# Software NOMBRE

Desarrollo de una interfaz de programación para talleres de Robótica Educativa

14 de abril de 2011

# Índice general

# Capítulo 1

# Survey

Realizamos un survey de interfaces gráficas de programación.

Estos son todos proyectos para niños, la mayoría implementados para la OLPC (One Laptop Per Child Project):

- StarLogo TNG
  http://education.mit.edu/drupal/starlogo-tng
- EToys (Smalltalk/Squeak)
  http://wiki.laptop.org/go/Etoys
  http://www.squeakland.org/download/
- Scratch (Squeak)
  http://scratch.mit.edu/

También hay un framework de Microsoft:

- Microsoft Robotics Developer Studio (RDS)
  http://msdn.microsoft.com/en-us/library/cc998476.aspx
- Microsoft Visual Programming Language (VPL)
  http://msdn.microsoft.com/en-us/library/bb483088.aspx
- Microsoft Visual Simulation Environment (VSE)
  http://msdn.microsoft.com/en-us/library/bb483076.aspx

# Capítulo 2

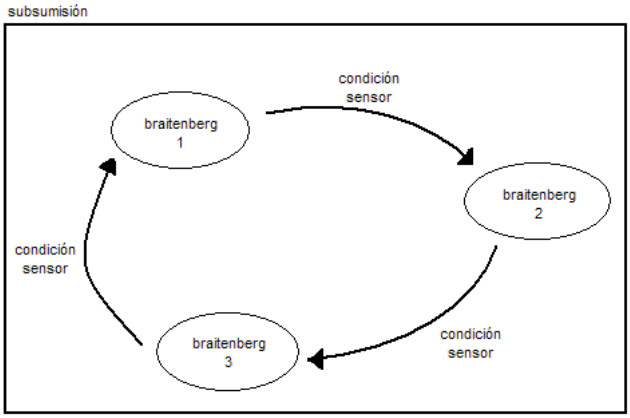
# ERBPI: Easy Robot Behaviour Programming Interface

La idea general del software es que permita, a través de una interfaz gráfica y sencilla, programar los robots para realizar distintas experiencias de vehículos de Braitenberg y comportamiento basado en subsumisión[1].

Para eso, nos basamos en el survey que realizamos, principalmente los programas *StarLogo* y *Scratch*, que resultaron los mejorcitos en cuanto a la interfaz gráfica de programación y la idea de la interfaz gráfica de proveer menus y submenúes con los objetos predefinidos para ir agregando...

La idea es poder combinar *Braitenberg* y *Subsumisión*, de manera que cada estado de la maquina de estados de subsimisión sea un braitenberg. Entonces, en principio, se puede hacer sólo Braitenberg. Luego, se puede hacer Subsumisión "insertando" en cada estado un braitenberg definido anteriormente.

Algo así:



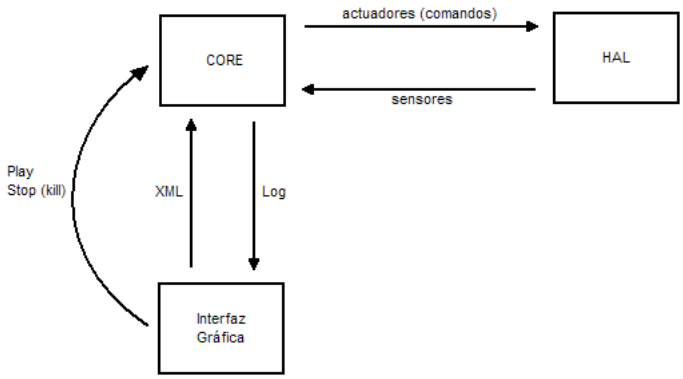

[1] *Subsumption Architecture*, Behavior-Based Robotics, R. C. Arkin.

# Capítulo 3

# Diseño

El software estaría compuesto por tres módulos independientes. Un módulo *Core*, un módulo *RAL* y un módulo *GUI*. Esto nos permite realizar el desarrollo de cada módulo completamente por separado.

Algo así:




## 3.1. Core

Lo hacemos en `C++` para que sea lo más rápido y potente posible.

Para la funcionalidad de parseo del XML utilizamos el *Xerces XML Parser 3.0.1*[1] para `C++`. Recompilamos las librerías del Xerces de forma estática para que estén incluidas en el ejecutable del *Core* y no sea necesario transportalas.

El parseo del XML es bastante estricto y flexible al mismo tiempo. Sólo se obtienen del XML los datos necesarios para la ejecución del Core, cualquier otro atributo o especificación son ignorados. De esta forma, el XML podría contener información adicional, que el Core ignorará, pero que serviría para otros módulos como la *GUI*. Para más detalle sobre la estructura que el Core obtiene del XML, ver "XML - Datos para la Ejecución del Core" en el punto 3.4.1 en la página 20.

El Core debería, a grandes rasgos, hacer las siguientes cosas:

1. **Parsear el XML:** Levantar el archivo XML, chequear que no haya elementos repetidos por *id*, chequear que no existan ciclos en el grafo formado por los sensores + cajas + actuadores, chequear que los predecesores de cada elemento sean elementos existentes de modo que el grafo sea consistente,

[1] *Xerces XML Parser 3.0.1*, http://xerces.apache.org/xerces-c/

realizar un *topological sorting*[2] del grafo, y por último, devolver la *tabla de orden de ejecución secuencial* con la cual se realizará toda la ejecución posterior.

2. **Chequear que los sensores y actuadores del Core y el RAL se correpondan entre sí:** Obtener la lista de sensores y actuadores del *RAL* y realizar el chequeo de que el *RAL* contenga los sensores y actuadores que tiene el *Core*.

3. **Definir la frecuencia de trabajo:** Obtener la *frecuencia de trabajo* del *RAL* para luego ejecutar como máximo a esta *frecuencia*.

4. **Ejecutar:** Obtener el nuevo estado (valor) de los sensores del *RAL* y actualizar sus valores en la *tabla de orden de ejecución*, para cada elemento de la *tabla de orden de ejecución* actualizar sus valores en función de sus predecesores, y por último, enviarle al *RAL* el nuevo valor para los actuadores y actualizar el archivo de *LOG* con el valor de todos los elementos de la tabla de orden de ejecución.

**Observaciones:**

- En el parseo, no se chequea que una *Caja* no tenga como entrada a un *Actuador*.
- En el *LOG*, no se guarda el estado de sensores del *RAL*, se desprenden de la *tabla de orden de ejecución*.
- En el chequeo entre sensores y actuadores del *Core* y el *RAL*, no se chequea que estén en el mismo orden, sólo que los sensores y actuadores del *Core* sean un subconjunto de los sensores y actuadores del *RAL*

### 3.1.1. Ejecución en Linux 32 bits

Para ejecutar el *Core* se deberán especificar 3 parámetros:

1. **ArchivoXML:** El archivo *XML* a parsear.
2. **ArchivoLOG:** El archivo de *LOG* donde se guardará el *log* de la ejecución.
3. **RAL_ID:** la especificación del *RAL* que se utilizará.

Por ejemplo: `./core ArchivoXML.xml ArchivoLOG.log RAL_ID`

donde `RAL_ID` podría ser: *exabot*, *khepera*, *yaks*, etc.

#### Ejecución del Core junto con el RAL

Como ya dijimos, el *Core* se encuentra compilado con una *librería dinámica* del *RAL*. Por lo tanto, es necesario indicarle al *sistema operativo* dónde buscar la librería dinámica `libRAL.so` cuando el *Core* llame a funciones de la misma. De lo contrario, la ejecución falla.

La forma de hacer esto en *Linux* es, en la misma consola donde se ejecutará el *Core*, ejecutar las siguientes dos líneas para agregar al sistema operativo un *path* para la búsqueda de librerías:

```
# LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<path>
# export LD_LIBRARY_PATH
```

donde <*path*> debe ser la ruta (absoluta) donde se encuentra la librería dinámica *libRAL.so*, por ejemplo:

[2] *Topological Sort, Introduction to Algorithms*, Cormen, Leiserson, Rivest, and Stein.

```
# LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/usuario/desktop/soft_src/ral/src
```

También podría modificarse el archivo de configuración del usuario `.profile` para agregar esta ruta de forma permanente. Para más información sobre el manejo de librerías dinámicas en *Linux*, puede consultarse http://www.chuidiang.com/clinux/herramientas/librerias.php

**IMPORTANTE !!:** Por ahora las librerías del RAL se llaman todas iguales libRAL.so y vamos pisando con la que corresponde en la carpeta de ejecución del Core... Luego, hay que hacer un "if" en el Core, para que cargue en tiempo de ejecución la librería que corresponda (*libRAL-yaks.so* o *libRAL-exabot.so*). También va a ser necesario tocar unas cositas en el RAL para que esto quede bien. Diego sabe bien cómo hacer lo de cargar la librería correspondiente en tiempo, preguntarle!!! FALTA HACER ESTO !!!!!

La forma de hacer esto en *Windows* es, FALTA COMPLETAR ESTO !!!!!

### 3.1.2. Finalización en Linux 32 bits

Al comenzar a ejecutar el *Core*, el mismo entra en un *ciclo infinito* en el que va ejecutando y actualizando los valores de todos los elementos.

Para terminar la ejecución del *Core*, el mismo tiene definida un *rutina de atención de señales*, en particular, para la señal `SIGINT`, que al ser recibida por el *Core* termina su ejecución de forma ordenada, cerrando correctamente el archivo de *log*.

Las señales se encuentran definidas en la librería estandar $< signal.h >$. Puede verse su especificación en http://publications.gbdirect.co.uk/c_book/chapter9/signal_handling.html. La señal `SIGINT`, es una señal de atención interactiva, generalmente generada por la teclas `Ctrl+C` en la consola de ejecución, pero que también puede ser enviada por otro programa.

La idea es que sea la *GUI* la que inicia la ejecución del *Core* y la que termine la ejecución del mismo enviando la señal `SIGINT`.

### 3.1.3. Estructuras

Para realizar el manejo en el *Core* de los sensores, cajas y actuadores, se utilizarán los siguientes objetos (`C++ class`):

```
clase Elemento (es una clase abstracta)
    atributos:
        string _id
        int _valor        // es el valor de salida
        int _entrada      // es la sumatoria de todas sus entradas
        int _tipo         // tipo: sensor, caja o actuador

    métodos:
        string getId()    // devuelve _id
        int setValor(int) // setea _valor y lo devuelve
        int getValor()    // devuelve _valor
        int getEntrada()  // devuelve _entrada
        int ejecutar()    // es virtual, llama a la de la clase hija...

clase Sensor (hereda de clase Elemento)
    atributos:
        // ninguno
    métodos:
        int ejecutar()    // devuelve valor de Elmemento::_valor

clase Caja (hereda de clase Elemento)
    atributos:
        vector<Elemento*> _entradas;
        vector<Punto> _puntos;
```

```
    métodos:
        int ejecutar()    // para cada "i" en _entradas, acumula *(_entradas[i]).getValor(),
                          // realiza resultado = función(acumulador), y luego, setea
                          // el "resultado" en Elemento::_valor, setea el "acumulador"
                          // en Elemento::_entrada y devuelve Elemento::_valor

clase Actuador (hereda de clase Elemento)
    atributos:
        vector<Elemento*> _entradas;
    métodos:
        int ejecutar()    // para cada "i" en _entradas, acumula *(_entradas[i]).getValor()
                          // luego setea el acumulador en Elemento::_valor
                          // y devuelve Elmemento::_valor

struct Punto
  int x;
  int y;

string es la clase de C++ STL.
vector es la clase de C++ STL.
```

De esta forma, la *tabla de orden de ejecución secuencial* será un vector de la clase `Elemento`:

```
vector<Elemento> TablaEjecucion
```

Así, la ejecución sólo consistirá en recorrer la tabla secuencialmente y, por cada elemento, realizar el `ejecutar()` que se encargará de obtener los valores requeridos en $\mathcal{O}(1)$, ya que cada elemento contiene *punteros* a los elementos que le son predecesores: `TablaEjecucion[i].ejecutar()`

**Observaciones:**

- El método `Caja::ejecutar()`, asume que los 2 puntos están ordenados, es decir, `Caja._puntos[0].x` $\leqslant$ `Caja._puntos[1].x`. Por lo tanto, calcular el valor de la función se reduce a 3 casos:
  - $entrada \leqslant x_0 \implies resultado = y_0$
  - $entrada \geqslant x_1 \implies resultado = y_1$
  - $x_0 < entrada < x_1 \implies resultado = \frac{y_0 - y_1}{x_0 - x_1} \times (entrada - x_0) + y_0$ (ecuacion de la recta)
- Por como están diseñadas las clases, y posteriormente los algoritmos, obliga a que los atributos `_entradas` y `_puntos` sean públicos. De lo contrario, habría que especificarlos como privados y especificar sus métodos correspondientes. Que sean atributos públicos y no tengan sus métodos correspondientes, hace que cualquier función pueda modificar a su antojo cualquier atributo de la clase, y que, en los algoritmos como el Parser, para crear los elementos haya que agregar las `&(Elemento)` en `_entradas` y los puntos en `_puntos` manualmente...

### 3.1.4. Pseudocódigos

***Core***

*Preprocesamiento:*
vector<Elemento> TablaEjecucion ⟵ Parsear(ArchivoXML, TablaEjecucion)
**if** (ids sensores en TablaEjecucion) $\nsubseteq$ (ids RAL.getListaSensores()) **then**
  Error: los sensores no se corresponden y Terminar
**end if**
**if** (ids actuadores en TablaEjecucion) $\nsubseteq$ (ids RAL.getListaActuadores()) **then**
  Error: los actuadores no se corresponden y Terminar
**end if**
frecuencia ⟵ RAL.getFrecuenciaTrabajo()
*Ejecución:*

```
while frecuencia lo permita do
  vector<<id;valor>> sensoresRAL ⟵ RAL.getEstadoSensores()
  actualizar el nuevo valor de cada sensor en TablaEjecucion con sensoresRAL
  for cada elemento i de TablaEjecucion do
    TablaEjecucion[i].ejecutar()
  end for
  vector<<id;valor>> actuadoresRAL ⟵ generado con los actuadores de la TablaEjecucion
  RAL.setEstadoActuadores(actuadoresRAL)
  LOG ⟵ actualizar con TablaEjecucion
end while
```

***Parsear(in ArchivoXML, inout TablaEjecucion)***

```
vector<<tipo:char; id:string; entradas:vector<string>; puntos:vector<<int;int>>>> vectorAuxiliar
⟵ generado con cada elemento (sensor, caja o actuador) parseado de ArchivoXML
for cada elemento i en vectorAuxiliar do
  if ∃j, j ≠ i / vectorAuxiliar[j].id = vectorAuxiliar[i].id then
    Error: hay IDs repetidos y Terminar
  end if
end for
for cada elemento i en vectorAuxiliar do
  for cada elemento j en vectorAuxiliar[i].entradas do
    if ∄k, 0 ⩽ k < long(vectorAuxiliar) / vectorAuxiliar[k].id = vectorAuxiliar[i].entradas[j] then
      Error: hay elementos que tienen "entradas" que no existen y Terminar
    end if
  end for
end for
if HayCiclos(vectorAuxiliar) then
  Error: el grafo contiene ciclos y Terminar
end if
vectorAuxiliar ⟵ TopologicalSorting(vectorAuxiliar)
vector<Elemento> TablaEjecucion ⟵ vacío
for cada elemento i en vectorAuxiliar do
  if vectorAuxiliar[i].tipo = Sensor then
    nuevo sensor(vectorAuxiliar[i].id)
    sensor.setValor(0)
    agrego el sensor en TablaEjecucion al final
  end if
  if vectorAuxiliar[i].tipo = Caja then
    nuevo caja(vectorAuxiliar[i].id)
    for cada elemento j en vectorAuxiliar[i].puntos do
      agrego vectorAuxiliar[i].puntos[j] en caja._puntos al final
    end for
    for cada elemento j en vectorAuxiliar[i].entradas do
      for cada elemento k en TablaEjecucion (0 ⩽ k ⩽ i) do
        if vectorAuxiliar[i].entradas[j] = TablaEjecucion[k].getId() then
          agrego &(TablaEjecucion[k]) en caja._entradas al final
        end if
      end for
    end for
    agrego la caja en TablaEjecucion al final
  end if
  if vectorAuxiliar[i].tipo = Actuador then
    nuevo actuador(vectorAuxiliar[i].id)
    for cada elemento j en vectorAuxiliar[i].entradas do
      for cada elemento k en TablaEjecucion (0 ⩽ k ⩽ i) do
```

```
            if vectorAuxiliar[i].entradas[j] = TablaEjecucion[k].getId() then
              agrego &(TablaEjecucion[k]) en actuador._entradas al final
            end if
          end for
        end for
        agrego el actuador en TablaEjecucion al final
      end if
    end for
    return TablaEjecucion
```

**Observaciones:** Por el momento, al generar el `vectorAuxiliar` se chequea que la *caja* tenga definidos exactamente 2 puntos, de lo contrario, termina con *ERROR*.

## 3.2. RAL - Robot Abstraction Layer

La idea es que sea una capa de abstraction respecto del hardware específico que hay del otro lado, es decir, qué tipo de robot, simulador, qué tipo y cantidad de sensores y actuadores, etc. Por lo tanto, para el *Core* va a ser transparente, sólo se comunicará con el *RAL* para recibir el estado de los sensores y enviar el nuevo estado para los actuadores. Luego, será el *RAL* el que se comunicará directamente con el hardware o simulador según corresponda (Khepera, ExaBot, Yaks, etc.).

Lo hacemos en `C++` como una *librería dinámica multiplataforma* (.DLL o .SO) para interactuar directamente con el *Core*, sin la necesidad de recompilar el *Core* para distintos *RALs*. Luego, para interactuar con otro robot o simulador, simplemente se le especificará por línea de comandos al *Core* cuál será el *RAL_ID* que se utilizará.

Por lo tanto, la *librería diámica del RAL* será una sola, y el mismo *RAL* deberá poder diferenciar sobre qué hardware deberá trabajar... ¿ESTO LO HACEMOS CON UN PARÁMETRO? ¿ESTE PARÁMETRO DEBERÍA IR EN C/U DE LAS FUNCIONES DEL RAL? RESOLVER ESTO...

Debería hacer las siguientes cosas:

- **getListaSensores().** Devolver una lista de IDs de los sensores que posee el hardware o simulador que se se está utilizando.
- **getListaActuadores().** Devolver una lista de IDs de los actuadores que posee el hardware o simulador que se se está utilizando.
- **getEstadoSensores().** Devolver una lista de <id;valor> con el nuevo estado de cada sensor del hardware o simulador que se se está utilizando.
- **getFrecuenciaTrabajo().** Devolver a qué frecuencia sensa y es posible asignarle a los actuadores el hardware o simulador que se se está utilizando, para que el *Core* lo tenga en cuenta y trabaje a esta frecuencia como máximo...
- **setEstadoActuadores().** Recibir una lista de <id;valor> con el nuevo valor para cada actuador y actualizar los actuadores en el hardware o simulador que se se está utilizando.
- **inicializarRAL().** Inicializar el hardware o simulador que se se está utilizando.
- **finalizarRAL().** Finalizar el hardware o simulador que se se está utilizando.
- **Comunicación con el Hardware.** Realizar la conexión por software con el hardware específico o simulador que se utilizará y enviar los comandos correspondientes para que se mueva...

### 3.2.1. RALs

Por el momento, la idea es tener las siguientes *RALs*:

- RAL Yaks Linux
- RAL Yaks Windows
- RAL Khepera Linux - para cable serial
- RAL Khepera Windows - para cable serial
- RAL Khepera Linux - para radio
- RAL Khepera Windows - para radio
- RAL ExaBot Linux
- RAL ExaBot Windows
- RAL CheBot Linux
- RAL CheBot Windows

### 3.2.2. Normalización de Mediciones

El RAL debe normalizar las mediciones de los sensores entre 0 y 1 !!!!

FALTA HACER ESTO EN EL RAL !!!!

## 3.3. GUI - Graphical User Interface

Este módulo se encarga de la interfaz con el usuario y su función principal es la de permitir la programación gráfica del comportamiento del robot. El módulo GUI cuenta con las siguientes funcionalidades:

- Permitir en modo gráfico diseñar el modelo de Braitenberg mediante la interconexión de sensores con actuadores. Cada una de estas conexiones debe implementar funciones matemáticas parametrizables. De esta forma se define un *grafo de ejecución* que representa el comportamiento a realizar, donde los nodos son sensores, actuadores o funciones matemáticas. En la Figura 3.1 se muestra esta idea.

Figura 3.1: Un ejemplo de grafo de ejecución

- Permitir en modo gráfico diseñar una arquitectura de subsumisión para coordinar los distintos comportamientos.
- Realizar chequeos para validar los comportamientos diseñados y su coordinación.
- Guardar en un archivo el comportamiento diseñado y la configuración de sensores y actuadores requerida en un robot para poder llevar adelante ese comportamiento. Este archivo será leído y ejecutado por el Core.
- Ejecutar la aplicación, indicándole al CORE cuándo iniciar y finalizar la ejecución del comportamiento.
- Guardar y cargar configuraciones de distintos robots (sensores y actuadores).
- Realizar un replay de la experiencia, utilizando para ello un archivo generado por el Core durante la ejecución donde se almacena el estado de los sensores y actuadores en cada momento (archivo de LOGs).

- Replay (Debug). Leer el *LOG* para cuando se esté debuggeando e ir mostrando en la pantalla el estado de la máquina de estados, encendiendo con colores las cosas que se van activando para saber qué es lo que pasó...
- WebCam. De alguna forma, cuando el *RAL* es un robot real, se debería poder seleccionar que una WebCam grabe lo que sucede. Así sería un "debugging" para un robot real. Esto respetaría la filosofía de que no es posible debuggear como estamos acostumbrados, las cosas en un robot no funcionan así. Entonces, lo grabo y lo reproduzco en camara lenta...

Está organizado en tres paquetes:

- model: aca esta toda la parte "funcional". Por ejemplo, la clase Program tiene el programa con sus cajas y conexiones y la clase Robot tiene la descripción de cada robot.
- gui: todo lo que tiene que ver con la interacción con el usuario (paneles, cajas, dibujos, intereacción con el mouse, etc).
- model.persist: carga y graba de archivos xml.
- utils: métodos que facilitan algunas tareas.
- thirdparty: librerías que bajé programadas por otras personas.

La conexión entre el modelo y la gui se da por el método *publish-suscribe*: hay definidas interfaces de *listener*, y las clases pueden suscribirse a diferentes acciones. Por ejemplo, la clase *JConnectionsPanel*, que dibuja las conexiones, se suscribe al programa para que le avise cuando se genera una nueva conexión. También, por ejemplo, el panel con el esquema del robot se suscribe a la clase Robot para que le avise cuando algun sensor entra en "foco" y lo pinta.

La parte de gui es la más enquilombada, pero no pude hacerlo más fácil.

Hay un archivo xml que define cada robot, y uno con configuración general (*config.xml*). Desde ahí se puede cambiar las cajas que aparecen en las herramientas, los dibujos, etc. Los dibujos están todos en la carpeta images.

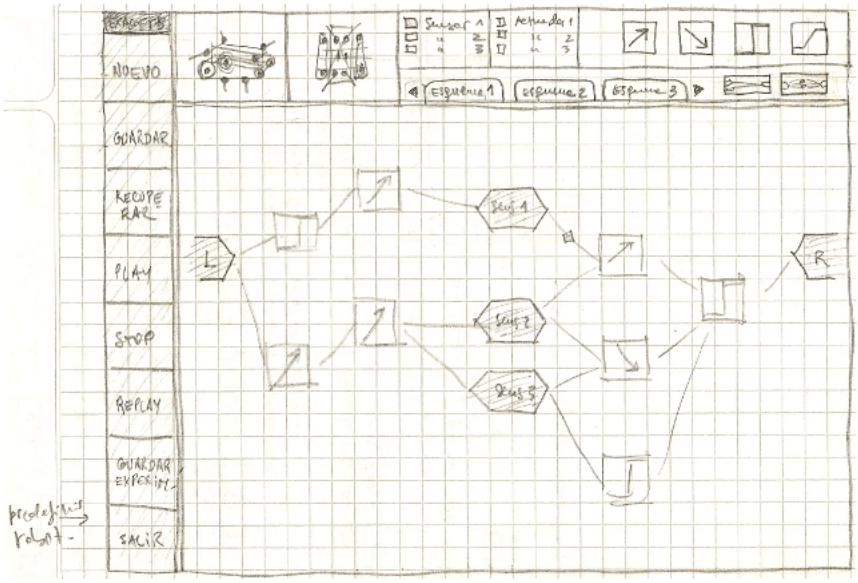
- PONER LO DE LOS XMLs PARA CONFIGURAR !!!! (configGral y Robots!!)
- SON ESPECTACULARES!!! AGREGANDO Y TOCANDO AHÍ, SE PARAMETRIZA TODO!!! LAS CONFIGs
- GRALES DE LA GUI Y TOCANDO EL DE LOS ROBOTS SE CONFIGURAN LOS ROBOTS, ANDO UNA MASA!!!
- DOCUMENTAR CADA PARAMETRO, MOSTRAR TAMBIÉN LOS PNGs (SENSORES Y ROBOTS)...

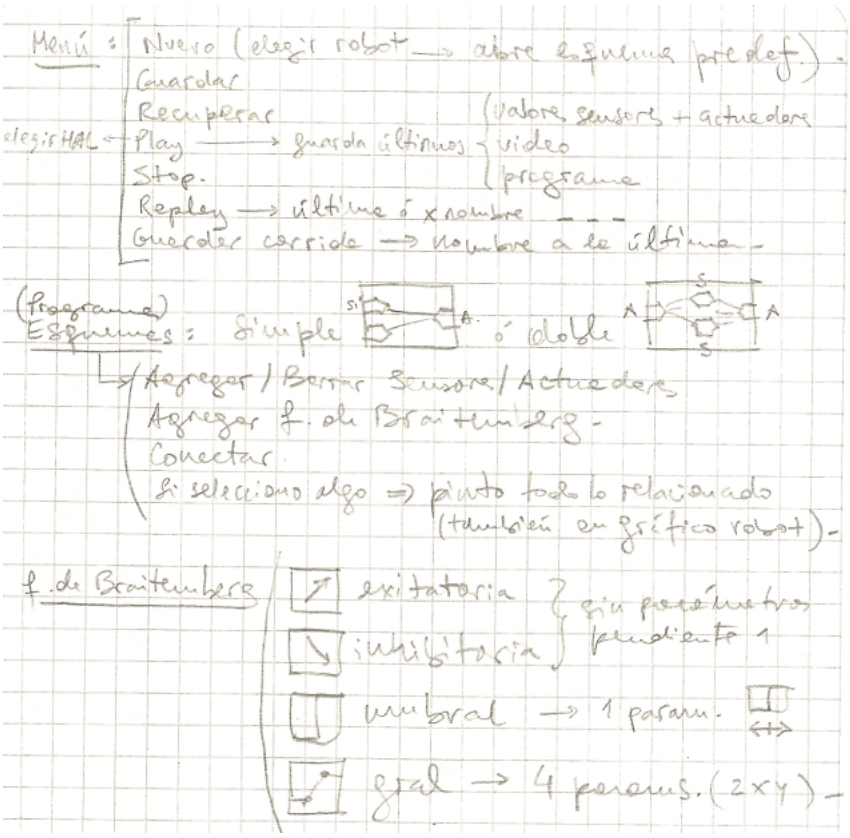
**IMPLEMENTACION**

El módulo GUI está implementado en `Java`. Elegimos este lenguaje por la capacidad de portabilidad y la no necesidad de recompilar para distintos Sistemas Operativos. El único requerimiento en la PC para ejecutar la GUI es tener instalado el JVM (Java Virtul Machine).

Para desarrollar la interfaz gráfica, usamos la *Swing API* (JFC/Swing). Ver http://java.sun.com/docs/books/tutorial/uiswing/index.html y http://en.wikipedia.org/wiki/Swing_(Java)

La ventana de la *GUI* sería algo así:




FALTA DESCRIBIR BIEN QUE HACE CADA BOTÓN Y CUAL ES LA IDEA !!!!

## 3.4. XML

El archivo *XML* servirá, por un lado, para la definición de datos que la *GUI* establecerá para que el *Core* ejecute. Por otro lado, el *XML* contendrá también información propia de la *GUI*.

El archivo *XML* contendrá varias cosas:

- Los datos necesarios para que el *Core* pueda realizar la ejecución.
- Los datos necesarios que la GUI requerirá para poder funcionar, como las especificaciones gráficas, objetos, ubicación de los mismos, etc; y todas las opciones sobre los proyectos realizados...
- ¿algo más?

De esta forma, en principio el *XML* podría tener en secciones separadas los datos para el *Core* y para la *GUI*. Tal vez, no necesariamente estén completamente separados. De modo que, por ejemplo, el *Core* deberá buscar en el *XML* sólo los datos necesarios para lograr la ejecución e ignorar el resto de los datos innecesarios...

### 3.4.1. Datos para la Ejecución del Core

Básicamente, el *Core* busca en la "estructura de arbol" del *XML* el elemento raíz de nombre:

```
<ejecucion>   ... </ejecucion>
```

Cualquier otro elemento distinto de $< ejecucion >$ será ignorado.

**Importante:** El elemento $< ejecucion >$ debe ser el primero en orden de definición dentro del *XML* ya que el *Core* parsea al *XML* utilizando la API que implementa el estandar DOM[3]. Cualquier otro elemento posterior es ignorado.

Entonces, la definición de los datos para la ejecución del *Core* en el *XML* constaran de 3 grandes cosas:

```
<ejecucion>
    <sensores>   ... </sensores>
    <cajas>      ... </cajas>
    <actuadores> ... </actuadores>
</ejecucion>
```

#### Sensores

Por un lado estaría la definición de los sensores existentes y su identificación (*id*). Por ahora, el *id* indicará todo lo referido al sensor, es decir, su tipo (sonar, telémetro, encoder, random) y su ubicación relativa al robot en ángulos (de 0º a 360º), por ejemplo:

```
<sensor id='sonar.0'/>
<sensor id='telemetro.20'/>
<sensor id='telemetro.340'/>
<sensor id='encoder.motor.izquierda'/>
<sensor id='encoder.motor.derecha'/>
<sensor id='sonar.180'/>
<sensor id='random'/>
```

Vimos de agregar un tipo de sensor *random*, que no sería un sensor real en el hardware, sino un sensor simulado en software para poder agregar "aleatoriedad"...

#### Cajas

Después, definir las *cajas* que irían entre los sensores y actuadores. En principio, estas *cajas* sólo tendrían definidas las *entradas* (sensores y otras cajas) y la función. Por ahora sólo tenemos en cuenta

[3]Document Object Model (DOM), http://xerces.apache.org/xerces-c/api-3.html, http://www.w3.org/DOM/

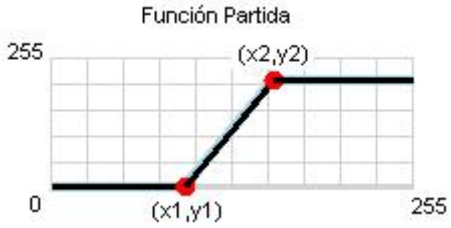
la “función partida” en tres tramos (constante + lineal + constante) que la definimos con 2 puntos en el plano $(x_1; y_1)$ y $(x_2, y_2)$. También definir el *id* de la funcion.

Internamente, la salida de la *caja* será el resultado de aplicar la *función*, definida por los puntos $(x_1; y_1)$ y $(x_2, y_2)$, a la sumatoria de todas sus entradas.

Por ejemplo:

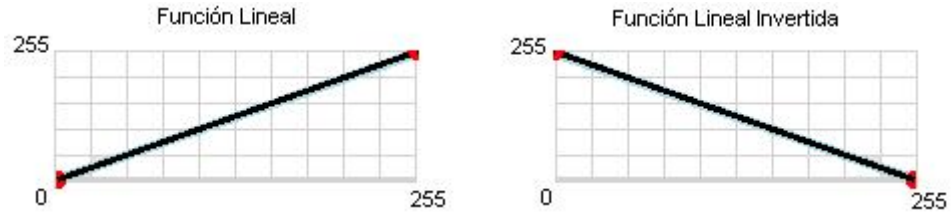
```
<caja id='caja1'>
  <entradas>
    <entrada id='sonar.0'/>
    <entrada id='telemetro.340'/>
    <entrada id='random'/>
  </entradas>
  <puntos>
    <punto x='100' y='0'/>
    <punto x='150' y='255'/>
  </puntos>
</caja>

<caja id='caja2'>
  <entradas>
    <entrada id='telemetro.20'/>
    <entrada id='caja1'/>
  </entradas>
  <puntos>
    <punto x='150' y='255'/>
    <punto x='100' y='0'/>
  </puntos>
</caja>
```

Algo así:




Las funciones *lineal* y *lineal invertida* podrían representarse con $\{(0; 0), (255; 255)\}$ y $\{(255; 255), (0; 0)\}$.

Algo así:




### Actuadores

Ahora sólo nos queda definir los actuadores, su *id* y cúales *cajas* son sus entradas. El sentido del *id* es igual al que se intenta dar en los sensores, por ejemplo:

Internamente, la salida del *actuador* será la sumatoria de todas sus entradas.

```
<actuador id='motor.izquierda'>
  <entradas>
    <entrada id='caja1'/>
    <entrada id='caja2'/>
  </entradas>
</actuador>

<actuador id='motor.derecha'>
  <entradas>
    <entrada id='caja2'/>
  </entradas>
</actuador>
```

**Importante**

Las etiquetas y atributos definidos anteriormente deben ser estrictamente definidos de esa forma en el *XML* (en minúsculas). **Cualquier otra etiqueta o atributo distintos de:**

```
<ejecucion></ejecucion>
    <sensores></sensores>
        <sensor id='?'/>
    <cajas></cajas>
        <caja></caja>
            <entradas></entradas>
                <entrada id='?'/>
            <puntos>
                <punto x='?' y='?'/>
    <actuadores></actuadores>
        <actuador></actuador>
            <entradas></entradas>
                <entrada id='?'/>
```

**serán ignorados.**

### 3.4.2. Datos para la GUI

## 3.5. LOG

El archivo de *log* de cada ejecución se encarga de esribirlo el *Core*. Siempre sobreescribe el archivo especificado, o lo crea si no existe, es decir, sólo queda en el archivo el contenido de la última ejecución.

El *log* tiene la siguiente especificación:

1. **La primera línea.** Consiste de la secuencia, separada por comas, de los *ids* de la tabla de ejecución en el orden en que se encuentran en la misma.
2. **Siguientes líneas.** Son todas iguales. Consiste de varios valores, separados por comas, de la siguiente forma:
   - *TimeStamp*. Es el tiempo en *milisegundos* para cada línea relativo al comienzo, es decir, comienza en cero.
   - *Valor de los elementos*. En el mismo orden en que fueron detallados en la primera línea, si es una `caja` son los valores *entrada* y *salida* de la caja, y si es un *sensor* o un *actuador* es simplemente el valor de *salida*.

Por ejemplo, el siguiente archivo de *log* corresponde a 10 ejecuciones del *Core*:

```
sonar.0, sonar.1, sonar.2, actuador.0, caja.0, caja.1, actuador.1,
0, 6, 8, 7, 8, 22, 13, 8, -9, 20,
103, 3, 7, 6, 7, 17, 13, 7, -8, 19,
```

```
204, 8, 3, 2, 3, 14, 13, 3, -6, 15,
304, 7, 4, 10, 4, 15, 13, 4, -6, 23,
405, 7, 3, 7, 3, 13, 13, 3, -6, 20,
505, 10, 8, 9, 8, 26, 13, 8, -9, 22,
606, 4, 3, 1, 3, 10, 13, 3, -6, 14,
707, 3, 10, 6, 10, 23, 13, 10, -13, 19,
807, 8, 10, 9, 10, 28, 13, 10, -13, 22,
1009, 3, 2, 7, 2, 7, 8, 2, -6, 15,
```

**PENDIENTE**

REPLAY Y DEBUG...

# Capítulo 4

# Robots, Hardwares y Simuladores

## 4.1. YAKS - Yet Another Khepera Simulator

YAKS es un simulador de cóìdigo abierto, escrito en `C++`, de robots tipo `Khepera`, desarrollado por Johan Carlsson. Su nombre proviene del acrónimo *Yet Another Khepera Simulator*. Posee las siguientes características:

- Permite incluir en el entorno obstáculos circulares, paredes, luces y definir zonas.
- Permite definir y manipular un número ilimitado de robots.
- Permite separar el programa de control del simulador, ya que los robots pueden ser manejados a través de una conexión `TCP/IP`.
- Soporta una gran variedad de sensores: proximidad, luminosidad, energía, encoders de las ruedas, compás y sensor de tierra (para detección de zonas).

### 4.1.1. Requerimientos de Compilación ???

COMPLETAR ESTO CON http://profesores.elo.utfsm.cl/~tarredondo/memorias/2005-memoria-fquiros.pdf QUE EXPLICA MUUY BIEN TODO EL YAKS...

### 4.1.2. Ejecución en Linux 32 bits

Una vez compilado el *YAKS*, podemos proceder a su ejecución. Antes se deberán realizar las siguientes tareas:

- Copiar el ejecutable *gsim* del *YAKS* (que en la compilación fue creado en *src/bin*) en la carpeta raíz de los fuentes *src*.
- Crear el archivo `yaks-params.opt` de parámetros para la ejecución del *YAKS* en la carpeta raíz del *YAKS*. Puede verse un ejemplo de este archivo en http://www.exa.unicen.edu.ar/catedras/irobotic/yaks-params.htm.
  Es necesario tener en cuenta que la mayoría de estos ejemplos se encuentran hechos para Windows, por lo tanto es necesario editar todas las rutas de archivos y carpetas de ".\" a "./" para que funcione correctamente en Linux. Por ejemplo, en el archivo de parámetros de ejemplo, es necesario cambiar "`WORLD_PATH .\worlds`" por "`WORLD_PATH ./worlds`".
- Agregar en el archivo `yaks-params.opt` de parámetros la línea `CAMERA_PATH ./cam` para que el *YAKS* sepa dónde buscar estos archivos.

- Ejecución: ejecutar el simulador *YAKS* como `./gsim yaks-params.opt` en la carpeta raíz del *YAKS*.

**Errores en la Ejecución**

Es muy posible que al intentar ejecutar, la librería *GTK* arroje errores de ejecución como `Gdk-ERROR`.

En Linux, esto se debe a que la librería *GTK* necesita que los efectos visuales de pantalla del sistema operativo estén deshabilitados.

En Windows, HAY PROBLEMAS ??? FALTA COMPLETAR ESTO!!!

**Solución:** Para deshabilitar los efectos visuales, se ejecuta en la consola el comando:
`export XLIB_SKIP_ARGB_VISUALS=1`

## 4.2. YAKS para Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

## 4.3. RAL YAKS Linux

## 4.4. RAL YAKS Windows

## 4.5. Khepera

*Khepera* es un robot móvil desarrollado por la empresa K-Team. Tiene un cuerpo circular, de 5,5$cm$ de diámetro, y consta de dos ruedas (actuadores) y ocho pares de sensores infrarojos, que pueden funcionar como sensores de proximidad o de luz direccionales. Estos robots pueden ser controlados desde una PC a través de una interface serie o de una interface de radio.

Para más información ver:

- http://ftp.k-team.com/khepera/documentation/
- http://ftp.k-team.com/khepera/documentation/KheperaUserManual.pdf
- http://ftp.k-team.com/khepera/documentation/RadioBaseManual.pdf
- http://ftp.k-team.com/khepera/documentation/RadioTurretManual.pdf

## 4.6. RAL Khepera

En función de las especificaciones del robot *Khepera* y las características particulares necesarias para la conexión con el mismo, ya sea a través de *Cable Serial* o *Radio Frecuencia*, en los distintos sistemas operativos, fue necesario crear cuatro *RALs* de *Khepera* destintos para cada una de las posibilidades:

- RAL Khepera Linux - Cable
- RAL Khepera Windows - Cable
- RAL Khepera Linux - Radio

- RAL Khepera Windows - Radio

A continuación se detallan cada una de ellas:

### 4.6.1. Conexión por Cable

Primero es importante chequear que el robot esté correctamente configurado para este tipo de conexión. En la sección *3.1.3 - Jumpers, reset button and settings* del manual de usuario se detalla los modos de conexión.

**Importante:** La configuración correcta es `MODE 1`, que implica una conexión por *cable serial a 9600 Baud*.

#### RAL Khepera Linux 32 bits

En Linux, para controlar el puerto serial, o más conocido como `COM1`, se realiza a través del archivo de sistema `/dev/ttyS0`, donde en general:

- `/dev/ttyS0` ó `/dev/cua0` corresponde con el puerto `COM1` en Windows
- `/dev/ttyS1` ó `/dev/cua1` corresponde con el puerto `COM2` en Windows
- `/dev/ttyS2` ó `/dev/cua2` corresponde con el puerto `COM3` en Windows
- `/dev/ttyS3` ó `/dev/cua3` corresponde con el puerto `COM4` en Windows

Por lo tanto en `C++`, generar una conexión a través del puerto serial, leer, escribir y cerrar el mismo se realiza con las funciones comunes para manejo de *streams* y archivos:

```
#include <iostream>
open( "/dev/ttyS0" );
close( file_descriptor );
write( file_descriptor );
read( file_descriptor );
```

Es importante el modo en el que se abre el archivo del `COM1` y más importante, configurar el puerto para la conexión necesaria para el robot Khepera. Esto se hace de la sigueinte manera:

```
// se abre el COM1 en modo "non-blocking"
com1_file_descriptor = open( "/dev/ttyS0", O_RDWR | O_NOCTTY | O_NONBLOCK );
        // O_RDWR - open read-write.
        // O_NOCTTY - open TTY without it becoming controlling tty.
        // O_NONBLOCK ó O_NDELAY - open in non-blocking mode (read will return immediately)
```

Luego se configura el puerto antes de comenzar a utilizarlo:

```
// se definen constantes para simplificar
#define BAUDRATE    B9600   // BAUDRATE = 9600
#define DATABITS_8  CS8     // DATABITS = 8 bits
#define STOPBITS_2  CSTOPB  // STOPBITS_2 = 2
#define PARITYON    0       // es igual a PARITY_NONE ó PARITY_DISABLED
#define PARITY      0       // es igual a PARITY_NONE ó PARITY_DISABLED
// se crea la estructura para setear la configuración del puerto
struct termios com1_new_set;
com1_new_set.c_cflag = ( BAUDRATE | CRTSCTS | DATABITS_8 | STOPBITS_2 | PARITYON | PARITY | CLOCAL | CREAD );
com1_new_set.c_iflag = IGNPAR;
com1_new_set.c_oflag = 0;
com1_new_set.c_lflag = 0;
com1_new_set.c_cc[VMIN] = 1;
com1_new_set.c_cc[VTIME] = 0;
```

```
tcflush( com1_file_descriptor, TCIFLUSH );
// se setea la nueva configuración para el puerto COM1
tcsetattr( com1_file_descriptor, TCSANOW, &com1_new_set );
```

La compilación de este *RAL* es básicamente la misma a la de todos los demás, con la diferencia que el proceso de *linkeo* para generar la librería dínamica debió ser levemente cambiado. El error ocurría al intentar *linkear* el ejecutable del *Core* dinámicamente con *libRAL.so*, producía el siguiente error:

```
hidden symbol '__dso_handle' in /usr/lib/gcc/i486-linux-gnu/4.3.3/crtbegin.o is referenced by DSO
/usr/bin/ld: final link failed: Nonrepresentable section on output
collect2: ld returned 1 exit status
```

Por lo tanto, en el *Makefile* incluído en los archivos fuentes de este *RAL*, para generar la librería dinámica *libRAL.so* el proceso de *linkeo* que se realizaba mediante *ld -o libRAL.so RAL.o -shared* fue cambiado por ***g++ -shared -Wl -o libRAL.so RAL.o***.

Por último, la frecuencia de trabajo que devuelve esta librería (*getFrecuenciaTrabajo()*) es igual al valor que devuelve el *RAL YAKS* ($100mseg$). Según las pruebas que se realizaron parece andar bien, pero si aparecieran inconvenientes será necesario revisar este valor. Tener en cuenta que la frecuencia de trabajo debería quedar determinada por:

- Envío de comando y tiempo de transmisión de esa cantidad de caracteres a 9600 baudios.
- Tiempo de sensado (de todos los sensores ¿16?) del *Khepera*.
- Tiempo de transmisión de las cantidad de caracteres de la respuesta a 9600 baudios.
- Sumatoria de todo lo anterior...

Para probar el robot manualmente (chequear que se tiene conexión con el mismo), se pueden instalar y usar los siguientes programas para el manejo del puerto (como el *hyperterminal* de Windows):

```
sudo apt-get install gtkterm
sudo apt-get install setserial
```

#### RAL Khepera Windows

### 4.6.2. Conexión por Radio

#### RAL Khepera Linux

#### RAL Khepera Windows

# Capítulo 5

# Compilación

## 5.1. Core

El código fuente del Core cuenta con los siguientes archivos:

1. **core.cpp** Código principal para la ejecución del *Core*.
2. **Estructuras.h** Encabezados de las clases utilizadas para la ejecución del *Core*.
3. **Estructuras.cpp** Código de las clases utilizadas para la ejecución del *Core*.
4. **Parser.h** Encabezados de las funciones para el parseo del archivo *XML*.
5. **Parser.cpp** Código de las funciones para el parseo del archivo *XML*.

Además, el código fuente del Core necesita para su compilación los siguientes archivos del *RAL*:

1. **RAL.h** Encabezados de las funciones del *RAL*.
2. **libRAL.so** Librería dinámica del *RAL*.

Puede verse la especificación para la compilación de estos archivos del *RAL* en el punto 5.3.1 en la página 31.

### 5.1.1. Linux 32 bits

#### Makefile

El código fuente del Core incluye un archivo *Makefile* con las siguientes funciones para facilitar la compilación, linkeo estático con el *Xerces Parser*, linkeo dinámico con el *RAL* y el testeo del Core:

1. **all:** Ejecuta las funciones *compilar_core* y *enlazar_ejecutable*.
2. **compilar_core:** Compila los archivos del código fuente del *Core* generando los objetos (**.o*) necesarios para la creación del ejecutable del Core de la siguiente manera:
   *g++ -c Estructuras.cpp -o Estructuras.o*
   *g++ -c Parser.cpp -o Parser.o*
   *g++ -c core.cpp -o core.o*

3. **enlazar_ejecutable:** Genera el ejecutable del *Core* (`test_core`) enlazando con las *librerías estáticas* del *Xerces* y con las *librerías dinámicas* del *RAL* de la siguiente manera:
*g++ -o test_core core.o Estructuras.o Parser.o -lxerces-c -lpthread -L<path> -Bdynamic -lRAL*
donde *<path>* debe ser la ruta (puede ser relativa) donde se encuentra la librería dinámica *libRAL.so*, por ejemplo `-L../../ral/src`

4. **clean:** Borra todos los archivos **.o* y el ejecutable *test_core*.

5. **run:** Ejecuta *test_core* con los siguientes parámetros:
*./test_core xml_file_test_07.xml archivoLOG_01.log yaks*

**Script**

Para facilitar algunas cuestiones en la compilacion, también se incluye un archivo script "core_compilar.sh". El mismo puede ejecutar simplemente con `./core_compilar.sh`.

### 5.1.2. Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

## 5.2. Xerces XML Parser

### 5.2.1. Linux 32 bits - Librerías Estáticas

El código fuente del Core se compila incluyendo las librerías estáticas del Xerces para que estén incluidas en el ejecutable del Core y no sea necesario transportalas. Para eso, es necesario obtener el código fuente de las librerías del Xerces y recompilarlas de forma estática antes de poder compilar el Core.

Para recompilar las librerías del Xerces de forma estática, los pasos son los siguientes:

1. Bajar el archivo `xerces-c-3.0.1.zip` del código fuente del Xerces de http://apache.xmundo.com.ar/xerces/c/3/sources/xerces-c-3.0.1.zip

2. Descomprimir el archivo `xerces-c-3.0.1.zip` en alguna carpeta, por ejemplo, en */home/.../workspace/xerc c-3.0.1-static/*

3. En la carpeta */home/.../workspace/xerces-c-3.0.1-static/xerces-c-3.0.1/* ejecutar "*./configure –disable-shared –disable-network*" para que no compile las librerías dinámicas (*.so*), y sólo compile las estáticas (*.a*). La opción *–disable-network* podría obviarse.

4. En la carpeta */home/.../workspace/xerces-c-3.0.1-static/xerces-c-3.0.1/* ejecutar "*make*" para compilar.

5. En la carpeta */home/.../workspace/xerces-c-3.0.1-static/xerces-c-3.0.1/* ejecutar "*sudo make install*" para instalar las librerías estáticas en el sistema. Por defecto, las mismas se instalan en */usr/local/bin*, */usr/local/lib*, */usr/local/include*.

Por último, para compilar cualquier código fuente que incluya las librerías, es necesario indicarle al *linker* que incluya las librerías estáticas */usr/local/lib/libxerces-c.a* y *libpthread.a*. Si esto último se hace desde alguna IDE de programación en `C++` simplemente se agrega en las opciones del programa, *Sección Linker*, la inclusión de las librerías mencionadas. Si la compilación se realiza manualmente, los parámetros son los siguientes: `g++ -static -o nombreEjecutable Codigo.cpp -lxerces-c -lpthread`

### 5.2.2. Windows 32 bits - Librerías Estáticas

Para recompilar las librerías del Xerces de forma estática en *Windows*, los pasos son los siguientes:

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

## 5.3. RAL

### 5.3.1. Linux 32 bits - Librería Dinámica

El código fuente del *RAL* cuenta con los siguientes archivos:

1. **RAL.h** Encabezados de las funciones para la utilización de la librería dinámica del *RAL*.
2. **RAL.cpp** Código de las funciones de la librería dinámica del *RAL*.

#### Makefile

Además, el código fuente incluye un archivo *Makefile* con las siguientes funciones para facilitar la compilación de la librería dinámica del *RAL*:

1. **all:** Compila y enlaza los archivos del código fuente generando la librería dinámica *libRAL.so* de la siguiente manera:
   *g++ -c RAL.cpp -o RAL.o*
   *ld -o libRAL.so RAL.o -shared* ó *g++ -shared -Wl -o libRAL.so RAL.o* (dependiendo el caso)
2. **clean:** Borra todos los archivos **.o* y **.so* del *RAL*.

Para más información sobre la creación, compilación y enlace de librerías dinámicas en *Linux*, puede consultarse http://www.chuidiang.com/clinux/herramientas/librerias.php

#### Error de Compilación en 64 bits

Si el Linux es de 64 bits, es probable que falle la compilación de *RAL.cpp*. La solución es agregar en la línea de compilación el parámetro "`-fPIC`", de forma que la línea antes indicada quede como "*g++ -c RAL.cpp -o RAL.o -fPIC*".

### 5.3.2. Windows 32 bits - Librería Dinámica

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!! FALTA COMPLETAR ESTO!!! Ojo con esto, porque aunque simula el GNU-GCC, no necesariamente todas las librería incluidas andan, porque algunas son especificas de Linux, por ejemplo "sys/socket.h", que me parece que en Windows hay que cambiarla por "winsock.h'0'. HAY QUE VER BIEN ESTO y ANOTAR !!!!

FALTA COMPLETAR ESTO!!! Nota: cualquier cosa, probar también Cygwin 5.1.6 (GNU + Cygnus + Windows) que contiene de http://www.cygwin.com/. ojo con esto porque me parece que sí o sí necesita "cygwin1.dll" en la PC para que después pueda andar... Probar!!!

## 5.4. YAKS

El código fuente del YAKS se encuentra en http://www.exa.unicen.edu.ar/catedras/irobotic/YAKS.htm En particular, bajamos dos archivos:

- http://www.exa.unicen.edu.ar/catedras/irobotic/SOFTWARE/YAKS-src-update-2.zip: Código Fuente Completo para Linux.
- http://www.exa.unicen.edu.ar/catedras/irobotic/SOFTWARE/yaks-linux-patch.tar.gz: Patch de YAKS para que no tire errores al compilarlo en Unix.

Luego se descomprime el contenido de los dos archivos fuentes en alguna carpeta, por ejemplo `yaks/src`.

### 5.4.1. Linux 32 bits

- En la carpeta donde se encuentra el código fuente del *YAKS*, creamos dos carpetas *bin* y *lib*, que son necesarias para crear los archivos resultantes de la compilación.
- Instalar la librería *GTK*, que es una librería estandard de `C++` para el manejo de ventanas gráficas. Se puede instalar de la siguiente forma: `sudo apt-get install libgtk1.2-dev`
- Compilar: en la carpeta donde se encuentra el código fuente del *YAKS*, ejecutar `make`

#### Errores en la Compilación

Es muy posible que al intentar compilar, el compilador arroje errores relacionados con `iostream.h`, `cout` y `endl` en los archivos fuentes `ann.cpp`, `ann.h`, `ga.cpp` y `ga.h`.

En Linux, este problema se origina al enlazar con la librería *GTK*, ya que no es posible utilizar `std::cout` porque no "entiende" a qué pantalla o consola tiene que mandar el *stream*, puesto que la librería *GTK* maneja varias pantallas.

**Solución:** Es necesario editar los archivos fuentes `ann.h`, `ann.cpp` y `ga.h`, sacando de los mismos las líneas `#include<iostream.h>` y `std::cout` (con sus respectivos operadores <<). Si se deseara mantener la impresión por pantalla, se deberían cambiar los `std::cout` por la función `printf` de `C`.

#### Error de Librería GTK

Al parecer, la librería *libgtk1.2-dev* está desactualizada (muy vieja) y no está más disponible en los repositorios de Ubuntu. La opción que queda es instalar la *libgtk2.0-dev*. Pero con esta nueva versión, el YAKS da errores de compilación.

**Solución 1:** Agregar en la lista de fuentes del *apt* el lugar desde donde bajar la *libgtk1.2-dev*, así:

```
sudo gedit /etc/apt/sources.list
```

agregar al final la línea "*deb http://cz.archive.ubuntu.com/ubuntu hardy main universe*" y ejecutar:

```
sudo apt-get update
sudo apt-get install libgtk1.2-dev
```

ahora debería instalarse automáticamente.

Si no anda, es necesario bajar y compilar a mano la librería desde: http://packages.ubuntu.com/hardy/i386/libgtk1.2-dev/download.

**Solución 2:** Corregir los fuentes del YAKS para que compile con la *libgtk2.0-dev*. Parece complicado... ESTO ESTÁ PENDIENTE !!!!

Y HAY QUE HCERLO PORQUE EN LOS LABOS NO DAN MAS SOPORTE PARA *libgtk1.2-dev*, SÓLO VAN A TENER LA *libgtk2.0-dev* !!!! HABLAR COM Maximiliano Geier !!!

#### Script

Para facilitar la compilacion, se incluye un archivo script "yaks_compilar.sh". El mismo puede ejecutar simplemente con `./yaks_compilar.sh`.

### 5.4.2. Windows 32 bits

FALTA COMPLETAR ESTO!!! Nota: cualquier cosa, probar también Cygwin 5.1.6 (GNU + Cygnus + Windows) que contiene de http://www.cygwin.com/. ojo con esto porque me parece que sí o sí necesita "cygwin1.dll" en la PC para que después pueda andar... Probar!!!

## 5.5. GUI

### 5.5.1. Linux y Windows 32 bits

El código fuente de la GUI cuenta con los siguientes archivos:

- ./gui:
  - .classpath
  - .project
  - gui_ejecutar.sh
  - gui_instalar.sh
- ./gui/extension:
  - config.xml
  - ExtensionApp.java
- ./gui/extension/gui:
  - BoxColumnLayout.java
  - ComponentDragger.java
  - JBox.java
  - JBoxPanel.java
  - JBoxTemplate.java
  - JConnectionsPanel.java
  - JInlineDialog.java
  - JParametrosCajaEnergia.java
  - JParametrosCaja.java
  - JProgramPanel.java
  - JRoboticaFrame.java
  - JRobotPanel.java
  - PopupMenuMouseAdapter.java

- ./gui/extension/images:
  - activar_no.png
  - activar_si.png
  - act_rueda.png
  - conexion_cola.png
  - conexion_punta.png
  - f_energia.png
  - f_exitatoria.png
  - f_inhibitoria.png
  - f_parametrica.png
  - menu_abrir.png
  - menu_ejecutar.png
  - menu_guardar.png
  - menu_nuevo.png
  - menu_pausa.png
  - menu_salir.png
  - seleccionar.png
  - sen_contacto.png
  - sen_linea.png
  - sen_luz.png
  - sen_proximidad.png
  - sen_sonar.png
  - sen_telemetro.png
- ./gui/extension/model:
  - ActuatorBox.java
  - ActuatorType.java
  - Box.java
  - BoxListener.java
  - ConnectionMaker.java
  - ConnectionMakerListener.java
  - Diagram.java
  - FunctionBox.java
  - FunctionTemplate.java
  - GlobalConfig.java
  - ImageMapFeature.java
  - ImageMap.java
  - Panel.java
  - Program.java
  - ProgramListener.java
  - Robot.java
  - RobotListener.java
  - SensorBox.java
  - SensorType.java

- ./gui/extension/model/persist:
  - GlobalConfigXml.java
  - ProgramXml.java
  - RobotXml.java
- ./gui/extension/robots:
  - exabot.xml
  - khepera.xml
  - robot_exabot.png
  - robot_khepera.png
  - robot_yaks.png
  - yaks.xml
- ./gui/extension/utils:
  - FileUtils.java
  - IconBank.java
  - XmlUtils.java
- ./gui/thirdparty/dragnghost:
  - AbstractGhostDropManager.java
  - DragnGhostDemo.java
  - DragnGhostDemo.jnlp
  - GhostComponentAdapter.java
  - GhostDropAdapter.java
  - GhostDropEvent.java
  - GhostDropListener.java
  - GhostDropManagerDemo.java
  - GhostGlassPane.java
  - GhostMotionAdapter.java
  - GhostPictureAdapter.java
  - GlassPaneExtension.java
  - HeaderPanel.java
  - UIHelper.java

Para la compilación y debugging de este código, se cuenta con un *Eclipse Proyect* cuyas definiciones se ecuentran en los archivos *gui/.project* y *gui/.classpath*.

# Capítulo 6

# Instalación

Estructuralmente el software se divide en dos carpetas:

1. ***ERBPI/src:*** contiene todos los fuentes necesarios del software para su compilación.

2. ***ERBPI/bin:*** contiene todos los ejecutables, producto de la compilasión de los fuentes, necesarios para la ejecución completa del software.

Cada módulo y RALs contienen su fuentes y, por lo tanto, sus scripts necesarios para su compilacion e instalación. La instalación de cada módulo produce la estructura de binarios siguientes:

- ERBPI/bin:
  - /core
  - /gui
  - /yaks

Opcionalmente también se introduce en la carpeta *ERBPI/bin/comportamientos* que contiene archivos *XML* con comprtamientos estándard ya programados.

## 6.1. Core

### 6.1.1. Linux 32 bits

Para la instalación se ejecuta el script *ERBPI/src/core/core_instalar.sh*. Es necesario que el Core y las librerías dinámicas de cada RAL ya se encuentren compiladas con anterioridad. Para compilación de Core y RAL ver puntos 5.1 y 5.3.

### 6.1.2. Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

## 6.2. GUI

### 6.2.1. Linux 32 bits

Para la instalación se ejecuta el script *ERBPI/src/gui/gui_instalar.sh*. Es necesario que la GUI se encuentre compilada con anterioridad. Para compilación de GUI ver punto 5.5.

### 6.2.2. Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

## 6.3. YAKS

### 6.3.1. Linux 32 bits

Para la instalación se ejecuta el script *ERBPI/src/yaks/yaks_instalar.sh*. Es necesario que el YAKS se encuentre compilado con anterioridad. Para compilación de YAKS ver punto 5.4.

### 6.3.2. Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

# Capítulo 7

# Ejecución

## 7.1. Core

### 7.1.1. Linux 32 bits

No es necesario ejecutar el Core manualmente, de esto se encarga la GUI.

#### Script

De todas formas, si se quisera probar manualmente su ejecución, puede utilizarse el script *ERBPI/src/core/core_ejecutar.sh* editándolo y modificándolo con los parámetros requeridos.

### 7.1.2. Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

## 7.2. GUI

### 7.2.1. Linux 32 bits

La GUI se ejecuta de la siguiente forma: *java extension.ExtensionApp.*

#### Script

Para facilitar esta ejecución, se incluye un archivo script *ERBPI/bin/gui/gui_ejecutar.sh.*

### 7.2.2. Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

FALTA COMPLETAR ESTO!!!

## 7.3. YAKS

### 7.3.1. Linux 32 bits

El YAKS debe ejecutarse manualmente cada vez que se quiera utilizar com RAL desde la GUI. El YAKS se ejecuta de la siguiente forma: *gsim yaks-params.opt*. Debido a que es necesario setear parámetros, tanto para el YAKS en *yaks-params.opt* como en el sistema operativo para la librería GTK, se proporciona un script que tiene en cuenta estos detalles.

#### Script

Para facilitar esta ejecución, se incluye un archivo script *ERBPI/bin/yaks/yaks_ejecutar.sh*.

### 7.3.2. Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

# Capítulo 8

# Compiladores

## 8.1. Core

### 8.1.1. C++ Linux 32 bits

Utilizamos los siguientes compiladores para C++:

- gcc: GCC (GNU Compiler Collection) C compiler.
- g++: GCC (GNU Compiler Collection) C++ compiler.

Ver http://gcc.gnu.org/

### 8.1.2. C++ Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!! Para que los comandos como *gcc*, *g++*, *make* (*mingw32-make*) anden en la consola de windows, es necesario modificar la variable de sistema `PATH` de Windows y agregar la ruta `C:/MinGW/bin`.

FALTA COMPLETAR ESTO!!! Ojo con esto, porque aunque simula el GNU-GCC, no necesariamente todas las librería incluidas andan, porque algunas son especificas de Linux, por ejemplo "sys/socket.h", que me parece que en Windows hay que cambiarla por "winsock.h'0'. HAY QUE VER BIEN ESTO y ANOTAR !!!!

FALTA COMPLETAR ESTO!!! Nota: cualquier cosa, probar también Cygwin 5.1.6 (GNU + Cygnus + Windows) que contiene de http://www.cygwin.com/. ojo con esto porque me parece que sí o sí necesita "cygwin1.dll" en la PC para que después pueda andar... Probar!!!

## 8.2. Xerces XML Parser

### 8.2.1. C++ Linux 32 bits

Utilizamos los siguientes compiladores para C++:

- gcc: GCC (GNU Compiler Collection) C compiler.

- g++: GCC (GNU Compiler Collection) C++ compiler.

Ver http://gcc.gnu.org/

### 8.2.2. C++ Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!! Para que los comandos como *gcc*, *g++*, *make* (*mingw32-make*) anden en la consola de windows, es necesario modificar la variable de sistema `PATH` de Windows y agregar la ruta `C:/MinGW/bin`.

FALTA COMPLETAR ESTO!!! Ojo con esto, porque aunque simula el GNU-GCC, no necesariamente todas las librería incluidas andan, porque algunas son especificas de Linux, por ejemplo "sys/socket.h", que me parece que en Windows hay que cambiarla por "winsock.h'0'. HAY QUE VER BIEN ESTO y ANOTAR !!!!

FALTA COMPLETAR ESTO!!! Nota: cualquier cosa, probar también Cygwin 5.1.6 (GNU + Cygnus + Windows) que contiene de http://www.cygwin.com/. ojo con esto porque me parece que sí o sí necesita "cygwin1.dll" en la PC para que después pueda andar... Probar!!!

## 8.3. RAL

### 8.3.1. C++ Linux 32 bits

Utilizamos los siguientes compiladores para C++:

- gcc: GCC (GNU Compiler Collection) C compiler.
- g++: GCC (GNU Compiler Collection) C++ compiler.

Ver http://gcc.gnu.org/

### 8.3.2. C++ Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!! Para que los comandos como *gcc*, *g++*, *make* (*mingw32-make*) anden en la consola de windows, es necesario modificar la variable de sistema `PATH` de Windows y agregar la ruta `C:/MinGW/bin`.

FALTA COMPLETAR ESTO!!! Ojo con esto, porque aunque simula el GNU-GCC, no necesariamente todas las librería incluidas andan, porque algunas son especificas de Linux, por ejemplo "sys/socket.h", que me parece que en Windows hay que cambiarla por "winsock.h'0'. HAY QUE VER BIEN ESTO y ANOTAR !!!!

FALTA COMPLETAR ESTO!!! Nota: cualquier cosa, probar también Cygwin 5.1.6 (GNU + Cygnus + Windows) que contiene de http://www.cygwin.com/. ojo con esto porque me parece que sí o sí necesita "cygwin1.dll" en la PC para que después pueda andar... Probar!!!

## 8.4. YAKS

### 8.4.1. C++ Linux 32 bits

Utilizamos los siguientes compiladores para C++:

- gcc: GCC (GNU Compiler Collection) C compiler.
- g++: GCC (GNU Compiler Collection) C++ compiler.

Ver http://gcc.gnu.org/

### 8.4.2. C++ Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!! Para que los comandos como *gcc*, *g++*, *make* (*mingw32-make*) anden en la consola de windows, es necesario modificar la variable de sistema `PATH` de Windows y agregar la ruta `C:/MinGW/bin`.

FALTA COMPLETAR ESTO!!! Ojo con esto, porque aunque simula el GNU-GCC, no necesariamente todas las librería incluidas andan, porque algunas son especificas de Linux, por ejemplo "sys/socket.h", que me parece que en Windows hay que cambiarla por "winsock.h'0'. HAY QUE VER BIEN ESTO y ANOTAR !!!!

FALTA COMPLETAR ESTO!!! Nota: cualquier cosa, probar también Cygwin 5.1.6 (GNU + Cygnus + Windows) que contiene de http://www.cygwin.com/. ojo con esto porque me parece que sí o sí necesita "cygwin1.dll" en la PC para que después pueda andar... Probar!!!

## 8.5. GUI

### 8.5.1. Java Linux 32 bits

Utilizamos el siguiente paquete: OpenJDK 6 (openjdk-6-jdk).

Con el paquete Open Source Java Development Kit obtenemos compilador e intérprete para Java Standard Edition.

### 8.5.2. Java Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!