer pruebas llegamos a otro problema, ¿Cómo programamos el microcontrolador? Los programadores eran bastante caros para 2 o 3 usos que le íbamos a hacer y en los departamentos no había ninguno para productos de Microchips así que decidimos hacernos uno nuestro.

Un compañero de la universidad me ayudo a buscar información y diseñó el posible programador, es más, la finalidad del trabajo era que fuera un programador universal, dejo el esquemático:



Al final entre problemas 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 no era tan buena como para a día de hoy asegurar que este programador funcionaría, 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, y un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring (aunque es posible utilizar otros muchos lenguajes). Processing/Wiring tiene una sintaxis muy amena que permite, en pocas horas, desarrollar proyectos completos.

Características de Arduino:

* 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

Después de lo antes comentado, 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 en lo que a funcionalidad se respecta. Además sería posible en un futuro, 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 por menos de 4 euros.

## Coloreado

### Circuito eléctrico

#### Componentes necesarios

##### LED RGB

En primera instancia explicaremos el proceso que experimente hasta encontrar la solución al problema de ambientar una habitación con LEDs de colores.

Buscando información sobre los LEDs, como funcionaban, de que tipo los había (tipo de luminancia, colores, potencia) y al poco concluí que necesitaba un LED RGB o 3 LEDs emparejados a 3 (que básicamente es de lo que está constituido un LED RGB), vi varios modelos y finalmente compré este modelo.

La pregunta instantánea que surge es ¿Cómo consigo conectar un dispositivo que requiere una entrada analógica para la regulación del voltaje si lo que voy a tener es un circuito formado por un microcontrolador. La respuesta se llama PWM (Pulse-Width Modulation) o lo que es lo mismo “modulación por ancho de pulsos”

##### Transistor!!!

#### Tratamientos del circuito

Ya tenemos en nuestro poder el código y un módulo programado en C# para enviar mensajes a Arduino, ya solo nos queda un detalle: La cinta de LED tiene 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

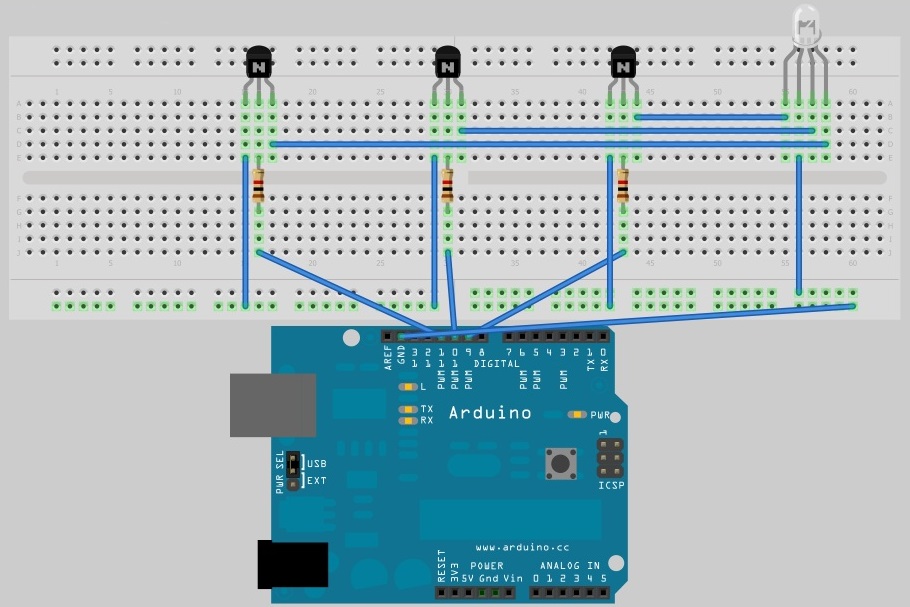
Haciendo un total de 240 mA a 12 V

Arduino ofrece 40 mA por PWM a 5 V, evidentemente no podemos conectarlo directamente:

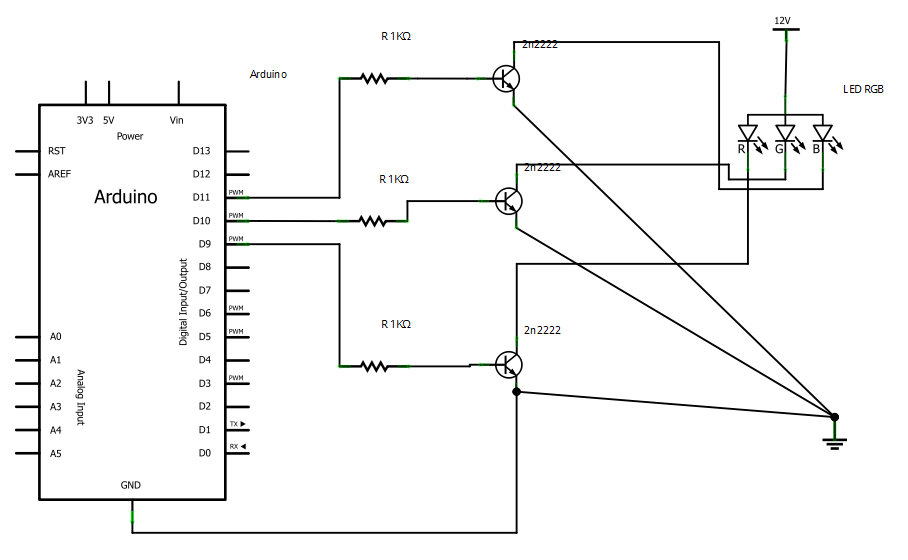
1. Porque muy posiblemente terminara quemándose el microcontrolador
2. En caso de no hacerlo no le llegaría suficiente voltaje a los LEDs

Para solucionarlo seleccionamos una fuente alternativa al circuito que ofrecía 9 Voltios a 300 mA, 3 transistores NPN modelo 2n2222 para aislar el circuito y 3 resistencias de 1KΩ (posiblemente cambiadas en un futuro)

#### Circuito en prototipo



#### Circuito en esquemático



### Funciones y protocolo de mensajes

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, de momento vamos a trabajar con este protocolo.

#### Formato general

La conexión entre la aplicación general y Arduino se hace de la siguiente manera:

Se hace el envío de una cadena que tiene la siguiente estructura:

[FUNCION] [Arg1] [Arg2]… [Argn]

#### Modo directo

Nombre función -> DIRECT

R -> Valor color Rojo, en tamaño byte

G-> Valor color Verde, en tamaño byte

B-> Valor color Blue, en tamaño byte

Comportamiento: Cambia de color directamente al color establecido por valores RGB

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

#### Modo degradado

Nombre función -> DEGRADED

R -> Valor color Rojo

G-> Valor color Verde

B-> Valor color Blue

Time -> Número de milisegundos que estará degradando

Comportamiento: Cambia de color gradualmente en un tiempo de [Time] milisegundos

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

#### Modo aleatorio

Nombre función -> RANDOM

RandomMode -> 0 = desactivar RandomMode

1 = activar RandomMode

TimeRandom -> Tiempo, en milisegundos que hay entre el paso de un color a otro

Comportamiento: Cambia de color gradual y aleatoriamente en un tiempo fijado de milisegundos

Ejemplo: RANDOM 1 1000-> Activamos la función aleatoria para que cambie de color cada 1000 milisegundos.

## Dispositivos

El control de dispositivos eléctricos del hogar es otro de los apartados de IntelliRoom. En un comienzo íbamos a usar un PIC de Microchips para aislar el problema de iluminación y conexión de dispositivos, pero finalmente, al utilizar el ATMega328 tenemos muchos pines a nuestra disposición decidimos finalmente que fuera una placa Arduino la encargada de solucionar ambas necesidades.

Para ello teníamos que hacer algunos cambios en el código, ya que anteriormente solo utilizábamos Arduino para una sola finalidad.

En aspectos cuantitativos introducimos la nueva funcionalidad, el conectado y desconectado de dispositivos eléctricos. Pero en aspectos cualitativos teníamos que hacer un sistema que, en el caso de encontrarse haciendo un proceso de degradación (un proceso que ocupa la potencia de cálculo de Arduino) y en ese instante llegara una nueva orden del tipo “enciende el dispositivo 1” o “degrada a este otro color en t tiempo” fuera capaz de gestionarla.

¿Cómo hacemos esto? Muy sencillo generando una interrupción por cada mensaje que llegue por puerto serie. En ese instante Arduino trata el nuevo comando y posteriormente continuaría con el procesamiento de un posible degradado.

PSEUDOCODIGO NUEVO

¿COMPONENTES?

[Relés]

¿CIRCUITOS?

Nuevos diagramas control dispositivo

¿ORDENES?

Conjunto de nuevas ordenes

Trabajo final de Arduino

Código final

Esquemático finales (XBee??, fuente….)

### Circuito eléctrico

#### Componentes necesarios

#### Tratamientos del circuito

#### Circuito en prototipo

#### Circuito en esquemático

### Funciones y protocolo de mensajes

#### Formato general

Es análogo al envío de mensajes del apartado de iluminación, se hace el envío de una cadena que tiene la siguiente estructura:

[FUNCION] [Arg1] [Arg2]… [Argn]

#### Encender dispositivo

Nombre función -> SWITCHON

Dispositivo -> Numero de referencia del dispositivo que quiere encenderse.

Comportamiento: Enciende un dispositivo conectado en la salida “Dispositivo”

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

#### Apagar dispositivo

Nombre función -> SWITCHOFF

Dispositivo -> Numero de referencia del dispositivo que quiere apagarse.

Comportamiento: Apaga un dispositivo conectado en la salida “Dispositivo”

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

## Código Arduino

Con Arduino en nuestras manos y el IDE instalado procedemos a ver el código del programa para el control de luz y la explicación del mismo.

### Pseudocódigo

iniciaVariables

iniciaConexionConPuertoSerie

configuraEntradaYSalida

mientras

si (HayUnMensajeDisponible) -> ConfiguraMensaje()

si no Haz otros tratamientos si están disponibles

fmientras

configuraMensaje()

si existe el mensaje

configuraArduinoParaNuevoMensaje()

eliminaPilaDeMensaje

### Explicación

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, configura las salidas del Arduino si están activados el modo “aleatorio” o el “degradado”

Si hay un nuevo mensaje, comprueba que el mensaje que ha llegado es interpretable por uno de los mensajes que tiene a disposición, si es así configura Arduino. Y vuelve al bucle principal

### Código final

[COPIAR CODIGO ARDUINO]

## Circuito completo

## Puerto serie

## Agradecimientos

Creador de la librería “Mesages”

ayuda minolo??

esquemáticos realizados con Fritzing.??

# Módulo Arduino

El módulo Arduino es el encargado, como su nombre indica, de controlar todo lo referente a Arduino, en dos aspectos:

1. Comunicación con ATMega328
2. Sistema de mensajería que modele los comandos que, posteriormente procese Arduino

## Objetivos

Los objetivos que cumple este módulo son:

* Tener un sistema de comunicación por puerto serie óptimo para nuestro problema y obviando 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

## Casos de usos relevantes

### Creación de conexión Serial

Miramos los puertos COM disponibles

Preguntamos cual es el nuestro

Definimos velocidad a ¿¿?¿?¿ y salto de carro a ¿?¿?

### Procesamiento de un mensaje

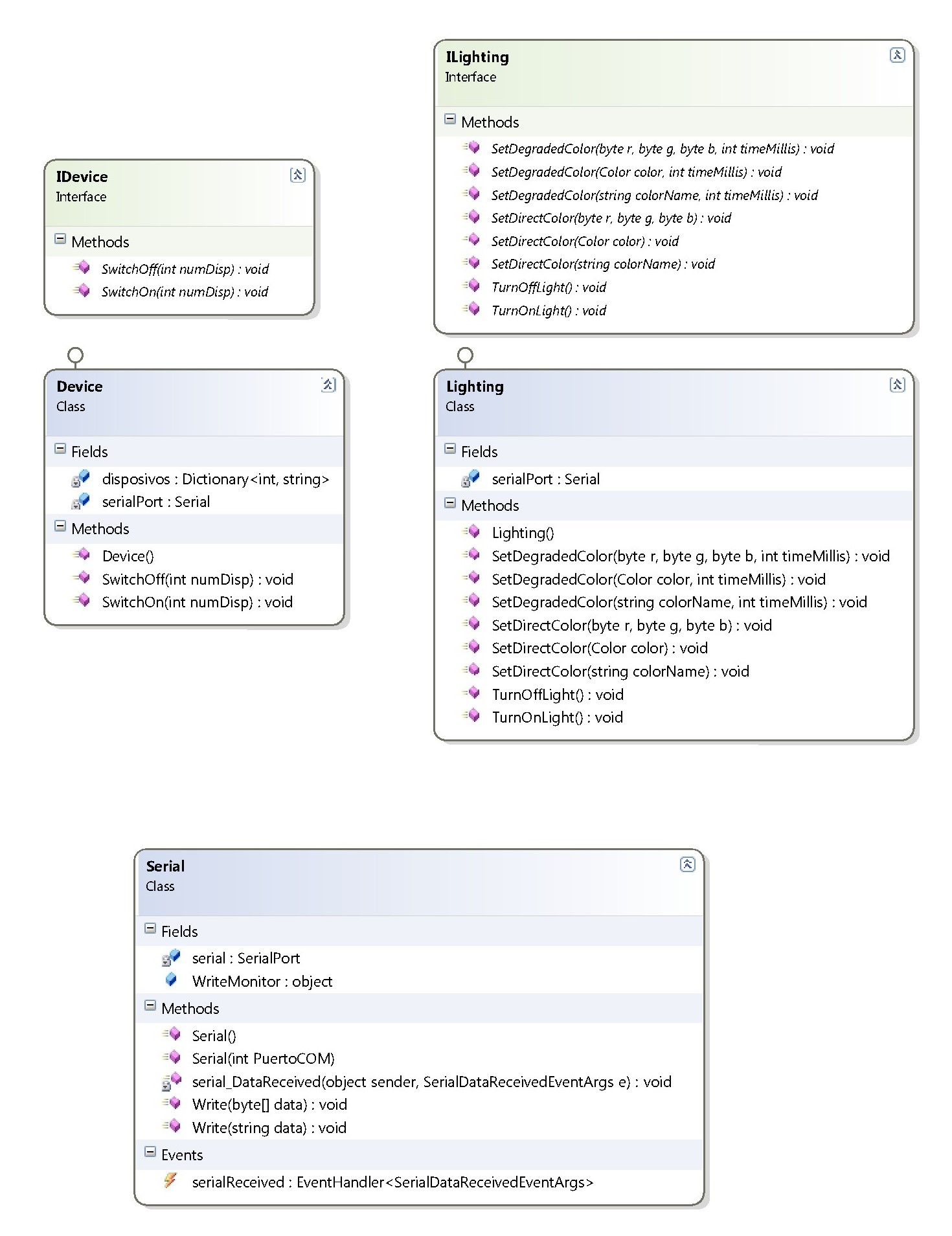
Creamos objeto Serial

Enviamos un mensaje

Llega a Arduino

Arduino procesa el mensaje

## Diagrama de diseño



# Módulo Voice

## Introducción

El módulo Voice es el modulo 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 sistema.

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

**Reconocimiento de automático del habla (RAH)** 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. En nuestro caso la semántica y la sintaxis tan muy acotadas y normalmente se proveen de paquetes que incluyen mucho entrenamiento para bastantes idiomas.
* La dependencia del hablante: si es necesario un entrenamiento personal por cada uno de los hablantes, en nuestro caso la API que posteriormente explicaremos, va entrenándose conforme vas haciendo uso de ella, además Windows tiene un apartado de entrenamiento. Según mis pruebas es posible la independencia del hablante, pero funciona mucho mejor con entrenamiento personal.
* 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.

Para más información sobre RAH en el artículo de la Wikipedia donde he extraído esta información <http://es.wikipedia.org/wiki/Reconocimiento_del_habla>.

**Sintetización 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.

Información extraída de <http://es.wikipedia.org/wiki/Sintetizaci%C3%B3n_del_habla>.

## 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, mas 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 nosotros usaremos la versión 5.4 (que es la versión para Windows 7, que es la que tengo instalada en mi máquina) pero para el uso que le vamos a dar es completamente compatible con la versión 5.3 (para 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 a nivel cuantitativas que posee, vamos a ver como SAPI las cumple cualitativamente desde el punto de vista de las pruebas que he estado realizando.

En relacion al reconocimiento funciona realmente bien, tiene mucho soporte para una gran cantidad de idiomas y, siguiendo el esquema antes mencionado de las caracteristicas que posee el reconocimiento del habla vamos a verlas una por una:

* La entrenabilidad: En el caso de IntelliRomm la semántica y la sintaxis están muy acotadas, además con los paquetes de idiomas esta necesidad esta solventada.
* 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, incluso incluye un apartado 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 la cosa cambia, 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 donde quiere ser sintetizado. Además 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.

## Objetivos

Crear un conjunto de clases que modele el reconocimiento y sintetización en el mismo momento, es decir, si estamos sintetizando paremos el reconocimiento para que no interprete lo que sintetice y de posibles errores.

Que sea capaz de gestionar normas sintácticas y gramaticales así como la posibilidad de cambiar la sensibilidad u otros parámetros con la que decide si una muestra escuchada pertenece o no a un patrón definido.

## Casos de uso relevantes

### Sintetizar

### Añadir gramática

### Línea de proceso de Reconocimiento

## Diagrama Diseño

# 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 SKD 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 archivos alojados en la nube”, “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 interactuacion con el reproductor de windows media (por lo que necesitaremos tenerlo instalado).

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

## Objetivos

* 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 esos archivos.

## Casos de usos relevantes

### Cargar librería de música

## Diagrama de diseño

# Módulo Camera

[INTRODUCCION]

## OpenCV / EmguCV

OpenCV o (Open Computer Vision) 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.

Solo tenemos un problema, como hemos comentado OpenCV está desarrollado en C y C++ entonces... ¿Cómo podemos utilizar esta biblioteca en nuestro proyecto desarrollado en .NET? [HABLAR DE COMO ESTA HECHO .NET PARA EL INTERPRETADO DE CODIGO C y C++]. Por lo que concluimos que podemos ejecutar código C y C++ en la plataforma .NET gracias a sus funciones nativas. Y aquí es donde entra en juego EmguCV

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. En mi caso, para conocer conceptos básicos, utilice las siguientes referencias:

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

Para la ampliación de conceptos utilice 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.

## Objetivos

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

## Casos de usos relevantes

## Diagrama de diseño

# Módulo Utils

## Introducción

El módulo *utils* está pensado para añadir funcionalidad extra que no es posible que sea 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/hora actual, climatología o gestión de un simple sistema de alarmas.

## Objetivos

* 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 sobre climatología: temperatura, condición climatológica, humedad, dirección del viento…
* Administración de alarmas.

## Weather API

Para satisfacer el objetivo de información climatológica usamos 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.

### 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 de la que se desea obtener información climatológica. Esta definición se puede mediante el nombre de la ciudad o su código postal. 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 que en el que quieres que te devuelva el XML con este comando, 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 es:



Hemos visto que podemos adquirir de manera sencilla este XML, ahora solo queda parsearlo para tener la información necesaria, en mi caso todos esos problemas los trato 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 muchísima más documentación (google carece completamente de ella) y proporciona de información que no podemos obtener con la API de Google.

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

## Casos de uso relevantes

## Diagrama de diseño

# Capa de presentación

# Capa de Datos

# Anexos

REFERENCIAS WIKIPEDIA???

BIBLIOGRAFIA UTILIZADA???