UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Ingeniería Informática

Proyecto Fin de Carrera

Automatización y optimización en procesos de actuación y adquisición de datos con microcontroladores Atmel orientados a robótica educativa

David Mera Ortiz

Septiembre, 2017

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Ingeniería Informática

Proyecto Fin de Carrera

Automatización y optimización en procesos de actuación y adquisición de datos con microcontroladores Atmel orientados a robótica educativa

Autor: David Mera Ortiz

Director/es: José Moreno del Pozo y Francisco Andrés Hernández

Tribunal Calificador

Presidente: Antonio Manuel Silva Luengo

Secretario: Pedro Núñez Trujillo

Vocal: Pilar Bachiller Burgos

FECHA: 25 de Septiembre de 2017

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN 5
2. BASES DEL PROYECTO 7
   2.1. LEGUAJES ESPECÍFICOS DE DOMINIO (DSLS) 7
      2.1.1. PROBLEMAS QUE RESUELVE UN DSL 8
      2.1.2. VENTAJAS Y DESVENTAJAS DE LOS DSLS 10
      2.1.3. TIPOS DE DSLS 11
      2.1.4. ESTRUCTURA DE UN DSL 14
      2.1.5. GRAMÁTICA FORMAL 15
   2.2. ARDUINO 17
      2.2.1. HARDWARE ARDUINO 19
      2.2.2. SOFTWARE ARDUINO 29
3. TECNOLOGÍAS ESTUDIADAS PARA LA DETERMINACIÓN DEL PROBLEMA 31
   3.1. GENERACIÓN AUTOMÁTICA DE UN PARSER 31
      3.1.1. PYPARSING 31
   3.2. INTERFACES GRÁFICAS EN PYTHON 34
      3.2.1. TKINTER 34
      3.2.2. WXPYTHON 35
      3.2.3. PYGTK 36
      3.2.4. PYQT 36
   3.3. ARDUINO-SKETCH 0.2 37
4. ARQUITECTURA 39
   4.1. INTRODUCCIÓN 39
   4.2. DESARROLLO DEL DSL 40
   4.3. EJEMPLO FICHERO DE ENTRADA Y DE SALIDA 45
   4.4. CARGA DEL FICHERO DE SALIDA EN LA PLACA 45
5. DIAGRAMA DE CLASES 47
   5.1. DIAGRAMA DE CLASES DEL DSL 47
6. MANUAL DE USUARIO 48
7. CONCLUSIONES Y TRABAJOS FUTUROS 50
8. BIBLIOGRAFÍA 51
9. ÍNDICE DE ILUSTRACIONES 53
10. ÍNDICE DE TABLAS 54

11. ANEXO I: BUS I2C 55

1. INTRODUCCIÓN

La robótica educativa es un campo en auge en la actualidad. El mercado se encuentra inundado de una gran variedad de circuitos sensores y actuadores, que en la mayoría de los casos pueden acarrear una gran complejidad de uso, sobre todo para personas no expertas en la materia. Por ello, el siguiente Trabajo Fin de Carrera pretende desarrollar una aplicación que optimice y simplifique el uso de un abanico de sensores y actuadores del mercado, facilitando el uso de esta tecnología a personas que se están iniciando en la materia.

Los objetivos que se pretenden cumplir son los siguientes:

a) La integración automática de sensores y actuadores para su uso con placas de desarrollo Arduino. Para lo cual se diseñará una herramienta gráfica que facilite la generación de código DSL (Domain Specific Language o Lenguaje Específico de Dominio), para su posterior integración automática en la placa Arduino.
b) El código generado debe realizar el envío de los datos de los sensores serializados en formato json, a través de canales de comunicación establecidos en placas Arduino (puerto serie, ethernet, bluetooth, etc) así como la gestión de comandos para los actuadores.

Además, la aplicación supone un gran ahorro de tiempo, ya que se automatiza la puesta en marcha de los sensores y actuadores generando gran cantidad de código. Por ello, su uso también está recomendado para personas que trabajan habitualmente con dicha tecnología y no sólo para novatos. Se puede generar el código de forma automática y después hacer las modificaciones que consideren oportunas.

Por otro lado, la serialización de los datos de los sensores y el estado de los actuadores, a través de un canal de comunicación, permite comunicar nuestra placa arduino con otros dispositivos. Esto permite conformar un sistema en el que varias placas Arduino envían sus datos a una placa central (por ejemplo una Raspberry Pi). De esta forma centralizamos todos los datos en la Raspberry, que será la encargada de procesar y gestionar todos los datos recibidos.

Para la consecución de los objetivos descritos anteriormente se ha utilizado el lenguaje de programación Python, y se ha desarrollado una interfaz gráfica, haciendo uso del módulo Tkinter y un DSL haciendo uso del módulo Pyparsing. Además, se ha utilizado el módulo arduino-sketch para poder subir el fichero Arduino generado a una placa Arduino directamente desde la interfaz, sin tener que utilizar el IDE de Arduino.

2. BASES DEL PROYECTO

2.1. LEGUAJES ESPECÍFICOS DE DOMINIO (DSLS)

En el desarrollo de software, un lenguaje específico de dominio (DSL, Domain Specific Languages) es un lenguaje de programación dedicado a un problema de dominio en particular, o una técnica de representación o resolución de problemas específica. Así pues, un DSL es un lenguaje ideado para expresar cierta parte de un sistema. Por eso se dice que es un lenguaje de propósito específico, a diferencia de los lenguajes de propósito general (GPL).

Los lenguajes con los que estamos acostumbrados a trabajar, como Java, C o Groovy tienen la característica de que pueden ser utilizados para resolver problemas de programación de cualquier índole, por lo que decimos que son lenguajes de propósito general (GPL por sus siglas en inglés).

Estos lenguajes son Turing-completos, siendo capaces de expresar cualquier algoritmo, y pudiendo ser aplicados a cualquier dominio. Con estos lenguajes, con mayor o menor facilidad o eficiencia, podemos construir sistemas de cálculo impositivo, implementar algoritmos de aprendizaje de máquina, etc. Es decir, son lenguajes que pretenden ser igualmente efectivos (igualmente buenos o malos) en todos los campos de acción.

Sin embargo, el empleo de GPLs para expresar problemas muy específicos, si bien es posible, puede significar mayor esfuerzo de lo que uno desearía, dado que las abstracciones que estos dominios plantean no están soportadas nativamente por el lenguaje; no son primitivas. Por ejemplo, escribir una transposición de matrices, incluso contando con una API (Interfaz de Programación de Aplicaciones) bien diseñada, es ciertamente más complejo en Java que en Mathematica y lenguajes como SQL están específicamente diseñados para realizar operaciones sobre una base datos.

Estos lenguajes son específicos de un dominio particular, y si bien no pueden resolver todos los problemas, resuelven aquellos para los que fueron diseñados mejor que los GPL.

Suelen ser lenguajes muy sencillos, que pueden ser escritos y leídos por personas que conocen el dominio del problema, pero que no tienen que tener conocimientos de programación.

En la siguiente tabla se pueden apreciar las principales diferencias ente un DSL y un GPL:

	Lenguaje de Propósito General	Lenguaje Específico de Dominio
Abarca	La totalidad de la aplicación	Una parte de la aplicación
Aplicación	Cualquiera	Un solo tipo
Turing completo	Sí	No tiene por qué
Lenguaje	Complejo	Sencillo

Tabla 1 - Comparación GPL y DSL

2.1.1. PROBLEMAS QUE RESUELVE UN DSL

En general cuando nos enfrentamos a un problema de programación aparecen varias actividades que tenemos que realizar:

1. Entendimiento del problema/dominio
2. Formación conceptual de una solución
3. Implementación en un lenguaje de programación
4. Compilación, ejecución y pruebas.

Normalmente trabajamos con un único paradigma de programación y unos pocos lenguajes. Estos lenguajes permiten expresar sobre ellos cualquier tipo de dominio, es decir, que se utilizan para desarrollar cualquier tipo de aplicación. Por esto se llaman lenguajes de propósito general.

En el proceso de entendimiento del dominio hay que abstraerse del lenguaje de programación, teniendo en cuenta sólo la información y los requerimientos. Hay que realizar un análisis del problema independiente del lenguaje de programación, e incluso del paradigma.

Durante la formación conceptual de una solución ya debemos pensar el dominio dentro de un paradigma, de acuerdo a nuestra experiencia, o lo que veamos que mejor se adapte a la problemática. Por ejemplo, hay problemas que son inherentemente más fáciles de implementar en el paradigma lógico que en POO (Programación Orientada a Objetos), o funcional, etc. Sin embargo, todavía podríamos pensar en una solución independiente del lenguaje.

A continuación, tenemos que implementar esta solución abstracta en los pasos 3 y 4, y para eso utilizamos un GPL, y aquí está el problema, no suele ser trivial. A veces, no tenemos soporte del lenguaje para ciertas abstracciones de nuestro dominio, como puede ser el patrón singleton. Por ello, estamos forzados a adaptar el dominio y nuestra solución al lenguaje, y eso es lo que hacemos siempre, adaptamos al lenguaje que tenemos "a mano". Eso nos lleva a que nuestra solución va a estar siempre de aquí en adelante expresada en un lenguaje que no es el más cercano al dominio del problema, sino más bien a un lenguaje de programación general. Esta situación lleva asociada las siguientes consecuencias:

- **Legibilidad**

El código contendrá una mezcla entre conceptos de dominio (una Cuenta, un Cliente, etc) y palabras propias del lenguajes (keywords, como public class, etc). Quien se incorpore al proyecto o quiera revisar la solución deberá, naturalmente, hacer el proceso inverso, como una ingeniería reversa, a partir del código y de lo expresado en el GPL abstraerse para generar una representación mental del problema/dominio.

- **Flexibilidad (cambios de requerimientos o dominio)**

Naturalmente solo podrán ser implementados por desarrolladores que entiendan no sólo el dominio, sino además del GPL. Cada nuevo cambio deberá ser traducido nuevamente de dominio hacia el GPL.

- **Duplicación**

Tendremos al menos dos representaciones de la solución, la mental (que muchas veces además se plasma en documentos de especificación y análisis) y la traducción/implementación en el GPL.

Estas consecuencias hacen que la programación dedique la mayor parte del tiempo y esfuerzo en lidiar con los problemas de traducción e implementación de la solución al GPL.

Entonces, una vía alternativa sería en pensar que en lugar de adaptar nuestra solución a un lenguaje, podemos adaptar el lenguaje a nuestra solución. A esto se lo conoce como Language-Oriented Programming, desde el punto de vista de un nuevo "paradigma". Y una de las prácticas es crear un nuevo lenguaje para expresar nuestra solución o una parte de la solución. Esto es un DSL.

2.1.2. VENTAJAS Y DESVENTAJAS DE LOS DSLS

Teniendo en cuenta todo lo descrito anteriormente, las razones para utilizar un DSL son numerosas. A continuación, se resumen las ventajas y los inconvenientes de utilizar un DSL.

Ventajas

- El lenguaje es más legible y más corto.
- No necesita ser Turing completo.
- Puede ser usado por personal no técnico.
- Generación de código más sencilla.
- Flexible y adaptable.
- Es código: podemos transformarlo, almacenarlo, distribuirlo, representarlo, etc.

Desventajas

- Alto coste de diseño, implementación y mantenimiento.
- Difícil ajustar el dominio.
- Código menos eficiente.
- Más difícil de depurar.

- Dificultad para sopesar las ventajas y desventajas entre las construcciones de los DSL y de los lenguajes de propósito general.

¿Cuándo necesito un DSL? Como aproximación, podemos afirmar que si la cantidad de problemas en este domino específico que queremos resolver es pequeño, o la complejidad de crear el DSL es mayor que la resolver el problema en nuestro GPL, probablemente no lo necesitemos. De lo contrario, será una alternativa a considerar.

2.1.3. TIPOS DE DSLS

Existen una gran variedad de DSLs con dominios de aplicación muy diversos. Entre ellos podemos destacar:

- **SQL**: permite extraer información de un sistema de bases de datos relacional.
- **Expresiones regulares:** permite buscar y extraer patrones de texto dentro de otros textos.
- **CSS:** para especificar la apariencia gráfica de elementos HTML (No es un lenguaje de programación, y sus usuarios no son necesariamente desarrolladores)
- **Otros ejemplos:** HTML, Django, XML, JSON, Yacc, Lex, etc.

A grandes rasgos, los DSLs se pueden clasificar a través de las siguientes categorías. Existen autores que refinan mucho más a detalle la categorización, incluyendo nuevos tipos. Pero con el fin de explicar la idea general a nivel de programación, la clasificación queda acotada de la siguiente manera:

1) Parser + Compilador o Interprete

- *<u>Compilador:</u>* traduce a lenguaje maquina ejecutable y puede ser ensamblador o un lenguaje ejecutable por una Máquina Virtual como java, etc.
- *<u>Interprete:</u>* a medida que se va parseando (leyendo) el código expresado en el DSL, se va ejecutando, sin haber "generado" código ejecutable como paso intermedio.

2) Traductor

Los traductores al igual que los compiladores, se utilizan para generan código, la diferencia es que los traductores generan código que no es “ejecutable” de por sí, sino más bien código de otro lenguaje. Ejemplo: generadores de código a través de Java APT (annotation processing tool).

3) Embebidos:

Utiliza un lenguaje GPL, pero lo hace de tal manera que simula o se asemeja en gran medida a un lenguaje propio del dominio, Aprovecha las características del lenguaje GPL existente y evita tener que hacer un parser, compilador e interprete. Un ejemplo claro de DSL embebido es Ruby.

4) DSL Interno

Los DSLs internos son escritos en un lenguaje de programación de propósito general, cuya sintaxis se ha inclinado a parecerse más al lenguaje natural. Es más fácil para los lenguajes que ofrecen azúcar sintáctica[1] y posibilidades de formato, como Ruby o Scala, que para otros lenguajes que no lo hacen, como Java. Muchos DSL internos envuelven API existentes, bibliotecas o código de negocio para proveer un contenedor con un acceso más eficiente a sus funcionalidades. Dependiendo de la implementación y el dominio, son usados para construir estructuras de datos, definir dependencias, ejecutar procesos o tareas, comunicarse con otros sistemas o validar entradas de usuario. La sintaxis de un DSL interno está contenida en el lenguaje anfitrión. Hay muchos patrones, por ejemplo, constructores de expresiones, encadenadores de métodos y anotaciones, que pueden ayudar a adaptar el lenguaje anfitrión al DSL. Si el lenguaje anfitrión no requiere recompilación, entonces un DSL interno puede ser desarrollado rápidamente trabajando conjuntamente con los expertos del dominio.

---

[1] **Azúcar sintáctica o sugar syntax:** Término acuñado por Peter J. Landin en 1964 para referirse a los añadidos a la sintaxis de un lenguaje de programación diseñados para hacer algunas construcciones más fáciles de leer o expresar.

5) DSL Externo

Los DSLs externos son expresiones gráficas o textuales de un lenguaje, aunque los DSLs textuales tienden a ser más comunes que los gráficos. Las expresiones textuales pueden ser procesadas por una cadena de herramientas que incluyen léxico, un analizador, un transformador de modelo, generadores, y cualquier otro tipo de posprocesamiento. Los DSLs externos son frecuentemente leídos en modelos internos, los cuales forman los fundamentos para su posterior procesamiento. Es útil definir una gramática BNF[2]. Una gramática provee un punto de partida para la generación de partes de la cadena de herramientas (por ejemplo, editor, visualizador, generador de analizadores). Para los DSL sencillos, un analizador hecho a mano podría ser suficiente; usando, por ejemplo, expresiones regulares. Los analizadores personalizados pueden llegar a ser difíciles de manejar si se espera mucho de ellos, así que tiene sentido mirar las herramientas diseñadas específicamente para trabajar con gramáticas del lenguaje. Es también común el definir DSL externos como los dialectos XML, aunque la legibilidad es frecuentemente un problema; sobre todo para los lectores no técnicos.

	DSL Interno	DSL Externo
Ventajas	✓ Permite trabajar a través de un IDE (Entorno de Desarrollo integrado) ✓ No necesita leer y parsear los datos	✓ Mayor expresividad del dominio ✓ Libertad sobre la sintaxis del lenguaje (sólo limitada por la capacidad de implementación del parser) ✓ Puede ser editado fácilmente por usuarios finales
Desventajas	✗ No puede ser utilizado por personas sin conocimientos de programación ✗ Puede resultar difícil de implementar ✗ Muy complicado de depurar	✗ Complejidad al tener que implementar el parser y el compiler ✗ Trabajar con ficheros de texto en lugar de un IDE (Entorno de Desarrollo Integrado)

Tabla 2 - Comparación de DSL interno y externo

[2] **BNF:** la notación de Backus-Naur, también conocida por sus denominaciones inglesas Backus-Naur form (BNF), Backus-Naur formalism o Backus normal form, es un metalenguaje usado para expresar gramáticas libres de contexto: es decir, una manera formal de describir lenguajes formales.

Si comparamos los DSLs internos y externos, en general los segundos serán más flexibles, pero también el esfuerzo de implementarlos será mayor, no sólo porque la complejidad de implementar un parser a mano es mayor, sino que probablemente el lenguaje huésped nos proveerá muchas construcciones y bibliotecas útiles. Aprovechar las bibliotecas existentes para otro lenguaje es un punto muy importante también a la hora de diseñar un GPL. Así es como surge gran cantidad de lenguajes que corren sobre la máquina virtual de Java o .Net).

2.1.4. ESTRUCTURA DE UN DSL

La estructura de un DSL consta principalmente de cuatro fases, que pueden recibir distintos nombres dependiendo del autor que las cite, pero básicamente son las que se detallan a continuación. Cada capa o fase usa los servicios de la anterior y pasa sus resultados a la siguiente

1) Analizador léxico

El analizador léxico, también llamado Tokenizer o Lexer, lee una secuencia de bytes y devuelve un conjunto de unidades léxicas o tokens. No es complicado de implementar y la forma más usual de hacerlo es utilizando expresiones regulares. De esta forma, un analizador léxico recibe como entrada un fichero de texto y lo recorre secuencialmente buscando cadenas de caracteres que encajen con las expresiones regulares que han sido definidas previamente. En caso de que la encuentre lleva a cabo las acciones asociadas a dicha expresión regular. Puede generar un fichero de texto como salida y en el caso de que el analizador léxico forme parte de un compilador o de un intérprete, la entrada será el programa fuente que debe ser traducido y la salida será la entrada del analizador sintáctico y estará formada por una secuencia de tokens (símbolos terminales de la gramática)

2) Analizador sintáctico o Parser

El analizador sintáctico, también llamado parser, recibe como entrada los tokens que genera el analizador léxico y comprueba si estos tokens van llegando en el orden correcto. Siempre que no se hayan producido errores, la salida teórica de esta fase del compilador será un árbol sintáctico. Si el programa es incorrecto se generarán los mensajes de error correspondientes.

3) Interprete, render o generador de código

Es la última fase, haciendo uso del árbol sintáctico, genera un programa ejecutable o un programa objeto escrito en lenguaje intermedio.

2.1.5. GRAMÁTICA FORMAL

Una gramática formal define matemáticamente a un lenguaje y permite saber si un determinado texto pertenece al lenguaje o no. Para ello, establece unas reglas de producción; si usando dichas reglas se puede obtener un texto, dicho texto pertenece al lenguaje. La gramática es utilizada por los analizadores léxicos para detectar si la entrada es correcta, es decir, pertenece al lenguaje. Organiza los tokens en forma de árbol sintáctico o árbol de sintaxis abstracta (AST). Se le denomina abstracto, porque no define fielmente todos los detalles de la sintaxis.

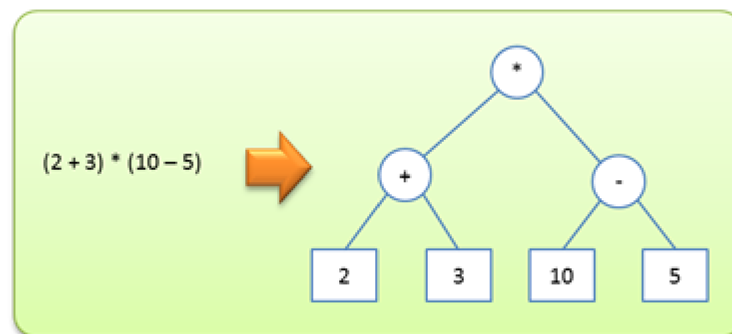

Ilustración I - Ejemplo de Árbol sintáctico

Desde el punto de vista formal, una gramática es un conjunto de reglas llamadas producciones. De esas reglas una debe ser la inicial y tienen la forma que se detalla a continuación, y se puede interpretar como parte derecha se define como parte izquierda.

$$< Parte\ izquierda > \rightarrow < Parte\ derecha >$$

A continuación tenemos un ejemplo de gramática que define las operaciones de suma y multiplicación:

< asignacion > → < variable > = < expresion >

< expresion > → < numero >

< expresion > → < variable >

< expresion > → < expresion > + < expresion >

< expresion > → < expresion > ∗ < expresion >

Cualquier símbolo que aparezca a la izquierda de una flecha es un símbolo No Terminal, mientras que los que no aparecen a la izquierda se denominan símbolos Terminales o Literales. En la parte derecha pueden aparecer tanto símbolos terminales como no terminales.

Para la definición de las reglas se utilizan símbolos especiales, que se detallan en la siguiente tabla.

Símbolo	Utilidad
|	Representa alternativas, a|b significa que se puede generar a o b
*	Repetición de cero o más
+	Repetición de uno o más
?	Opcional (uno o cero)
[a-z]	Rango de caracteres
[0-9]	Se podría representar como 0|1|2|3|4|5|6|7|8|9

Tabla 3 - Símbolos especiales de una gramática

2.2. ARDUINO

Arduino es una compañía de hardware libre y una comunidad tecnológica que diseña y manufactura placas de desarrollo de hardware, compuestas por Microcontroladores, elementos pasivos y activos. Por otro lado, las placas son programadas a través de un entorno de desarrollo (IDE), el cuál compila el código al modelo seleccionado de placa.

Arduino se puede utilizar para desarrollar elementos autónomos, conectándose a dispositivos e interactuar tanto con el hardware como con el software. Nos sirve tanto para controlar un elemento, pongamos por ejemplo un motor que nos suba o baje una persiana basada en la luz existente es una habitación, gracias a un sensor de luz conectado al Arduino, o bien para leer la información de una fuente, como puede ser un teclado, y convertir la información en una acción como puede ser encender una luz y pasar por un display lo tecleado.

Arduino se enfoca en acercar y facilitar el uso de la electrónica y programación de sistemas embebidos en proyectos multidisciplinarios. Toda la plataforma, incluyendo sus componentes de hardware (esquemáticos) y Software, son liberados con licencia de código abierto que permite libertad de acceso a los mismos.

El hardware consiste en una placa de circuito impreso con un microcontrolador, usualmente Atmel AVR, puertos digitales y analógicos de entrada/salida, los cuales pueden conectarse a placas de expansión (shields), que amplían los funcionamientos de la placa Arduino. Asimismo, posee un puerto de conexión USB desde donde se puede alimentar la placa y establecer comunicación con el computador.

Por otro lado, el software consiste en un entorno de desarrollo (IDE) basado en el entorno de processing y lenguaje de programación basado en Wiring, así como en el cargador de arranque (bootloader) que es ejecutado en la placa. El microcontrolador de la placa se programa mediante un computador, usando una comunicación serial mediante un convertidor de niveles RS-232 a TTL serial.

La primera placa Arduino fue introducida en 2005, ofreciendo un bajo costo y facilidad de uso para novatos y profesionales. Buscaba desarrollar proyectos

interactivos con su entorno mediante el uso de actuadores y sensores. A partir de octubre de 2012, se incorporaron nuevos modelos de placas de desarrollo que usan microcontroladores Cortex M3, ARM de 32 bits, que coexisten con los originales modelos que integran microcontroladores AVR de 8 bits. ARM y AVR no son plataformas compatibles en cuanto a su arquitectura y por lo que tampoco lo es su set de instrucciones, pero se pueden programar y compilar bajo el IDE predeterminado de Arduino sin ningún cambio.

Las placas Arduino están disponibles de dos formas: ensambladas o en forma de kits "Hazlo tú mismo" (por sus siglas en inglés "DIY"). Los esquemas de diseño del Hardware están disponibles bajo licencia Libre, con lo que se permite que cualquier persona pueda crear su propia placa Arduino sin necesidad de comprar una prefabricada. Adafruit Industries estimó a mediados del año 2011 que, alrededor de 300.000 placas Arduino habían sido producidas comercialmente y en el año 2013 estimó que alrededor de 700.000 placas oficiales de la empresa Arduino estaban en manos de los usuarios.

Arduino se puede utilizar para desarrollar objetos interactivos autónomos o puede ser conectado a software tal como Adobe Flash, Processing, Max/MSP, Pure Data, etc. Una tendencia tecnológica es utilizar Arduino como tarjeta de adquisición de datos desarrollando interfaces en software como JAVA, Visual Basic y LabVIEW. Las placas se pueden montar a mano o adquirirse. El entorno de desarrollo integrado libre se puede descargar gratuitamente.

Arduino como herramienta educativa es muy útil y efectiva. Y es que existe un factor muy importante a tener en cuenta. Gran parte del éxito de Arduino, se debe a la comunidad que apoya todo este desarrollo, comparte conocimiento, elabora librerías para facilitar el uso de Arduino y publica sus proyectos para que puedan ser replicados, mejorados o ser base para otro proyecto relacionado.

En resumen:

$$\boldsymbol{Arduino = HW + SW + Comunidad}$$

2.2.1. HARDWARE ARDUINO

El HW de Arduino es básicamente una placa con un microcontrolador. Un microcontrolador (abreviado µC, UC o MCU) es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica. Un microcontrolador incluye en su interior las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida.

Existen MCUs genéricos y otros de propósito especial como los DSP (Procesador Digital de Señal), para aplicaciones de voz y video por ejemplo.

Microcontroladores AVR

Los microcontroladores AVR son una familia de microcontroladores RISC del fabricante estadounidense Atmel. Las placas Arduino usan principalmente microcontroladores de la gama AVR de Atmel, pero también de otras gamas de Atmel y de Intel.

Es importante saber que Arduino no es el único microcontrolador ni la única plataforma. Lo mismo que hacemos con Arduino se puede hacer con otros microcontroladores y otras plataformas de desarrollo, pero Arduino es la más extendida, con más documentación y soporte de la comunidad.

Microcontroladores ARM de Atmel

Los nuevos Arduinos ya no son microcontroladores Atmel AVR de 8-bit como hasta ahora eran casi todos, sino que ha apostado por la nueva gama de Atmel con arquitectura ARM.

Los Arduino y en general los microcontroladores tienen puertos de entrada y salida y de comunicación. En Arduino podemos acceder a esos puertos a través de los pines.

- **Pines digitales:** pueden configurarse como entrada (para leer, sensores) o como salida (para escribir, actuadores)

- **Pines analógicos de entrada:** usan un conversor analógico/digital y sirven para leer sensores analógicos como sondas de temperatura.
- **Pines analógicos de salida (PWM):** la mayoría de Arduino no tienen conversor digital/analógico y para tener salidas analógicas se usa la técnica PWM. No todos los pines digitales soportan PWM.
- **Puertos de comunicación:** USB, serie, I2C y SPI.

Otro aspecto importante es la memoria, Arduino tiene tres tipos de memoria:

- **SRAM**: donde Arduino crea y manipula las variables cuando se ejecuta. Es un recurso limitado y debemos supervisar su uso para evitar agotarlo.
- **EEPROM**: memoria no volátil para mantener datos después de un reset o apagado. Las EEPROMs tienen un número limitado de lecturas/escrituras, tener en cuenta a la hora de usarla.
- **Flash**: Memoria de programa. Usualmente desde 1 Kb a 4 Mb (controladores de familias grandes). Donde se guarda el sketch.

2.2.1.1. PLACAS ARDUINO

Como ocurre con las distribuciones Linux, Arduino también cuenta con multitud de ediciones, cada una pensada para un público concreto o para una serie de tareas específicas. Existen tal variedad de modelos oficiales, no oficiales y compatibles que es normal que la gente no sepa diferenciar con exactitud las características de cada una de estas maravillosas placas.

- **Oficiales:** son aquellas placas manufacturadas por la compañía italiana Smart Projects y algunas han sido diseñadas por la empresa estadounidense SparkFun Electronics (SFE) o por la también estadounidense Gravitech. Incluso el gigante Intel ha colaborado en el diseño de una de estas placas… Arduino Pro, Pro Mini y LilyPad son las manufacturadas por SFE y Arduino Nano por Gravitech, el resto se crean en Italia. Estas placas son las reconocidas oficialmente, incluyen el logo y son las únicas que pueden llevar la marca registrada de Arduino.
- **No oficiales o compatibles:** son placas compatibles con Arduino pero no pueden estar registradas bajo el nombre de Arduino. Por supuesto son diseñadas y fabricadas por otras compañías ajenas. Estos desarrollos no aportan nada al

desarrollo propio de Arduino, sino que son derivados que han salido para cubrir otras necesidades. Estas frecuentemente utilizan un nombre que integra el sufijo “duino” para identificarlas, como por ejemplo Freeduino del que ya hablaremos.

A la hora de seleccionar la placa para nuestro proyecto tenemos que tener esto muy presente para no llevarnos sorpresas. Puede que nos interese una placa compatible por ciertas cualidades del hardware que no posee Arduino o por cuestiones de licencias y sin embargo querer que sea compatible con el entorno de desarrollo Arduino IDE. En otras ocasiones puede que simplemente se desee compatibilidad en cuanto a los shields pero se tiene la necesidad de emplear otro compilador (AVR Studio, Makefiles,…).

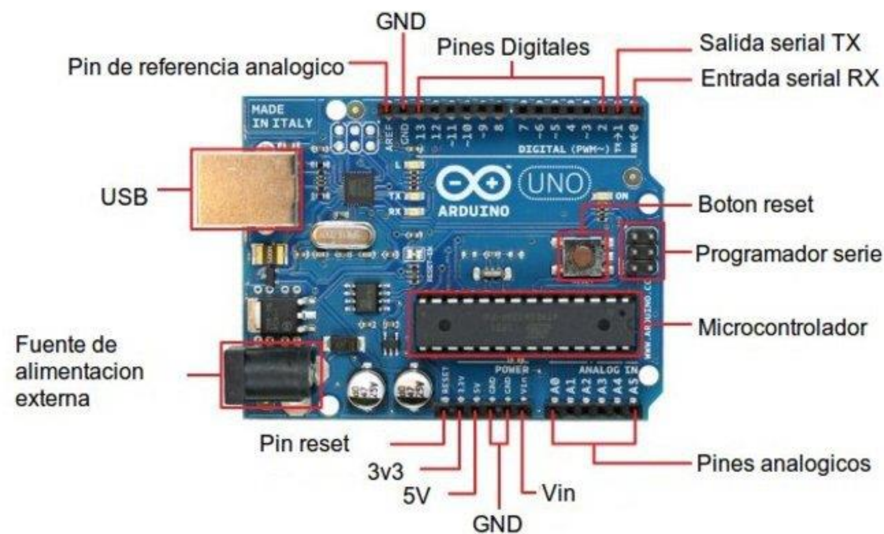

Ilustración II -Características generales placa Arduino

Lo principal que debemos saber es que tipo de proyectos vamos a implementar. Con esto nos da una idea de la cantidad de pines analógicos y digitales (normales y de tipo PWM o modulados por ancho de pulso para simular una salida analógica) que necesitamos para nuestro trabajo. Este primer escrutinio nos permite descartar algunas placas más simples que no tengan suficientes pines o, al contrario, descartar las de mayor número de ellos para reducir los costes puesto que con menos pines nos conformamos.

También podemos deducir el tamaño de código que vamos a generar para nuestros sketchs. Un programa muy largo, con muchas constantes y variables demandará una cantidad mayor de memoria flash para su almacenamiento, por lo que se debe elegir una placa adecuada para no quedarnos cortos.

La RAM será la encargada de cargar los datos para su inmediato procesamiento, pero no es uno de los mayores escollos, puesto que esto solo afectaría a la velocidad de procesamiento. La RAM va ligada al microcontrolador, puesto que ambos afectan a la agilidad de procesamiento de Arduino.

En los Arduinos oficiales podemos diferenciar entre dos tipos fundamentales de microcontroladores, los de 8 y 32 bits basados en ATmega AVR y los SMART basados en ARM de 32 bits y con un rendimiento superior, ambos creados por la compañía Atmel. En principio no debes guiarte por tu deseo de tener un chip de 32 bits, puesto que para la mayoría de proyectos que implementamos uno de 8 bits basta.

Por último, en cuanto al voltaje, no importan demasiado a nivel electrónico, excepto en algunos casos, para tener en cuenta la cantidad de tensión que la placa puede manejar para montar nuestros circuitos. Esto no supone mayor problema, puesto que una placa de Arduino podría trabajar incluso con tensiones de 220v en alterna con el uso por ejemplo de relés. Pero cuando queremos prescindir de una fuente de alimentación externa, hay que tener en cuenta que este es el voltaje que se puede manejar. Y entre otras cosas marcar el límite para no destruir la placa con sobretensiones no soportadas. Pero no confundas el voltaje al que trabaja el microcontrolador y al que funcionan los periféricos de la placa.

De todas formas, la placa más vendida y que es la más aconsejable para la mayoría de proyectos, sobre todo si estás empezando, es la Arduino UNO. Es suficiente para la mayoría de proyectos, tiene un buen precio y dispone de unos parámetros equilibrados.

Arduino UNO

Es la plataforma más extendida y la primera que salió al mercado, por ello nos podemos basar en esta para hacer la comparativa con el resto de placas. Todas las características de esta placa estarán implementadas en casi todas las placas restantes, a excepción de algunas que ya veremos. Se basa en un microcontrolador Atmel ATmega320 de 8 bits a 16Mhz que funciona a 5v. 32KB son correspondientes a la memoria flash (0,5KB reservados para el bootloader), 2KB de SRAM y 1KB de EEPROM. En cuanto a memoria es una de las placas más limitadas, pero no por ello resulta insuficiente para casi todos los proyectos que rondan la red. Las salidas pueden trabajar a voltajes superiores, de entre 6 y 20v pero se recomienda una tensión de trabajo de entre 7 y 12v. Contiene 14 pines digitales, 6 de ellos se pueden emplear como PWM. En cuanto a pines analógicos se cuenta con hasta 6. Estos pines pueden trabajar con intensidades de corriente de hasta 40mA.

Arduino Mega

Su nombre proviene del microcontrolador que lo maneja, un ATmega2560. Este chip trabaja a 16Mhz y con un voltaje de 5v. Sus capacidades son superiores al ATmega320 del Arduino UNO, aunque no tan superiores como las soluciones basadas en ARM. Este microcontrolador de 8 bits trabaja conjuntamente con una SRAM de 8KB, 4KB de EEPROM y 256KB de flash (8KB para el bootloader). Como puedes apreciar, las facultades de esta placa se asemejan al Due, pero basadas en arquitectura AVR en vez de ARM. En cuanto a características electrónicas es bastante similar a los anteriores, sobre todo al UNO. Pero como se puede apreciar a simple vista, el número de pines es parecido al Arduino Due: 54 pines digitales (15 de ellos PWM) y 16 pines analógicos. Esta placa es idónea para quien necesita más pines y potencia de la que aporta UNO, pero el rendimiento necesario no hace necesario acudir a los ARM-based.

Arduino Yun

Se basa en el microcontrolador ATmega32u4 y en un chip Atheros AR9331 (que controla el host USB, el puerto para micro-SD y la red Ethernet/WiFi), ambos comunicados mediante un puente. El procesador Atheros soporta la distribución Linux

basadas en OpenWrt llamada OpenWrt-Yun. Se trata de una placa similar a Arduino UNO pero con capacidades nativas para conexión Ethernet, WiFi, USB y micro-SD sin necesidad de agregar o comprar shields aparte. Contiene 20 pines digitales, 7 pueden ser usados en modo PWM y 12 como analógicos. El microcontrolador ATmega32u4 de 16Mhz trabaja a 5v y contiene una memoria de solo 32KB (4KB reservados al bootloader), SRAM de solo 2,5KB y 1KB de EEPROM. Como vemos, en este sentido queda corto. Sin embargo se complementa con el AR9331 a 400Mhz basado en MIPS y trabajando a 3v3. Este chip además contiene RAM DDR2 de 64MB y 16MB flash para un sistema Linux embebido.

Otras placas

Arduino TRE, Arduino/Genuino 101, Arduino Zero, Arduino Zero Pro, Arduino Leonardo, Arduino Due, Arduino Ethernet, Arduino Fio, Arduino Nano, Arduino Pro, etc.

2.2.1.2. SENSORES Y ACTUADORES

SENSORES

Un sensor es un dispositivo capaz de detectar magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas en variables eléctricas. Las variables de instrumentación pueden ser por ejemplo: temperatura, intensidad lumínica, distancia, aceleración, inclinación, desplazamiento, presión, fuerza, torsión, humedad, movimiento, pH, etc. Una magnitud eléctrica puede ser una resistencia eléctrica (como en una RTD), una capacidad eléctrica (como en un sensor de humedad o un sensor capacitivo), una tensión eléctrica (como en un termopar), una corriente eléctrica (como en un fototransistor), etc.

Los sensores se pueden clasificar en función de los datos de salida en:

- Digitales
- Analógicos

Y dentro de los sensores digitales, estos nos pueden dar una señal digital simple con dos estados como una salida de contacto libre de tensión o una salida en bus digital.

A la hora de elegir un sensor para Arduino debemos tener en cuenta los valores que puede leer las entradas analógicas o digitales de la placa para poder conectarlo o sino adaptar la señal del sensor a los valores que acepta Arduino.

Una vez comprobado que el sensor es compatible con las entradas de Arduino, hay que verificar cómo leer el sensor mediante la programación, comprobar si existe una librería o es posible leerlo con los métodos disponibles de lectura de entrada analógica o digital.

Por último verificar cómo alimentar el sensor y comprobar si podemos hacerlo desde el propio Arduino o necesitamos una fuente exterior. Además, en función del número de sensores que queramos conectar es posible que Arduino no pueda alimentar todos.

Características de los sensores

- **Rango de medida:** dominio en la magnitud medida en el que puede aplicarse el sensor.
- **Precisión:** es el error de medida máximo esperado.
- **Offset o desviación de cero:** valor de la variable de salida cuando la variable de entrada es nula. Si el rango de medida no llega a valores nulos de la variable de entrada, habitualmente se establece otro punto de referencia para definir el offset.
- **Linealidad o correlación lineal.**
- **Sensibilidad de un sensor:** suponiendo que es de entrada a salida y la variación de la magnitud de entrada.
- **Resolución:** mínima variación de la magnitud de entrada que puede detectarse a la salida.
- **Rapidez de respuesta:** puede ser un tiempo fijo o depender de cuánto varíe la magnitud a medir. Depende de la capacidad del sistema para seguir las variaciones de la magnitud de entrada.
- **Derivas:** son otras magnitudes, aparte de la medida como magnitud de entrada, que influyen en la variable de salida. Por ejemplo, pueden ser condiciones ambientales, como la humedad, la temperatura u otras como el envejecimiento (oxidación, desgaste, etc.) del sensor.

- **Repetitividad:** error esperado al repetir varias veces la misma medida.

Ejemplos Sensores para Arduino

- Sensor ultrasónico de distancia
- Sensor de corriente
- Sensor de temperatura y humedad
- Sensor de presión barométrica y altura
- Caudalimetro
- Sensor láser distancia
- Sensor de obstáculos infrarrojo
- Sensor infrarrojo para calcular distancias
- Fotosensor LDR
- Sensor 4 en 1: temperatura+presión+altitud+luz comunicado por I2C.

Sensores con comunicación por bus.

Un bus (o canal) es un sistema digital que transfiere datos entre los componentes de un dispositivo electrónico o entre varios. Está formado por cables o pistas en un circuito impreso, dispositivos como resistencias y condensadores además de circuitos integrados.

La tendencia en los últimos años hacia el uso de buses seriales como el USB, Firewire, etc… para comunicaciones con periféricos, reemplazando los buses paralelos, a pesar de que el bus serial posee una lógica compleja (requiriendo mayor poder de cómputo que el bus paralelo) se produce a cambio de velocidades y eficacias mayores.

Arduino dispone de buses serie I2C (para más información consultar ANEXO I) y SPI para comunicarse con dispositivos sin necesidad de HW adicional

Existen muchos tipos de buses de comunicaciones, algunos de ellos los implementa arduino mediante controladores HW integrados en la MCU (I2C) o mediante una librería como “Wire”. En otros casos es necesario un hardware adicional para adaptar la señal con un transceiver y manejar el protocolo con un controlador, por ejemplo can bus o modbus.

ACTUADORES O PERIFÉRICOS

Un actuador es un dispositivo capaz de transformar energía hidráulica, neumática o eléctrica en la activación de un proceso con la finalidad de generar un efecto sobre elemento externo. Este recibe la orden de un regulador, controlador o en nuestro caso un Arduino y en función a ella genera la orden para activar un elemento final de control como, por ejemplo, una válvula.

Existen varios tipos de actuadores como son:

- Electrónicos
- Hidráulicos
- Neumáticos
- Eléctricos
- Motores
- Bombas

Periférico es la denominación genérica para designar al aparato o dispositivo auxiliar e independiente conectado a la unidad central de procesamiento o en este caso a Arduino. Se consideran periféricos a las unidades o dispositivos de hardware a través de los cuales Arduino se comunica con el exterior, y también a los sistemas que almacenan o archivan la información, sirviendo de memoria auxiliar de la memoria principal. Ejemplos de periféricos:

- Pantallas LCD
- Teclados
- Memorias externas
- Cámaras
- Micrófonos
- Impresoras
- Pantalla táctil
- Displays numéricos
- Zumbadores
- Indicadores luminosos, etc…

Para cada actuador o periférico, necesitamos un “driver” o manejador para poder mandar órdenes desde Arduino.

Hay que tener en cuenta que los pines de Arduino sólo pueden manejar un máximo de 40mA y recomendable usar 20mA de forma continua y un total de 200 mA de salida. Es decir que la corriente máxima que admite Vcc y GND son 200 mA.

Además, los pines Arduino sólo pueden tener los valores de 5V (3.3V en algunos modelos) y 0V. No es posible cualquier otro valor de tensión.

La alimentación máxima del pin de 5V y del pin de 3.3V dependerá del regulador de tensión que tenga la placa, en el caso de Arduino UNO la limitación es 1 A para 5V y 150 mA para 3.3V

A la hora de seleccionar un actuador o periférico para usar con arduino habrá que ver sus características y cómo hacer el interface con arduino.

Ejemplos de Actuadores y Periféricos para Arduino

- Relés
- Actuadores lineales
- Displays
- Dimmer AC (regulador de intensidad)
- Motores
- Servos
- LEDS

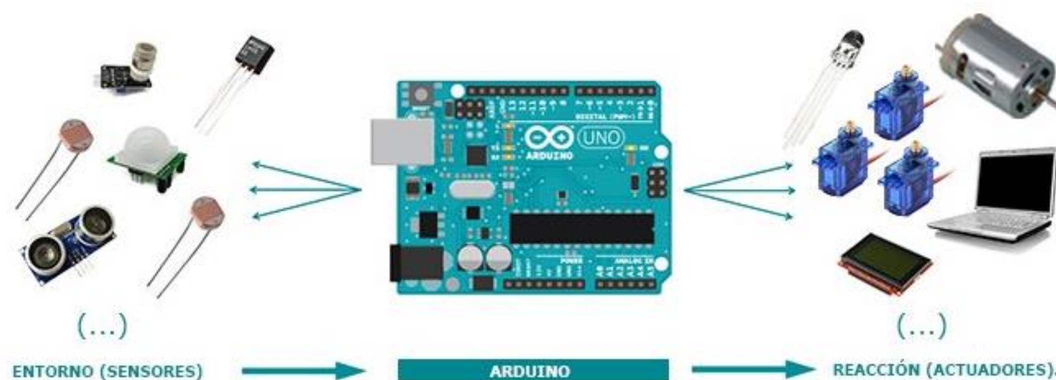

Ilustración III – Esquema sensores y actuadores

2.2.2. SOFTWARE ARDUINO

El software de Arduino es un IDE, entorno de desarrollo integrado (siglas en inglés de Integrated Development Environment). Es un programa informático compuesto por un conjunto de herramientas de programación.

El IDE de Arduino es un entorno de programación que ha sido empaquetado como un programa de aplicación; es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Además incorpora las herramientas para cargar el programa ya compilado en la memoria flash del hardware.

Es destacable desde la aparición de la versión 1.6.2 la incorporación de la gestión de librerías y la gestión de placas muy mejoradas respecto a la versión anterior y los avisos de actualización de versiones de librerías y cores.

Un programa de Arduino se denomina sketch o proyecto y tiene la extensión .ino, que debe estar en un directorio con el mismo nombre que el sketch para que funcione correctamente..

No es necesario que un sketch esté en un único fichero, pero si es imprescindible que todos los ficheros estén dentro del mismo directorio que el fichero principal y que este contenga obligatoriamente las funciones setup() y loop().

```
void setup() {
// put your setup code here, to run once:
}
void loop() {
// put your main code here, to run repeatedly:
}
```

La estructura básica de un sketch de Arduino es bastante simple y se compone de al menos dos partes. Estas dos partes son obligatorias y encierran bloques que contienen declaraciones, estamentos o instrucciones.

En donde setup() es la parte encargada de recoger la configuración y loop() es la que contiene el programa que se ejecutará cíclicamente (de ahí el término loop –bucle-). Ambas funciones son necesarias para que el programa trabaje.

La función de configuración (setup) debe contener la inicialización de los elementos y esta función sólo se ejecuta una vez justo después de hacer el reset y no se vuelve a ejecutar hasta que no haya otro reset. Es la primera función a ejecutar en el programa y se utiliza para configurar, inicializar variables, comenzar a usar librerías, etc…

La función bucle (loop) siguiente contiene el código que se ejecutará continuamente (lectura de entradas, activación de salidas, etc). Esta función es el núcleo de todos los programas de Arduino y se usa para el control activo de la placa.

La función loop se ejecuta justo después de setup.

Los componentes principales de un sketch de Arduino son:

- Variables, son un espacio en memoria donde se almacenan datos y estos datos pueden variar.
- Funciones, son un trozo de código que puede ser usado/llamado desde cualquier parte del sketch. A la función se le puede llamar directamente o pasarle unos parámetros, en función de cómo esté definida.
- setup() y loop, son dos funciones especiales que es obligatorio declarar en cualquier sketch.
- Comentarios, fundamentales para documentar el proyecto

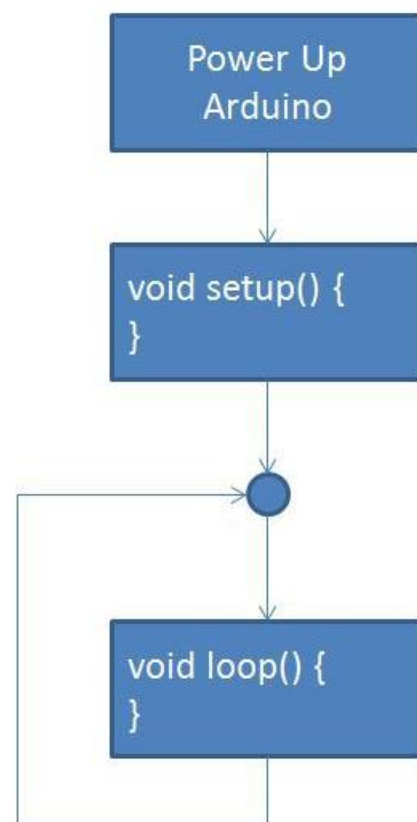

Ilustración IV - Diagrama de flujo de un sketch Arduino

3. TECNOLOGÍAS ESTUDIADAS PARA LA DETERMINACIÓN DEL PROBLEMA

Una vez investigado en qué consisten los DSLs y su funcionamiento, y haber realizado pruebas con el hardware de Arduino (placas, sensores y actuadores), se han estudiado diversas tecnologías hasta conseguir una solución para el problema planteado. A continuación, se explica cada una de estas tecnologías.

3.1. GENERACIÓN AUTOMÁTICA DE UN PARSER

La creación de un analizador sintáctico manualmente, puede ser una tarea muy costosa, pero afortunadamente, hay formas de generar un parser a partir de la gramática.

- Lex y Yacc: Lex se encarga del analizador léxico y Yacc del sintáctico, generando código C.
- Flex y Bison: funcionamiento similar a los dos anteriores, pero generan código C++.
- ANTL-R: es una herramienta desarrollada en Java, pero que puede generar código en Python entre otros lenguajes.
- Centrándonos en Python: Plex, Graco, PLY, Pyparsing y muchos más.

De entre todas las herramientas para generar analizadores léxicos y sintácticos mencionadas anteriormente, trabajaremos con Pyparsing ya que es muy sencillo de utilizar, pero potente al mismo tiempo, como Python. Además, es código 100% Python, por lo que no es necesario aprender un nuevo lenguaje para utilizarlo y aunque la gramática está reflejada en el código, es legible. Se puede probar por partes y aunque es un poco lento, no suele ser problema para un DSL.

3.1.1. PYPARSING

El propósito del módulo pyparsing es dar a los programadores que usan el lenguaje de programación Python una herramienta para extraer información de datos textuales estructurados.

En términos de potencia, este módulo es más potente que las expresiones regulares, tal como está incorporado en Python el módulo de expresiones regulares, pero no tan general como un compilador completo.

Para encontrar información dentro del texto estructurado, debemos ser capaces de describir esa estructura. El módulo pyparsing se basa en la tecnología de descripción de sintaxis fundamental incorporada en la Forma de Backus-Naur, o BNF.

La forma en que funciona el módulo pyparsing es hacer coincidir los patrones en el texto de entrada usando un analizador de descenso recursivo: escribimos producciones de sintaxis de tipo BNF, y pyparsing proporciona una máquina que coincide con el texto de entrada con esas producciones.

El módulo pyparsing funciona mejor cuando se puede describir la estructura sintáctica exacta del texto que está analizando. Una aplicación común de pyparsing es el análisis de archivos de registro. Las entradas de archivo de registro generalmente tienen una estructura predecible que incluye campos como fechas, direcciones IP, etc.

A continuación se ofrece un breve resumen del procedimiento general para escribir un programa que utiliza el módulo pyparsing.

1. Escribir el BNF que describe la estructura del texto que está analizando.
2. Instalar el módulo pyparsing si es necesario. Los paquetes de instalación más recientes de Python lo incluyen. Si no lo tiene, puede descargarlo desde la página principal de pyparsing. Se encuentra disponible en el Python Package Index o PyPI, por lo que podemos instalarlo desde línea de comandos:

sudo pip install pyparsing

3. Importar el módulo pyparsing en nuestro código Python. Le recomendamos que lo importe de esta manera:

import pyparsing

4. Su script montará un parser que coincida con su BNF. Un parser es una instancia de la clase base abstracta ParserElement que describe un patrón general.
5. La creación de un analizador para el formato de archivo de entrada es un proceso de abajo hacia arriba. Comience por escribir analizadores para las piezas más pequeñas, y ensamblarlas en pedazos más grandes y más grandes hasta completar un parser para el archivo completo.

6. Cree una cadena Python (tipo str o unicode) que contenga el texto de entrada que se va a procesar.

Si el analizador es p y el texto de entrada es s, este código intentará emparejarlos:

p.parseStringa(s)

Si la sintaxis de s coincide con la sintaxis descrita por p, esta expresión devolverá un objeto que representa las partes que coinciden. Este objeto será una instancia de clase ParseResults.

Si la entrada no coincide con su analizador, lanzará una excepción de la clase ParseException. Esta excepción incluirá información sobre en qué parte de la entrada el parse falló, como la línea y la columna.

El método parseString() procesa de forma secuencial el texto de entrada, utilizando los fragmentos de su analizador para coincidir con trozos del texto. Los analizadores de nivel más bajo a veces se llaman tokens, y los analizadores en niveles más altos se llaman patrones.

Puede asociar las acciones de análisis a cualquier analizador de componentes. Por ejemplo, el analizador para un entero puede tener una acción de análisis aneja asociada que convierte la representación de cadena en un int de Python.

Los caracteres no imprimibles, como el espacio y el tabulador, normalmente son ignorados entre tokens, aunque este comportamiento puede cambiarse. Esto simplifica en gran medida muchas aplicaciones, ya que no tiene que estorbar la sintaxis con una gran cantidad de patrones que especifican dónde se puede omitir el espacio en blanco.

7. Extraiga la información de su aplicación de la instancia devuelta ParseResults. La estructura exacta de esta instancia depende de cómo construyó el analizador.

3.2. INTERFACES GRÁFICAS EN PYTHON

Para realizar la GUI (Interfaz Gráfica de Usuario) se ha utilizado Python, por ello se han estudiado las herramientas de creación de interfaces con Python más populares:

3.2.1. TKINTER

TkInter (de TK Interface) es un módulo que nos permite construir interfaces gráficas de usuario multiplataforma en Python utilizando el conocido toolkit Tk. Python incluye este módulo por defecto, lo que hace que sea un toolkit muy popular. TkInter, además, es robusto, maduro y muy sencillo de aprender y de utilizar, contando con una amplia documentación.

Por otro lado hasta la versión 8.5 Tk era famoso por lo poco atractivo de sus widgets (cosa que se podía solucionar hasta cierto punto gracias a Tile). No es hasta esta versión que contamos con cosas tan básicas como textos con antialiasing en X11 o widgets como Treeview. En esta versión también se incluye Tile por defecto, por lo que contamos con un mejor aspecto general para todas las plataformas.

Sin embargo Python no se distribuye con Tk 8.5 hasta la versión 2.6, por lo que, para versiones de Python anteriores, es necesario recompilar TkInter para Tk 8.5 por nuestra cuenta, o bien usar Tile si no necesitamos ninguna de las nuevas características.

Es más, para poder usar la mayor parte de las nuevas características de Tk 8.5 es necesario instalar una librería que actúe como wrapper de Ttk (el nombre con el que han denminado al conjunto de los nuevos widgets y temas de Tk).

TkInter se distribuye bajo la PSFL (Python Software Foundation License) una licencia compatible con la GPL creada para la distribución de software relacionado con el proyecto Python. La PSFL carece de la naturaleza viral de la GPL, por lo que permite crear trabajos derivados sin que estos se conviertan necesariamente en software libre.

- Pros: Popularidad, sencillez y amplia documentación.
- Contras: Herramientas, integración con el sistema operativo, lentitud.
- Recomendado para: Desarrollo de prototipos rápidos.

3.2.2. WXPYTHON

WxPython es un wrapper open source para el toolkit anteriormente conocido como wxWindows: wxWidgets. Es posiblemente el toolkit para desarrollo de interfaces gráficas en Python más popular, superando incluso a TKinter, que como comentamos, se incluye por defecto con el intérprete de Python. WxPython cuenta con más y mejores widgets que TKinter, y ofrece un muy buen aspecto en todas las plataformas, utilizando MFC en Windows y GTK en Linux.

WxPython cuenta además con herramientas muy interesantes como wxGlade, una aplicación RAD (Desarrollo Rápido de Aplicaciones) para diseñar las interfaces gráficas de forma visual.

Sin embargo, la API sufre de una cierta falta de consistencia y un estilo muy alejado de Python y más cercano a C++, ya que, de hecho, uno de sus objetivos es no distanciarse demasiado del estilo de wxWidgets. Esto ha provocado que hayan aparecido distintos proyectos para abstraer al programador de los entresijos del toolkit, como Dabo o Wax, aunque estos han tenido un éxito muy comedido.

Tanto wxPython como wxWidgets se distribuyen bajo una licencia “wxWindows Licence”, que consiste esencialmente en una LGPL con la excepción de que las implementaciones derivadas en formato binario se pueden distribuir como el usuario crea conveniente.

Algunos ejemplos de aplicaciones conocidas creadas con wxPython son DrPython, wxGlade, Boa Constructor, Stani’s Python Editor y ABC.

- Pros: Popularidad, herramientas, multiplataforma.
- Contras: API muy poco pythonica.
- Recomendado para: Desarrollo multiplataforma.

3.2.3. PYGTK

PYGTK es un binding[3] de GTK, la biblioteca utilizada para desarrollar GNOME. Cuenta con una API muy clara, limpia y elegante y es, además, muy sencillo de aprender, solo superado en ese aspecto por Tkinter. PyGTK también cuenta con grandes herramientas para construir la interfaz de forma gráfica, como Glade o Gazpacho.

Un punto negativo es que, hasta hace poco, era necesario instalar X11 para poder usar PyGTK en Mac OS, dado que GTK no había sido portado. Actualmente se puede utilizar el GTK+ OS X Framework que se encuentra todavía en versión beta.

PyGTK se distribuye bajo licencia LGPL.

Algunas aplicaciones escritas con PyGTK son Deluge, Exaile, Listen, Envy, WingIDE, DeVeDe o emesene.

- Pros: Popularidad, sencillez, herramientas.
- Contras: Ligeramente más complicado de instalar y distribuir en Mac OS.
- Recomendado para: Cualquier tipo de aplicación, Especialmente interesante para Gnome.

3.2.4. PYQT

Es posible que PyQt, el binding de Qt para Python, sea la menos popular de las cuatro opciones, aunque es un toolkit sencillo de utilizar y con muchas posibilidades. Es especialmente interesante para el desarrollo en KDE, dado que Qt es la librería utilizada para crear este entorno.

No obstante el interés en Qt no se limita a KDE, sino que es una biblioteca multiplataforma que, además, desde la versión 4, utiliza widgets nativos para las distintas plataformas (anteriormente Qt emulaba el aspecto de la plataforma en la que corría).

Como aplicación de RAD se puede utilizar Qt Designer.

[3] **Binding:** es una adaptación de una biblioteca para ser usada en un lenguaje de programación distinto de aquel en el que ha sido escrita

PyQt utiliza un modelo de licencias similar al de Qt, con una licencia dual GPL/PyQt Commercial. Si nuestro programa es compatible con la licencia GPL, es decir, si vamos a publicar el código fuente y permitir a los usuarios modificar nuestra aplicación, podremos usar PyQt sin más preocupaciones. En caso contrario tendremos que pagar para obtener una licencia comercial.

Un par de ejemplos de aplicaciones que usan PyQt son Eric y QTorrent.

- Pros: Sencillez, herramientas, multiplataforma.
- Contras: Ligeramente más complicado de instalar y distribuir en Mac OS. Licencia.
- Recomendado para: Cualquier tipo de aplicación. Especialmente interesante para KDE.

Una vez analizadas las características de cada una de las herramientas de desarrollo de interfaces, se ha decidido trabajar con Tkinter, ya que es la más sencilla y fácil de aprender y está recomendada para desarrollo de prototipos rápidos. Además, viene preinstalado con Python en casi todas las plataformas y podemos hacer uso de una amplia y completa documentación.

3.3. ARDUINO-SKETCH 0.2

Arduino-sketch 0.2 es un paquete de Python escrito en código abierto y que se encuentra disponible en el Python Package Index o PyPI, que es el repositorio de software oficial para aplicaciones de terceros en el lenguaje de programación Python.

Este paquete permite compilar y subir un programa desarrollado en Arduino desde la línea de comandos, por lo que podemos prescindir del IDE de Arduino. Lo cual es una gran ventaja, ya que podemos usar nuestro editor favorito para escribir los ficheros Arduino .ino y ejecutar el comando “arduino-sketck -u my.ino”. Además, recuerda el nombre del último sketch ejecutado, por lo que permite llamar solamente “arduino-sketch” la próxima vez. También dispone de un comando de ayuda “arduino-sketch -help”.

Arduino-sketch usa configuración local, (.arduino_sketch) y una configuración de usuario (~/.arduino_sketch). Algunas opciones de configuración son:

- arduino_dir: ruta al directorio “core” de Arduino. Debe contener los directorios “hardware” y “tolos” de Arduino.
- avr_tools_path: ruta a ”avr-xxx binaries”.
- arduino_port: hace referencia al puerto serie donde está conectado nuestro dis. Por defecto es /dev/ttyUSB*, pero esto solo funcionará si tiene un único dispositivo serie conectado.
- board_tag: el nombre/tipo de nuestro dispositivo Arduino.

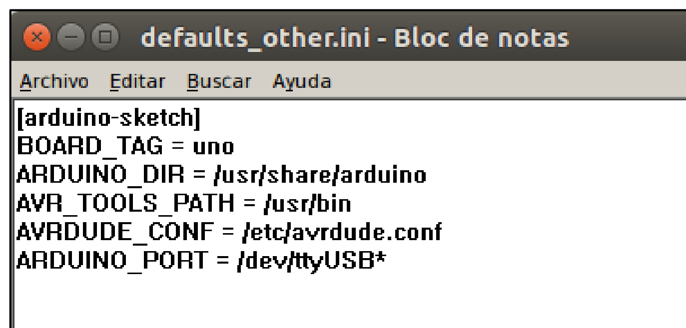

Ilustración V - Fichero de configuración de arduino-sketch

Para llevar a cabo la *instalación* podemos hacer uso de pip or easy_install, ya que arduino-sketch se encuentra disponible en PyPI.

sudo pip install arduino-sketch

Y además, necesitaremos, los componentes del core de Arduino 1.0 o superior. Para desinstalarlo tendremos que hacer uso de la siguiente instrucción:

sudo pip uninstall arduino-sketch

4. ARQUITECTURA

4.1. INTRODUCCIÓN

La arquitectura que compone la solución encontrada para el proyecto se basa en una combinación de varias de las tecnologías explicadas anteriormente.

Esta arquitectura se fundamenta en el uso de un DSL, para transformar los ficheros ".bot" en ficheros ".ino" (sketch de Arduino) y de esta forma integrar automáticamente sensores y actuadores para su uso en placas Arduino.

En primer lugar, la interfaz lee los ficheros placas.json y sensores.json para incorporar al sistema las placas y los sensores disponibles. De las placas recoge información como los pines analógicos y digitales de los que dispone cada placa, ya que puede haber mucha diferencia entre unas placas y otras. Mientras que de los sensores sólo almacena un identificador y si son de tipo analógico o digital.

Una vez cargado estos dos ficheros, la interfaz nos permite incorporar a nuestro sistema distintos tipos de sensores y actuadores, indicándole el pin donde va a ir conectado. Además, podemos incluir buses del tipo I2C y asignarle una dirección a cada uno.

A continuación, podemos guardar los datos de nuestro robot (sensores, actuadores y buses introducidos) generando un fichero .bot. Para que después la aplicación nos genere de forma automática el código Arduino. Además, también podemos acceder a dicho código, realizar modificaciones si así lo deseamos y cargarlo en nuestra placa Arduino.

El código generado, realiza el envío de los datos de los sensores serializados en formato json, a través del puerto serie y gestiona los comandos recibidos para los actuadores. Las funciones para gestionar dichos comandos se encuentran almacenadas en el fichero "función.py".

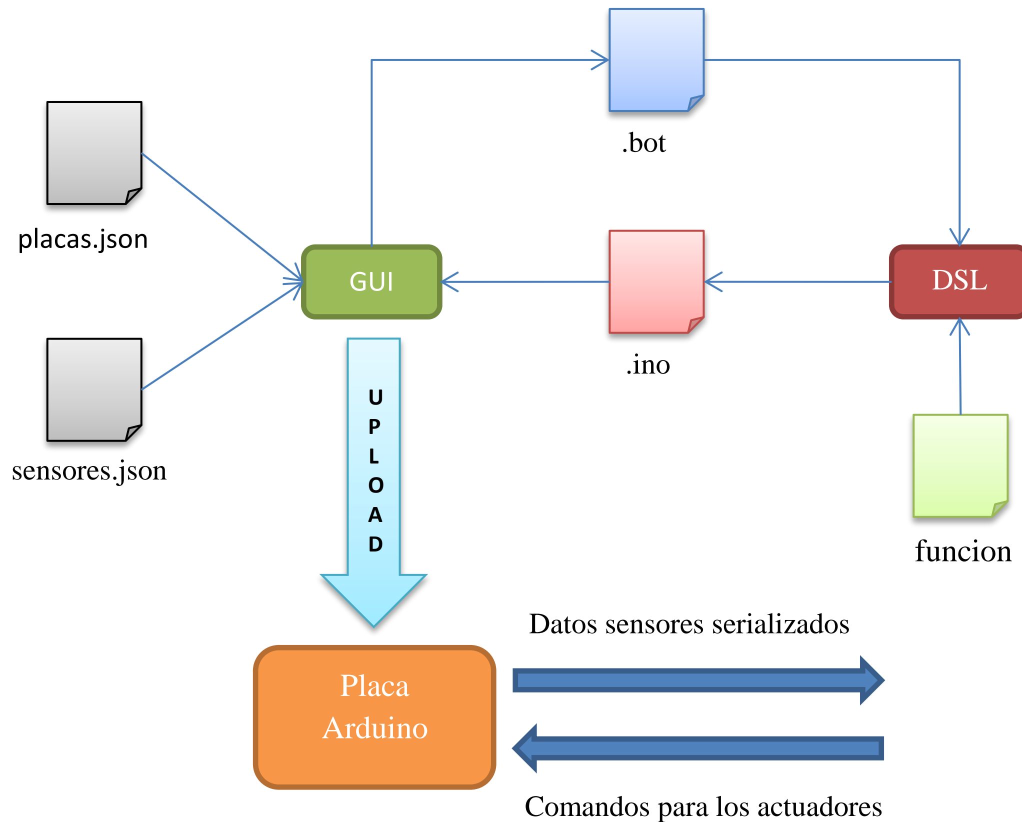

Ilustración VI - Diagrama de la Arquitectura del Sistema

4.2. DESARROLLO DEL DSL

Como ya se ha explicado anteriormente un DSL es un pequeño lenguaje para el dominio específico de un problema que hace más sencillo trabajar en dicho dominio. Cuando estamos trabajando en un dominio particular, hay que escribir el código en una sintaxis que se ajusta al domino.

Recordemos que podemos clasificar los DSLS en dos grandes grupos:

- DSL externo: el código está escrito en un archivo externo o como una cadena, que es leída y analizada por la aplicación.
- DSL interno: utiliza las funciones del lenguaje (como metaclases) para permitir a las personas escribir código en Python que se asemeja a la sintaxis del dominio.

Para guardar los datos de nuestro robot en el fichero ".bot", utilizaresmos un DSL Externo.

```
david
UNO
9600
LED, pin:12
S, pin:9
I2C, 80
```

Donde la primera línea se corresponde con el nombre de nuestro robot, la segunda con el tipo de placa utilizado, la tercera es el “rate” de transmisión y a continuación se encuentran los sensores, actuadores y buses definidos.

Como se puede apreciar, la estructura del robot es sencilla de entender, y puede ser editado fácilmente por los usuarios finales, sin necesidad de tener conocimientos de programación. Aunque en nuestro sistema su edición se hará a través de la interfaz. El problema que se presenta a continuación, es que tenemos que leer y parsear el fichero .bot.

Para parsear los datos de nuestro DSL externo utilizamos pyparsing, para ello lo primero que tenemos que hacer es implementar una gramática. Haciendo uso de las funciones proporcionadas por pyparsing podemos definir nuestra gramática de una forma rápida y sencilla.

```
r1 = "[aA][0-9]+"
eol = "\n"
sep = ","
colon = ":"
nombre = Word(alphanums).setResultsName("nombre")
placa = Word(alphas)
ratio = Word(nums).setResultsName("ratio")
clave = Word(alphanums).setResultsName("param_clave")
valor = Word(alphanums).setResultsName("param_valor")
param = Group(sep + clave + colon + valor).setResultsName("sensor_param")
sensor_type = Word(alphas).setResultsName("sensor_type")
pin_analog = Regex(r1)
pin = Suppress("pin:")+(Word(nums) | pin_analog).setResultsName("pin")
direccion = Word(nums).setResultsName("direccion")
bus = Group("I2C" + sep + direccion + eol).setResultsName("bus")
sensor = Group(sensor_type + sep + pin +
Optional(OneOrMore(param)).setParseAction(convert_param_to_dict) +
eol).setResultsName("sensor")
robot = nombre + eol + placa + eol + ratio + eol +
Optional(OneOrMore(sensor)).setResultsName("sensores") +
Optional(OneOrMore(bus)).setResultsName("buses")
```

Podemos observar en el código que PyParsing refleja casi perfectamente la gramática BNF. Una vez realizado el parseado, la salida generada por print robot.parseString(input_string) es la siguiente:

```
['david', '\n', 'UNO', '\n', '9600', '\n', ['LED', ',', '12', {}, '\n'], ['S', ',', '9', {}, '\n'],
['I2C,', '80', '\n']]
```

PyParsing ha parseado cuidadosamente la cadena de entrada en tokens, pero podemos ver que también ha procesado muchos tokens "ruidosos" como saltos de líneas y separadores, como son la "," y ":". Por ello, con la función Suppress los eliminamos de la cadena de salida, que quedará como se muestra a continuación.

```
eol = Suppress("\n")
sep = Suppress(",")
colon = Supress(":")
```

```
['david', 'UNO', '9600', ['LED', '12', {}], ['S', '9', {}], ['I2C', '80']]
```

Haciendo uso de la función Group conseguimos que los datos de un mismo sensor o actuador queden recogidos en una misma lista, y haciendo uso de la función setResultsName podremos referirnos posteriormente al token parseado por su nombre.

Además, por medio de la función convert_param_to_dict los parámetros son parseados en un diccionario.

```
def convert_param_to_dict(tokens):
        param_dict = {}
        for token in tokens:
                param_dict[token.param_clave] = token.param_valor
        return param_dict
```

Una vez que tenemos parseado el fichero .bot el siguiente paso es generar una salida con código Arduino. Para ello necesitamos crear una sintaxis que coincida con el dominio de salida. En lugar de limitamos a trabajar con sentencias condicionalse sobre las cadenas recibidas por el parser, ya que el código resultante queda ilegible y muy difícil de modificar, desarrollaremos una sintaxis contenida en el lenguaje anfitrión, que en este caso es Python.

```
def render(self, robot):
	s_dict = {"LED": Led, "IR": Infrarrojo, "US": Ultrasonido, "S": Servo, "LDR": 
LDR}
	lista_componentes = []
	for sensor in robot.sensores:
		s_dict[sensor.sensor_type].num += 1
	lista_componentes.append(s_dict[sensor.sensor_type](nombre = sensor.sensor_type, 
pin = sensor.pin, num = s_dict[sensor.sensor_type].num, **sensor[2]))
	for sensor in robot.sensores:
		s_dict[sensor.sensor_type].num = 0
	for bus in robot.buses:
		I2C.num += 1
		lista_componentes.append(I2C(direccion = bus.direccion, num = I2C.num))
	for bus in robot.buses:
		I2C.num = 0
	return Robot(*lista_componentes, ratio = robot.ratio)
```

El render se encarga de transformar las cadenas parseadas que recibe del parser en código Arduino. Para ello, según el tipo de componente que le llegue (sensor/actuador o bus) crea un objeto de la clase que corresponda, pasándole los “keywords arguments” necesarios. Y por último crea una instancia de la clase Robot, que recibe como argumentos todos los componentes recogidos del parser.

Para cada tipo de sensor/actuador hay definida una clase que le proporciona una serie de atributos mediante variables de clase. A continuación se muestran algunas de ellas:

```
class Infrarrojo(Sensor):
	num = 0
	tipo = "analog"
	e_s = "OUTPUT"
	libreria = ""

class Ultrasonido(Sensor):
	num = 0
	tipo = "digital"
	e_s = "OUTPUT"
	e_s2 = "INPUT"
	libreria = ""

class Servo(Sensor):
	num = 0
	tipo = "digital"
	e_s = ""
	libreria = "Servo.h"
	default_setup = "{}.attach({});"
```

Estas clases heredan de la clase Sensor, que se encarga de generar varias cadenas de caracteres para cada instancia de un sensor/actuador. Estas cadenas contienen las líneas de código que posteriormente conformaran nuestro fichero Arduino. Al igual que la clase Sensor, la clase I2C también genera dichas cadenas, aunque en este caso sólo para incluir la librería "Wire.h" y definir una constante con el valor de dirección asignado. Las cadenas generadas para cada una de las instancias de la clase sensor y bus son las siguientes:

- Include: include con la librería necesario o si no vacía.
- Define: definición de pines o direcciones, si no vacía.
- Instancia: contiene la creación de instancias si es necesario, como por ejemplo en el caso del servo, si no vacía.
- Cad_enteros: contiene la declaración de variables enteras para almacenar los datos que serán leídos y posteriormente serializados.
- Setup: contiene las líneas de código que deberían ir en el setup.
- Loop: contiene las líneas de código que deberían ir en el loop.

Por último la clase Robot, de la que heredan las clases Sensor e I2C se encarga de unir las cadenas anteriores de cada sensor y bus y conformar el fichero de salida.

Para el ejemplo de fichero .bot que veíamos anteriormente que constaba de un LED, un servo y un I2C, la estructura del fichero Arduino generado es la siguiente:

led.include + s.include + i2c.include

led.define + s.define + i2c.define

led.instancia + s.instancia + i2c.instancia

led.cad_enteros + s.cad_enteros + i2c.cad_enteros

void setup(){ led.setup + s.setup + i2c.setup }

void loop (){ led.loop + s.loop + i2c.loop}

Ilustración VII - Estructura fichero Arduino generado

4.3. EJEMPLO FICHERO DE ENTRADA Y DE SALIDA

Fichero de entrada

```
david
UNO
9600
LED, pin:12
S, pin:9
I2C, 80
```

Fichero de salida

```
#include <Servo.h>
#include "Wire.h"

#define I2C0 80
#define pin_LED0 12
#define pin_S0 9
Servo s0;
int LED0, S0;

void serializarDatosLeidos(){
		Serial.print("{");
		Serial.print("LED:[");
		Serial.print(LED0);
		Serial.print("],");
		Serial.print("S:[");
		Serial.print(S0);
		Serial.print("]");
		Serial.println("}");
}

void setup(){
		Serial.begin(9600);
		pinMode(pin_LED0, INPUT);
		s0.attach(pin_S0);
}

void loop(){
		LED0 = digitalRead(pin_LED0);
		S0 = s0.read();
		serializarDatosLeidos();
		delay(1000);
}
```

Ilustración VIII - Ejemplo de ficheros de entrada y de salida

4.4. CARGA DEL FICHERO DE SALIDA EN LA PLACA

Para cargar el código Arduino en la placa se utiliza el módulo arduino-sketch que se ejecuta desde Python haciendo uso de las librerías shlex y subprocess. Estas dos librerías nos permiten generar un nuevo proceso para ejecutar el comando permitiendo controlar su ejecución y obtener su salida y/o errores que pudieran darse

La implementación de esta funcionalidad se muestra a continuación:

```
def uploadCodigo(self):
        f_salida = "{}.ino".format(self.robot.getNombre())
        mensaje = self.sistema_fich.uploadCodigo(f_salida)
        if mensaje.find("Thank you") != -1:
                mbox.showinfo("Información", "Código subido correctamente")
        elif mensaje.find("[reset] Error 1") != -1:
                mbox.showwarning("Aviso", "Placa desconectada")
        else:
                mbox.showerror("Error", mensaje)

def uploadCodigo(self, f_salida):
        comando = "arduino-sketch -u "+f_salida
        proceso = subprocess.Popen(shlex.split(comando), stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
        out, err = proceso.communicate()
        errores = err.decode("utf-8")
        salida = out.decode("utf-8")
        return errores
```

5. DIAGRAMA DE CLASES

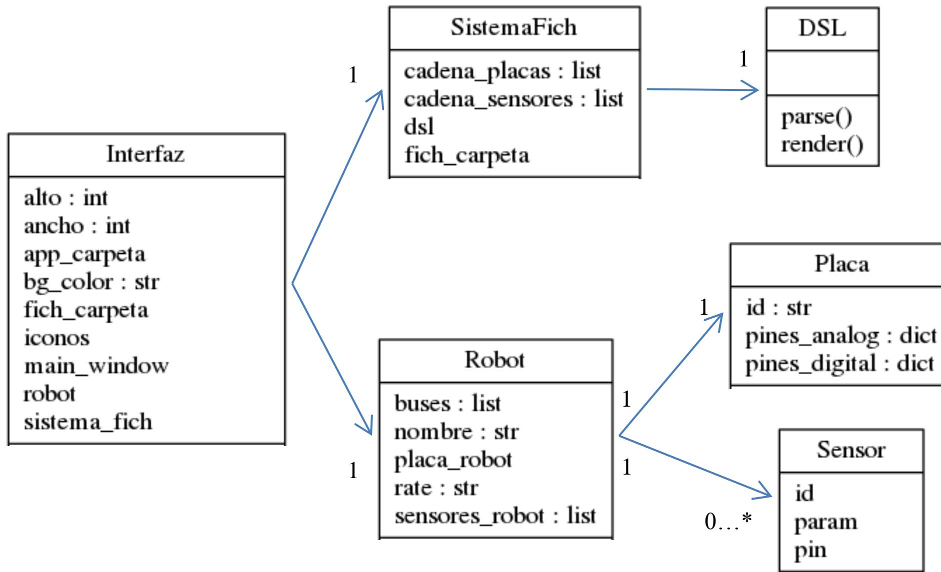

Ilustración IX - Diagrama de clases

5.1. DIAGRAMA DE CLASES DEL DSL

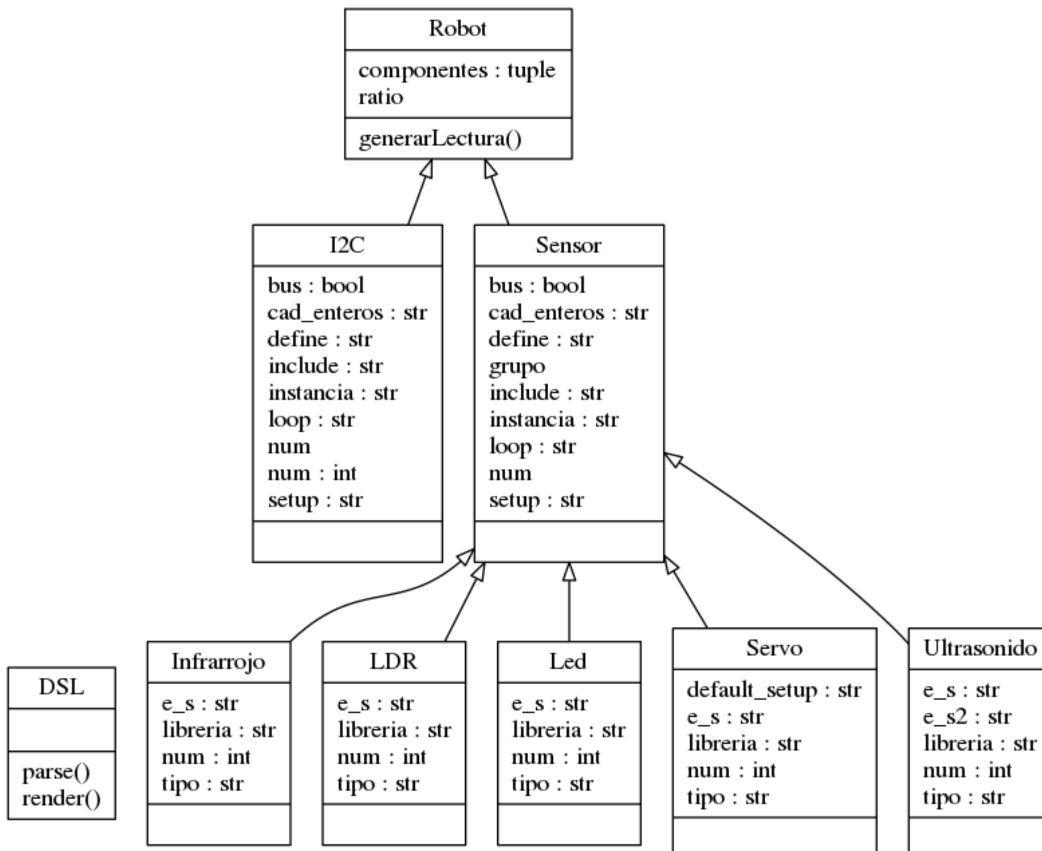

Ilustración X - Diagrama de clases del DSL

6. MANUAL DE USUARIO

La aplicación LearnBot es multiplataforma, pero para que funcione correctamente, es necesario tener instalado los paquetes de Python pyparsing y arduino-sketch. Ambos se encuentran disponibles en el repositorio PyPI de Python.

Ilustración XI - Ventana inicio LearnBot

La aplicación es muy sencilla e intuitiva, por lo que resulta muy fácil de utilizar. Al iniciar la aplicación, en el Menú Archivo → "**Nuevo**", podemos crear un nuevo robot. Para ello hay que introducir un nombre, tiene que ser una sola palabra, puede contener números y no puede quedar vacío. El tipo de placa que vamos a utilizar y el rate de transferencia que queremos establecer para la comunicación serie. Una vez hecho esto, la aplicación mostrará los datos introducidos en la pantalla.

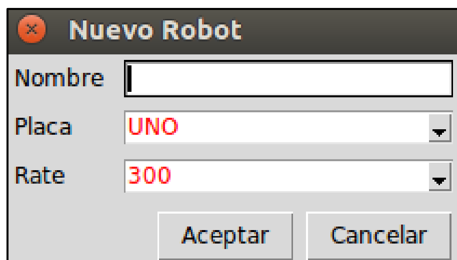

Ilustración XIV - Creando un nuevo robot

El siguiente paso, será introducir los sensores, para ello en el menú Herramientas seleccionamos la opción "**Insertar Sensor**". Podemos insertar tantos sensores como pines libres haya en nuestra placa. Según el tipo de sensor/actuador que seleccionemos (digital o analógico) la aplicación nos ofrecerá los pines que están libres según corresponda. Para el sensor de Ultrasonido, tendremos que seleccionar un segundo pin, que se corresponde con el echo.

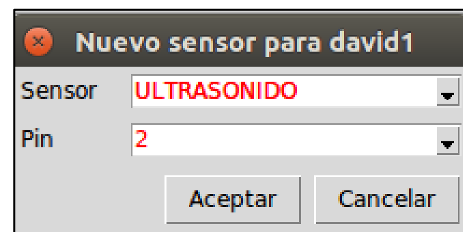

Ilustración XIII - Insertando un nuevo sensor

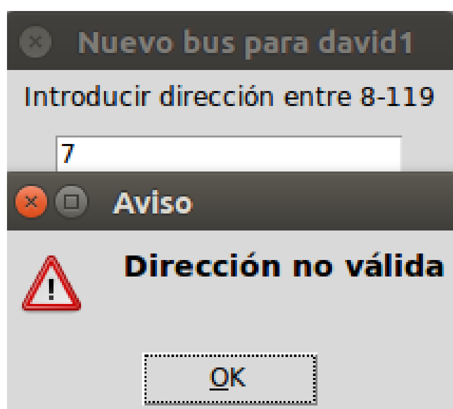

Ilustración XII - Dirección no válida

También podremos seleccionar la opción "**Insertar bus**", para utilizar un bus I2C. La

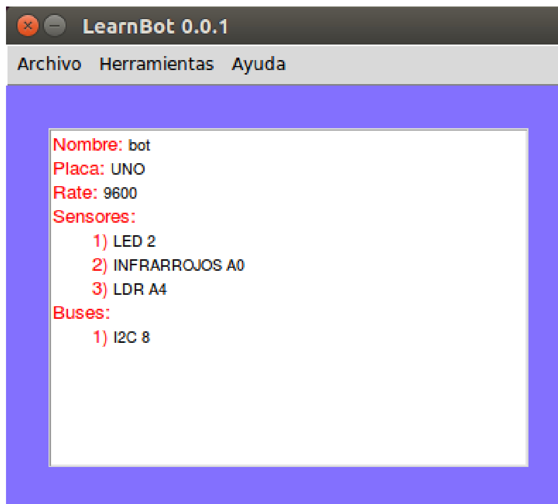

Ilustración XV - Informació del robot

aplicación nos permitirá seleccionar una dirección entre 8 y 119 que no haya sido ya asignada. Todas las inserciones quedan reflejadas en la pantalla inicial de la aplicación.

Una vez que ya hemos intertado los sensores y los buses podemos generar el código Arduino en el menú Herramientas → **"Generar .ino".** La aplicación nos indicará la ruta donde ha generado el código Arduino. En Herrmaientas podemos modificar el código generado mediante la opción **"Modificar código".** Se abrira una ventana donde aparecerá el código, que podremos editarlo y guardar las modificaciones realizadas.

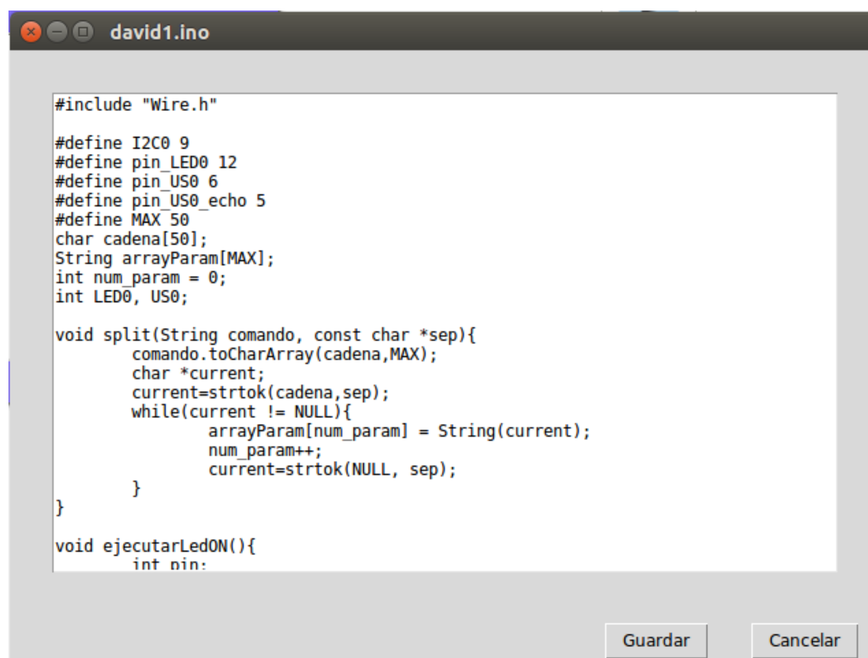

Ilustración XVI - Modificando código generado

Por último, mediante la opción **"Upload .ino"** que se encuentra disponible en el menú Herramientas podemos subir el código a la placa Arduino. Sí la placa está desconectada o se produce algún herror durante la subuda, la aplicación mostrará un mensaje de error y si toda va correcto, nos indicara que se ha subido correctamente.

7. CONCLUSIONES Y TRABAJOS FUTUROS

El resultado final es un producto totalmente funcional, orientado a facilitar el trabajo a las personas que se están iniciando en el uso de sensores y actuadores con placas Arduino. El uso de la interfaz, además de generar el código automáticamente, previene de errores, como asignar un mismo pin a distintos sensores o utilizar direcciones para buses I2C duplicadas.

Además, la aplicación supone un gran ahorro de tiempo, ya que se automatiza la puesta en marcha de los sensores y actuadores generando gran cantidad de código. Por ello, su uso también está recomendado para personas que trabajan habitualmente con dicha tecnología y no sólo para novatos. Se puede generar el código de forma automática y después hacer las modificaciones que consideren oportunas.

Al haber sido realizado completamente en Python, nos permite usarlo en distintas plataformas.

La herramienta implementada cumple los objetivos propuestos inicialmente: integra automáticamente sensores y actuadores para su uso con placas de desarrollo Arduino y envía los datos de los sensores serializados en formato json, a través del puerto serie, y gestiona los comandos para los actuadores.

Sería deseable en un futuro incorporar otros canales de comunicación, además del puerto serie, como puede ser ethernet o bluetooh. También se podrían incorporar nuevas funcionalidades a la interfaz, como la de abrir un fichero ".bot" que hemos definido anteriormente y donde se encuentran especificados los sensores y sus pines correspondientes. Por último, se podría incorporar la gestión de comandos a través de la interfaz, pudiendo introducir distintos tipos de comandos y que esta genere las cabeceras de los métodos que se van a encargar de ejecutarlos. Esto supondría un gran ahorro de tiempo y esfuerzo.

8. BIBLIOGRAFÍA

[1] Lenguajes Específicos del Dominio (DSL) http://97cosas.com/programador/dsl.html

[2] DSL http://wiki.uqbar.org/wiki/articles/dsl.html

[3] Introducción a DSL (Lenguajes Específicos de Dominios) con Python https://es.slideshare.net/jileon/introduccin-a-dsl-lenguajes-especficos-de-dominios-con-python

[4] DSL in Python https://in.pycon.org/2011/static/files/talks/24/dsl_with_python.pdf

[5] Aprendiendo Arduino (Microcontroladores) https://aprendiendoarduino.wordpress.com/category/atmel/

[6] Arduino https://www.arduino.cc/

[7] Arduino-sketch 0.2 https://pypi.python.org/pypi/arduino-sketch

[8] Pyparsing quick reference http://infohost.nmt.edu/tcc/help/pubs/pyparsing/web/index.html

[9] How to use pyparsing http://pyparsing.wikispaces.com/HowToUsePyparsing

[10] Tutorial de Tkinter http://python-para-impacientes.blogspot.com.es/p/tutorial-de-tkinter.html

[11] Tkinter Tutorial http://zetcode.com/gui/tkinter/

[12] Tkinter 8.5 reference: a GUI for Python http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html

[13] An Introduction To Tkinter http://effbot.org/tkinterbook/tkinter-index.htm

[14] Lista desplegable Combobox en Tkinter http://recursospython.com/guias-y-manuales/lista-desplegable-combobox-en-tkinter/

[15] Curso de Pyton para principiantes http://www.eugeniabahit.com/

[16] Interfaces Gráficas de Usuario en Python http://mundogeek.net/archivos/2008/11/24/interfaces-graficas-de-usuario-en-python/

[17] Pyreverse : UML Diagrams for Python https://www.logilab.org/blogentry/6883

[18] Estructura de un sketch en Arduino https://aprendiendoarduino.wordpress.com/2015/03/24/estructura-de-un-sketch-en-arduino/

[19] Qué es Arduino https://aprendiendoarduino.wordpress.com/2016/09/25/que-es-arduino/

[20] Sensores Arduino https://aprendiendoarduino.wordpress.com/2017/06/24/sensores-arduino-2/

[21] Electrónica, sensores, actuadores y periféricoas https://aprendiendoarduino.wordpress.com/2016/11/06/electronica-sensores-actuadores-y-perifericos/

[22] El bus I2C en Arduino https://www.luisllamas.es/arduino-i2c/

[23] I2C https://aprendiendoarduino.wordpress.com/2016/11/14/bus-i2ctwi/

[24] Reloj RTC https://tuelectronica.es/modulo-rtc-ds1307-arduino/#esquemas-y-montaje

[25] Servomotor con Arduino https://programarfacil.com/tutoriales/fragmentos/servomotor-con-arduino/

[26] Arduino y fotosensores LDRs https://www.prometec.net/ldrs/

[27] Sensor de ultrasonido http://elcajondeardu.blogspot.com.es/2014/03/tutorial-sensor-ultrasonidos-hc-sr04.html

[28] Tutorial sensor de distancia Sharp http://www.naylampmechatronics.com/blog/55_tutorial-sensor-de-distancia-sharp.html

9. ÍNDICE DE ILUSTRACIONES

Ilustración I - Ejemplo de Árbol sintáctico 15
Ilustración II -Características generales placa Arduino 21
Ilustración III – Esquema sensores y actuadores 28
Ilustración IV - Diagrama de flujo de un sketch Arduino 30
Ilustración V - Fichero de configuración de arduino-sketch 38
Ilustración VI - Diagrama de la Arquitectura del Sistema 40
Ilustración VII - Estructura fichero Arduino generado 44
Ilustración VIII - Ejemplo de ficheros de entrada y de salida 45
Ilustración IX - Diagrama de clases........ 47
Ilustración X - Diagrama de clases del DSL 47
Ilustración XI - Ventana inicio LearnBot 48
Ilustración XII - Dirección no válida 48
Ilustración XIII - Insertando un nuevo sensor 48
Ilustración XIV - Creando un nuevo robot........ 48
Ilustración XV - Informació del robot 49
Ilustración XVI - Modificando código generado 49
Ilustración XVII - Bus I2C........ 55
Ilustración XVIII - Funcionamiento I2C 57
Ilustración XIX - Direcciones de esclavo 58
Ilustración XX - Funciones básicas I2C........ 60

10. ÍNDICE DE TABLAS

Tabla 1 - Comparación GPL y DSL 8
Tabla 2 - Comparación de DSL interno y externo 13
Tabla 3 - Símbolos especiales de una gramática 16
Tabla 4 - Direcciones reservadas 58
Tabla 5 - Pines asociados al I2C 59

11. ANEXO I: BUS I2C

El bus I2C tiene un gran interés porque, de forma similar a lo que pasaba con el bus SPI, una gran cantidad de dispositivos disponen de conexión mediante I2C, como acelerómetros, brújulas, displays, etc.

El estándar I2C (Inter-Integrated Circuit) fue desarrollado por Philips en 1982 para la comunicación interna de dispositivos electrónicos en sus artículos. Posteriormente fue adoptado progresivamente por otros fabricantes hasta convertirse en un estándar del mercado.

I2C también se denomina TWI (Two Wired Interface) únicamente por motivos de licencia. No obstante, la patente caducó en 2006, por lo que actualmente no hay restricción sobre el uso del término I2C.

El bus I2C requiere únicamente dos cables para su funcionamiento, uno para la señal de reloj (CLK) y otro para el envío de datos (SDA), lo cual es una ventaja frente al bus SPI. Por contra, su funcionamiento es un poco más complejo, así como la electrónica necesaria para implementarla.

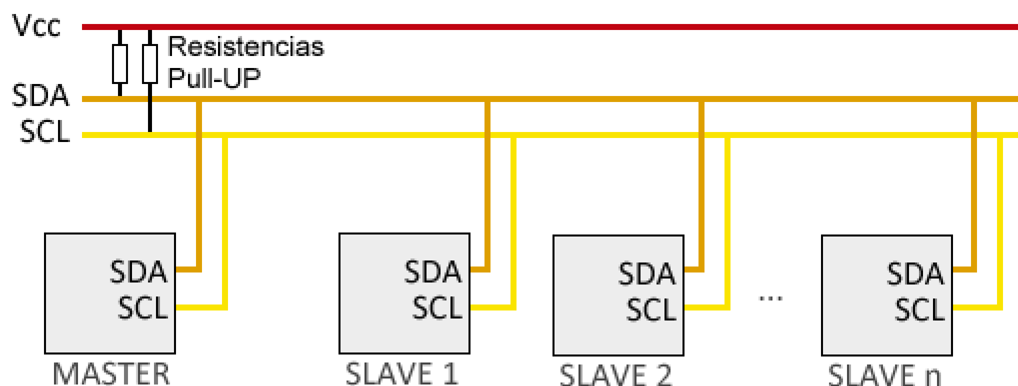

Ilustración XVII - Bus I2C

En el bus **cada dispositivo dispone** de una dirección, que se emplea para acceder a los dispositivo de forma individual. Esta dirección puede ser fijada por hardware (en cuyo caso, frecuentemente, se pueden modificar los últimos 3 bits mediante jumpers o interruptores) o totalmente por software.

En general, cada dispositivo conectado al bus **debe tener una dirección única**. Si tenemos varios dispositivos similares tendremos que cambiar la dirección o, en caso de no ser posible, implementar un bus secundario.

El bus I2C tiene una **arquitectura de tipo maestro-esclavo**. El dispositivo maestro inicia la comunicación con los esclavos, y puede mandar o recibir datos de los esclavos. Los esclavos no pueden iniciar la comunicación (el maestro tiene que preguntarles), ni hablar entre si directamente.

Es posible disponer de más de un maestro, pero **sólo uno puede ser el maestro cada vez**. El cambio de maestro supone una alta complejidad, por lo que no es algo frecuente.

El bus I2C es síncrono. El maestro proporciona una señal de reloj, que mantiene sincronizados a todos los dispositivos del bus. De esta forma, se elimina la necesidad de que cada dispositivo tenga su propio reloj, de tener que acordar una velocidad de transmisión y mecanismos para mantener la transmisión sincronizada (como en UART)

El protocolo I2C prevé **resistencias de Pull-UP de las líneas a Vcc**. En Arduino veréis que frecuentemente no se instalan estas resistencias, ya que la librería Wire activa las resistencias internas de Pull-UP. Sin embargo las resistencias internas tienen un valor de entre 20-30kOhmnios, por lo que son unas resistencias de Pull-UP muy blandas.

Usar unas resistencias blandas implica que los flancos de subida de la señal serán menos rápidas, lo que implica que podremos usar velocidades menores y distancias de comunicación inferiores. Si queremos emplear velocidades o distancias de transmisión superiores, deberemos poner físicamente resistencias de Pull-UP de entre 1k a 4K7.

FUNCIONAMIENTO DEL BUS I2C

Para poder realizar la comunicación con solo un cable de datos, el bus I2C emplea una trama (el formato de los datos enviados) amplia. La comunicación costa de:

- 7 bits a la dirección del dispositivo esclavo con el que queremos comunicar.
- Un bit restante indica si queremos enviar o recibir información.
- Un bit de validación
- Uno o más bytes son los datos enviados o recibidos del esclavo.
- Un bit de validación

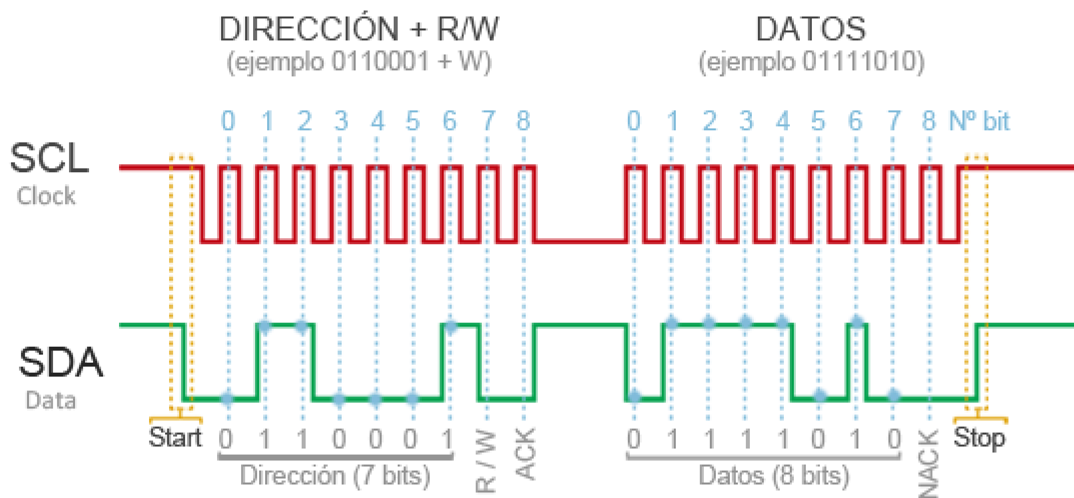

Ilustración XVIII - Funcionamiento I2C

Con estos 7 bits de dirección es posible acceder a 112 dispositivos en un mismo bus (16 direcciones de las 128 direcciones posibles se reservan para usos especiales)

Este incremento de los datos enviados (18bits por cada 8bits de datos) supone que, en general, la velocidad del bus I2C es reducida. La velocidad estándar de transmisión es de 100Mhz, con un modo de alta velocidad de 400Mhz.

En principio, el número de dispositivos que se puede conectar al bus no tiene límites, aunque hay que observar que la capacidad máxima sumada de todos los dispositivos no supere los 400 pF. El valor de los resistores de polarización no es muy crítico, y puede ir desde 1K8 (1.800 ohms) a 47K (47.000 ohms). Un valor menor de resistencia incrementa el consumo de los integrados pero disminuye la sensibilidad al

ruido y mejora el tiempo de los flancos de subida y bajada de las señales. Los valores más comunes en uso son entre 1K8 y 10K.

Lo más común en los dispositivos para el bus I2C es que utilicen direcciones de 7 bits, aunque existen dispositivos de 10 bits. Este último caso es raro. Una dirección de 7 bits implica que se pueden poner hasta 128 dispositivos sobre un bus I2C, ya que un número de 7 bits puede ir desde 0 a 127. Cuando se envían las direcciones de 7 bit, de cualquier modo la transmisión es de 8 bits. El bit extra se utiliza para informarle al dispositivo esclavo si el dispositivo maestro va a escribir o va a leer datos desde él. Si el bit de lectura/escritura (R/W) es cero, el dispositivo maestro está escribiendo en el esclavo. Si el bit es 1 el maestro está leyendo desde el esclavo. La dirección de 7 bit se coloca en los 7 bits más significativos del byte y el bit de lectura/escritura es el bit menos significativo.

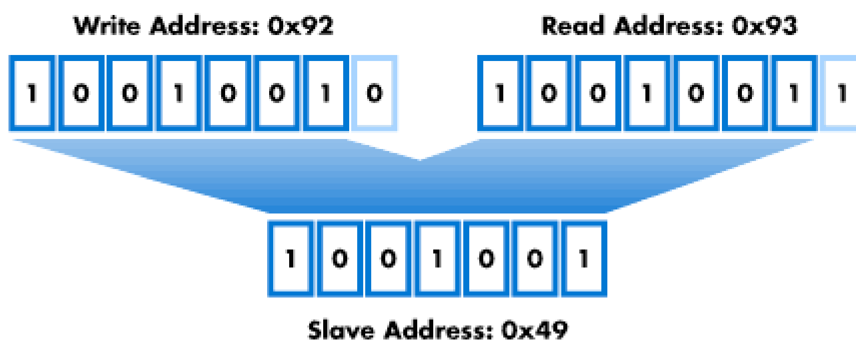

Ilustración XIX - Direcciones de esclavo

SLAVE ADDRESS	$R/\overline{W}$ BIT	DESCRIPTION
0000 000	0	General call address
0000 000	1	START byte
0000 001	X	CBUS address
0000 010	X	Reserved for different bus format
0000 011	X	Reserved for future purposes
0000 1XX	X	Hs-mode master code
1111 1XX	X	Reserved for future purposes
1111 0XX	X	10-bit slave addressing

Tabla 4 - Direcciones reservadas

VENTAJAS Y DESVENTAJAS DEL I2C

VENTAJAS

- Requiere pocos cables
- Dispone de mecanismos para verificar que la señal hay llegado

DESVENTAJAS

- Su velocidad es media-baja
- No es full duplex
- No hay verificación de que el contenido del mensaje es correcto

EL BUS I2C EN ARDUINO

Arduino dispone de soporte I2C por hardware vinculado físicamente a ciertos pines. También es posible emplear cualquier otro grupo de pines como bus I2C a través de sofware, pero en ese caso la velocidad será mucho menor.

Los pines a los que está asociado varían de un modelo a otro. La siguiente tabla muestra la disposición en alguno de los principales modelos. Para otros modelos, consultar el esquema de patillaje correspondiente.

MODELO	SDA	SCK
Uno	A4	A5
Nano	A4	A5
Mini Pro	A4	A5
Mega	20	21

Tabla 5 - Pines asociados al I2C

Para usar el bus I2C en Arduino, el IDE Standard proporciona la librería “Wire.h”, que contiene las funciones necesarias para controlar el hardware integrado.

Algunas de las funciones básicas son las siguientes:

```
1  Wire.begin()  // Inicializa el hardware del bus
2  Wire.beginTransmission(address); //Comienza la transmisión
3  Wire.endTransmission(); // Finaliza la transmisión
4  Wire.requestFrom(address,nBytes);  //solicita un numero de bytes al esclavo en la dirección addres
   s
5  Wire.available();  // Detecta si hay datos pendientes por ser leídos
6  Wire.write();  // Envía un byte
7  Wire.read();   // Recibe un byte
8  
9  Wire.onReceive(handler); // Registra una función de callback al recibir un dato
10 Wire.onRequest(handler); // Registra una función de callback al solicitar un dato
```

Ilustración XX - Funciones básicas I2C

Existen otras librerías más avanzadas que Wire.h para manejar el bus I2C, como por ejemplo I2Cdevlib o I2C library.