# MÁQUINA MIDI BASADA EN ARDUINO

AUTOR:
ÀNGEL DAVID SEMPERE SÁNCHEZ

TUTOR:
JOSÉ MANUEL IÑESTA QUEREDA

ÁREA DE LENGUAJES Y SISTEMAS INFORMÁTICOS

Curso:
2012/2013

# Índice

# 1. Introducción

## 1.1 Objetivos

El siguiente proyecto pretende proporcionar las herramientas necesarias para que cualquier persona, sin que tenga conocimientos específicos de programación, pueda construir sus propios instrumentos musicales electrónicos, haciendo uso de componentes de bajo coste, plataformas libres y estándares reconocidos por sintetizadores y programas de producción musical.

Las herramientas que proporcionará son dos. La primera de ellas es un lenguaje de definición de instrumentos que, a modo de lenguaje de programación, se encargará de asociar reacciones de salida a las acciones de entrada registradas mediante los componentes electrónicos.

La segunda de ellas es un traductor que, a modo compilador, traduzca el código de definición de instrumentos en un código compilable y ejecutable por una plataforma de desarrollo libre.

Con estas herramientas no sólo se pretende poder replicar instrumentos musicales existentes, sino abrir la puerta a la creación de instrumentos que exploren otra forma de despachar sonidos, haciendo uso de componentes electrónicos poco empleados, como *joysticks*, añadiendo componentes nunca empleados, como lectores de tarjetas RFID/NFC, o estableciendo un marco estándar para la comunicación entre instrumentos mediante el Internet de las Cosas[1] o módulos Bluetooth.

Dado que los usos descritos requieren un largo período de desarrollo, este proyecto se centra en asentar la parte básica del proyecto, que sea suficiente para poder replicar instrumentos de viento o instrumentos cuyos fundamentos sean similares a éstos. Además, se demostrará su funcionamiento mediante la definición, construcción y uso de una réplica de dulzaina valenciana afinada en sol.

## 1.2 Alcance

Esta plataforma pretende ser de uso común para personas con los siguientes perfiles:

- Músicos experimentales

  Músicos que busquen nuevas formas de crear instrumentos y que deseen experimentar con algo más que el sonido, que deseen crear herramientas que actualmente no tienen disponibles para llevar a cabo sus actuaciones musicales, así como compartirlas con la comunidad.

- Estudiantes de niveles superiores de enseñanza obligatoria y preuniversitaria

  A los estudiantes que se están introduciendo en el mundo de la electrónica y la programación, con asignaturas como Tecnología, Tecnología Industrial o Informática, para que experimenten codificando instrumentos conocidos (empleados en otras materias) y editando el código generado, así como variando los componentes con los que formar sus prototipos.

- Personas relacionadas con el mundo DIY

  Cualquier persona interesada en el movimiento *hágalo usted mismo*, que le guste experimentar en el ámbito de la electrónica o de la música.

1 The Internet of Things (Dave Evans, Cisco) http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

## 1.3 Contexto

Existen ejemplos de instrumentos funcionando con las misma tecnología electrónica que se empleará para la realización de este proyecto y, por ello, sabemos que el *back-end* que emplearemos será capaz de ofrecer una respuesta adecuada. Como véase, esta ocarina http://www.youtube.com/watch?v=vwTeDGsEqCI o este xilófono http://www.youtube.com/watch?v=92VIEDtQKVI . Por otra parte y dada la popularidad del movimiento DIY en el momento actual, resulta muy económico hacer estos instrumentos con placas controladoras y elementos electrónicos, y los IDEs y lenguajes de desarrollo son sencillos y poseen una curva de aprendizaje muy pronunciada.

# 2. Contenido

## 2.1 Tecnologías

### 2.1.1 Traductor

El traductor será programado en Java para conseguir de forma sencilla que sea multiplataforma. Se generará mediante la herramienta ANTLR, y junto a la descripción de la gramática y las acciones invocadas en las reglas se emplearán clases auxiliares.

Según la web del proyecto, ANTLR se define como:

> "ANTLR, ANother Tool for Language Recognition, is a language tool that provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions containing actions in a variety of target languages"

Para introducirnos en la sintaxis de sus especificaciones, en formato Extended Backus-Naur Form (EBNF, vea enlaces en sección 4), una gramática ejemplo que reconoce los términos y factores de una expresión matemática sería la siguiente:

```
expr    : term ( ( PLUS | MINUS )  term )* ;
term    : factor ( ( MULT | DIV ) factor )* ;
factor  : NUMBER ;

NUMBER  : (DIGIT)+ ;
```

### 2.1.2 Microcontrolador

Se ha elegido la plataforma más extendida en estos momentos en el mercado para emplearla como objetivo de la traducción. Ésta es la plataforma Arduino (vea enlaces en sección 4). Arduino es una plataforma de hardware libre diseñada para convertirse en un estándar de facto para la comunicación y control de componentes electrónicos. Posee muchos complementos *plug & play* que permiten añadir características adicionales como las ya citadas placas lectoras de RFID o módulos Bluetooth.

Estos elementos se conectan a una serie de puertos de entrada y salida del tipo analógico y digital. Las placas Arduino poseen 13 puertos digitales y 6 puertos analógicos. En los puertos digitales podemos conectar elementos que poseen dos estados (como un botón, pulsado o sin pulsar; o un LED, encendido o apagado) o elementos que se comunican mediante algún protocolo serie (determinados sensores avanzados; pantallas o módulos de comunicación, como tarjetas WIFI o Bluetooth). En los puertos analógicos podemos conectar sensores que poseen multitud de estados (comúnmente 1024), como botones sensibles a la presión, sensores de intensidad de soplido o potenciómetros.

Tras realizar las conexiones obtendremos el valor del estado del sensor con las llamadas a la función de lectura, ya sea 1 ó 0 para los puertos digitales (bytes enteros en caso de comunicación serie), o un valor entre 0 y 1023 para los sensores analógicos.

Su entorno de desarrollo es muy sencillo, su código es libre y dispone de un gran número de librerías para hacer sencilla la programación de los módulos adheridos a la placa.

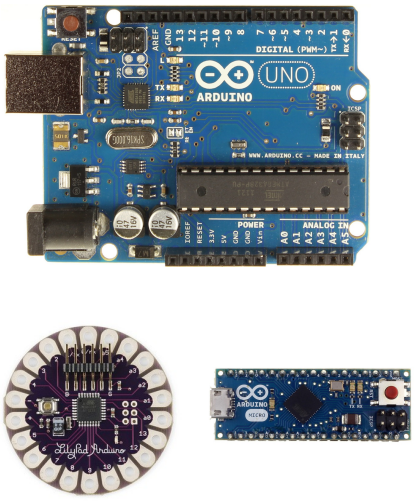



*Fig. 1. Diferentes modelos de placa Arduino*

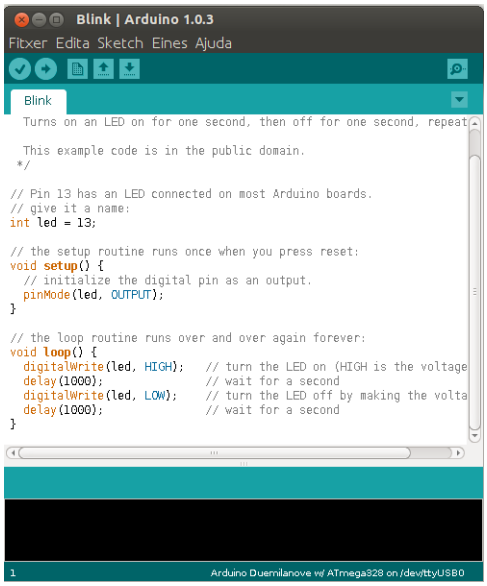



*Fig. 2. Entorno de desarrollo oficial*

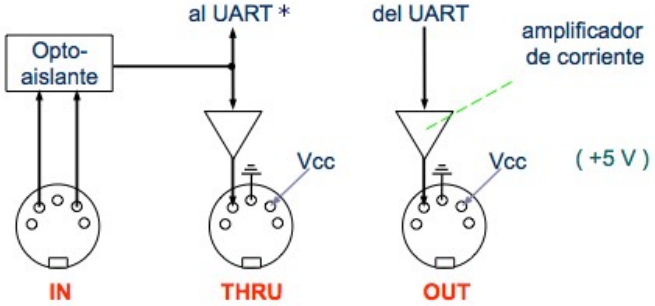
### 2.1.3 Protocolo de comunicación

Para comunicar el instrumento con diferentes elementos se empleará el protocolo MIDI. Este protocolo, especificado en 1983, se define en la Wikipedia como:

> **“**MIDI son las siglas de Interfaz Digital de Instrumentos Musicales. Se trata de un protocolo de comunicación serie estándar que permite a los computadores, sintetizadores, secuenciadores, controladores y otros dispositivos musicales electrónicos comunicarse y compartir información para la generación de sonidos.”

Un cable MIDI, según especifica el estándar, posee un conector DIN de 5 pines, de los cuales, únicamente uno de ellos se emplea para enviar datos (pin 5), los pines 2 y 4 se utilizan como entrada de 5V y tierra, respectivamente, y los pines restantes quedan sin función específica (vea figura 3).

Cada aparato MIDI puede tener hasta 3 puertos diferentes, dependiendo de la cantidad de características que soporte. Cada uno de estos está dedicado a enviar, recibir o transmitir, respectivamente, los diferentes mensajes serie.




**UART: Transmisor-Receptor Asíncrono Universal*

*Fig. 3: Esquema de conexiones de los diferentes puertos MIDI*

## 2.2 Lenguaje de especificación de instrumentos

Para facilitar la tarea de construcción de un instrumento se ha diseñado un lenguaje sencillo que sea inteligible para personas ajenas tanto al mundo de la informática y de la electrónica como del mundo de la música.

Este lenguaje es un lenguaje LL(1) (para más detalles, vea la siguiente sección). Analizaremos la sintaxis mediante la definición de un instrumento sencillo, con una entrada analógica (como, por ejemplo, un sensor de intensidad de soplido) y una entrada digital de dos estados (un pulsador).

El código del instrumento en este caso quedaría:

```
monophonic instrument test1:

	elements:
		sensorA connected to port A0 (0, 1023),
		butB connected to port 2.


	generators:
		sensorA (0,5) states:
			from 0 to 2 state silent,
			from 3 to 5 state blowing..

	modifiers:
		butB sounds DO3 with sensorA.silent and SOL4 with sensorA.blowing,
		default sounds LA4 with sensorA.blowing..
```

A simple vista, observamos que los ámbitos, que en muchos lenguajes imperativos se abren y cierran mediante los símbolos '{' y '}', en este caso se hace uso de los símbolos ':' y '.'. Las sentencias son finalizadas con el caracter ','. El uso de estos símbolos, junto con la sustitución de determinados símbolos por conectores comunes ('and', 'with') evitan un primer rechazo por parte del lector que no este familiarizado con los delimitadores empleados en lenguajes de programación tradicionales.

El código queda dividido en tres partes:

- Elements:
  En esta sección, se especifican todos los componentes que compondrán el instrumento y a qué puerto serán conectados

- Generators:
  En esta sección se describirán los elementos que sean de múltiples estados, que serán los que generen el sonido. En este caso tenemos la lectura de un sensor analógico de presión del soplido.

- Modifiers:
  En esta sección se describirán los elementos que modifiquen el sonido mediante un comportamiento binario, describiendo cuál es su comportamiento al ser pulsados, incluyendo combinaciones con otros elementos.

El código se inicia con una declaración del tipo de instrumento del que se trata (en este caso monofónico) y asignándole un nombre.

Encontramos ciertas particularidades, como la diferencia en la declaración de componentes (sección *elements)* de un elemento de dos estados *butB* con uno de múltiples estados *sensorA*, pues en el de múltiples estados especificamos el rango de posibles valores del sensor que queremos tratar. Los nombres de estos sensores son un identificador para referenciar en el resto del código el estado de cada uno de ellos.

Estos valores tendrán una correspondencia con los valores proporcionados en el intervalo descrito en la sección *generators* – (0,5) -, cuya finalidad es hacer más amable la discretización en estados de estas lecturas, de forma que se puedan abstraer de los valores que realmente indica el sensor. Para ayudar con esta abstracción se nombran los intervalos del estado del sensor. Esto nos será útil en la siguiente sección para asignar una respuesta a las acciones de entrada sin ninguna referencia directa a los componentes electrónicos. Por otra parte, establece una capa de abstracción útil en caso de que queramos reemplazar alguno de los sensores de entrada por otros de diferente sensibilidad.

En la última sección, *modifiers*, hacemos referencia al estado de los pulsadores simplemente nombrándolos, pues en caso de estar activos es cuando los consideraremos. Así mismo, asociaremos un estado de los generadores e indicaremos como respuesta la ejecución de una nota, la cual será la nota que será emitida mientras el estado de los sensores y pulsadores no cambie. La nota puede estar escrita en notación anglosajona o latina. Indicamos el sostenido con el caracter '#' y el bemol con la letra b minúscula 'b'. Tras el nombre de la nota, indicamos el número de octava al que pertenece, entre -1 y 9 (FA#3, FAb3, B#3).

Estas especificaciones pueden estar hechas en cualquier orden, y en caso de que existan dos posibles asociaciones con un mismo estado del instrumento, en caso de instrumentos monofónicos se elegirá la asociación que involucre más elementos en ella y en caso de instrumentos polifónicos se emitirán todas las notas correspondientes a cada una de ellas.

Al igual que podríamos reemplazar el sensor de viento por otro sensor de viento de diferente sensibilidad, también podríamos reemplazar ese mismo sensor por otro de otra clase, como un joystick o un potenciómetro, sin cambiar el código y obteniendo otro comportamiento radicalmente diferente, que podría imitar, en el caso del joystick, al frotado de una cuerda o, en el caso del potenciómetro, a la intensidad con que se aprieta la bolsa de una gaita.

## 2.3 Traductor

### 2.3.1 ASD

El traductor se ha descrito mediante un esquema de traducción dirigido por la sintaxis (ETDS) con sintaxis ANTLR. Un ETDS es una herramienta para diseñar traductores de una sola pasada, donde las acciones semánticas se sitúan en la parte derecha de la regla a emparejar.

Un traductor de una sola pasada es un traductor que necesita recorrer únicamente una vez el código en lenguaje fuente para generar el código objeto. ANTLR funciona mediante análisis sintáctico descendente (ASD). El ASD es descendente puesto que comienza con el símbolo inicial de la gramática y recorre todo el árbol de derivación hasta llegar a los nodos terminales, leyendo la cadena de entrada de izquierda a derecha, obteniendo derivaciones válidas por la izquierda (lenguajes de la clase LL) y, en nuestro caso, eligiendo qué regla aplicar con la vista de un único símbolo (LL(1)).

La gramática se ha sometido a estudio y revisión para cumplir todas las condiciones de la clase LL(1). Aquí presentaremos únicamente la gramática final.

Para más información sobre teoría de traductores y compiladores, consulte las referencias bibliográficas al final del documento.

### 2.3.2 Descripción

Dado el tamaño del ETDS, realizaremos una pequeña descripción basándonos en el instrumento de un sensor y un botón descrito en el capítulo anterior y repasando el análisis de las partes centrales, simplificadas, de cada uno de los bloques, para que sea posible entender cómo se forma el código traducido. En el anexo primero podrá encontrar la gramática y tokens del lenguaje, y en el repositorio indicado en la sección de enlaces el código completo del ETDS.

#### a. Análisis del bloque *elements*

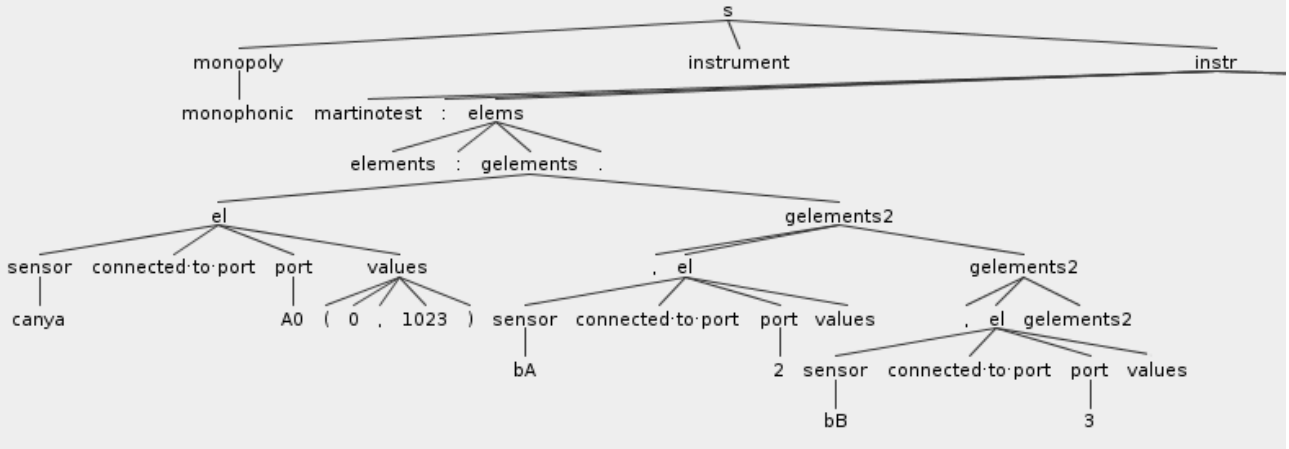



*Fig 4. Visualización del árbol de derivación del bloque 'elements', con el sensor 'canya' y los botones 'bA' y 'bB'*

Observamos cómo se expanden las reglas raíz, la definición de la clase de instrumento y las regla *el* tres veces describiendo los elementos que comentaremos. En las hojas observamos los nombres de las variables y puertos empleados.

Este fragmento se traducirá por declaraciones de variables enteras en las que almacenar los valores de los sensores y realizar las lecturas pertinentes. Observamos que la sentencia que asigna el valor a la variable *_canya* contiene código generado a partir del siguiente bloque y asigna a una variable auxiliar.

```
int canya;
int bA;
int bB;
//código omitido

_canya=map(analogRead(A0),0,1023,0,10);
//código omitido

bA=digitalRead(2);
bB=digitalRead(3);
```

Como ejemplo, ilustraremos con el código ANTLR cómo se realiza esta traducción para poder apreciar el uso de clases auxiliares empleadas en el proyecto para administrar la tabla de símbolos.

```
el      returns [String tcode, String readingDigitals]
        @init{
                Symbol element;
                $readingDigitals = "";
                $tcode = "";

        }: sensor CONNECTED port{
                values{
                        $tcode = "int " + $sensor.name + ";\n";
                        if($port.digital){
                                $readingDigitals+="\t"+$sensor.name+"=digitalRead("+
        $port.portn+");\n";
                        }

                        if($values.withValues){
                                element = new Symbol($sensor.name, $sensor.type,$port.portn,
        $values.min, $values.max);

                        } else {
                                element = new Symbol($sensor.name, $sensor.type,$port.portn);
                        }
                        table.addSymbol(element);
        };
```

### b. Análisis del bloque *generators*

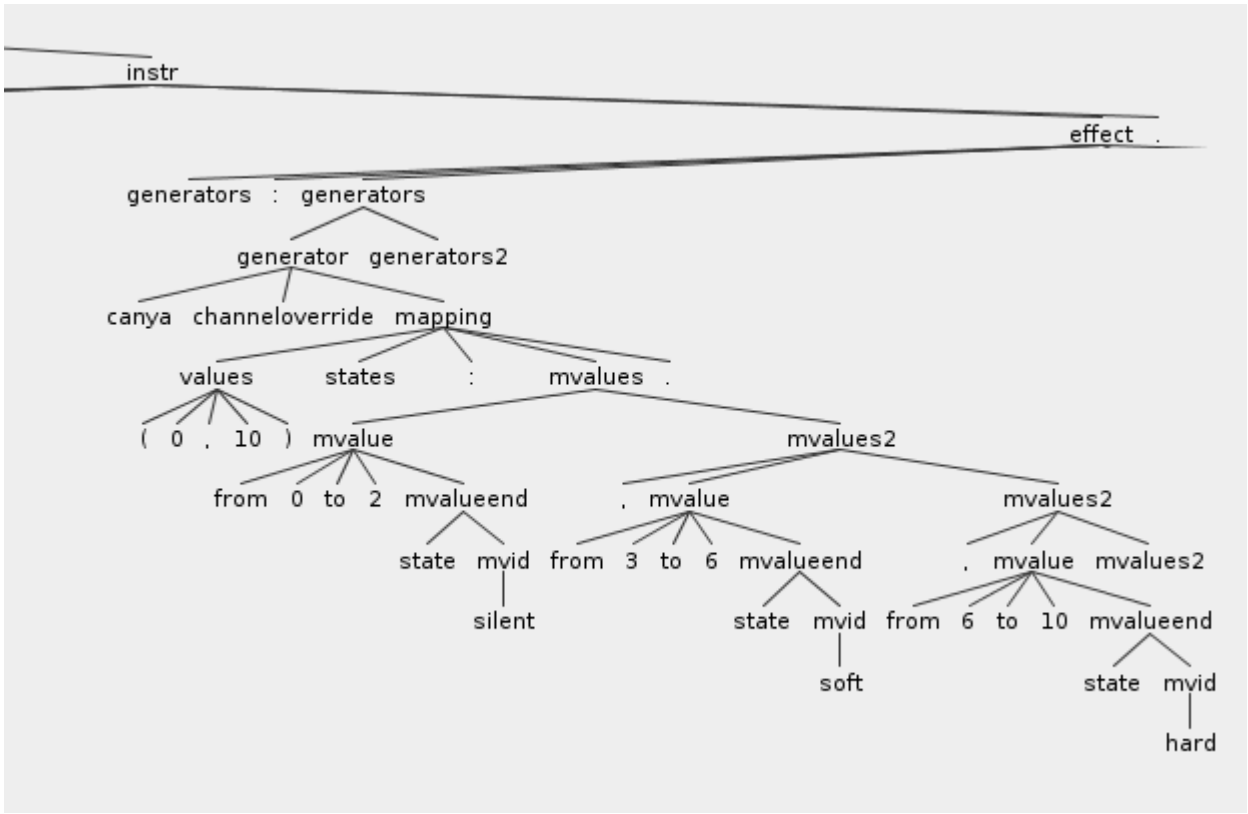



*Fig 5. Visualización del árbol de derivación del bloque 'generators'*

Observando el árbol de derivación podemos encontrar los diferentes estados descritos, así como observar que, al no haberse indicado el canal MIDI por el que el instrumento emitirá, esa regla ha derivado en vacío.

El código traducido afecta, una vez más, a la declaración de variables para manejar los diferentes estados en los que discretizaremos el rango de salidas posibles del sensor de intensidad y en la asignación de estos valores.

```
int _canya;
int canya_silent=0;
int canya_soft=1;
int canya_hard=2;

//código omitido

_canya=map(analogRead(A0),0,1023,0,10);
if(_canya>=0&&_canya<=2){
        canya=canya_silent;
}       else    if(_canya>=3&&_canya<=6){
        canya=canya_soft;
}else   if(_canya>=6&&_canya<=10){
        canya=canya_hard;
}
```

### c. Análisis del bloque *modifiers*

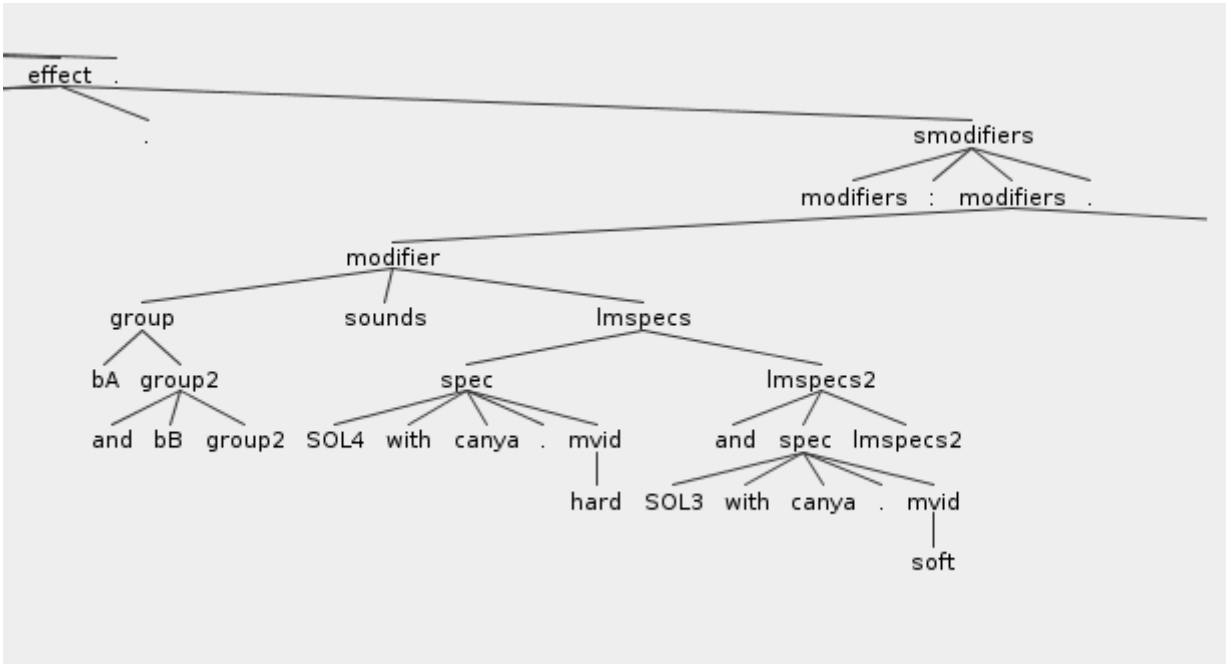



*Fig 6. Visualización del árbol de derivación del bloque 'modifiers'*

En este bloque observamos como se expanden cada una de las combinaciones de pulsaciones y valores del sensor que hemos descrito anteriormente. Para realizar la traducción se emplean clases auxiliares que reordenan las combinaciones, unifican las diferentes notaciones musicales y calculan el código de la nota a emitir. El resultado de la traducción es el siguiente:

```
void noteOn(int channel, int pitch, int velocity) {
        int noteOnCMD = 0x90;
        if(__notesOn[0]!=pitch){
                noteOff(__notesOn[0], pitch, velocity);
                __notesOn[0]= pitch;
                Serial.write(noteOnCMD+channel);
                Serial.write(pitch);
                Serial.write(velocity);
        }
}
//código omitido

} else if(bA==HIGH&&bB==HIGH&&canya==canya_soft){
                //Playing on channel 0 note SOL3 with normal speed
                noteOn(0x90+0,55,0x45);
} else if(bA==HIGH&&canya==canya_hard){
                //Playing on channel 0 note MI4 with normal speed
                noteOn(0x90+0,64,0x45);
}
//código omitido
```

Para entender cómo se ha formado esta traducción, revise el Anexo II con el código ANTLR encargado de ello.

## 2.4 Replicado ejemplo de un instrumento

Para no desvirtuar el objetivo de este proyecto con implementaciones de instrumentos extravagantes, se llevará a cabo la implementación de una réplica de una dulzaina valenciana afinada en sol.

La digitación estándar de una dulzaina valenciana se puede encontrar en el siguiente diagrama:

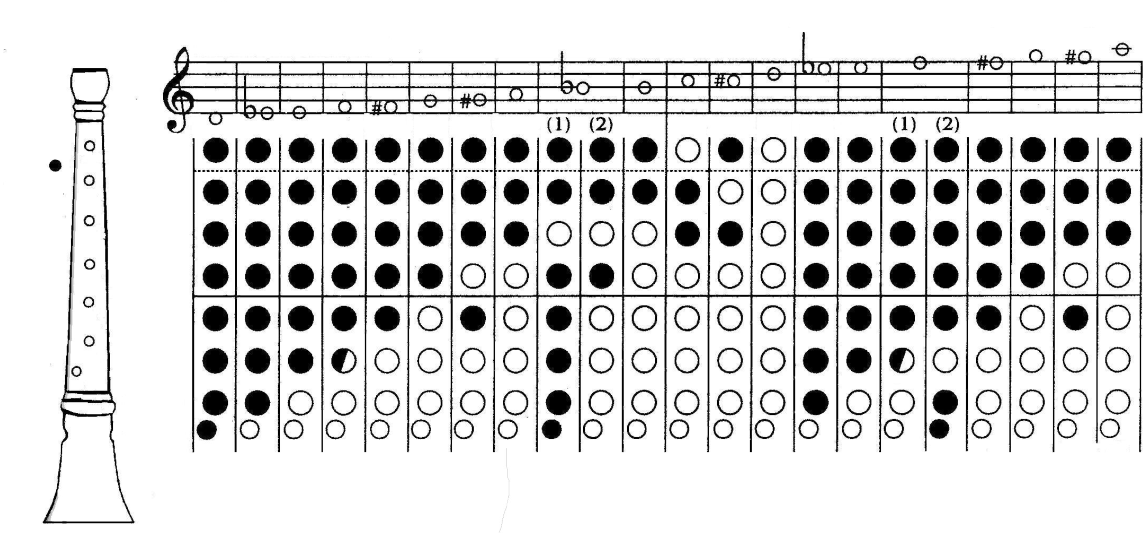



*Fig 7. Digitación de una dulzaina valenciana afinada en Sol*

En sustitución de la caña se empleará un sensor de intensidad absoluta de soplido, en concreto el Freescale MPX5010GP, con una sensibilidad de 0 kPa a 10 kPa, que abarca todo el espectro de presión con el que un adulto puede soplar (consulte bibliografía para más información).

Para reproducir la obturación de los orificios emplearemos 7 pulsadores grandes para los 6 orificios superiores delanteros y el orificio trasero, y 2 pequeños, uno para el orificio inferior y otro para complementar al tercer orificio (empezando por el lado inferior), puesto que en éste pueden darse dos posiciones, la posición de MI y la de FA#.

El siguiente esquema reproduce la totalidad de las conexiones necesarias para montar el instrumento:

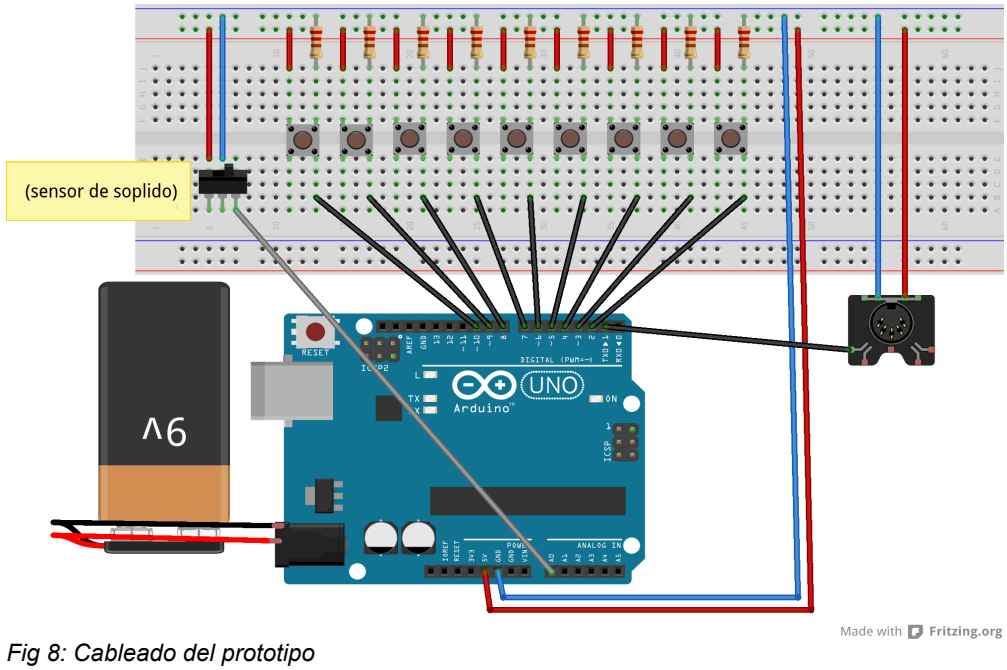



*Fig 8: Cableado del prototipo*

Elementos en la imagen:

- Baterías (9V aprox.), conectado a la toma de corriente
- 9 Botones, conectados a las entradas digitales 2 a 10 incluidas, a VCC y a GND mediante resistencia
- Placa MIDI conectada a los puertos digitales 0 y 1
- Sensor de intensidad de soplido, conectada a VCC, GND y el puerto analógico 0

Vista la tabla de digitalización y las conexiones, ahora podremos entender de forma mucho más sencilla el código que define este instrumento:

```
monophonic instrument martinotest:

        elements:
                canya connected to port A0 (0,1023),
                bSI connected to port 2,
                bLA connected to port 3,
                bSOL connected to port 4,
                bFA connected to port 5,
                bMI connected to port 6,
                bREs connected to port 7,
```

```
        bRE connected to port 8,
        bBack connected to port 9,
        bFAs connected to port 10.

generators:
        canya (0,10) states:
               from 3 to 6 state soft,
               from 6 to 10 state hard..
modifiers:
        bBack and bRE and bREs and bMI and bFA and bSOL and bLA and bSI sounds RE3
with canya.soft and RE4 with canya.hard,
        bBack and bREs and bMI and bFA and bSOL and bLA and bSI sounds RE#3 with
canya.soft and RE#4 with canya.hard,
        bBack and bMI and bFA and bSOL and bLA and bSI sounds MI3 with canya.soft and
        MI4 with canya.hard,
        bBack and bFA and bSOL and bLA and bSI sounds FA3 with canya.soft and FA4
with canya.hard,
        bBack and bSOL and bLA and bSI sounds SOL3 with canya.soft and SOL4 with
canya.hard,
        bBack and bLA and bSI sounds LA3 with canya.soft and LA4 with canya.hard,
        bBack and bSI sounds SI3 with canya.soft and SI4 with canya.hard,

        bBack and bFAs and bFA and bSOL and bLA and bSI sounds FA#3 with canya.soft
        and FA#4 with canya.hard,
        bBack and bRE and bREs and bMI and bFA and bLA and bSI sounds LA#3 with
canya.soft and LA#4 with canya.hard,

        bLA and bSOL sounds DO3 with canya.soft and DO#2 with canya.hard,
        bBack and bLA sounds DO#3 with canya.soft and DO#2 with canya.hard,

        default sounds RE3 with canya.soft and RE2 with canya.hard...
```

En las siguientes imágenes observamos la evolución desde el primer prototipo (prototipo de factibilidad) hasta el último prototipo (prototipo final).

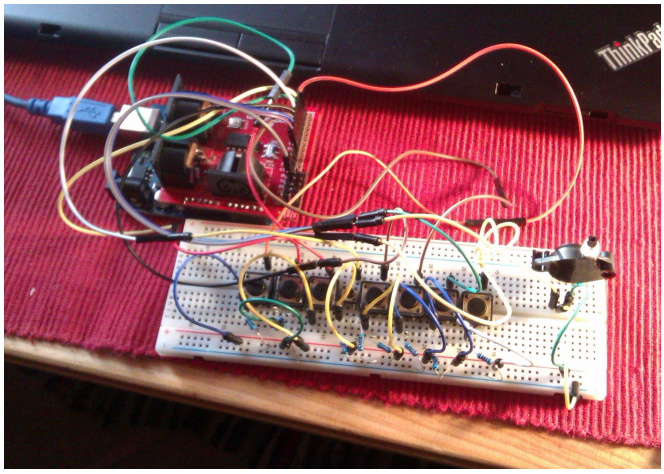


*Fig 9. Primera construcción del prototipo*

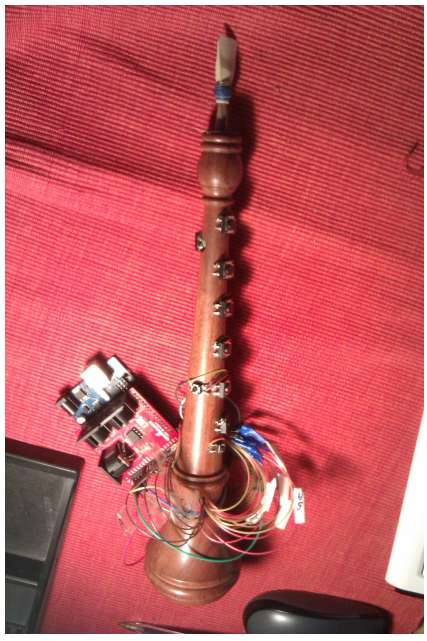


*Fig 10. Última construcción del prototipo*

## 2.5 Trabajo futuro

Este proyecto es un proyecto joven que necesita de un desarrollo de una envergadura mayor al realizado hasta el momento, dado que sus metas son muy ambiciosas: existen muchos tipos de instrumentos a crear y muchos componentes con los que crearlos. Las adiciones inmediatas que podrían realizarse sobre el trabajo de base son las siguientes:

**a. Mayor variedad de instrumentos**

Se debe añadir compatibilidad con instrumentos de percusión y cuerda explorando estos campos y complementando el lenguaje con nuevas o refinadas expresiones que mejoren la capacidad de descripción de estos.

**b. Compatibilidad con módulos de comunicación**

No sólo es importante crear compatibilidad con nuevos instrumentos, también con módulos de comunicación que establezcan un mecanismo de ejemplo de proceder en este aspecto. Creando un ejemplo de uso de módulos Bluetooth, por ejemplo, estableceremos el camino a seguir para la implementación de compatibilidad con módulos WIFI o 3G.

**c. Visualizadores de salida**

Como paso más allá, es interesante poder tener visualizadores (pantallas, LEDs numéricos, LEDs simples) conectados a nuestro instrumento, que nos den *feedback* del estado en el que se encuentra. Para ello deberíamos añadir expresiones que nos permitan interactuar con estos componentes.

**d. Modo de depuración**

Los mensajes MIDI, al componerse de series de bytes que forman códigos numéricos, son completamente ininteligibles a la hora de depurar errores sin ayudas externas. Además el protocolo MIDI emplea un *bitrate* (32150 Baudios) incompatible con el visor serie del IDE Arduino. Debería programarse un *flag* de *debug* para que el código ejecutable se oriente a su trazado y no a su emisión a un componente esclavo.

**e. Repositorio de instrumentos**

La intencionalidad de este proyecto es acercar la creación de instrumentos al público general. Por ello se debe tener una base de ejemplos y soluciones creativas que exploren todas las posibilidades, inspiren nuevas creaciones y orienten el desarrollo futuro de la plataforma. Para ello habría que formar un repositorio público de instrumentos.

**f. Creación de un IDE propio**

Imitando a los proyectos Processing y Arduino, que han llevado la programación de representaciones gráficas y la electrónica al público general con IDEs sencillos, concretos y con ejemplos precargados, una buena forma de incluir el acceso a los repositorios, resaltado y autocompletado de código, depuración y consejos de construcción es disponer de un IDE para la plataforma.

# 3. Conclusiones

Este proyecto es un punto de encuentro entre varios movimientos que, actualmente, se encuentran en su momento de mayor viveza. El mundo de la música experimental, que, en la escena valenciana, mira de reojo a la música tradicional; y el movimiento Maker que vive su época de esplendor con elementos como el Internet de las Cosas, las impresoras 3D o el Raspberry Pi.

Para complementar este punto de encuentro, se propone una herramienta que ha resultado eficaz y útil: el lenguaje. El diseño de traductores de pequeños lenguajes que faciliten tareas técnicas especializadas no está al alcance de cualquier principiante en el mundo de la tecnología, y, sin embargo, produce unos resultados que confieren mayor sencillez y agilidad para la creación que otras herramientas interactivas e incluso gráficas. No solo en el campo de la música, sino en cualquier otro, podemos crear una capa de abstracción que invite a la gente a acercarse a esos nuevos mundos de la forma menos traumática y más atractiva posible.

Tras analizar los resultados finales, sin duda, hacer uso de esta tecnología para la construcción de instrumentos “caseros” o “Frankenstein” no sólo resulta más creativo que comprarlos a la industria tradicional, también mucho más económico, puesto que los componentes (sin contar el soporte de madera) que componen este instrumento cuestan, dependiendo del proveedor, entre 25€ y 35€, y pueden ser reutilizados para la construcción de diferentes variaciones. El protocolo MIDI hace de calzador para que la funcionalidad de nuestras creaciones esté a la altura de los productos comerciales.

Por otra parte y, respecto a las tecnologías empleadas, la diferencia en la técnica a la hora de trabajar en alto nivel (la creación del traductor) y bajo nivel (el código del microcontrolador) resulta un inconveniente para aquel que no esté acostumbrado a trabajar en los dos mundos. Un equipo formado por uno o más especialistas de cada uno de los ámbitos puede llegar a diseñar una arquitectura de traductor menos orientada al microcontrolador y unos resultados de traducción menos monolíticos y más preparados para la adición de características futuras.

Tenemos, en definitiva, una herramienta para crear y compartir instrumentos musicales y un ejemplo del uso de nuevos lenguajes *ad-hoc* como alternativa al entorno gráfico para desarrollar soluciones a problemas específicos.

# 4. Bibliografía y enlaces de interés

## 4.1 Bibliografía

*Diseño de compiladores*, A. Garrido Alenda, J.M. Iñesta Quereda, F. Moreno Seco, J.A. Pérez Ortiz. Universidad de Alicante (2002).

*Introducció a la Teoria de Llenguatges Formals,* R.C. Carrasco, M.L. Forcada. Inédito.

*Physiology of Respiration*, Michael P. Hlastala, Albert J. Berger. Oxford University Press (2001).

## 4.2 Enlaces de interés

Código del proyecto (públicamente llamado Proyecto Martino):
https://code.google.com/p/martino-project/

*Arduino Language Reference*
http://arduino.cc/en/Reference/HomePage

ANTLR
http://www.antlr.org/

Notación EBNF
http://es.wikipedia.org/wiki/Notaci%C3%B3n_de_Backus-Naur

*MIDI protocol guide*, Hinton Instruments
http://hinton-instruments.co.uk/reference/midi/protocol/index.htm

*MIDI Messages*, MIDI Manufacturers Association
http://www.midi.org/techspecs/midimessages.php

*MIDI tutorial*, Arduino Examples
http://arduino.cc/en/Tutorial/Midi

*MIDI output using Arduino*, New York University
http://itp.nyu.edu/physcomp/Labs/MIDIOutput

*Note names, MIDI numbers and frequencies*, University New South Wales
http://www.phys.unsw.edu.au/jw/notes.html

*HIDUINO*, MTIID
http://mtiid.calarts.edu/research/hiduino

*4bitsynth*, a MIDI-controlled digital synthesizer using Atmel AVR ATMega48.
https://code.google.com/p/4bitsynth/

*Dulzaina,* Wikipedia
https://es.wikipedia.org/wiki/Dulzaina

## Anexo I: Gramática y tokens del lenguaje de definición de instrumentos

```
s
        : monopoly INSTRUMENT  instr ;
monopoly
        : MONOPHONIC  | POLYPHONIC;
instr
        : ID COLON elems effect DOT | ;
effect
        : GENERATORS COLON generators DOT smodifiers |;
elems
        : ELEMS COLON gelements DOT | ;
gelements
        : el gelements2;
gelements2
        : COMMA el gelements2 | ;
el
        : sensor CONNECTED port values ;
port
        : INT | AINT;
sensor
        : ID;
values
        : PARI INT COMMA INT PARD | ;
generators
        : generator generators2       | ;
generators2
        : COMMA generator generators2 | ;
generator
        : ID channeloverride mapping;
channeloverride
        : CHANNEL INT | ;
mapping
        : values STATES COLON mvalues DOT | ;
mvalues
        : mvalue mvalues2 ;
mvalues2
        : COMMA mvalue mvalues2 | ;
mvalue
        : FROM INT TO INT mvalueend ;
mvalueend
        : STATE mvid;
mvid
        : INT | ID ;
smodifiers
        : MODIFIERS COLON modifiers DOT | ;
modifiers
        : modifier modifiers2 | ;
modifiers2
        : COMMA modifier modifiers2   | ;
modifier
        : group SOUNDS lmspecs | ID PITCHES;
group:
```

```
        ID group2 | DEFAULT;
group2:
        AND ID group2;
lmspecs:
        spec lmspecs2;
lmspecs2:
        AND spec lmspecs2;
spec:
        NOTE WITH ID DOT mvid;


MODIFIERS
        : 'modifiers';
SOUNDS  : 'sounds';
STATE   : 'state';
PITCHES : 'is pitch bending';
SEMITONES
        : 'semitone' | 'semitones';
STATES  : 'states';
FROM    : 'from';
TO      : 'to';
AND     : 'and';
WITH    : 'with';
CONNECTED
        : 'connected to port';
CHANNEL : 'airs on channel' ;
ELEMS   : 'elements' ;
GENERATORS
        : 'generators';
DEFAULT : 'default';
MONOPHONIC
        : 'monophonic';
POLYPHONIC
        : 'polyphonic';
INSTRUMENT
        :  'instrument';
PARI    : '(';
PARD    : ')';
COLON   : ':';
DOT     : '.';
COMMA   : ',';

AINT    : ('A'|'a')('0'..'9')+
        ;
NOTE    : ('A'..'G'|'DO'|'RE'|'MI'|'FA'|'SOL'|'LA'|'SI'|'TI')('b'|'#')?('-1'|'0'..'9');
ID      : ('a'..'z'|'A'..'Z') ('a'..'z'|'A'..'Z'|'0'..'9')*;
INT     : ('0'..'9')+
        ;
```

## Anexo II: Código de traducción del bloque *modfiers* (reglas principales únicamente)

```
modifier returns [String pitchbend]
	: group SOUNDS lmspecs{
		$pitchbend = "";
		int numcombinations = $lmspecs.notes.size();
		Combination comb;
		for(int i=0; i< numcombinations; i++){
			comb = $group.comb.clone();
			comb.note = $lmspecs.notes.get(i);
			comb.attachedSymbol = $lmspecs.generatorsl.get(i);
			comb.attachedState = $lmspecs.states.get(i);
			combinations.elementCombinations.add(comb);
		}
	}| ID PITCHES {
		$pitchbend = "";
		Symbol bender = table.get($ID.text);
		if(bender!=null){
			$pitchbend = "\n\tint _"+$ID.text+" = map(analogRead(" + bender.port
+"),0,1023,0,16383);\n";
			$pitchbend += "\tif(_"+$ID.text+"!="+$ID.text+"){\n";
			$pitchbend += "\t\t"+$ID.text + "=_"+$ID.text+";\n";
			$pitchbend += "\t\tpitchbend("+bender.channel+","+$ID.text+");\n";
			$pitchbend += "\t}\n";
		}
	};

smodifiers returns [String tcode]
	: MODIFIERS COLON modifiers DOT {
		String elseTxt = "";
		Symbol generator = null;
		$tcode = "\n\t";
		if(combinations.elementCombinations.size()>0){
			combinations.sort();
			for (Combination c: combinations.elementCombinations){
				$tcode+=elseTxt;
				$tcode+="if(";
				for(String s: c.elements){
					$tcode+=s+"==HIGH&&";
				}
				generator = table.get(c.attachedSymbol.name);
				$tcode += c.attachedSymbol.name
			+"=="+c.attachedSymbol.name+"_"+c.attachedState+"){\n";
				$tcode += "\t\t//Playing on channel "+generator.channel+
			" note "+c.note+" with normal speed\n";
				$tcode += "\t\tnoteOn(0x90+"+generator.channel+"," +
				noteManager.get(c.note)+",0x45);\n";
				if(polyphonic==1){
					elseTxt = "\t}\n\t";
				} else {
					elseTxt = "\t} else ";
				}
			}
			$tcode+="\t}";
			if(polyphonic==0){
				$tcode+="else {\n\t\tsoundOff("+generator.channel+");\n\t}";
			}
		} else {
			$tcode = "";
		}

	}| { $tcode = "";};
```

**Puede encontrar el código completo en https://code.google.com/p/martino-project/**