Proyecto de Grado

IDE Android para la Programación de Comportamientos Robóticos

Informe Final

Andrés Nebel - Renzo Rozza

Tutores
Ing. Andrés Aguirre, Ing. Rafael Sisto

24 de Abril del 2014

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay

Resumen

La tecnología en el área de los dispositivos móviles ha experimentado grandes avances, dando como resultado dispositivos cada día más potentes, que ofrecen inclusive tantas prestaciones como los ordenadores de escritorio. Asimismo, el gran aporte del Proyecto Butiá a la robótica educativa, ha otorgado a niños de todas las edades la posibilidad de controlar al robot Butiá usando entornos de desarrollo (se destaca TortuBots entre ellos) desde las computadoras XO. Sin embargo, estas herramientas no son soportadas por dispositivos móviles y no promueven la programación basada en el paradigma reactivo.

En el marco de este proyecto de grado se investigaron distintos tipos de arquitecturas robóticas pertenecientes al paradigma reactivo, lenguajes de programación visual y alternativas para consolidar la implementación de un entorno de desarrollo (IDE por sus siglas en inglés) basado en comportamientos que permita programar al robot Butiá y pueda ser ejecutado desde dispositivos con el sistema operativo Android.

Yatay fue el nombre elegido para el sistema resultante de este proyecto, el cual está estructurado como una aplicación web en HTML5. Utiliza la biblioteca Blockly como lenguaje de programación visual y recibe influencias de sistemas como TortuBots y Scratch. El IDE implementa una arquitectura robótica basada en prioridades perteneciente al paradigma reactivo, para administrar los comportamientos que controlan al robot Butiá. Se decidió usar un kit extendido de éste robot, agregando una placa Raspberry Pi donde se aloja un servidor web implementado en torno a los sistemas Lumen (entorno cooperativo multitarea) y Toribio (plataforma para crear aplicaciones robóticas). A lo largo de este documento se describe y detalla el proceso realizado para cumplir las metas planteadas en este proyecto de grado.

Palabras Claves: Robótica educativa, Robot Butiá, IDE, Android, HTML5, Raspberry Pi.

Índice

Resumen 2
Capítulo 1 10
Introducción 10
1.1. Motivación 10
1.2. Objetivos 11
1.3. Organización del documento 11
1.4. Etapas del proyecto 13
Capítulo 2 15
Resumen del estado del arte 15
2.1. Paradigmas y arquitecturas robóticas 15
2.1.1. Paradigma Reactivo 15
2.1.2. Arquitecturas robóticas 16
2.2. Plataforma robótica 18
2.2.1. Hardware y Software 19
2.2.2. Complementos (Raspberry Pi) 20
2.3 Constructivismo y enseñanza de la robótica 21
2.4. Comunicación con USB4Butiá 23
24.1. USB Host 23
2.4.2. WiFi 25
2.5. Concurrencia y orientación a comportamientos 27
2.5.1. Semáforos y Threads en Android 27
2.5.2. Toribio y Lumen 28
2.6. HTML5 31
2.7. Lenguajes de programación visual 31
2.7.1. Blockly 33
2.8. Conclusiones del Estado del Arte 34
2.8.1. Portar un IDE anterior o crear uno nuevo 35

2.8.2. Subsumption vs Arquitectura basada en prioridades 35
2.8.3. HTML5 vs Android App 36
2.8.4. Lumen y Toribio 36
2.8.5. Blockly 37
2.8.6. Lua vs Python 37
Capítulo 3 40
Requerimientos 40
3.1. Alcance 40
3.2. Requerimientos funcionales 40
3.2.1. Entorno y Ejecución 40
3.2.2. Robot 41
3.2.3. Proyectos 41
3.2.4. Comportamientos 41
3.3. Requerimientos no funcionales 42
Capítulo 4 45
Prototipos y pruebas realizadas 45
4.1. Pruebas de USB Host 45
4.2. Prueba del servidor web y creación dinámica de tareas. 45
4.3. Prueba de Waterbear como LPV 46
4.4. Prueba de Blockly como LPV 47
Capítulo 5 49
Arquitectura del sistema 49
5.1. Estilo arquitectónico 49
5.2. Subsistemas 50
5.2.1. Descripción 51
5.3. Distribución 53
5.3.1. Escenario: Nodos 53
5.4. Diagrama de clases 54
5.5. Diagramas de comunicación 56
5.5.1. Crear comportamiento 56
5.5.2. Task switching 57

5.5.3. Refrescar dispositivos 57
Capítulo 6 60
Detalles de implementación 60
6.1. Algoritmo principal 60
6.2. Gestión de proyectos 61
6.2.1. Base de datos 61
6.2.2. Guardar proyectos 62
6.2.3. Abrir proyectos 62
6.3. Diferenciación de dispositivos 63
6.3.1. Bibliotecas YepNope y Modernizr 63
6.4. Calibración y creación de nuevos sensores 63
Capítulo 7 66
Configuración 66
7.1. Servidor 66
7.2. Kit Robótico 68
7.3. Navegador Web 69
Capítulo 8 71
Verificación del sistema 71
8.1. Casos de prueba 71
8.2. Errores destacables corregidos 76
8.3. Limitaciones tecnológicas 78
8.3.1. Websockets 78
8.3.2. Features HTML5 79
8.3.3. SVG 80
8.3.4. Features CSS3 81
8.3.5. Bibliotecas de edición y highlight de código 82
8.4. Errores conocidos 82
8.4.1. Exportación de comportamientos en Android Browser 83
8.4.2. Edición de código generado en Android Browser 83
8.4.3. Soporte parcial para Internet Explorer Mobile 83
Capítulo 9 86

Extensibilidad del Sistema 86
9.1. Creación de bloques nuevos 86
9.2. Kit robótico 87
9.2.1. Robot Interface 87
9.3. Arquitectura robótica 88
Capítulo 10 92
Conclusiones y trabajo a futuro 92
10.1. Conclusiones 92
10.2.1 Características de la aplicación 93
10.2. Trabajo a futuro 95
10.2.1. Versiones finales de Blockly y mejoras de otras bibliotecas 95
10.2.2. Administración y persistencia de sensores creados 95
10.2.3. Kit Robótico 96
10.2.4. Otros paradigmas robóticos o arquitecturas 96
10.2.5. Módulo para administración de proyectos 96
10.2.6. Edición multiusuario de comportamientos 96
10.2.7. Extender el soporte multilenguaje 97
Apéndice A 103
Glosario 103

Índice de Figuras

1. Etapas del proyecto, dedicación en meses por actividad.
2. Diagrama comparativo entre arquitecturas deliberativas y reactivas.
3. Diagrama representativo para una arquitectura basada en prioridades.
4. Taller del Proyecto Butiá en Colonia del Sacramento, Junio 2013.
5. Robot Butiá 2.0.
6. Componentes de la placa Raspberry Pi.
7. Robot Butiá/Yatay.
8. Robot tortuga construido por Seymour Papert.
9. Ilustración de la Dynabook descrita por Alan Kay en su paper.
10. Pendrive conectado a una Tablet mediante un cable USB OTG.
11. Diagrama de componentes haciendo uso de la placa Raspberry Pi.
12. Ejecución de tareas en nuevo thread con Android SDK.
13. Ejemplo de creación y utilización de semáforos en Android SDK.
14. Ejemplo de una tarea activando a otra mediante un signal.
15. Ejemplo de una tarea que mueve los motores según el valor de un sensor.
16. Ejemplo de código en Tortubots.
17. Interfaz de Blockly.
18. Diagrama de estilo arquitectónico.
19. Diagrama de subsistema estáticos.
20. Diagrama de subsistemas estáticos y dinámicos.
21. Diagrama de distribución.
22. Diagrama de clases server-side.
23. Diagrama de clase de un comportamiento genérico.
24. Diagrama de comunicación para ejecutar comportamiento.
25. Diagrama de comunicación para el task switching.
26. Diagrama de comunicación para refrescar los bloques de dispositivos conectados.
27. Código utilizado para identificar dispositivos.
28. Árbol de directorios del framework de Yatay.
29. Pantalla inicial de Yatay en Tablet de 10’’.
30. Echo Test con websocket.org para Firefox en un ordenador.
31. Tabla representativa del soporte del atributo download por browsers.
32. Tabla representativa del soporte de SVG por browsers.
33. Tabla representativa del soporte de función calc() por browsers.
34. Código JavaScript para agregar un bloque en Blockly.
35. Código JavaScript para agregar el código generado por un bloque en Blockly.

Parte I

Generalidades

Capítulo 1

Introducción

Este documento enmarca un proyecto de grado en la carrera de Ingeniería en Computación de la Universidad de la República (UdelaR). Los integrantes de este equipo decidieron trabajar en esta propuesta motivados por la robótica educativa con la intención de acompañar los cambios tecnológicos a nivel mundial. Construyendo un entorno de desarrollo basado en el paradigma reactivo que permite controlar al robot Butiá desde computadores y dispositivos móviles (Tablets o celulares), apuntado a estudiantes y docentes de educación primaria y secundaria.

1.1. Motivación

La robótica educativa es una disciplina en la que se concibe, diseña, desarrolla y operan robots como forma de introducir a los estudiantes desde jóvenes en el estudio de las ciencias y la tecnología [30]. La inclusión de ésta en las aulas ha demostrado ser una herramienta pedagógica poderosa. En general los alumnos se muestran más motivados frente a una propuesta educativa enriquecida y presentan mayor compromiso con sus propios procesos de aprendizaje. Además, estas logran que los alumnos apliquen conocimientos de ciencias (matemáticas, mecánica, programación, entre otras) y mejoren tanto su cooperación como su capacidad de formular hipótesis, investigar soluciones factibles y obtener conclusiones. [8][40]

Desde el año 2009 el grupo MINA del INCO (Instituto de Computación) trabaja en el desarrollo de una plataforma robótica llamada Butiá, que ha impulsado un progresivo avance de la robótica educativa a nivel nacional. Disponibilizando, hasta la fecha, más de 100 robots ubicados en escuelas y liceos de todo el país.

En el contexto de la robótica móvil autónoma, existe un conjunto de arquitecturas de control pertenecientes al paradigma reactivo. Este paradigma es bioinspirado y puede ser enfocado para promover el desarrollo del educando, ya que intenta reflejar la conducta animal entendida por el estudiante. Se utilizan en el paradigma módulos que implementan comportamientos sencillos (sensar y actuar) o la combinación de ellos para lograr comportamientos más complejos. Esto permite que los procesos de enseñanza y de aprendizaje se lleven a cabo en base a conceptos simplificados ya que el paradigma reactivo elimina la planificación y no mantiene un modelo de mundo.

Por otro lado, los avances tecnológicos en los últimos años han fomentado una evolución de los dispositivos móviles. Las Tablets y teléfonos inteligentes han aumentado sus capacidades de

procesamiento y dejaron de poseer sistemas embebidos para incorporar sistemas de propósito general. A modo de ejemplo el Plan Ceibal, en este año (2014) entregará por primera vez Tablets a los estudiantes de primer año escolar [31].

Existen varios IDE que permiten programar al robot Butiá. Entre ellos se destaca TortuBots, uno de los más utilizados en los talleres de robótica ofrecidos en el marco del proyecto Butiá. Durante el 2011 y 2012 un grupo de la asignatura proyecto de grado desarrolló un entorno, llamado eButiá, basado en el lenguaje de programación visual Etoys. Dicho entorno de desarrollo, se diferencia en utilizar una arquitectura reactiva Subsumption, que permite mediante comportamientos controlar al robot Butiá desde las computadoras XO [1].

A pesar de que existen varios entornos para programar al robot Butiá, ninguno de éstos permite su ejecución en dispositivos móviles, lo cual restringe su utilización a computadores. Asimismo, salvo puntuales excepciones (como eButiá), no promueven la programación basada en el paradigma reactivo.

Por lo tanto, se considera interesante acompasar las metodologías de enseñanza a los avances tecnológicos, motivando a la investigación de distintas alternativas para proporcionar una herramienta que permita programar al robot Butiá desde dispositivos móviles. Incorporando los beneficios otorgados por las arquitecturas basadas en el paradigma reactivo (véase 2.1.1 Paradigma Reactivo) y los lenguajes de programación visual, conformando un entorno de desarrollo que promueva el crecimiento de la robótica educativa.

1.2. Objetivos

Este proyecto de grado propone la investigación e implementación de un entorno de desarrollo (IDE) basado en el paradigma reactivo que permita programar al robot Butiá, a través de un lenguaje de programación visual (basado en bloques), que pueda ser utilizado tanto desde computadoras como desde dispositivos móviles. Dicha investigación tiene como objetivo encontrar la mejor solución para lograr esto, analizando la adaptación de IDEs existentes, así como también la implementación entera de un nuevo IDE que logre objetivos similares a los anteriormente creados para computadoras.

1.3. Organización del documento

Este documento está organizado en capítulos, donde se pretende introducir al lector progresivamente en los conceptos manejados a lo largo del proyecto. A grandes rasgos se identifican

tres partes: en primer lugar, se resume el análisis realizado en el estudio del estado del arte. Luego se tratan las características fundamentales, la organización y desarrollo del sistema construido, para terminar dando detalles de las evaluaciones y experiencias obtenidas a partir de este proyecto.

La parte de *Generalidades* contiene los capítulos:

- Capítulo 1 - **Introducción**: Se presenta la gestación del proyecto; sus motivaciones, objetivos y alcance.

- Capítulo 2 - **Resumen del estado del arte**: Análisis y estudio de las áreas que enmarcan al proyecto. Se introducen los conceptos básicos, detallando técnicas y tecnologías candidatas para el desarrollo del sistema. Por último, se mencionan las decisiones tomadas al finalizar la etapa de investigación, previa al desarrollo.

La parte de *Desarrollo del Sistema* contiene los capítulos:

- Capítulo 3 - **Requerimientos**: Se enumeran los requerimientos funcionales y no funcionales identificados para la construcción del sistema.

- Capítulo 4 - **Prototipos y pruebas realizadas**: Contempla los prototipos y pruebas realizadas para la valoración de tecnologías y mitigación de riesgos.

- Capítulo 5 - **Arquitectura del sistema**: Expone las distintas características de la arquitectura y diseño del sistema, describiendo desde varios enfoques la distribución del software y sus principales componentes.

- Capítulo 6 - **Detalles de implementación**: Se exhiben ciertas características destacables del sistema desde el punto de vista de la implementación de las mismas, dando a conocer algoritmos, técnicas y tecnologías usadas.

- Capítulo 7 - **Configuración**: Se describen las distintas configuraciones necesarias para la utilización de la aplicación, dando detalles para cada nodo del sistema.

- Capítulo 8 - **Validación del sistema**: Desglose de las pruebas realizadas para verificar y validar el funcionamiento del sistema. Además, se mencionan los errores conocidos y las distintas limitaciones tecnológicas.
- Capítulo 9 - **Extensibilidad del sistema**: Se enumeran las distintas posibilidades existentes para extender el software desarrollado, sus dificultades y el impacto que podrían tener en el mismo.

La parte de *Resultados* contiene los capítulos:

- Capítulo 10 - **Conclusiones y trabajo a futuro**: Se establecen las conclusiones finales del proyecto y se proponen diversas actividades que podrían complementar el trabajo realizado.

1.4. Etapas del proyecto

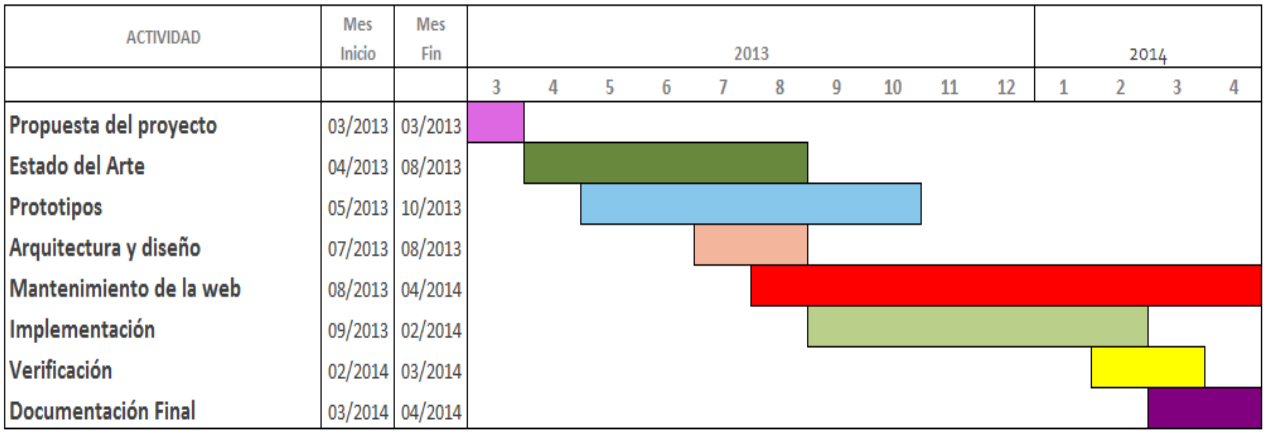


Figura 1: Etapas del proyecto, dedicación en meses por actividad.

Capítulo 2

Resumen del estado del arte

En este capítulo se brinda al lector una introducción sobre los conceptos fundamentales investigados en este proyecto. Se puede obtener más información sobre los temas tratados en el documento Estado del arte [25].

2.1. Paradigmas y arquitecturas robóticas

Llamamos arquitectura de un robot autónomo a la manera general de organizar un sistema de control. Ésta describe un conjunto de componentes y la forma en que interactúan [2]. Permitiendo determinar las conductas que exhibe un robot autónomo y definiendo, cómo se activan y cómo se resuelve el problema cuando múltiples comportamientos se activan al mismo tiempo.

Cuanto más complejo es un sistema más relevancia cobra el papel de la organización de sus componentes, siendo la arquitectura quien decide cómo organizar estos procesos internos en robots. Un robot, tiene la tarea de construir o seleccionar señales de entrada a partir de información no específica que sus sensores ofrecen. La naturaleza de los objetivos de estos puede variar enormemente, con prioridades dinámicas que dependen de las necesidades actuales y de la situación del entorno. Por estas razones, no existe una arquitectura válida para todos los entornos y para todos los comportamientos. Tradicionalmente este campo se ha dividido en tres grandes corrientes paradigmáticas: los sistemas deliberativos (o paradigma jerárquico), los reactivos y los híbridos. Estas corrientes se diferencian principalmente en cómo toma la decisión un robot de actuar a partir de lo sensado, más específicamente, en la interrelación entre las 3 principales acciones de la robótica: sensar, planificar y actuar.

Este proyecto se centrará en el tipo de paradigma reactivo, el cual veremos que se caracteriza por eliminar el concepto de planificación y establecer una relación directa entre sensar y actuar. A continuación se hará un análisis de este paradigma, y se explicarán sus ventajas y desventajas, así como el motivo de su elección para este trabajo.

2.1.1. Paradigma Reactivo

Este proyecto se centrará en el tipo de paradigma reactivo, el cual se caracteriza por eliminar

el concepto de planificación y establecer una relación directa entre sensar y actuar, sin mantener un modelo del mundo y no pasando por las etapas de modelar y planificar como hacen los robots deliberativos, logrando de esta manera, una reacción más rápida a los eventos. La figura 2, muestra lo anterior en una comparación entre paradigmas deliberativo y reactivo.

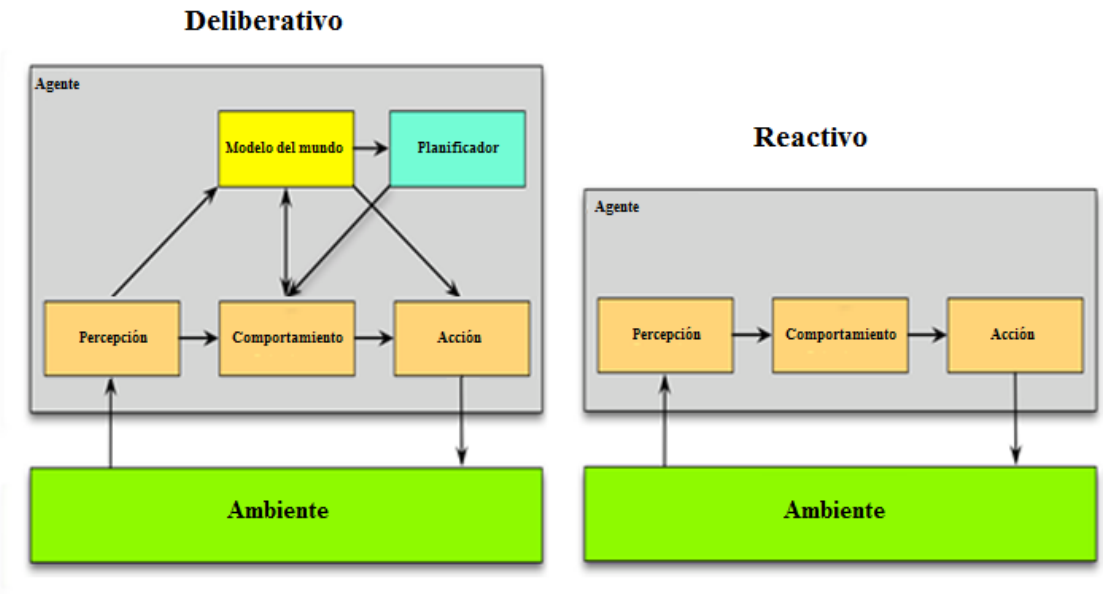


Figura 2: Diagrama comparativo entre arquitecturas deliberativas y reactivas [10].

Dentro de las ventajas encontradas, por un lado este paradigma permite no depender de un hardware potente para hacer cálculos simbólicos complicados y por el otro la sencillez de una arquitectura reactiva brinda mayor facilidad a la hora de enseñar que una deliberativa por eliminar la planificación y el concepto de modelo de mundo. De esta forma, permite apuntar las experiencias educativas a un rango mayor de edades.

2.1.2. Arquitecturas robóticas

Dentro del paradigma reactivo existen distintas arquitecturas robóticas conocidas que implementan el concepto principal de sensar-actuar del paradigma. El equipo de trabajo consideró el análisis de arquitecturas robóticas realizado por el proyecto de grado “IDE de programación orientado al desarrollo de arquitecturas robóticas basadas en comportamientos” [1] debido a que comparte objetivos con este proyecto (se apunta al mismo rango de usuario). En dicho análisis se argumenta que la arquitectura reactiva de Campos de potencial tiene como desventaja mapear los problemas con vectores de fuerza, lo cual resulta poco intuitiva y es un concepto que aún no manejan muchos de los estudiantes a los que apunta este proyecto. También se menciona que, la arquitectura Teoría de los Esquemas no define la coordinación de comportamientos la que queda a cargo del programador y en este caso éste posee poca o nula experiencia. A partir de estos argumentos se

descartan éstas dos arquitecturas robóticas ya que no se adecuan a los objetivos de este proyecto.

A continuación se entrará en detalle sobre la arquitectura Subsumption, la cual fue utilizada por el proyecto de grado anteriormente mencionado.

2.1.2.1. Subsumption

Una de las arquitecturas es llamada Subsumption, que agrupa comportamientos en capas, por lo cual establece una jerarquía entre las tareas del robot. Las tareas de capas mayores corresponden a comportamientos de más alto nivel de abstracción, mientras que las de más abajo contienen a las tareas más simples. Cada uno de los comportamientos en sí está implementado como una máquina de estados. Los de capas superiores pueden alterar la entrada o salida de las tareas de capas inferiores, lo cual les da cierto control de manipulación sobre estas. Subsumption describe dos formas de realizar lo anterior: inhibir, el cual apaga la salida de un módulo de menor nivel si hay una salida de una tarea superior, o suprimir que sustituye la entrada de un módulo de menor nivel con la salida de un módulo de mayor nivel. Cada capa puede poseer varios comportamientos, por lo tanto el resultado de los actuadores del Robot en un momento dado es el conjunto de todas las ejecuciones concurrentes de la capa actual.

2.1.2.1.1. Simplificación de Subsumption

Existen diversas implementaciones de la arquitectura Subsumption. Algunas de estas, simplifican algunos de sus principales conceptos teóricos. En este proyecto se consideró una de estas implementaciones [20], que tiene como variante que cada comportamiento se implementa en forma de máquina de estados pero con la diferencia de que cada uno de los comportamientos tiene a su vez un valor numérico de prioridad único asociado a él. Se define por tanto una jerarquía de ejecución de comportamientos, ya que existen tareas con mayor prioridad de ejecución de otras. La implementación establece que siempre habrá un y sólo un comportamiento activo ejecutándose que hará uso de los actuadores. Cualquiera de los comportamientos puede activarse en cualquier momento, pero al activarse, se compara su prioridad con la prioridad de la tarea actualmente activa como puede observarse en la figura 3. Si su prioridad es mayor, entonces desplazaran a la otra tarea para convertirse en la nueva tarea activa, por lo cual pasará a ejecutarse. De lo contrario, continuará la ejecución la tarea que estaba activa, ya que posee mayor prioridad. A esta variante de Subsumption se le llamará a lo largo de este documento “Arquitectura basada en prioridades”.

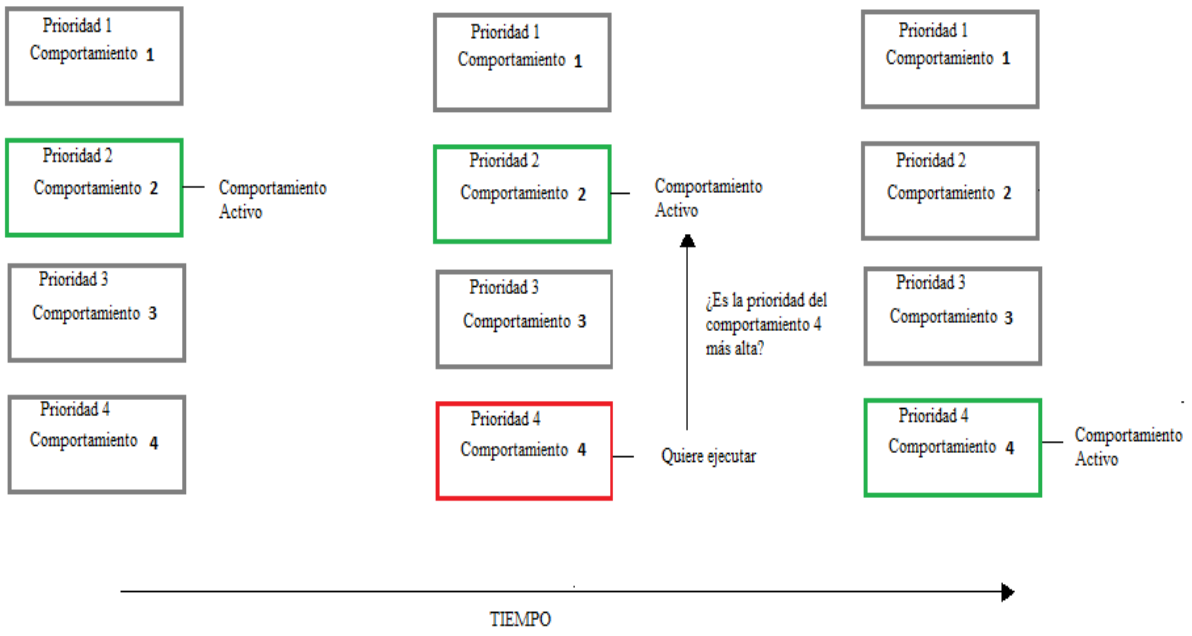


Figura 3: Diagrama representativo para una arquitectura basada en prioridades.

2.2. Plataforma robótica

El proyecto Butiá fue lanzado en el año 2009 y con el objetivo de crear una plataforma simple, la cual ponga al alcance de estudiantes escolares o liceales las herramientas necesarias para permitirles interiorizarse con la programación de comportamientos para robots.

Figura 4: Taller del Proyecto Butiá en Colonia del Sacramento, Junio 2013.

2.2.1. Hardware y Software

El hardware del robot Butiá v2.0, que se puede observar en la figura 5, cuenta con una placa USB4Butiá (adaptación para el Butiá de la placa USB4All [39]). Esta placa posee 6 puertos RJ45 (Ethernet) con soporte Plug&Play y Hotplug y un bus para motores AX-12 y de corriente continua. El robot posee sensores de: grises, luz, contacto (botón) y distancia. Por el lado de los actuadores el Butiá posee leds y soporta la inclusión de motores Dynamixel AX-12, aunque en versiones recientes del Butiá se han sustituido a estos por motores de corriente continua los cuales son más económicos.

El único software que forma parte íntegra de Butiá es el firmware de la placa USB4Butiá. Este se encarga de resolver la interacción con los sensores y actuadores, exponiendo una interfaz que permite controlarlos de manera sencilla a través de USB utilizando un protocolo llamado User Protocol. Cada sensor/actuador es modularizado como una entidad y su lógica está encapsulada en un módulo llamado User Module que se encarga de resolver la comunicación a bajo nivel con el sensor/actuador.

Para utilizar el Butiá desde un computador, se creó una aplicación demonio, Bobot Server [1], que recibe funciones en formato de texto plano a través del puerto 2009, o a través de una interfaz web. Este es el encargado de realizar la comunicación a bajo nivel con la placa USB4Butiá y de exponer una interfaz a través de una conexión TCP/IP. A través de ella se pueden listar las funcionalidades ofrecidas por la interfaz e invocar cada una de ellas. De esta forma se independiza a las aplicaciones que utilizan la placa, de la implementación de la misma.

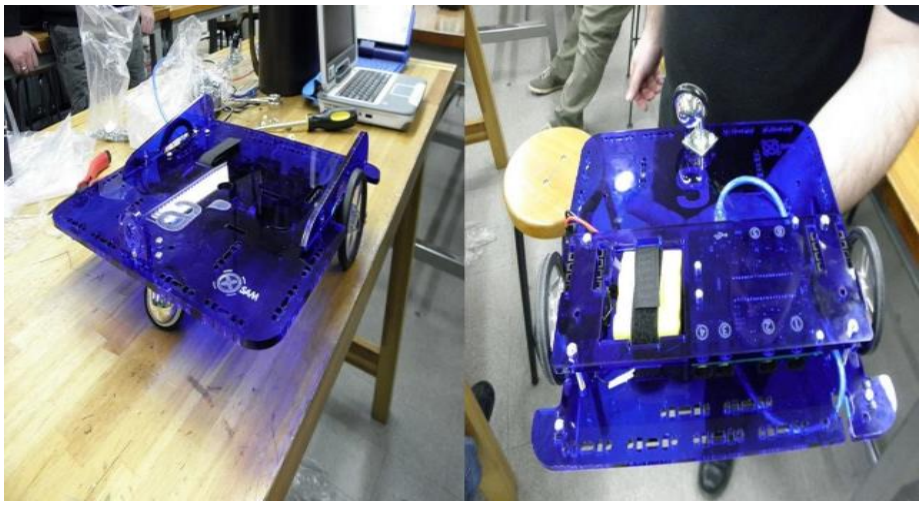


Figura 5: Robot Butiá 2.0.

2.2.2. Complementos (Raspberry Pi)

Para complementar las funcionalidades del robot Butiá se trabaja con distintas componentes de hardware entre las que se encuentra la Raspberry Pi [13], existiendo la posibilidad de agregar éste tipo de placas al kit del robot o en primera instancia hacer uso de esta placa de forma experimental para proyectos como este. Desde el punto de vista del hardware (figura 6), la placa Raspberry Pi cuenta con unas dimensiones de 8.5 por 3.5 cm, donde se ubica un system on a chip Boardcom BCM2835, que contiene un procesador ARM11 con varias frecuencias de funcionamiento y la posibilidad de realizar overclocking hasta 1 GHz sin perder la garantía, un procesador gráfico VideoCore IV capaz de obtener vídeo a 1080p y audio de alta calidad a través de su conector HDMI, y memoria RAM entre 256 y 512 MB. La misma puede ser dotada con una sistema operativo y programas por medio de una tarjeta SD con por lo menos 1 GB y clase 4 (velocidad de lectura/escritura) o mejor. Posee una conexión Ethernet 10/100 y permite conexión WiFi por medio de una interfaz USB WiFi gracias a dos puertos USB incluidos.

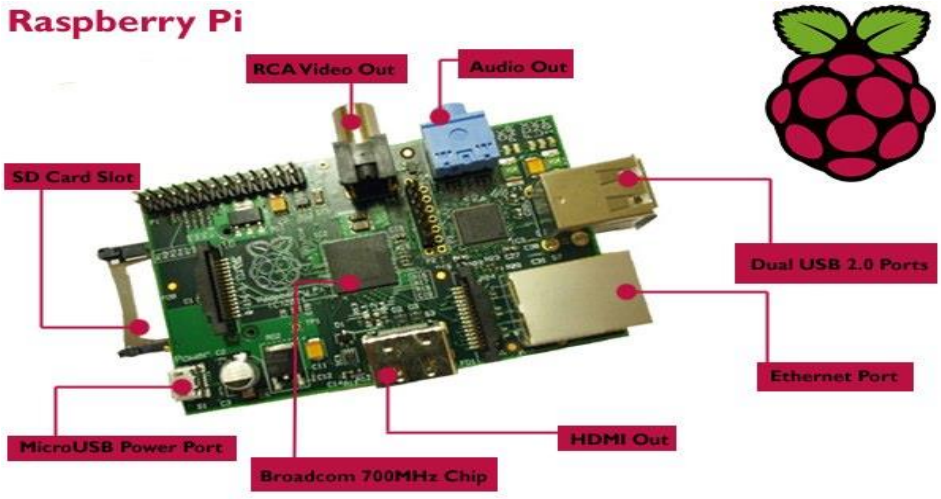


Figura 6: Componentes de la placa Raspberry Pi.

Contando con esta placa en el kit del robot Butiá se tiene la posibilidad de establecer una comunicación inalámbrica con el robot, e incluso la posibilidad de levantar un servidor Web en la placa que provea los servicios para controlar los comportamientos del robot Butiá. Es interesante observar que poseer una SBC dentro de la arquitectura de nuestra solución abre las puertas a la utilización de otras tecnologías, como por ejemplo realizar un enfoque Web y utilizar HTML5 para lograr la compatibilidad con celulares ampliando el soporte de la solución a otros usuarios que no manejen Android como sistema operativo de sus dispositivos, logrando accesibilidad desde iOS, Windows Phone, entre otros. También permite un manejo más eficiente y menos procesamiento del lado de los dispositivos móviles, ya que le quita el procesamiento de la concurrencia entre los comportamientos creados por el usuario.

En la figura 7 se muestra una imagen del prototipo del Robot Butiá Yatay, que incluye una Raspberry Pi como complemento al Hardware del Butiá 2.0 y un nuevo diseño del Robot.

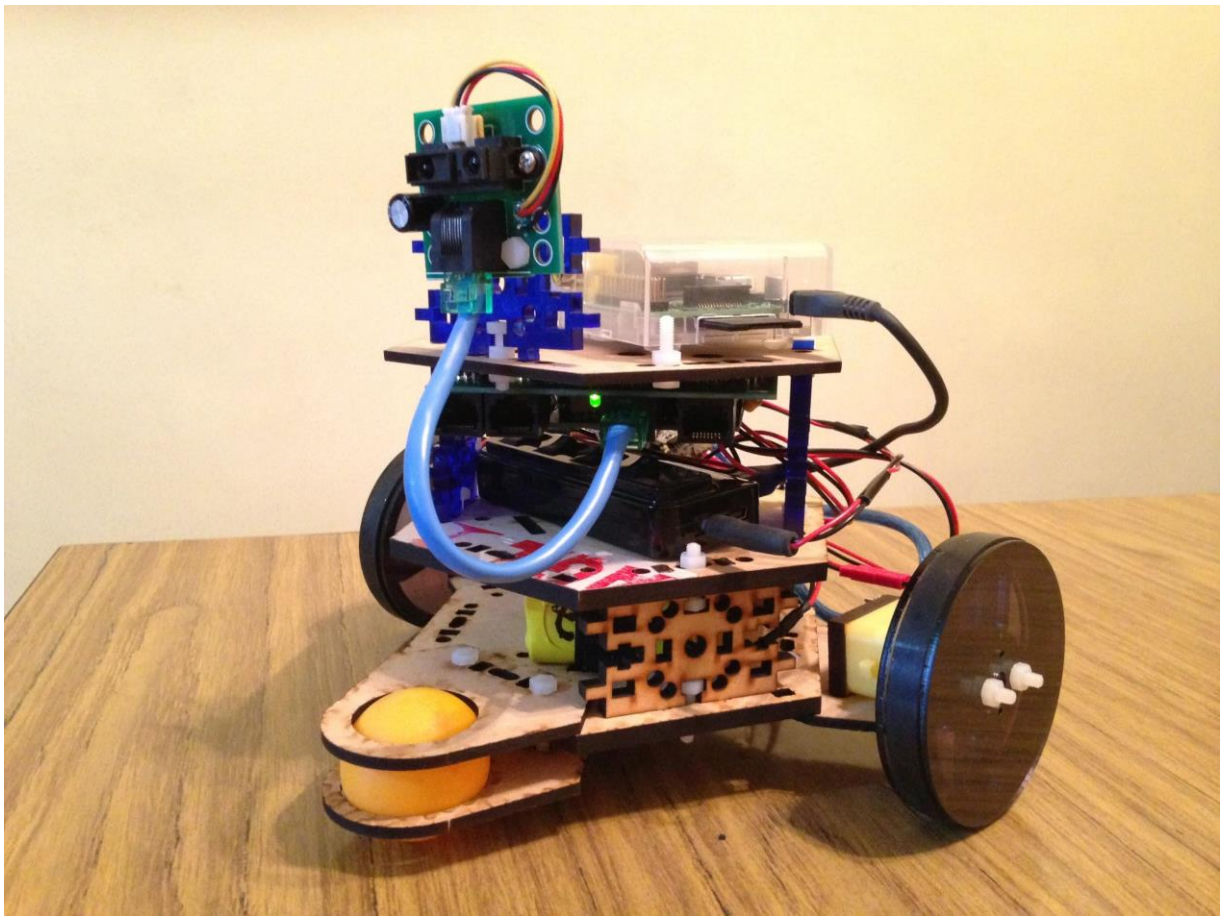


Figura 7: Robot Butiá/Yatay.

2.3 Enseñanza de la robótica

La enseñanza de la robótica surge como una disciplina en la que se concibe, diseña, desarrolla y operan robots como forma de introducir a los estudiantes desde jóvenes en el estudio de las ciencias y la tecnología [30]. La inclusión de la Robótica en la enseñanza intenta enriquecer las experiencias educativas y atraer la atención de los estudiantes que se ven motivados por la oportunidad de manipular un Robot. Existen numerosas investigaciones y experiencias en este ámbito a nivel global (muchas detalladas en el documento Estado del arte [25]), entre ellas se pueden destacar varias: En 1975 en Francia en la Universidad Du Maine, en 1989 en la universidad Autónoma de México, en 1998 el proyecto “Robótica y Aprendizaje por Diseño”, realizado conjuntamente por el Centro de Innovación Educativa de la Fundación Omar Dengo y el Ministerio de Educación Pública de Costa

Rica (Fundación Omar Dengo, 2004) [30], también el Proyecto Butiá en el ámbito local desde el 2009.

El constructivismo es una corriente pedagógica. Los primeros trabajos asociados a esta fueron los de Ernst von Glasersfeld. Tuvo como gran referente a Jean Piaget, y postula que "*el individuo, tanto en los aspectos cognitivos y sociales del comportamiento como en los afectivos, no es un mero producto del ambiente ni un simple resultado de sus disposiciones internas, sino una construcción propia que se va produciendo día con día como resultado de la interacción entre esos dos factores* "[33].

Para esta corriente, el conocimiento de un alumno "*no es una copia fiel de la realidad, sino una construcción del ser humano"* [33], a partir de sus conocimientos previos. Esta construcción es un proceso activo del estudiante, realizado de forma dinámica y participativa. [33][15].

Seymour Papert es un pionero en el área de la inteligencia artificial, científico informático, matemático y educador. Fue uno de los más destacados discípulos de Piaget de quien se influenció y quien llegó a mencionar en una ocasión que nadie comprendía mejor sus ideas que Papert [35]. En 1967 crea Logo (como se menciona en la próxima sección) basado en sus estudios con Piaget. Lo definió no sólo como un lenguaje de programación, sino como una filosofía de educación. Guiado por esa idea, observó varios puntos donde la tecnología con robots ayudaba a los estudiantes, lo cual lo llevó a apoyar sus experiencias educativas con un robot "Tortuga" de su creación (figura 8), siendo pionero también en este ámbito.

Figura 8: Robot tortuga construido por Seymour Papert.

La idea de Logo probó ser una forma efectiva de lograr que los estudiantes comprendan los conceptos de programación, además de que es una herramienta para acercar a las matemáticas a aquellos que no se sienten motivados por ella, dado que con la tortuga se realizan cálculos para definir el avance y los movimientos. También se destaca la sencillez para aquellas personas con

capacidades diferentes. En base a esta filosofía se desarrolló un plan de estudios que contenía dentro del programa grandes áreas como matemáticas, lenguaje y ciencia [1].

2.4. Comunicación con USB4Butiá

Actualmente, la única forma de establecer una comunicación con la USB4Butiá es a través de USB. Teniendo como punto de partida que la comunicación será serial a través de USB, restando definir qué dispositivo establecerá la comunicación con esta componente. Por un lado, se analizará la viabilidad de una comunicación directa entre los dispositivos móviles y la USB4Butiá. Por otro el establecimiento de una conexión a través de WiFi usando la placa Raspberry Pi (mencionada anteriormente) como nodo intermedio entre los dispositivos móviles y la USB4Butiá.

2.4.1. USB Host

El USB On-The-Go conocido también por USB OTG, es una extensión de la norma USB 2.0 que permite a los dispositivos USB tener una mayor flexibilidad en la gestión de la conexión USB. Permite que dispositivos, actúen como host, lo cual nos permite establecer la conexión con la USB4Butiá. Cuando un dispositivo móvil se encuentra en modo USB Host, actúa como el host o master (maestro) de la conexión, alimentando el Bus e identificando a los dispositivos que estén conectados a él. La idea principal de esto, es mediante USB Hosting lograr que las Tablets o celulares establezcan una conexión como maestros con la placa. En la figura 10 se muestra un ejemplo de un pendrive conectado por USB a una Tablet que actúa como Host.

2.4.1.1. Soporte de Android a USB Host

No todas las versiones del sistema operativo Android soportan el modo de USB Host. Esta es una funcionalidad nueva de Android agregada a partir de la versión 3.1 (Honeycomb), permite a las aplicaciones desarrolladas para Android y no incluidas en su Kernel (llamadas usualmente third-party apps) tener acceso para comunicarse con el dispositivo USB que estamos hosteando. Para poder lograr esto, se debe utilizar la API de USB Host incluida en el package **android.hardware.usb** del Android SDK. Esta API nos otorga funcionalidades básicas para lograr la comunicación con el dispositivo USB y está disponible a partir de la API level 12 que coincide con la versiones de Android 3.1 en adelante.

Figura 10: Pendrive conectado a una Tablet mediante un cable USB OTG.

Con los conceptos manejados hasta el momento, se puede deducir que para lograr una conexión con la USB4Butiá y poder lograr la comunicación desde los dispositivos Android con el robot Butiá sin nodos intermedios se necesita obligatoriamente (a menos que se actualicen manualmente los módulos del kernel) una versión de Android igual o posterior a la 3.1. Esto ya determina un problema de compatibilidad no despreciable, reduciendo la cantidad de usuarios finales con posibilidades de usar el IDE en cuestión a sólo aquellos que cumplan dicha condición, lo cual no respeta uno de los objetivos secundarios de este proyecto: lograr compatibilidad con la mayor cantidad de dispositivos posible. Y en contra de lo estipulado en la presentación de este proyecto, que plantea lograr compatibilidad con Android 2.3 en adelante.

2.4.1.2. Estadísticas de compatibilidad de USB Host

Para determinar el margen de compatibilidad que ofrecen los dispositivos con sistema operativo Android respecto a la posible conexión con la placa USB4Butiá mediante USB Host, decidimos realizar una breve estadística que nos otorgue la capacidades la disponibilidad de la API (necesaria para USB Host) en algunas de las Tablets disponibles en el mercado local. Para esto utilizamos USB Host Diagnostics, la cual sube a una página web los resultados del diagnóstico para todo usuario que lo consienta. Se forma así una base de datos que expone para los distintos dispositivos donde se hizo el test, los resultados de este acerca de las capacidades de USB Host. Recorriendo distintas tiendas locales buscamos Tablets a la venta en Uruguay que posean resultados de tests de la herramienta para completar la estadística, incluyendo también algunos resultados de Tablets personales testeadas por el equipo a las cuales se tuvo acceso.

A continuación se muestran los resultados obtenidos:

Modelo	API USB Host presente para 3rd party apps	Porcentaje de positivos
Samsung n8000	Si	55,2 % (42/76)
Ledstar AQ8	Solo rooted	(2/2)
ACER B1	No	0% (1/1)
Asus TF300T	Si	70% (46/65)
Lenovo A2107	No	0% (1/1)
Samsung PT5100	Si	44,7% (30/67)
Google Nexus 7	Si	50%

2.4.2. WiFi

Hoy en día el hardware del robot Butiá no incluye una SBC y como la placa USB4Butiá no soporta el manejo de un dispositivo USB, para incluir un dongle WiFi, la posibilidad de establecer una conexión inalámbrica entre los nodos principales del sistema se sujeta a que se realice una expansión o agregado al kit de la plataforma Butiá. Este consta de incorporar una SBC con sistema operativo de propósito general y puertos USB que permitan agregar dispositivos de WiFi (dongle) así como conectar la USB4Butiá a la placa SBC. Para las siguientes secciones se asumirá que existe una Raspberry Pi como la detallada en la sección *"Complementos"* de *"Plataforma robótica"*. Un diagrama sencillo del despliegue de los componentes sería el mostrado en la figura 11.

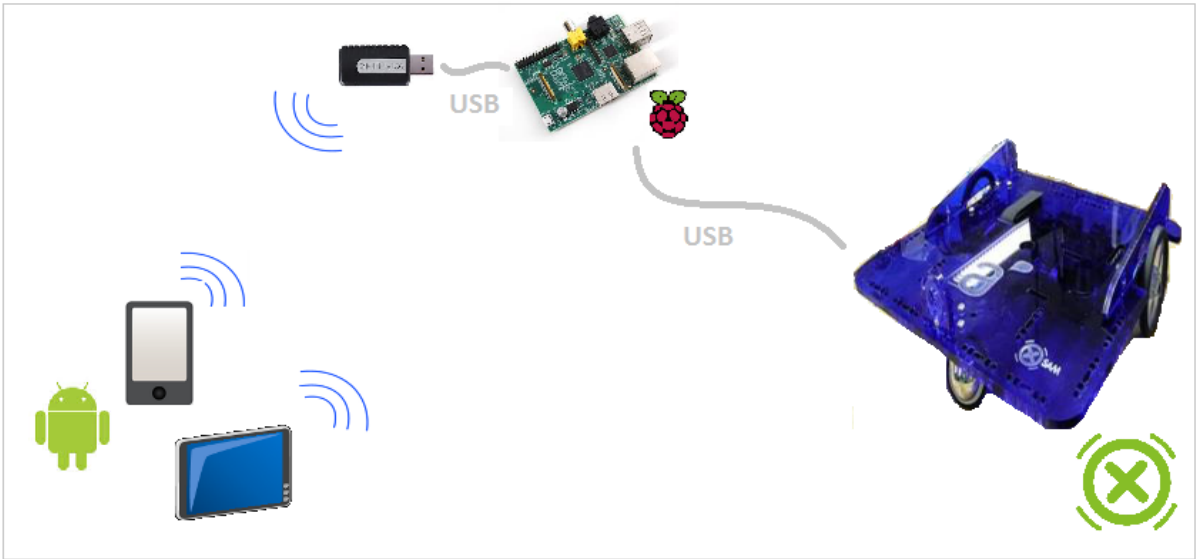


Figura 11: Diagrama de componentes haciendo uso de la placa Raspberry Pi.

En el marco de este proyecto como ya se identificó en múltiples ocasiones existirán dos actores principales: los dispositivos móviles (usuarios) y el robot Butiá. El primero dispone de soporte para conexiones WiFi, tradicionalmente reconocido por los mercados tecnológicos actuales y por las características de los sistemas operativos, en particular Android. El segundo (robot Butiá) no lo soporta actualmente este tipo de conexión, sin embargo se tomará como hipótesis que el kit del robot se ve expandido con una SBC que soporte USB y tenga un sistema operativo de propósito general (como por ejemplo Raspbian). Asumiendo esto, se incluirá un dispositivo dongle WiFi que permite a la SBC establecer una conexión WiFi con los dispositivos móviles.

Para establecer una comunicación existen varias opciones: Una es asumir que ambas están en una red WLAN controlada por un router o WAP (access point) y que el usuario establezca de manera directa la IP correspondiente al servidor alojado en la SBC. Otra es mediante la utilización de WiFi Direct, tecnología que permite el establecimiento de una conexión peer to peer de manera segura. También, se puede utilizar una red Ad-Hoc entre el dispositivo móvil y la SBC.

Si planteamos una arquitectura en la cual asumimos que ambos nodos de la conexión se encuentran bajo una red WLAN controlada por algún manager estamos introduciendo de antemano una limitante en nuestro sistema; la necesidad de que exista un tercer actor que es el administrador de la conexión como puede ser un router. No existe mucha diferencia entre las dos últimas opciones de comunicación WiFi que marcamos. Por un lado WiFi Direct ofrece un nivel más alto de seguridad encriptada que Ad-Hoc (WPA2) y además permite que cualquier nodo que participe en la conexión pueda conectarse a su vez, a otras redes inalámbricas al mismo tiempo. Existen dos problemas principales que identificamos para estos últimos dos escenarios: en primer lugar para ofrecer los servicios deseados de búsqueda e identificación de potenciales equipos (hotspot) está disponible solo en versiones de Android iguales o superiores a la 4.0 y en segundo el dispositivo dongle WiFi debe

tener ciertas características especiales para brindar una conexión Ad-Hoc, dependiendo exclusivamente del hardware del mismo. El usuario del sistema puede utilizar cualquiera de las opciones aquí planteadas para establecer una conexión inalámbrica con el servidor.

2.5. Concurrencia y orientación a comportamientos

Los distintos comportamientos del robot que sean programados en este IDE deben poder coexistir y ejecutarse de manera concurrente. Si se asume la arquitectura robótica basada en prioridades, las tareas programadas deben tener un determinado valor de prioridad y su ejecución debe ser acorde y coherente a las prioridades establecidas.

Se deduce de lo anterior que las tareas deben ser capaces de interrumpirse entre sí, es decir, que si se dan las condiciones para que se ejecute una tarea de mayor prioridad, la de menor prioridad postergue su ejecución y de paso a la de mayor prioridad. Para contemplar este problema hay muchas opciones, de las cuales fueron consideradas principalmente dos.

2.5.1. Semáforos y Threads en Android

El SDK de Android posee diversas herramientas para la creación y administración de threads [32]. Si se considera que la solución está auto contenida en el dispositivo Android, se puede manejar los comportamientos en forma de threads sincronizados mediante semáforos utilizando las herramientas brindadas por Android SDK. Una idea sería por ejemplo tener una tarea *"manager"* que se encargue de la administración de la ejecución de las tareas creadas por el usuario tomando en cuenta las prioridades de este.

```
new Thread(new Runnable() {
    public void run() {
        //do stuff
    }
}).start();
```

Figura 12: Ejecución de tareas en nuevo thread con Android SDK.

En la figura 12, se puede observar un ejemplo muy sencillo de ejecución de una tarea en un thread nuevo. Se puede sincronizar a cada tarea que sea creada con el administrador *"manager"* utilizando semáforos, ofrecidos por el Android SDK. En la figura 13 se muestra un ejemplo sencillo de creación y manejo de un semáforo simple.

```
private static final int MAX_AVAILABLE = 100;
private final Semaphore available = new Semaphore(MAX_AVAILABLE, true);

public void ActivateButiaLaserGun() throws InterruptedException {
    available.acquire();
    //..
    //Do stuff
    //..
    available.release();
}
```

Figura 13: Ejemplo de creación y utilización de semáforos en Android SDK.

2.5.2. Toribio y Lumen

Lumen [41] es un entorno creado por Jorge Visca para ejecuciones multitarea basadas en corutinas. Consiste básicamente en un scheduler, inspirado en la descripción del scheduler de Sierra. Lumen no posee dependencias ni código de C y corre en Lua sin modificaciones (funciona con Lua 5.1, 5.2 y LuaJIT). Posee además herramientas de logging, remote shell y web server, detalladas posteriormente.

Al crear tareas usando Lumen éstas son agregadas al scheduler para su ejecución. Se maneja el concepto de señales, una señal es un evento emitido por una tarea, estando centralizado el manejo de señales por el scheduler de Lumen. Una tarea (task) en particular puede bloquearse esperando por una señal emitida por una o más tareas como se muestra en la figura 14, o iniciar su ejecución cuando esto sucede.

```lua
local sched = require 'sched'
-- task emits signals
local emitter_task = sched.run( function()
                                    for i = 1, 10 do
                                        sched.signal('an_event', i)
                                        sched.sleep(1)
                                    end
                                end
                               )

-- task receives signals
sched.run( function()
               local waitd = {emitter=emitter_task, events={'an_event'}}
               while true do
                   local _, _, data = sched.wait(waitd)
                   print (data)
               end
           end
         )

sched.go()
```

Figura 14: Ejemplo de una tarea activando a otra mediante un signal.

Toribio [42] es un entorno de desarrollo de aplicaciones de robótica para plataformas embebidas. Está construido alrededor de Lumen y provee un entorno que simplifica el desarrollo y despliegue de aplicaciones robóticas. Entre los servicios que provee están:

- Un sistema centralizado de configuración.
- Objetos que abstraen el hardware disponible ("devices").
- Repositorio central de tareas, devices.

La ventaja principal que otorga trabajar con Toribio es la creación organizada de tareas centralizadas que se comuniquen entre sí a través del scheduler de Lumen con signals, permitiendo establecer de manera simplificada la comunicación con el hardware (devices). Cada tarea a ejecutarse con Toribio se declara en un archivo de configuración en conjunto con variables o atributos propios (globales) de la tarea que también pueden declararse allí. Toribio incluye a Bobot como uno de los devices por defecto creados, lo cual permite que cualquier tarea creada pueda comunicarse con Bobot de manera sencilla y obtener fácilmente información de los sensores así como manipular los actuadores. En la figura 15 se puede observar un ejemplo.

```
local toribio = require 'toribio'
local sched = require 'sched'
-- Espera a que se carguen los motores y el boton (bb es por Bobot)
local motors = toribio.wait_for_device('bb-motors')
local button = toribio.wait_for_device('bb-button:2')

--Pregunta si el boton del sensor esta siendo presionado
if button.getValue() == 1 then
    --Establece la velocidad de ambos motores en 700 hacia adelante
    motors.setvel2mtr(1,700,1,700)
    --Libera el control del procesador
    sched.sleep(2)

end
```

Figura 15: Ejemplo de una tarea que mueve los motores según el valor de un sensor.

2.5.2.1. Servidor Web en Lumen

Como fue mencionado anteriormente, Lumen posee una funcionalidad muy interesante que es la capacidad de actuar como servidor web. Si se utiliza este servidor web como una tarea de Toribio, permite desde una página web poder hacer POSTs para transferir información a Toribio. Es interesante notar, que si se realiza una implementación en HTML5 se podría enviar información referente a las tareas implementadas por el usuario desde el IDE a Toribio mediante la utilización de mensajes HTTP al servidor web de Toribio para ser procesados por este.

Cabe notar que uno de los requisitos básicos que tiene la aplicación a desarrollar es la funcionalidad de debugging. Esto significa que la aplicación debe mostrar al usuario de alguna forma los bloques que están siendo ejecutados actualmente. Para esto se necesita dar feedback a la aplicación Web de lo que el robot Butiá está ejecutando. Si se realiza una implementación web, se podría hacer un POST inicial con los comportamientos desarrollados por el usuario (considerados como las tareas a ejecutar por el robot) y luego mediante polling de Ajax hacer sucesivos GETs al servidor web para mostrar un feedback de lo que está siendo ejecutado por Toribio. A primera vista, esta solución resulta costosa por la cantidad de mensajes HTTP que manejaría la aplicación, pero existen métodos para mejorar el polling. Entre ellos se encuentra el long polling, que propone por medio de Ajax Push [18] evitar los sucesivos GETs al servidor, reteniendo la respuesta del lado del mismo mientras no exista un nuevo mensaje para el cliente. De esta manera, además de reducir la cantidad de pedidos, se logra mejorar el rendimiento ya que la información desde el servidor será enviada inmediatamente al cliente.

2.6. HTML5

La iniciativa de realizar el IDE para el robot Butiá a través de la Web surge de los beneficios que este sistema engloba: potencia la portabilidad en cualquier ordenador, Tablet o dispositivo móvil, otorga independencia del sistema operativo, disminuye la cantidad de procesamiento que deben hacer los dispositivos móviles, entre otros. En contrapartida limita a desarrollar el IDE en un lenguaje soportado por los navegadores web y presenta problemas de compatibilidades para diversos dispositivos, aquí entra en juego HTML5.

La quinta versión del lenguaje HTML recoge las ventajas que introdujo XHTML y elimina muchas de las restricciones y limitaciones que tenían las versiones anteriores. Se destaca, además del código simplificado y sencillo, la facilidad para desarrollar aplicaciones que sean utilizables en dispositivos móviles tanto como en ordenadores, dando soporte a los navegadores de teléfonos y Tablets, agregando por ejemplo nuevas funcionalidades como el arrastre de objetos/imágenes (drag & drop) de importancia fundamental para los fines de este proyecto.

Si bien no obviamos la importancia de conceptos interesantes que introduce HTML5 como nuevos tags y corrección de errores, inclusión del DOM en el estándar (antes era a criterio del navegador), o contenido multimedia, vale la pena mencionar con mayor énfasis alguna de las nuevas APIs que el mismo incorpora. Una de ellas es la interfaz para WebSockets, la cual habilita la comunicación bidireccional a través de una conexión persistente, permitiendo incrementar la interactividad entre cliente y servidor o múltiples clientes (y/o dispositivos), por lo tanto permite conexiones "cross-device" en tiempo real. Otra ventaja interesante es la inclusión de Web Storage. Esta introduce una nueva forma de persistencia client-side, que originalmente solo era posible a través de cookies. Web Storage permite almacenar información en el browser sin fecha de caducidad para dichos datos y ofrece un tamaño máximo mucho mayor que las cookies, siendo este de hasta 5mb. Existen dos formas de Web Storage: una es localStorage donde la información no caduca; la otra es sessionStorage donde la información se pierde al cerrar la ventana o tab. La forma de utilizarlo es mediante un objeto string donde se puede almacenar información en formato JSON.

2.7. Lenguajes de programación visual

Dentro de la robótica educativa, uno de los desafíos más importantes es lograr, para el control del robot, una interfaz gráfica amigable y clara que permita de manera intuitiva lograr códigos de programación. La psicología educativa puede ayudar a definir las características deseables de un sistema de educación asistida por computadora y fortalecer el éxito de la herramienta creada en este proyecto. Se debe orientar la interfaz gráfica hacia un lenguaje de programación visual basado en bloques, de tal manera que conserve y posea las ventajas pedagógicas que plantea dicho paradigma.

La interfaz es el verdadero determinante del éxito de este tipo de herramientas pedagógicas: si esta no cumple con las características anteriormente mencionadas, la experiencia educativa puede fracasar en su objetivo. Es por esto que destacamos la interfaz gráfica y la interacción con el usuario como uno de los problemas grandes a solucionar.

En los lenguajes de programación tradicionales, se utilizan líneas de texto para codificar los programas. Estas líneas de texto se consideran como de dimensión única. Por otro lado, los lenguajes de programación visual (LPV) utilizan más de una dimensión para transmitir la semántica, donde tales dimensiones adicionales son adquiridas con el uso de objetos (un ejemplo son los bloques) y donde cada uno de estos objetos potencialmente significativos son símbolos, al igual que en los lenguajes de programación tradicionales cada palabra es un símbolo [3]. Estos poseen muchas características favorables: es más fácil tener una idea general de la estructura del programa, la percepción en dos dimensiones es más natural y eficiente que la lectura de texto, es más fácil de leer puesto que se reducen los elementos puramente sintácticos, la visión humana está optimizada para información multidimensional, entre otras. Gracias a esto siguieron surgiendo nuevos sistemas de esta índole, muchos de ellos con propósito educativo, siendo Scratch [21] uno de los más importantes por su utilización. En el ámbito local, en el 2010, con la aparición del plugin Butiá para TortugArte comienza a manejarse el concepto de un IDE para el control del robot Butiá. TortugArte está inspirado en los conceptos de Logo, sistema impulsado por Seymour Papert, quien propuso desarrollar una nueva forma de ver el uso de la computadora para la educación [7]. Y también refleja las características visuales de Scratch [21], persiguiendo sus mismos fines pero incorporando interfaces gráficas modernas y continuando los lineamientos de Sugar. Poniendo al alcance de niños conceptos de programación, mediante una interfaz gráfica icónica donde cada instrucción se mapea como un bloque. El proyecto Butiá realizó modificaciones sobre TortugArte agregando algunos plugins en forma de paletas que permiten controlar diferentes kits robóticos. A este plugin agregado a TortugArte se lo denominó TortuBots (figura 16) [37][38]. Con este complemento se permite a los alumnos programar usando un LPV distintos comportamientos para controlar al robot Butiá.

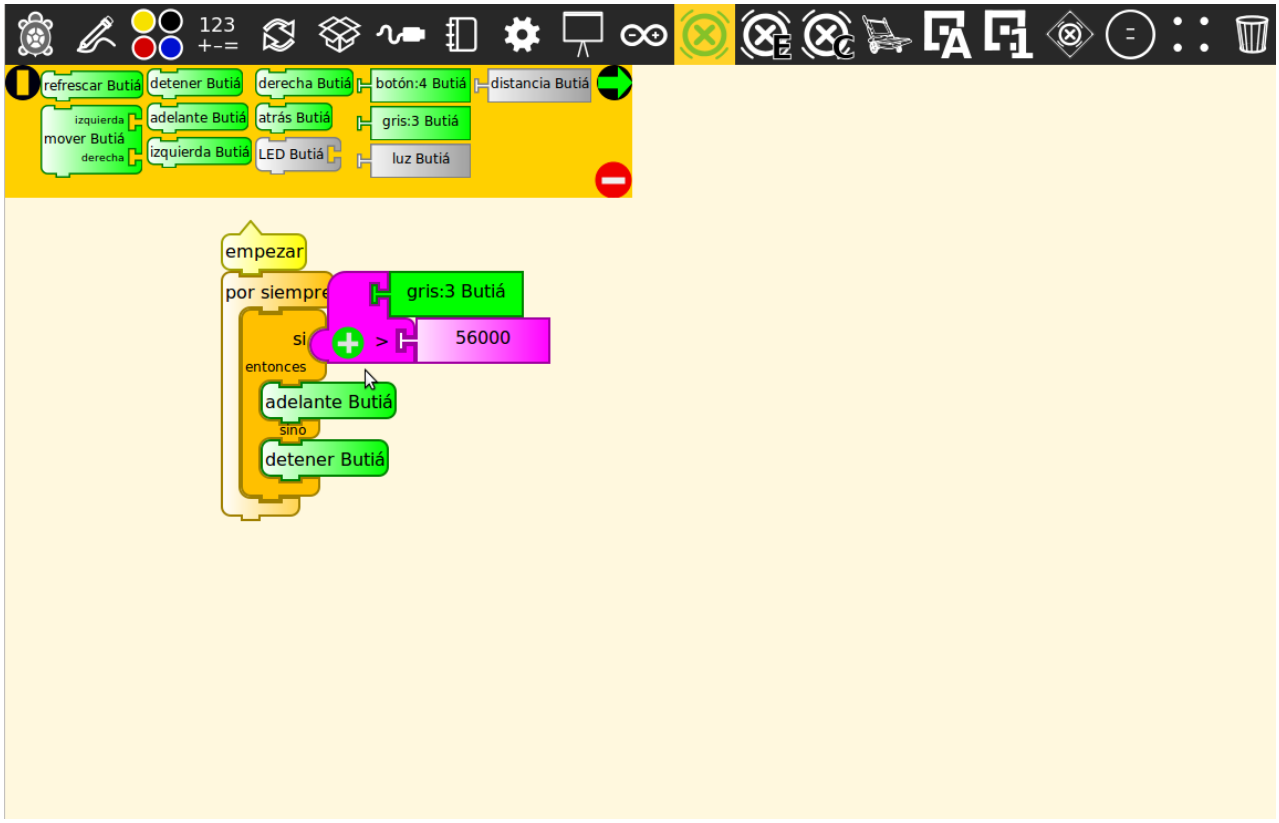


Figura 16: Ejemplo de código en Tortubots.

Para obtener un panorama actual en ésta área, fueron analizadas diversas bibliotecas modernas que modelan LPV con las características requeridas. Tales como Scratch 2.0, Blockly, Waterbear, Snap!, Design Blocks, entre otras bibliotecas de código abierto con la finalidad de facilitar la construcción de la interfaz gráfica del sistema. A continuación se detalla la biblioteca Blockly usada para modelar el LPV del sistema de este proyecto. Se decidió no abordar las bibliotecas descartadas para no reiterar innecesariamente información (consultar documento Estado del arte [25]).

2.7.1. Blockly

Blockly [14] es un LPV open source creado por Neil Frase, Quynh Neutron, Chris Pirich, Ellen Spertus y Phil Wagner, desarrolladores en el marco Google Code, que permite manipular y arrastrar bloques para construir aplicaciones, sin necesidad de typeo, basada en Scratch. Brinda a sus usuarios la posibilidad de crear programas simples.

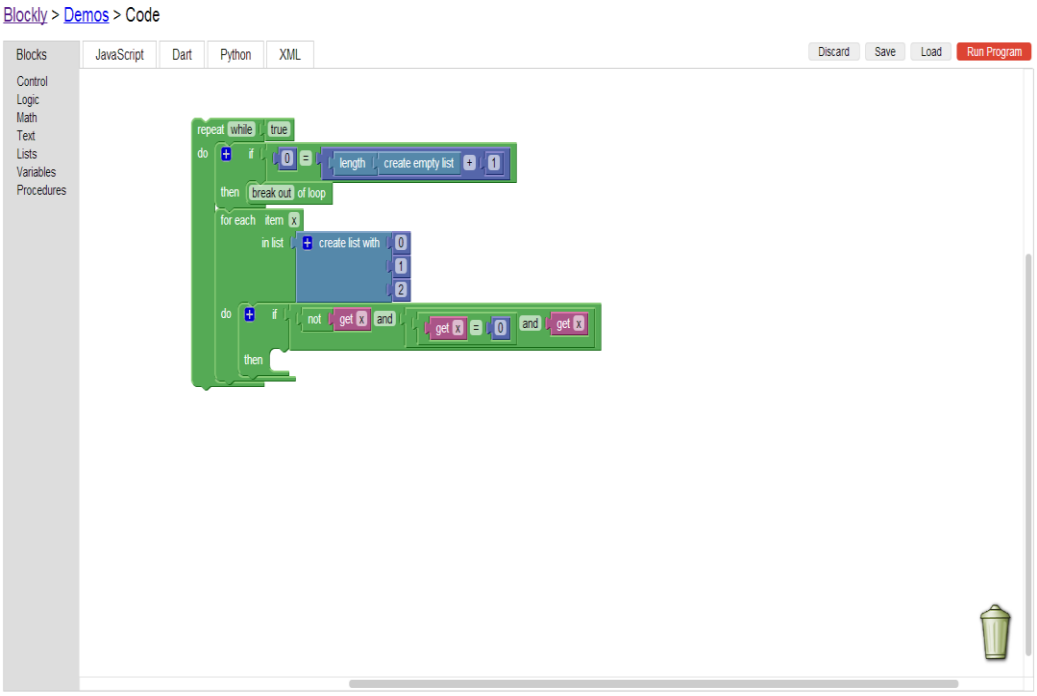


Figura 17: Interfaz de Blockly.

Está implementado en JavaScript y utiliza SVG (Scalable Vector Graphics) para renderizar los bloques. Diseñado para ejecutarse client-side en una página Web, dando la posibilidad de generar a través de los bloques código JavaScript, XML, Dart o Python, que puede ser ejecutado independientemente. Blockly presenta varias ventajas sobre sus similares: una codificación sencilla y extensible (pensada para crear apps nuevas a partir de ella, varios ejemplos disponibles en su repositorio), buena velocidad de renderizado de los bloques para el drag & drop y mayor estabilidad, desde lo visual resulta amigable como se observa en la figura 17 (incluso comparando con Waterbear, Snap! o Design Blocks). Por último, al estar basada en SVG es compatible con la gran mayoría de los browsers actuales, para Android a partir de la versión 2.3. El proyecto está actualmente activo y tiene como desventaja que los desarrolladores de Blockly no intentan mantener la compatibilidad hacia atrás de este, lo cual hace que actualizar a nuevas versiones necesite de una reescritura parcial del código.

2.8. Conclusiones del Estado del Arte

En esta sección se enumeran las decisiones tomadas por el grupo para afrontar los desafíos más importantes de este proyecto y su justificación. Para esto se utilizará como referencia la información presentada en el estudio del estado del arte.

2.8.1. Portar un IDE anterior o crear uno nuevo

Una de las primeras decisiones que debió tomar el equipo, fue decidir si crear un nuevo sistema o portar uno anterior. La opción principal, para el caso de portar un IDE existente, es la del sistema eButiá. Para lograr esto es necesario encontrar una forma de portar Etoys y Squeak en el sistema operativo Android.

Se analizaron las distintas posibilidades en el documento de Estado del Arte [25] y se llegó a la conclusión de que el proyecto más avanzado con ese objetivo es Cogdroid, un proyecto aún activo que pretende portar Squeak en Android. Sin embargo Cogdroid no soportaba Etoys (por lo tanto tampoco eButiá), además se mostraba inestable al probar el sistema con Squeak.

Aun así, las opciones de portar un IDE anterior presentaban problemas inherentes a los proyectos: la interfaz gráfica portada está orientada a escritorio, lo cual presenta dificultades frente a la resolución de la pantalla y al input del usuario (touch), pudiendo esto resultar inmanejable o incómodo. Se constató que las máquinas virtuales como CogDroid consumen una gran cantidad recursos de los dispositivos, enlenteciendo la ejecución. Como punto final, si se intentara portar Etoys, se suma la complejidad de extender el código del proyecto CogDroid que resultó extremadamente extenso y poco comprensible, con poca e incompleta documentación. Lo cual implica una tarea muy riesgosa extender el proyecto mencionado, con pocas garantías de éxito para el tiempo que conlleva un proyecto de grado.

Esto condujo al equipo a plantear el diseño e implementación un nuevo entorno de desarrollo. Considerando que de esta forma se poseen mayores garantías de éxito que con la opción de portar eButiá y permite realizar correcciones, reuniendo las ventajas y desventajas de las experiencias anteriores.

2.8.2. Subsumption vs Arquitectura basada en prioridades

Se analizaron las diferentes arquitecturas reactivas y se tomó la decisión de elegir una que fuese intuitiva con respecto a la forma de pensar cotidiana de los estudiantes. El objetivo es estimular la atracción por el paradigma reactivo, permitiendo modelar la manera de actuar del robot de acuerdo a su propia forma de pensar. Es por esto que de inmediato se descartaron arquitecturas como Campos de Potencial, que requieren una forma de pensar compleja que involucra el uso de vectores de fuerza, concepto que aún no manejan muchos de los estudiantes a los cuales apunta el proyecto. Así mismo, la arquitectura subsumption a nuestro criterio resulta difícil de comprender para estudiantes que dan sus primeros pasos en la robótica, por ejemplo: el hecho de que capas superiores suplanten o subsumen las salidas de capas inferiores o el hecho de comprender la forma de actuar como una

separación en capas de mayor nivel y de menor nivel, son conceptos bien conocidos para quien estudia computación pero no tanto para estudiantes escolares o liceales. Además, el hecho de que la acción que realice el robot sea el producto de la ejecución de varios comportamientos actuando de forma simultánea puede hacer que predecir el resultado de este sea difícil. Es por esto que se decidió utilizar una arquitectura basada en prioridades, ya que el concepto principal de la arquitectura resulta simple para explicar y es fácil predecir la salida del robot.

2.8.3. HTML5 vs Android App

Las ventajas de implementar una aplicación web en HTML5 contra desarrollar una app para Android son varias. Por un lado, se amplía la compatibilidad a un mayor rango de dispositivos, tanto computadoras como Tablets y celulares con diferentes sistemas operativos. Si fuese una app nativa sería únicamente para dispositivos Android. Además, que Yatay sea una aplicación web facilita la distribución del sistema. De otra forma debería ser descargado e instalado antes de poder usarse, ocupando recursos del dispositivo. Otra ventaja es que al existir un servidor web, todo el procesamiento no se realiza en el dispositivo Android, sino que se reparte entre este y el servidor. Esto significa una mejor utilización de los recursos del dispositivo cliente. Por otro lado, la arquitectura web centraliza su lógica en el servidor permitiendo la integración de múltiples usuarios simultáneos sin tener que mantener un estructura peer to peer (inapropiada en esta realidad) para lograrlo.

2.8.4. Lumen y Toribio

En las etapas tempranas del proyecto se tomó la decisión de utilizar en el sistema Lumen (entorno cooperativo multitarea) y Toribio (plataforma para crear aplicaciones robóticas) que centraliza los beneficios de Lumen junto con abstracciones del hardware y configuraciones. Estos ofrecen muchas funcionalidades deseadas por el sistema, programadas en el lenguaje Lua y diseñadas para conformar un entorno que permita crear sistemas embebidos para robótica. Ofreciendo facilidades también para interactuar con Bobot y con la plataforma Butiá. Principalmente, las características destacables que utilizamos en el desarrollo de este proyecto son: un scheduler para crear, sincronizar y finalizar tareas, la comunicación entre tareas (por medio de signals), un servidor web liviano y rápido, integración con Bobot, catálogos de memoria compartida, entre otros.

2.8.5. Blockly

La interfaz gráfica del proyecto, debía utilizar un lenguaje de programación visual (LPV) basado en bloques, que sea amigable y claro, y que permita a personas sin conocimientos de sistemas lograr, de manera intuitiva, códigos de programación simples para controlar el robot Butiá. Para esto se analizaron varias alternativas como se especificó en el documento Estado del Arte [25], de las cuales se descartaron algunas por motivos que allí se detallan. Finalmente se decidió utilizar Blockly ya que dentro de las bibliotecas de LPV para HTML5 fue la que ofreció un rango más amplio de compatibilidad, ya que funcionó de forma correcta en las pruebas realizadas en Android Browser (v3.0 y superiores), Firefox versión mobile y de escritorio, Chrome versión mobile y de escritorio, Safari y Opera. Además se mostró más simple en el código, más rápido y mejor visualmente que otros como Waterbear o DesignBlocks, siendo más liviano (y estable) que Snap!.

2.8.6. Lua vs Python

A la hora de elegir un lenguaje de scripting para implementar el back-end debimos elegir entre los dos más potentes utilizados en contextos similares a los del proyecto: Lua y Python. Tanto Lua como Python son dos lenguajes reconocidos por la comunidad, donde cada uno tiene sus propias ventajas y desventajas. Existen numerosos debates en distintos sitios web que comparan los lenguajes, sin embargo lo que nos permite tomar una decisión clara es las características de hardware y el contexto en los cuales se enmarca este proyecto de grado. El robot está conformado con placas que poseen en general pocos recursos de memoria y de procesamiento, además los comportamientos deben ejecutarse muy rápido, ya que el robot debe responder en tiempo real a los valores arrojados por los sensores que posee. Es por esto que decidimos utilizar Lua. Por un lado Lua es extremadamente liviano, su núcleo posee alrededor de 17000 líneas de código C, su binario ocupa alrededor de 200 KB en la versión 5.1 mientras que Python ocupa 2 MB [36]. Además, Lua es mucho más compacto en el tamaño ocupado en memoria principal que Python, aunque este último sea más rico en las bibliotecas que ofrece. Por otro lado, Lua es rápido, tanto el interpretador como el JIT y algunos Benchmarks lo posicionan por encima de Python [17]. Por último, Lumen y Toribio nos ofrecían muchas funcionalidades deseadas para nuestro sistema, y éstos están desarrollados en Lua, motivando la utilización éste lenguaje para codificar la lógica del servidor.

Parte II

Desarrollo del Sistema

Capítulo 3

Requerimientos

Se resumen en esta parte del informe, los requerimientos identificados para el sistema que se propone en este proyecto y que fueron detallados en profundidad en el documento ***"Especificación de Requerimientos"*** [24]. Su propósito es brindar un acercamiento a la descripción del sistema, detallando el alcance del mismo según lo relevado en el documento de presentación de proyecto y en las reuniones de relevamiento de requerimientos con los tutores del proyecto, además de lo definido en el análisis del estado del arte. Estos requerimientos fueron aprobados por los tutores y utilizados como forma de validación del sistema.

3.1. Alcance

El proyecto consiste en la realización de un entorno de desarrollo (IDE) para la implementación de comportamientos robóticos usando un lenguaje de programación visual (LPV). El mismo, debe ser soportado por el sistema operativo Android y estar orientado a alumnos y docentes de primaria y secundaria. Debe poder programar tareas que controlen a la plataforma robótica Butiá utilizando una arquitectura perteneciente al paradigma reactivo.

La metodología de desarrollo de los comportamientos debe ser mediante la construcción de figuras (bloques) que representen con simplicidad código de programación robótica. Dicha funcionalidad, deberá estar inspirada en sistemas como TortuBots y Scratch. Dentro de los dispositivos con sistema operativo Android estará principalmente orientado a las Tablets. El sistema debe permitir ejecutar los comportamientos implementados y poseer la funcionalidad de debugging, en el sentido de que debe ilustrar al usuario en tiempo real que parte del código está ejecutando el robot Butiá.

3.2. Requerimientos funcionales

3.2.1. Entorno y Ejecución

Debugging: El sistema deberá permitir al usuario ver qué bloques implementados están siendo ejecutados en tiempo real, enlenteciendo la ejecución, si fuese necesario, de forma que el usuario pueda apreciar gráficamente la ejecución de un determinado comportamiento implementado.

Múltiples usuarios: Deberá permitir que cada usuario desarrolle un comportamiento y que lo envíe para su ejecución al servidor donde se ejecutarán en conjunto según la arquitectura basada en prioridades definida en el documento Estado del Arte [25]. No será un requisito trabajar de manera cooperativa sobre el mismo comportamiento.

3.2.2. Robot

Sensores Genéricos: La aplicación tendrá alguna funcionalidad que permita la utilización de sensores nuevos que sean incluidos en el kit Robótico.

Calibración del Robot: El sistema debe proveer una funcionalidad que permita la calibración de los sensores y actuadores del robot Butiá. Esto permite que el usuario pueda comprobar el estado de los sensores y actuadores.

3.2.3. Proyectos

Gestión de proyectos: En la aplicación se deben presentar herramientas con las funcionalidades de gestión de proyectos: creación, guardado y apertura.

Pantalla de gestión: La ubicación de las funcionalidades de gestión dentro del sistema deberá ser en la pantalla única principal o a lo sumo estar a un acceso de distancia, regla que se mantendrá en general para todas las funcionalidades.

Persistencia de proyectos: Se debe poder almacenar un proyecto con el código generado de cada comportamiento.

3.2.4. Comportamientos

Nombre y Prioridad: Los comportamientos creados deben estar identificados por un nombre elegido por el usuario, además de una prioridad asignada correspondiente a la arquitectura descrita en el documento Estado del Arte [25]. Tanto el nombre como la prioridad de cualquiera de los comportamientos podrán ser modificados en cualquier momento, a excepción de cuando estos se estén ejecutando.

Pantalla de comportamientos: La metodología de trabajo dentro del sistema consistirá de un espacio para la edición de comportamientos y otro espacio donde se podrán colocar minimizados los comportamientos ya creados, de forma que en cualquier momento se puedan seleccionar de manera sencilla para su edición. Esto está definido gráficamente en el documento ***"Pautas para la Interfaz de Usuario"*** [27].

3.3. Requerimientos no funcionales

Entorno de ejecución: Debe ejecutar en Tablets con sistema operativo Android, siendo opcional el funcionamiento aceptable en dispositivos celulares con sistema operativo Android así como otros dispositivos y/o ordenadores con sistemas operativos distintos. El servidor Web deberá correr en un sistema operativo GNU/Linux y funcionar de manera aceptable con los recursos de hardware de una SBC Raspberry Pi.

Orientación pedagógica: El sistema tendrá una orientación hacia el aprendizaje constructivista y estará basado en las ideas de TortuBots, Scratch.

Tiempo de reacción: Los comportamientos programados al ser ejecutados deben dispararse en tiempo real de acuerdo a la información obtenida de los sensores. Análogamente, el feedback de los comportamientos en ejecución en la plataforma robótica debe ser desplegado al usuario en tiempo real.

Portabilidad: El sistema debe contemplar portabilidad a la mayor cantidad de dispositivos móviles que cumplan con las restricciones especificadas, haciendo énfasis en los dispositivos Android.

Escalabilidad: El sistema debe permitir agregar nuevas funcionalidades sin encontrar limitaciones desde la codificación, concretamente debe ser posible incorporar nuevos bloques para controlar sensores que se agreguen en la plataforma robótica, debe ser extensible la lógica que define la arquitectura robótica, entre otras.

Mantenibilidad: El sistema debe poder adaptarse a cambios que se generen tanto en la plataforma robótica como en la aplicación extendida para la interfaz gráfica.

Multilenguaje: El sistema debe permitir la configuración del lenguaje desplegado por la interfaz gráfica del mismo, dando la posibilidad en principio de visualizar el texto en español e inglés.

Modularización: El sistema debe mantener un nivel de modularización tal que le permita con facilidad realizar los puntos de Mantenibilidad y Escalabilidad, obteniendo como resultado una aplicación creada en módulos que se adaptan a los cambios de las bibliotecas que usan así como también permiten la re-implementación o extensión de un módulo en particular sin que los demás resulten afectados.

Paradigma robótico: La aplicación debe permitir la construcción de comportamientos robóticos que estén basados en el paradigma reactivo, siguiendo en dicha construcción y ejecución la arquitectura basada en prioridades definida en el documento Estado del Arte [25].

Kit Robótico: El robot al que estará orientado este sistema, es principalmente el robot Butiá definido en el documento Estado del Arte [25]. Sin embargo, se debe mantener una cierta modularización que permita extender el sistema a otros kit robóticos.

Simplicidad: Las herramientas que ofrece el sistema deben orientar la implementación a comportamientos sencillos que estén basados en el paradigma reactivo.

Capítulo 4

Prototipos y pruebas realizadas

En este capítulo se contemplan las distintas pruebas realizadas para valorar la viabilidad de ciertas tecnologías y los posteriores prototipos construidos con la intención de mitigar en gran medida los riesgos de usar distintas bibliotecas externas (third-party libraries) para los fines de este proyecto.

4.1. Pruebas de USB Host

Objetivos: Probar la viabilidad del uso de USB Host y confirmar si algunos dispositivos Android poseen el feature de USB Host deshabilitado de fábrica.

Motivación: Lograr la comunicación de un dispositivo móvil Android con el robot Butiá. USB Host es una tecnología prometedora ya que planteaba la idea de que el dispositivo móvil establezca la comunicación y alimente al robot Butiá mediante USB.

Prueba: La prueba consistió en la implementación de una aplicación para Android que manifestara si el celular permite o no el uso de USB Host.

Resultados: Las pruebas no fueron exitosas. Se obtuvo como resultado que sólo una minoría de los dispositivos Android probados poseían disponible la API de USB Host para el uso de 3rd party apps (aplicaciones no nativas). El resto o bien se debía agregar la API con permisos de root o directamente no es soportada. Todos los detalles de la investigación sobre USB Host se encuentran en la sección de USB Host del documento Estado del Arte [25].

4.2. Prueba del servidor web y creación dinámica de tareas.

Objetivos: Probar la viabilidad del uso del servidor web de Lumen y la creación de tareas sincronizables, utilizando Toribio y el scheduler de Lumen.

Motivación: La opción de un IDE web basado en HTML5 planteaba: Por un lado, la necesidad de tener un servidor web liviano (dados los recursos del hardware) que corriera sobre un S.O Unix. Por otro la necesidad de que a partir del código Lua (en formato texto plano), enviado al servidor web desde los dispositivos móviles, sea posible crear y eliminar dinámicamente tareas (que

representan los comportamientos robóticos) sincronizables, utilizando el scheduler de Lumen y Toribio.

Prueba: La prueba consistió en la implementación de una página web y una instancia del servidor web de Lumen en las tasks de Toribio. Se integró el scheduler de Lumen para la parte server-side de la web.

Resultados: Los resultados fueron muy positivos, se pudo de forma exitosa enviar texto para ser parseado como código Lua y crear tareas sincronizables con Lumen y Toribio a partir de este. Se logró tanto crear como, matar las tareas, observando que la utilización de Lumen y Toribio es una opción viable para desarrollar el IDE en cuestión.

4.3. Prueba de Waterbear como LPV

Objetivos: Probar la viabilidad del uso de Waterbear como LPV, observando su compatibilidad con los distintos dispositivos móviles y los navegadores así como su extensibilidad.

Motivación: La opción de un IDE web basado en HTML5 contempla la utilización de un LPV extensible para poder adaptarlo al paradigma y al kit robótico. Que permita al usuario modelar de forma fácil la estructura de los comportamientos.

Prueba: La prueba consistió en levantar con el servidor web de lumen, la última versión al momento de Waterbear y probarla en los distintos dispositivos.

Resultados: La versión de Waterbear probada no funcionó en ninguna versión de Android Browser, ni en la versión mobile de IE entre otros. Se reportó el bug a los desarrolladores de Waterbear sin embargo, estos anunciaron que no intentarían implementar dicha compatibilidad por el momento. Este motivo y el hecho de que se observó que era engorroso de expandir y con algunas características visuales desfavorables para dispositivos móviles, como ser la extensión en ancho de cada bloque envolvente al crecer sus bloques hijos, motivo al equipo a buscar otras alternativas y descartar a Waterbear como LPV a utilizar.

4.4. Prueba de Blockly como LPV

Objetivos: Probar la viabilidad del uso de Blockly como LPV, observando su compatibilidad con los distintos dispositivos móviles y los navegadores así como su extensibilidad.

Motivación: La opción de un IDE web basado en HTML5 contempla la utilización de un LPV extensible para poder adaptarlo al paradigma y al kit robótico. Que permita al usuario modelar de forma fácil la estructura de los comportamientos.

Prueba: La prueba consistió en levantar con el servidor web de Lumen, la última versión al momento de Blockly y probarla en los distintos dispositivos.

Resultados: La prueba se realizó en dos ocasiones durante este proyecto, con resultados muy distintos. Una fue realizada en mayo del 2013 aprox., en esta fecha, la utilización de Blockly como LPV había sido prácticamente descartada debido a que los eventos de drag & drop no se estaban capturando para cualquiera de las versiones de Android Browser en las que se probó e incluso algunos otros navegadores mobile. La segunda prueba hecha entre Agosto y Septiembre del 2013, surge debido al éxito que obtiene el grupo con pruebas sobre un LPV llamado Anwide, que luego de estudiado se observó que era en realidad una versión vieja de Blockly modificada, lo cual llevó a pensar que los problemas encontrados no eran originales del LPV sino que fueron introducidos en alguna versión posterior de Blockly. Luego de esto, el grupo retoma las pruebas con Blockly y analiza los foros de corrección de bugs, encontrando que los problemas que se habían reportado originalmente fueron corregidos en las últimas versiones de Blockly. Como resultado de las últimas pruebas el grupo elige Blockly como LPV, por su compatibilidad con los navegadores modernos de Android Browser (3.0+), así como Firefox, Chrome, Safari y Opera en sus versiones mobile, y por distinguir facilidad para extender de los bloques y el código de la biblioteca.

Capítulo 5

Arquitectura del sistema

Este capítulo intenta detallar los aspectos arquitectónicos del sistema que fueron documentados en el documento **"*Arquitectura y Diseño del Sistema*"** [23]. Las distintas secciones del capítulo muestran desde varios enfoques cómo está distribuido el software y cuáles son sus principales componentes.

5.1. Estilo arquitectónico

El sistema presenta una arquitectura distribuida en 4 capas como se observa en la figura 18.

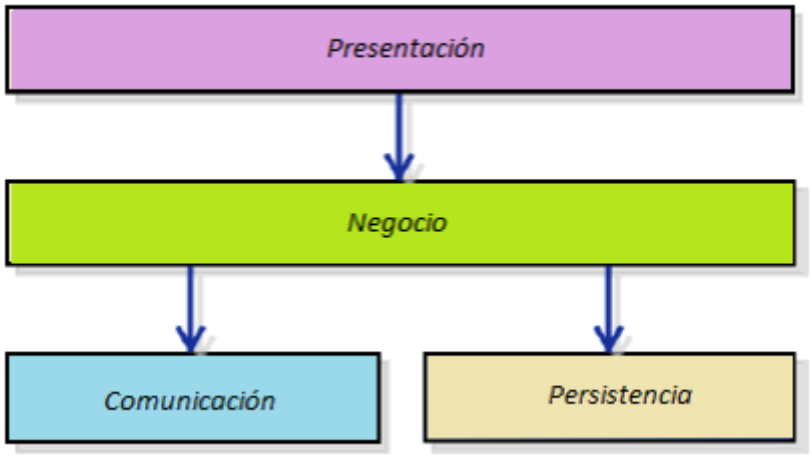


Figura 18: Diagrama de estilo arquitectónico.

Presentación: Refiere a la capa de interfaz gráfica. Es la parte del sistema encargada de interactuar con el usuario y de la presentación visual. Está compuesta por la biblioteca Blockly y los componentes web del sistema, como ser bibliotecas de JavaScript, CSS y código HTML.

Negocio: Esta capa es la que posee la lógica principal del sistema. Maneja y ejecuta los comportamientos según la arquitectura robótica reactiva definida. Formada por Lumen, Toribio y el código Lua de negocio de la aplicación.

Comunicación: Es la encargada de la comunicación entre los módulos de la capa de negocio y los sensores o actuadores del kit robótico. Está compuesta por Bobot, a través de sus adaptadores en Toribio (deviceloaders) y por el subsistema RobotInterface.lua.

Persistencia: Es la capa que posee como función la administración y el almacenamiento de los comportamientos y proyectos en la base de datos. Está conformada por una base de datos en SQLite y por el subsistema Persistance.lua.

5.2. Subsistemas

Es interesante que para los subsistemas se presenten dos diagramas distintos, ya que los comportamientos creados por el usuario se generan y ejecutan de manera dinámica por lo cual no son módulos definidos de la arquitectura y sin embargo al generarse forman parte importante del sistema. Por lo tanto mostramos dos diagramas de componentes, en la figura 19 se muestran simplemente los subsistemas estáticos del sistema y en la figura 20 se incluye dentro de la arquitectura como se integran los comportamientos generados dinámicamente.

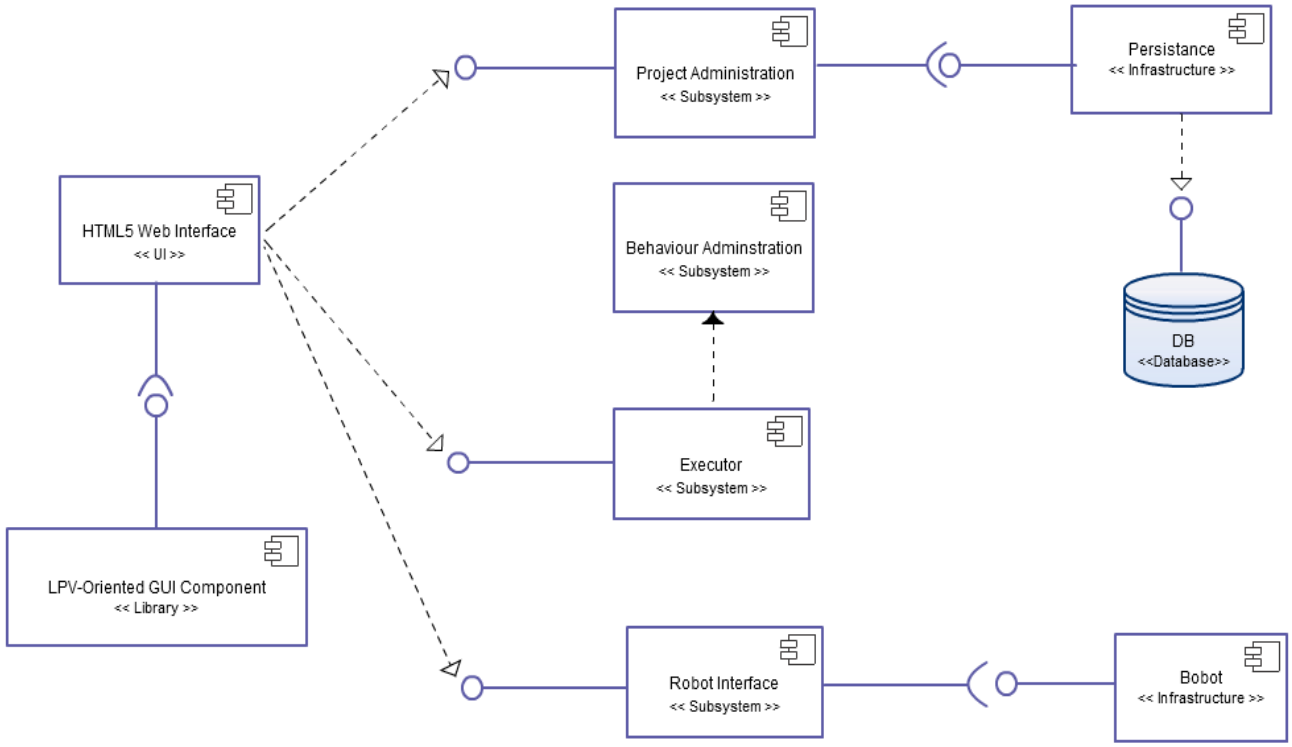


Figura 19: Diagrama de subsistema estáticos.

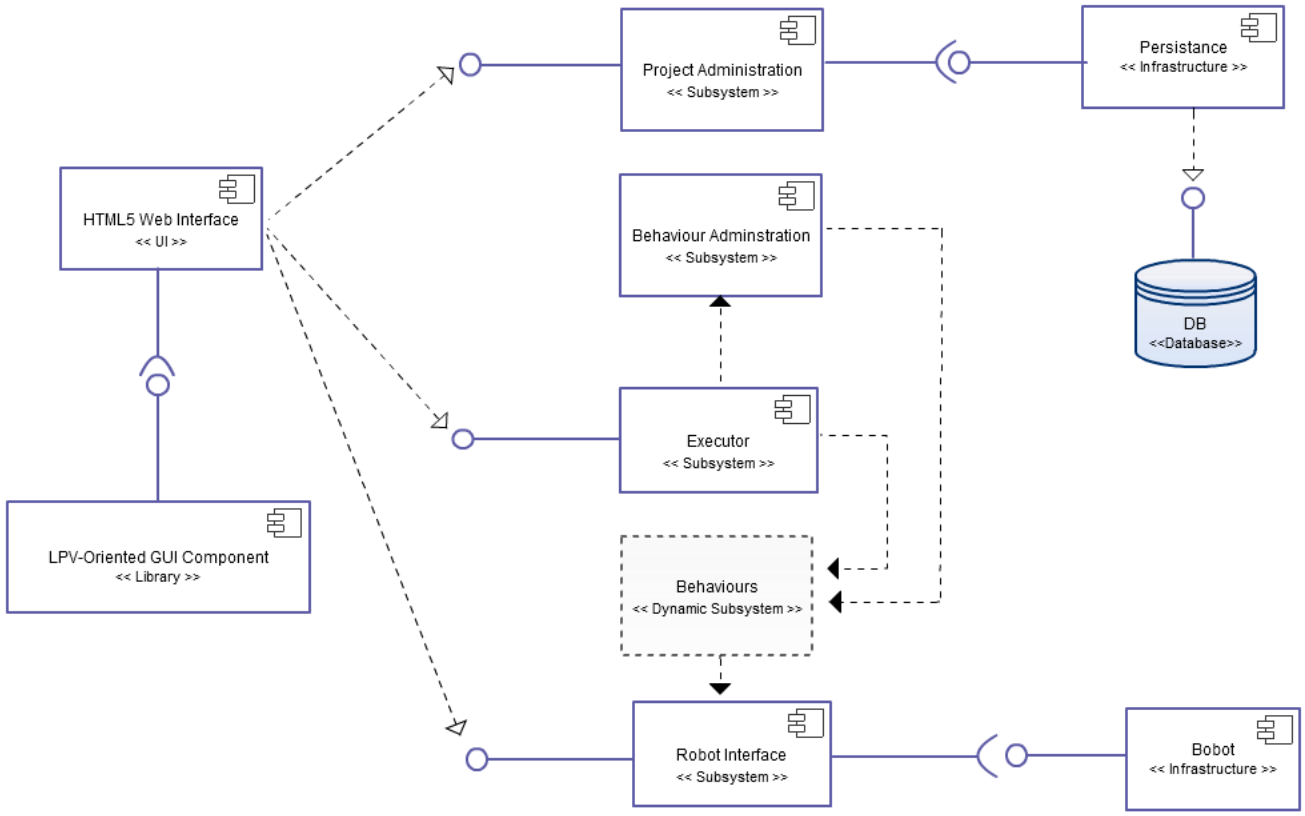


Figura 20: Diagrama de subsistemas estáticos y dinámicos.

5.2.1. Descripción

Los subsistemas diagramados y que se detallan a continuación fueron desarrollados en el marco de este proyecto a excepción del componente orientado a LPV y del sistema Bobot.

5.2.1.1. HTML5 Web Interface

La interfaz de usuario del sistema es responsable de la gestión de la interacción por medio de la Web, siendo parte de la capa de presentación. Está encargada de presentar el sistema al usuario, comunicar la información y capturar la información ingresada por este. Esta capa se comunica únicamente con la capa de negocio, y consume las prestaciones de la biblioteca LPV-Oriented.

5.2.1.2. LPV-Oriented GUI Component

Biblioteca que modela un lenguaje de programación visual (LPV) y brinda sus prestaciones para el uso por parte de la componente de interfaz de usuario del sistema. Está formada por la biblioteca Blockly.

5.2.1.3. Project Administration

Subsistema encargado de la administración de proyectos incluido en la capa de negocios, el mismo provee a la capa de presentación sus servicios y consume las prestaciones de la capa de persistencia.

5.2.1.4. Behaviour Administration

Subsistema encargado de la administración de comportamientos incluido en la capa de negocios, este subsistema modela la arquitectura robótica basada en prioridades. Encargado del task switching mientras el sistema se encuentra en ejecución.

5.2.1.5. Executor

Subsistema encargado de crear y ejecutar en Toribio los comportamientos robóticos. Ofrece a la capa de presentación los servicios para la ejecución de tareas, es decir, convertir los comportamientos creados en base al LPV en tareas de Toribio para luego ejecutar las mismas y también permite eliminar las tareas en ejecución.

5.2.1.6. Robot Interface

Este subsistema está diseñado para generalizar la comunicación con interfaces robóticas, el mismo encapsula las prestaciones de Bobot y brinda sus servicios para controlar los sensores y actuadores de plataformas robóticas. Además, parsea el archivo de configuración que contiene la información sobre el kit robótico.

5.2.1.7. Bobot

Infraestructura encargada de realizar la comunicación a bajo nivel con la placa USB4Butiá y de exponer una interfaz a través de una conexión TCP/IP para controlar los sensores y actuadores del robot Butiá.

5.2.1.8. Persistence

Infraestructura encargada de realizar la comunicación con la base de datos del sistema, la misma provee a la capa de negocio sus servicios.

5.3. Distribución

Se presenta en la figura 21 la arquitectura técnica del sistema indicando los nodos contenidos en la infraestructura tecnológica esperada y la localización de los componentes en dichos nodos. Considerando la distribución de la aplicación desde el punto de vista de los procesos, es posible identificar tres tipos de nodos; Browser, Web Server, y USB4Butiá.

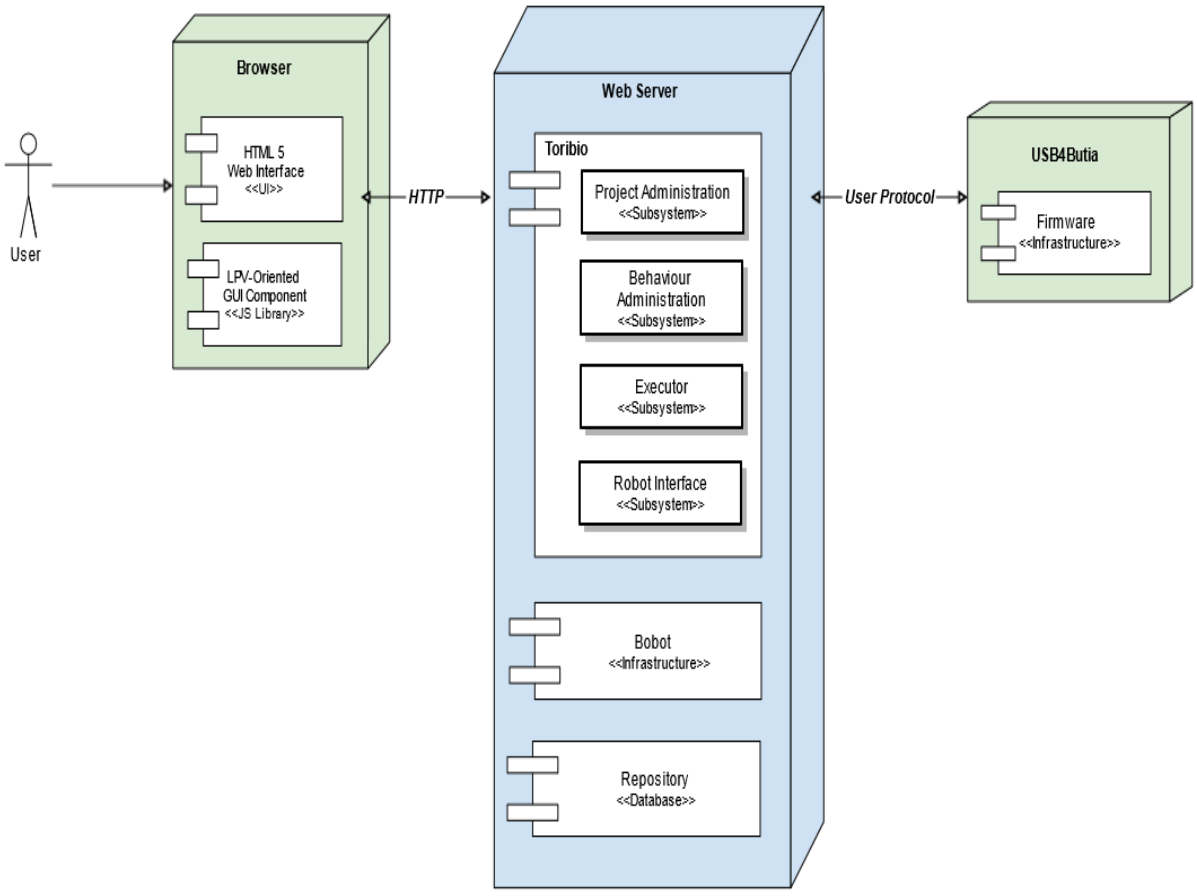


Figura 21: Diagrama de distribución.

5.3.1. Escenario: Nodos

Browser: Este nodo representa el entorno para los usuarios finales, siendo presentado el sistema a través de un navegador con la habilitación para ejecutar JavaScript. En la práctica este nodo se ejecuta en el dispositivo móvil cliente.

Web Server: Este nodo centraliza todos los requerimientos funcionales del sistema, soportado por el framework Toribio y Lumen, brindando el último los servicios para levantar un servidor web. En la práctica estará contenido en la placa Raspberry Pi.

USB4Butiá: Este nodo centraliza la comunicación con los actuadores y sensores, brindando el acceso para controlar de la plataforma robótica Butiá. En la práctica este nodo se encuentra en el robot Butiá.

Conexiones: La comunicación establecida entre el nodo Browser y el nodo Web Server es mantenida a través de HTTP, mientras que la comunicación que establece el nodo Web Server con el nodo USB4Butiá es a través de un protocolo llamado User Protocol.

5.4. Diagrama de clases

El sistema fue desarrollado en los lenguajes Lua para la parte server-side y JavaScript para la parte client-side. Como ambos son lenguajes de scripting que no están orientados a objetos, el concepto de clase carece de sentido en ambos. Sin embargo, para la parte server-side, se puede especificar un diagrama que detalle los subsistemas, ya que estos están compuestos por archivos .lua y cada uno de estos por una única tabla con funciones y atributos que se asemeja bastante al concepto de clase. Son estos y sus interdependencias los que se diagraman en la figura 22.

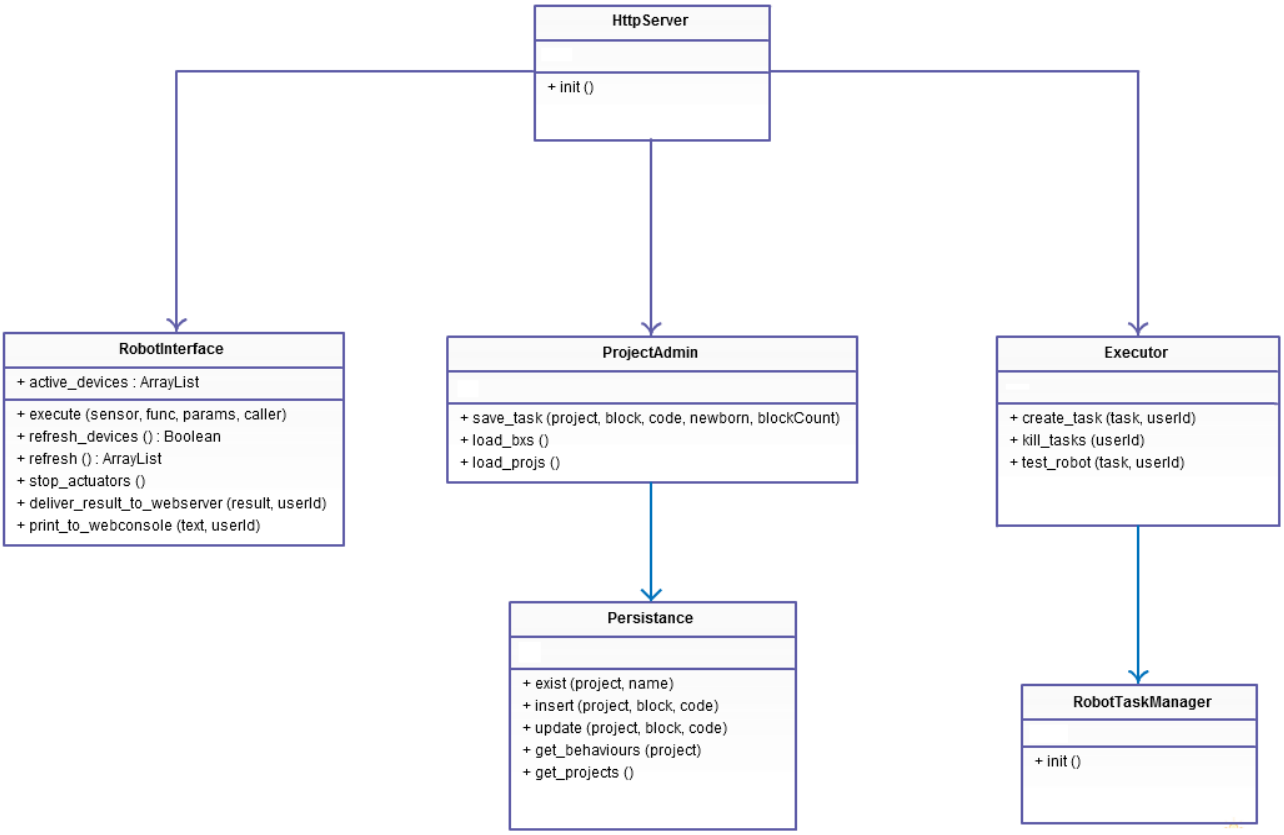


Figura 22: Diagrama de clases server-side.

Si bien los comportamientos son generados dinámicamente, estos tienen una estructura predeterminada, con funciones y atributos predefinidos. Solo variará su función *run()* y *compete_for_active()* cuando el usuario edite con Blockly la acción y disparador del comportamiento. Es por esto que es interesante ver también su diagrama de clase de forma independiente en la figura 23.

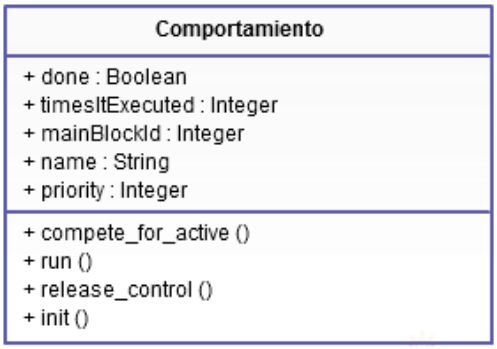


Figura 23: Diagrama de clase de un comportamiento genérico.

5.5. Diagramas de comunicación

En esta sección se muestran diagramas de comunicación de algunas funciones cuya interacción entre componentes es destacable. Como fue mencionado, si bien estas anotaciones UML son orientadas a la programación orientada a objetos, se despliegan los diagramas con la intención de ayudar al lector en la comprensión del sistema.

5.5.1. Crear comportamiento

El diagrama de la figura 24 representa la interacción entre componentes para la creación de un comportamiento. Este es enviado desde el dispositivo cliente, en forma de POST con código Lua en formato de texto. La generación de este código Lua se hace desde código JavaScript que extiende a Blockly. Éste código Lua es parseado y ejecutado, el comportamiento es agregado a al catálogo de Lumen y el administrador de comportamientos *RobotTaskManager* es activado y comienza a ejecutar de acuerdo a la arquitectura basada en prioridades.

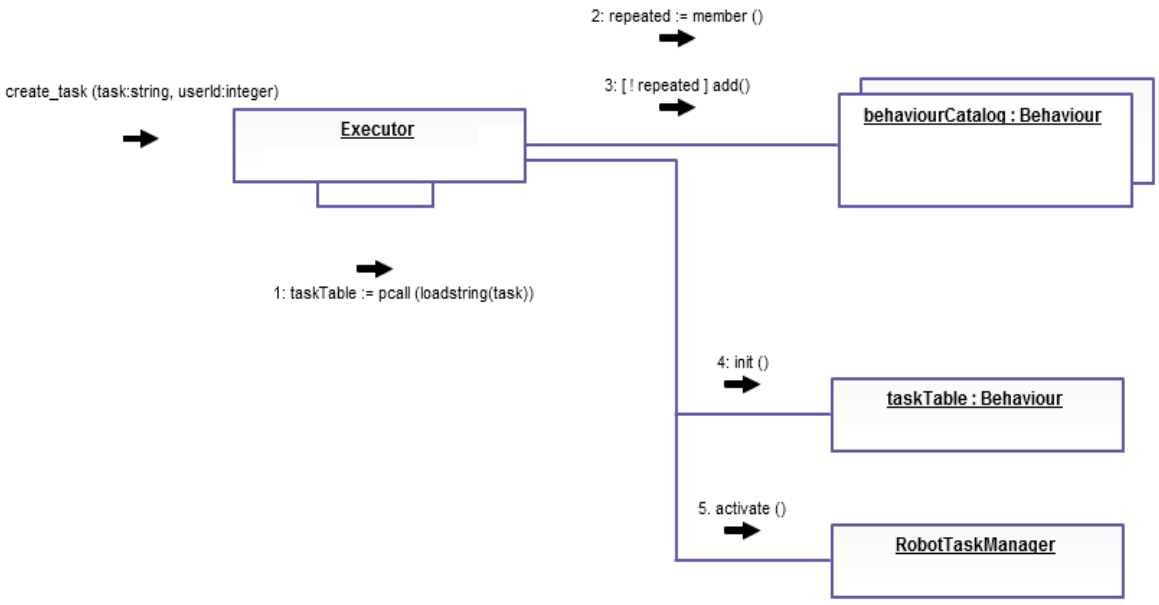


Figura 24: Diagrama de comunicación para ejecutar comportamiento.

5.5.2. Task switching

El diagrama de la figura 25 corresponde al código ejecutado en una iteración del *RobotTaskManager*. Éste una vez activado entra en un loop infinito donde cada vez llama a competir a los comportamientos por el lugar de comportamiento activo. Una vez que estos compiten ejecuta la acción del comportamiento que quedó como activo, deteniendo la ejecución del comportamiento anterior (si es distinto al nuevo activo).

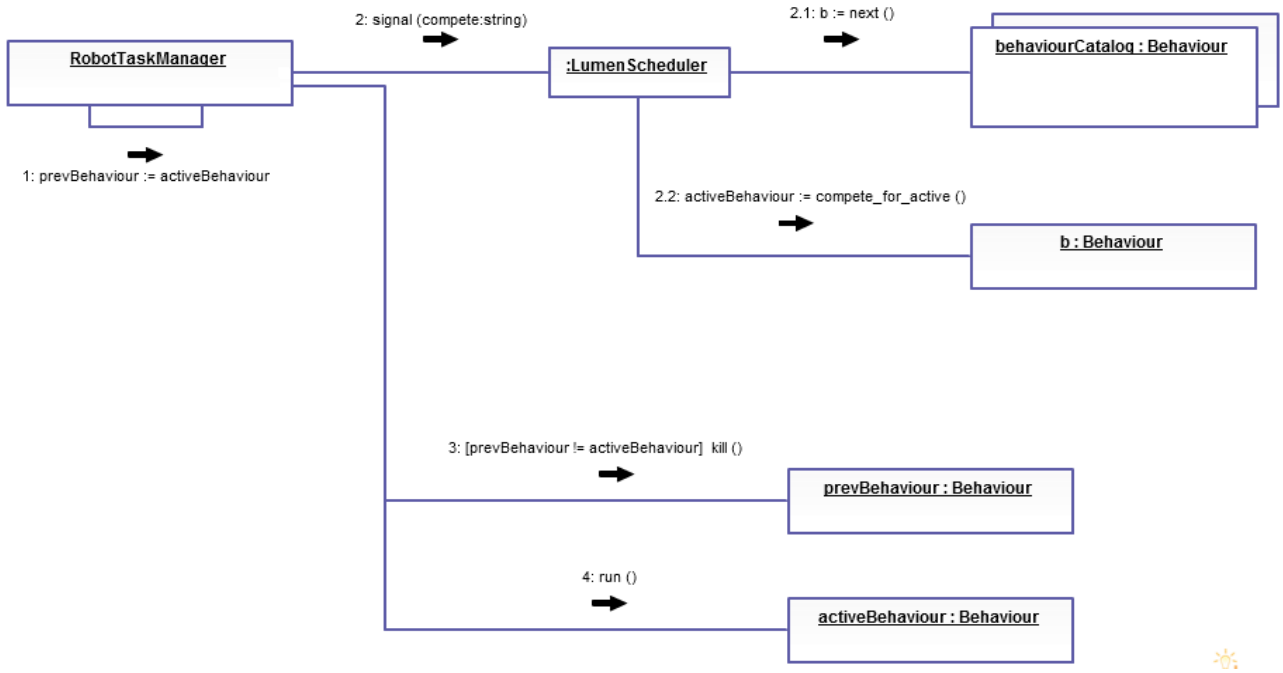


Figura 25: Diagrama de comunicación para el task switching.

5.5.3. Refrescar dispositivos

El diagrama de la figura 26 corresponde al código que se ejecuta cada 5 segundos y que consulta si ha habido cambios en los dispositivos conectados. Para esto se consulta a la interfaz robótica que a su vez consulta a Toribio cuales son los devices conectados. Luego compara estos devices con los últimos registrados y si hubo modificaciones entonces regenera los archivos de definición de bloques de Butiá y de generación de código de estos. Por último notifica retornando verdadero que se debe recargar la página. Esto es necesario para regenerar la toolbox de Blockly y marcar deshabilitados los bloques de dispositivos que ya no existen más.

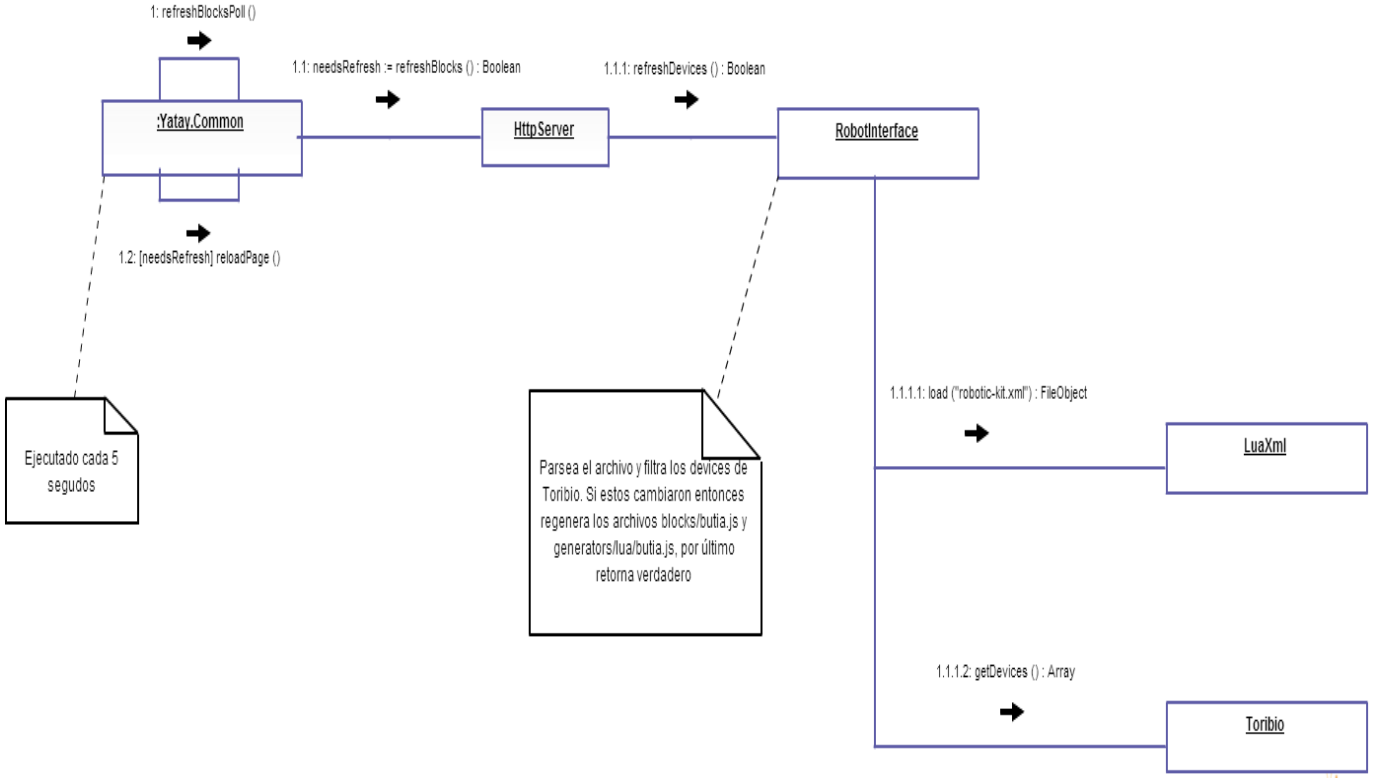


Figura 26: Diagrama de comunicación para refrescar los bloques de dispositivos conectados.

Capítulo 6

Detalles de implementación

Se mencionarán en esta sección algunas características del sistema que aún no habían sido comentadas y que cumplen un rol importante en éste. Detallando cada una de estas características desde el enfoque de la implementación de las mismas.

6.1. Algoritmo principal

En esta sección se expone el algoritmo que implementa la arquitectura robótica basada en prioridades (perteneciente al paradigma reactivo y orientada a comportamientos). Se enumeran primero las principales características de esta arquitectura.

- Está basada en comportamientos.
- Cada comportamiento tiene prioridad y nombre que los identifica.
- En cada momento, si la condición de ejecución (disparador) del comportamiento se cumple, entonces este compite por ejecutarse.
- Dados n comportamientos compitiendo en un instante dado, se ejecuta el de mayor prioridad.
- Si existen varios comportamientos con el mismo número de prioridad y este es máximo, se ejecuta solo uno de ellos, que es el primero que se inicializó en Lumen (el caso ideal es que no haya comportamientos con misma prioridad intentando ejecutarse).

Los comportamientos tienen estructura detallada en la sección *Diagrama de Clases* del capítulo anterior. Donde *Done*, es una variable que permite saber si el comportamiento finalizó su ejecución. *TimesItExecuted* es un número que dice cuántas veces se ha ejecutado éste (es decir, cantidad de veces que ejecutó la función run). *MainBlockId* es el id del bloque de comportamiento en Blockly y sirve para iluminarlo en modo debug. *Name* y *Priority* son el nombre y la prioridad.

El encargado de implementar la administración de los comportamientos es el subsistema *RobotTaskManager*, el cual en cada iteración emite una signal *"compete!"* que provoca que cada uno de los comportamientos registrados ejecute la función *compete_for_active()*. Dadas las características del scheduler de Lumen, no existen problemas de concurrencia. La función *compete_for_active()* revisa si se ha cumplido la condición para que se ejecute éste comportamiento y en caso afirmativo, compara su prioridad con la del comportamiento actualmente activo: si es mayor o si el otro ya finalizó de ejecutar (*done* = true), entonces se establece a sí mismo como el comportamiento activo.

Luego de que los comportamientos compiten y quedó estableció el nuevo comportamiento activo, el administrador *RobotTaskManager* recupera el control de la ejecución. Pregunta si se ha cambiado de activo y en caso afirmativo, termina la ejecución del comportamiento anterior (el que antes de esta iteración estaba activo) y emite un signal con el nombre del nuevo comportamiento activo para que este finalmente se ejecute (función *run*).

Para que los comportamientos puedan escuchar la signals *"compete!"* y la de ejecutar, en la función de inicialización, *init()*, estos registran en Lumen que su función *compete_for_active()* responderá a la signal *"compete!"* y su función *run()* responderá a la signal con su nombre de comportamiento (variable *name*).

6.2. Gestión de proyectos

El concepto de proyectos dentro de un entorno de desarrollo da ventajas a la hora de organizar los trabajos que se realicen en el mismo. En particular puede ser beneficioso para los talleres con la plataforma Butiá en las escuelas o liceos, ya que el estudiante puede crear un nuevo proyecto o unirse a un proyecto existente, para formar varios grupos de estudiantes donde cada uno mantenga sus comportamientos de forma independiente.

Por estas razones el sistema maneja proyectos y ofrece tres formas para almacenar comportamientos: en primer lugar se realiza un guardado automático cada vez que un comportamiento es modificado (se agrega o elimina un bloque). Por otro lado es posible guardar localmente (en el dispositivo móvil) en formato XML. Por último, se realiza una persistencia en el caché del browser para mejorar la usabilidad. Teniendo en cuenta se permite almacenar comportamientos resulta indispensable ofrecer funcionalidades para poder cargar los proyectos.

6.2.1. Base de datos

Se decidió utilizar una base de datos liviana y rápida ya que las tablas e información a persistir poseen pocos datos y no resulta probable que se lleguen a almacenar cantidades considerables de información. Otra característica importante es el lenguaje de comunicación con la interfaz de la base de datos, en este caso Lua. Optando por hacer uso de la biblioteca LuaSQLite3 [11], wrapper para el lenguaje Lua del motor de base de datos SQLite. Esta biblioteca permite crear y manipular una base de datos y es ideal para los fines de este proyecto por ser liviana y simple de usar.

Básicamente, se mantiene una tabla con tres columnas: una para el nombre de proyecto, otra con el nombre de bloque, y el código (XML generado por la biblioteca Blockly a partir de los

bloques). Teniendo como clave primaria el nombre de proyecto y el nombre de bloque. SQLite3 permite definir distintos statements y queries que se aplican sobre la base de datos creada, de forma muy similar a las sentencias conocidas de SQL.

6.2.2. Guardar proyectos

Cada proyecto creado por un grupo de estudiantes será mantenido a nivel de base de datos. La funcionalidad de autosave, aloja cada comportamiento en edición sin la necesidad de que el estudiante deba preocuparse por la posibilidad de perder sus datos. El programa cliente que ejecuta en el navegador web realiza un POST con los datos inmediatamente después de agregar o quitar un bloque. Por detrás, el servidor procesa estos mensajes actualizando la base de datos y respaldando el trabajo de los estudiantes en los proyectos correspondientes.

En cuanto al almacenamiento local, lo ideal tecnológicamente sería generar dinámicamente el archivo con los comportamientos en edición por parte del usuario al momento de querer guardar en su dispositivo el trabajo realizado. Dado que la biblioteca gráfica permite generar un XML a partir de los bloques completamente client-side, se investigó cómo lograr exportar este XML al dispositivo móvil sin necesidad de consumir recursos desde el servidor. Existen varias alternativas para lograr esto [19], se decidió hacer uso de la biblioteca FileSaver.js: trabaja con blobs (abreviación del inglés *binary large object*) e implementa la interfaz *saveAs()* especificada para HTML5 para los navegadores que no la ofrecen de forma nativa. Más adelante, se detallan algunos problemas encontrados con el uso de esta biblioteca y cómo fueron solucionados.

La persistencia en el caché del browser a partir del uso de WebStorage de HTML5 permite que el estudiante no pierda sus comportamientos al refrescar la página o incluso si cierra el browser y lo abre de nuevo, ya que dicha información no caduca. Esta técnica fue detallada en la sección *HTML5* en el capítulo *Resumen del Estado del Arte.*

6.2.3. Abrir proyectos

Se ofrece funcionalidad para cargar o abrir comportamientos alojados en los dispositivos móviles, y también, para los comportamientos alojados en el servidor. Respectivamente, desde la implementación resulta bastante directo cargar un comportamiento local, por la forma en que la biblioteca Blockly maneja los bloques: permite que estos sean importados desde un archivo XML al workspace (área de trabajo) a través de las funciones *textToDom()* y *domToWorkspace()*.

Por otro lado, se le ofrece al usuario la posibilidad de cargar los comportamientos que poseen cada uno de los proyectos almacenados en el servidor. Es utilizada la biblioteca Json4Lua en el

servidor para organizar los datos que el navegador procesa cuando el usuario intenta elegir un comportamiento desde base de datos.

6.3. Diferenciación de dispositivos

6.3.1. Bibliotecas YepNope y Modernizr

En el documento Estado del arte [25] fue mencionado el potencial de estas 2 bibliotecas web. Por un lado la biblioteca Modernizr permite detectar la disponibilidad de ciertas características que se derivan de las especificaciones HTML5 y CSS3. Tradicionalmente, se utilizaba la propiedad UserAgent para detectar las propiedades del navegador (UA sniffing), pero Modernizr realiza detección de características para discriminar que puede o no hacer el navegador y que dispositivo renderiza la página. En particular posee una opción muy interesante de consulta por JavaScript que retorna en base a CSS Media Queries la cantidad de pixeles de alto y ancho en que será renderizada la página web [22]. Por otro lado, la biblioteca YepNope funciona como un cargador condicional permitiendo cargar a demanda los Scripts o CSS requeridos [34]. En Yatay fueron utilizadas estas bibliotecas en conjunto para determinar cuándo se deben cargar las bibliotecas de CSS y JavaScript para Tablets y cuando se deben de cargar para Smartphones. El código es simple de entender y pregunta si el ancho de pantalla es mayor a 480px, en caso afirmativo se considera el dispositivo como Tablet y si no como Smartphones. En la figura 27 se puede observar un ejemplo de lo descrito.

```
Modernizr.load({
    test: Modernizr.mq('only all and (max-device-width:480px)'),
    yep: ["css/mobile.css", "js/mobile.js"],
    nope: ["css/Tablet.css", "js/Tablet.js"]
});
```

Figura 27: Código utilizado para identificar dispositivos.

6.4. Calibración y creación de nuevos sensores

El sistema permite la calibración a través de una funcionalidad donde se despliegan únicamente los sensores y actuadores de la plataforma robótica para ser probados de forma individual. Se puede testear de forma rápida el valor que retorna un sensor determinado o probar los actuadores, sin tener que crear un comportamiento. Dando la posibilidad al estudiante que está programando con el sistema Yatay de identificar cuáles son los valores que necesita alguno de sus comportamientos y reconocer cuáles sensores o actuadores están funcionando de forma correcta y

cuáles no. El sistema se comporta de forma similar con respecto a ejecutar y detener comportamientos en cuanto al envío de tareas al servidor. En este caso las tareas son más sencillas, conteniendo la función *run()* que ejecuta el bloque del sensor o actuador seleccionado y la función *init()* que pone a ejecutar el test.

También es posible crear nuevos sensores a partir de los sensores del kit robótico. Los nuevos sensores pueden ser operaciones aritméticas sobre sensores del kit o conjunciones de dos o más sensores. El nuevo sensor almacena una expresión que es reevaluada cada vez que se lo invoca, y es sumamente flexible, puede ser tanto un número como una expresión booleana. Por ejemplo, un nuevo sensor podría ser el valor del sensor de grises dividido un coeficiente k. Otro sensor podría ser una condición booleana que sea verdadera si los sensores de gris derecho e izquierdo están por encima de un determinado valor. Esta funcionalidad fue implementada usando la biblioteca Blockly, creando un bloque especial que ofrece la posibilidad de realizar las acciones comentadas. La creación de sensores se realiza mediante un bloque llamada "crear sensor" y luego se lo referencia con el bloque "sensor".

6.5. Edición de código generado

Una funcionalidad muy interesante que ofrece el sistema Yatay es la posibilidad de ver y editar el código que fue generado a partir de los bloques de comportamiento. De esta forma, los estudiantes pueden dar un paso más en el intento de comprender la programación que están realizando y meterse en niveles de más bajo nivel que los bloques gráficos. Esta funcionalidad utiliza una biblioteca llamada CodeMirror para decorar sintaxis (como lo hace un IDE) y mostrar de forma prolija el código Lua generado por los bloques que introdujo el usuario. El estudiante puede modificar este código y probarlo en el robot así como exportarlo en un archivo. Dado que Lua es un lenguaje muy sencillo y comprensible de leer, esta funcionalidad permite que los estudiantes den sus primeros pasos en la programación observando el código que se generó para el programa que ellos hicieron, motivándolos a hacer modificaciones simples o complejas y permitiéndoles observar el resultado.

Una limitación inevitable, es que el código editado no se corresponde directamente con los bloques del LPV. Incluso se podría agregar código que no pertenezca al conjunto de bloques de la paleta. Por éstas razones al probar el código editado no se ofrece la funcionalidad de debug. Además los cambios realizados sobre el código son descartados al concluir la edición, aunque se le da la posibilidad al usuario de exportar el código.

Capítulo 7

Configuración

A modo de resumen, el sistema se basa en 3 nodos. El navegador web en el dispositivo móvil ejecuta bibliotecas JavaScript y despliega la interfaz al usuario, mientras que el servidor (en la Raspberry Pi) atiende los pedidos HTTP y ejecuta la lógica del sistema comunicándose con el tercer nodo: la placa USB4Butiá. En esta sección se detallan todas las configuraciones necesarias para lograr la correcta ejecución del sistema en cada uno de los nodos.

7.1. Servidor

En la figura 28 se muestra un diagrama con la estructura del sistema representada por el árbol de directorios (resumido), para facilitar la ubicación de los distintos archivos mencionados a lo largo del documento.

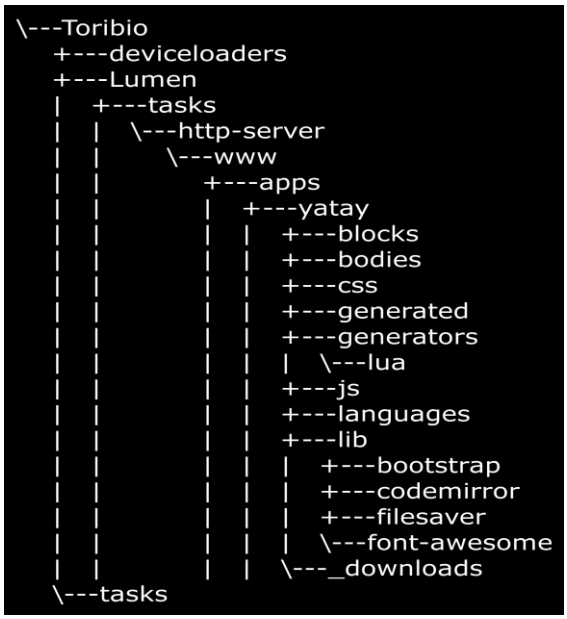


Figura 28: Árbol de directorios del framework de Yatay.

En el directorio Toribio se encuentra el archivo toribio-go.lua que permite inicializar Toribio y de esta manera ejecutar las distintas tasks del sistema Yatay, entre las que se encuentra el servidor web. En ese mismo directorio se encuentra un archivo de configuración, llamado yatay.conf, que permite a Toribio reconocer que tareas deben ser ejecutadas. En este archivo se pueden identificar

las tareas correspondientes al sistema Yatay, ubicadas en /Toribio/tasks/ que son cargadas de la siguiente forma:

```
tasks.Httpserver.load = true
tasks.Executor.load = true
tasks.Persistence.load =true
tasks.ProjectAdmin.load = true
tasks.RobotTasksManager.load = true
tasks.RobotInterface.load = true
```

Además, es destacable mencionar que para usar la biblioteca bobot que permite acceder a los módulos de la placa USB4Butiá, es necesario cargar la misma en este archivo (yatay.conf) de la siguiente forma:

```
deviceloaders.bobot.load = true
deviceloaders.bobot.path = '/home/pi/Framework/bobot'
deviceloaders.bobot.comms = {"usb"}
deviceloaders.bobot.timeout_refresh = 10 --negative or nil disables
```

Siendo .path el camino al directorio donde se encuentra bobot y .comms el servicio comm a ser usado (a modo de prueba, se puede usar “chotox” en caso de no tener una placa USB4Butiá).

Entonces para correr Yatay, desde el directorio Toribio se debe ejecutar desde la consola el siguiente comando:

sudo lua toribio-go.lua -c yatay.conf -d NONE

Donde la bandera -c permite especificar un archivo de configuración para la ejecución de Toribio y la bandera -d determina las impresiones del modo debug (usando NONE no se imprimen mensajes). Al ejecutar el comando, para comprobar una correcta ejecución se deben observar las siguientes impresiones en consola:

```
YATAY: RobotInterface is up...
YATAY: refreshing!
YATAY: DataBase is up...
YATAY: RobotTaskManager is up...
YATAY: Server is up...
```

7.2. Kit Robótico

En el directorio Toribio también se encuentra el archivo de configuración para el kit robótico, llamado "robotic-kit.xml", que posee la estructura detallada a continuación. Para describir el archivo XML se enumeraran los elementos (-) y sus atributos (+):

- **devices**: Contiene los dispositivos (sensores y actuadores) del kit.
  + **from**: Origen de los dispositivos a considerar por el sistema, puede ser "bobot" o "xml".
  + **except**: Dispositivos que no serán considerados por el sistema.

- **device**: Dispositivo del kit, permite crear alias para las funciones del mismo. Se coloca dentro del tag "devices".
  + **device_type**: Tipo del dispositivo, puede ser "sensor", "actuator" o "generic" (wildcard para deshabilitar funciones de todos los devices).
  + **name**: Nombre del dispositivo.

- **function**: Función de un dispositivo en particular, se coloca dentro del tag "device".
  + **name:** Nombre de la función.
  + **alias:** Alias que se le asigna a la función (nombre del bloque en la GUI).
  + **params:** Cantidad de parámetros que recibe la función.
  + **ret:** cantidad de resultados (retornos) de la función.
  + **tooltip:** Mensaje de ayuda desplegado en la GUI (opcional).
  + **values:** Valores por defecto para invocar a la función (opcional, si es usado la función debe necesariamente tener params > 0).
  + **disabled:** Permite deshabilitar una función, valores posibles: "yes" o "no" (opcional, en caso de no usarlo se considera con el valor "no").

Un ejemplo acotado de cómo usar este archivo de configuración sería así:

```
<devices from="xml" except="bb-ax, bb-admin, bb-hackp">
  <device device_type="sensor" name="bb-distanc:1">
      <function name="getValue" alias="medir distancia" params="0" ret="1" />
  </device>
  <device device_type="actuator" name="bb-motors">
      <function name="setvel2mtr" alias="adelante" params="4" ret="0" values="1,500,1,500" />
  </device>
  <device device_type="generic">
      <function name="getVersion" disabled="yes" />
  </device>
</devices>
```

7.3. Navegador Web

Los usuarios para conectarse a la aplicación, con el navegador web desde sus dispositivos móviles, deben conocer la dirección IP que posee la placa Raspberry Pi cuando es inicializado el servidor. Entonces, para ingresar al sistema Yatay se debe acceder a la siguiente dirección:

IP:8080/apps/yatay/

En la figura 29 se muestra la pantalla inicial, en donde la dirección IP corresponde a 192.168.1.9.

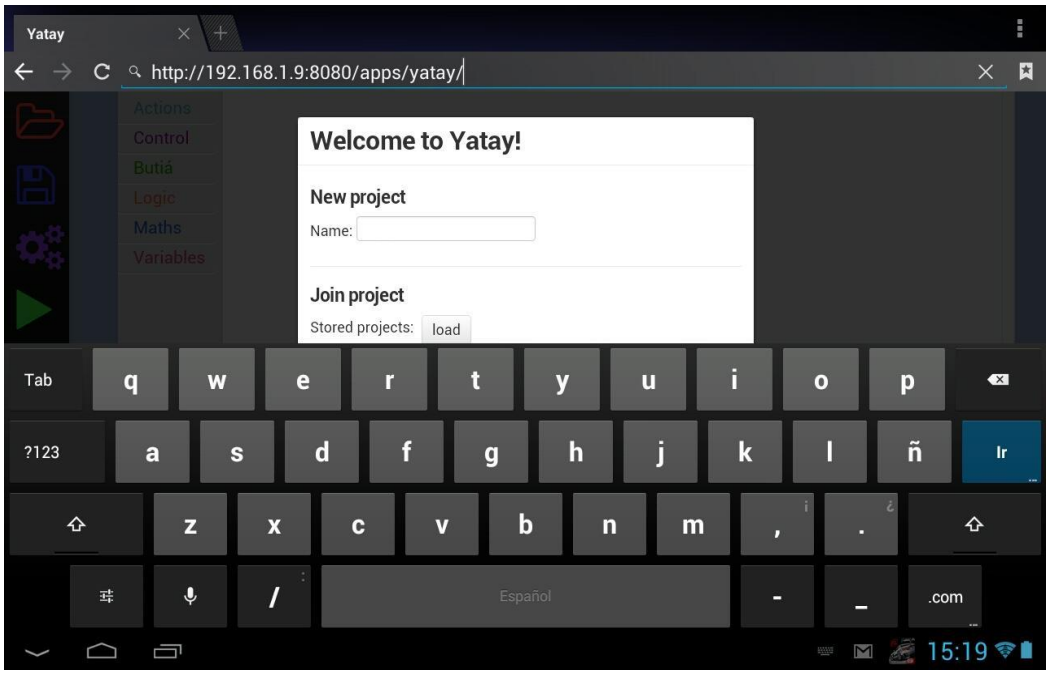


Figura 29: Pantalla inicial de Yatay en Tablet de 10’’.

Capítulo 8

Verificación del sistema

En este capítulo se detallan las pruebas realizadas para verificar y validar el funcionamiento del sistema, además se identifican los errores conocidos y las limitaciones tecnológicas encontradas.

8.1. Casos de prueba

Se realizaron casos de pruebas para las funcionalidades principales del sistema especificando la entrada y la salida esperada así como cuales funcionalidades se están testeando en el caso. Se diseñan además pruebas para verificar el cumplimiento de requerimientos no funcionales. Los casos de prueba deben ser probados en cada uno de los dispositivos a los cuales apunta esta aplicación, estos son: Tablets de 7" y 10", celulares con resolución mayor a 320 x 480 y computadoras de escritorio [28].

N°	Funcionalidad	Entrada	Salida esperada
1.1	Ingreso (1era vez)	El usuario ingresa por primera vez a la aplicación.	Se despliega un mensaje de bienvenida solicitando al usuario identificar el proyecto de trabajo. Dicho mensaje no puede saltarse sin identificar el proyecto.
1.2	Ingreso (1era vez)	El usuario ingresa un nombre de proyecto y presiona "Comenzar".	El mensaje de bienvenida desaparece y el tutorial comienza.
1.3	Ingreso (1era vez)	1) El usuario presiona cargar. 2) Selecciona un proyecto y presiona "Comenzar".	1) El sistema disponibiliza los proyectos existentes en la base de datos. 2) El mensaje de bienvenida desaparece y el tutorial comienza.
2	Ingreso (No 1era vez)	El usuario ingresa al sistema luego de haber ingresado previamente a este.	El sistema mantiene el identificador de usuario y su nombre de proyecto. Si había comportamientos en edición la última vez que ingresó, estos se disponibilizan al usuario, pero solo estos y ninguno más. La paleta de Butiá muestra los sensores y actuadores disponibles actualmente. Y deshabilitados los que están configurados en el archivo xml pero no están conectados.
3	Visibilidad (Tablet)	Ingreso al sistema desde Tablets de 10" y de 7".	La interfaz gráfica puede visualizarse de forma correcta, siendo ésta similar a la especificada para Tablets en el documento ***"Manual de Usuario"*** [26].

3.1	Visibilidad (Celular)	Ingreso al sistema desde Celulares con pantalla igual o mayor a 320x480px.	La interfaz gráfica puede visualizarse de forma correcta, siendo ésta similar a la especificada para Celulares en el documento ***"Manual de Usuario"*** [26].
4	Drag and drop	1) Se ingresan mediante drag and drop dos bloques conectables. 2) Se realiza drag and drop para conectarlos entre sí. 3) Se realiza drag and drop para moverlos como conjunto.	1) El drag and drop es fluido y manejable. 2) Los bloques se conectan entre sí. 3) Los bloques se mueven de forma fluida en conjunto.
5.1	Creación de comportamientos	1) Se crea un comportamiento de nombre "Adelante" que no tenga disparador, con prioridad 1 y con un bloque de mover Butiá hacia adelante conectado. 2) Se lo marca como listo.	1) El comportamiento es creado correctamente. 2) Se enlista como comportamiento listo vaciando el área de trabajo o pizarra.
5.2	Creación de comportamientos	1) Se crea un comportamiento de nombre "GrisIzq" con prioridad 2. Como disparador tiene una condición sensor gris 1 mayor a cierto valor identificado con el color negro en el ambiente de prueba. Como acción debe tener un bloque de girar el Butiá a la derecha. 2) Se lo marca como listo.	1) El comportamiento es creado correctamente. 2) Se enlista como comportamiento listo vaciando el área de trabajo o pizarra.
5.3	Creación de comportamientos	1) Se crea un comportamiento de nombre "GrisDer" con prioridad 2. Como disparador tiene una condición sensor gris 2 mayor a cierto valor identificado con el color negro en el ambiente de prueba. Como acción debe tener un bloque de girar el Butiá a la izquierda. 2) Se lo marca como listo.	1) El comportamiento es creado correctamente. 2) Se enlista como comportamiento listo vaciando el área de trabajo o pizarra.
5.4	Creación de comportamientos	1) Se crea un comportamiento de nombre "Detener" con prioridad 3. Como disparador tiene una condición sensor gris 1 y sensor gris 2 mayor a cierto	1) El comportamiento es creado correctamente. 2) Se enlista como comportamiento listo vaciando el área de trabajo o pizarra.

		valor identificado con el color negro en el ambiente de prueba. Como acción debe tener un bloque de detener el Butiá. 2) Se lo marca como listo.	
6	Comportamientos listos	Se seleccionan al azar comportamientos marcados como listos.	El sistema despliega los bloques correspondientes al comportamiento seleccionado marcando previamente como listo, si existía, el comportamiento en edición en la pizarra o área de trabajo.
7.1	Eliminar	1) Se presiona el botón eliminar. 2) Se selecciona “Solo pizarra”.	1) El sistema despliega un modal para seleccionar “Solo pizarra” o “todo”. 2) Al presionar “Solo pizarra” se eliminan todos los bloques que están en el área de trabajo. Los comportamientos listos no se eliminan.
7.2	Eliminar	1) Se presiona el botón eliminar. 2) Se selecciona “todo”.	1) El sistema despliega un modal para seleccionar “Solo pizarra” o “todo”. 2) Al presionar “todo” se eliminan todos los bloques que están en el área de trabajo y los marcados como listos.
8	AutoSave y Cargar	1) Se selecciona el botón de cargar (ver manual de usuario). 2) Se presiona el boton obtener. 3) Se seleccionan todos los comportamientos del proyecto actual. (Si se hacen en orden las pruebas, son los eliminados en la prueba 8) y se presiona “abrir”.	1) El sistema despliega el modal de cargar comportamientos. 2) El sistema despliega para cada nombre de proyecto en la base de datos, todos los comportamientos de estos. (Si se hacen en orden las pruebas, son los eliminados en la prueba 8) 3) Se recuperan y marcan como listos los comportamientos seleccionados.
9	Guardar Archivo	Se selecciona el boton de guardar (ver manual de usuario).	El sistema brinda un diálogo de descarga de un archivo con los bloques con los que trabajaba el usuario.
10	Cargar desde Archivo	1) Se selecciona el boton de cargar (ver manual de usuario). 2) Se presiona el boton “Seleccionar Archivo”. 3) Se presiona “abrir”.	1) El sistema despliega el modal de cargar comportamientos. 2) El sistema muestra un diálogo de selección de archivo en el file system del dispositivo. 3) El sistema recupera y marca como listos los comportamientos existentes en el archivo.
11	Ejecución	1) Teniendo creados los comportamientos del caso de prueba 6, se selecciona el boton de ejecución (ver manual de usuario).	1) Desaparecen todos los botones quedando disponibles solo los botones de “Ejecución sin debug”, “Debug” y “Detener”. 2) Se ejecutan los comportamientos en el robot, siguiendo una ejecución acorde a la arquitectura de prioridades. Se muestran los resultados de los

		2) Se selecciona el boton de ejecución sin debug (ver manual de usuario).	sensores que se están midiendo. Los bloques quedan inhabilitados para editar.
12	Debug	1) Teniendo creados los comportamientos del caso de prueba 6, se selecciona el boton de ejecución (ver manual de usuario). 2) Se selecciona el boton de Debug (ver manual de usuario).	1) Desaparecen todos los botones quedando disponibles solo los botones de "Ejecución sin debug", "Debug" y "Detener". 2) Se ejecutan los comportamientos en el robot, siguiendo una ejecución acorde a la arquitectura de prioridades. Se muestran los resultados de los sensores que se están midiendo. Se muestra en el área de trabajo el comportamiento que se está ejecutando, se ilumina el bloque que está actualmente en ejecución. Los bloques quedan inhabilitados para editar.
13	Detener y reiterar ejecución.	1) Mientras se está en el caso de prueba 12 o 13 se presiona detener. 2) Se comienza otra vez la ejecución del caso de prueba 12 o 13.	1) Se detienen los actuadores del robot y se restauran los botones iniciales. Se oculta el área de resultados. 2) Se producen los mismos resultados esperados de la sección 12 o 13.
14.1	Ver Código	Se presiona el boton de ver código (ver manual de usuario).	Se abre un modal con pestañas por cada comportamiento, en donde para cada pestaña se muestra el código Lua de dicho comportamiento.
14.2	Ver y Editar Código	1) Se edita el código de alguno de los comportamientos y se selecciona "Probar". 2) Se detiene la ejecución.	1) Se ejecuta el código editado. 2) Se detiene la ejecución y se vuelve a mostrar el modal de ver código.
14.3	Guardar código editado	Dentro del modal de ver comportamiento se presiona "Guardar".	El sistema brinda un diálogo de descarga de un archivo con el código lua editado.
15	Test Butiá	1) Se presiona el boton de calibrar (ver manual de usuario). 2) Se selecciona un bloque de actuador o sensor del Butiá y se presiona "Ejecutar". 3) Se presiona detener.	1) El sistema disponibiliza solo los botones de ejecución, detener y borrar. Se disponibiliza solo los bloques de "Butiá". 2) Se ejecuta el bloque de actuador o sensor y se disponibiliza en tiempo real la información del sensor seleccionado si corresponde. 3) Se restauran los botones iniciales y los demás bloques. Si el usuario tenía comportamientos guardados estos aparecen disponibles como comportamientos listos.
16	Funcionamiento de bloques	Se prueba la ejecución de cada uno de los bloques existentes, dentro de un contexto de bloques que permita dicha prueba.	Al ejecutarse, el bloque presenta el comportamiento descrito para dicho bloque.

17	Tiempo de ejecución.	Se ejecutan el caso de prueba 12.	Para cualquiera de los comportamientos, el sistema demora menos de un segundo en detectar la condición que lo dispara.
18	Tiempo de reacción del sistema.	Cualquier acción de drag and drop o presionado de botones.	El sistema no demora más de un segundo y medio en responder a la acción.
19	Test Butiá con múltiples usuarios	Repetir el caso de prueba 16 pero con 2 más usuarios al mismo tiempo.	El sistema devuelve a cada usuario el valor de los sensores correspondiente a lo que ejecutó dicho usuario. Sobre los actuadores, la ejecución de acciones distintas de varios usuarios al mismo tiempo no garantiza la ejecución deseada por dicho usuario por ejecutarse estos concurrentemente.
20	Ejecución con múltiples usuarios	Repetir el caso de prueba 12, pero siendo cada comportamiento del caso de prueba 6, creado por un usuario distinto.	Los comportamientos creados por los distintos usuarios se ejecutan de la misma forma esperada en el caso de prueba 12 para un único usuario.
21	Debug con múltiples usuarios	Repetir el caso de prueba 13, pero siendo cada comportamiento del caso de prueba 6, creado por un usuario distinto.	Los comportamientos creados por los distintos usuarios se ejecutan de la misma forma esperada en el caso de prueba 12 para un único usuario. Solo se iluminan los bloques del comportamiento de cada usuario cuando dicho comportamiento se disparó y está en ejecución. El resto de los casos igual mostrará los resultados de sensores y consola pero no iluminando bloques de comportamientos (pues no se están ejecutando).
22	Multilenguaje	Se presiona el boton de multilenguaje y se cambia el lenguaje.	Todos los textos desplegados por la aplicación se encuentran en el lenguaje seleccionado.
23	Comportamientos largos	Se crea un comportamiento cuyo largo y ancho superen ampliamente el tamaño predeterminado del área de trabajo.	El sistema se autoajusta al tamaño del comportamiento y permite a través de los scrollbars la correcta visualización del comportamiento.
24	Estabilidad mantenida en el tiempo	Se utiliza la aplicación durante un tiempo aproximado de una hora.	El sistema continúa respondiendo bien a los casos de prueba.
25	Bloques de sensores que no están conectados	Se cargan o se tiene en edición comportamientos que poseen sensores que no se encuentran más conectados en el robot.	Se muestran dichos bloques deshabilitados y su generación de código no genera ningún código.

26	Alteración de sensores conectados al robot Butiá	Se desconectan o conectan sensores al robot Butiá,	Si no se está en ejecución, entonces en un periodo de tiempo de hasta 10 segundos, el sistema refresca los bloques de la paleta de Butiá mostrando los sensores disponibles actualmente.
27	XML Devices	1. No existe el archivo. 2. Se configura un archivo con el atributo “from” en “bobot”, y se definen devices para los sensores. 2.1. Se agrega el atributo “except” con devices conocidos que no son usados. 2.2. Se agregan devices con alias para los sensores y actuadores (probando el atributo “values” para valores por defecto). 2.3. Se define un device genérico para excluir funciones. 3. Se configura un archivo con el atributo “from” en “xml”, y se definen devices para los sensores. 3.1-3. Igual a 2.1-3.	1. Se crean los bloques parseando bobot y se muestran con los nombres (hardcode para Butiá) y puertos. 2.1. Se muestran los bloques de la categoría Butiá con el nombre y puerto, no se muestran los devices descartados. 2.2. Se actualizan los nombres para los aliases agregados, y se despliegan nuevos bloques para las funciones por defecto de los actuadores. 2.3. Se eliminan las funciones que fueron deshabilitadas para el device genérico. 3. Se debe cumplir lo mencionado en el punto 2, considerando únicamente los datos del XML.

8.2. Errores destacables corregidos

Se enlistan aquí algunos de los errores más destacables corregidos y su gravedad así como versión del sistema en que se introdujo el fix.

Error	Descripción	Prioridad	Versión reparado
Modal de inicio no aparece.	El modal de inicio no aparece la primera vez que se ingresa aunque si se refresca la página si lo hace.	Media	6abc274
Hay que presionar de forma prolongada sobre el toolbox para que se abra.	Hay que presionar durante un breve tiempo para que Blockly reconozca el evento de touch. Sucede en el toolbox y al seleccionar bloques.	Baja	1062c85

Los motores no responden.	Los motores funcionan desde Bobot usando telnet, pero no desde Toribio y por tanto no funcionan desde Yatay.	Alta	4989116
Las scrollbars de Blockly no se muestran.	Las barras de navegación que permiten moverse a través del workspace no se muestran debido a un conflicto con bootstrap.	Baja	d22349c
Algunas secuencias de acciones llevan a que los bloques pierdan sus conexiones y se desarmen.	Al ejecutar en debug o marcar como listos los comportamientos y empezar a aleatoriamente mover los bloques estos se desarman y pierden sus conexiones.	Media	060c7c2
El guardado de bloques no funciona en Android Browser.	En Android browser en vez de descargarse los bloques como archivo xml, se abren en una nueva pestaña.	Media	836a876
Los bloques por fuera de los bloques principales de comportamiento.	Los bloques que no son de comportamiento y no quedan unidos a un bloque de comportamiento provocan errores en la ejecución, debugging y en otras funcionalidades del sistema.	Media	4f1ddc7
Las variables y los sensores no iniciados provocan un error.	Las variables y los sensores aparecen en Blockly usando la misma biblioteca y por tanto aparecen indistintamente en los combo-box, el uso de una variable/sensor no iniciado provoca un error.	Baja	d22349c
Conflicto entre Blockly y Bootstrap.	El conflicto no permite presionar ningún boton del menú al ejecutar en la aplicación en los navegadores Chrome o Safari.	Alta	150b5c6712
Bloques eliminados al actualizar kit robótico.	Cuando se actualiza el kit robótico, los bloques de sensores que son eliminados (ya sea porque se cambiaron de puerto o se quitaron) generan inconsistencias en los comportamientos que usaban estos bloques.	Media	f5c143a5b7
Generador de código para el bloque if.	Al usar el mutator para crear bloques con sucesivas cláusulas “else if” dentro de un bloque if el código generado tiene errores.	Medio	1f9d6edb
Scroll de diálogos.	Al desplegar un diálogo no es posible realizar scroll para visualizar, por ejemplo el código de los comportamientos, o los comportamientos a ser cargados desde el servidor.	Medio	2e3e36d898

8.3. Limitaciones tecnológicas

Ciertas herramientas tecnológicas fueron descartadas con la intención de lograr una aplicación capaz de adaptarse a la mayor cantidad de dispositivos móviles y navegadores. Algunas de estas serán mencionadas a continuación, dando detalles sobre su incompatibilidad para lograr lo mencionado anteriormente y destacando las ventajas que podrían obtenerse con su uso.

8.3.1. Websockets

Una tecnología muy interesante a los objetivos de nuestro proyecto, es la de Websockets. Esta permite abrir una única conexión entre cliente y servidor para transferir información de tiempo real, como ser resultados de sensores o retroalimentación del código en ejecución (modo debug). Ésta es mucho más eficiente que utilizar pollings de posts para obtener información del servidor y descongestiona al servidor permitiéndole consumir menos recursos y ejecutar de forma más veloz.

Para determinar compatibilidades realizamos una sencilla prueba de echo mediante websockets, la página websocket.org (http://www.websocket.org/echo.html) ofrece este sencillo servicio de Echo Test para determinar si el browser soporta o no el uso de web sockets.

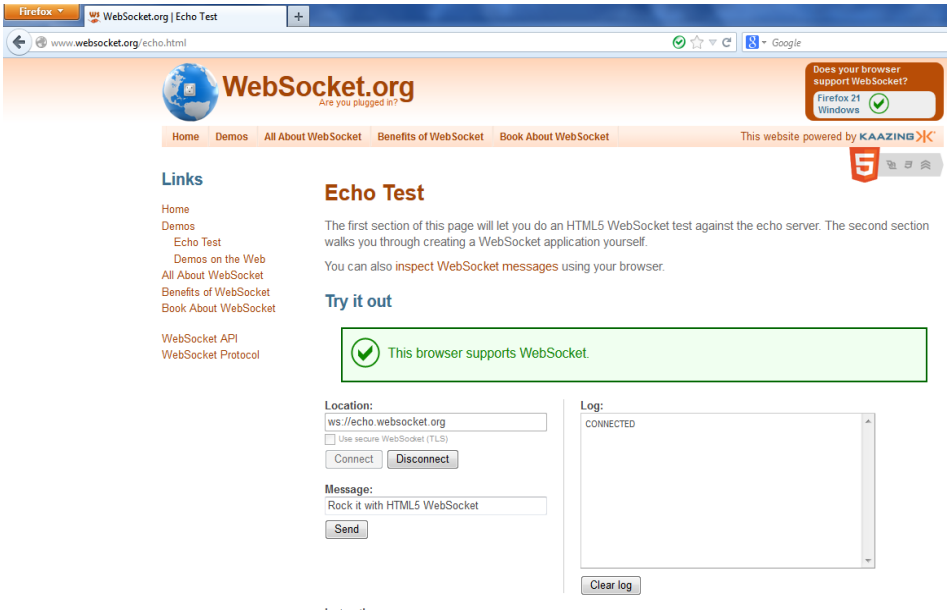


Figura 30: Echo Test con websocket.org para Firefox en un ordenador.

Los resultados en un celular con S.O Android 2.3 (Gingerbread) y una Tablet con Android 4.1 (Jelly Bean) fueron que el navegador por defecto no soporto la utilización de websockets (confirmando una tabla de compatibilidad que presenta Wikipedia [43]). Por lo que, a pesar de ser

un feature con ventajas para la comunicación, su incompatibilidad con el navegador nativo de Android hace que sea descartado.

8.3.2. Features HTML5

Como fue mencionado, HTML5 es una tecnología emergente que brinda facilidades para desarrollar aplicaciones que se adapten tanto a dispositivos móviles como ordenadores. Dentro de las novedades de HTML5 se encuentra el atributo download, el cual permite especificar un hyperlink como un recurso o archivo a ser descargado. El valor que se le atribuye a este atributo dará nombre al archivo. Entonces, para cumplir el requerimiento funcional que permite descargar localmente (en los dispositivos móviles) los comportamientos creados por el usuario, existe la posibilidad de generar client-side el archivo y combinando el feature mencionado con Data URI Scheme (forma de exponer datos desde páginas web como si fuesen recursos externos) o alguna biblioteca en javascript como FileSaver [12] se puede satisfacer el requerimiento sin tener que descargar un archivo alojado en el servidor.

Surgieron varios inconvenientes a la hora de aplicar este enfoque, principalmente por limitaciones del navegador nativo de Android que no soporta el atributo download (con la excepción de la última versión) [5]. Esto impacta en la usabilidad del requerimiento, dado que en vez de descargar el archivo localmente el navegador abre una nueva ventana con el contenido del mismo (ya que sabe interpretar el formato del archivo, en este caso XML). Lo que derivó en la implementación del caso de uso agregando soporte para Android Browser tomando en cuenta el User Agent consumiendo el archivo desde el servidor. Una tabla de compatibilidad puede verse en la figura 31.

Download attribute - **Working Draft**

*Usage stats: Global
Support: 51.93%

When used on an anchor, this attribute signifies that the resource it points to should be downloaded by the browser rather than navigate to it.

Show all versions	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	IE Mobile
								2.1		
								2.2		
						3.2		2.3		
						4.0-4.1		3.0		
	8.0					4.2-4.3		4.0		
	9.0		31.0			5.0-5.1		4.1		
	10.0	26.0	32.0			6.0-6.1		4.2-4.3	7.0	
Current	11.0	27.0	33.0	7.0	19.0	7.0	5.0-7.0	4.4	10.0	10.0
Near future		28.0	34.0		20.0					
Farther future		29.0	35.0		21.0					
3 versions ahead		30.0	36.0							

Figura 31: Tabla representativa del soporte del atributo download por browsers [5].

8.3.3. SVG

Scalable vector graphics es una notación XML para programar gráficos vectoriales bidimensionales tanto estáticos como animados. Ha tenido un gran nivel de aceptación ya que es una forma sencilla de superar limitaciones gráficas CSS o de facilitar implementaciones que de otra forma con CSS y JavaScript serían extremadamente complicadas. Se convirtió en una recomendación del W3C en septiembre de 2001 y hoy en día la mayoría de los browsers soportan SVG. La biblioteca Blockly que hemos descrito en otras secciones de este documento y que se utilizó para desarrollar el sistema de este proyecto de grado, lo utiliza para renderizar los gráficos de los bloques y sus interacciones. SVG sin embargo no es soportado por todas las versiones de Android Browser, las versiones anteriores a la 3.0 no lo soportan [6]. Aun así, dado que para los dispositivos Android viejos existe la posibilidad de instalar browsers modernos que los soportan, no consideramos esta limitante como un hecho que nos hiciera descartar la tecnología, sobre todo porque las Tablets suelen poseer de fábrica versiones nunca menores a la 4.0. Una tabla de compatibilidad puede verse en la figura 32.

SVG (basic support) - Recommendation

Method of displaying basic Vector Graphics features using the embed or object elements

Resources: SVG Web: Flash-based polyfill, Wikipedia, Web-based SVG editor, SVG showcase site, A List Apart article, has.js test

Global user stats*:

Support:	85.8%
Partial support:	0.02%
Total:	85.82%

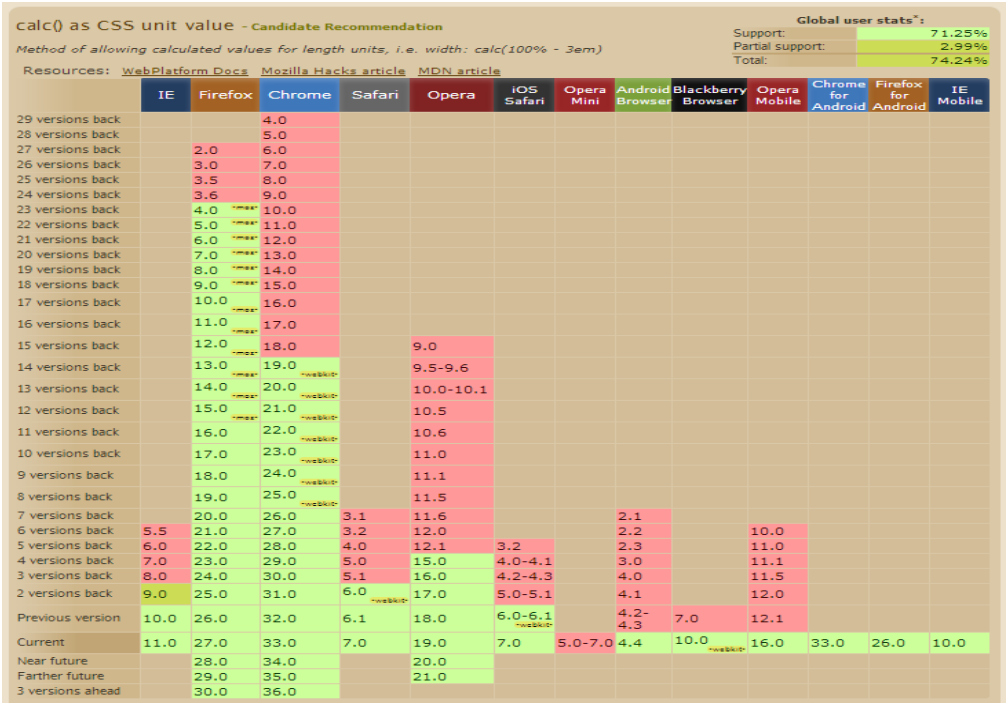
	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android	IE Mobile
29 versions back			4.0										
28 versions back			5.0										
27 versions back		2.0	6.0										
26 versions back		3.0	7.0										
25 versions back		3.5	8.0										
24 versions back		3.6	9.0										
23 versions back		4.0	10.0										
22 versions back		5.0	11.0										
21 versions back		6.0	12.0										
20 versions back		7.0	13.0										
19 versions back		8.0	14.0										
18 versions back		9.0	15.0										
17 versions back		10.0	16.0										
16 versions back		11.0	17.0										
15 versions back		12.0	18.0		9.0								
14 versions back		13.0	19.0		9.5-9.6								
13 versions back		14.0	20.0		10.0-10.1								
12 versions back		15.0	21.0		10.5								
11 versions back		16.0	22.0		10.6								
10 versions back		17.0	23.0		11.0								
9 versions back		18.0	24.0		11.1								
8 versions back		19.0	25.0		11.5								
7 versions back		20.0	26.0	3.1	11.6			2.1					
6 versions back	5.5	21.0	27.0	3.2	12.0			2.2		10.0			
5 versions back	6.0	22.0	28.0	4.0	12.1	3.2		2.3		11.0			
4 versions back	7.0	23.0	29.0	5.0	15.0	4.0-4.1		3.0		11.1			
3 versions back	8.0	24.0	30.0	5.1	16.0	4.2-4.3		4.0		11.5			
2 versions back	9.0	25.0	31.0	6.0	17.0	5.0-5.1		4.1		12.0			
Previous version	10.0	26.0	32.0	6.1	18.0	6.0-6.1		4.2-4.3	7.0	12.1			
Current	11.0	27.0	33.0	7.0	19.0	7.0	5.0-7.0	4.4	10.0	16.0	33.0	26.0	10.0
Near future		28.0	34.0		20.0								
Farther future		29.0	35.0		21.0								
3 versions ahead		30.0	36.0										

Figura 32: Tabla representativa del soporte de SVG por browsers [6].

8.3.4. Features CSS3

El último estándar para CSS es conocido como CSS3, dividido en módulos que contiene las especificaciones anteriores y se agregan nuevas características dentro de las cuales se puede destacar selectores, fondos y bordes, efectos para texto, transformaciones en 2D y 3D, animaciones, entre otros. A grandes rasgos, CSS es un lenguaje que permite describir la presentación de una página web. Por lo cual, para lograr un sistema user-friendly y dar soporte al variado espectro de dispositivos móviles se usaron, entre otras cosas, estilos CSS3 que permitan adaptar la interfaz de usuario dependiendo de las características de los dispositivos.

Al implementar la interfaz para los dispositivos celulares, la cual desliza la paleta de bloques (toolbox) hacia un borde de la pantalla para que el área de trabajo (workspace) permita visualizar los bloques. Fue identificada una limitación del navegador nativo de Android con la función *calc()* que provee el lenguaje mencionado, esta función permite por ejemplo calcular el ancho de un elemento de la siguiente forma "*width: calc(100% - 50px)"* lo que resulta cómodo y sencillo para adaptarse a cualquier tamaño de display. La alternativa fue generar dinámicamente parte del archivo CSS al cargar la página web, usando JQuery para averiguar las medidas del dispositivo que intenta ejecutar el sistema [4]. Una tabla de compatibilidad puede verse en la figura 33.



calc() as CSS unit value - Candidate Recommendation

Method of allowing calculated values for length units, i.e. width: calc(100% - 3em)

Resources: WebPlatform Docs, Mozilla Hacks article, MDN article

Global user stats*: Support: 71.25%, Partial support: 2.99%, Total: 74.24%

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android	IE Mobile
29 versions back			4.0										
28 versions back			5.0										
27 versions back		2.0	6.0										
26 versions back		3.0	7.0										
25 versions back		3.5	8.0										
24 versions back		3.6	9.0										
23 versions back		4.0 -moz-	10.0										
22 versions back		5.0 -moz-	11.0										
21 versions back		6.0 -moz-	12.0										
20 versions back		7.0 -moz-	13.0										
19 versions back		8.0 -moz-	14.0										
18 versions back		9.0 -moz-	15.0										
17 versions back		10.0 -moz-	16.0										
16 versions back		11.0 -moz-	17.0										
15 versions back		12.0 -moz-	18.0		9.0								
14 versions back		13.0 -moz-	19.0 -webkit-		9.5-9.6								
13 versions back		14.0 -moz-	20.0 -webkit-		10.0-10.1								
12 versions back		15.0 -moz-	21.0 -webkit-		10.5								
11 versions back		16.0	22.0 -webkit-		10.6								
10 versions back		17.0	23.0 -webkit-		11.0								
9 versions back		18.0	24.0 -webkit-		11.1								
8 versions back		19.0	25.0 -webkit-		11.5								
7 versions back		20.0	26.0	3.1	11.6			2.1					
6 versions back	5.5	21.0	27.0	3.2	12.0			2.2		10.0			
5 versions back	6.0	22.0	28.0	4.0	12.1	3.2		2.3		11.0			
4 versions back	7.0	23.0	29.0	5.0	15.0	4.0-4.1		3.0		11.1			
3 versions back	8.0	24.0	30.0	5.1	16.0	4.2-4.3		4.0		11.5			
2 versions back	9.0	25.0	31.0	6.0 -webkit-	17.0	5.0-5.1		4.1		12.0			
Previous version	10.0	26.0	32.0	6.1	18.0	6.0-6.1 -webkit-		4.2-4.3	7.0	12.1			
Current	11.0	27.0	33.0	7.0	19.0	7.0	5.0-7.0	4.4	10.0 -webkit-	16.0	33.0	26.0	10.0
Near future		28.0	34.0		20.0								
Farther future		29.0	35.0		21.0								
3 versions ahead		30.0	36.0										

Figura 33: Tabla representativa del soporte de función calc() por browsers [4].

8.3.5. Bibliotecas de edición y highlight de código

Para la funcionalidad que permite ver el código generado por los comportamientos creados con Blockly, es útil una biblioteca que nos permita estructurar y remarcar las palabras reservadas para poder visualizarlas como si estuviésemos editando el código desde un IDE. Esto permite una mayor comprensión de este así como permite al programador novato (y también al experto) organizarse y escribir su código con claridad y facilidad. Dado que al usuario al que apunta no tiene experiencia en programación, ni conoce Lua, esta herramienta lo ayuda a entender el código, lo cual es el objetivo de esta funcionalidad.

El problema de estas bibliotecas es que se basan en la manipulación de los eventos de presionado de teclas (keyboard events) para detectar palabras claves del lenguaje al que están remarcando. Para colorear y adornar palabras utilizan código HTML con estilos por CSS y una copia del texto actual escondido en un textarea donde la biblioteca detecta con expresiones regulares las palabras claves. En distintos browsers se manejan los eventos de diferente forma. Sin embargo, en el browser nativo de Android el evento de la tecla “backspace” rompe con el paradigma de los otros browsers, lo cual crea un problema difícil de resolver para estas bibliotecas [9]. Este navegador no envía ningún evento de “key press” del tipo “KEYCODE_DEL”, lo cual dificulta a la biblioteca el poder actualizar el código HTML con estilos frente a la eliminación de un carácter. Realizamos pruebas de las últimas versiones de CodeMirror, Ace, EditArea, CodePress entre otros y, o bien no poseían extensiones para Lua y eran complicados de extender, o no funcionaban bien con el backspace de Android, no permitiendo borrar texto (lo cual era posible si el textarea no estaba influenciado por la biblioteca).

Finalmente la única alternativa encontrada por este equipo fue utilizar una versión muy antigua de CodeMirror en donde se había hecho un workaround para este problema. Aunque el backspace no funciona completamente normal y algunas veces se traba frente a espacios en blanco, lo cual se soluciona haciendo touch para remarcar el cursor en dicha posición, resultó ser la única versión de biblioteca probada que se comporta de manera aceptable. Lo anterior es uno de los “known issues” o errores conocidos del sistema, que se origina en la adaptación solamente parcial de estas bibliotecas a Android Browser.

8.4. Errores conocidos

Luego de que se finalizó el desarrollo y verificación del sistema se reconocen algunos errores conocidos del sistema que surgen de las propias limitaciones de tecnología o de bibliotecas utilizadas.

8.4.1. Exportación de comportamientos en Android Browser

El navegador Android Browser presenta limitaciones para descargar archivos. En primer lugar, como se destacó en la sección de limitaciones tecnológicas, Android Browser no soporta el atributo Download hasta la versión 4.4. Se buscaron varias alternativas, pero no se encontró ninguna que nos permita en dicho navegador generar el archivo client-side y descargarlo, ya que Android Browser frente a formatos que conoce como ser xml, txt, entre otros los abre en el propio navegador en vez de descargarlos.

Frente a esta limitación tecnológica, el equipo decidió utilizar generación client-side del archivo para todos los navegadores excepto para Android Browser. Para este último, la generación del archivo a descargar es server-side. Aun así (creando el archivo en el servidor), el navegador nativo de Android solo permite descargarlo si es un archivo de determinadas extensiones, ".pdf", ".apk", entre otros. Los demás formatos de archivo simplemente los intenta abrir en una nueva tab, así que para este navegador el xml debe ser descargado con una extensión que no sea ".xml". El usuario de Android Browser puede luego elegir cambiar la extensión al descargarla o no, ya que puede perfectamente cargar dicho archivo y el sistema los reconocerá y cargará los comportamientos.

8.4.2. Edición de código generado en Android Browser

Debido a la limitación de la biblioteca CodeMirror (y de sus similares) explicada en la sección 6.2.5 existe un pequeño error conocido en el feature de edición de código en el navegador Android Browser. Consiste en que la tecla backspace deja de borrar frente a espacios vacíos, se debe hacer touch para volver a posicionar el cursor para que el backspace funcione otra vez correctamente.

8.4.3. Soporte parcial para Internet Explorer Mobile

El sistema fue probado en Internet Explorer Mobile donde la experiencia fue inferior a la deseada. Por las siguientes razones: Los iconos de la biblioteca FontAwesome [16] no son renderizados, se muestran cuadrados en vez de los iconos utilizados, eso se debe a que el navegador IE de Windows Phone no soporta fuentes descargables como EOT, TTF/OTF y WOFF. Por más que el feature de CSS3 @font-face permite usar una fuente desde un recurso externo al fichero CSS, las fuentes utilizadas por FontAwesome no son soportadas.

Por otro lado, se tiene que no es posible realizar drag & drop de los bloques de la biblioteca Blockly. El comportamiento es bastante extraño, dado que se renderizan los bloques y pueden ser

seleccionados desde el toolbox, quedando posicionados en el workspace pero quedan en una posición fija sin poder ser arrastrados. Este es un error conocido por el equipo de desarrollo de Blockly que plantea en sus trabajos a futuro mayor soporte para el navegador Internet Explorer.

Capítulo 9

Extensibilidad del Sistema

En este capítulo se mencionarán algunas formas de extender el sistema, sus diferentes dificultades y el impacto de estas opciones sobre las demás funcionalidades ya implementadas.

9.1. Creación de bloques nuevos

Para agregar bloques nuevos el desarrollador debe considerar 3 partes. La primera es agregar la definición del bloque en alguno de los archivos JavaScript que están en la carpeta "blocks". La estructura es la mostrada en la figura 34.

```
Blockly.Blocks['NOMBRE_DEL_BLOQUE'] = {
    init: function() {
    }
}
```

Figura 34: Código JavaScript para agregar un bloque en Blockly.

Dentro de la función init, se inicializa el bloque. Se recomienda observar otros bloques para copiar sus atributos instanciados en la función init, también se puede utilizar la aplicación BlockFactory de blockly.

La segunda parte es agregar el código lua que generará el bloque nuevo. Para esto se debe incluir dicho código en alguno de los archivos JavaScript que se encuentran en la carpeta generators/lua. Su forma es la mostrada en la figura 35.

```
Blockly.Lua["NOMBRE_DEL_BLOQUE"] = function(block) {
}
```

Figura 35: Código JavaScript para agregar el código generado por un bloque en Blockly.

Dicha función debe retornar el código Lua que genera el bloque.

La tercera y última parte es agregar el tag del bloque al toolbox en la categoría que se quiera. Esto se realiza en el archivo de cada idioma en la carpeta "generated". Por ejemplo un tag simple sería "*<block type="NOMBRE_DEL_BLOQUE"></block>*".

9.2. Kit robótico

En esta sección se detallara el subsistema RobotInterface con la intención de puntualizar las características necesarias para dar soporte al kit del robot Butiá y exponer los cambios que deben ser introducidos para extender este subsistema con otro kit robótico.

9.2.1. Robot Interface

Este subsistema fue detallado en el capítulo "Arquitectura del Sistema", dando detalles de su diseño que permite generalizar la comunicación con interfaces robóticas. Por un lado ofrecer a los usuarios avanzados una forma de configuración externa para el kit robótico desde un archivo XML llamado "robotic-kit.xml" que mantiene el formato detallado en la sección "Configuración". Y por otro lado, permite extender la codificación del archivo RobotInterface.lua para adaptarse a nuevos kits robóticos.

9.2.1.1. Configuración externa

El elemento "devices" del archivo XML mencionado permite definir desde que interfaz el subsistema RobotInterface interpreta los dispositivos que contiene el kit robótico. En principio, es posible definir el atributo "from" con el valor "bobot" para utilizar la biblioteca Bobot que tiene acceso a los módulos de la placa USB4Butiá o "xml" para interpretar el archivo de configuración mencionado (en este caso también se valida la disponibilidad de los dispositivos con respecto a bobot server). Entonces, este parámetro debe indicar la nueva interfaz robótica a utilizar con este parámetro, por ejemplo usando from="arduino".

9.2.1.2. Codificar un nuevo kit robótico

Para extender el fichero RobotInterface.lua se debe implementar, en primer lugar, las funciones necesarias para controlar los sensores y actuadores:

M.execute(sens, func, params, caller): Ejecuta la función "func" del sensor/actuador "sens" con los parámetros "params". Si la función del sensor retorna un valor, este se entrega al usuario que invocó, "caller".

M.stop_actuators(): Detiene todos los actuadores del robot de forma inmediata.

Luego, para poder adaptarse a un kit robótico que cambia dinámicamente fue implementado un procedimiento que se divide en 3 etapas, éstas serán mencionadas a continuación indicando que funciones deben ser extendidas:

- Identificación de dispositivos: se debe proporcionar una función que siga la lógica de parse_bobot o parse_xml para la nueva interfaz robótica. Esta nueva función inicializa una tabla que contiene cada dispositivo del kit, y para cada dispositivo se mantiene una tabla con todas las funciones que éste ofrece (se deben respetar la estructura de esta tabla). Además, es necesario agregar en la función **M.list_devices_functions()** una sentencia else if que consulte por el valor del atributo "from" (siguiendo el ejemplo anterior "arduino").
- Generación de código javascript: este subsistema genera dinámicamente los archivos mencionados en la sección de "Generación de bloques nuevos" para el kit en cuestión. Por lo que es necesario sobrescribir las funciones **write_blocks()** y **write_code()** que reciben como parámetros los dispositivos y las funciones que van a ser generados.
- Actualización de dispositivos: la interfaz robótica está sujeta a constantes cambios en tiempo de ejecución, en esta etapa se comprueba cada cierto periodo de tiempo que los dispositivos disponibles para el usuario no quedaron obsoletos (ya sea por ejemplo, cuando se cambia un sensor de puerto). De todas formas esta etapa no se deben aplicar cambios desde la codificación.

9.3. Arquitectura robótica

Modificar el sistema para adaptarlo a una arquitectura nueva puede resultar relativamente sencillo si se mantiene la orientación a comportamientos y bastante complejo en caso contrario. La implementación de la arquitectura en el sistema se divide en dos partes. La primera parte es la implementación de los comportamientos. Esta está comprendida en la definición de los bloques de comportamiento y acción sin disparador y además en las distintas lógicas para tomar dichos bloques como bloques bases y únicos. Estas lógicas que incluyen marcarlos como listos, guardarlos, abrirlos, entre otras, están en las bibliotecas de JavaScript common, yatay y Tablet. La segunda parte es la administración de los comportamientos, es decir cómo se organizan para ejecutarse de forma ordenada. El encargado de esto es la biblioteca de Lua "RobotTaskManager" como se explica en el capítulo "Algoritmo Principal".

Para implementar una arquitectura nueva dependerá entonces de la naturaleza de esta para ver cuánto se debe modificar el sistema. Si está basada en comportamientos entonces basta con modificar la biblioteca "RobotTaskManager" y los bloques de comportamiento y acción sin disparador. Sin embargo si no está basada en comportamientos entonces su impacto es alto y también se deberá modificar las bibliotecas common.js, yatay.js y Tablet.js. En estas se deberán eliminar los diferentes controles que detectan que haya un bloque de comportamiento o acción sin disparador

como bloque base para efectuar las distintas acciones, como ser autosave (yatay.js), marcar como listos (common.js y Tablet.js), ejecutar (common.js), entre otras. Así mismo se debe modificar el caso de uso editar y ver código (common.js) ya que actualmente muestra en pestaña cada uno de los comportamientos listos. Además se debe modificar el código de la ejecución y debug para quitar la manipulación de los códigos basadas en comportamientos (common.js).

Parte III

Resultados

Capítulo 10

Conclusiones y trabajo a futuro

10.1. Conclusiones

En el transcurso de este proyecto de grado el grupo debió enfrentarse con diversos desafíos entre los que se destaca la utilización de variadas componentes de hardware, como la placa Raspberry Pi y la placa USB4Butiá, así como una plataforma robótica que sigue evolucionando constantemente. A eso se suman las dificultades que surgen en la elaboración de una aplicación flexible a una cantidad tan variada de dispositivos (Tablets, teléfonos inteligentes e inclusive ordenadores de escritorio), sistemas operativos y navegadores web. En especial a la hora de brindar compatibilidad con el navegador nativo de Android que posee numerosas limitaciones, especificadas anteriormente. Otro aspecto a considerar son los LPVs (basados en bloques) que fueron estudiados, los cuales se encuentran en su mayoría en etapa de desarrollo generando incompatibilidades e inestabilidades, causadas por las constantes variaciones en dichas bibliotecas. Incluso, es importante mencionar, las consideraciones y precauciones que conlleva el desarrollo de una herramienta con fines educativos, pensada para contribuir a la robótica educativa.

El proceso de investigación comprendió, a gran escala, el análisis en profundidad de dos grandes alternativas para lograr la comunicación con la placa USB4Butiá, la evaluación de distintas arquitecturas robóticas pertenecientes al paradigma reactivo y alternativas que promuevan su construcción, así como la valoración de los LPVs más actuales. Esto determinó que el grupo necesitara más tiempo del estimado para esta primera etapa, pero dejando como saldo positivo los hallazgos y decisiones que favorecieron categóricamente al desarrollo del sistema y la realización de distintos prototipos que mitigaron posibles factores de riesgo para la etapa de implementación.

Para lograr una buena coordinación del trabajo el grupo utilizó herramientas como Google Drive para la documentación y GitHub como repositorio del código fuente, además del constante mantenimiento de la página web del proyecto para facilitar a todos los interesados la información sobre el mismo. Se tuvieron gratificantes experiencias como la participación en el evento Sumo.uy para presentar los avances del proyecto y varias participaciones en las reuniones del Proyecto Butiá con intención de acercar el sistema a los primeros usuarios.

Por último, resaltar que los objetivos del proyecto fueron cumplidos satisfactoriamente por el grupo, logrando proporcionar una nueva herramienta construida en base a tecnologías emergentes,

que modela un entorno de desarrollo con las características requeridas para suministrar a los estudiantes, escolares y liceales, un modo de acercarse a la robótica educativa usando los dispositivos móviles de última generación.

10.2.1 Características de la aplicación

A continuación se listan algunas de las características de la aplicación Yatay:

Multiplataforma: Yatay es una aplicación Web desarrollada en HTML5, y como tal, es independiente del sistema operativo.

Adecuada para dispositivos móviles: Se desarrollaron interfaces distintas para Smartphones y para Tablets, por lo cual la aplicación puede utilizarse indistintamente desde computadoras, Tablets o Smartphones.

Gratuito: El sistema desarrollado es completamente gratuito, y también lo son cada una de sus dependencias y bibliotecas utilizadas.

Código Abierto: El código de este sistema es abierto y está publicado en GitHub. Este está disponible para su utilización y modificación basada en las condiciones del software libre, bajo la MIT License [29].

Amigable: La interfaz del sistema fue pensada con el propósito de ser amigable, simple e intuitiva de utilizar. Se consideraron en este sentido algunas influencias de sistemas como Tortubots y Scratch que ya han sido utilizados exitosamente por estudiantes.

Interfaz Guiada: La aplicación si bien es intuitiva, también guía al estudiante en su utilización mediante un tutorial al inicio y mensajes breves.

Didáctico: Yatay permite en todo momento al estudiante observar el código Lua generado por los bloques que programó. Además este código Lua puede ser editado y probado por el estudiante, al que se le da también la oportunidad de exportar su trabajo. De esta forma se incentiva el aprendizaje de la programación en edades tempranas y permite que el estudiante pueda alcanzar una comprensión mayor de su trabajo que con otros IDE's.

Soporte Hotplug: Yatay reacciona dinámicamente a los cambios de hardware de la plataforma Butiá, disponibilizando automáticamente al usuario los sensores y actuadores conectados sin la necesidad de que el sistema deba ser reiniciado. El sistema consulta por cambios en el hardware conectado cada unos pocos segundos y disponibiliza los cambios.

Autosave y persistencia simple: La persistencia en el sistema Yatay es simple y garantiza evitar la pérdida de datos, ya que posee guardado automático en la base de datos del servidor además de realizar guardados automáticos en el web cache del dispositivo cliente (WebStorage) y permitir la exportación de los datos al filesystem del cliente.

Cooperativo: El sistema está pensado para que pueda ser utilizado de forma cooperativa por grupos de estudiantes, trabajando sobre el mismo proyecto y el mismo robot.

Extensible: Como se detalla en el capítulo de extensibilidad, el sistema puede ser modificado fácilmente para agregar nuevos kits robóticos así como bloques e incluso modificar la arquitectura robótica.

Multilenguaje: El sistema permite desplegar la interfaz en español o en Ingles, siendo además muy sencillo extender el soporte a nuevos lenguajes.

10.2. Trabajo a futuro

Si bien el software logrado cumple los requerimientos funcionales y no funcionales y supera algunos como el de compatibilidad, existen varios agregados y modificaciones interesantes que se podrían plantear como trabajo a futuro para esta aplicación. Se enumeran y detallan a continuación las que el grupo encontró más interesantes:

10.2.1. Versiones finales de Blockly y mejoras de otras bibliotecas

Al momento de escritura de este informe, el proyecto de Blockly está aún muy activo, realizando commits diarios de correcciones y agregados. Si bien este proyecto se ajustó a una versión estable reciente de Blockly, se plantea como trabajo a futuro mantener la adaptación de Yatay a las últimas versiones por salir de la biblioteca para recoger las mejoras y correcciones que esta introduzca, más aún dado el hecho que no se ha presentado una release final. Así mismo, se presenta un desafío similar con las demás bibliotecas o sistemas utilizados, como CodeMirror (quizás corrigiendo la limitación explicada en 6.2.5), FontAwesome y FileSaver en la parte cliente, y Toribio, Lumen y Bobot en la parte del servidor.

10.2.2. Administración y persistencia de sensores creados

Si bien escapó al alcance y requerimientos de este proyecto, parece útil que el sistema sea capaz de almacenar y disponer a futuro los sensores creados (no los pertenecientes al kit robótico sino los creados por el usuario con el bloque “crear sensor”). Esto permitiría la utilización de sensores creados anteriormente por, inclusive, otros usuarios. Fomentando la reutilización, a través de la persistencia de estos “sensores nuevos”, y con la posibilidad de obtenerlos al inicio o de forma manual. Cabe aclarar que estas funcionalidades sólo introducen mayor facilidad en la manipulación de dichos sensores, ya que actualmente no se presentan limitaciones para lograr lo anterior: La persistencia de comportamientos permite recuperar éste tipo de bloques, que si bien resulta más engorroso no limita la creación de sensores. Por último, destacar que estas funcionalidades son para usuarios un poco más avanzados y no a los que apuntó este proyecto.

10.2.3. Kit Robótico

Otra tarea interesante sería que se diera soporte para otro Kit Robótico en Yatay además de Butiá. Por ejemplo, el kit de Lego Mindstorms NXT ha tenido éxito en la robótica educativa y es utilizado también en Uruguay. Para lograr esto se debería implementar las funciones definidas en la interfaz robótica "RobotInterface" que ya se detallaron en la Parte II.

10.2.4. Otros paradigmas robóticos o arquitecturas

En este proyecto de grado se utilizó una arquitectura basada en prioridades dentro del paradigma reactivo. Sin embargo, se podrían implementar otros paradigmas robóticos o arquitecturas. Por ejemplo, dentro del mismo paradigma, se podría implementar subsumption. Estas implementaciones distintas permitirían utilizar Yatay para enseñar arquitecturas y paradigmas robóticos. La lógica de la arquitectura se encuentra en la biblioteca lua "RobotTaskManager", además habría que hacer modificaciones en los bloques de comportamiento y en las bibliotecas de JavaScript common.js y yatay.js.

10.2.5. Módulo para administración de proyectos

Hoy en día el sistema no tiene una funcionalidad para la administración directa de proyectos. El sistema si posee la funcionalidad de creación, que se realiza al ingresar el nombre de un proyecto nuevo en el modal de inicio. Sin embargo, sería útil poder renombrar, editar y eliminar proyectos almacenados, como forma de mantener una base de datos poblada solo con datos útiles y por tanto aumentando la eficacia del uso de esta.

10.2.6. Edición multiusuario de comportamientos

Una idea un poco compleja aunque quizás útil es la de que varios usuarios puedan editar al mismo tiempo un mismo comportamiento. Esto significa que los cambios hechos por un usuario sobre un comportamiento, pueden ser vistos en tiempo real por los demás usuarios. Esto por un lado estimularía aún más el trabajo cooperativo, pero por el otro, al ser los comportamientos pensados para ser breves, este nivel de multiusuario puede que no ofrezca una ventaja tan útil como parece, convirtiendo el desarrollo en algo entreverado y hasta confuso al intentar editar pocas partes entre varios.

10.2.7. Extender el soporte multilenguaje

Hoy en día Yatay soporta inglés y español como lenguajes del sistema. Se facilita el cambio de lenguaje mediante un boton con doble flecha en la barra de acciones lateral. Si bien la cobertura de estos dos idiomas en el sistema son las más importantes, sería interesante extender el soporte a aún más lenguajes, como ser portugués, francés, entre otros.

Referencias

[1] Achkar, Alejandro; Margalef, Andrés. IDE de programación orientado al desarrollo de arquitecturas robóticas basadas en comportamientos: Estado del Arte. Proyecto de grado. Universidad de la República, Facultad de Ingeniería, Montevideo, 2012.

[2] Arkin, Ronald C. Behavior-Based Robotics. The MIT Press, 1998. 441p. ISBN 0-262-01165-4

[3] Burnett, Margaret M. Visual Programming. John G. Webster (ed.). Encyclopedia of Electrical and Electronics Engineering. John Wiley & Sons Inc. 1999. Disponible en Web: <http://www.cs.auckland.ac.nz/courses/compsci732s1c/archive/2005/lectures/WhatIsVP.pdf>

[4] Can I use calc() as CSS unit value? Compatibility table for support of calc() as CSS unit value in desktop and mobile browsers [en línea]. Actualizada: abril de 2014 [Fecha de consulta: enero de 2014]. Disponible en Web: <http://caniuse.com/calc>

[5] Can I use download attribute? Compatibility table for support of Download attribute in desktop and mobile browsers [en línea]. Actualizada: abril de 2014 [Fecha de consulta: enero de 2014]. Disponible en Web: <http://caniuse.com/download>

[6] Can I use SVG? Compatibility table for support of SVG in desktop and mobile browsers [en línea]. Actualizada: abril de 2014 [Fecha de consulta: enero de 2014]. Disponible en Web: <http://caniuse.com/SVG>

[7] Casares Charles, Juan Pablo. AMIVA: Ambiente para la instrucción visual de algoritmos. Trabajo de grado. Instituto tecnológico autónomo de México, México DF, 1999. Disponible en Web: <http://usablehack.com/amiva/AMIVA.pdf>

[8] Castro Rojas, María Dolores; Acuña Zuñiga, Ana Lourdes. Propuesta Comunitaria con robótica educativa: valorización y resultados de aprendizaje. Teoría de la Educación. Educación y Cultura en la Sociedad de la Información. 2012. Disponible en Web: <http://www.redalyc.org/articulo.oa?id=201024390006> ISSN: 1138-9737

[9] CodeMirror. Issues. Android Chrome Backspace and Enter do not fire keydown events [en línea]. Actualizada: febrero de 2014 [Fecha de consulta: febrero de 2014]. Disponible en Web: <https://github.com/marijnh/CodeMirror/issues/2234>

[10] Deliberative Agents. Notes [en línea]. Actualizada: 2007 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://cgi.cse.unsw.edu.au/~aishare/index.php>.

[11] Donizio, Tiago; Currie, Doug. LuaSQLite3. Documentation [en línea]. Actualizada: octubre de 2013 [Fecha de consulta: enero de 2014]. Disponible en Web: <http://lua.sqlite.org/index.cgi/doc/tip/doc/lsqlite3.wiki>

[12] Eligrey. GitHub FileSaver Repository [en línea]. Actualizada: febrero de 2014 [Fecha de consulta: febrero de 2014]. Disponible en Web: <https://github.com/eligrey/FileSaver.js/>

[13] eLinux. The Raspberry Pi Wiki [en línea]. Actualizada: marzo de 2014 [Fecha de consulta: marzo de 2014]. Disponible en Web: <http://elinux.org/RaspberryPiBoard>

[14] Frase, Neil; Neutron, Quynh; Chris, Pirich; Spertus, Ellen; Wagner Phil. Blockly. A visual programming editor [en línea] [Fecha de consulta: julio de 2013]. Disponible en Web: <https://code.google.com/p/blockly/>

[15] Gallego, Crómulo; Pérez, Royman; Gallego, Adriana; Pascuas, John. Didáctica Constructivista: Aportes y Perspectivas [en línea]. 2004 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.saber.ula.ve/bitstream/123456789/19856/2/articulo14.pdf>. ISSN: 1316-4910

[16] Gandy, Dave. FontAwesome. The iconic font designed for Bootstrap [en línea]. Actualizada: enero de 2014 [Fecha de consulta: enero de 2014]. Disponible en Web: <http://fortawesome.github.io/Font-Awesome/>

[17] Gouy, Isaac. The Computer Language Benchmarks Game [en línea] [Fecha de consulta: julio de 2013]. Disponible en Web: <http://benchmarksgame.alioth.debian.org/u64/benchmark.php?test=all&lang=lua&lang2=python3&data=u64>

[18] Gravelle, Rob. Comet Programming: Using Ajax to Simulate Server Push [en línea]. Actualizada: marzo de 2009. [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.webreference.com/programming/javascript/rg28/index.html>

[19] HiddenTao, JavaScript client-side file generation and download [en línea]. Actualizada: julio de 2011 [Fecha de consulta: febrero de 2014]. Disponible en Web: <https://www.hiddentao.com/archives/2011/07/04/javascript-client-side-file-generation-and-download/>

[20] LEJOS. Java for LEGO Mindstorms. Behavior programming [en línea]. Actualizada: noviembre de 2008. [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.lejos.org/nxt/nxj/tutorial/Behaviors/BehaviorProgramming.htm>

[21] Maloney, John; Resnick, Mitchel; Rusk, Natalie; Silverman, Brian; Eastmond, Evelyn. The Scratch Programming Language and Environment. ACM Trans. Comput. Educ., noviembre de 2010. Vol. 10, No. 4, Art 16. Disponible en Web: <http://web.media.mit.edu/~jmaloney/papers/ScratchLangAndEnvironment.pdf>

[22] Modernizr. Documentation [en línea]. Actualizada: abril de 2013. [Fecha de consulta: julio de 2013]. Disponible en Web: <http://modernizr.com/docs/>

[23] Nebel, Andres; Rozza, Renzo. IDE Android para la Programación de Comportamientos Robóticos: Arquitectura y Diseño del Sistema. Proyecto de grado. Universidad de la República, Facultad de Ingeniería, Montevideo, 2014.

[24] Nebel, Andres; Rozza, Renzo. IDE Android para la Programación de Comportamientos Robóticos: Especificación de Requerimientos. Proyecto de grado. Universidad de la República, Facultad de Ingeniería, Montevideo, 2013.

[25] Nebel, Andres; Rozza, Renzo. IDE Android para la Programación de Comportamientos Robóticos: Estado del Arte. Proyecto de grado. Universidad de la República, Facultad de Ingeniería, Montevideo, 2013.

[26] Nebel, Andres; Rozza, Renzo. IDE Android para la Programación de Comportamientos Robóticos: Manual de Usuario. Proyecto de grado. Universidad de la República, Facultad de Ingeniería, Montevideo, 2014.

[27] Nebel, Andres; Rozza, Renzo. IDE Android para la Programación de Comportamientos Robóticos: Pautas para la interfaz de usuario. Proyecto de grado. Universidad de la República, Facultad de Ingeniería, Montevideo, 2013.

[28] Nebel, Andres; Rozza, Renzo. IDE Android para la Programación de Comportamientos Robóticos: Plan de pruebas. Proyecto de grado. Universidad de la República, Facultad de Ingeniería, Montevideo, 2014.

[29] Open Source Initiative. MIT License [en línea] [Fecha de consulta: febrero de 2014]. Disponible en Web: <http://opensource.org/licenses/mit-license.html>

[30] Pinto Salamanca, María Luisa; Barrera Lombana, Nelson; Pérez Holguín, Wilson Javier. Uso de la robótica educativa como herramienta en los procesos de enseñanza. I^2+D. Oscar Oswaldo Rodríguez Díaz. 2010, Vol 10, No 1, p. 15-23. Disponible en Web: <http://virtual.uptc.edu.co/revistas2013f/index.php/ingenieria_sogamoso/article/download/912/912>

[31] Presidencia de la República. Noticias. Plan Ceibal entregará Tablets a alumnos de 1.er año y XO con pantalla táctil a los de 3° [en línea]. Actualizada: febrero de 2014. [Fecha de consulta: marzo de 2014]. Disponible en Web: <http://www.presidencia.gub.uy/comunicacion/comunicacionnoticias/brechner-plan-ceibal-Tablets-alumnos-primer-ano>

[32] Processes and Threads. Android for Developers. Official Website [Fecha de consulta: julio de 2013]. Disponible en Web: <http://developer.android.com/processes-and-threads.html>

[33] Ramírez Toledo, Antonio. El constructivismo pedagógico [en línea]. [Veracruz, México] Universidad Veracruzana [Fecha de consulta: julio de 2013]. Disponible en Web: <http://ww2.educarchile.cl/UserFiles/P0001/File/El%20Constructivismo%20Pedag%C3%B3gico.pdf>

[34] Sexton, Alex; Holzmann, Ralph. Yepnope. Home page [en línea]. Actualizada: abril de 2012 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://yepnopejs.com/>

[35] Seymour Papert. FactoriaHistorica [en línea]. Actualizada: noviembre de 2013. [Fecha de consulta: noviembre de 2013]. Disponible en Web: <http://factoriahistorica.wordpress.com/2013/11/19/seymour-papert/>

[36] Shin, Hochul. SK Planet. Python vs Lua [en línea]. Actualizada: marzo de 2012. [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.slideshare.net/cybrshin/lua-vs-python>

[37] Sugar Labs. Turtle Art. Wiki Sugar Labs [en línea]. Actualizada: julio de 2013. [Fecha de consulta: julio de 2013]. Disponible en Web: <http://wiki.sugarlabs.org/go/Activities/TurtleArt>

[38] TortuBots. Wiki Proyecto Butiá [en línea]. Actualizada: junio de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.fing.edu.uy/inco/proyectos/Butiá/TortuBots>

[39] USB4All. Wiki Proyecto Butiá [en línea]. Actualizada: abril de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.fing.edu.uy/inco/proyectos/Butiá/Usb4all>

[40] Vavassori Benitti, Fabiane Barreto; Vahldick, Adilson; Urban, Diego Leonardo; Krueger, Matheus Luan; Halma, Arvid. Experimentação com Robótica Educativa no Ensino Médio: ambiente, atividades e resultados. En: XXIX Congresso da Sociedade Brasileira de Computação. [Bento Gonçalves]: 2009. p. 1811-1820. Disponible en Web: <http://robolab.inf.furb.br/robolab/artigos/robolab/wie2009.pdf>

[41] Visca, Jorge. Lumen [en línea]. Actualizada: febrero de 2014 [Fecha de consulta: febrero de 2014]. Disponible en Web: <https://github.com/xopxe/Lumen>

[42] Visca, Jorge. Toribio [en línea]. Actualizada: febrero de 2014 [Fecha de consulta: febrero de 2014]. Disponible en Web: <https://github.com/xopxe/Toribio>

[43] WebSocket. Browser support [en línea]. Actualizada: julio de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://en.wikipedia.org/wiki/WebSocket>

Apéndice A

Glosario

En esta sección se detallan y definen ciertos términos básicos para la comprensión del material presentado en este documento.

Ajax (*Asynchronous JavaScript and XML*): Es una técnica de desarrollo web que permite intercambiar información con un servidor web de forma asíncrona.

API (*Application Programming Interface*): Conjunto de funciones y procedimientos que ofrece una determinada biblioteca para ser usada por otro software manteniendo un nivel de abstracción.

ARM (*Advanced RISC Machine*): Arquitectura RISC de 32 bits desarrollada por la compañía ARM Holdings.

Client-Side: En un sistema, se denomina client-side al código que se ejecuta en el dispositivo cliente que accede a la aplicación.

Cookie: Es información enviada por un servidor Web y mantenida por un navegador web de forma local en la maquina cliente, esta información puede ser accedida luego de forma local (client-side) por la aplicación que la envió.

CSS (*Cascading Style Sheets*): Lenguaje de estilo utilizado para describir la apariencia y el formato de un documento escrito en un lenguaje de marcas (HTML, XML, SVG).

DOM (*Document Object Model*): API que proporciona un conjunto de objetos para representar documentos HTML o XML. Un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

Dynamixel AX-12: Actuador de rotación (servo actuator en inglés) usado para robótica de mediano o pequeño porte, en particular éste tipo de motores sirven para mover las ruedas en el robot Butiá.

GET: Método definido en el protocolo HTTP que solicita información a un servidor web.

GUI (*Graphical User Interface*): Programa informático que funciona como interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

HDMI (*High-Definition Multimedia Interface*): Interfaz compacta de audio y video para la transferencia de datos entre una fuente de audio o video digital, como puede ser un ordenador, y un monitor compatible o televisor digital.

Hotplug: Es la capacidad de reconocer y detectar la conexión y desconexión de dispositivos sin la necesidad de reiniciar el sistema.

Hotspot: Dispositivos que dan soporte físico para la creación y mantenimiento de redes inalámbricas, generalmente a través de WiFi. Administrando la conexión y desconexión de dispositivos a la red.

HTTP (*HyperText Transfer Protocol*): Protocolo de comunicación de datos digitales perteneciente a la capa de aplicación. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor, usado en cada transacción de la *World Wide Web*.

IDE (*Integrated Development Environment*): Software compuesto por un conjunto de herramientas para la programación de sistemas. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios.

IP (*Internet Protocol*): Protocolo de comunicación de datos digitales perteneciente la capa de red, que permite transmitir datos a través de una red de distintas redes físicas previamente enlazadas. En general se utiliza el término, dirección IP, para referir al número que identifica un nodo dentro de una red que utilice el protocolo IP.

JavaScript: Lenguaje de programación interpretado, se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Comúnmente utilizado para agregar funcionalidad y estilos de una página web en client-side.

JQuery: Biblioteca de JavaScript que permite simplificar la manera de interactuar con los documento HTML, manipular el árbol DOM, desarrollar animaciones, entre otras cosas.

JSON (*JavaScript Object Notation*): Formato ligero para el intercambio de datos.

Plug&Play: Término usado para definir dispositivos informáticos con la característica de reconocer una componente de hardware sin requerir configuraciones previas o reinicios de sus sistemas.

POST: Método definido en el protocolo HTTP que envía información a un servidor web para que sea procesada por este.

RAM (*Random-Access Memory*): Memoria volátil de trabajo para el sistema operativo encargada de almacenar todas las instrucciones que ejecuta el procesador y otras unidades de cómputo.

RJ45 (*Registred Jack 45*): Interfaz física usada comúnmente para conectar redes cableadas, como cables de red Ethernet.

SBC (*Single Board Computer*): Computadora completa con un sólo circuito. Se centra en un solo microprocesador con RAM, E/S y el resto de las características funcionales de un computador en una sola placa de tamaño reducido.

SDK (*Software Development Kit*): Conjunto de herramientas para el desarrollo de software que le permite al programador aplicaciones para un sistema en concreto.

Server-Side: En un sistema, se denomina server-side al código que se ejecuta en el servidor de la aplicación y no en el dispositivo que accede a éste.

Smalltalk: Lenguaje reflexivo de programación, orientado a objetos y con tipado dinámico.

SQL (*Structured Query Language*): Lenguaje declarativo de acceso a base de datos relacionales que permite especificar diversos tipos de operaciones en ellas.

Squeak: Lenguaje de programación o dialecto dentro de Smalltalk.

Sugar: Interfaz gráfica de usuario, libre y de código abierto, desarrollada y diseñada para el proyecto OLPC. Enfocada para ofrecer a niños un entorno de aprendizaje interactivo.

TCP (*Transmission Control Protocol*): Protocolo de comunicación de datos digitales perteneciente a la capa de transporte, orientado a la comunicación fiable.

UML (*Unified Modeling Language*): Lenguaje de modelado de sistemas de software. Lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

USB (*Universal Serial Bus*): Estándar de la industria que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre ordenadores y periféricos.

WiFi: Sistema de comunicación de datos inalámbrico, que utiliza ondas electromagnéticas con una cierta modulación para intercambiar mensajes entre equipos, sin necesidad de una conexión física directa entre éstos.

WLAN (*Wireless Local Area Network*): Sistema de comunicación inalámbrico flexible. Frecuentemente utilizada como alternativa a las LAN cableadas o como extensiones de éstas.

WPA (*WiFi Protected Access*): Sistema para proteger las redes inalámbricas.

XML (*Extensible Markup Language*): Lenguaje de marcas desarrollado por la W3C utilizado para almacenar datos de forma legible.

Proyecto de Grado

IDE Android para la Programación de Comportamientos Robóticos

Estado del arte

Andrés Nebel - Renzo Rozza

Tutores
Ing. Andrés Aguirre, Ing. Rafael Sisto

24 de Abril del 2014

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay

Índice

Capítulo 1 7
Introducción 7
1.1 Motivación y objetivo del proyecto 7
Capítulo 2 10
Paradigmas Robóticos 10
2.1 Arquitectura y Paradigmas de Sistemas Robóticos 10
2.2 Paradigma Reactivo 11
2.3 Arquitectura Subsumption 13
2.4 Arquitectura basada en prioridades 14
Capítulo 3 15
Enseñanza de la Robótica 15
3.1 Constructivismo e influencias en la enseñanza robótica 15
3.2 Lenguajes de Programación Visual 17
Capítulo 4 21
Experiencias anteriores 21
4.1 Experiencias regionales e internacionales 21
Capítulo 5 26
Proyecto Butiá y Plataforma Robótica 26
5.1 Acerca del Proyecto 26
5.2 Hardware 27
5.3 Software 27
5.4 Bobot 28
5.5 Complementos 29
Capítulo 6 32
Aspectos básicos de arquitectura 32
6.1 Introducción 32

6.2 Arquitecturas 32
Capítulo 7 35
HTML5 y bibliotecas útiles 35
7.1 Ventajas de HTML5 35
7.2 Bibliotecas 36
Capítulo 8 39
Análisis y estudio de soluciones viables 39
8.1 Identificación de problemas 39
8.2 Comunicación y arquitectura 39
8.3 Interfaz gráfica e interacción con el usuario 40
8.4 Concurrencia y orientación a comportamientos 40
Capítulo 9 43
Estudio de comunicación con USB4Butiá 43
9.1 Introducción 43
9.2 Conexión USB y USB Hosting 43
9.3 Soporte de Android a USB Host 45
9.3.1 Prototipo de prueba de disponibilidad de la API USB Host 46
9.3.2 Estadísticas de compatibilidad de USB Host 48
9.3.3 Modo Root en dispositivos Android 49
9.4 Comunicación vía Wi-Fi 49
Capítulo 10 53
Interfaz Gráfica 53
10.1 Introducción 53
10.2 Scratch 53
10.3 Opciones para HTML5 54
10.4 Opciones para Java con Android SDK 59
10.4.1 Catroid 59
Capítulo 11 63
Concurrencia y orientación a comportamientos 63
11.1 Introducción 63

11.2 Semáforos y Threads en Android 63
11.3 Toribio y Lumen 64
11.4 Servidor Web y Remote Shell de Lumen 66
11.5 Websockets en browsers para Android 67
Capítulo 12 70
Portar eButiá en Android 70
12.1 Introducción 70
12.2 IDE eButiá 70
12.3 Portabilidad de Smalltalk 71
12.3.1 Estado actual 71
12.3.2 Comentarios finales y conclusiones de esta sección 74
Capítulo 13 77
Comentarios finales y conclusiones 77
Apéndice A 86
Glosario 86

Índice de Figuras

1 - Rodney Brooks, profesor del MIT
2 - Diagrama de arquitectura deliberativa y reactiva
3 - Módulo de subsumption con inhibir y suprimir.
4 - Activación y ejecución, arquitectura basada en prioridades.
5 - Robot tortuga de Papert.
6 - Ilustración de la Dynabook como fue descrita por Kay en su paper.
7 - Interfaz gráfica del sistema Logo.
8 - NASA Robotics Education Project.
9 - Interfaz gráfica del sistema TortuBots (plugin Butiá).
10 - Proyecto Butiá en Colonia del Sacramento, Junio 2013.
11 - Robot Butiá 2.0.
12 - Placa Raspberry Pi.
13 - Cable USB OTG y USB regular.
14 - Pendrive conectado a una Tablet mediante un cable USB OTG.
15 - USB Host Diagnostics.
16 - Diagrama de componentes (Wi-Fi con Raspberry Pi).
17 - Sistema Scratch 1.2.1.
18 - Interfaz de Scratch 2.0.
19 - Interfaz de Blockly.
20 - Interfaz de Snap!.
21 - Interfaz de Waterbear.
22 - Catroid en un celular con Android.
23 - Ejecución de tarea en nuevo thread con Android SDK.
24 - Creación y utilización de un semáforo simple en Android SDK.
25 - Ejemplo de una tarea activando a otra mediante un signal.
26 - Extracto de una tarea que mueve los motores según sensor.
27 - Echo Test con websocket.org para Firefox en un ordenador.
28 - Soporte de browsers a los distintos protocolos de websockets.
29 - VM tradicional y Event-Driven VM.
30 - Squeak image con CogDroid en Android 2.3 (Motorola Defy).

Capítulo 1

Introducción

1.1 Motivación y objetivo del proyecto

A partir del lanzamiento del Proyecto Butiá en el año 2009, de mano de la Universidad de la República, se ha logrado introducir en cierto grado la robótica a las aulas liceales. Esta es una herramienta pedagógica que propicia la generación de entornos de aprendizaje que potencian necesariamente, entre otros aspectos, la multi e interdisciplinariedad escolar, la exploración, la interacción entre los conocimientos teóricos y su aplicabilidad práctica, la creatividad de los estudiantes fomentando sus capacidades de observación, percepción y sensibilidad así como el desarrollo de la curiosidad y la imaginación.

Desde el año 2009 el grupo MINA del INCO (Instituto de Computación) trabaja en el desarrollo de una plataforma robótica llamada Butiá, que ha impulsado un progresivo avance de la robótica educativa a nivel nacional. Disponibilizando, hasta la fecha, más de 100 robots ubicados en escuelas y liceos de todo el país.

En el contexto de la robótica móvil autónoma, existe un conjunto de arquitecturas de control pertenecientes al paradigma reactivo. Este paradigma es bioinspirado y puede ser enfocado para promover el desarrollo del educando, ya que intenta reflejar la conducta animal entendida por el estudiante. Se utilizan en el paradigma módulos que implementan comportamientos sencillos (sensar y actuar) o la combinación de ellos para lograr comportamientos más complejos. Esto permite que los procesos de enseñanza y de aprendizaje se lleven a cabo en base a conceptos simplificados ya que el paradigma reactivo elimina la planificación y no mantiene un modelo de mundo.

Por otro lado, los avances tecnológicos en los últimos años han fomentado una evolución de los dispositivos móviles. Las Tablets y teléfonos inteligentes han aumentado sus capacidades de procesamiento y dejaron de poseer sistemas embebidos para incorporar sistemas de propósito general. A modo de ejemplo el Plan Ceibal, en este año (2014) entregará por primera vez Tablets a los estudiantes de primer año escolar [31].

Existen varios IDE que permiten programar al robot Butiá. Entre ellos se destaca TortuBots, uno de los más utilizados en los talleres de robótica ofrecidos en el marco del proyecto Butiá. Durante el 2011 y 2012 un grupo de la asignatura proyecto de grado desarrolló un entorno, llamado eButiá, basado en el lenguaje de programación visual Etoys. Dicho entorno de desarrollo, se diferencia en utilizar una arquitectura reactiva Subsumption, que permite mediante comportamientos controlar al robot Butiá desde las computadoras XO [1].

A pesar de que existen varios entornos para programar al robot Butiá, ninguno de éstos permite su ejecución en dispositivos móviles, lo cual restringe su utilización a computadores. Asimismo, salvo puntuales excepciones (como eButiá), no promueven la programación basada en el paradigma reactivo.

Por lo tanto, se considera interesante acompasar las metodologías de enseñanza a los avances tecnológicos, motivando a la investigación de distintas alternativas para proporcionar una herramienta que permita programar al robot Butiá desde dispositivos móviles. Incorporando los beneficios otorgados por las arquitecturas basadas en el paradigma reactivo (véase 2.1.1 Paradigma Reactivo) y los lenguajes de programación visual, conformando un entorno de desarrollo que promueva el crecimiento de la robótica educativa.

Capítulo 2

Paradigmas Robóticos

2.1 Arquitectura y Paradigmas de Sistemas Robóticos

Llamamos arquitectura de un robot autónomo a la manera general de organizar un sistema de control. Ésta describe un conjunto de componentes y la forma en que interactúan [3]. Permitiendo determinar las conductas que exhibe un robot autónomo y definiendo, cómo se activan y cómo se resuelve el problema cuando múltiples comportamientos se activan al mismo tiempo.

Cuanto más complejo es un sistema más relevancia cobra el papel de la organización de sus componentes, siendo la arquitectura quien decide cómo organizar estos procesos internos en robots. Un robot, tiene la tarea de construir o seleccionar señales de entrada a partir de información no específica que sus sensores ofrecen. La naturaleza de los objetivos de estos puede variar enormemente, con prioridades dinámicas que dependen de las necesidades actuales y de la situación del entorno. Por estas razones, no existe una arquitectura válida para todos los entornos y para todos los comportamientos.

Por estas razones, no existe una arquitectura válida para todos los entornos y para todos los comportamientos. Tradicionalmente este campo se ha dividido en tres grandes corrientes paradigmáticas: los sistemas deliberativos (o paradigma jerárquico), los reactivos y los híbridos**.** Estas corrientes se diferencian principalmente en cómo toma la decisión un robot de actuar a partir de lo sensado, más específicamente, en la interrelación entre las 3 principales acciones de la robótica: sensar, planificar y actuar.

Este proyecto se centrará en el tipo de paradigma reactivo, el cual veremos que se caracteriza por eliminar el concepto de planificación y establecer una relación directa entre sensar y actuar. A continuación se hará un análisis de este paradigma, y se explicarán sus ventajas y desventajas, así como el motivo de su elección para este trabajo.

2.2 Paradigma Reactivo

En particular el paradigma reactivo o también denominado paradigma PA (Percepción-Acción) surgió en la década de los '80 y fue desarrollado como alternativa al paradigma Jerárquico (sistemas deliberativos). El concepto fue introducido por Rodney Brooks, profesor del Massachusetts Institute of Technology (MIT). El laboratorio de Inteligencia Artificial de esta Universidad se convertiría luego en el pionero en construcción de robots basados en esta arquitectura de control.

Figura 1: Rodney Brooks, profesor del MIT.

El paradigma reactivo lo que propone es saltarse o disminuir mucho la complejidad de la etapa de planificación, de manera que esta no maneje "modelos del mundo" (estructura de datos global con una estructura a priori del ambiente más información sensorial y cognitiva) enfocándose en una correspondencia más directa entre la percepción y la acción. El nuevo enfoque liga directamente los sensores y los actuadores, sin pasar por las etapas intermedias de modelar y planificar que utilizan los robots deliberativos, logrando de esta manera, una reacción a los eventos mucho más rápida. Se considera al enfoque reactivo como subsimbólico, gracias a que no es necesaria la representación simbólica, ni el razonamiento sobre símbolos para generar comportamiento.

Su aparición parte del comprendimiento de la comunidad científica de que el intelecto animal y también el humano no están completamente atado al pensamiento abstracto, sino que presenta comportamientos que no parecen seguir un patrón deliberativo para decidir cómo actuar. Un ejemplo simple es un reflejo: si me estoy quemando la mano, realizo un movimiento involuntario y repentino para cambiar dicha situación. Como resultado de este cambio de paradigma se reduce la capacidad necesaria para el procesamiento de

percepciones y sus transformaciones a sus respectivas acciones.

Se trata de enfocarse más en los razonamientos simples que comparte todo el reino animal en vez de particularmente los de alto nivel del ser humano. Tiene como beneficio su sencillez, pero como contrapartida una incapacidad de realizar tareas con un nivel de complejidad alta, tales como por ejemplo efectuar cálculos óptimos para toma de decisiones. Mostramos a continuación un esquema a modo de ejemplo de una posible arquitectura reactiva en comparación a una deliberativa [11]:

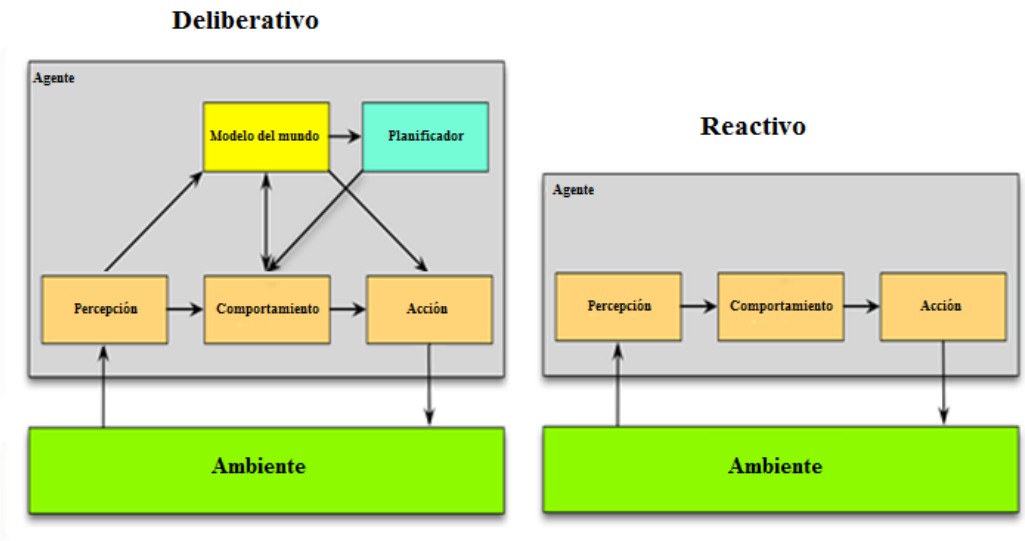


Figura 2: Comparación de dos ejemplos de arquitecturas, una deliberativa y otra reactiva.

Como observamos en la Figura 2, la arquitectura reactiva elimina la necesidad de generar un modelo del mundo para luego efectuar un procesamiento exhaustivo de este y lograr un plan de acción, sino que simplemente toma las percepciones obtenidas mediante los sensores y ejecuta comportamientos preestablecidos.

Situándonos de nuevo en el marco de este proyecto, la robótica educativa, se cuenta con el robot Butiá, dispositivos con sistema operativo Android (de aquí en adelante abreviamos esto a “dispositivos Android”) y posiblemente una placa SBC de gama económica, tenemos que la capacidad de procesamiento de cualquiera de nuestras componentes físicas es poco elevada. Siendo la mejor opción de arquitectura robótica las contempladas por el paradigma reactivo. Por un lado estas permiten no depender de un hardware potente para hacer cálculos simbólicos complicados y por el otro la sencillez de una arquitectura reactiva brinda mayor facilidad a la hora de enseñar que una deliberativa por eliminar la planificación y el concepto de modelo de mundo.

2.3 Arquitectura Subsumption

Dentro del paradigma reactivo existen varias arquitecturas robóticas conocidas que implementan de distinta manera el concepto principal de sensar-actuar del paradigma. En esta sección hablaremos de una de ellas propuesta por Rodney Brooks llamada Subsumption.

Subsumption agrupa comportamientos en capas, por lo cual establece una jerarquía entre las tareas del robot. Las tareas de capas mayores corresponden a comportamientos de más alto nivel de abstracción, mientras que las de más abajo contienen a las tareas más simples. Cada uno de los comportamientos en sí está implementado como una máquina de estados.

Los módulos de capas superiores pueden alterar la entrada o salida de las tareas de capas inferiores, lo cual les da cierto control de manipulación sobre estas. Subsumption describe dos formas de realizar lo anterior: inhibir, el cual apaga la salida de un módulo de menor nivel si hay una salida de una tarea superior, o suprimir que sustituye la entrada de un módulo de menor nivel con la salida de un módulo de mayor nivel. En la figura 3 se ve un diagrama de lo anterior.

Cada capa puede poseer varios comportamientos, por lo tanto el resultado de los actuadores del Robot en un momento dado es el conjunto de todas las ejecuciones concurrentes de la capa actual.

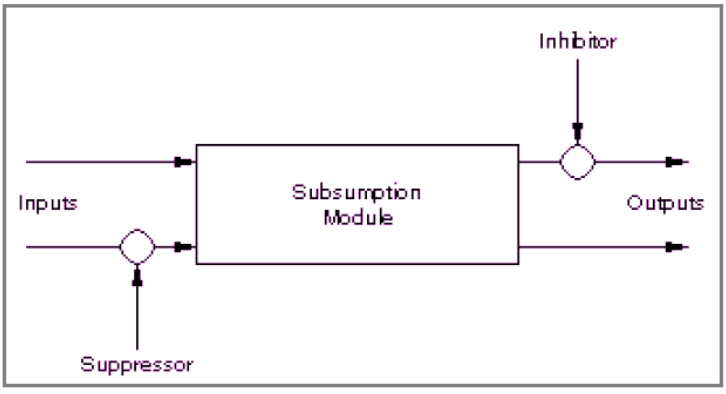


Figura 3: Módulo de subsumption con inhibir y suprimir.

2.4 Simplificación de Subsumption

Existen diversas implementaciones de la arquitectura Subsumption. Algunas de estas, simplifican algunos de sus principales conceptos teóricos. En este proyecto se consideró una de estas implementaciones [25], que tiene como variante que cada comportamiento se implementa en forma de máquina de estados pero con la diferencia de que cada uno de los comportamientos tiene a su vez un valor numérico de prioridad único asociado a él. Se define por tanto una jerarquía de ejecución de comportamientos, ya que existen tareas con mayor prioridad de ejecución de otras. A esta variante de Subsumption se le llamará a lo largo de este documento "Arquitectura basada en prioridades".

La arquitectura establece que siempre habrá un y sólo un comportamiento activo ejecutándose que hará uso de los actuadores. Cualquiera de los comportamientos puede activarse en cualquier momento, pero al activarse, se compara su prioridad con la prioridad de la tarea actualmente activa. Si su prioridad es mayor, entonces desplazaran a la otra tarea para convertirse en la nueva tarea activa, por lo cual pasará a ejecutarse. De lo contrario, continuará la ejecución la tarea que estaba activa, ya que posee mayor prioridad [25]. Un ejemplo se muestra en la figura 4.

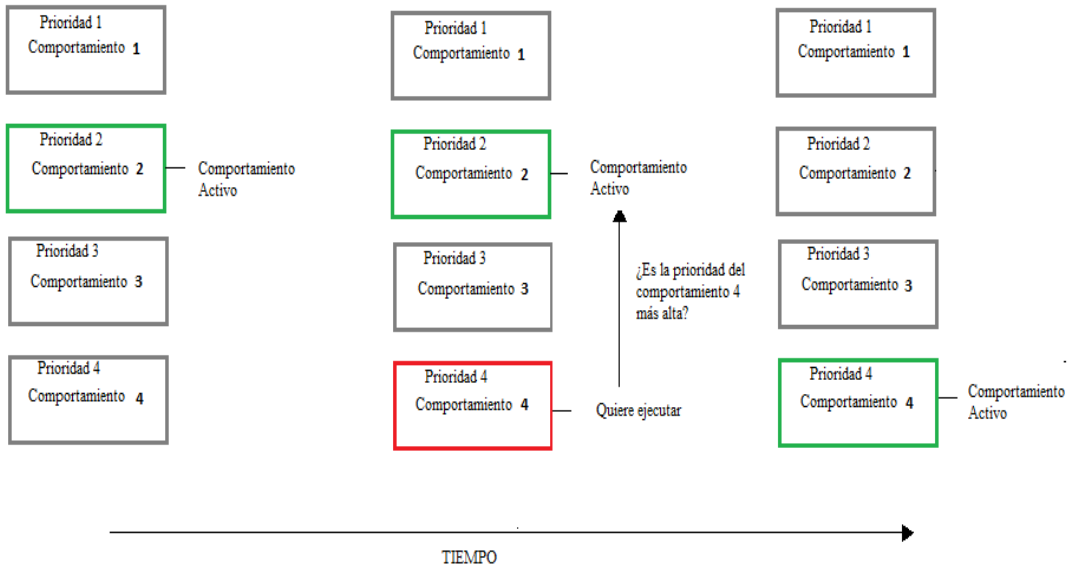


Figura 4: Activación y ejecución de una tarea de mayor prioridad que la tarea activa.

Capítulo 3

Enseñanza de la Robótica

3.1 Constructivismo e influencias en la enseñanza robótica

La enseñanza de la robótica surge como una disciplina en la que se concibe, diseña, desarrolla y operan robots como forma de introducir a los estudiantes desde jóvenes en el estudio de las ciencias y la tecnología [31]. La inclusión de la Robótica en la enseñanza intenta enriquecer las experiencias educativas y atraer la atención de los estudiantes que se ven motivados por la oportunidad de manipular un Robot. Existen numerosas investigaciones y experiencias en este ámbito a nivel global, entre ellas se pueden destacar varias, muchas detalladas en próximos capítulos de este documento. En 1975 en Francia en la Universidad Du Maine, en 1989 en la universidad Autónoma de México, en 1998 el proyecto "Robótica y Aprendizaje por Diseño", realizado conjuntamente por el Centro de Innovación Educativa de la Fundación Omar Dengo y el Ministerio de Educación Pública de Costa Rica (Fundación Omar Dengo, 2004), también el Proyecto Butiá en el ámbito local desde el 2009. [31]

El constructivismo es una corriente pedagógica. Los primeros trabajos asociados a esta fueron los de Ernst von Glasersfeld. Tuvo como gran referente a Jean Piaget, y postula que "*el individuo, tanto en los aspectos cognitivos y sociales del comportamiento como en los afectivos, no es un mero producto del ambiente ni un simple resultado de sus disposiciones internas, sino una construcción propia que se va produciendo día con día como resultado de la interacción entre esos dos factores*" [36].

Para esta corriente, el conocimiento de un alumno "*no es una copia fiel de la realidad, sino una construcción del ser humano"* [36], a partir de sus conocimientos previos. Esta construcción es un proceso activo del estudiante, realizado de forma dinámica y participativa [36].

Seymour Papert es un pionero de la inteligencia artificial, científico informático, matemático y educador. Fue uno de los más notables discípulos de Piaget, quien llegó a mencionar en una ocasión que nadie comprendía mejor sus ideas que Papert [38]. En

1967 crea Logo (como se menciona en la próxima sección) basado en sus estudios con Piaget. Lo definió no sólo como un lenguaje de programación, sino como una filosofía de educación. Guiado por esa idea, observó varios puntos donde la tecnología con robots ayudaba a los estudiantes, lo cual lo llevó a apoyar sus experiencias educativas con un robot "Tortuga" (figura 5) de su creación, siendo pionero también en este ámbito.

Figura 5: Robot tortuga de Papert.

La idea de Logo probó ser una forma efectiva de lograr que los estudiantes comprendan los conceptos de programación, además de que es una herramienta para acercar a las matemáticas a aquellos que no se sienten motivados por ella, ya que con la tortuga se realizan cálculos para definir el avance y los movimientos. También se destaca la sencillez para aquellas personas con capacidades diferentes. En base a esta filosofía llegó a desarrollar un plan de estudios que contenía dentro del programa grandes áreas como matemáticas, lenguaje y ciencia [2].

Dentro del mismo contexto, e impulsado fuertemente por los trabajos de Piaget y Papert, Alan Kay introdujo en 1968 el concepto de Dynabook. La idea de Kay fue hace un ordenador para niños de todas las edades, con el cual pudieran aprender y acercarse al mundo digital de manera interactiva. Si bien el avance de la tecnología de hardware estaba lejos aún en ese año para lograr plasmar con exactitud las ideas de Kay, el proyecto derivó en varios prototipos que tuvieron como hecho más destacado desde el punto de vista del software, el nacimiento del lenguaje de programación Smalltalk padre de los lenguajes y entornos educativos como Squeak. Hoy en día las ideas del proyecto Dynabook continúan vigentes y hemos heredado los conceptos de Smalltalk, los cuales se pueden ver en herramientas educativas de hoy. Alan Kay cabe destacar, que continuó sus

trabajos y hoy en día trabaja en el proyecto One Laptop Per Child (OLPC) de manera activa.

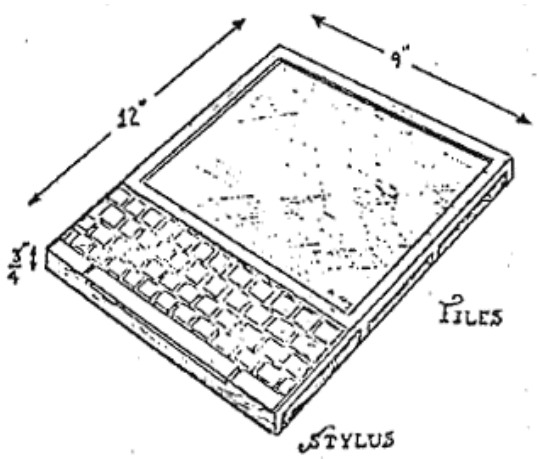


Figura 6: Ilustración de la Dynabook como fue descrita por Kay en su paper.

3.2 Lenguajes de Programación Visual

En los lenguajes de programación visual (LPV) más de una dimensión se utiliza para transmitir la semántica, donde tales dimensiones adicionales son adquiridas con el uso de objetos (un ejemplo son los bloques) y donde cada uno de estos objetos potencialmente significativos son símbolos, al igual que en los lenguajes de programación tradicionales cada palabra es un símbolo [6].

En sus inicios la programación visual tomó dos direcciones: una enfocada a los lenguajes de programación tradicionales (diagramas de flujo de ejecución) y otro alejado de los lenguajes de programación tradicionales que intentaba mostrar las acciones deseadas en pantalla para realizar la codificación. Estos primeros sistemas resultaron intuitivos pero al poco tiempo fueron considerados de poco porte e inadecuado para el trabajo "real", siendo delegados para el ejercicio académico.

Sin embargo, los LPVs poseen muchas características favorables: es más fácil tener una idea general de la estructura del programa, la percepción en dos dimensiones es más natural y eficiente que la lectura de texto, es más fácil de leer puesto que se reducen los elementos puramente sintácticos, la visión humana está optimizada para información multidimensional, etc. Gracias a esto siguieron surgiendo nuevos sistemas de esta índole, siendo uno de los más importantes por su influencia, el sistema llamado Logo que se

muestra en la figura 7; impulsado por Seymour Papert, quien propuso desarrollar una nueva forma de ver el uso de la computadora para la educación [7].

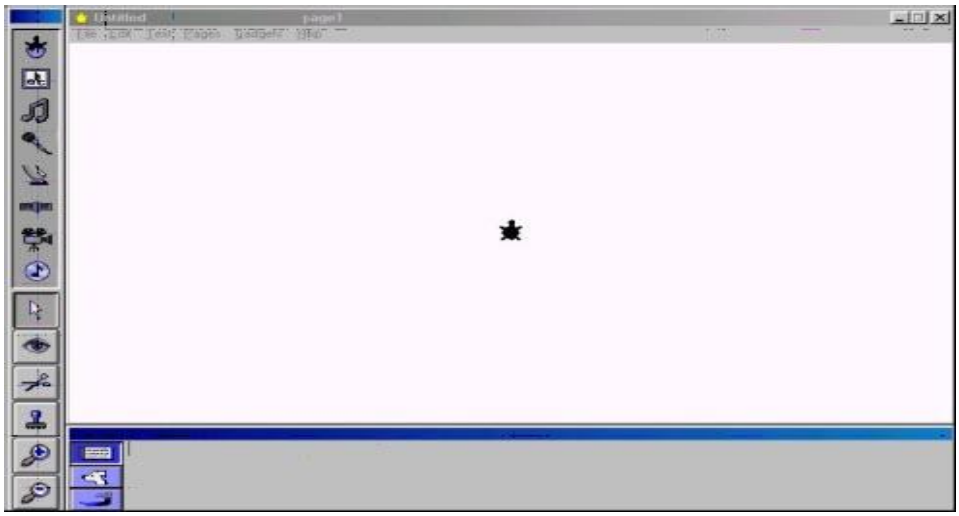


Figura 7: Interfaz gráfica del sistema Logo.

Este pionero en la inteligencia artificial, inventó el lenguaje Logo para fomentar el razonamiento y la lógica. Lograr un lenguaje accesible para niños era su meta, por lo que debía simplificar la definición de procedimientos para tareas sencillas, no numéricas, que fueran difíciles de implementar con lenguajes tradicionales. Luego de su lanzamiento en 1967, este lenguaje se convirtió en un movimiento para la computación en la primaria, y posteriormente dio lugar a la robótica educativa por medio de sistemas inspirados en Logo para programar comportamientos robóticos, teniendo como ejemplo los ampliamente utilizados robots hechos con piezas de Lego.

Así, el estudiante aprende a definir un problema y definir los pasos necesarios para resolverlo, lo cual pone al descubierto la presencia fuerte de los principios del constructivismo de Jean Piaget [23] aplicados a la enseñanza de informática y robótica. Fundamentalmente los niños enfrentan retos intelectuales que pueden ser resueltos mediante el desarrollo de programas en Logo. El proceso de revisión manual de los errores contribuye a que los niños desarrollen habilidades meta-cognitivas al poner en práctica el concepto de autocorrección. Esto se ve fuertemente beneficiado con el uso de *gráficos tortuga* [44], es decir, poder dar instrucciones a una tortuga virtual (ver Figura 7) o cursor usado para crear dibujos mediante palabras que representan instrucciones.

Citamos la siguiente frase de Papert que resulta apropiada [5]: “Al enseñarle a pensar al ordenador, los chicos se embarcan en una exploración del modo en que ellos mismos piensan”. Actualmente, existen variados LPVs desarrollados con fines educativos los cuales persiguen los objetivos planteados para el desarrollo de este IDE, abriendo

posibilidades para lograr una interfaz gráfica con un LPV basado en bloques de tal manera que conserve y posea las ventajas pedagógicas que plantea dicho paradigma.

Capítulo 4

Experiencias anteriores

4.1 Experiencias regionales e internacionales

En el contexto de la robótica educativa se intenta profundizar en los conceptos de la física, matemática, dibujo y programación, generando una construcción propia del conocimiento por parte de los alumnos que integran las distintas disciplinas en una síntesis particular para el desarrollo de los proyectos. Así se pone al alcance de los alumnos las herramientas necesarias para que desarrollen dispositivos físicos, externos a la computadora, que serán controlados por ésta [32].

Desde 1975, en la Universidad Du Maine, en Le Mans, Francia, aparece una primera utilización con fines educativos de la robótica, con el desarrollo de un sistema de control automatizado para la administración de experiencias en el campo de la psicología. De estas investigaciones surge el concepto de encargado-robot, siendo el mismo un sistema que tiene por objetivo hacer trivial la percepción de experiencias de laboratorio en el dominio de la psicología experimental. A través de este sistema, el alumno puede plantearse múltiples preguntas sobre el campo de estudio, establecer estrategias para responder a cada una de las preguntas, experimentar e interpretar los resultados visualizados en la pantalla del computador. Luego en 1990, Martial Vivet del Laboratorio de Informática de la misma universidad, define la *microrobótica pedagógica* como "una actividad de concepción, creación, puesta en práctica, con fines pedagógicos, de objetivos técnicos físicos que son reducciones bastante fiables y significativas de procedimientos y herramientas realmente utilizadas en la vida cotidiana, particularmente en el medio industrial". Es a partir de estas definiciones, que se han realizado muchas investigaciones y trabajos que pretenden contribuir al desarrollo de un marco teórico y conceptual en la educación para la robótica pedagógica, así como para la construcción de entornos de aprendizaje en distintos medios y niveles (págs. 112, 113 del libro "La educatrónica: innovación en el aprendizaje de las ciencias y la tecnología" [37]).

Numerosas investigaciones ya demuestran el interés global por la inserción de herramientas robóticas en las aulas de clase. Siendo implementadas diversas experiencias robóticas cuyos énfasis son muy variados, como por ejemplo: en 1998 se inició el proyecto "Robótica y Aprendizaje por Diseño", realizado conjuntamente por el Centro de

Innovación Educativa de la Fundación Omar Dengo y el Ministerio de Educación Pública de Costa Rica. Se pretende concretar ciertos desempeños y habilidades relacionadas al diseño tecnológico, brindando cursos y capacitación tanto a alumnos como docentes de escuelas públicas.

En 1999 comienza la sucesiva competencia FIRST LEGO League robot competition, que da la posibilidad a estudiantes de construir robots que compiten según sus habilidades o capacidades de movilidad y discriminación de datos. En otras palabras, los estudiantes entre 9 y 14 años de edad buscan soluciones a los distintos problemas que se les fueron asignados y exponen su investigación y su proyecto en concursos regionales, los cuales hoy en día son llevados a cabo en todo el mundo; en 1999 también surge el proyecto NASA Robotics Education Project (figura 8) que intenta otorgar recursos de robótica a ciertas instituciones para que puedan ser manipulados por los alumnos en los salones de clase con la finalidad de crear modelos o construir prototipos que demuestren leyes o comportamientos físicos.

Figura 8: La NASA hizo varias simplificaciones de robots construidos para que sigan fines didácticos y sirvan de motivación académica.

A partir del nuevo milenio se expande fuertemente la robótica educativa impulsada principalmente por el proyecto mundial OLPC, impulsado por una organización sin ánimos de lucro creada por catedráticos del Laboratorio de Multimedia del MIT para diseñar, fabricar y distribuir una laptop por niño. Y a raíz de estas experiencias, hoy en día existe una cantidad creciente de proyectos relacionados a la robótica educativa en Latinoamérica, incluyendo países como México, Argentina, Chile, Venezuela.

4.2 Ámbito Local

En el 2007 en el ámbito local, a partir del Plan Ceibal creado *"con el fin de realizar estudios, evaluaciones y acciones, necesarios para proporcionar un computador portátil a cada niño en edad escolar y a cada maestro de la escuela pública, así como también capacitar a los docentes en el uso de dicha herramienta, y promover la elaboración de propuestas educativas acordes con las mismas"* [10] y con el complemento en 2009 del proyecto Butiá (detallado posteriormente) que intenta ampliar las capacidades sensoriales y de actuación de la computadora portátil del Plan Ceibal, transformándola en una plataforma robótica móvil [34], se tienen los primeros indicios de robótica educativa a nivel masivo. Además en el 2010, con la aparición del plugin Butiá para TortuBots comienza a manejarse el concepto de un IDE para el control del robot Butiá. TortugArte es un programa inspirado por Logo que persigue sus mismos fines pero posee interfaces gráficas modernas, continuando los lineamientos de Sugar, con este se pone al alcance de niños conceptos de programación, mediante una interfaz gráfica icónica, donde cada instrucción se mapea como un bloque. El proyecto Butiá realizó modificaciones sobre TortugArte agregando algunos plugins en forma de paletas que permiten controlar diferentes kits robóticos. A este plugin agregado a TortugArte se lo denominó TortuBots [41][43]. Con este complemento se permite a los alumnos programar desde dicho entorno de desarrollo el comportamiento del robot Butiá. En la figura 9 se muestra un ejemplo de la paleta del plugin.

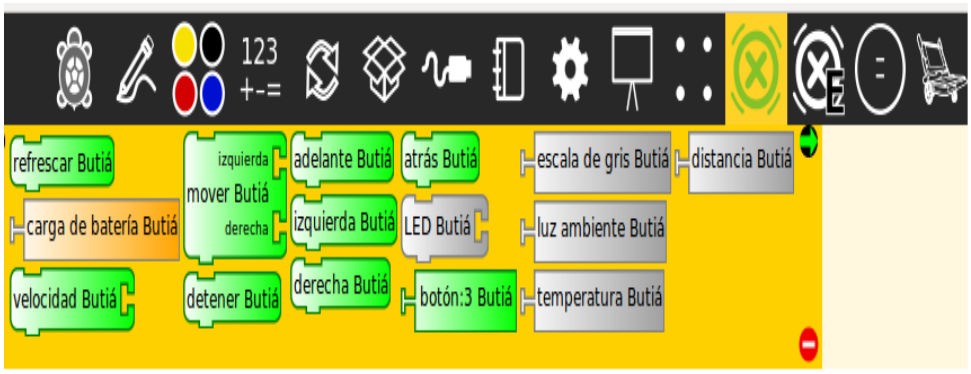


Figura 9: Interfaz gráfica del sistema TortuBots (plugin Butiá).

Vinculando concretamente las experiencias anteriores con los fines de este proyecto; desarrollar un IDE para el sistema operativo Android que pueda manejar comportamientos del robot Butiá, se debe detallar el proyecto de grado introducido al comienzo de este documento. Este fue desarrollado entre el 2011 y 2012, sobre la versión de escritorio de Sugar con el objetivo de ser ejecutado en una computadora portátil XO. Este entorno de desarrollo de programación, orientado por arquitecturas robóticas basadas en comportamientos, pretende actuar como herramienta para crear programas que definan

la forma de actuar del robot Butiá siguiendo una determinada estructura [1]. Este proyecto intenta mejorar las prestaciones de TortuBots, quien presenta como limitaciones la baja visibilidad global del código en las aplicaciones de mediano o gran porte y además no está orientado a programación de comportamientos robóticos. Proponiendo como solución, un IDE orientado a comportamientos usando como base el lenguaje de programación visual Etoys, estructurado por una arquitectura Subsumption, y una comunicación con Bobot (detallado en la próxima sección) por medio de sockets TCP/IP.

Por otro lado, y entrando en aspectos más específicos, es importante mencionar la experimentación realizada en el 2013 por los creadores del robot Butiá en la cual se agregó una SBC Raspberry Pi al kit original de este robot, donde fue configurado Sugar para poder ejecutar TortuBots. Luego, desde una Tablet por medio de VNC [52] se logró ejecutar el plugin Butiá ofrecido por TortuBots y así controlar el robot Butiá desde una Tablet [42]. Sin embargo, dado los problemas de performance que implica el graphical desktop sharing, se percibió que la calidad de la experiencia fue muy distinta que la original desde un ordenador de escritorio. Por un lado la velocidad de respuesta no es la misma. Por el otro, la calidad de la imagen renderizada es usualmente peor que la original y tiene como problema final que tanto la interfaz gráfica como el software en general están pensados para su uso en un ordenador y no en un dispositivo móvil, dejando de lado un montón de aspectos conocidos que mejoran potencialmente la experiencia en un dispositivo móvil cuyas características de manejo y de tamaño de pantalla, distan mucho de las de un ordenador de escritorio. Estas cuestiones son uno más de los motivos que impulsan el desarrollo de este proyecto.

Capítulo 5

Proyecto Butiá y Plataforma Robótica

5.1 Acerca del Proyecto

El proyecto Butiá fue lanzado en el año 2009 con el objetivo de crear una plataforma simple, la cual ponga al alcance de estudiantes escolares o liceales las herramientas necesarias para permitirles interiorizarse con la programación de comportamientos para robots. Actualmente la implementación 2.0 del proyecto está siendo utilizado en formato de kit, distribuido a más de 100 centros educativos del Uruguay, con un set de sensores y piezas que permiten cambiar la ubicación de los sensores en la plataforma móvil donde se coloca la computadora XO [36].

En el año 2012 se puso en marcha con el respaldo de Antel el trabajo de la segunda versión de la plataforma robótica Butiá, mediante la incorporación de tecnología nacional, buscando reducir costos, lograr compatibilidad mecánica y electrónica con otros kits robóticos comerciales así como brindar soporte a nuevos lenguajes de programación y expandir su uso en aplicaciones asociadas a las telecomunicaciones.

Figura 10: Proyecto Butiá en Colonia del Sacramento, Junio 2013.

5.2 Hardware

El robot Butiá (figura 11) está basado en una plataforma de acrílico, la cual hace de estructura y soporte a una serie de sensores y actuadores controlados por una interfaz de entrada/salida.

Si bien posee soporte para una variedad de controladoras distintas, en general se utilizó la Arduino Mega en las primeras versiones del proyecto, y actualmente la USB4Butiá [49], adaptación para el Butiá de la placa USB4All [48]. Esta placa posee 6 puertos RJ45 (Ethernet) con soporte Plug&Play y Hotplug y un bus para motores AX-12 y de corriente continua.

El proyecto actualmente no utiliza una SBC (Single Board Computer) para el robot y por lo tanto Butiá no posee un sistema operativo de propósito general propio. Además USB4Butiá actualmente no reconoce dispositivos USB ni posee conectores para hacerlo, lo cual impide una comunicación a través de Wi-Fi o Bluetooth ya que no se pueden agregar dispositivos para soportarlo (por el lado del software, el firmware tampoco soporta esto). La única comunicación que permite hoy es serial. Más adelante veremos que lo último nos constituye un problema, y analizaremos las posibles alternativas para manejar dicho problema.

El Robot posee sensores de: luz, grises, contacto (botón) y distancia. Por el lado de los actuadores el Butiá posee leds y soporta la inclusión de motores Dynamixel AX-12, aunque en versiones recientes del Butiá se han sustituido a estos por motores de corriente continua los cuales son más económicos. Hoy en día el robot está orientado a trabajar con una XO, la cual funciona como “cerebro” de este para las tareas programadas por los alumnos, comunicándose con USB4Butiá para ordenarle distintas acciones a los actuadores o pedir distintos valores a los sensores según se haya programado en la XO.

5.3 Software

Dado que Butiá no maneja una SBC esta no posee un sistema operativo. El único software que forma parte íntegra de Butiá es el firmware de la placa USB4Butiá. Este se encarga de resolver la interacción con los sensores y actuadores, exponiendo una interfaz que permite controlarlos de manera sencilla a través del uso de un protocolo llamado User Protocol.

Cada sensor/actuador es modularizado como una entidad y su lógica está encapsulada en un módulo llamado User Module que se encarga de resolver la comunicación a bajo nivel con el sensor/actuador. Existen programas creados por los docentes del MINA que encapsulan a alto nivel las funcionalidades de la USB4Butiá.

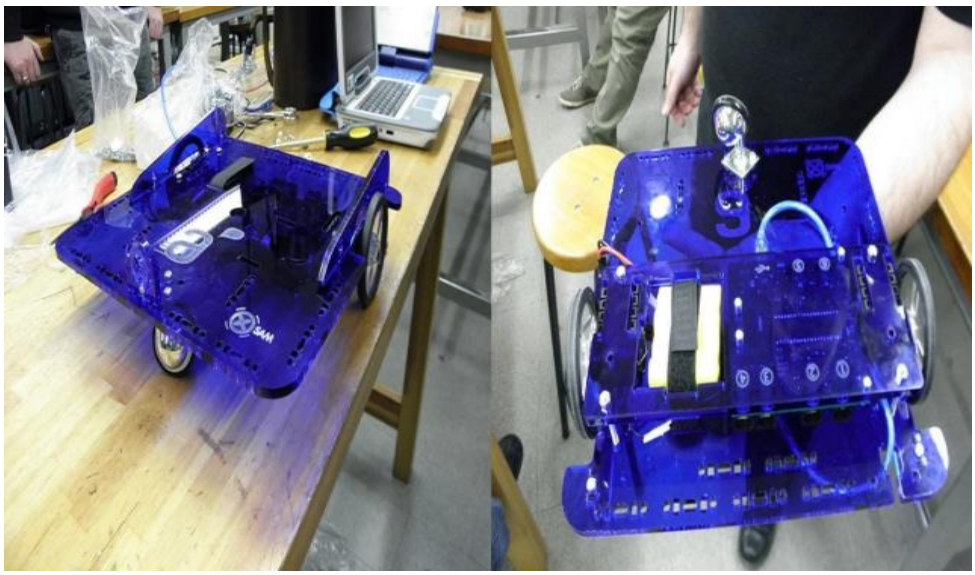


Figura 11: Robot Butiá 2.0.

5.4 Bobot

Para utilizar el Butiá desde un computador, se creó una aplicación demonio, Bobot Server [2], que recibe funciones en formato de texto plano a través del puerto 2009, o a través de una interfaz web. Este es el encargado de realizar la comunicación a bajo nivel con la placa USB4Butiá y de exponer una interfaz a través de una conexión TCP/IP. A través de ella se pueden listar las funcionalidades ofrecidas por la interfaz e invocar cada una de ellas. De esta forma se independiza a las aplicaciones que utilizan la placa, de la implementación de la misma.

Los programas que controlan al robot pueden estar entonces implementados en cualquier lenguaje y plataforma de desarrollo capaz de interactuar a través de una conexión TCP/IP. A continuación se describe los comandos proporcionados por Bobot para interactuar con la USB4Butiá:

1. LIST
   - Lista los módulos actualmente disponibles en la placa USB4Butiá.
2. DESCRIBE *moduleName*
   - Describe las operaciones implementadas por un componente específico.

3. OPEN *moduleName*
   - Abre un cierto componente para ejecutar sus operaciones.

4. CALL *moduleName operation param1, param2, ...,paramN*
   - Llama a una operación de un módulo pasándole una serie de parámetros.

5. CLOSE *moduleName*
   - Cierra el módulo pasado como parámetro

6. CLOSEALL
   - Cierra la conexión sin devolver ningún parámetro.

5.5 Complementos

Para complementar las funcionalidades del robot Butiá se trabaja con distintas componentes de hardware entre las que se encuentra la Raspberry Pi [15], o en primera instancia hacer uso de esta placa de forma experimental para proyectos como este. Esta es una SBC desarrollada por Raspberry Pi Foundation, organización sin ánimos de lucro iniciada en el año 2008, la cual da origen a la Raspberry Pi como ordenador de bajo coste para facilitar la enseñanza de la informática en los centros educativos.

En el 2012 comenzó a fabricarse con colaboraciones del laboratorio de informática de la Universidad de Cambridge y de Boardcom, siendo diseñada con fin de ser lo más barato posible y llegar a la mayor cantidad de usuarios. Desde el punto de vista del Hardware la placa Raspberry Pi cuenta con unas dimensiones de 8.5 por 3.5 cm, donde se ubica un system on a chip Boardcom BCM2835, que contiene un procesador ARM11 con varias frecuencias de funcionamiento y la posibilidad de realizar overclocking hasta 1 GHz sin perder la garantía, un procesador gráfico VideoCore IV capaz de obtener vídeo a 1080p y audio de alta calidad a través de su conector HDMI, y memoria RAM entre 256 y 512 MB. La misma puede ser dotada con una sistema operativo y programas por medio de una tarjeta SD con por lo menos 1 GB y clase 4 (velocidad de lectura/escritura) o

mejor. Posee una conexión Ethernet 10/100 y permite conexión Wi-Fi por medio de una interfaz USB Wi-Fi gracias a dos puertos USB incluidos. En la figura 12 se ve un diagrama del hardware.

Contando con esta placa en el kit del robot Butiá se tiene la posibilidad de establecer una comunicación inalámbrica con el robot, e incluso la posibilidad de levantar un servidor Web en la placa que provea los servicios para controlar los comportamientos del robot Butiá.

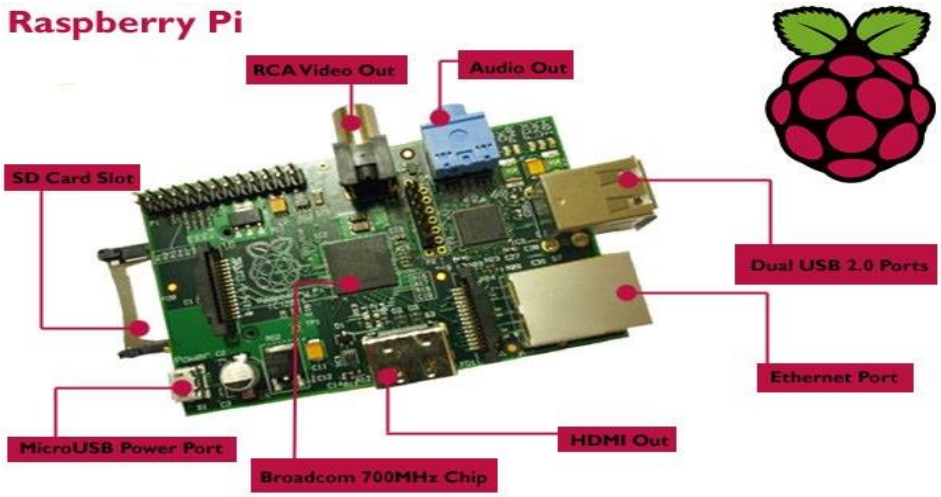


Figura 12: Placa Raspberry Pi [14].

Capítulo 6

Aspectos básicos de arquitectura

6.1 Introducción

Introduciremos a grandes rasgos en esta sección, para luego ampliar a lo largo del documento, las dos principales arquitecturas de solución que hemos investigado y analizado. En próximas secciones se analizarán el estado actual de distintas tecnologías vinculadas a las distintas problemáticas y caminos a recorrer que implica implícitamente cada uno de estos dos enfoques de arquitectura, mencionando sus virtudes y limitaciones y comparándolas constantemente con los objetivos y requerimientos básicos planteados para este proyecto. Por lo tanto, es de importancia introducir aquí aspectos muy generales que se tomarán en cuenta en próximas secciones.

Nuestro sistema debe ser capaz de comunicar los dispositivos Android con el robot Butiá, más específicamente, debe ser capaz de comunicarse con la controladora USB4Butiá, para poder así ser capaces (dentro del contexto de nuestra aplicación) de controlar los sensores y actuadores del robot Butiá para que efectúen las diversas tareas programadas por el usuario.

6.2 Arquitecturas

Una arquitectura posible a considerar es aquella en donde la aplicación se concentra enteramente en el dispositivo Android, y se comunica éste con la controladora USB4Butiá vía USB (vimos en la sección “Proyecto Butiá y Plataforma Robótica” que la controladora sólo permite comunicación USB). Dada las características de la controladora, en este tipo de solución el dispositivo Android será el responsable no sólo de la parte gráfica y visual, sino también de manejar los distintos comportamientos programados por el usuario y asegurarse de la ejecución correcta de estos.

Otra solución viable es no tener solo el dispositivo Android y la controladora USB4Butiá como encargados de la comunicación del primero al Robot, sino agregar un tercer dispositivo que intermedie en la comunicación. Este tercer dispositivo puede hacer

de intermediario y comunicarse tanto con la USB4Butiá vía USB como con el dispositivo Android mediante otras vías que no le son posibles a la controladora, como por ejemplo Wi-Fi o Bluetooth. El tercer dispositivo podría ser por ejemplo una SBC con un dispositivo Dongle Wi-Fi o con un dispositivo Bluetooth. Tomaremos a futuro en cuenta la Raspberry Pi descrita en la sección "Proyecto Butiá y Plataforma Robótica" como la SBC que implementa esta solución, debido a su reducido costo y otras ventajas ya mencionadas. Observemos ahora que de esta arquitectura se desprenden dos posibles distribuciones del sistema: en la primera concentramos sólo la resolución de cierta parte de la aplicación al dispositivo Android (como ser la Interfaz Gráfica) delegando el resto de la tarea para que sea resuelta por la SBC (por ejemplo el asegurarse de la ejecución en forma correcta y ordenada de las tareas programadas por el usuario). En la segunda mantenemos una estructura de solución similar a la arquitectura del párrafo anterior, concentrando toda la aplicación en el dispositivo Android, por lo cual la SBC solamente tendría la tarea de actuar como puente de comunicación entre la USB4Butiá y el dispositivo Android.

Una vez explicado lo anterior, es interesante observar que poseer una SBC dentro de la arquitectura de nuestra solución abre las puertas a la utilización de otras tecnologías, como por ejemplo realizar un enfoque Web y utilizar HTML5 para lograr la compatibilidad con el celular pero ampliando la compatibilidad de la solución a otros clientes que no manejen Android como dispositivos iOS, Windows Phone, etc. También permite un manejo más eficiente y menos procesamiento del lado de los dispositivos móviles, ya que le quita procesamiento de la concurrencia entre los comportamientos creados por el usuario.

Capítulo 7

HTML5 y bibliotecas útiles

7.1 Ventajas de HTML5

La iniciativa de realizar el IDE para el Butiá en la Web surge de los beneficios que este sistema engloba: potencia la portabilidad en cualquier ordenador, Tablet o dispositivo móvil, otorga independencia del sistema operativo, disminuye la cantidad de procesamiento que debe hacer los dispositivos etc. En contrapartida limita a desarrollar el IDE en un lenguaje soportado por los navegadores web y presenta problemas de compatibilidades para diversos dispositivos, es aquí donde entra en juego HTML5.

HTML5 recoge las ventajas que introdujo XHTML y elimina muchas de las restricciones y limitaciones que tenían las versiones anteriores. Se destaca, además del código simplificado y sencillo, la facilidad para desarrollar aplicaciones que sean utilizables en dispositivos móviles tanto como en ordenadores, ya que se da soporte a los navegadores de teléfonos Smartphone y Tablets, agregando por ejemplo nuevas funcionalidades como el arrastre de objetos/imágenes (drag & drop) las que podrían ser fundamentales para el desarrollo del IDE.

Si bien no obviamos la importancia de conceptos interesantes que introduce HTML5 como nuevos tags y corrección de errores, inclusión del DOM en el estándar (antes era a criterio de los browsers), o contenido multimedia, vale la pena mencionar con mayor énfasis alguna de las nuevas APIs que el mismo incorpora. Una de ellas es la interfaz para WebSockets, la cual habilita la comunicación bidireccional a través de una conexión persistente, permitiendo incrementar la interactividad entre cliente y servidor o múltiples clientes (y/o dispositivos), por lo tanto permite conexiones “cross-device” en tiempo real. Otra ventaja interesante es la inclusión de Web Storage. Esta introduce una nueva forma de persistencia client-side, que originalmente solo era posible a través de cookies. Web Storage permite almacenar información en el browser sin fecha de caducidad para dichos datos y ofrece un tamaño máximo mucho mayor que las cookies, siendo este de hasta 5mb. Existen dos formas de Web Storage: una es localStorage en donde la información no caduca; la otra es sessionStorage en donde la información se pierde al cerrar la ventana o tab. La forma de utilizarlo es mediante un objeto string en donde se puede almacenar información en formato JSON.

7.2 Bibliotecas

Además del potencial que surge del estándar HTML5, vale la pena mencionar tres bibliotecas de JavaScript que a partir de las investigaciones realizadas podrían facilitar la portabilidad del IDE en dispositivos móviles y Tablets.

Las primeras dos potencian su utilidad si son usadas en forma complementaria, por un lado la biblioteca Modernizr permite detectar la disponibilidad de ciertas características que se derivan de las especificaciones HTML5 y CSS3. Tradicionalmente se utilizaba la propiedad UserAgent para detectar las propiedades del navegador (UA sniffing), pero Modernizr realiza detección de características para discriminar que puede o no hacer el dispositivo que renderiza la página. En particular posee una opción muy interesante de consulta por JavaScript que retorna en base a CSS Media Queries la cantidad de pixeles de alto y ancho en que será renderizada la página web. Por otro lado, la biblioteca YepNope funciona como un cargador condicional permitiendo cargar a demanda los Scripts o CSS requeridos. Entonces, haciendo uso de ambas bibliotecas se puede con comodidad contemplar la variedad de dispositivos que el IDE para Butiá requiere soportar. Para ejemplificar cómo pueden ser usadas se mostrará el código que permite identificar con Modernizr la resolución del display y con YepNope actuar adecuadamente para que la página web consuma el css adecuado:

```
<script type="text/javascript">
        Modernizr.load({
                test: Modernizr.mq( "only all and (max-width: 340px)" ),
                yep: ["stylesheets/m.blocks.css"],
                nope: ["stylesheets/blocks.css"]
        });
</script>
```

Este código es sencillo, YepNope provee las propiedades “test” para testear una condición booleana, “yep” específica lo que será cargado en caso que la condición se cumpla, y “nope” lo que será cargado en caso negativo. Además, existen más propiedades opcionales como “callback” para realizar acciones luego del cargado, entre otras. Modernizr pretende poner fin a la práctica de UA sniffing presentando múltiples tests (más de 40) para características de navegadores de última generación creando un objeto JavaScript “Modernizr” que contiene los resultados booleanos de estas pruebas [30].

Por último, cabe nombrar JQuery Mobile como una biblioteca JavaScript o también conocida como un Mobile Framework dependiente de JQuery enfocada en crear un

marco compatible con la amplia y heterogénea gama de Smartphone y Tablets que hoy en día se venden en el mercado, permitiendo desarrollar aplicaciones Web móviles genéricas. JQuery Mobile usa etiquetas HTML con atributos definidos por el Framework que al momento de mostrar la página, son leídos por JQuery y tomado como metadatos para crear la interfaz de usuario. Es decir, JQuery Mobile propone una dinámica donde usando la información que se utiliza en las etiquetas crea HTML dinámicamente para desplegar la interfaz. Para representar esto con ejemplos se puede mencionar la capacidad de este Framework para crear en un mismo archivo varias páginas agregando dentro del body varios DIVs con data-role=“page” y con identificadores diferenciando cada página. Permitiendo la creación de links entre las páginas por medio de los identificadores de la siguiente manera: <a href=“#idpagina”>. Como esta existen otras capacidades que permiten considerar a JQuery Mobile como un nivelador entre HTML5, CSS3 y JavaScript para los distintos dispositivos móviles.

Capítulo 8

Análisis y estudio de soluciones viables

8.1 Identificación de problemas

Hemos introducido las características principales del robot Butiá y el ambiente donde se realizará el trabajo, permitiendo comenzar el análisis de las distintas opciones y soluciones a los grandes problemas a atacar que tiene nuestro sistema. Esto nos permite establecer un marco de referencia frente a la siguiente etapa de este proyecto, que tiene como objetivo estudiar y analizar el estado de tecnologías pertinentes al proyecto, incluso intentar mitigar los riesgos de manera previa a la implementación, mediante un profundo análisis de las herramientas y tecnologías disponibles. De esta manera, se realizará un balance sobre las ventajas y desventajas de dichas herramientas y tecnologías para orientar el rumbo del proyecto hacia una solución viable que cumpla los objetivos propuestos. Es necesario a modo de establecer un marco de referencia, definir los principales problemas implícitos al sistema que deben ser atacados.

8.2 Comunicación y arquitectura

Dentro del campo de la robótica es frecuente la utilización de interfaces de control (llamados también controladoras) las cuales reúnen todos los sistemas de conversión y acondicionamiento que necesita un ordenador para actuar como “cerebro” del robot. En el marco del proyecto Butiá esta tarea es realizada por la placa USB4Butiá [49], la cual concentra y controla actuadores y sensores del robot. Para lograr interactuar con actuadores y sensores (y por ende el robot), es un factor determinante a resolver el medio para lograr la comunicación entre la controladora USB4Butiá y los dispositivos Android a los cuales apunta este proyecto.

Por lo tanto identificamos como uno de los problemas más grandes a resolver, la comunicación entre los dispositivos Android y la interfaz de control. Notemos que este problema implica también resolver la arquitectura de nuestro sistema, a nivel tanto de hardware como de software. Lo primero se desprende del hecho de que para lograr la comunicación entre USB4Butiá y el dispositivo Android podrían o no haber dispositivos

intermediarios que resuelvan ese problema en particular, así como también nuevos elementos de Hardware que expandan el kit que actualmente posee el robot Butiá. Lo segundo se deduce de que el software que permita generar comportamientos desde un dispositivo Android podría estar auto contenido en el dispositivo o estar distribuido entre este y otro elemento intermediario de Hardware (por ejemplo un placa SBC).

8.3 Interfaz gráfica e interacción con el usuario

Dentro de la robótica educativa, uno de los desafíos más importantes es lograr, para el control del robot, una interfaz gráfica amigable y clara que permita de manera intuitiva lograr códigos de programación. La psicología educativa puede ayudar a definir las características deseables de un sistema de educación asistida por computadora y fortalecer el éxito de la herramienta creada en este proyecto.

Por las razones que presentamos en secciones anteriores, se debe orientar la interfaz gráfica hacia un lenguaje de programación visual basado en bloques, de tal manera que conserve y posea las ventajas pedagógicas que plantea dicho paradigma. La interfaz es el verdadero determinante del éxito de este tipo de herramientas pedagógicas: si esta no cumple con las características anteriormente mencionadas, la experiencia educativa puede fracasar en su objetivo. Es por esto que destacamos la Interfaz Gráfica y la interacción con el usuario como uno de los problemas grandes a solucionar.

Veremos en próximas secciones que para distintos tipos de tecnologías y arquitecturas que presentará este grupo como solución potencial, existen distintas herramientas que cumplen con los requerimientos preestablecidos.

8.4 Concurrencia y orientación a comportamientos

Este trabajo tiene como objetivo estar orientado a la programación robótica educativa basada en comportamientos. Para los distintos comportamientos del Robot que sean programados en este IDE el sistema debe resolver los problemas que contemplan las arquitecturas Robóticas. Si se sigue la arquitectura basada en prioridades estas tareas programadas deben tener una determinada prioridad y su ejecución debe ser acorde y en coherencia a esas prioridades establecidas. Dicha prioridad será establecida programáticamente en el propio IDE. Si se sigue la arquitectura Subsumption, debe

implementarse la resolución de conflictos de ejecución entre comportamientos de distintas capas.

Se deduce de lo anterior que las tareas deben ser capaces de interrumpirse entre sí, es decir que si se dan las condiciones para que se ejecute una tarea de mayor prioridad, la de menor prioridad debe postergar su ejecución y dar paso a la de mayor prioridad. Similar para el caso de subsumption. El manejo de la concurrencia o resolución de conflictos de activación de múltiples comportamientos también identificamos como uno de los problemas importantes a atacar, ya que no solo es de una complejidad considerable, sino que además contempla uno de los objetivos principales de este proyecto.

Capítulo 9

Estudio de comunicación con USB4Butiá

9.1 Introducción

Al haber introducido las componentes del Robot Butiá es importante detenerse en el dispositivo más importante: la USB4Butiá, que funciona como controladora de E/S que permite manejar y exponer en alto nivel tanto los servicios de sensores como de actuadores del robot. De lo anterior se deduce que comunicar el dispositivo Android con la USB4Butiá es de vital importancia, ya que constituye la opción más viable de interactuar con sensores y actuadores del Butiá.

Actualmente la única forma de establecer una comunicación con la USB4Butiá basada en el protocolo User Protocol es a través de USB. Citamos a continuación algo relevante a esto, de la sección "Hardware" de "Proyecto Butiá y Plataforma Robótica" de este trabajo: "*USB4Butiá actualmente no reconoce dispositivos USB ni posee conectores para hacerlo, lo cual impide una comunicación a través de Wi-Fi o Bluetooth ya que resulta extremadamente complejo agregar dispositivos para soportarlo (por el lado del software, el firmware tampoco soporta esto).*"

Entonces, como punto de partida se puede inferir que la comunicación con USB4Butiá de manera serial a través de USB, es una posibilidad para solucionar este desafío. Siendo el USB Hosting la propuesta más prometedora que se planteará para resolver la comunicación sin dispositivos intermedios.

9.2 Conexión USB y USB Hosting

El Universal Serial Bus (USB) es un estándar de conexión del tipo serial desarrollado en los años '90. La especificación define los atributos del bus, la definición del protocolo, los tipos de transacciones, la administración del bus, el hardware y la interfaz de programación requerida para diseñar y construir sistemas y periféricos que cumplan con el estándar [45]. Este dispone de cuatro líneas, una para datos, una para corriente y otra de toma de tierra [28].

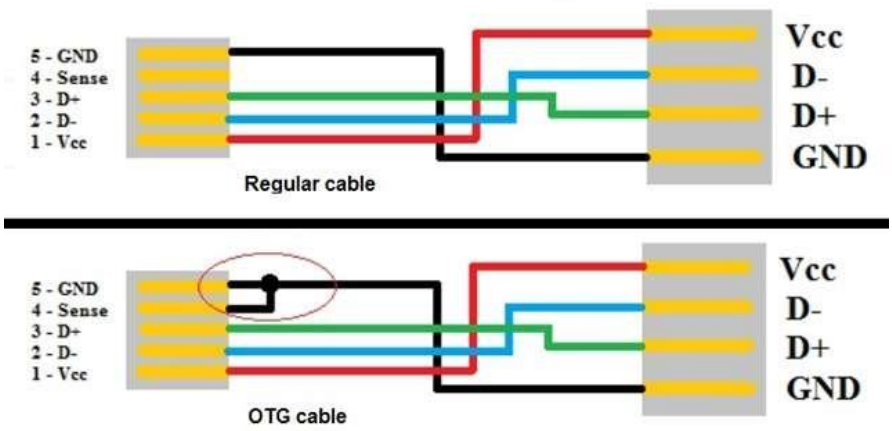


Figura 13: Cable USB OTG y USB regular.

El USB On-The-Go conocido también por USB OTG, es una extensión de la norma USB 2.0 que permite a los dispositivos USB tener una mayor flexibilidad en la gestión de la conexión USB. Permite que dispositivos, actúen como host, lo cual nos permite establecer la conexión con la USB4Butiá [22]. Como se observa en la figura 13, los periféricos compatibles OTG disponen de un conector de tipo mini AB (o micro AB), es decir, capaz de aceptar un conector A (maestro) o B (esclavo). Gracias a que no es obligatorio que ambos dispositivos sean compatibles con OTG para comunicarse entre ellos, podemos utilizar este medio para conectar punto a punto el robot con el dispositivo Android, siendo este último quien actúa como maestro en la conexión.

Cuando un dispositivo Android se encuentra en modo USB Host, actúa precisamente como el host o master (maestro) de la conexión, alimentando el Bus e identificando a los dispositivos que estén conectados a él. La idea principal de esto es mediante USB Hosting conectar dispositivos USB a dispositivos Android (como se muestra en la figura 14) y lograr que estos actúen como maestros de la conexión así como alimentar la placa. Esto será de interés ya que nos permitirá hablar con la USB4Butiá de manera directa usando User Protocol a través de USB.

Figura 14: Pendrive conectado a una Tablet mediante un cable USB OTG.

9.3 Soporte de Android a USB Host

No todas las versiones del sistema operativo Android soportan el modo de USB Host. Esta es una funcionalidad nueva de Android agregada a partir de la versión 3.1 (Honeycomb) [46]. Este feature permite conectar dispositivos USB al dispositivo Android y que este lo reconozca. El sistema operativo ya incorpora ciertas funcionalidades básicas que permite utilizar los dispositivos más comunes de USB como ser los de almacenamiento de datos. Sin embargo lo anteriormente mencionado no es suficiente para los objetivos perseguidos, ya que una cosa es que Android de soporte para USB Host y que tenga funcionalidades básicas para dispositivos comunes, y otra muy distinta es que aplicaciones desarrolladas para Android y no incluidas en su Kernel (llamadas usualmente 3rd party Apps) tengan acceso a comunicarse con el dispositivo USB que estamos hosteando.

Para realmente poder lograr que una aplicación desarrollada para Android pueda comunicarse con el dispositivo USB conectado se debe utilizar la API de USB Host incluida en el package **android.hardware.usb** del Android SDK. Esta API nos otorga funcionalidades básicas para lograr la comunicación con el dispositivo USB y está disponible a partir de la API level 12 que coincide con la versiones de Android 3.1 en adelante [46].

Con los conceptos manejados hasta el momento, se puede deducir que para poder lograr una conexión con la USB4Butiá y poder lograr la comunicación del dispositivo Android al Robot Butiá sin dispositivos intermedios se necesita obligatoriamente (a menos que se actualice manualmente módulos del Kernel) una versión de Android igual o posterior a la 3.1. Esto ya determina un problema de compatibilidad no despreciable, reduciendo la cantidad de usuarios finales con posibilidades de usar el IDE en cuestión a sólo aquellos que cumplan dicha condición, lo cual va en contra de un objetivo secundario de este proyecto que es lograr compatibilidad con la mayor cantidad de dispositivos posible y en contra de lo estipulado en la presentación de este proyecto que plantea lograr compatibilidad con Android 2.3 en adelante.

Sin embargo el problema de compatibilidad no termina ahí. Se investigó de distintas fuentes y se comprobó empíricamente que incluso el hecho de que un dispositivo posea una versión de Android igual o posterior a 3.1 no garantiza que este posea efectivamente la API de USB Host que permitiría el manejo de USB Host por aplicaciones externas al kernel. Uno de los motivos por lo que se da esto, es que las versiones de Android distribuidas en los distintos dispositivos electrónicos (ej. celulares, Tablets, entre otros) suelen no ser las originales de Android, sino que son versiones customizadas por los fabricantes de los dispositivos electrónicos. Estas versiones personalizadas incluyen menos funcionalidades en el kernel, haciéndolo más liviano. En general se da la casualidad de que una de las funcionalidades que a veces se quita en estas versiones "custom" es, lamentablemente para los intereses de este proyecto, la API de USB Host.

9.3.1 Prototipo de prueba de disponibilidad de la API USB Host

Para probar lo anterior se desarrolló un prototipo en java usando Android SDK que listara los dispositivos conectados al cable USB OTG, solicitando primero los permisos especiales necesarios al usuario (de que requerimos el acceso al recurso de USB Host) y retornando un mensaje de error controlado si no se lograba acceso a los dispositivos. Cabe comentar que los emuladores de Android que fueron utilizados como el de Eclipse no permiten emular la conexión de un dispositivo USB, por lo cual se hicieron pruebas en dispositivos reales.

Se pudo probar la aplicación en una Tablet Ledstar con Android 4.0.4, donde los resultados al conectar un pendrive por cable OTG y correr la aplicación fueron que el programa retorno un mensaje advirtiendo que no se podía acceder ni detectar los dispositivos conectados.

A partir de esto, se decidió utilizar como comprobación de lo anterior y como información adicional una aplicación gratuita llamada USB Host Diagnostics descargada desde el store de Google Play. Esta corre un diagnóstico que permite saber en primer lugar si el kernel soporta USB Host y en segundo lugar si la API se encuentra disponible para ser utilizada por 3rd party apps. También ofrece una funcionalidad que copia el archivo de la API al kernel de Android si se tienen disponibles permisos de root en el dispositivo (luego detallaremos que usualmente esos permisos no se tienen). En la figura 15 se muestra un ejemplo de estadística.

Al correr el diagnóstico de la aplicación se obtuvo correctamente lo que ya sabíamos por nuestro prototipo: La Tablet posee soporte a USB Host pero no contiene la API de USB Host disponible para 3rd party apps.

Figura 15: Ejemplo de USB Host Diagnostics mostrando soporte completo a USB OTG.

Al intentar utilizar la funcionalidad de USB Host Diagnostics que copia el archivo en el kernel necesario para habilitar la API, se obtuvo un resultado exitoso dado que la Tablet ya poseía permisos de root. Luego de hecho esto, el diagnóstico de la aplicación retornó que la API se encontraba disponible para 3rd Party Apps, lo cual fue comprobado con el prototipo desarrollado por el equipo en Java que esta vez, sí retornó los dispositivos conectados mostrando el funcionamiento correcto de la API.

9.3.2 Estadísticas de compatibilidad de USB Host

En base a los resultados obtenidos en la sección anterior, que nos determinan claramente la dificultad de determinar el margen de compatibilidad que ofrece nuestra aplicación respecto a la posible conexión con la placa USB4Butiá mediante USB Host, decidimos realizar una breve estadística que nos otorgue la capacidades de USB Host y disponibilidad de la API para algunas de las Tablets disponibles en el mercado local.

Para esto utilizamos precisamente la herramienta descrita en la sección anterior: USB Host Diagnostics, la cual sube a una página web [47] los resultados del diagnóstico (que también se describió en la sección anterior) para todo usuario que lo consienta. Se forma así una base de datos que expone para los distintos dispositivos donde se hizo el test, los resultados de este acerca de las capacidades de USB Host. Recorriendo distintas tiendas on-line locales buscamos Tablets en venta en Uruguay que posean resultados de tests de la herramienta para llenar la estadística, incluyendo también algunos resultados de Tablets personales testeadas por el equipo a las cuales pudimos acceder.

Cabe aclarar que para algunos dispositivos existían varias entradas de datos en la página, que no siempre coincidieron en las conclusiones del análisis. Mostramos a continuación los resultados obtenidos:

Modelo	**API USB Host presente para 3rd party apps**	**Porcentaje de positivos**
Samsung n8000	Si	55,2 % (42/76)
Ledstar AQ8	Solo rooted	(2/2)
ACER B1	No	0% (1/1)
Asus TF300T	Si	70% (46/65)
Lenovo A2107	No	0% (1/1)

Samsung PT5100	Si	44,7% (30/67)
Google Nexus 7	Si	50% (2/4 los demás solo rooted)

*Solo Rooted significa que originalmente no poseían la API pero que permiten hacerse de permisos root y ejecutar la funcionalidad de copia de la API al kernel luego de lo cual si quedo disponible para 3rd party apps.

9.3.3 Modo Root en dispositivos Android

Como se especificó en las secciones anteriores, el problema de compatibilidad USB Host se podría reducir o disipar un poco si se considera acceder en el dispositivo a los permisos de root de este (conocido como "Rooting"). Si esto se logra, entonces se podría copiar o modificar el archivo necesario del Kernel para que la API de USB Host se vuelva funcional, reduciendo así la lista de dispositivos incompatibles. Sin embargo este grupo descarta esta idea basándose en las siguientes consideraciones que asume como inaceptables:

- El hecho de tener que hacer modificaciones en el dispositivo o pasos de instalación que sean complicados o no triviales para los usuarios, reduce por un lado la simplicidad de la aplicación, así como también el rango de usuarios a los que apunta la aplicación, ya que aquellos que tengan dificultades en los pasos previos de hacerse con permisos de Root o no entiendan lo que les solicita la aplicación descartaran la aplicación por sentirla poco práctica.

- Muchos de los fabricantes de dispositivos Android tienen como cláusula de rescisión de la garantía el Rooting del dispositivo o la modificación del Kernel instalado de fábrica. Por lo cual acceder a permisos de root para habilitar la API de USB Host podría dejar nula la garantía del dispositivo Android, según cual sea su fabricante.

9.4 Comunicación vía Wi-Fi

Tomando en cuenta que se desea contemplar un medio de comunicación que favorezca el uso de una aplicación Web y que evite dentro de lo posible cables que hagan engorroso

el diseño del Robot Butiá, teniendo este además la necesidad de constante movimiento, es lógico pensar que el mejor medio para realizar esto es por medio de una conexión inalámbrica. Sin embargo dado que el hardware del robot Butiá hoy en día no incluye una SBC y como USB4 Butiá no soporta el manejo de un dispositivo USB, lo cual permitiría la inclusión de un dispositivo USB de Wi-Fi, la posibilidad de establecer una conexión inalámbrica entre los elementos principales del sistema se sujeta a que se realice una expansión o agregado al kit de Butiá. Este consta de incorporar una SBC con sistema operativo de propósito general y puertos USB que permitan agregar dispositivos de Wi-FI como Dongle Wi-Fi así como conectar la USB4Butiá a la placa SBC. Para las siguientes secciones se asumirá que existe una SBC Raspberry Pi como la detallada en la sección "Complementos" de "Proyecto Butiá y Plataforma robótica". Un diagrama simple de los componentes sería el siguiente:

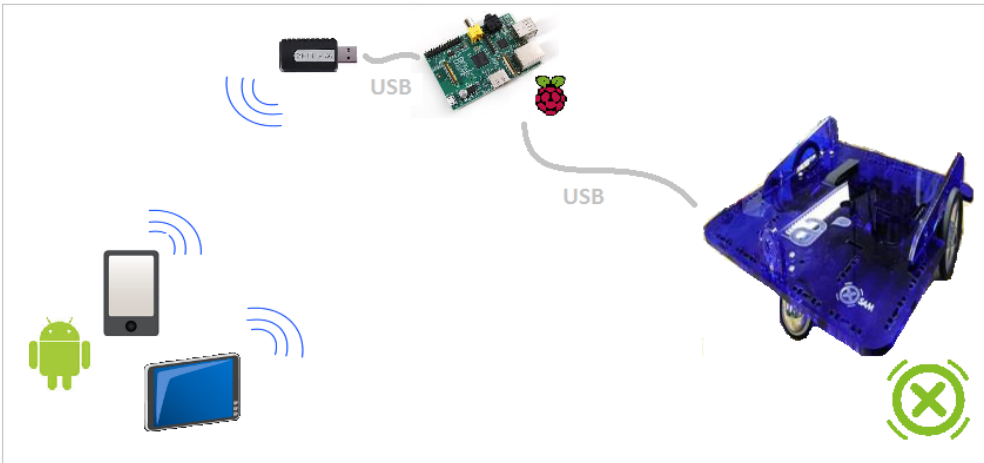


Figura 16: Diagrama de componentes asumiendo conexión Wi-Fi con Raspberry Pi.

En la Figura 16 se observa que los dispositivos Android se conectan vía Wi-Fi (se detallarán cómo a continuación) con un Dongle Wi-Fi USB agregado a la Raspberry Pi. Así mismo la SBC está conectada mediante USB al Robot Butiá (más específicamente a la placa USB4Butiá).

Dado el esquema anterior, es interesante como primer paso repasar algunos aspectos de la muy popular tecnología Wi-Fi.

Cuando hablamos de Wi-Fi nos referimos al sistema de comunicación de datos inalámbrico, que utiliza ondas electromagnéticas con una cierta modulación para intercambiar mensajes con otro equipo, sin necesidad de una conexión física directa con este.

Habitualmente en las redes Wi-Fi convencionales existen dispositivos de control, también conocidos como puntos de acceso inalámbrico o hotspots. Estos dispositivos dan soporte físico para la creación y mantenimiento de redes inalámbricas, administrando la conexión y desconexión de dispositivos a la red.

En el marco de este proyecto como ya se identificó en múltiples ocasiones existirán dos actores principales: el dispositivo Android cliente y el robot Butiá. El primero se conoce por los mercados tecnológicos actuales y por las características de Android que dispone de soporte para conexión Wi-Fi. El segundo (robot Butiá) no lo soporta actualmente, sin embargo se tomará como cierta la hipótesis que se asume en múltiples ocasiones de este documento de que el kit robot Butiá se expandirá para incluir una SBC que soporte USB y tenga un sistema operativo de propósito general (como por ejemplo una Raspberry pi aunque el modelo a efectos prácticos no interesa). Asumiendo esto, se incluirá un dispositivo dongle Wi-Fi que permite a la SBC establecer una conexión Wi-Fi con el dispositivo Android.

Una vez afirmado el soporte Wi-Fi de ambas partes existen varias opciones para establecer una comunicación entre ellas. Una es asumir que ambas están en una red WLAN controlada por un router o WAP (access point) y que el usuario establezca de manera directa la IP correspondiente a la SBC. Otra es mediante la utilización de Wi-Fi Direct, tecnología que permite el establecimiento de una conexión peer to peer de manera segura. También se puede utilizar una red Ad-Hoc entre el dispositivo Android y la SBC.

Si planteamos una arquitectura en la cual asumimos que ambos nodos de la conexión se encuentran bajo una red WLAN controlada por algún manager estamos introduciendo de antemano una limitante importante en nuestro sistema; la necesidad de que exista un tercer actor que es el administrador del Wi-Fi como ser un router u otro. Esto limita la cantidad de escenarios y lugares en donde se puede utilizar nuestro sistema, agregando otro requisito no menor de usuario para usar al sistema.

No existe mucha diferencia entre las dos últimas opciones de comunicación Wi-Fi que marcamos. Por un lado Wi-Fi Direct ofrece un nivel más alto de seguridad encriptada que Ad-Hoc (WPA2) y además permite que cualquier nodo que participe en la conexión pueda conectarse a su vez, a otras redes inalámbricas al mismo tiempo. El problema principal que identificamos en Wi-Fi Direct es que para ofrecer los servicios deseados de búsqueda e identificación de potenciales equipos a conectar está disponible solo en versiones de Android iguales o superiores a la 4.0, lo cual siendo coherentes con lo establecido hasta ahora la conexión Ad-Hoc sería la opción más razonable.

Capítulo 10

Interfaz Gráfica

10.1 Introducción

Se han analizado en la sección “Identificación de Problemas”, de manera previa al análisis de requerimientos que se incluirá en este proyecto, las distintas cualidades que son necesarias para la interfaz gráfica; la cual debe modelar un lenguaje de programación visual (LPV) basado en bloques, que sea amigable y claro, y que permita a personas sin conocimientos informáticos (especialmente niños) de manera intuitiva lograr códigos de programación simples para controlar el robot Butiá. Inevitablemente la interfaz de usuario, por más que deba respetar ciertas características, se verá condicionada y delimitada por las tecnologías que se usen para desarrollarla. A lo largo de este documento fueron consideradas dos posibilidades fuertes para el desarrollo de un IDE nuevo: una es realizar una aplicación con Java y Android SDK, la otra es realizar una aplicación web con HTML5. A continuación se detallan las posibles opciones de bibliotecas e interfaces gráficas que fueron consideradas para cumplir con los objetivos marcados en los requerimientos principales de este proyecto.

Primero que nada es interesante introducir el sistema Scratch por ser de vital influencia para los LPVs considerados, por más que la versión original no pueda ser considerada por no entrar dentro de las categorías mencionadas.

10.2 Scratch

Scratch, tiene un importante respaldo en la comunidad con alrededor de 37 mil visitas a su página Web en el último mes [35] y con una cantidad de 1500 nuevos proyectos subidos a la Web por día [27]. Scratch ha influenciado fuertemente a gran parte de los proyectos posteriores de este tipo, posicionándose como una relevante referencia para los propósitos de este proyecto.

La construcción de Scratch comenzó en el año 2003 y fue publicado en el año 2007, siendo un software libre disponible en más de 50 lenguajes, desarrollado en el lenguaje de

programación Squeak por “The Lifelong Kindergarten group” en el Medialab del MIT (Massachusetts Institute of Technology) por un equipo a cargo de Mitchel Resnick. Este grupo de desarrolladores se basó en las ideas que lanzó Logo y posteriormente Etoys para lograr el objetivo de introducir en la programación a personas sin experiencia en este campo, esto condujo a decisiones de diseño como la elección de un lenguaje visual basado en bloques, la interfaz de usuario con una única ventana, y un conjunto pequeño de comandos, como se observa en la figura 17. Y otras no tan evidentes como la gestión de errores (no se despliegan mensajes de error), o las modificaciones en el sistema a partir de los resultados obtenidos por los usuarios.

Scratch es un entorno de programación visual que permite a los usuarios crear proyectos interactivos de mediano porte, como historias animadas, juegos, videos de música, simulaciones, entre otras. Ha sido usualmente distribuido por escuelas y organizaciones educativas, el ejemplo más significativo es la distribución de Scratch por One Laptop Per Child siendo incluido en millones de laptops XO. Además este sistema posee una gran compatibilidad, permitiendo su instalación gratuitamente en cualquier ordenador con Windows, Linux, o Mac OS.

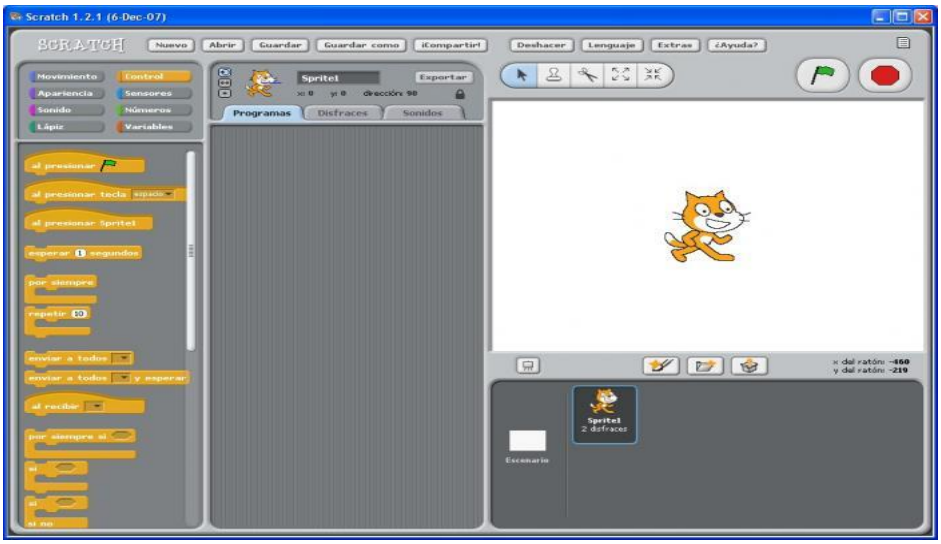


Figura 17: Sistema Scratch 1.2.1.

10.3 Opciones para HTML5

En esta sección se describirán las bibliotecas o proyectos (de código abierto) consideradas que cumplen las características pedidas para la aplicación Web, están basados en lenguajes de programación visual con las características mencionadas

anteriormente. Además, se analizará la potencial portabilidad de estos sistemas en dispositivos móviles tanto como en Tablets y cuán adaptables son estos a las necesidades del IDE para el robot Butiá.

- **Scratch 2.0:** es un LPV secuela de Scratch y bastante reciente creado por Media Lab del MIT, es un entorno de programación constituido por símbolos iconográficos denominados bloques. Esta versión de Scratch fue desarrollada en ActionScript (Adobe Flash) permitiendo editar proyectos directamente desde el navegador Web.

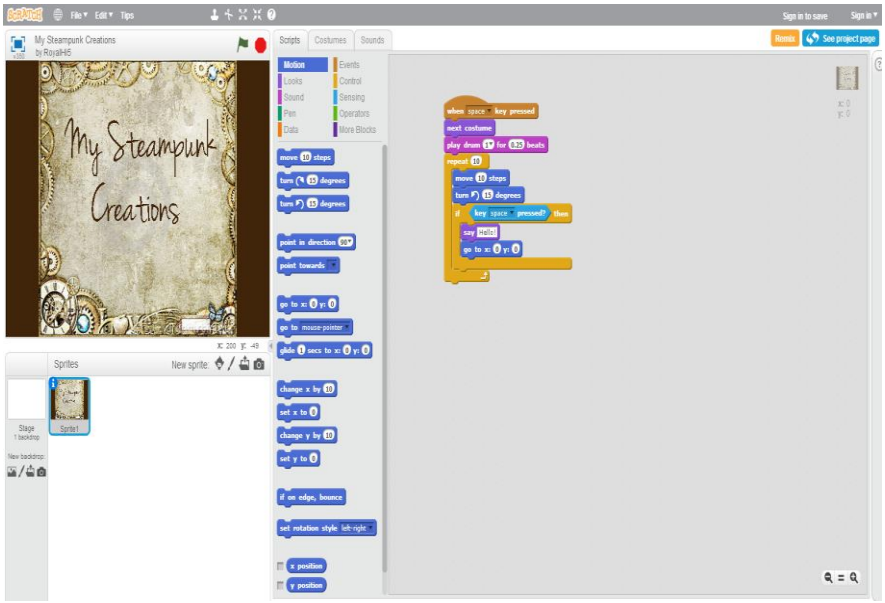


Figura 18: Interfaz de Scratch 2.0.

La versión original de este sistema forma parte del software de las XO y es compatible con la mayoría de los sistemas operativos, siendo reconocida y aceptada a nivel educativo, como fue mencionado anteriormente. Esta nueva versión permite crear y mezclar proyectos en forma colectiva, se da la posibilidad de crear nuevos bloques (lo cual equivale a crear procedimientos), también permite cierto control de gestos y hacer uso de la webcam para interactuar con los proyectos moviendo las manos o el cuerpo. Además se pueden subir datos a la nube, fortaleciendo características en términos de la comunidad; añadiendo página de perfil donde un usuario puede mostrar sus proyectos y comentar al resto de los usuarios sobre que está trabajando actualmente. Por estas razones, a priori, Scratch 2.0 se posicionó como una excelente alternativa. Sin embargo, Scratch 2.0 esta desarrollada en Adobe Flash lo cual implica una gran desventaja ya que muchos dispositivos móviles y Tablets no soportan este software a nivel de usuario o necesitan instalarse APIs para poder ejecutarlo siendo incluso necesario en algunos casos rootear el dispositivo para poder instalarlo.

- **Blockly:** es un LPV open source creado por Neil Frase, Quynh Neutron, Chris Pirich, Ellen Spertus y Phil Wagner, desarrolladores en el marco Google Code, que permite manipular y arrastrar bloques para construir aplicaciones, sin necesidad de typeo, basada en Scratch. Brinda a sus usuarios la posibilidad de crear simples programas como macros o scripts [18].

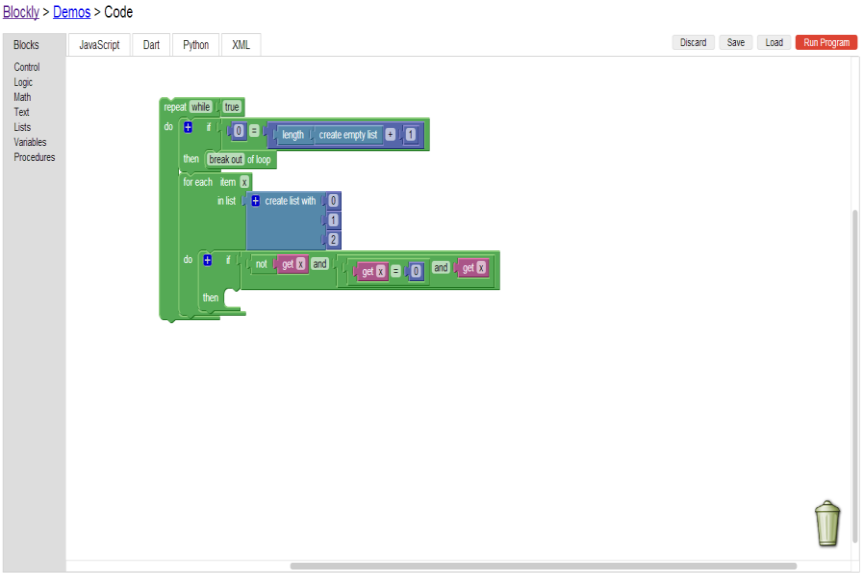


Figura 19: Interfaz de Blockly.

Está implementado en JavaScript y como tal está diseñado para ejecutarse client-side en una página Web, dando la posibilidad de generar a través de los bloques código JavaScript, XML, Dart o Python, que puede ser ejecutado independientemente. Blockly, es un proyecto muy activo y muy cambiante actualmente. Al principio se había descartado su inclusión a nuestro sistema, debido a su mal funcionamiento en Tablets y Smartphones (se probó un prototipo hosteado en Jboss en dispositivos con Android 2.3 y 4.0.4), el equipo reportó los bugs a los creadores y continuó la investigación, sin embargo se reiteraron las pruebas unos meses después de las primeras y se observó que todos los problemas importantes que poseía había sido corregidos. El único problema relevante que presenta actualmente es que su motor de render está basado en SVG (Scalable Vector Graphics), el cual no es compatible en versiones viejas de Android Browser como la 2.3 y anteriores.

- **Snap!:** es un LPV open source desarrollado por Jens Mönig de MioSoft Corporation, con ayuda en el diseño y documentación de Brian Harvey y con colaboraciones de estudiantes de Berkeley. Snap! surge como sucesor de BYOB (Build

Your Own Blocks) una avanzada modificación realizada sobre el código fuente de Scratch, enfocada en extender las funcionalidades del mismo dando, entre las mejoras más destacadas, la posibilidad de crear nuevos bloques o utilizar recursión. BYOB intenta incrementar el rango de edad de los usuarios consumidores, está orientado para estudiantes de computación o alumnos liceales. Snap! [21] es una re implementación extendida de Scratch para la Web en JavaScript, diseñada para soportar las limitaciones de los software basados en el navegador.

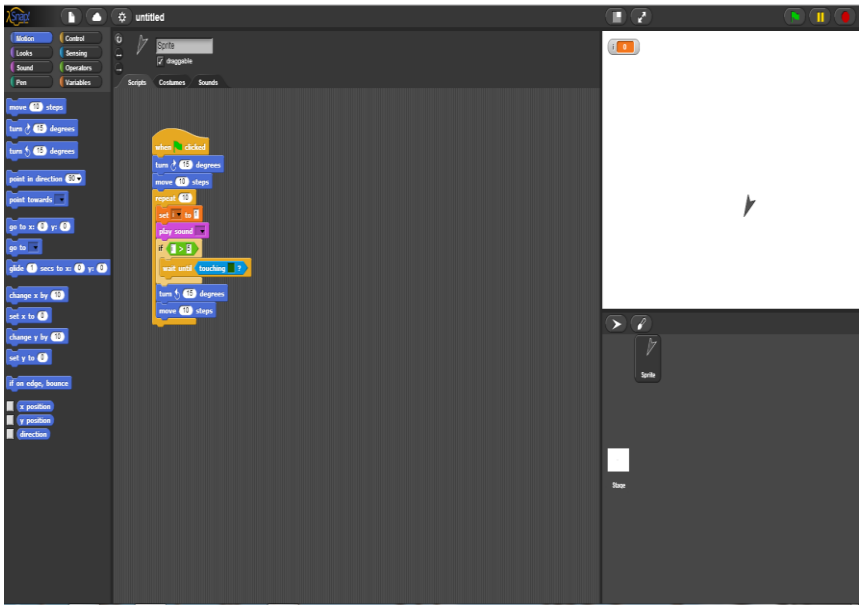


Figura 20: Interfaz de Snap!.

Snap! incorpora elementos muy interesantes desde el punto de vista de la programación pero no vale la pena entrar en detalle sobre ellos porque escapa de los fines de este proyecto, dado que los usuarios a los que se apunta serán principalmente estudiantes con pocos o nulos conocimientos de sistemas y difícilmente hagan uso de elementos avanzados de programación. Además posee, al igual que TortuBots, un escenario donde se simulan las acciones al ejecutar los distintos scripts como se ve en la figura 20, y una sintaxis donde los movimientos son medidos con distancias. Estas funcionalidades no resultan acordes al paradigma elegido para la arquitectura robótica el cual se basa en comportamientos. En primer lugar, el escenario de simulación no contempla los posibles obstáculos con los que el robot puede encontrarse y resulta muy costoso realizar una simulación fiel a los movimientos del robot, luego se desea definir bloques que se mapeen a comportamientos robóticos como por ejemplo “mover robot hacia delante” en vez de definir bloques de la forma “mover robot 1 metro hacia delante” para evitar errores de precisión en los comportamientos robóticos. Al realizar pruebas en dispositivos móviles y Tablets de este sistema, que cabe mencionar está en pleno desarrollo (actualmente en fase de testeo de una versión beta), tuvimos resultados positivos desde la

portabilidad, pudiendo llegar a ejecutarlo, sin embargo resultó bastante lento en el renderizado y se mostró inestable, ya que se presentaron en varias ocasiones bugs bloqueantes que cerraron el navegador.

- **Waterbear:** es un LPV open source creado por Dethe Elza inspirado en el lenguaje Scratch. Destinado a principiantes en especial niños, considerado una herramienta educativa que permite crear programas con tan solo arrastrar y montar bloques sobre una superficie de diseño. Waterbear está actualmente en desarrollo y se construyó utilizando HTML5, CSS3 y JavaScript. Contiene a modo de plugins módulos con bloques que permiten generar código JavaScript o también bloques para Arduino, siendo posible también la expansión del proyecto para agregar bloques personalizados cuyo código generado sea también custom. Se encuentra actualmente en versión pre-alpha [16].

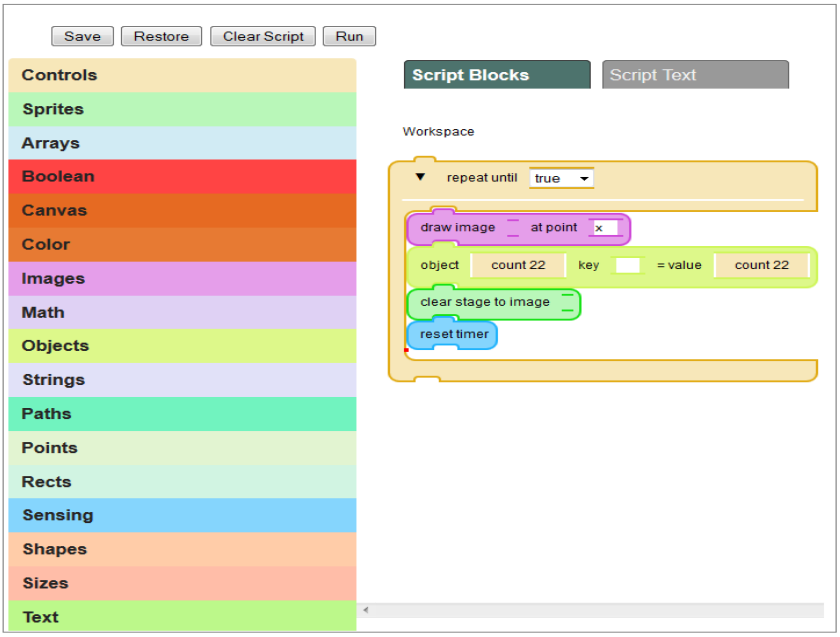


Figura 21: Interfaz de Waterbear.

Considerando el estado en construcción de esta herramienta vale la pena mencionar, sin entrar en detalle, que para poder experimentar con ella se realizaron ciertas modificaciones. Entre ellas se destaca que la versión más reciente al momento de la prueba poseía problemas en el drag and drop, a lo cual se tomaron códigos de versiones anteriores al proyecto que si tenían estabilidad (la que se podía percibir en waterbearlang.com), además se incluyeron bloques de prueba customizados a modo de extensión (como plugin) y se editó el HTML del playground original para forzar la carga de plugins a solo el plugin custom que creamos. Estas modificaciones, fueron a modo de

prueba, destacando que no serán necesarias (salvo la extensión que es inherente al código de Waterbear) cuando la aplicación salga de la fase alpha y se vuelva estable. En un principio en las pruebas realizadas se notó una velocidad de respuesta un poco lenta, pero esto se pudo mejorar ajustando los CSS para eliminar la transparencia de bloques al hacer drag. El problema principal que se observó en las pruebas fue la no compatibilidad con Android Browser y con IE (se hicieron pruebas sobre Windows Phone), aunque si con Firefox y Chrome tanto las versiones de escritorio como móviles.

Por último queda mencionar aquellos sistemas que también forman parte de la investigación pero que han sido descartados por incompatibilidades ya analizadas o que presentan menor cantidad de prestaciones, siendo el caso de Designblocks [13] quien al igual que Scratch 2.0 está desarrollado en ActionScript3, phpBlocks [12] el cual no está orientado principalmente al desarrollo educativo o LilyApp [26] el cual es un LPV pero que no está basado en bloques.

10.4 Opciones para Java con Android SDK

Para Java y Android SDK se consideró Catroid, la cual será presentada a continuación. Los buenos resultados obtenidos, llevaron a no continuar la investigación de otros proyectos similares para esta tecnología como App Inventor for Android y otros.

10.4.1 Catroid

Catroid es un sistema open source de programación visual on-device para dispositivos Android que está inspirado en Scratch. Corre en dispositivos móviles y Tablets, y está desarrollado especialmente para brindar a niños o personas sin conocimiento en programación, la posibilidad de crear programas arrastrando y uniendo bloques mediante drag and drop. Este sistema, a diferencia de Scratch o App Inventor, no necesita el uso de un ordenador ni teclado por lo que intenta aprovechar los beneficios de las pantallas más pequeñas y táctiles.

Los creadores denominan al conjunto de tools, y al “lenguaje” que se genera con los bloques de Catroid, como “Catrobat” [8] o “lenguaje Catrobat”. Existen actualmente versiones para iOS y versiones de Catrobat similares a Catroid para HTML5 que están en desarrollo y por lo tanto no están estables o completas. Así mismo la expansión del denominado proyecto Catroid (que incluye sistemas para distintas plataformas que

manejen Catrobat y sean similares a Catroid) se encuentra actualmente planteada en el "Summer Of Code" de Google [9].

Resulta interesante de esta aplicación su estabilidad y su interfaz gráfica (figura 22), que cumple con los requisitos principales que identificamos previamente. Al ser proyecto Open Source, se descargó el código con la intención de analizar y estudiar su posible extensión para incluir las funcionalidades de control de Butiá que se desean. Cabe mencionar que uno de los proyectos registrados de Catroid (proyectos de personas que clonan el principal y abren una Branch aparte del Catroid) es una extensión que incluye bloques para el control del robot Lego MindStorm con el cual se comunica mediante una conexión vía Bluetooth desde el dispositivo. Dicha versión permite acceso a gran parte de sus sensores como el acelerómetro, giroscopio, o GPS [39].

Figura 22: Catroid en un celular con Android.

Luego de instalar la aplicación y abrir la solución con Eclipse se pudo apreciar que Catroid es un proyecto muy bien estructurado y modularizado, lo cual permitió entender las características básicas de la solución en un periodo de tiempo razonable. Si bien el agregado de nuevos bloques para programar los comportamientos del robot Butiá podría llegar a ser aceptable, lo más intuitivo para estudiantes que quieran utilizar el sistema que se desarrollará sería tener solo las funcionalidades necesarias para programar al Butiá y no todas las otras funcionalidades de Catroid que resultan inutilizables e inaplicables a los objetivos de este proyecto, por lo cual podría llegar a ser interesante realizar una reestructuración más profunda del código (si se elige tomar este camino) y limitar sus funcionalidades. Dada la estructuración, modularización y fácil aprendizaje del código, se considera a Catroid como opción perfectamente viable. Aunque tenga algunos puntos en contra, como que el desarrollar el sistema planteado en este proyecto de grado provoca

desviarse mucho del proyecto original perdiendo un poco las ventajas de posibles mejoras de Catroid a futuro.

Capítulo 11

Concurrencia y orientación a comportamientos

11.1 Introducción

Como ya fue mencionado en otras secciones, este trabajo tiene como objetivo la programación robótica educativa basada en comportamientos. Los distintos comportamientos del Robot que sean programados en este IDE deben poder coexistir y ejecutarse de manera concurrente. Si se asume la arquitectura robótica basada en prioridades, las tareas programadas deben tener un determinado valor de prioridad y su ejecución debe ser acorde y en coherencia a estas prioridades establecidas.

Se deduce de lo anterior que las tareas deben ser capaces de interrumpirse entre sí, es decir que si se dan las condiciones para que se ejecute una tarea de mayor prioridad, la de menor prioridad debe postergar su ejecución y dar paso a la de mayor prioridad. Similar para el caso de subsumption. Para contemplar este problema hay muchas opciones. Fueron consideradas principalmente dos.

11.2 Semáforos y Threads en Android

El SDK de Android posee diversas herramientas para la creación y administración de Threads [33]. Si se considera que la solución está autocontenida en el dispositivo Android, se puede manejar los comportamientos en forma de threads sincronizados mediante semáforos utilizando las herramientas brindadas por Android SDK. Una idea sería por ejemplo tener una tarea “Manager” que se encargue de las tareas de sensado del robot y la administración de la ejecución de las tareas creadas por el usuario tomando en cuenta las prioridades de este.

```
new Thread(new Runnable() {
    public void run() {
        //do stuff
    }
}).start();
```

Figura 23: Ejecución de tarea en nuevo thread con Android SDK.

En la figura 23 se puede observar un ejemplo muy sencillo de ejecución de una tarea en un thread nuevo. Se puede sincronizar a cada tarea que sea creada con la tarea "Manager" utilizando semáforos como se observa en la figura 24, Android SDK ofrece diversas herramientas para crear y manejar a estos. A continuación vemos un ejemplo sencillo de creación y manejo de un semáforo simple.

```
private static final int MAX_AVAILABLE = 100;
private final Semaphore available = new Semaphore(MAX_AVAILABLE, true);

public void ActivateButiaLaserGun() throws InterruptedException {
    available.acquire();
    //..
    //Do stuff
    //..
    available.release();
}
```

Figura 24: Creación y utilización de un semáforo simple en Android SDK.

11.3 Toribio y Lumen

Lumen es un entorno simple creado por Jorge Visca para ejecuciones multitarea basadas en corutinas. Consiste básicamente en un scheduler con herramientas complementarias y fue inspirada en la descripción del scheduler de Sierra. Lumen no posee dependencias ni código de C y corre en Lua sin modificaciones, (funciona con Lua 5.1, 5.2 y LuaJIT) [50]. Posee además herramientas de logging, remote shell y web server que nos serán de mucha ayuda y que comentaremos más adelante.

En Lumen se crean tareas que se agregan al scheduler para su ejecución. Este maneja el concepto de señales, una señal es un evento emitido por una tarea y que es manejado por el scheduler de Lumen. Una tarea (task) en particular puede bloquearse esperando por

una señal emitida por una o más tareas (figura 25) o iniciar su ejecución cuando esto sucede. Así, se puede tener una tarea que lee los sensores del Robot Butiá y al detectar, por ejemplo, que el botón fue presionado emitir una señal que despierte otra task que se encargue de realizar acciones correspondientes.

```
local sched = require 'sched'
-- task emits signals
local emitter_task = sched.run( function()
                                    for i = 1, 10 do
                                        sched.signal('an_event', i)
                                        sched.sleep(1)
                                    end
                                end
                               )

-- task receives signals
sched.run( function()
               local waitd = {emitter=emitter_task, events={'an_event'}}
               while true do
                   local _, _, data = sched.wait(waitd)
                   print (data)
               end
           end
         )

sched.go()
```

Figura 25: Ejemplo de una tarea activando a otra mediante un signal.

Toribio [51] es un entorno de desarrollo de aplicaciones de robótica para plataformas embebidas. Está construido alrededor de Lumen y provee un entorno que simplifica el desarrollo y despliegue de aplicaciones robóticas. Entre los servicios que provee están:

- Un sistema centralizado de configuración.
- Objetos que abstraen el hardware disponible ("devices").
- Repositorio central de tareas, devices.

La ventaja principal que otorga trabajar con Toribio es la creación organizada de tareas centralizadas que se comuniquen entre sí a través del scheduler de Lumen con signals y que permitan establecer de manera simplificada la comunicación con el hardware (devices).

Cada tarea a ejecutarse con Toribio se declara en un archivo de configuración en conjunto con variables o atributos propios (globales) de la tarea que también pueden declararse allí. Toribio incluye a Bobot (ver sección de Bobot) como uno de los devices por defecto creados, lo cual permite que cualquier tarea creada pueda comunicarse con

Bobot de manera sencilla y obtener fácilmente información de los sensores así como manipular los actuadores. Vemos en la figura 26 un ejemplo.

```
local toribio = require 'toribio'
local sched = require 'sched'
-- Espera a que se carguen los motores y el boton (bb es por Bobot)
local motors = toribio.wait_for_device('bb-motors')
local button = toribio.wait_for_device('bb-button:2')

--Pregunta si el boton del sensor esta siendo presionado
if button.getValue() == 1 then
    --Establece la velocidad de ambos motores en 700 hacia adelante
    motors.setvel2mtr(1,700,1,700)
    --Libera el control del procesador
    sched.sleep(2)

end
```

Figura 26: Extracto de una tarea que mueve los motores según sensor.

11.4 Servidor Web y Remote Shell de Lumen

Como fue mencionado anteriormente, Lumen posee una funcionalidad muy interesante que es la capacidad de actuar como servidor web. Si se utiliza este servidor web como una tarea de Toribio, podemos mantener de forma centralizada la comunicación entre el cliente y las interfaces robóticas y base de datos. Es interesante notar, que si se realiza una implementación en HTML5 (como se sugirió en secciones anteriores) se podría enviar la información referente a las tareas implementadas por el usuario en el IDE en forma de texto a Toribio mediante la utilización de POST al servidor web de Toribio y luego ser interpretadas y ejecutadas en este.

Cabe notar que uno de los requisitos básicos que tiene la aplicación a desarrollar es la funcionalidad de debugging. Esto significa que la aplicación debe mostrar al usuario de alguna forma los bloques que están siendo ejecutados actualmente. Para esto se necesita dar feedback a la aplicación Web de lo que el robot Butiá está ejecutando. Si se realiza una implementación web, se podría hacer un POST inicial con los comportamientos desarrollados por el usuario (considerados como las tareas a ejecutar por el robot) y luego al ejecutar, mediante polling de Ajax hacer sucesivos GETs al servidor web para mostrar un feedback de lo que ha sido ejecutado por Toribio. A primera vista, esta solución resulta costosa por la cantidad de mensajes HTTP que manejaría la aplicación, pero

existen métodos para mejorar el polling. Entre ellos se encuentra el long polling, que propone por medio de Ajax Push [20] evitar los sucesivos GETs al servidor, reteniendo la respuesta del lado del mismo mientras no exista un nuevo mensaje para el cliente. De esta manera, además de reducir la cantidad de pedidos, se logra mejorar el rendimiento ya que la información desde el lado del servidor será enviada inmediatamente al cliente.

Volviendo sobre las opciones para dar soporte a la funcionalidad de debugging, otra alternativa similar a la anterior es utilizar el Remote Shell de Lumen. Esta shell expone en un socket el servicio de shell. Esto significa que cualquier comando que sea enviado al socket se lo ejecutará dinámicamente como una tarea individual de Toribio y se ejecutará retornando a Lumen los resultados de dicha ejecución en el socket. En el interés del requisito de debugging mencionado, lo anterior nos permite que de realizarse una implementación Web, se pueda abrir un websocket para establecer una comunicación con la remote shell de Lumen y así enviar tareas y recibir feedback de estas para mostrar en la página web.

11.5 Websockets en browsers para Android

Respecto a lo anteriormente descrito (la comunicación mediante websockets entre una página web y la shell de Lumen) luego de realizar una investigación, observamos que el navegador instalado por defecto en el sistema operativo Android no soporta la utilización de websockets. Sin embargo, si existen otros browsers tanto para dispositivos móviles como para ordenadores de escritorio que soportan websockets. Para determinar compatibilidades realizamos una sencilla prueba de echo mediante websockets, la página websocket.org (http://www.websocket.org/echo.html) ofrece este sencillo servicio de Echo Test para determinar si el browser soporta o no el uso de web sockets, como se observa en la figura 27.

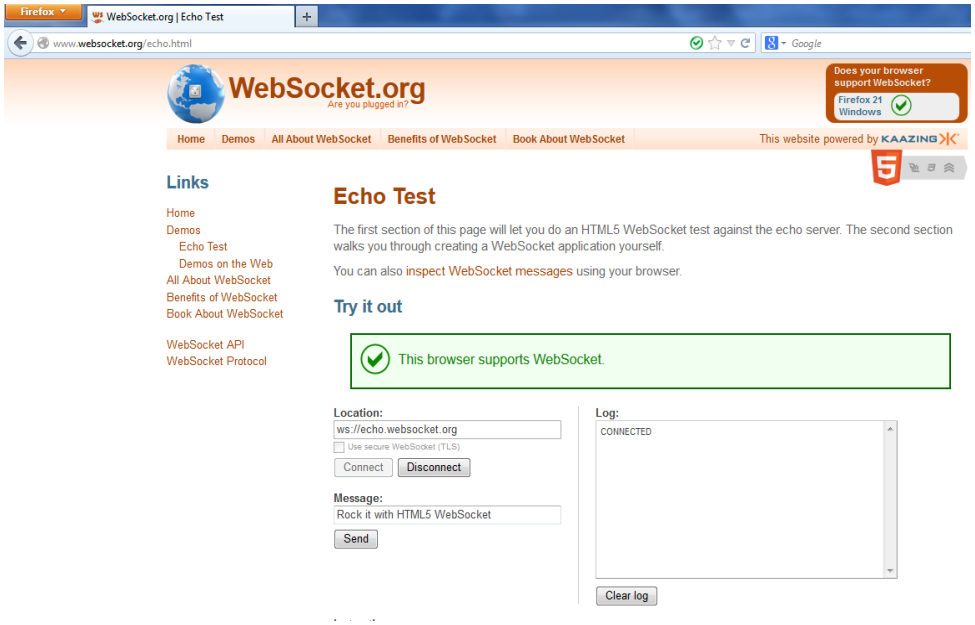


Figura 27: Echo Test con websocket.org para Firefox en un ordenador.

Los resultados en un celular con S.O Android 2.3 (Gingerbread) fueron que el navegador por defecto no soporto la utilización de websockets. Sin embargo en el mismo dispositivo se probó instalar un browser alternativo: Opera Mobile. Este último sí dio resultados positivos de soporte a websockets, poniendo en evidencia que la falta de compatibilidad no se debe al sistema operativo, sino al Browser utilizado, en este caso el que viene instalado por defecto en Android. Se repitieron exactamente las mismas dos pruebas para una Tablet con Android 4.1 (Jelly Bean) con iguales resultados confirmando las afirmaciones hechas anteriormente.

La tabla [53] de la figura 28 ilustra la compatibilidad de browsers con websockets (notar que las entradas en la tabla indican el número de versión para el cual se soporta el protocolo de websocket indicado).

Implementation status

Protocol	Draft date	Internet Explorer	Firefox[18] (PC)	Firefox (Android)	Chrome (PC, Mobile)	Safari (Mac, iOS)	Opera (PC, Mobile)	Android Browser
hixie-75	February 4, 2010				4	5.0.0		
hixie-76 hybi-00	May 6, 2010 May 23, 2010		4.0 (disabled)		6	5.0.1	11.00 (disabled)	
7 hybi-07	April 22, 2011		6[19]1					
8 hybi-10	July 11, 2011		7[20]1	7	14[21]			
13 RFC 6455	December, 2011	10[22]	11	11	16[23]	6	12.10[24]	

Figura 28: Soporte de browsers a los distintos protocolos de websockets.

Capítulo 12

Portar eButiá en Android

12.1 Introducción

La idea de portar eButiá en Android (se recuerda que eButiá corresponde al nombre del IDE desarrollado a partir del proyecto de grado mencionado en la sección "Experiencias anteriores") surge evidentemente por los objetivos que comparte con este proyecto.

eButiá es un IDE para programar comportamientos robóticos que a grandes rasgos fue realizado como una extensión del entorno de desarrollo Etoys y que está orientado a la arquitectura reactiva Subsumption. Resulta sumamente interesante investigar la posibilidad de levantar una máquina virtual u otra aplicación dentro de Android que pueda interpretar al menos parcialmente eButiá, ya que de esta forma se pueden continuar dichos esfuerzos y capitalizarlos en un IDE para programar comportamientos robóticos para el sistema Android que cumpla los requerimientos de este proyecto.

12.2 IDE eButiá

Es pertinente presentar las características principales del IDE eButiá y las herramientas utilizadas para su desarrollo dado que estas definen las consideraciones a ser tomadas en la investigación de su portabilidad para Android. Este IDE permite controlar la plataforma robótica Butiá de forma autónoma, y es ejecutado en la computadora portátil XO que se conecta de forma serial a esta plataforma. En la XO existe una aplicación que resuelve la comunicación y expone las funcionalidades del robot Butiá con quien mantiene una conexión vía USB, a través de una interfaz por sockets TCP/I P en el puerto 2009 de localhost. Esta aplicación es Bobot, la cual ya se ha mencionado, y es utilizada por eButiá para contactar con los sensores y actuadores del robot.

El IDE eButiá fue desarrollado sobre el entorno de desarrollo Etoys, una herramienta de software libre, multiplataforma y de código abierto que viene integrada en

las computadoras XO. Etoys es un entorno de programación visual desarrollado sobre Squeak SmallTalk, y se compone de un lenguaje, biblioteca de clases y un entorno de desarrollo que permite la implementación de aplicaciones. Es por esto que Etoys y Squeak SmallTalk son compatibles a usar la misma máquina virtual, comúnmente conocida como "Squeak VM", quien permite desde distintos sistemas operativos interpretar el lenguaje y ejecutar las imágenes de estos entornos. Por lo tanto, al igual que sucede con Etoys, la imagen del IDE eButiá fue implementada para ser levantada por la Squeak VM.

eButiá logra contemplar un entorno de desarrollo que permite a estudiantes desarrollar comportamientos robóticos orientados a una arquitectura que sigue el paradigma reactivo, haciendo uso de un lenguaje de programación visual basado en bloques.

12.3 Portabilidad de Smalltalk

12.3.1 Estado actual

El proyecto que dispara la idea de portar sistemas Smalltalk a Android surge en el año 2009 y es el proyecto Squeak-On-Android creado por Andreas Raab, que se encuentra (incluso actualmente) disponible en el market de Android. Como se menciona en la presentación del proyecto [19], Raab, participante de la comunidad de Squeak, desarrolló una primera versión de un sistema que conseguía portar, un poco más en sentido visual que funcional, Squeak en Android. Esta versión es de código abierto y como el propio Raab lo definió, es tan solo un primer esfuerzo que tiene muchos problemas pendientes de resolver, entre ellos los más importantes el soporte a la entrada de teclado de Android y el soporte a las funcionalidades de socket y network en general.

Descargamos y probamos la aplicación original de Raab en dos dispositivos distintos, primero un Motorola Defy con Android 2.3 y luego en una Tablets Ledstar con Android 4.0 logrando en ambas un resultado similar al descrito, el proyecto logró una portabilidad de Squeak poco aceptable, que no parece ser realmente usable todavía (quizás también porque la interfaz de Squeak está pensada para ordenadores escritorio).

La solución implementó una versión especial (event-driven) de la máquina virtual de Squeak dirigida por eventos, la cual deja de tener el tradicional bucle de eventos para actuar como controlador de una cola de eventos que retorna luego de procesar cada uno

de estos eventos. En la figura 29 se muestra una comparación.

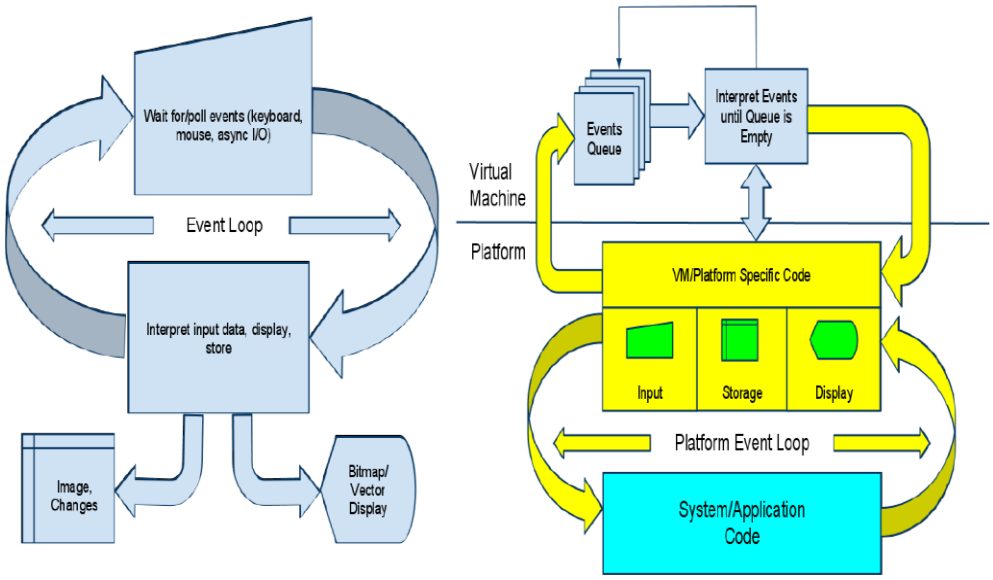


Figura 29: VM tradicional (derecha), Event-Driven VM (izquierda).

Esta modificación de la máquina virtual resulta muy conveniente a la hora de embeber otro lenguaje en tiempo de ejecución, el ejemplo en este caso es Squeak, en Java/Dalvik.

Dado que la interfaz de Squeak está pensada originalmente para ordenadores cuya pantalla es mayor a la que tienen la mayoría de los Smartphones y otros dispositivos Android de pantalla pequeña, fue creado en el 2011 una nueva branch [40] del proyecto principal de Raab que tenía como objetivo apuntar la portabilidad de Squeak a dispositivos Android con mayor tamaño y resolución de pantalla, como son las Tablets. Este proyecto, también de código abierto al igual que el original de Andreas Raab, logró mejorar varios de los aspectos del proyecto original, entre ellos lograr una integración estable con los inputs como el teclado touch, etc.

El trabajo continuó hasta principios del 2012, en donde la página declara las releases como obsoletas ante la aparición de un nuevo proyecto (que involucra en realidad casi a la misma gente) que orientó el rumbo de la portabilidad de Squeak no hacia la máquina virtual original de Squeak sino a la conocida Cog VM [29], una máquina virtual basada en la de Squeak pero con optimizaciones y funcionalidades agregadas. Este nuevo proyecto, de código abierto como sus antecesores, tuvo dos ramas bien diferenciadas: CogDroid y PharoDroid.

CogDroid ha mantenido actividad de desarrollo hasta Octubre del 2012. Lo que ofrece

es una versión estable para Android de la Cog VM. Su interfaz es simplemente una pantalla que ofrece los distintos archivos con extensión ".image" que se encontraron en el file system del dispositivo, y si se selecciona una imagen procede a cargarla con la máquina virtual. También ofrece un código pre compilado de la máquina virtual y un instructivo para poder agregar una imagen (.image) y compilar todo en un único instalable de Android ".apk". Por ejemplo, podemos tomar una imagen de Squeak y compilarla con la versión pre compilada de CogDroid para generar un único installer que combine ambos elementos. En principio, la VM de CogDroid no está limitada solamente a la imagen de Squeak, si bien el desarrollo de la VM tuvo a Squeak como objetivo principal. Un ejemplo de lo anterior fueron los trabajos de Hilaire Fernandes para portar Dr. Geo en Android [17]. Quien construyó un único package, reduciendo la dificultad de la instalación del sistema.

El proyecto de Pharodroid está en realidad basado en la misma idea. Pharodroid consiste en la VM pre-compilada de Cogdroid a la que le fue agregada una imagen de Pharo (la release actual tiene la versión Pharo 1.4) [24]. Por esta razón, centraremos nuestras pruebas solo en CogDroid ya que es el único que permite intentar cargar una imagen y por tanto intentar cargar eButiá. Se efectuaron las siguientes pruebas con CogDroid:

1) CogDroid + Squeak image: Esta prueba tuvo buenos resultados, CogDroid cargo correctamente la imagen (aunque con una leve lentitud) y se mostró una versión idéntica al Squeak para ordenadores en cuanto a calidad de imagen refiere (figura 30), en funcionalidad prueba ser mejor que el proyecto original de Andreas Raab, aunque posee aún ciertas inestabilidades, por ejemplo una que detectamos es que si se bloquea la pantalla del dispositivo o si surge algún evento que inactive temporalmente a CogDroid como tarea principal de Android, esta al reanudarse falla cerrándose la aplicación.

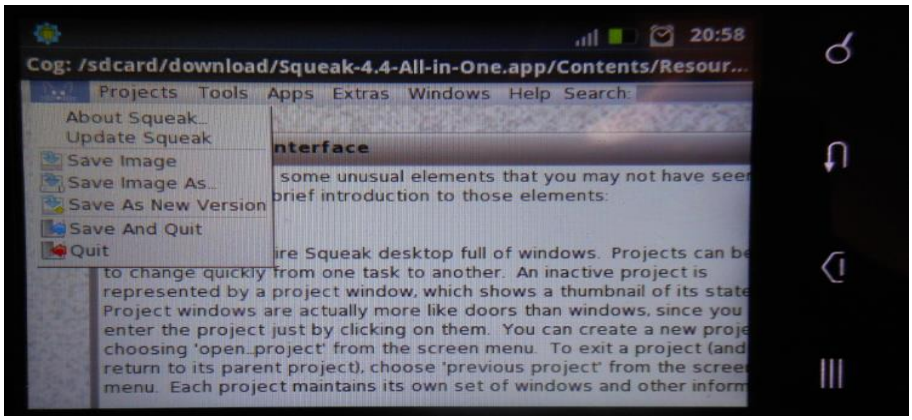


Figura 30: Squeak image con CogDroid en Android 2.3 (Motorola Defy).

2) CogDroid + Etoys image: Al intentar cargar la imagen de Etoys con CogDroid ocurre un error y se cierra la aplicación. Si observamos el log de la aplicación, como se detalla en múltiples ocasiones en la página de issues de CogDroid, el registro que aparece en el log es "Image could not be loaded".

3) CogDroid + eButiá image: Como era de esperar ya que eButiá está basado en Etoys, el resultado fue lo mismo que para la imagen de Etoys, al intentar cargar la imagen se cierra la aplicación. Observando el log, el mensaje de error es el mismo: "Image could not be loaded".

12.3.2 Comentarios finales y conclusiones de esta sección

Luego de las pruebas realizadas se descargó el código vigente de CogDroid (el cual resulta ser bastante grande, pesando 100 mb solo el código sin compilar) para observar sus características. El proyecto no está implementado simplemente en Java con Android SDK puro, sino que gran parte del código está almacenado en módulos escritos en C y C++ integrándose a las soluciones de Java mediante el uso de Android NDK [4], el cual es un toolset que permite implementar ciertas porciones del código de un proyecto para Android en lenguajes nativos como C y C++.

Observando el proyecto de la máquina virtual no fue fácil determinar un posible problema que repercute en la imposibilidad de traducir la imagen de Etoys. Dado que es un proyecto poco documentado y complejo y que se desconocen las características de los componentes de Etoys a fondo, así como de la portabilidad de estos, la curva de aprendizaje para hacer ingeniería inversa de CogDroid estimamos que tomaría un tiempo considerable. A esto se añade la problemática incertidumbre de que dicha curva de aprendizaje es solamente para determinar la causa del problema de cargar la imagen de Etoys y eButiá. El hecho de realmente lograr una implementación que solucione el problema puede quizás no ser realizable con la VM de CogDroid o tomar un tiempo no viable para el marco de este proyecto de grado.

Cabe agregar como análisis final, que en el supuesto caso de poder ejecutar la imagen de eButiá en Android, esta va a sufrir varios de los problemas detectados en el proyecto actual de CogDroid al portar Squeak, lo cual hace aún más cuestionable el esfuerzo descrito en el párrafo anterior. El problema mayor es que la interfaz gráfica tanto de Squeak como de Etoys y eButiá está pensada para ordenadores de escritorio como las computadoras XO y para dispositivos de entrada como el teclado o apuntadores

indirectos como el mouse. El llevar esta interfaz a dispositivos que hacen uso de la tecnología touch para la interacción con el usuario reduce enormemente la usabilidad y la dinámica del sistema original. Otro problema actual de CogDroid es que su máquina virtual es pesada para los recursos de procesamiento y memoria que suelen tener los dispositivos Android de hoy, por lo cual puede presentar cierta lentitud la máquina virtual quitando más dinamismo al sistema. La VM ocupa al menos 50 mb iniciales de memoria, más lo que consuma la imagen en si al ejecutarse lo cual se incrementa en el tiempo si se comienza a utilizar Squeak con normalidad, mientras que en el mercado muchos dispositivos Android de gamma media poseen tan solo 256 mb de RAM.

Capítulo 13

Comentarios finales y conclusiones

Al introducirse en el mundo de la robótica educativa hemos encontrado grandes contribuciones que desde mediados del siglo XX han aportado los fundamentos para el desarrollo y evolución de esta disciplina. Su masificación en países de Sudamérica está principalmente ligada al proyecto OLPC que brindó la infraestructura para poder utilizar la robótica educativa a nivel escolar y liceal. Esta realidad fue complementada por proyectos enfocados en esta rama, como lo es Butiá en el ámbito local. Siguiendo esta línea, este proyecto en concreto propone el desarrollo de un IDE basado en comportamientos robóticos que permita controlar al robot Butiá desde Tablets o dispositivos móviles con sistema operativo Android. En este documento se ha analizado el estado del arte de las tecnologías pertinentes, así como también los distintas conceptos teóricos en materia de robótica y en particular de robótica educativa, con el fin de brindar a los alumnos a los que apunta el proyecto una experiencia satisfactoria, y así aportar al crecimiento de la robótica educativa en nuestro país.

Comenzando con un alto nivel de abstracción, es inevitable detenerse a definir la corriente paradigmática que este sistema tendrá. La complejidad que conlleva controlar una plataforma robótica ha llevado a los investigadores a definir modelos que intentan ajustarse a escenarios concretos. En este caso hemos decidido enfocarnos en un paradigma reactivo, descartando los sistemas deliberativos e híbridos, dado que este brinda facilidad para desenvolver una lógica más transparente a los usuarios al eliminar la necesidad de planificación, ya que simplemente toma las percepciones obtenidas mediante los sensores y ejecuta comportamientos preestablecidos. Además, con la misma intención de facilitar la interpretación de los sucesos por parte de los usuarios, se decidió implementar una arquitectura robótica basada en prioridades y de esta forma eliminar la complejidad inherente a la arquitectura Subsumption. A criterio de este equipo, agrupar comportamientos en capas donde las capas superiores pueden alterar la entrada o salida de las tareas de capas inferiores, no resulta intuitivo en nuestra opinión para estudiantes escolares. A esto se le suma que el resultado de los actuadores del robot es el producto de las acciones de todos los comportamientos concurrentes de la capa que se está ejecutando, lo cual no permite que el estudiante pueda predecir de forma sencilla la salida del robot.

Siendo el sistema compuesto por dos componentes físicas, la plataforma robótica y los

dispositivos móviles, es fundamental tomar decisiones respecto al medio de comunicación entre ellos. Fueron manejadas dos grandes alternativas, por un lado se investigó la capacidad de los dispositivos Android para soportar USB Host, con la intención de lograr una comunicación serial directa entre las componentes. Se encontraron inconvenientes dependiendo de las versiones del sistema operativo Android para ejecutar aplicaciones de terceros que hagan uso de la API de USB Host. Para la otra alternativa, se asume el uso de una SBC (Raspberry Pi) como complemento en el kit del robot Butiá, asegurando la posibilidad de mantener una comunicación vía Wi-Fi entre las componentes, por medio de un dispositivo USB Wi-Fi. De esta forma, el sistema no depende de las capacidades del dispositivo móvil que posea el usuario, dando además mayor flexibilidad y alcance, además de contar con una comunicación sin cables.

Con el fin de aprovechar el agregado de la SBC, se utilizará Toribio y Lumen con el propósito de cumplir con las características de la arquitectura robótica elegida, ya que son dos sistemas que han sido desarrollados para los mismos o similares objetivos que los nuestros y que son rápidos debido a que aprovechan la ventaja de la velocidad del lenguaje Lua a los efectos de sus objetivos. Estos nos permitirán un entorno sencillo para manejar la coordinación y administración de los comportamientos robóticos.

Por último, se investigaron distintos sistemas que modelan un LPV basado en bloques y poseen las características deseadas para la interfaz de usuario, siendo agrupados a partir de las tecnologías con la que fueron desarrollados. Como detalle final, en la tabla que se presenta a continuación se resume de forma sencilla las ventajas y desventajas de cada una de las alternativas estudiadas. HTML5 y el uso de Lumen y Toribio se presentan como las tecnologías con mayores prestaciones, utilizando Blockly como LPV, debido a su superioridad en compatibilidad con navegadores, así como extensibilidad, estabilidad y visualización gráfica.

	Ventajas	**Desventajas**
HTML5	- Fácil acceso y distribución por ser web. - Amplia compatibilidad con Android, iOS, y sistemas operativos de escritorio (Linux, Windows, entre otros). - Bajo consumo de recursos del dispositivo Android (Browser). - Rapidez de los lenguajes de scripting (JavaScript y Lua si se utiliza Toribio como servidor). - Potencial gráfico de CSS3. - Proyectos de bibliotecas gráficas aún activos y con mejoras constantes.	- El sistema debe ser implementado desde 0, a excepción de la biblioteca gráfica. - Bibliotecas gráficas que implementan la programación en bloques se encuentran todavía en desarrollo o en estado beta.

Java con Android SDK basado en Catroid	- Potente aplicación (Catroid) para el soporte gráfico, que modela un LPV basado en bloques. - Código simple, fácil de extender y bien documentado -Experiencias similares con Lego Mindstorm.	- El sistema debe ser implementado desde 0, a excepción de la aplicación gráfica. - Poca portabilidad (debe ser instalado). - No es compatible para sistemas operativos distintos a Android - A menos que se haga una reestructuración fuerte de Catroid, se incluirían en el sistema muchas funcionalidades no pertinentes a este proyecto.
CogDroid + eButiá	- Fuertemente orientado a la pedagogía educativa por estar basado en Etoys. - eButiá ya está desarrollado, es basado en comportamientos y en el paradigma reactivo y comparte objetivos similares a los planteados en este trabajo. - Proyecto de CogDroid continua activo.	- Interfaz gráfica orientada a escritorio, presenta dificultades frente a la resolución de la pantalla y al input del usuario (touch). - Aún no implementado el soporte de CogDroid a Etoys y eButiá. - CogDroid aún con Squeak le faltan algunas funcionalidades, está en estado beta y es todavía un poco inestable. - Complejidad y riesgos para expandir el código debido al tamaño del proyecto y a la poca documentación de este. - Consume muchos recursos (procesamiento y RAM). - Poca portabilidad (debe ser instalado). - No es compatible para sistemas operativos distintos a Android.

Referencias

[1] Achkar, Alejandro; Margalef, Andrés. Presentación: Butiá con paradigmas reactivos sobre Etoys. Proyecto de grado. Universidad de la República, Facultad de Ingeniería, Montevideo, 2012. Disponible en Web: http://www.fing.edu.uy/~pgiderob/PresentacionSqueakFest.pdf, último acceso julio 2013.

[2] Achkar, Alejandro; Margalef, Andrés. IDE de programación orientado al desarrollo de arquitecturas robóticas basadas en comportamientos: Estado del Arte. Proyecto de grado. Universidad de la República, Facultad de Ingeniería, Montevideo, 2012.

[3] Arkin, Ronald C. Behavior-Based Robotics. The MIT Press, 1998. 441p. ISBN 0-262-01165-4.

[4] Android NDK. Android for Developers [en línea]. [Fecha de consulta: Julio de 2013]. Disponible en Web: <http://developer.android.com/tools/sdk/ndk/index.html>

[5] Bracco, Mireya. Porta, Darío. Tortugarte. Una actividad inspirada en Logo [en línea] [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.slideshare.net/ctepay/tortugarte-xo>

[6] Burnett, Margaret M. Visual Programming. John G. Webster (ed.). Encyclopedia of Electrical and Electronics Engineering. John Wiley & Sons Inc. 1999. Disponible en Web: <http://www.cs.auckland.ac.nz/courses/compsci732s1c/archive/2005/lectures/WhatIsVP.pdf>

[7] Casares Charles, Juan Pablo. AMIVA: Ambiente para la instrucción visual de algoritmos. Trabajo de grado. Instituto tecnológico autónomo de México, México DF, 1999. Disponible en Web: <http://usablehack.com/amiva/AMIVA.pdf>

[8] Catrobat. Catroid [en línea]. Repositorio: Github. Actualizada: julio de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web <https://github.com/Catrobat/Catroid>

[9] Catroid Project [en línea]. Google Summer Of Code 2013. Actualizada: 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.google-melange.com/gsoc/org2/google/gsoc2013/catroid_project>

[10] Consejo de educación secundaria. Plan Ceibal [en línea]. Actualizada: julio de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.ces.edu.uy/ces/index.php?option=com_content&view=category&id=49&Itemid=87>

[11] Deliberative Agents. Notes [en línea]. Actualizada: 2007 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://cgi.cse.unsw.edu.au/~aishare/index.php>.

[12] Demo for the Visual Programming SDK for PHP [en línea]. FreeGroup, [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.freegroup.de/software/phpBlocks/demo.html>

[13] DesignBlocks [en línea]. Lifelong Kindergarten. Actualizada: 2010 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.designblocks.net/>

[14] Doutel, Fernando. Conoce a la placa que quiere revolucionar tu mundo digital: Raspberry Pi a fondo (I) [en línea]. Actualizada: noviembre de 2012 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.xataka.com/componentes-de-pc>

[15] eLinux. The Raspberry Pi Wiki [en línea]. Actualizada: marzo de 2014 [Fecha de consulta: marzo de 2014]. Disponible en Web: <http://elinux.org/RaspberryPiBoard>

[16] Elza, Dethe. Waterbearlang Proyect [en línea]. Actualizada: julio de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://waterbearlang.com/>

[17] Fernandes, Hilaire. Dr geo, be a geometer! [en línea]. Actualizada: 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.drgeo.eu/home>

[18] Frase, Neil; Neutron, Quynh; Chris, Pirich; Spertus, Ellen; Wagner Phil. Blockly. A visual programming editor [en línea] [Fecha de consulta: julio de 2013]. Disponible en Web: <https://code.google.com/p/blockly/>

[19] Golubovsky, Dmitry. Squeak on Android [en línea]. Actualizada: 2011 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://wiki.squeakvm-tablet.com/Squeak_on_Android.pdf>

[20] Gravelle, Rob. Comet Programming: Using Ajax to Simulate Server Push [en

línea]. Actualizada: marzo de 2009. [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.webreference.com/programming/javascript/rg28/index.html>

[21] Harvey, Brian; Moning, Jens. Snap! Build your own blocks 4.0 [en línea]. [Fecha de consulta: julio de 2013]. Disponible en Web: <http://snap.berkeley.edu/SnapManual.pdf>

[22] Introduction to USB On-The-Go [en línea]. USB Implementers Forum, Inc. [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.usb.org/developers/onthego/USB_OTG_Intro.pdf>

[23] Jean Piaget [en línea]. Actualizada: julio de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://en.wikipedia.org/wiki/Jean_Piaget>

[24] JenkinsBuilds, brief description of CogDroid based builds at INRIA Jenkins server [en línea]. Actualizada: agosto de 2012 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://code.google.com/p/squeakvm-tablet/wiki/JenkinsBuilds>

[25] LEJOS. Java for LEGO Mindstorms. Behavior programming [en línea]. Actualizada: noviembre de 2008. [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.lejos.org/nxt/nxj/tutorial/Behaviors/BehaviorProgramming.htm>

[26] Lily Project [en línea]. Actualizada: julio de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.lilyapp.org/>

[27] Maloney, John; Resnick, Mitchel; Rusk, Natalie; Silverman, Brian; Eastmond, Evelyn. The Scratch Programming Language and Environment. ACM Trans. Comput. Educ., Noviembre de 2010. Vol. 10, No. 4, Art 16. Disponible en Web: <http://web.media.mit.edu/~jmaloney/papers/ScratchLangAndEnvironment.pdf>

[28] Mayans Roca, Vicent. Desarrollo multiplataforma de aplicaciones de control y comunicación para robots móviles. Valera Fernández, Angel (Director). Proyecto de grado. Universitat Politécnica de Valencia, Escuela Técnica Superior de Ingeniería Informática, Valencia, 2012. Disponible en Web: <http://riunet.upv.es/bitstream/Memoria.pdf?sequence=1>

[29] Miranda, Eliot. Cog, Speeding up Croquet and Squeak with a new open-source VM from Teleplace [en línea]. [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.mirandabanda.org/cog/>

[30] Modernizr. Documentation [en línea]. Actualizada: abril de 2013. [Fecha de

consulta: julio de 2013]. Disponible en Web: <http://modernizr.com/docs/>

[31] Pinto Salamanca, María Luisa; Barrera Lombana, Nelson; Pérez Holguín, Wilson Javier. Uso de la robótica educativa como herramienta en los procesos de enseñanza. I^2+D. Oscar Oswaldo Rodríguez Díaz. 2010, Vol 10, No 1, p. 15-23. Disponible en Web: <http://virtual.uptc.edu.co/revistas2013f/index.php/ingenieria_sogamoso/article/download/912/912>

[32] Plan Ceibal. Robótica Educativa [en línea] [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.ceibal.org.uy/docs/robotica11.pdf>

[33] Processes and Threads. Android for Developers. Official Website [Fecha de consulta: julio de 2013]. Disponible en Web: <http://developer.android.com/processes-and-threads.html>

[34] Proyecto Butiá. Wiki Proyecto Butiá [en línea]. Actualizada: junio de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.fing.edu.uy/inco/proyectos/Butia/mediawiki/index.php>

[35] Quantcast. Scratch [en línea]. Actualizada: julio de 2013. [Fecha de consulta: julio de 2013]. Disponible en Web: <https://www.quantcast.com/scratch.mit.edu>

[36] Ramírez Toledo, Antonio. El constructivismo pedagógico [en línea]. [Veracruz, México] Universidad Veracruzana [Fecha de consulta: julio de 2013]. Disponible en Web: <http://ww2.educarchile.cl/UserFiles/P0001/File/El%20Constructivismo%20Pedag%C3%B3gico.pdf>

[37] Ruiz, Enrique; Sanchez, Velasco. Educatrónica: innovación en el aprendizaje de las ciencias y la tecnología. 1ra ed. Madrid: Ediciones Díaz de Santos, 2007.

[38] Seymour Papert. FactoriaHistorica [en línea]. Actualizada: noviembre de 2013. [Fecha de consulta: noviembre de 2013]. Disponible en Web: <http://factoriahistorica.wordpress.com/2013/11/19/seymour-papert/>

[39] Slany, Wolfgang. Catroid: A Mobile Visual Programming System for Children. En: "11th International Conference on Interaction Design and Children". [Graz, Austria]: Institute for Software Technology, Graz University of Technology, 2012. Disponible en Web: <http://arxiv.org/ftp/arxiv/papers/1204/1204.6411.pdf>

[40] Squeakvm-Tablet. Port of the Squeak Smalltalk VM to Android targeting Tablet devices [en línea]. Actualizada: agosto de 2012 [Fecha de consulta: julio de 2013]. Disponible en Web: <https://code.google.com/p/squeakvm-tablet/>

[41] Sugar Labs. Turtle Art. Wiki Sugar Labs [en línea]. Actualizada: julio de 2013. [Fecha de consulta: julio de 2013]. Disponible en Web: <http://wiki.sugarlabs.org/go/Activities/TurtleArt>

[42] Sugar en RaspberryPi utilizado mediante conexión remota VNC. Wiki Proyecto Butiá [en línea]. Actualizada: abril de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.fing.edu.uy/inco/proyectos/Butia/mediawiki/index.php>

[43] TortuBots. Wiki Proyecto Butiá [en línea]. Actualizada: junio de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.fing.edu.uy/inco/proyectos/Butiá/TortuBots>

[44] Turtle graphics [en línea]. Actualizada: julio de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://en.wikipedia.org/wiki/Turtle_graphics>

[45] Universal Serial Bus Spefication, Revision 2.0 [en línea]. Compaq. Hewlett-Packard. Intel. Lucent. Microsoft. NEC. Phillips. Actualizada: abril de 2000 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://sdphca.ucsd.edu/Lab_Equip_Manuals/usb_20.pdf>

[46] USB Host. Android for Developers. Official Website [Fecha de consulta: julio de 2013]. Disponible en Web: <http://developer.android.com/guide/topics/connectivity/usb/host.html>

[47] USB Host Diagnostics. Actualizada: julio de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://usbhost.chainfire.eu/>

[48] USB4All. Wiki Proyecto Butiá [en línea]. Actualizada: abril de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.fing.edu.uy/inco/proyectos/Butiá/Usb4all>

[49] USB4Butiá. Wiki Proyecto Butiá [en línea]. Actualizada: abril de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.fing.edu.uy/inco/proyectos/Butia/USB4butia>

[50] Visca, Jorge. Lumen [en línea]. Actualizada: febrero de 2014 [Fecha de consulta: febrero de 2014]. Disponible en Web: <https://github.com/xopxe/Lumen>

[51] Visca, Jorge. Toribio [en línea]. Actualizada: febrero de 2014 [Fecha de consulta: febrero de 2014]. Disponible en Web: <https://github.com/xopxe/Toribio>

[52] VNC. AT&T [en línea]. Actualizada: marzo de 2001 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.hep.phy.cam.ac.uk/vnc_docs/index.html>

[53] WebSocket. Browser support [en línea]. Actualizada: julio de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://en.wikipedia.org/wiki/WebSocket>

Apéndice A

Glosario

En esta sección se detallan y definen ciertos términos básicos para la comprensión del material presentado en este documento.

Ajax (*Asynchronous JavaScript and XML*): Es una técnica de desarrollo web que permite intercambiar información con un servidor web de forma asíncrona.

API (*Application Programming Interface*): Conjunto de funciones y procedimientos que ofrece una determinada biblioteca para ser usada por otro software manteniendo un nivel de abstracción.

ARM (*Advanced RISC Machine*): Arquitectura RISC de 32 bits desarrollada por la compañía ARM Holdings.

Client-Side: En un sistema, se denomina client-side al código que se ejecuta en el dispositivo cliente que accede a la aplicación.

CSS (*Cascading Style Sheets*): Lenguaje de estilo utilizado para describir la apariencia y el formato de un documento escrito en un lenguaje de marcas (HTML, XML, SVG).

DOM (*Document Object Model*): API que proporciona un conjunto de objetos para representar documentos HTML o XML. Un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

Dynamixel AX-12: Actuador de rotación (servo actuator en inglés) usado para robótica de mediano o pequeño porte, en particular éste tipo de motores funcionan como ruedas en el robot Butiá.

GET: Método definido en el protocolo HTTP que solicita información a un servidor web.

GUI (*Graphical User Interface*): Programa informático que funciona como interfaz de usuario, utilizando un conjunto de imagenes y objetos graficos para representar la información y acciones disponibles en la interfaz.

HDMI (*High-Definition Multimedia Interface*): Interfaz compacta de audio y video para la transferencia de datos entre una fuente de audio o video digital, como puede ser un ordenador, y un monitor compatible o televisor digital.

Hotspot: Dispositivos que dan soporte físico para la creación y mantenimiento de redes inalámbricas, generalmente a través de WiFi. Administrando la conexión y desconexión de dispositivos a la red.

HTTP (*HyperText Transfer Protocol*): Protocolo de comunicación de datos digitales perteneciente a la capa de aplicación. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor, usado en cada transacción de la *World Wide Web*.

IDE (*Integrated Development Environment*): Software compuesto por un conjunto de herramientas para la programación de sistemas. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios.

IP (*Internet Protocol*): Protocolo de comunicación de datos digitales perteneciente la capa de red, que permite transmitir datos a través de una red de distintas redes físicas previamente enlazadas. En general se utiliza el término, dirección IP, para referir a el número que identifica un nodo dentro de una red que utilice el protocolo IP.

JavaScript: Lenguaje de programación interpretado, se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Comúnmente utilizado para agregar funcionalidad y estilos de una página web en client-side.

JSON (*JavaScript Object Notation*): Formato ligero para el intercambio de datos. Consiste en pares de clave y valor en formato de texto.

JQuery: Biblioteca de JavaScript que permite simplificar la manera de interactuar con los documento HTML, manipular el árbol DOM, desarrollar animaciones, entre otras cosas.

Plug&Play: Término usado para definir dispositivos informáticos con la característica de reconocer una componente de hardware sin requerir configuraciones previas o reinicios de sus sistemas.

POST: Método definido en el protocolo HTTP que envía información a un servidor web para que sea procesada por este.

RAM (*Random-Access Memory*): Memoria volátil de trabajo para el sistema operativo encargada de almacenar todas las instrucciones que ejecuta el procesador y otras unidades de cómputo.

RJ45 (*Registred Jack 45*): Interfaz física usada comúnmente para conectar redes cableadas, como cables de red Ethernet.
SBC (*Single Board Computer*): Computadora completa con un sólo circuito. Se centra en un solo microprocesador con RAM, E/S y el resto de las características funcionales de un computador en una sola placa de tamaño reducido.

SDK (*Software Development Kit*): Conjunto de herramientas para el desarrollo de software que le permite al programador aplicaciones para un sistema en concreto.

Server-Side: En un sistema, se denomina server-side al código que se ejecuta en el servidor de la aplicación y no en el dispositivo que accede a este.

Smalltalk: Lenguaje reflexivo de programación, orientado a objetos y con tipado dinámico.

Squeak: Lenguaje de programación o dialecto dentro de Smalltalk.

Sugar: Interfaz gráfica de usuario, libre y de código abierto, desarrollada y diseñada para el proyecto OLPC. Enfocada para ofrecer a niños un entorno de aprendizaje interactivo.

TCP (*Transmission Control Protocol*): Protocolo de comunicación de datos digitales perteneciente a la capa de transporte, orientado a la comunicación fiable.

USB (*Universal Serial Bus*): Estándar de la industria que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre ordenadores y periféricos.

WiFi: Sistema de comunicación de datos inalámbrico, que utiliza ondas electromagnéticas con una cierta modulación para intercambiar mensajes entre equipos, sin necesidad de una conexión física directa entre éstos.

WLAN (*Wireless Local Area Network*): Sistema de comunicación inalámbrico flexible. Frecuentemente utilizada como alternativa a las LAN cableadas o como extensiones de éstas.

WPA (*WiFi Protected Access*): Sistema para proteger las redes inalámbricas.

XML (*Extensible Markup Language*): Lenguaje de marcas desarrollado por la W3C utilizado para almacenar datos de forma legible.

Proyecto de Grado

IDE Android para la Programación de Comportamientos Robóticos

Arquitectura y diseño del sistema

Andrés Nebel - Renzo Rozza

Tutores
Ing. Andrés Aguirre, Ing. Rafael Sisto

24 de Abril del 2014

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay

Índice

1. Introducción
2. Vista de casos de uso
    2.1. Actores
    2.2. Casos de uso
        2.2.1. Crear comportamiento robótico
        2.2.2. Eliminar comportamiento robótico
        2.2.3. Agregar bloque de un comportamiento robótico
        2.2.4. Eliminar bloque de un comportamiento robótico
        2.2.5. Editar comportamiento
        2.2.6. Calibrar plataforma robótica
        2.2.7. Ver código
        2.2.8. Ejecutar/Detener
        2.2.9. Ejecutar en modo Debug
        2.2.10. Eliminar comportamientos
        2.2.11. Descargar comportamientos
        2.2.12. Cargar proyecto desde la base de datos
        2.2.13. Cargar desde archivo local
        2.2.14. Ejecución rápida de función de Sensor/Actuador
    2.3. Casos de uso críticos
3. Vista lógica
    3.1. Estilo arquitectónico
    3.2. Subsistemas
        3.2.1. Descripción
4. Vista de distribución
    4.1. Escenario
        4.1.1. Descripción
        4.1.2. Nodos
        4.1.3. Conexiones
5. Diagrama de clases
6. Diagramas de comunicación
    6.1. Crear comportamiento
    6.2. Task switching
    6.3. Refrescar dispositivos

1. Introducción

El proyecto consiste en la realización de un entorno de desarrollo (IDE) para la implementación de comportamientos robóticos que modele un lenguaje de programación visual (LPV). El mismo, debe ser soportado por el sistema operativo Android y permitir a personas de corta edad poder programar tareas que controlen a la plataforma robótica Butiá utilizando una arquitectura perteneciente al paradigma reactivo. Los usuarios acceden al sistema mediante una interfaz web, donde programan los comportamientos que luego serán ejecutados y enviados al robot por el servidor web alojado en la SBC.

2. Vista de casos de uso

2.1. Actores

Si bien se pueden destacar dos usuarios como se explicó en el documento de especificación de requerimientos, el sistema posee un único actor. Todos los casos de uso están orientados a este, si bien algunos como la calibración del robot pueden necesitar la colaboración de un docente para ayudar al estudiante, tanto los estudiantes como los docentes son considerados el mismo actor en el sistema, pues comparten todos los casos de uso.

2.2. Casos de uso

2.2.1. Crear comportamiento robótico

Los usuarios podrán crear un comportamiento al agregar un bloque específico para ello, donde deberán especificar el nombre del comportamiento creado y la prioridad del mismo.

2.2.2. Eliminar comportamiento robótico

Los usuarios podrán arrastrar hacia afuera del workspace un bloque de comportamiento y así eliminarlo del proyecto. De esta manera se descarta el comportamiento y toda la lógica que el mismo contenía.

2.2.3. Agregar bloque de un comportamiento robótico

Los usuarios podrán agregar bloques que representan instrucciones lógicas, de ahora en más bloques de instrucción, a los distintos comportamientos existentes en el workspace. Además, este caso de uso involucra un auto salvado del comportamiento en el servidor Web.

2.2.4. Eliminar bloque de un comportamiento robótico

Los usuarios podrán eliminar bloques de instrucción de un comportamiento existente en el workspace. Al igual que agregar bloque de un comportamiento robótico este caso de uso involucra un auto salvado del comportamiento.

2.2.5. Editar comportamiento

Un comportamiento puede ser marcado como listo por los usuarios, para quitar a este comportamiento del workspace y agregarlo a una lista de comportamientos listos para ser ejecutados. A su vez, el usuario puede editar cualquier comportamiento perteneciente a la lista de comportamientos listos, quitando al comportamiento seleccionado de esa lista e incluyendo nuevamente en el workspace.

2.2.6. Calibrar plataforma robótica

Los usuarios podrán calibrar mediante bloques específicos los sensores de la plataforma robótica, lo cual le permitirá definir las variables que podrán ser utilizadas en un momento determinado para controlar al robot Butiá.

2.2.7. Ver código

Los usuarios podrán acceder al código generado a partir de un comportamiento determinado, el código se desplegará en lugar del workspace que contiene los bloques de ese comportamiento.

2.2.8. Ejecutar/Detener

Los usuarios podrán ejecutar un conjunto de comportamientos ya programados y detener la ejecución mientras los comportamientos estén ejecutando.

2.2.9. Ejecutar en modo Debug

Los usuarios podrán debuggear un conjunto de comportamientos ya programados, permitiendo al mismo ver que comportamiento se está ejecutando en un determinado momento, y más específicamente qué instrucción lógica se encuentra actualmente en ejecución a través de los bloques.

2.2.10. Eliminar comportamientos

Los usuarios podrán eliminar la totalidad de los comportamientos programados, ya sean los agregados a la lista de comportamientos listos como el comportamiento que se encuentra en desarrollo en el workspace.

2.2.11. Descargar comportamientos

Los usuarios podrán descargar a un archivo localmente los comportamientos desarrollados, ya sea los desarrollados en su proyecto por medio del dispositivo móvil o todos los desarrollados por los distintos proyectos que interactuaron con el robot Butiá recientemente.

2.2.12. Cargar proyecto desde la base de datos

Los usuarios podrán cargar proyectos desde la base de datos alojada en la SBC, la cual contendrá los proyectos almacenados recientemente y les permitirá cargar cualquier proyecto, no sólo los desarrollados por ellos mismos.

2.2.13. Cargar desde archivo local

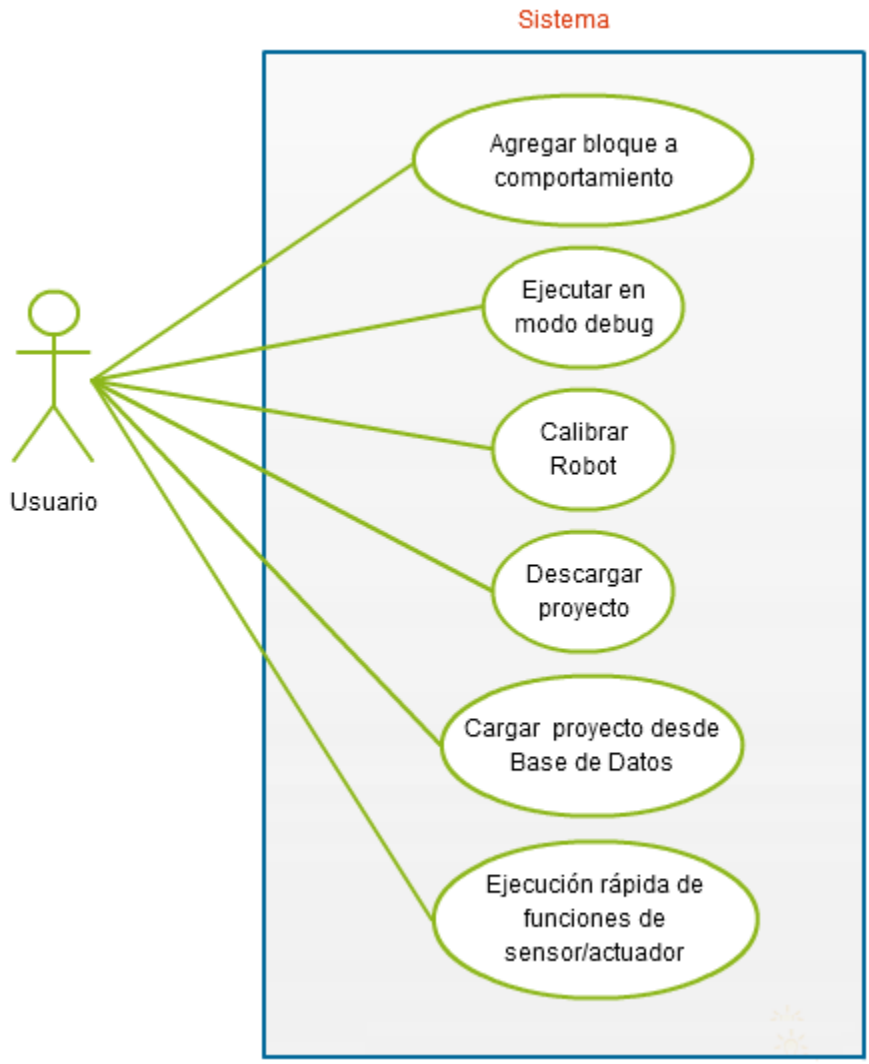
Los usuarios podrán cargar un proyecto alojado en su dispositivo móvil, recuperando todos los comportamientos almacenados en ese proyecto.

2.2.14. Ejecución rápida de función de Sensor/Actuador

Los usuarios podrán ejecutar de manera rápida, sin la necesidad de crear un comportamiento, una función de un actuador o un sensor. Esto sirve a los efectos de la calibración y chequeo de los sensores/actuadores.

2.3. Casos de uso críticos

Se identificaron 5 casos de uso críticos.



2.3.1 Calibrar

Descripción: Este caso de uso contempla la calibración por parte del usuario de los sensores del Robot. El usuario elige un sensor que haya sido reconocido por Bobot especifica una función (operación aritmética) a aplicar a los valores sensados para convertirlos a valores ya definidos en el sistema, como ser por ejemplo un rango de 0 a 100.

Precondiciones: Deben existir sensores conectados y activos para calibrar.

Postcondiciones: A cada valor medido por el sensor calibrado le será aplicada una función antes de ser manejado por el comportamiento programado por el usuario.

Flujo principal:

1- El usuario arrastra el bloque “Calibrar” al workspace.
2- El usuario elige un bloque de función de un sensor y lo arrastra dentro del bloque “Calibrar”.
3- El usuario agrega un operador aritmético al bloque de función del paso 2.
4- El sistema disponibiliza un bloque para ser usado por el usuario que representa el valor que surge de aplicar el operador aritmético a un valor sensado.

2.3.2 Agregar bloque a comportamiento

Descripción: Este caso de uso surge cuando el usuario quiere agregar lógica a un bloque de comportamiento creado. El sistema añade el bloque de lógica al comportamiento y guarda los cambios en el servidor, de forma de mantener almacenada la información del usuario de manera transparente.

Precondiciones: Debe existir un comportamiento creado y actualmente en edición.

Postcondiciones: El nuevo bloque queda asociado al comportamiento. El sistema almacena los cambios realizados.

Flujo Principal

1- El usuario selecciona un bloque de lógica y lo arrastra hasta el bloque de comportamiento. (FE1)

2- El sistema adjunta el nuevo bloque al bloque de comportamiento.

3- El sistema almacena los cambios realizados.

Flujos de Excepción

FE1:
1- El sistema no acopla el bloque al comportamiento, en caso de que no encastre en la posición sugerida por el usuario. El bloque queda por fuera del comportamiento en el workspace.

2.3.3 Ejecución rápida de función de sensor/actuador

Descripción: Este caso de uso surge como complemento a la calibración y contempla el caso en que el usuario desea ejecutar una función de un sensor o actuador sin la necesidad de crear un comportamiento. El caso de uso permite conocer el estado actual de lo que está midiendo un sensor, así como también probar un actuador, etc.

Precondiciones: Deben existir sensores o actuadores conectados y activos.

Flujo Principal

1- El usuario selecciona la pestaña "Ejecución Rápida".

2- El usuario arrastra un bloque de función de sensor o actuador hasta el workspace. (FE1)

3- El usuario presiona ejecutar.

4- El sistema invoca la función correspondiente del sensor o actuador desplegando en pantalla, si corresponde el valor retornado por la función.

Flujos de Excepción

FE1:
1- El usuario intenta arrastrar un bloque que no es una función de un sensor/actuador.
2- El sistema no permite que el bloque se añada al workspace sin desplegar mensaje de error.

2.3.4 Ejecutar en modo Debug

Descripción: Este caso de uso sucede cuando un usuario programó comportamientos y decide ejecutarlos en modo debug. El sistema ejecuta los comportamientos siguiendo la arquitectura robótica reactiva establecida y le despliega al usuario que comportamiento se encuentra actualmente en ejecución y más específicamente qué instrucción lógica de dicho comportamiento se está ejecutando.

Precondiciones: Debe existir al menos un comportamiento creado.

Flujo Principal

1- El usuario presiona el botón "Debug".

2- El sistema despliega los comportamientos creados por el usuario en una pantalla independiente al workspace.

3- El sistema comienza la ejecución del primer (o el siguiente) bloque de instrucción de un comportamiento robótico.

4- El sistema resalta en pantalla el bloque de instrucción que se encuentra en procesamiento (FA1).

5- El usuario detiene la ejecución.

6- El sistema regresa a la pantalla principal.

Flujos Alternativos

FA1:

1- Vuelve al paso 3 hasta que finalice la ejecución del sistema.

Flujos de Excepción

FE1:
1- El usuario intenta ejecutar en modo debug y no existen comportamientos a ejecutar.
2- El sistema no efectúa ninguna acción.

2.3.5 Descargar Proyecto

Descripción: Este caso de uso ocurre cuando un usuario desea descargar en forma de archivo los comportamientos creados por él o todos los comportamientos que han sido creados por el grupo de trabajo.

Precondiciones: Debe existir al menos un comportamiento creado por el usuario (si selecciona descargar solo sus comportamientos) o por el grupo de trabajo (si selecciona descargar todos los comportamientos creados por el grupo de trabajo). Debe existir espacio de almacenamiento suficiente en el dispositivo cliente.

Postcondiciones: El sistema genera un archivo con información de los comportamientos y lo disponibiliza para descargar.

Flujo Principal

1- El usuario selecciona el botón “Descargar”.
2- El sistema disponibiliza dos opciones para que el usuario seleccione una. Estas son:
“Descargar solo mis comportamientos“, y “Descargar comportamientos del grupo de trabajo”.
3- El usuario selecciona “Descargar solo mis comportamientos”. (FA1)
4- El sistema disponibiliza un archivo para descargar que contiene la información de los comportamientos desarrollados por el usuario.

Flujos Alternativos

FA1:
1- El usuario selecciona “Descargar comportamientos del grupo de trabajo”. (FA1)
2- El sistema disponibiliza un archivo para descargar que contiene la información de los comportamientos desarrollados por cualquiera de los usuarios pertenecientes al mismo grupo de trabajo que el usuario.

2.3.6 Cargar proyecto desde base de datos

Descripción: Este caso de uso surge a partir de la ventaja otorgada por el sistema que almacena en la base de datos los proyectos realizados por los distintos usuarios. El usuario puede desde la aplicación Web elegir entre los proyectos almacenados en la base y cargar los comportamientos desarrollados en esa instancia.

Precondiciones: Debe existir por lo menos un proyecto almacenado en la base de datos.

Postcondiciones: El sistema despliega los comportamientos contenidos en el proyecto cargado por el usuario.

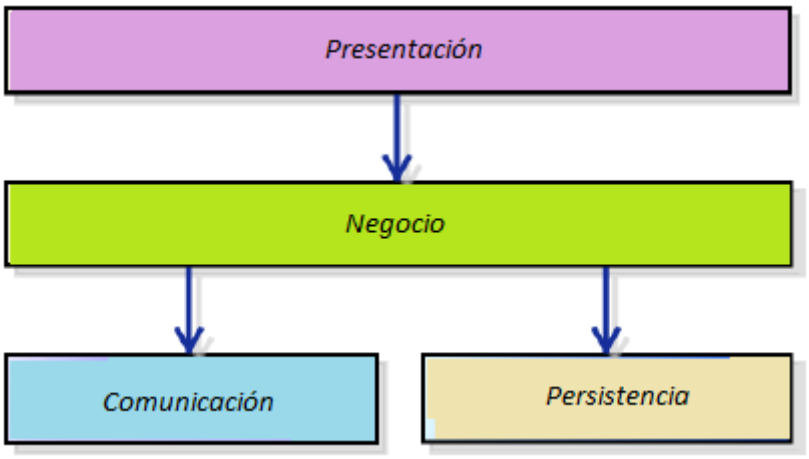
Flujo principal:

1- El usuario presiona el botón “Cargar”.
2- El usuario selecciona el proyecto que desea cargar, entre los proyectos almacenados en la base de datos.
3- El sistema carga los comportamientos almacenados en el proyecto seleccionado.

3. Vista lógica

3.1. Estilo arquitectónico

El sistema presenta una arquitectura distribuida en 4 capas:



Presentación: Refiere a la capa de interfaz gráfica. Es la parte del sistema encargada de interactuar con el usuario y de la presentación visual.

Negocio: Esta capa es la que posee la lógica principal del sistema y la que maneja y ejecuta los comportamientos según la arquitectura robótica reactiva definida.

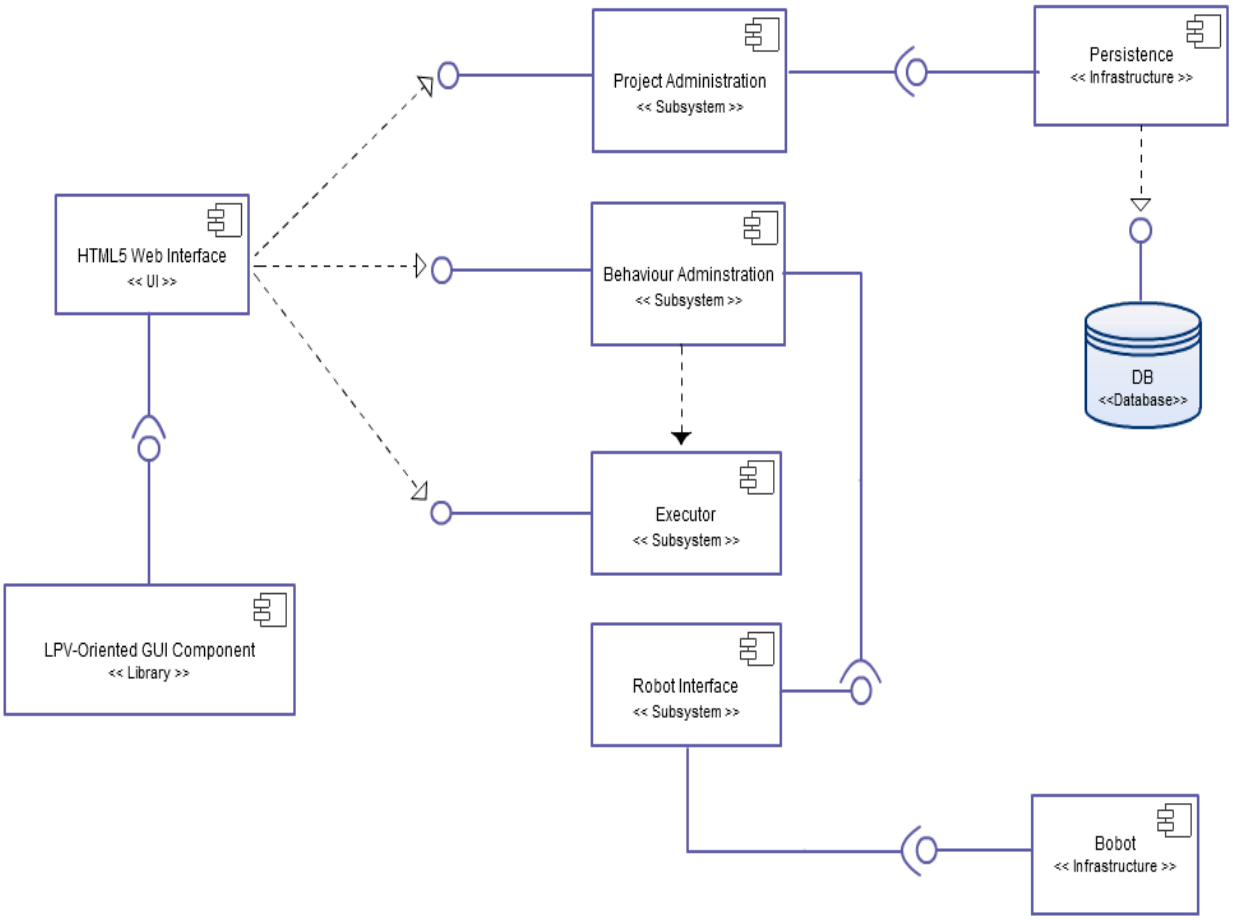
Comunicación: Es la encargada de la comunicación entre los módulos de la capa de negocio y los sensores y actuadores del kit robótico.

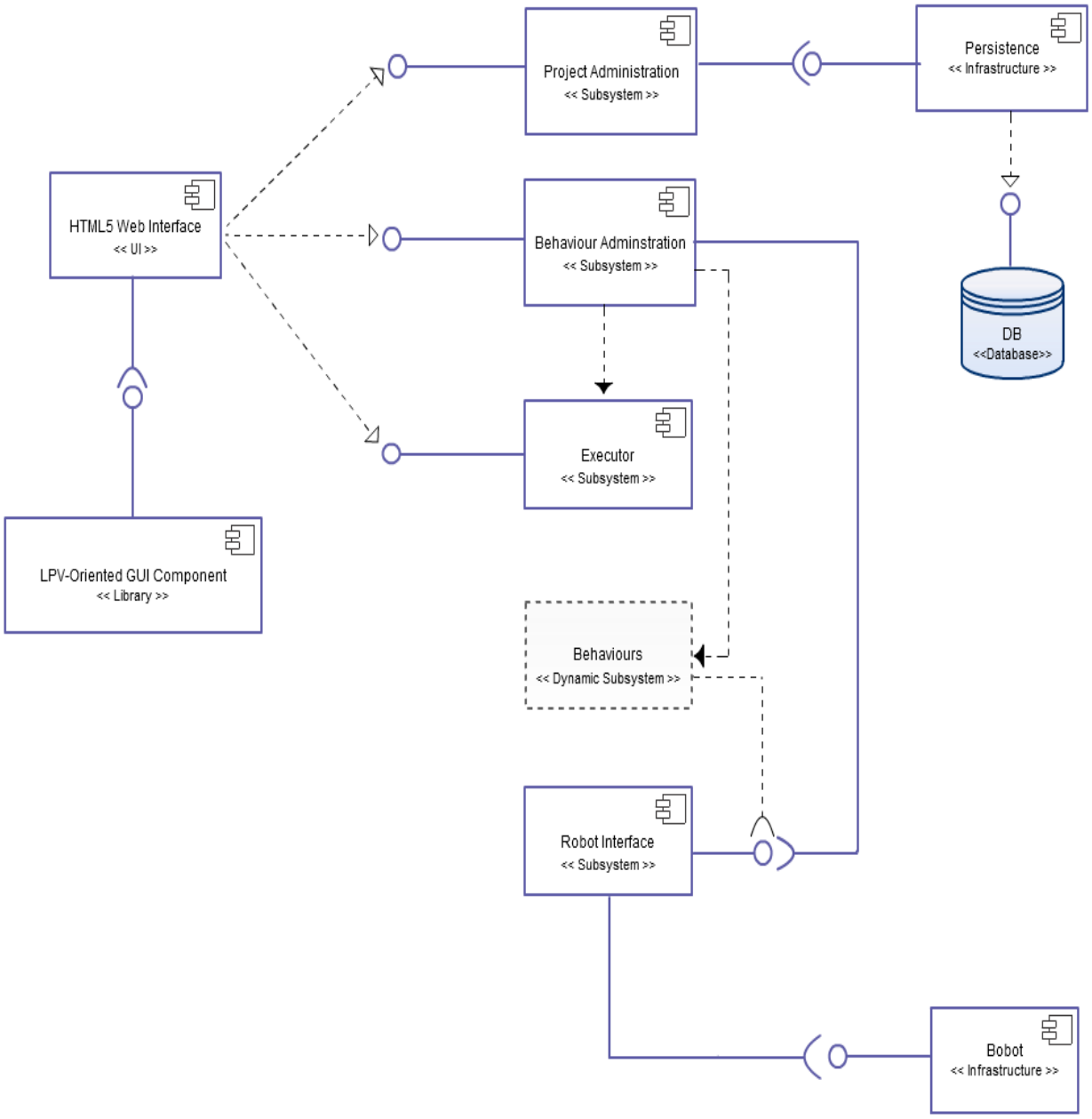
Persistencia: Es la capa que posee como función la administración y el almacenamiento de los comportamientos y proyectos en la base de datos.

3.2. Subsistemas

Es interesante que para los subsistemas presentemos dos diagramas distintos, ya que los comportamientos creados por el usuario se generan y ejecutan de manera dinámica por lo cual no son módulos definidos de la arquitectura y sin embargo al generarse forman parte importante del

sistema. Por lo tanto mostraremos dos diagramas de componentes, uno incluirá simplemente los subsistemas estáticos del sistema y el otro incluirá dentro de la arquitectura como se integran los comportamientos generados dinámicamente.





3.2.1. Descripción

1. HTML5 Web Interface

La interfaz de usuario del sistema es responsable de la gestión de la interacción por medio de la Web, siendo parte de la capa de presentación. Esta encargada de presentar el sistema al usuario, comunicar la información y capturar la información ingresada por este. Esta capa se comunica únicamente con la capa de negocio, y consume las prestaciones de la librería LPV-Oriented.

2. LPV-Oriented GUI Component

Librería que modela un lenguaje de programación visual (LPV) y brinda sus prestaciones para el uso por parte de la componente de interfaz de usuario del sistema.

3. Project Administration

Subsistema encargado de la administración de proyectos incluido en la capa de negocios, el mismo provee a la capa de presentación sus servicios y consume las prestaciones de la capa de persistencia.

4. Behaviour Administration

Subsistema encargado de la administración de comportamientos incluido en la capa de negocios, este subsistema modela la arquitectura robótica basada en prioridades. El mismo provee a la capa de presentación sus servicios y consume los servicios de la interfaz robótica para enviar señales a la plataforma robótica.

5. Executor

Subsistema encargado de crear y ejecutar en Toribio los comportamientos robóticos incluidos en la capa de negocio. Es considerado con el fin de proveer a la capa de presentación y convertir los comportamientos creados en base al LPV en tareas de Toribio.

6. Robot Interface

Este subsistema está diseñado para generalizar la comunicación con interfaces robóticas, el mismo encapsula las prestaciones de Bobot y brinda sus servicios para controlar los sensores y actuadores de plataformas robóticas.

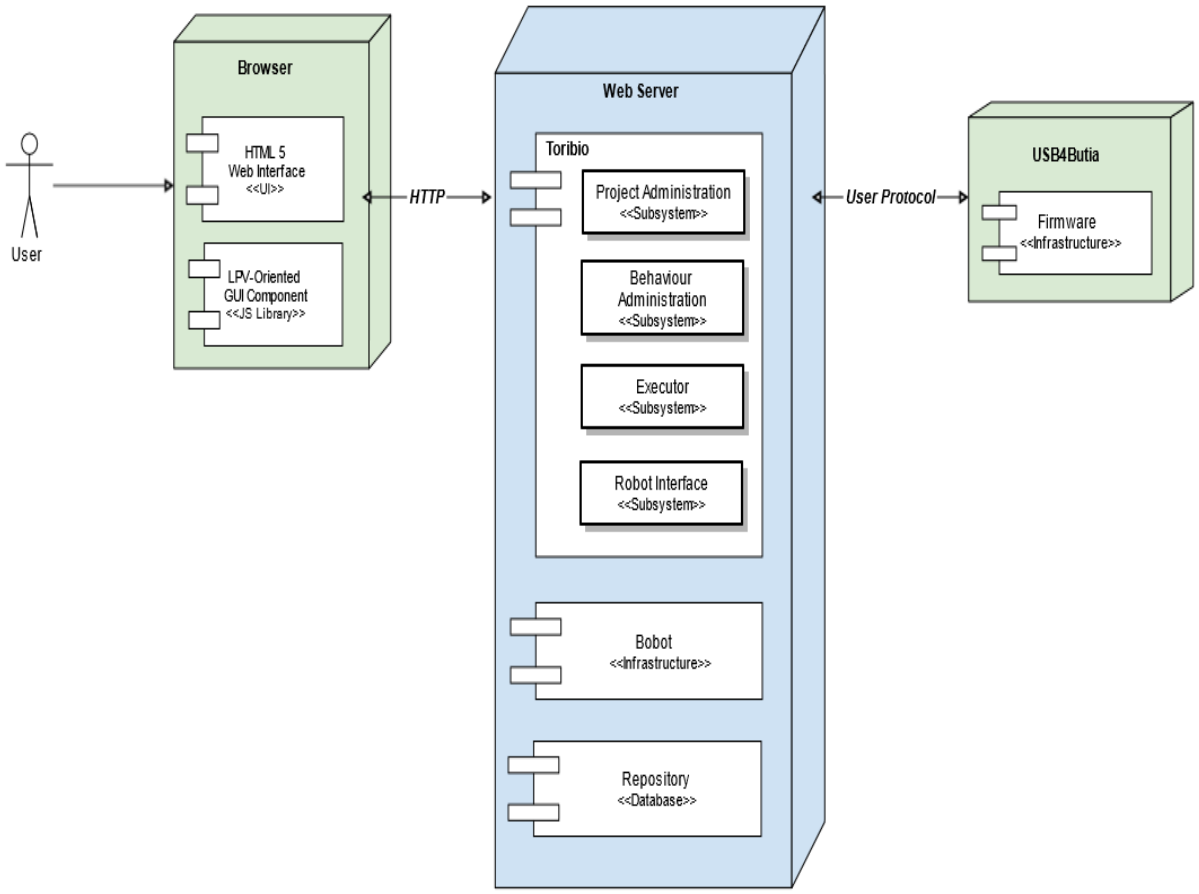
7. Bobot

Infraestructura encargada de realizar la comunicación a bajo nivel con la placa USB4Butiá y de exponer una interfaz a través de una conexión TCP/IP para controlar los sensores y actuadores del robot Butiá.

8. Persistence

Infraestructura encargada de realizar la comunicación con la base de datos del sistema, la misma provee a la capa de negocio sus servicios.

4. Vista de distribución



4.1. Escenario

4.1.1. Descripción

Se presenta aquí la arquitectura técnica del sistema indicando los nodos presentados en la infraestructura tecnológica esperada, y la localización de los componentes en dichos nodos. Considerando la distribución de la aplicación desde el punto de vista de los procesos, es posible identificar tres tipos de nodos; Browser, Web Server, y USB4Butiá.

4.1.2. Nodos

1. Browser

Este nodo representa el entorno para los usuarios finales, siendo presentado el sistema a través de un navegador con la habilitación para ejecutar JavaScript.

2. Web Server

Este nodo centraliza todos los requerimientos funcionales del sistema, soportado por el Framework Toribio quien brinda los servicios de un servidor web.

3. USB4Butiá

Este nodo centraliza la comunicación con los actuadores y sensores, brindando el acceso para controlar de la plataforma robótica Butiá.

4.1.3. Conexiones

La comunicación establecida entre el nodo Browser y el nodo Web Server es mantenida a través de HTTP, mientras que la comunicación que establece el nodo Web Server con el nodo USB4Butiá es a través de un protocolo llamado User Protocol.

5. Diagrama de clases

El sistema fue desarrollado en los lenguajes Lua para la parte server-side y Javascript para la parte client-side. Como ambos son lenguajes de scripting que no están orientados a objetos, el concepto de clase carece de sentido en ambos. Sin embargo, para la parte server-side, se puede especificar un diagrama que detalle los subsistemas, ya que estos están compuestos por archivos .lua y cada uno de estos por una única tabla con funciones y atributos que se asemeja bastante al concepto de clase. Son estos y sus interdependencias los que se diagraman a continuación:

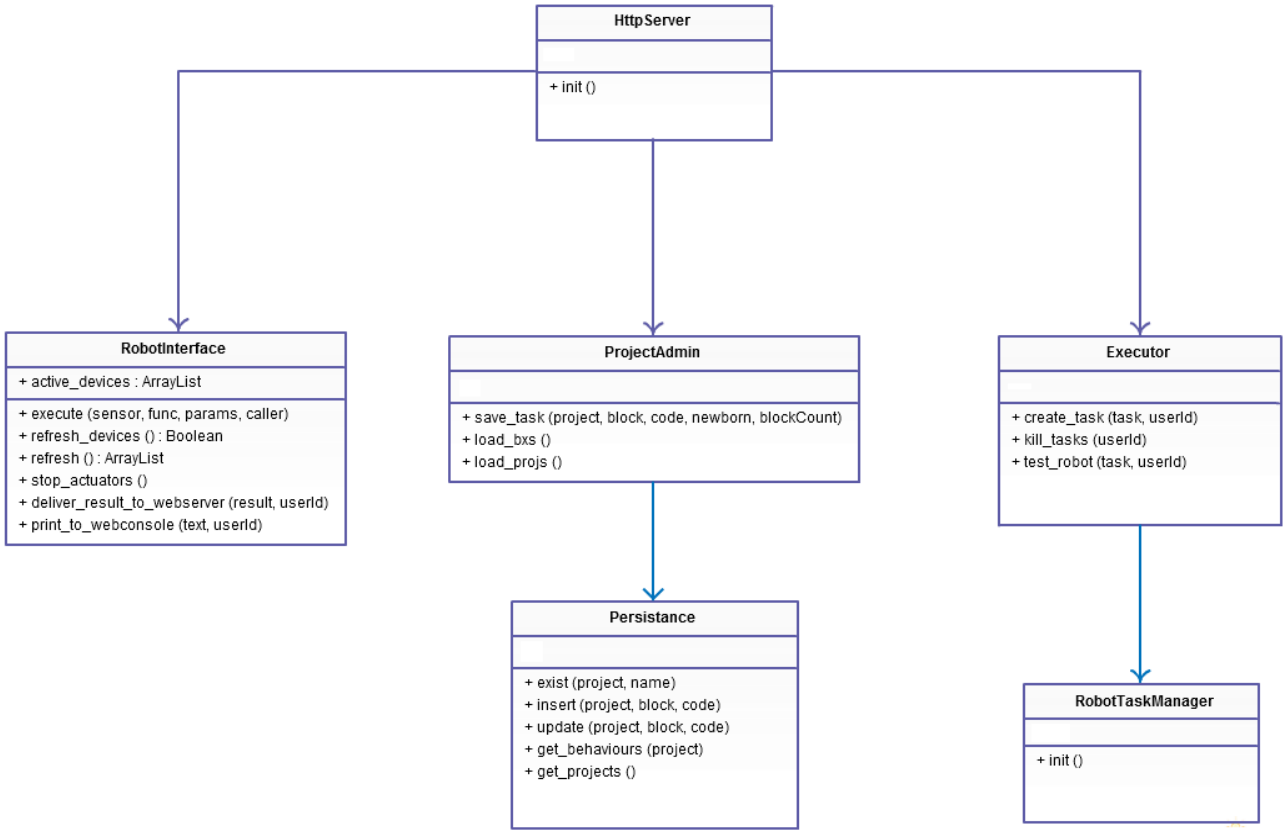


Diagrama de clases server-side.

Si bien los comportamientos son generados dinámicamente, estos tienen una estructura predeterminada, con funciones y atributos predefinidos. Solo variará su función *run()* y *compete_for_active()* cuando el usuario edite con Blockly la acción y disparador del

comportamiento. Es por esto que es interesante ver también su diagrama de clase de forma independiente.

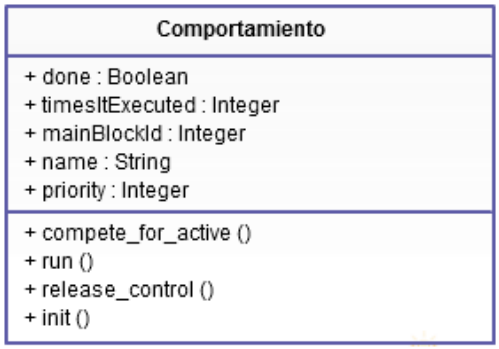


Diagrama de clase de un comportamiento genérico.

6. Diagramas de comunicación

En esta sección se muestran diagramas de comunicación de algunas funciones cuya interacción entre componentes es destacable. Como fue mencionado, si bien estas anotaciones UML son orientadas a la programación orientada a objetos, se despliegan los diagramas con la intención de ayudar al lector en la comprensión del sistema.

6.1. Crear comportamiento

El primer diagrama representa la interacción entre componentes para la creación de un comportamiento. Este es enviado desde el dispositivo cliente, en forma de POST con código Lua en formato de texto. La generación de este código Lua se hace desde código Javascript que extiende a Blockly. Éste código Lua es parseado y ejecutado, el comportamiento es agregado a al catálogo de Lumen y el administrador de comportamientos *RobotTaskManager* es activado y comienza a ejecutar de acuerdo a la arquitectura basada en prioridades.

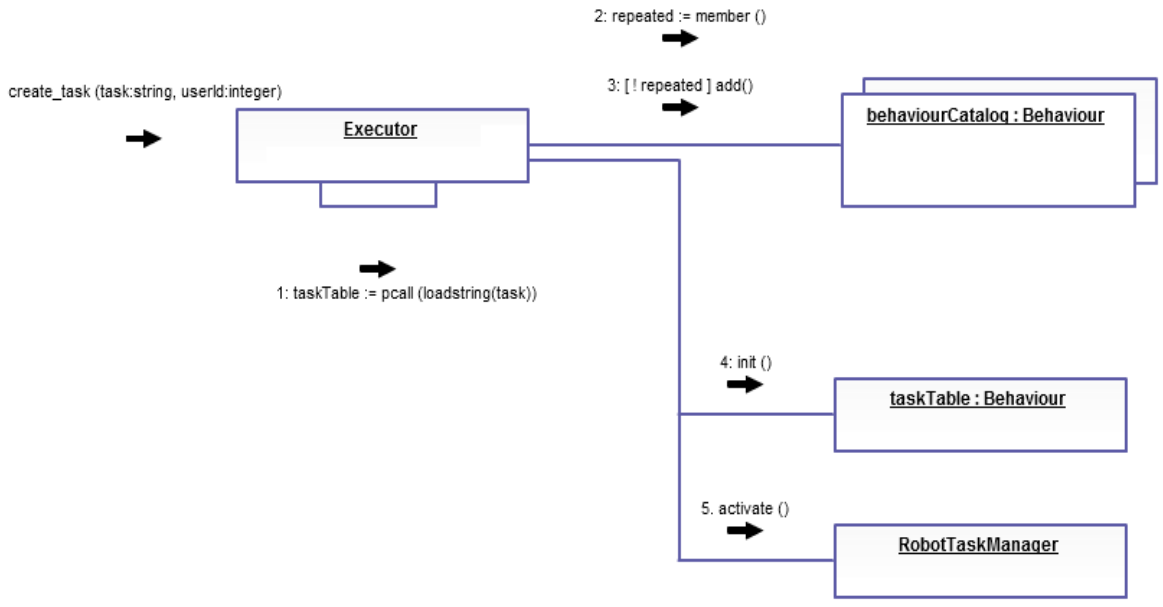


Diagrama de comunicación para ejecutar comportamiento.

6.2. Task switching

El segundo diagrama corresponde al código ejecutado en una iteración del *RobotTaskManager*. Éste una vez activado entra en un loop infinito en donde cada vez llama a competir a los comportamientos por el lugar de comportamiento activo. Una vez que estos compiten ejecuta la acción del comportamiento que quedó como activo, deteniendo la ejecución del comportamiento anterior (si es distinto al nuevo activo).

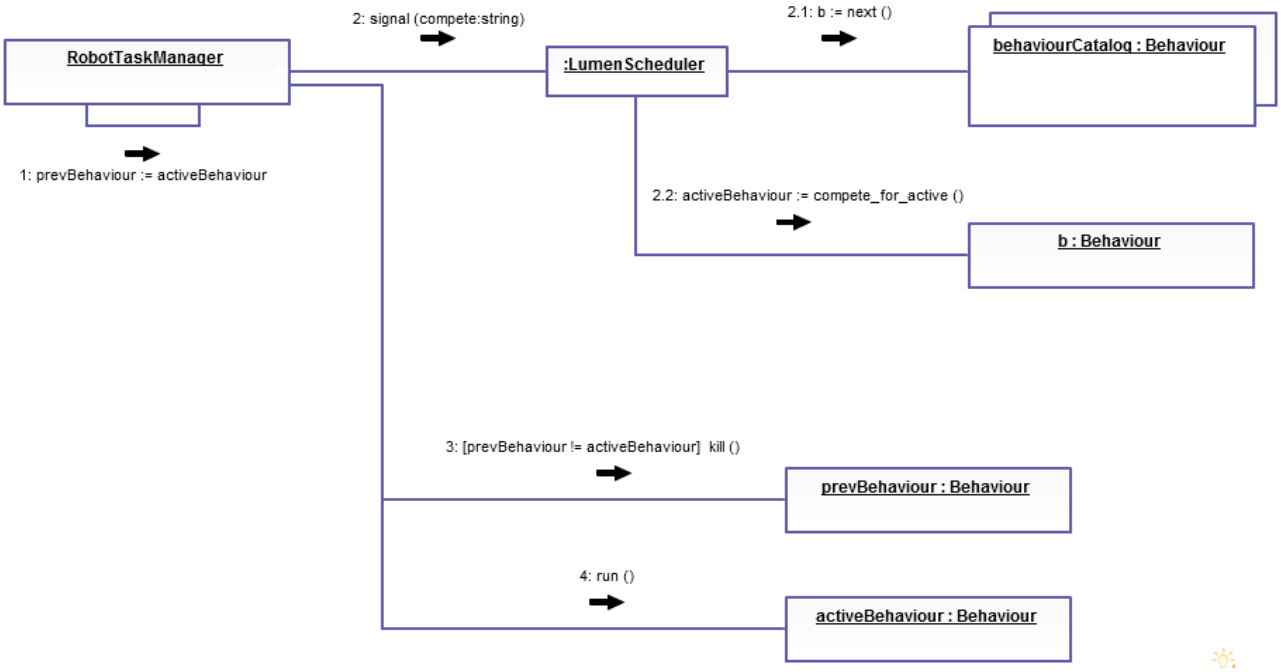


Diagrama de comunicación para el task switching.

6.3. Refrescar dispositivos

El diagrama de la figura 26 corresponde al código que se ejecuta cada 5 segundos y que consulta si ha habido cambios en los dispositivos conectados. Para esto se consulta a la interfaz robótica que a su vez consulta a Toribio cuales son los devices conectados. Luego compara estos devices con los últimos registrados y si hubo modificaciones entonces regenera los archivos de definición de bloques de Butiá y de generación de código de estos. Por último notifica retornando verdadero que se debe recargar la página. Esto es necesario para regenerar la toolbox de Blockly y marcar deshabilitados los bloques de dispositivos que ya no existen más.

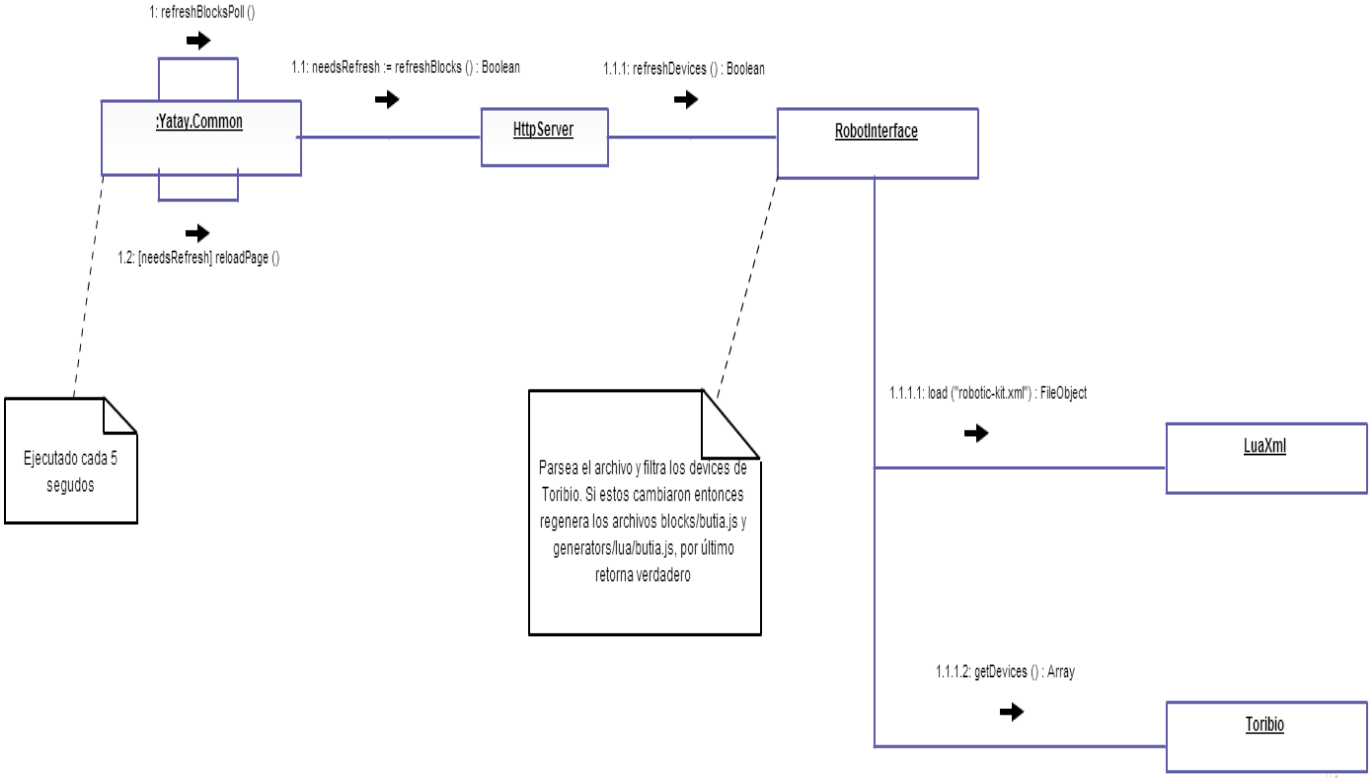


Figura 26: Diagrama de comunicación para refrescar los bloques de dispositivos conectados.

Proyecto de Grado

IDE Android para la Programación de Comportamientos Robóticos

Especificación de Requerimientos

Andrés Nebel - Renzo Rozza

Tutores
Ing. Andrés Aguirre, Ing. Rafael Sisto

24 de Abril del 2014

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay

Índice

1. Introducción
    1.1 Resumen
    1.2 Propósito
    1.3 Alcance
    1.4 Visión general
2. Descripción general
    2.1 Perspectiva del producto
        2.1.1. Interfaces de usuario
        2.1.2. Interfaces con hardware
        2.1.3. Interfaces con software
        2.1.4. Restricciones de memoria
    2.2 Usuarios
    2.3 Características de los usuarios
3. Requerimientos funcionales
    3.1 Entorno y Ejecución
    3.3 Proyectos
    3.4 Comportamientos
4. Requerimientos no funcionales
5. Requerimientos de documentación

1. Introducción

1.1 Resumen

En este documento se pretende dar una visión global sobre los requerimientos del proyecto propuesto, brindando un acercamiento a la descripción del sistema y detallando el alcance del mismo según lo relevado en el documento de presentación de proyecto y en las reuniones de relevamiento de requerimientos con los tutores del proyecto, además de lo definido en el análisis del estado del arte.

1.2 Propósito

Establecer un acercamiento a los requerimientos del sistema, los que serán confirmados y validados en futuras reuniones a realizar. De esta manera, se logra obtener una primera guía para ser utilizada en el diseño de la solución del sistema a construir. También será utilizado como herramienta de apoyo a la validación de la solución.

1.3 Alcance

El proyecto consiste en la realización de un entorno de desarrollo (IDE) para la implementación de comportamientos robóticos que modele un lenguaje de programación visual (LPV). El mismo, debe ser soportado por el sistema operativo Android y permitir a personas de corta edad poder programar tareas que controlen a la plataforma robótica Butiá utilizando una arquitectura perteneciente al paradigma reactivo.

La metodología de desarrollo de los comportamientos debe ser mediante la construcción de figuras de bloques que representen con simplicidad código de programación robótica. Dicha funcionalidad, deberá estar inspirada en sistemas como TortuBots y Scratch. Dentro de los dispositivos con sistema operativo Android estará principalmente orientado a las Tablets.

El sistema debe permitir ejecutar los comportamientos implementados y poseer la funcionalidad de debugging, en el sentido de que debe ilustrar al usuario en tiempo real que parte del código está ejecutando el Robot Butiá.

1.4 Visión general

A continuación en este documento puede encontrarse una descripción general del sistema, los requerimientos funcionales y no funcionales, así como también los requerimientos de documentación

2. Descripción general

El objetivo general de este sistema radica en la construcción de una nueva herramienta de software para uso escolar la cuál permita a los alumnos complementar su aprendizaje por medio de la programación de comportamientos robóticos. El mismo se modelará en base a las experiencias obtenidas con las actividades integradoras del proyecto Butiá y con el uso de TortuBots.

2.1 Perspectiva del producto

2.1.1. Interfaces de usuario

Ver Documento "Pautas para la interfaz de Usuario"

2.1.2. Interfaces con hardware

Se debe contar con un dispositivo con sistema operativo Android, que posea las características de tamaño y resolución de pantalla que posee una Tablet estándar que asumiremos como 800 x 480 píxeles y entre 7” y 10” de pantalla.

Se dispondrá de una SBC con características similares a la de una Raspberry PI y una controladora USB4Butiá para la cual se utilizará la aplicación Bobot como medio de disponibilizar sus funcionalidades a nuestro sistema.

2.1.3. Interfaces con software

Se destacan las interfaces de Toribio y Lumen para la administración de la ejecución de los comportamientos además del servidor web, y la utilización de la aplicación Bobot.

2.1.4. Restricciones de memoria

Las Tablets y los dispositivos que corren Android en general suelen ser artefactos de poca memoria RAM así como poca capacidad de procesamiento. Es importante que el sistema a desarrollar contemple dichas restricciones en su implementación.

2.2 Usuarios

Estudiantes:

Los usuarios a los que apunta esta aplicación son estudiantes que en general ya tuvieron sus primeras experiencias de desarrollo con TortuBots, o que han utilizado Scratch, por lo que se

puede asumir que conocen al robot Butiá y a la modalidad de construcción basada en bloques. Aunque esto no fuera así, la participación del docente capacitado permitirá al alumno resolver los distintos cuestionamientos que surjan. Los estudiantes suelen desarrollar sus aplicaciones en grupos, pensando en conjunto, desde el armado del robot, hasta la programación del comportamiento que resuelve la problemática.

Docentes:

Los estudiantes tendrán como ayuda una guía docente, pero en general la experiencia de los mismos no es mayor a la de los estudiantes. Esto es debido a que también dieron sus primeros pasos tanto en programación, como en robótica, con TortuBots. Los docentes sin embargo corren con la ventaja de haber tenido una capacitación básica por parte de docentes de Robótica que les permite resolver las problemáticas que escapan a la orientación trivial de la aplicación. Ambos tipos de usuarios utilizarán el sistema de la misma forma sin existir funcionalidades, vistas o privilegios distintos.

2.3 Características de los usuarios

Las características expuestas a continuación describen a los usuarios en su mayoría, pero no en su totalidad, ya que el conjunto de los mismos es muy amplio y es probable que existan aquellos cuyos conocimientos sean mayores a los descriptos a continuación.

Poseen conocimientos básicos de la programación y de la robótica adquiridos a través de la experiencia con la herramienta TortuBots. Conocen el robot Butiá, así como también, la forma de trabajar con los módulos conectados al mismo; averiguar su propósito, funcionalidades brindadas y el tipo de valores que reciben u ofrecen. No poseen experiencia ni conocimiento en la estructuración de aplicaciones a través de una cierta arquitectura. No son usuarios informáticos avanzados, por lo tanto no tienen conocimientos básicos de comunicación entre aplicaciones ni conocimientos de programación medios o avanzados.

3. Requerimientos funcionales

3.1 Entorno y Ejecución

Entorno de ejecución: Debe ejecutar de forma aceptable en Tablets con sistema operativo Android, siendo opcional el funcionamiento aceptable en dispositivos celulares con sistema operativo Android así como otros dispositivos y/o ordenadores con sistemas operativos distintos. El servidor Web deberá correr en Linux y funcionar de manera aceptable con los recursos de hardware de una SBC Raspberry Pi.

Debugging: El sistema deberá tener una funcionalidad que permita al usuario ver qué bloques implementados están siendo ejecutados en tiempo real, enlenteciendo la ejecución, si fuese necesario, de forma que el usuario pueda apreciar gráficamente la ejecución de un determinado comportamiento implementado.

Múltiples usuarios: El sistema deberá soportar múltiples usuarios. Permitiendo desarrollar a múltiples usuarios distintos comportamientos. Si bien no será un requisito trabajar de manera cooperativa sobre el mismo comportamiento o la misma paleta, deberá permitir que cada usuario desarrolle un comportamiento y que lo envíe para su ejecución al servidor donde se ejecutarán en conjunto según la arquitectura basada en prioridades definida en el documento de Estado del Arte.

Orientación pedagógica: El sistema tendrá una orientación hacia el aprendizaje constructivista y estará basado en las ideas de TortuBots, Scratch y eButia.

3.2 Robot

Paradigma robótico: La aplicación debe permitir la construcción de comportamientos robóticos que estén basados en el paradigma reactivo, siguiendo en dicha construcción y ejecución la arquitectura basada en prioridades definida en el documento de Estado del Arte.

Kit Robótico: El Robot al que estará orientado este sistema, es principalmente el Robot Butiá definido en el documento de Estado del Arte. Sin embargo, se debe mantener una cierta modularización que permita de manera simple extender el sistema a otros kit Robóticos.

Soporte a Robot Butiá: El sistema ofrecerá como se menciona en el requerimiento “Kit Robótico” soporte a Robot Butiá. Dentro del enfoque de la aplicación a la programación basada en bloques, el sistema debe tener bloques que permitan utilizar las funciones de los sensores y actuadores del robot Butiá que sean ofrecidos por el programa Bobot.

Sensores Genéricos: La aplicación tendrá alguna funcionalidad que permita la utilización de sensores nuevos que sean incluidos en el kit Robótico.

Calibración del Robot: El sistema debe proveer una funcionalidad que permita la calibración de los sensores y actuadores del robot Butiá.

3.3 Proyectos

Gestión de proyectos: En la aplicación se deben presentar herramientas con las funcionalidades de gestión de proyectos: creación, guardado y apertura.

Pantalla de gestión: La ubicación de las funcionalidades de gestión dentro del sistema deberá ser en la pantalla única principal o a lo sumo estar a un acceso de distancia, regla que se mantendrá en general para todas las funcionalidades.

Persistencia de proyectos: Se debe poder almacenar un proyecto con el código generado de cada comportamiento.

3.4 Comportamientos

Nombre y Prioridad: Los comportamientos creados deben estar identificados (no tiene por qué ser internamente en el sistema) por un nombre elegido por el usuario, lo cual permite un entendimiento ágil del comportamiento, además de una prioridad asignada que indicará la importancia de la ejecución del comportamiento frente a otros tal como fue descrito en el documento de Estado del Arte. Tanto el nombre como la prioridad de cualquiera de los comportamientos podrán ser modificados en cualquier momento, a excepción de cuando estos se estén ejecutando.

Pantalla de comportamientos: La metodología de trabajo dentro del sistema consistirá de un espacio para la edición de comportamientos y otro espacio en donde se podrán colocar minimizados los comportamientos ya creados, de forma en que en cualquier momento se puedan seleccionar de manera sencilla para su edición. Esto se definirá gráficamente en el documento de Pautas para la Interfaz de Usuario.

Simplicidad: Las herramientas que ofrece el sistema deben orientar la implementación a comportamientos sencillos que estén basados en el paradigma reactivo.

4. Requerimientos no funcionales

En esta sección se describirán los criterios que permiten juzgar la operatividad del sistema, los cuales serán de gran importancia a la hora de tomar consideraciones para la arquitectura del sistema.

Tiempo de reacción

Las órdenes ejecutadas en el sistema deben llegar al robot en un tiempo menor a un segundo y medio, de la misma forma el feedback de los comportamientos que llegan al sistema desde la plataforma robótica debe estar por debajo de esa cantidad.

Portabilidad

El sistema debe contemplar portabilidad a mayor cantidad de dispositivos móviles que cumplan con las restricciones especificadas, haciendo énfasis en los dispositivos Android.

Escalabilidad

El sistema debe permitir agregar nuevas funcionalidades sin encontrar limitaciones desde la codificación, concretamente debe ser posible incorporar nuevos bloques para controlar sensores que se agreguen al robot Butiá, debe ser extensible la lógica que define la arquitectura robótica, entre otras.

Costo

La aplicación debe ser de código abierto, libre, sin costo para su utilización. Todo usuario podrá hacer uso del sistema si cuenta con los componentes físicos requeridos, así como hacer modificaciones al sistema basados en las reglas del software libre.

Mantenibilidad

El sistema debe poder adaptarse a cambios que se generen tanto en la plataforma robótica como en la aplicación extendida para la interfaz gráfica.

Multilenguaje

El sistema debe permitir la configuración del lenguaje desplegado por la interfaz gráfica

del mismo, dando la posibilidad en principio de visualizar el texto en español e inglés.

Modularización

El sistema debe mantener un nivel de modularización tal que le permita con facilidad realizar los puntos de Mantenibilidad y Escalabilidad, obteniendo como resultado una aplicación creada en módulos que se adaptan a los cambios de las librerías que usan así como también permiten la re-implementación o extensión de un módulo en particular sin que los demás resulten afectados.

5. Requerimientos de documentación

Manual de usuario: Debe proveerse un manual de usuario orientado a los estudiantes. En el mismo debe explicarse cómo y porqué calibrar un robot. También como crear comportamientos, así como una descripción simple de la arquitectura

Manual de instalación: Se incluirá un manual de instalación que explicará paso a paso cómo realizar la instalación del sistema.

Página web: Se debe implementar una página web donde se irá actualizando toda la información relacionada con la elaboración del sistema. Esta poseerá noticias, mantendrá versiones del código en desarrollo y documentos del proyecto.

Pautas para la interfaz de usuario: Se realizará un documento de pautas de la interfaz de usuario. Este poseerá información de los principales estilos gráficos que se usarán en el sistema así como información del estructuramiento de las pantallas incluyendo si fuese necesario bocetos de estas.

Proyecto de Grado

IDE Android para la Programación de Comportamientos Robóticos

Manual de usuario

Andrés Nebel - Renzo Rozza

Tutores
Ing. Andrés Aguirre, Ing. Rafael Sisto

24 de Abril del 2014

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay

Índice

1. Introducción
    1.1. Visión general
    1.2. Comunicación con el robot
2. Funcionalidades
    2.1. Ingresar al sistema
    2.2. Área de trabajo
        2.2.1. Bloques destacados
    2.3. Calibrar
        2.3.1 Test rápido
    2.4. Marcar comportamiento como listo
    2.5. Ejecutar o depurar
    2.6. Cargar proyecto
    2.7. Otras funcionalidades
    2.8. Funcionalidades avanzadas
3. Tus primeros comportamientos
    3.1. Calibrar
    3.2. Comportamiento avanzar
    3.3. Comportamiento esquivar

1. Introducción

Para aquellos que tienen curiosidad por la robótica y buscan una herramienta para aprender y divertirse usando el robot Butiá, no deben pasar por alto el sistema Yatay.

Éste te permitirá desarrollar fácilmente comportamientos usando un entorno amigable, donde distintos bloquecitos son arrastrados y soltados en el área de trabajo para formar las acciones que tomará el robot. Además, puedes usarlo desde tu teléfono inteligente o tableta.

A continuación encontrarás todos los detalles necesarios para usar Yatay, desde su configuración, funcionalidades y ejemplos ilustrativos del uso del sistema.

1.1. Visión general

Yatay es un sistema que te permitirá desarrollar comportamientos robóticos a través de una interfaz amigable donde encontrarás los distintos bloques que determinan la lógica de cada comportamiento que construyas.

Un comportamiento es una acción o conjunto de acciones que realiza un robot a partir de un estímulo, en la robótica estos estímulos son advertidos por los sensores y las acciones son tomadas por los actuadores del robot.

En este sistema, estos comportamientos usan un valor numérico llamado prioridad para determinar cómo actuará el robot. Por lo tanto, el comportamiento con más prioridad que cumpla las condiciones para ser ejecutado será quien tome el control de los actuadores del robot Butiá en determinado momento.

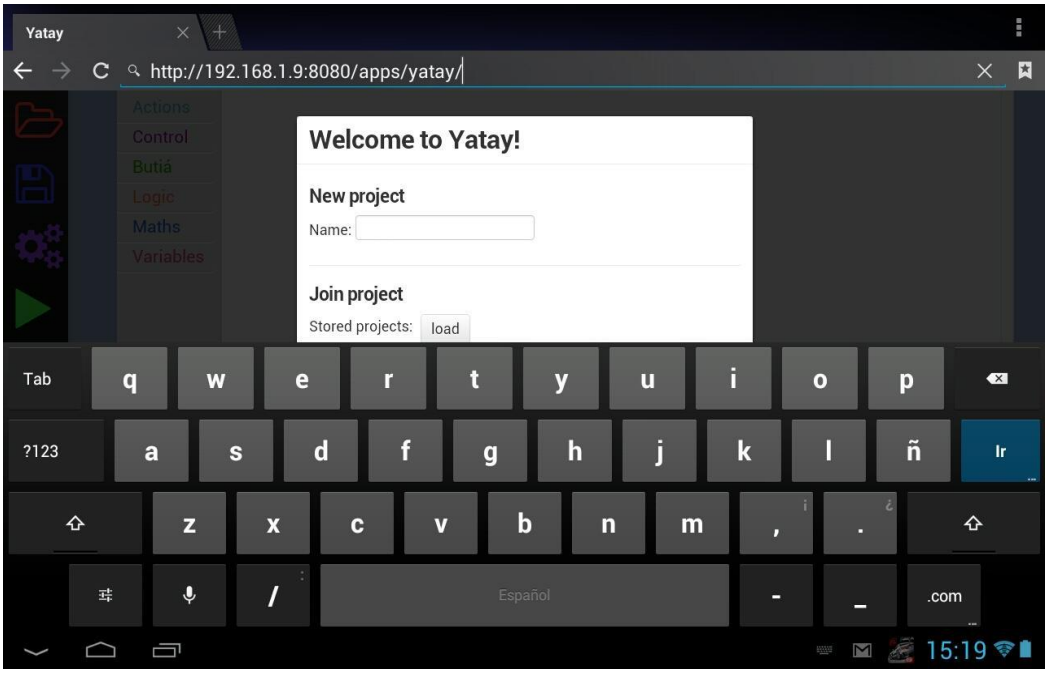
1.2. Comunicación con el robot

Yatay necesita de un navegador web para su ejecución. Un navegador web es un programa incluido en los dispositivos móviles que permite acceder a páginas web, algunos de los que puedes usar son Mozilla Firefox, Chrome, Safari, Android Browser.

Yatay fue implementado como una aplicación web, por lo tanto, utilizando un navegador web puedes acceder a la página del sistema. Se debe ingresar en el buscador del navegador web

la siguiente dirección: "*IP_Servidor*:8080/apps/yatay".

Por ejemplo, usando tu tableta puedes abrir el navegador que viene incluido por defecto e introducir la dirección web de la siguiente forma:

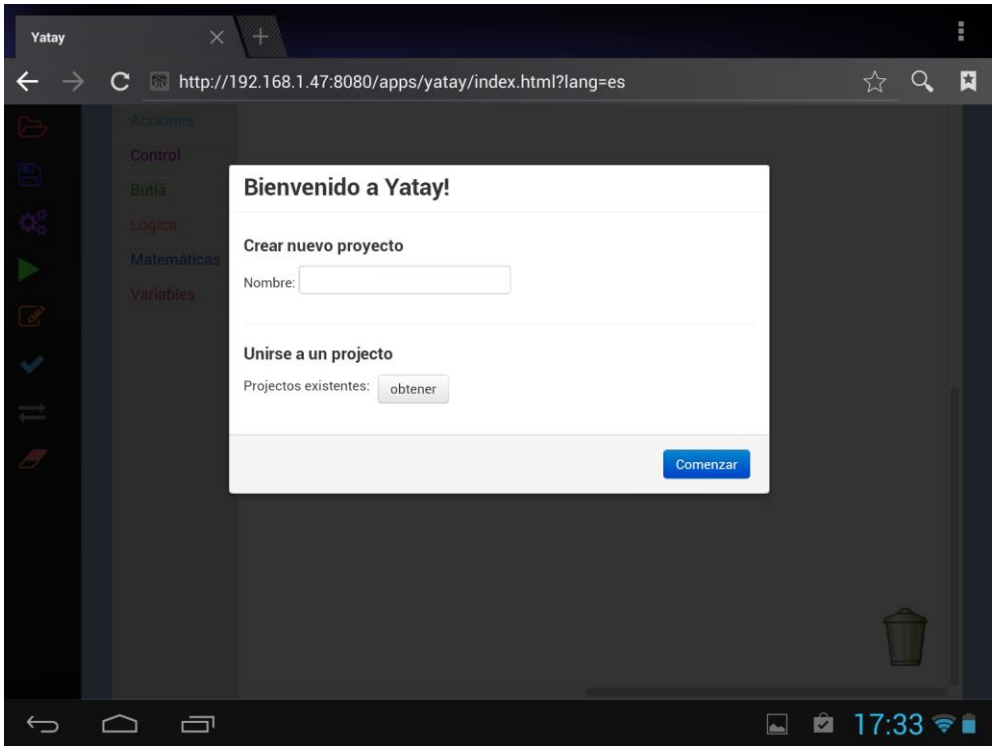


Android - Browser nativo (Tablet 10")

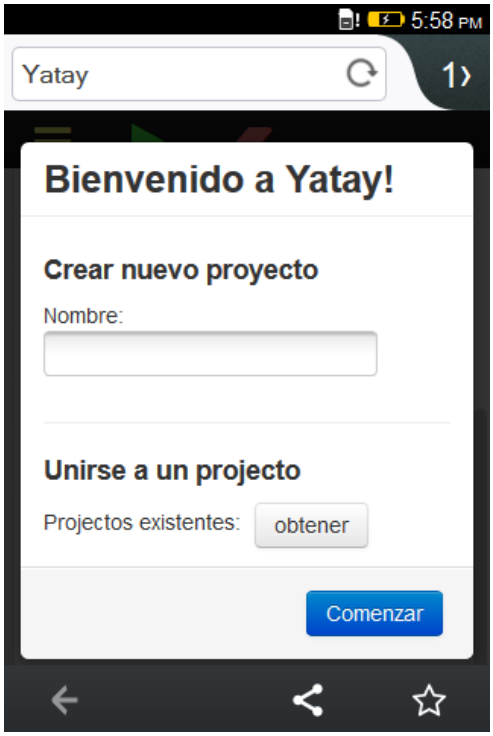
2. Funcionalidades

2.1. Ingresar al sistema

Encontrarás un diálogo de bienvenida, en el cuál podrás crear un proyecto nuevo (especificando un nombre) o unirte a un proyecto existente, lo cual es útil para trabajar con un grupo de compañeros sobre el mismo proyecto.

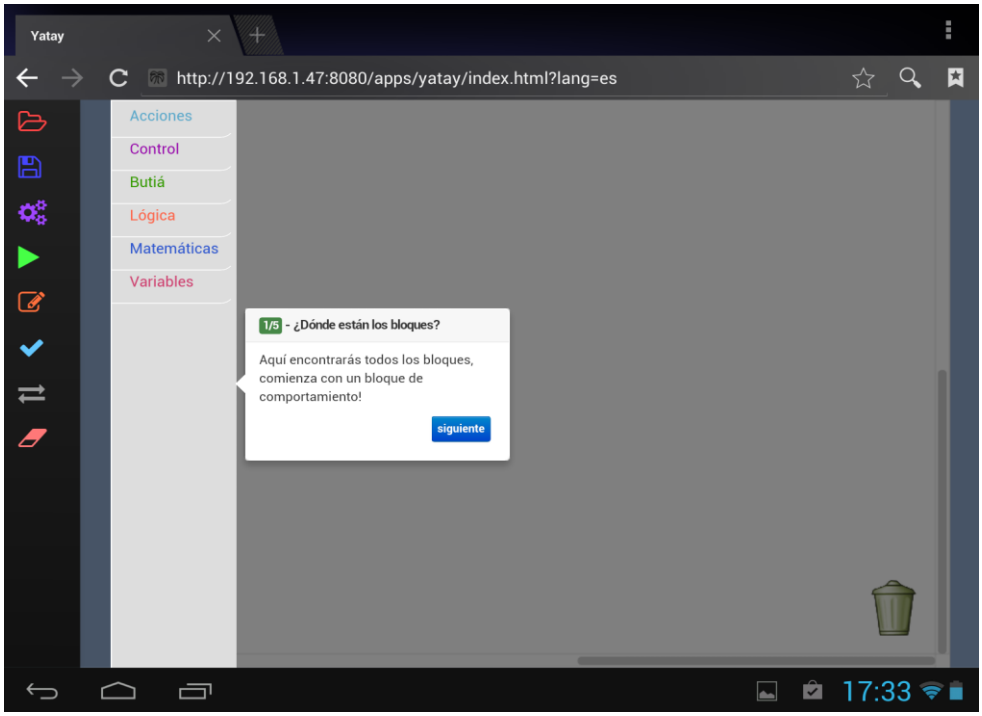


Android - Browser nativo (Tablet 10")

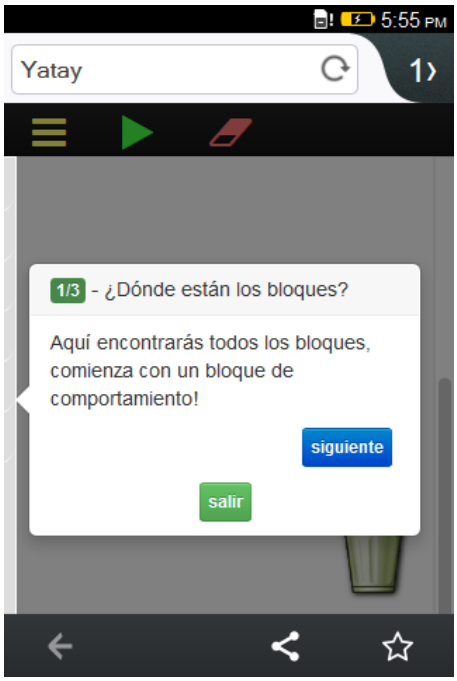


Firefox OS (celular 3.5’’)

Una vez que has ingresado a la aplicación, Yatay muestra un breve tutorial en el cual se explican los aspectos más importantes del entorno, para que puedas comenzar a crear tus comportamientos rápidamente.



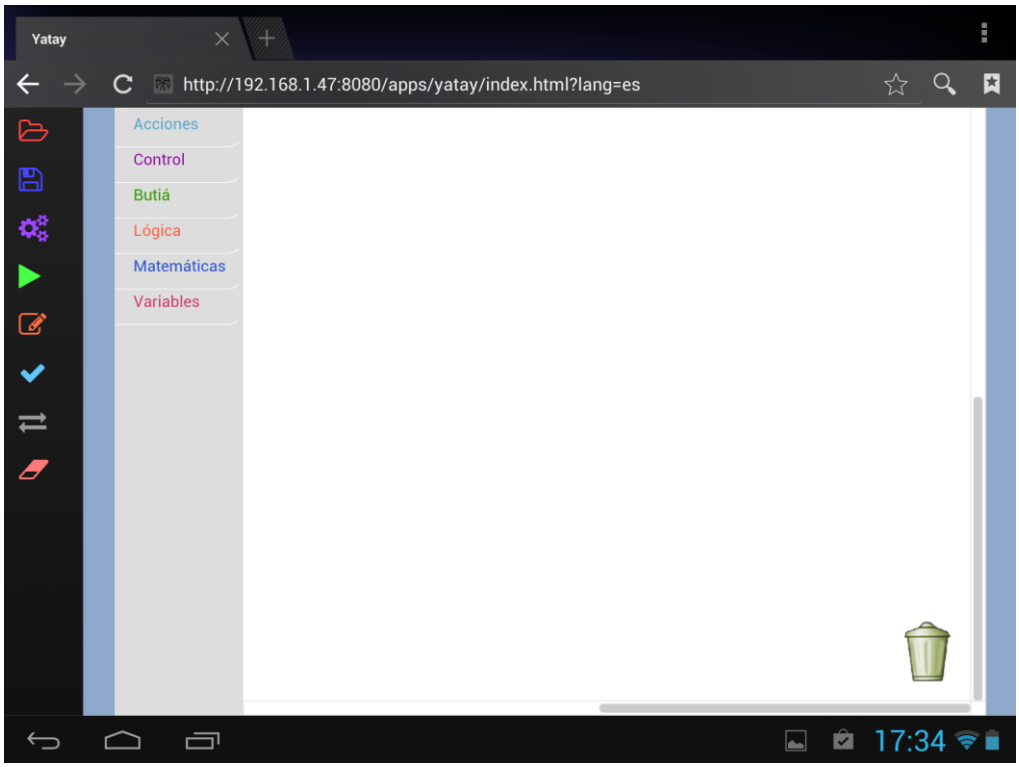
Android - Browser nativo (Tablet 10")



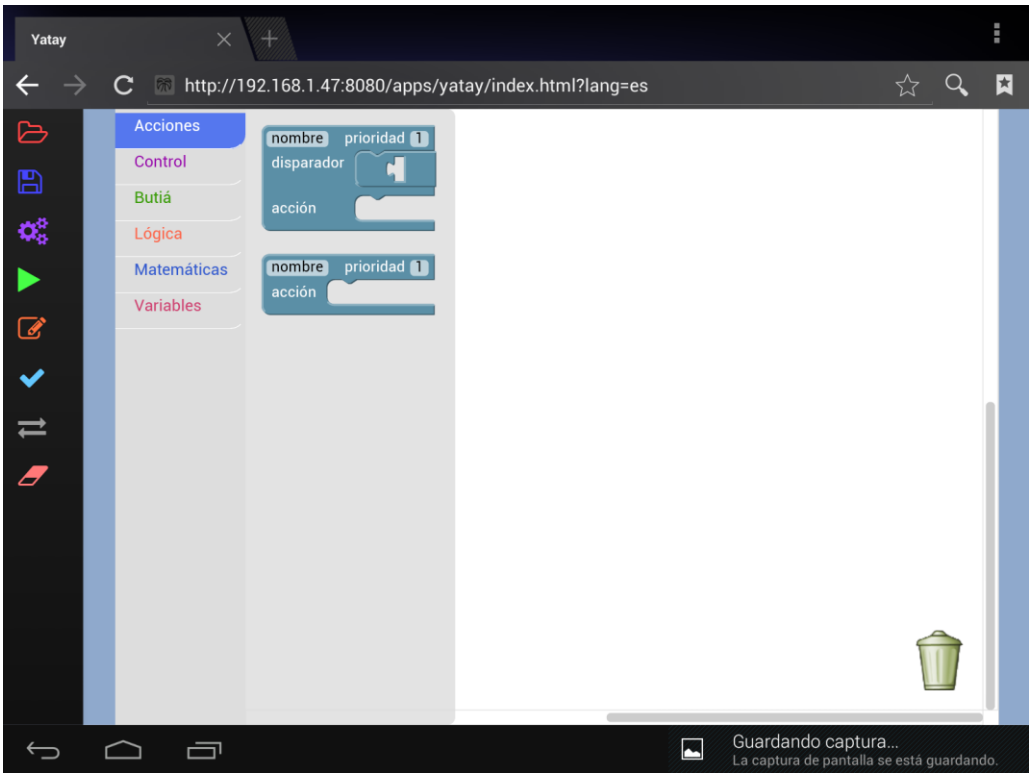
Firefox OS (celular 3.5'')

2.2. Área de trabajo

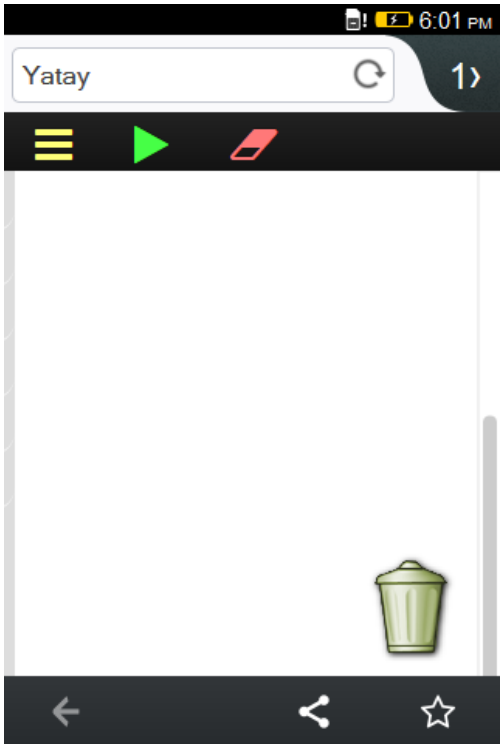
Como escenario principal Yatay cuenta con un área de trabajo, donde podrás encontrar y arrastrar los distintos bloques para formar tus comportamientos. Los bloques están agrupados en distintas categorías para que sea más fácil encontrarlos. Los que se encuentran en la categoría “Comportamiento” son los bloques principales, se debe comenzar con un bloque de esta categoría. Podrás editar un comportamiento a la vez en el área de trabajo.



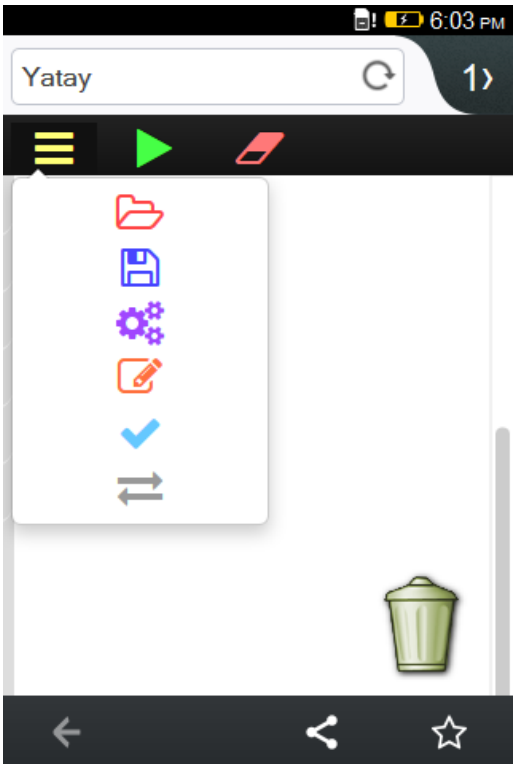
Android - Browser nativo (Tablet 10")



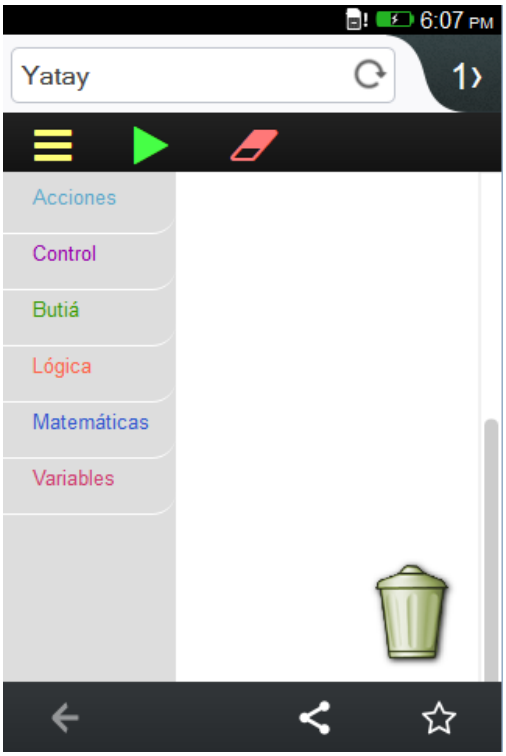
Android - Browser nativo (Tablet 10")



Firefox OS (celular 3.5'')



Firefox OS (celular 3.5’’)



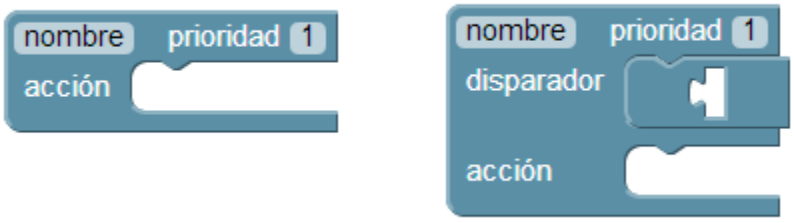
Firefox OS (celular 3.5’’)

Firefox OS (celular 3.5’’)

2.2.1. Bloques destacados

Es importante detenernos en algunos de los bloques más importantes que tiene el sistema para que no encuentres dificultades a la hora entender cómo funcionan:

Bloques de Acción:



Existen 2 bloques que permiten crear comportamientos para el robot. Estos se identifican por su nombre y tienen un número de prioridad, que indica a Yatay quien se debe ejecutar en casos en donde dos o más comportamientos intentan ejecutar al mismo tiempo. El sistema siempre elegirá el comportamiento de mayor prioridad.

El primer bloque, que en la imagen se muestra a la izquierda, permite crear una acción del robot que no posee disparador, es decir, que intentará todo el tiempo ejecutarse, compitiendo con su prioridad para ejecutar. Cuando sea el comportamiento con mayor prioridad, ejecutará los bloques anexados en “acción”.

El segundo bloque, que en la imagen se muestra a la derecha, permite crear un comportamiento el cual cuando se cumpla la condición de su disparador, intentará ejecutarse compitiendo con su prioridad correspondiente. Cuando cumpla la condición y sea el comportamiento con mayor prioridad, ejecutará los bloques anexados en “acción”.

Bloque “veces ejecutado”:

Este bloque retorna un número que indica cuantas veces se ha ejecutado el comportamiento en el que se encuentra. Internamente es un contador, que comienza en cero y cada vez que se ejecuta el comportamiento se incrementa en 1. Esto nos permite realizar varias cosas, algunos ejemplos simples:

- Si es la primera vez que se ejecuta el comportamiento, (“veces ejecutado” = 1) entonces realizar una acción, de lo contrario hacer otra.

- Realizar acciones una vez sí y otra vez no. Esto se realiza preguntando si “veces ejecutado” es par, entonces se realiza una acción, de lo contrario se hace otra.

- Ejecutar solo las primeras X veces. Esto se realiza preguntando si “veces ejecutado” es menor a X.

Bloques de variable:

El bloque “guardar en” (bloque de la izquierda en la imagen) permite almacenar bajo el nombre especificado (en la imagen es “ítem”) el valor retornado por un bloque conectado a este. Dicho valor es conservado y puede ser utilizado en cualquier parte que esté por debajo del bloque “guardar en” donde se almacenó. La forma de utilizar el valor almacenado es mediante el bloque que aparece a la derecha en la imagen, seleccionando en el menú de este el nombre de la variable que se almacenó (en la imagen es “ítem”).

Bloques de crear sensor:

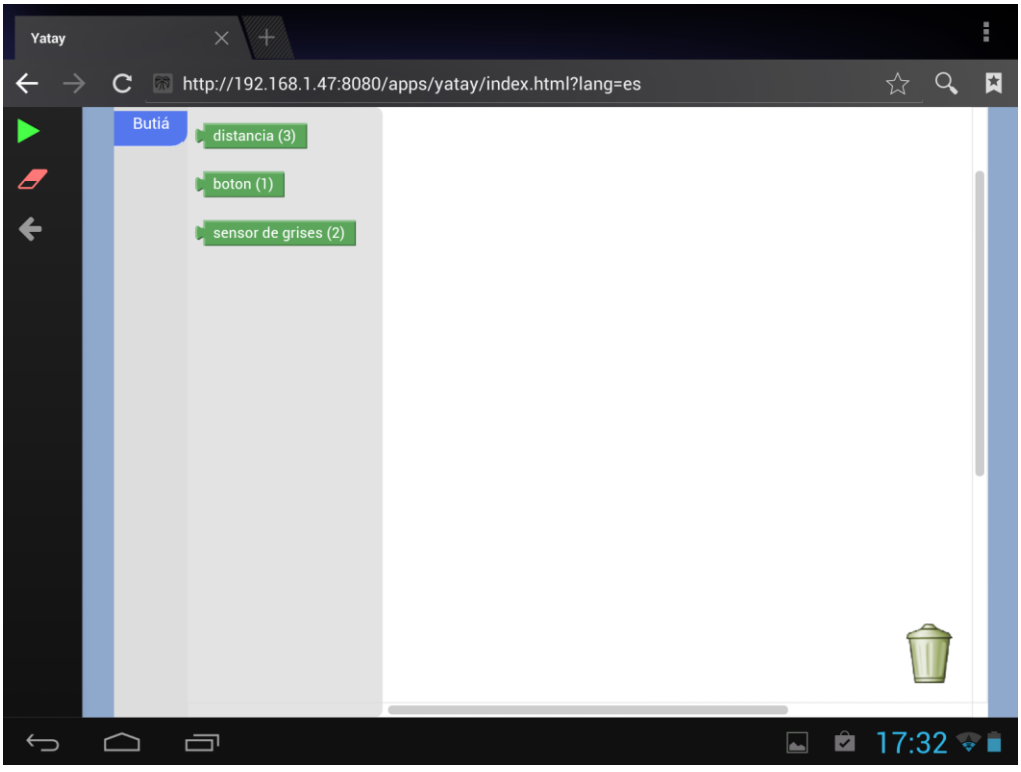
El bloque de crear sensor funciona de forma similar al bloque de variable, sin embargo es mucho más potente, ya que no almacena un valor sino que almacena la expresión misma. En una variable, una vez que se almacena un valor con el bloque “guardar en”, este permanece incambiado. Sin embargo, con crear sensor este valor se reevalúa cada vez que se utiliza el bloque “sensor” (bloque de la derecha de la imagen). El bloque “crear sensor” puede almacenar expresiones como ser operaciones aritméticas, operaciones lógicas (ej.: “sensorX= 1 y sensorY = 1”) y también números. El nombre “crear sensor” se debe a que nos permite crear un bloque que sea combinación de varios sensores, por ejemplo si se almacena el valor “sensorX + sensorY” o también “raíz cuadrada de sensorX”.

2.3. Calibrar

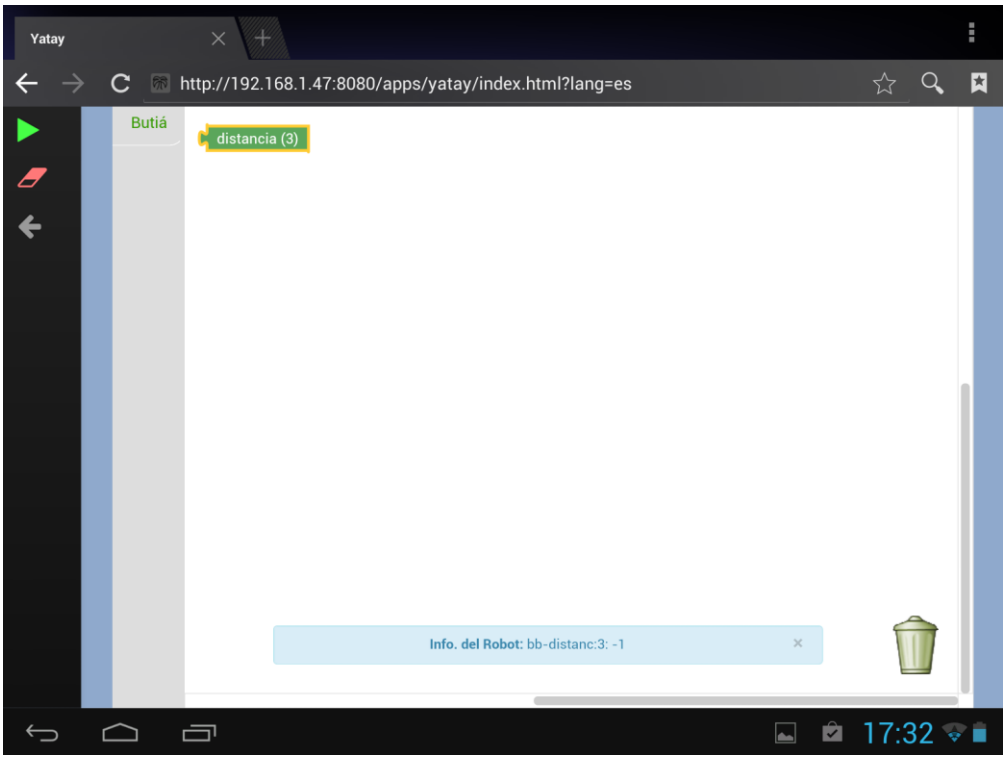
Puedes rápidamente conocer los valores que devuelven los sensores y probar los actuadores utilizando el botón en forma de engranajes. Debes, luego de haber apretado este botón, elegir un único bloque de la categoría “Butiá” y presionar el botón de comenzar, que tiene la siguiente forma. En caso de querer volver al escenario principal debes apretar el botón volver (en la versión móvil la flecha señala hacia arriba).

2.3.1 Test rápido

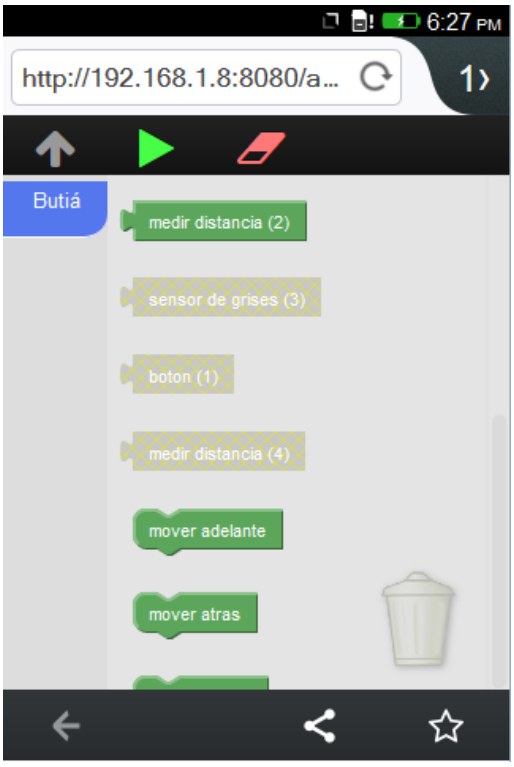
Existe una forma más simple de probar el valor de un sensor o un actuador. Si estás trabajando en un comportamiento, puedes seleccionar un bloque de sensor o actuador y presionar el botón en forma de engranajes. Esto es equivalente a entrar a la calibración y arrastrar dicho bloque desde la categoría “Butiá” y presionar.



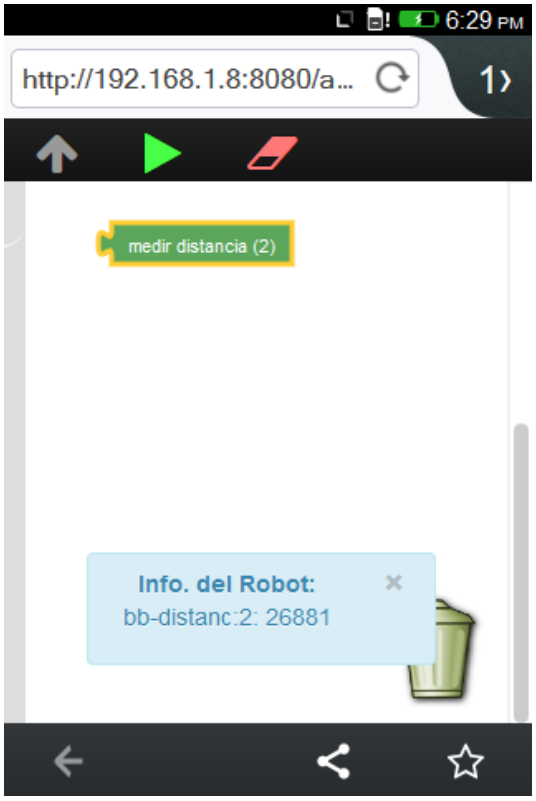
Android - Browser nativo (Tablet 10")



Android - Browser nativo (Tablet 10")



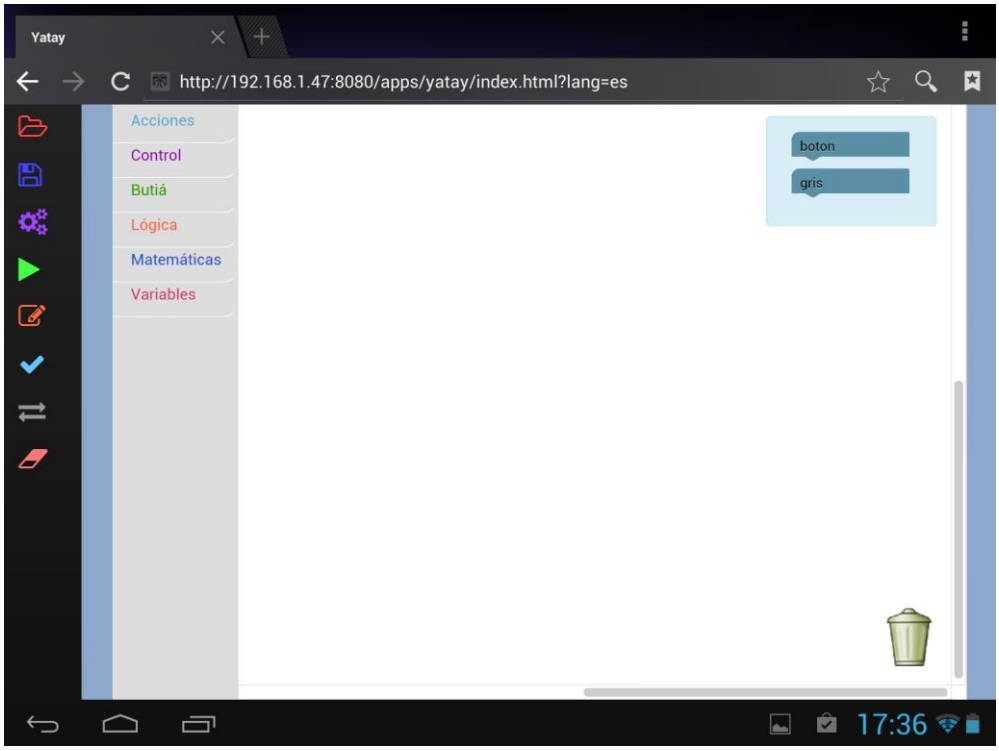
Firefox OS (celular 3.5’’)



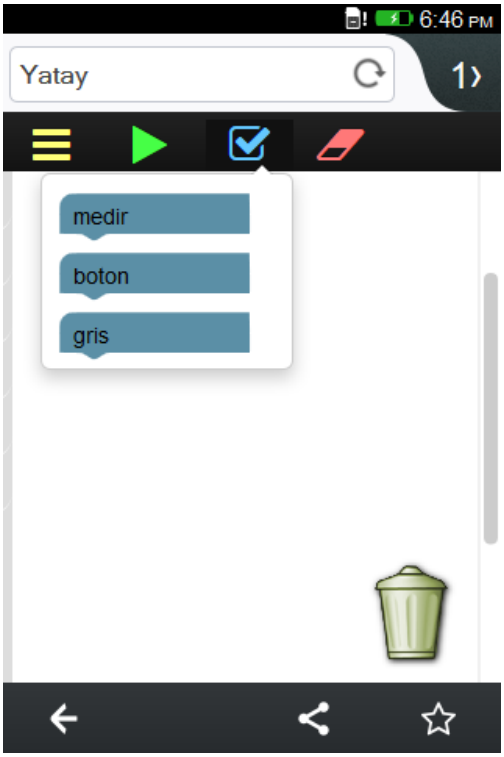
Firefox OS (celular 3.5’’)

2.4. Marcar comportamiento como listo

Cuando termines de desarrollar un comportamiento puedes usar el siguiente botón  para marcar el mismo como listo. El comportamiento aparecerá minimizado sobre el borde superior derecho en las tabletas y agrupados en un botón con la siguiente forma  en las versiones para celulares.



Android - Browser nativo (Tablet 10")

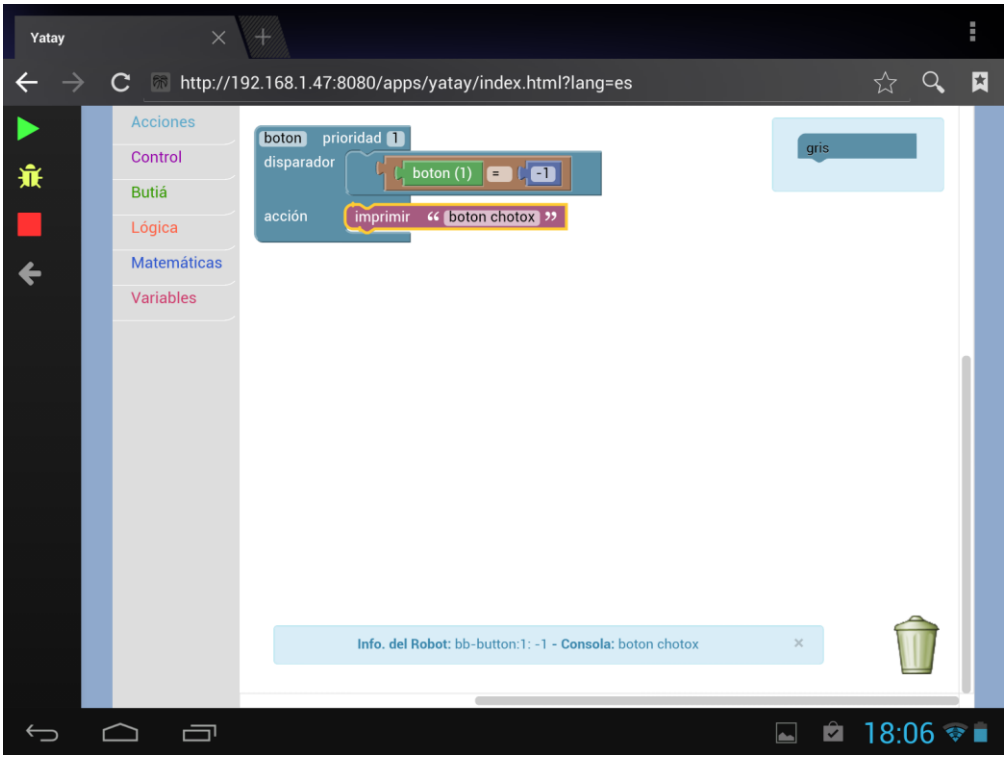


Firefox OS (celular 3.5’’)

2.5. Ejecutar o depurar

Para ejecutar tus comportamientos, debes apretar el botón de comenzar desde el menú principal, luego tienes dos formas de ejecutar: la primera usando el mismo botón de comenzar que se muestra en el escenario de ejecución y la segunda apretando el botón de depuración con la siguiente forma  . La primera enviará los comportamientos al servidor y serán ejecutados de forma normal sobre el robot Butiá, en cambio usando el modo depurar los comportamientos serán enviados al servidor pero su ejecución se verá enlentecida con la finalidad de que sea distinguible el bloque que está en ejecución en cada momento (se iluminará en el área de trabajo el bloque).

Además, cuando comiences alguna ejecución, se desplegará un diálogo con los mensajes provenientes de la ejecución (información del robot y las impresiones en consola). Mientras estés en modo ejecución Yatay no te permitirá editar o modificar los comportamientos. Para poder hacer esto, debes detener la ejecución y volver al menú inicial.

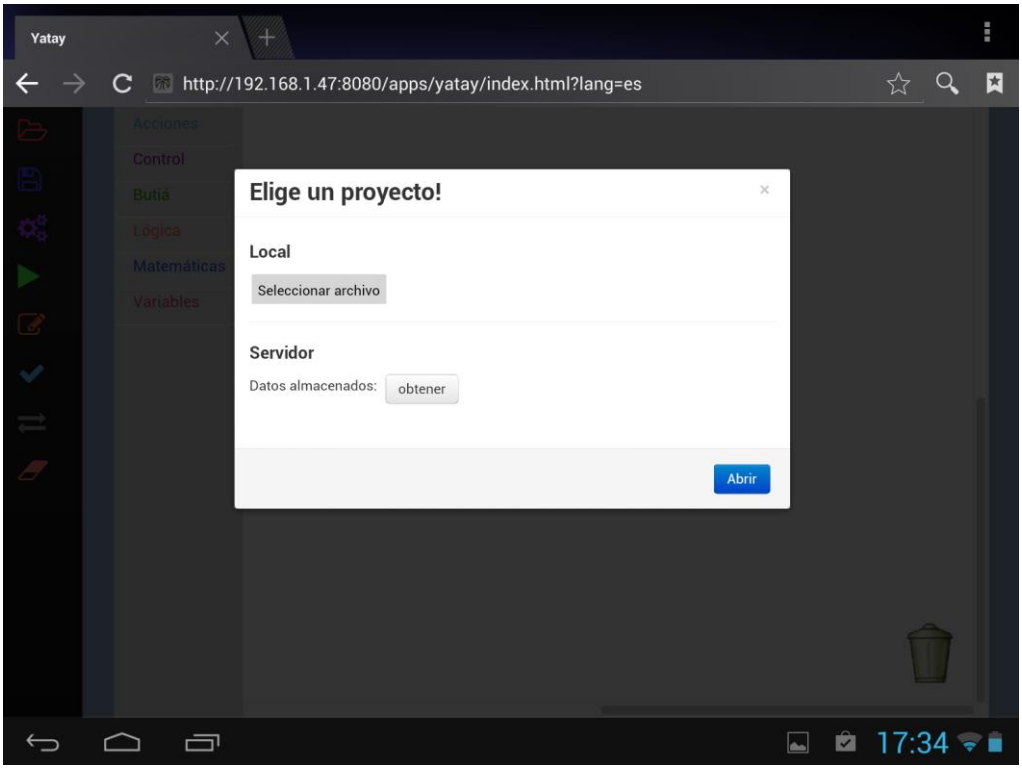


Android - Browser nativo (Tablet 10")

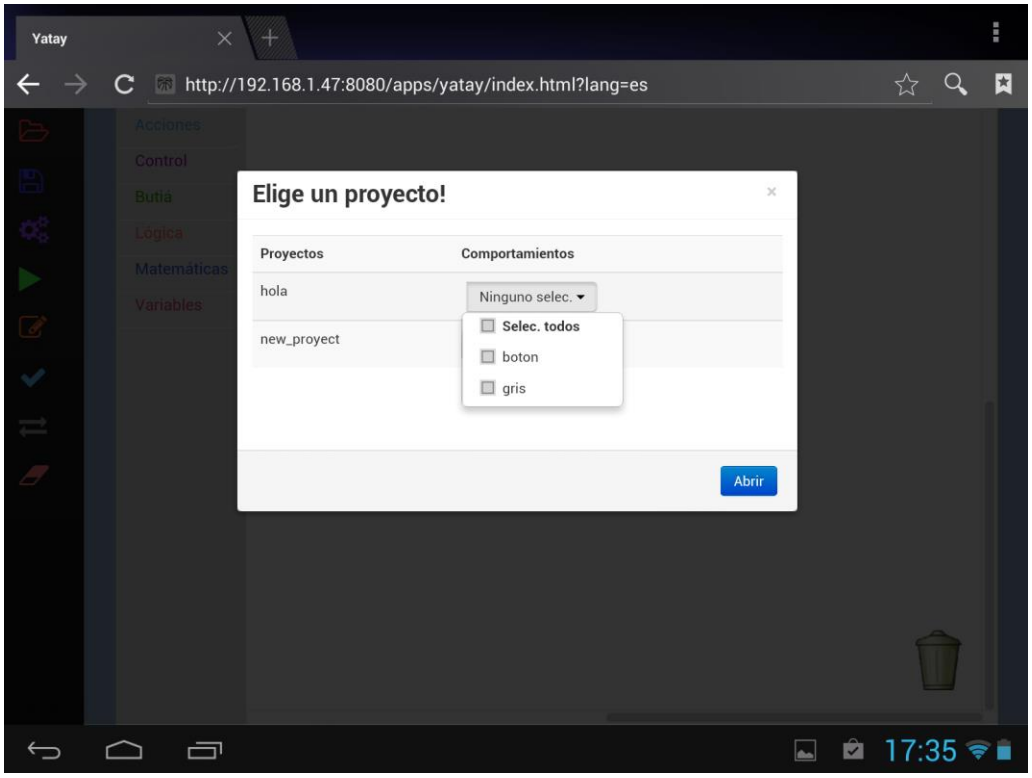
Firefox OS (celular 3.5’’)

2.6. Cargar proyecto

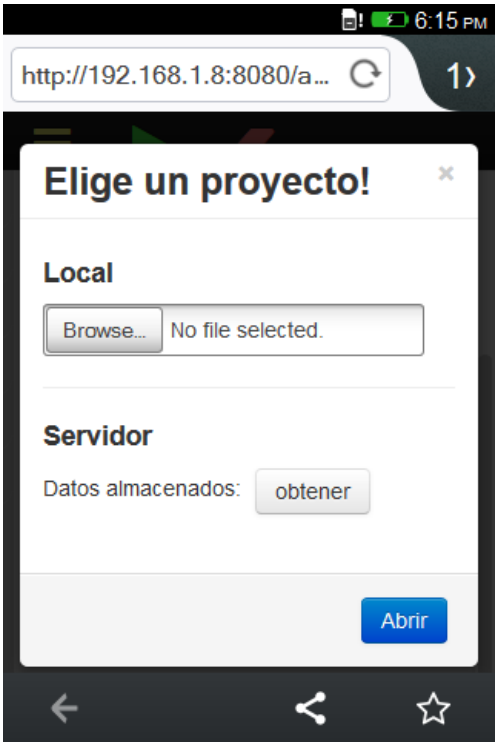
Al presionar el botón en forma de carpeta abierta  se abrirá un diálogo para que puedas seleccionar un archivo local con comportamientos, o elegir uno o varios comportamientos a cargar desde la base de datos alojada en el robot.



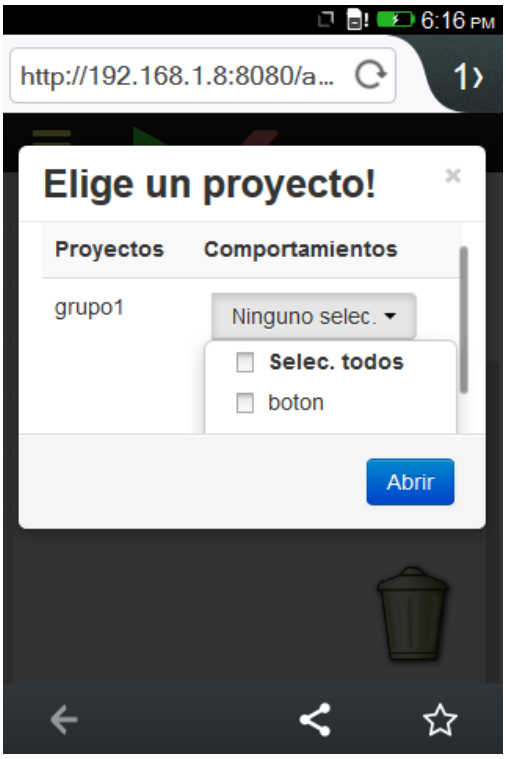
Android - Browser nativo (Tablet 10")



Android - Browser nativo (Tablet 10")



Firefox OS (celular 3.5'')



Firefox OS (celular 3.5’’)

2.7. Otras funcionalidades

- ¿Cómo guardar los comportamientos?

Usando el botón con la siguiente forma  puedes almacenar en tus dispositivos los comportamientos que hayas desarrollados, además el sistema guarda cada comportamiento que estés construyendo en la base de datos. Dando la posibilidad de obtenerlo usando la funcionalidad de cargar proyecto remoto (mencionada anteriormente).

- ¿Cómo borrar los comportamientos?

Los comportamientos pueden ser individualmente eliminados usando la papelera ubicada en el área de trabajo, además se pueden borrar todos los bloques de la pizarra o todos los comportamientos (incluyendo los listos) usando el siguiente botón  .

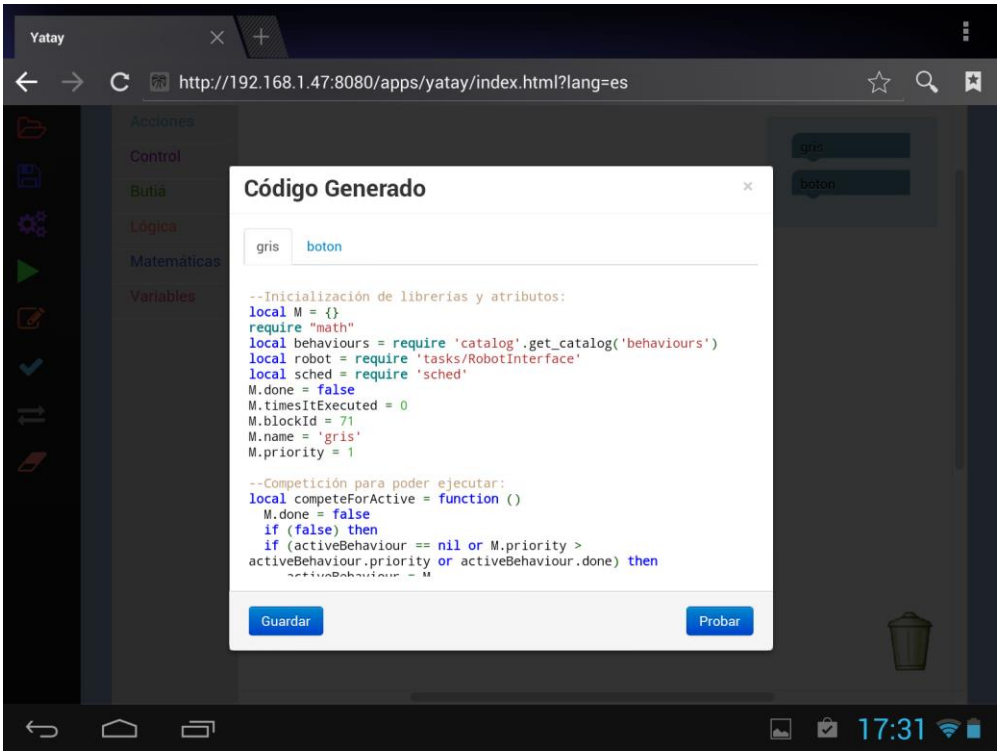
- ¿Puedo cambiar de lenguaje?

Hasta el momento los idiomas disponibles para Yatay son español e inglés, puedes cambiar el lenguaje usando el siguiente botón  .

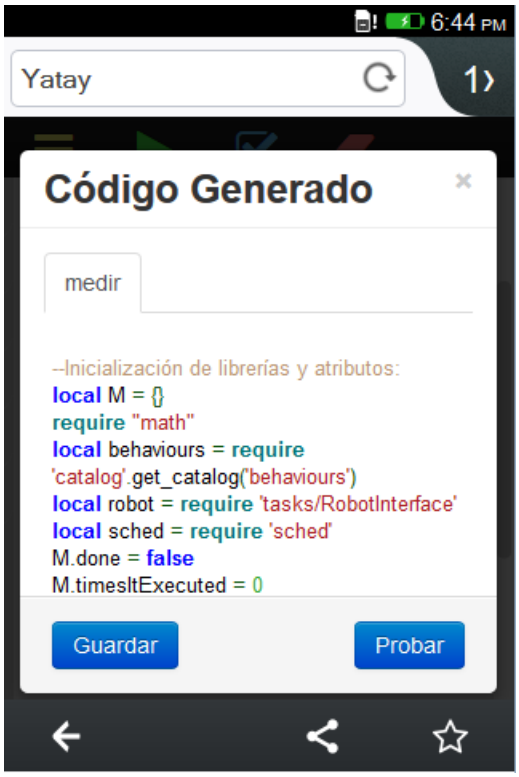
2.8. Funcionalidades avanzadas

Yatay ofrece la posibilidad de editar el código generado a partir de los bloques, este código está programado en el lenguaje Lua. En particular, esta funcionalidad fue implementada para usuarios avanzados que posean conocimientos sobre lenguajes de programación y estén interesados en investigar cómo Yatay genera el código para controlar al robot.

Apretando el botón con la siguiente forma  puedes modificar los comportamientos y probar tus cambios dando clic en el botón "Probar". Como opción adicional, si tus comportamientos funcionan bien, puedes guardarlos usando el botón de "Guardar", ten en cuenta que los archivos editados sólo podrán ser probados con el botón mencionado. Al cerrar el diálogo tus cambios se perderán.



Android - Browser nativo (Tablet 10")



Firefox OS (celular 3.5’’)

3. Tus primeros comportamientos

Supongamos que deseamos construir un comportamiento usando el sistema Yatay para que el robot Butiá se mueva hacia adelante y logre evitar chocar contra las paredes. ¿Cómo podríamos lograr esto?

Una opción sería usar el sensor de distancia para reconocer cuando el robot se encuentra cerca de una pared y en esos casos doblar para evitar el choque. A continuación se detalla paso a paso como construir los comportamientos necesarios para lograrlo.

3.1. Calibrar

Para comenzar necesitamos conocer el valor que el sensor de distancia posee cuando se encuentra "cerca" de una pared. Por lo tanto, como fue mencionado se debe apretar el botón con la siguiente forma  para entrar en el modo de calibración. Allí se podrá seleccionar desde la categoría "Butiá" en la paleta el bloque que representa el sensor de distancia y luego usar el botón de ejecutar  .

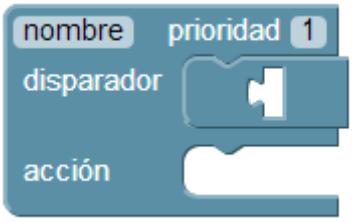
Firefox OS (celular 3.5'')

El robot fue colocado cerca de una pared y como se puede ver, el valor que devuelve el sensor de distancia es 25600. Para entender un poco más qué significa ese valor podemos,

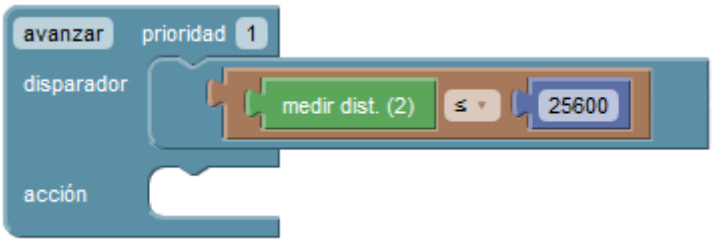
mientras la calibración está ejecutando acercar o alejar el robot de la pared. Si lo acercamos el valor del sensor crece, ¿no? y si lo alejamos el valor disminuye, ¿no? Entonces, el valor del sensor distancia nos puede servir para saber cuándo los comportamientos deben ejecutarse. Por esa razón el sensor distancia será determinante para definir el estímulo al cual el robot debe reaccionar. Al terminar tu calibración debes volver a la pantalla inicial usando el botón en forma de flecha.

3.2. Comportamiento avanzar

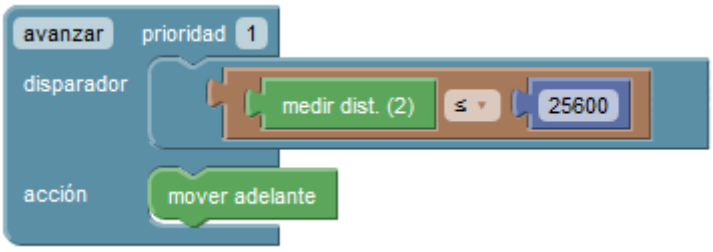
Como fue mencionado, el robot debe avanzar mientras no se encuentre cerca de una pared, pero ¿qué tan cerca? Como parte de este ejemplo supondremos que el valor límite es 25600, entonces si el sensor de distancia devuelve un valor menor a 25600 el robot debe avanzar. Para lograr esto, primero debemos agregar un bloque de comportamiento con disparador a la pizarra:



A este comportamiento se le debe asignar un nombre, en este caso utilizaremos el nombre “avanzar” para identificarlo y se dejará la prioridad en 1. Luego, debe ser agregado un bloque de la categoría “Lógica” de la paleta que nos permita comparar el valor que tiene el sensor de distancia con un número entero (bloque ubicado en la categoría “Matemáticas”), de la siguiente forma:



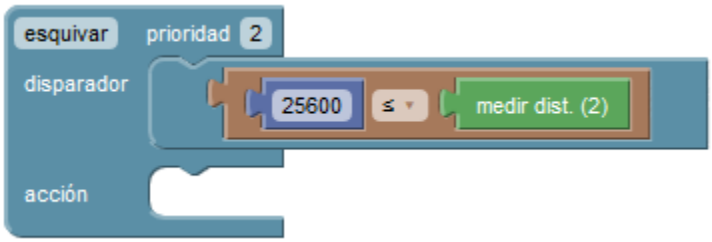
Por último, debemos agregar la acción que realizará el robot cuando se cumpla la condición contenida en el disparador. En este caso, debemos agregar un bloque dentro de la categoría “Butiá” que permita mover al robot hacia adelante.



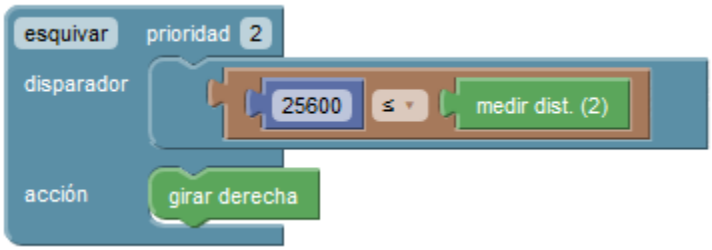
Hemos terminado nuestro comportamiento, ahora debemos comenzar con un nuevo comportamiento para que el robot evite chocar con las paredes. Debemos antes de continuar marcar nuestro comportamiento como listo, esto se puede lograr usando la funcionalidad detallada anteriormente.

3.3. Comportamiento esquivar

En este momento tenemos nuestro comportamiento “avanzar” terminado y nuestra pizarra vacía para comenzar un nuevo comportamiento. A este comportamiento lo llamaremos “esquivar” y como queremos que nuestro robot no se choque contra las paredes debemos usar una prioridad mayor a la utilizada en el comportamiento anterior. De esta forma, cuando el robot tenga que decidir que comportamiento ejecutar va a reconocer que es más importante esquivar la pared que seguir avanzando. Al igual que en el comportamiento anterior tendremos que agregar la comparación lógica para determinar qué dispara la acción del comportamiento, de la siguiente forma:



Para terminar este comportamiento tenemos que decidir qué acción va a tomar el robot cuando se encuentre cerca a la pared. En este ejemplo vamos a utilizar un bloque para que el robot gire a la derecha cuando lo mencionado suceda. Por lo tanto, el comportamiento esquivar estaría conformado por los siguientes bloques:



Resta solamente comenzar la ejecución para ver cómo se comporta el robot con estos dos comportamientos.

Proyecto de Grado

IDE Android para la Programación de Comportamientos Robóticos

Pautas para la interfaz de usuario

Andrés Nebel - Renzo Rozza

Tutores
Ing. Andrés Aguirre, Ing. Rafael Sisto

24 de Abril del 2014

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay

Índice

1. Objetivo
2. Interfaz de Usuario: requerimientos
    - 2.1. Visibilidad
    - 2.2. Velocidad de renderizado
    - 2.3. Eliminación de mensajes de error
    - 2.4. Paradigma reactivo: sintaxis
    - 2.5. Ejecución Visible
3. Interfaz de Usuario: aspecto visual
4. Bocetos de pantalla
    - 4.1. Bocetos de tabletas y ordenadores
    - 4.2. Bocetos de dispositivos de menor pantalla

1. Objetivo

Este documento pretende detallar las características que la interfaz de usuario debe cumplir para lograr una experiencia satisfactoria por parte de los usuarios al hacer uso del sistema. Para esto se considerará la investigación plasmada en el documento Estado del Arte [2] y los requerimientos relevados hasta el momento, detallados en el documento Especificación de Requerimientos [1].

El objetivo de la interfaz gráfica será presentar un sistema lo más intuitivo y amigable posible para el usuario, de modo que el mismo pueda familiarizarse fácilmente con las funciones que posee el entorno de desarrollo (IDE), y también desde el punto de vista gráfico atraiga y divierta. Definiremos una visión general de la interfaz de usuario, la cual modelará un lenguaje de programación visual (LPV) basado en bloques, como lo son TortuBots [3] y Scratch [4], que permita a personas sin conocimiento informáticos lograr códigos de programación simples para controlar la plataforma robótica Butiá.

Se intenta seguir en todo momento determinadas pautas a nivel gráfico: primero, dar el control al usuario, es decir, llevar a cabo un sistema que reaccione a las necesidades del usuario y que brinde facilidades para lograr los cometidos; segundo reducir la carga de trabajo por parte del usuario, por ej. siempre que sea posible el sistema debe almacenar la información pertinente; y por último, lograr que la interfaz sea consistente, toda la información visual está organizada de acuerdo con la capacidad de los dispositivos móviles para que su diseño haga uso de todas la presentaciones de los mismos.

2. Interfaz de Usuario: requerimientos

El sistema desde el punto de vista gráfico está definido por las tecnologías utilizadas en su desarrollo, siendo en este caso las principales herramientas HTML5, JavaScript, CSS3 y además otras librerías como complemento, ya sea Modernizr, YepNope y JQuery Mobile. Éstas son de suma importancia para dar soporte a dispositivos móviles, los cuales cuentan con características muy variadas y por lo tanto representa un gran desafío adaptar un LPV basado en bloques a los displays relativamente pequeños y la manipulación por controladores directos (tecnología táctil). Para satisfacer estas necesidades encontramos necesario definir los requerimientos para la interfaz de usuario:

2.1. Visibilidad

La interfaz gráfica del sistema debe contemplar la mayor visibilidad sobre los elementos desplegados en todo momento, dado fluidez para mantener al alcance todas las funcionalidades, haciendo uso de una única ventana.

2.2. Velocidad de renderizado

Siendo niños de corta edad los usuarios de este sistema, es importante tener presente que pueden distraerse si el sistema demora en desplegar los elementos, por lo que el renderizado de la interfaz gráfica debe ser ágil e intentar aprovechar al máximo las limitadas capacidades de procesamiento de los dispositivos móviles.

2.3. Eliminación de mensajes de error

Desde la interfaz gráfica no desplegará mensajes de error. Esto ha sido utilizado en muchos LPVs bajo la teoría de que la experimentación y experiencia enseñará a los usuarios cómo funciona. Además, al ser un LPV basado en bloques los errores de sintaxis no existen, dado que la única forma para que un comportamiento tenga sentido es cuando los bloques encastran.

2.4. Paradigma reactivo: sintaxis

A partir de las experiencias obtenidas con TortuBots, se debe tomar consideración sobre el manejo de variables numéricas para la representación de medidas. Siendo un claro ejemplo la distancia; resulta confuso manejar bloques que expresan la cantidad en metros que el robot Butiá avanzará ya que por lo general no se cumple con exactitud en práctica. Por lo tanto, se debe respetar una sintaxis adecuada para la implementación gráfica de los bloques de acuerdo al paradigma reactivo, manejando bloques que estén mapeados directamente con comportamientos robóticos.

2.5. Ejecución Visible

El sistema contará con la funcionalidad de debugging, y por lo tanto desde la interfaz gráfica se debe dar soporte para lograr que los usuarios entiendan que comportamiento se está ejecutando en determinado momento.

3. Interfaz de Usuario: aspecto visual

Según los requerimientos relevados hasta el momento, el aspecto visual del LPV basado en bloques tendrá a grandes rasgos tres grandes componentes que se desplegarán en pantalla: primero una paleta que contendrá los distintos bloques, los cuales representan tanto comportamientos del robot Butiá como instrucciones lógicas; segundo un workspace donde se podrán encastrar bloques para conformar distintos comportamiento y a su vez un espacio que contenga los comportamientos listos; y tercero, un menú que permita ejecutar, detener y debuggear los comportamientos programados y también opciones para abrir, cerrar y guardar los proyectos.

4. Bocetos de pantalla

Se incluyen en esta sección, bocetos de pantalla que representan los lineamientos de interfaz gráfica a seguir para desarrollar el sistema. Básicamente se pueden separar en 2 grupos: los bocetos para ordenadores de escritorio y tabletas que tendrán una disposición de los elementos igual, y otros bocetos para dispositivos móviles de resolución menor como ser celulares.

4.1. Bocetos de tabletas y ordenadores

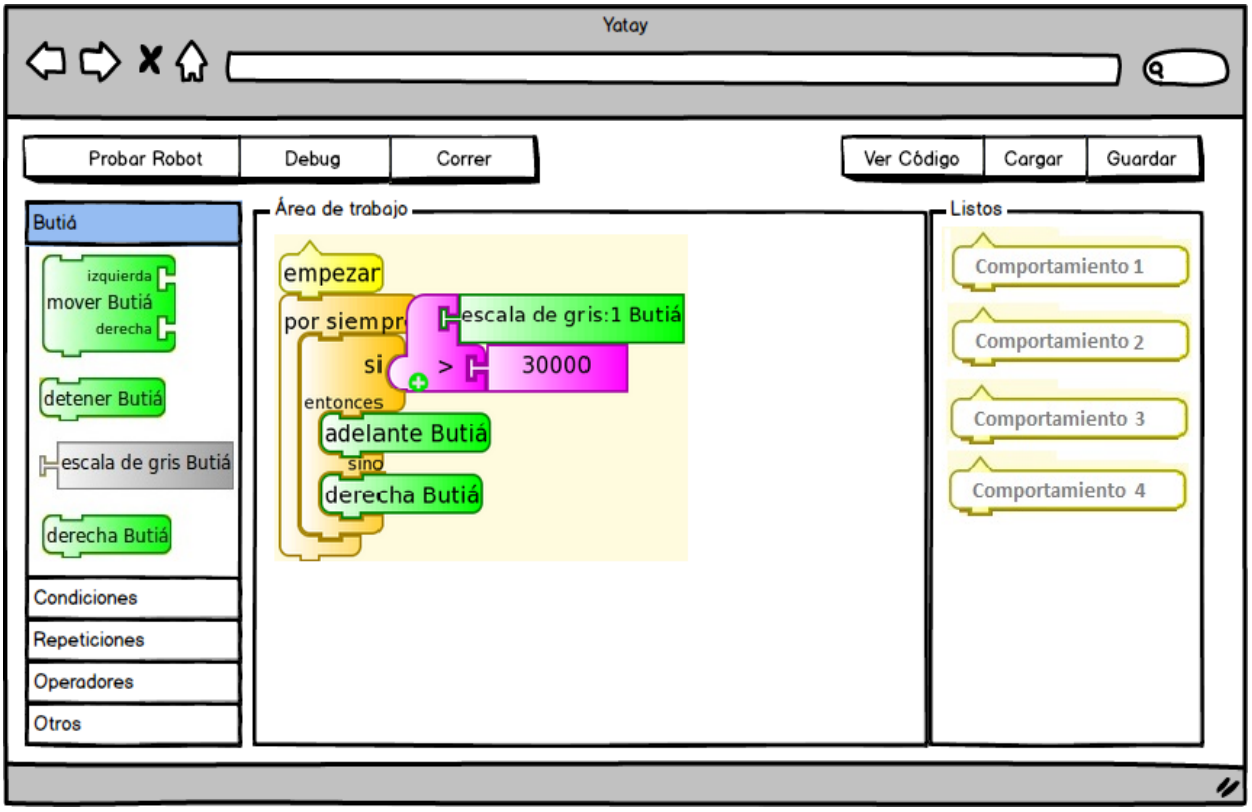


Imagen 1: Pantalla principal, entorno de trabajo. A la izquierda se muestran los bloques que existen en el sistema. Si están en verde están disponibles para agregar, mientras que si están en gris no. En el centro se encuentra el comportamiento que se encuentra actualmente en edición. A la derecha se muestra los comportamientos que han sido creados.

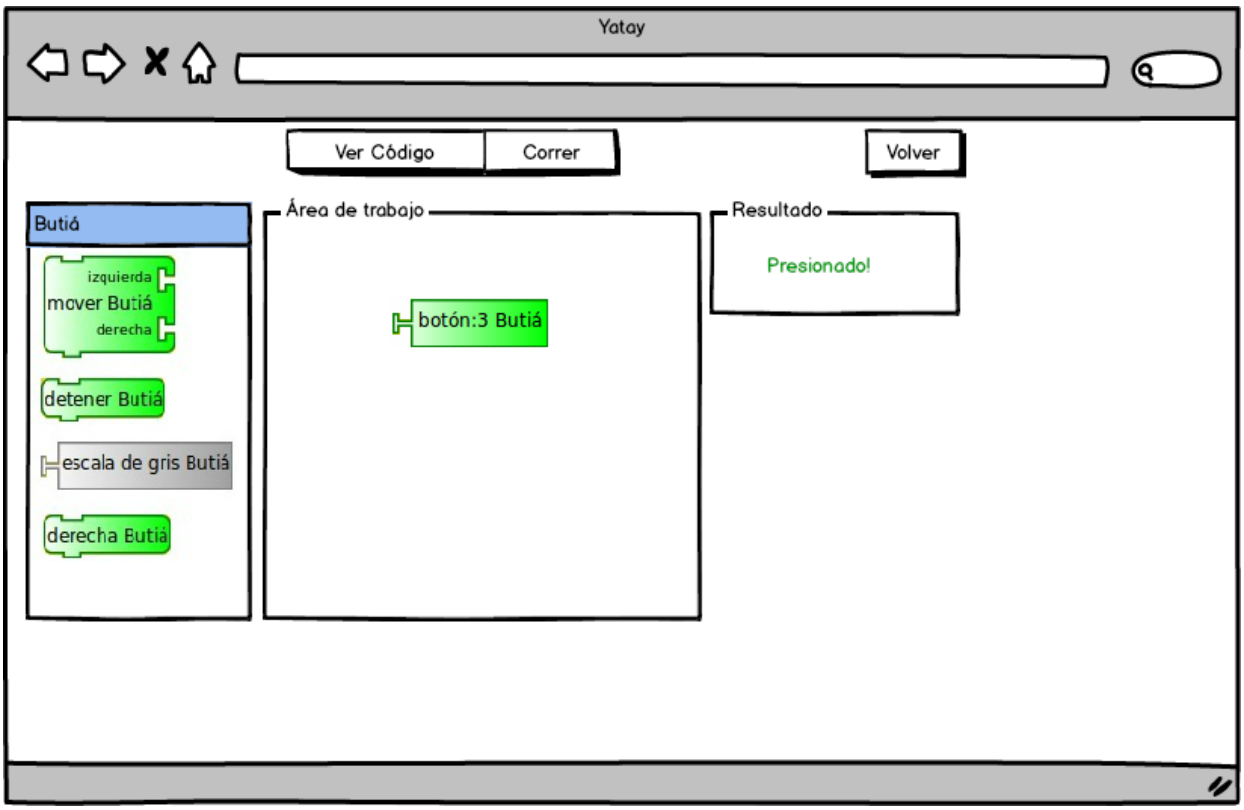


Imagen 2: Pantalla de ejecución rápida. Permite probar los sensores y actuadores del robot sin la necesidad de crear un comportamiento que lo contenga. Es útil a los efectos de la calibración del robot. A la izquierda muestra los sensores y actuadores disponibles (si están en verde) a agregar. En el centro se muestra el sensor/actuador siendo probado. A la derecha se muestra el resultado de la ejecución.

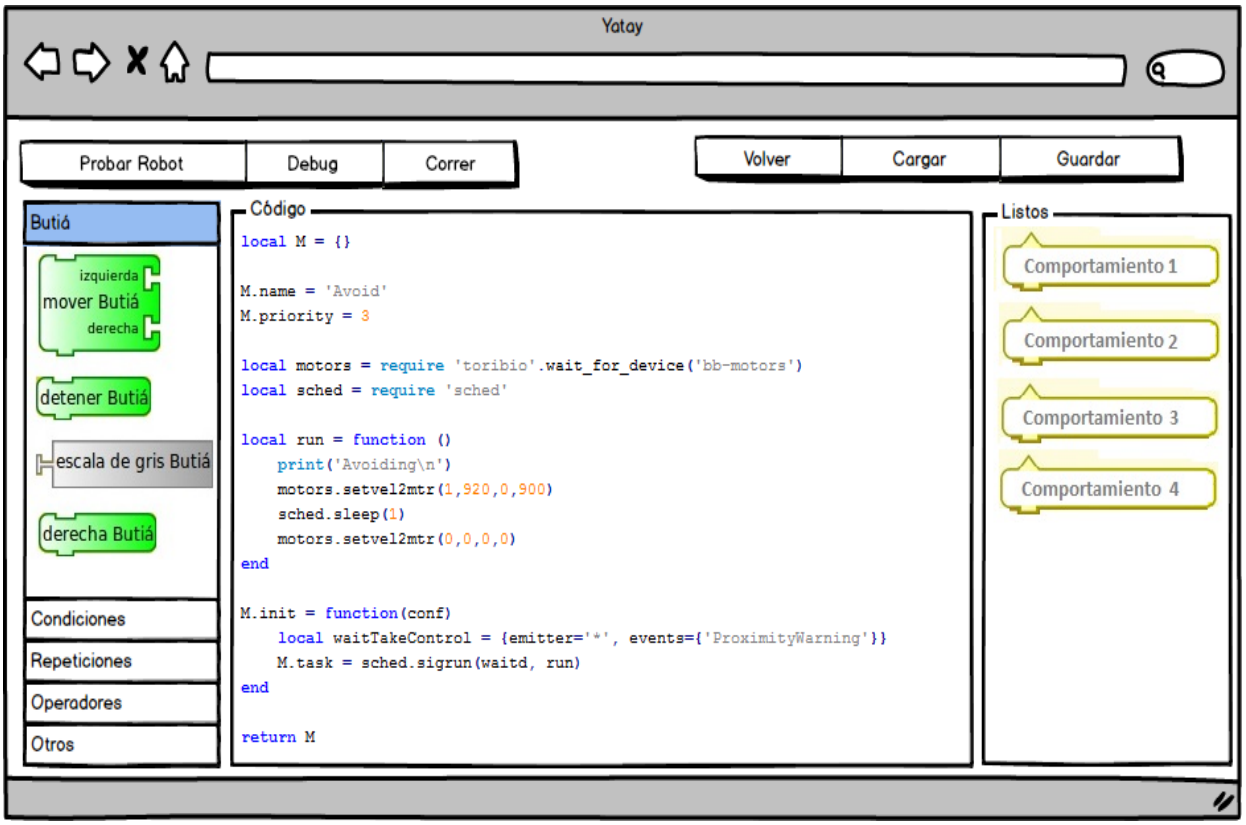


Imagen 3: Vista del código (en este sistema será código Lua) generado por el comportamiento en edición. Si se presiona volver se regresa a la imagen 1.

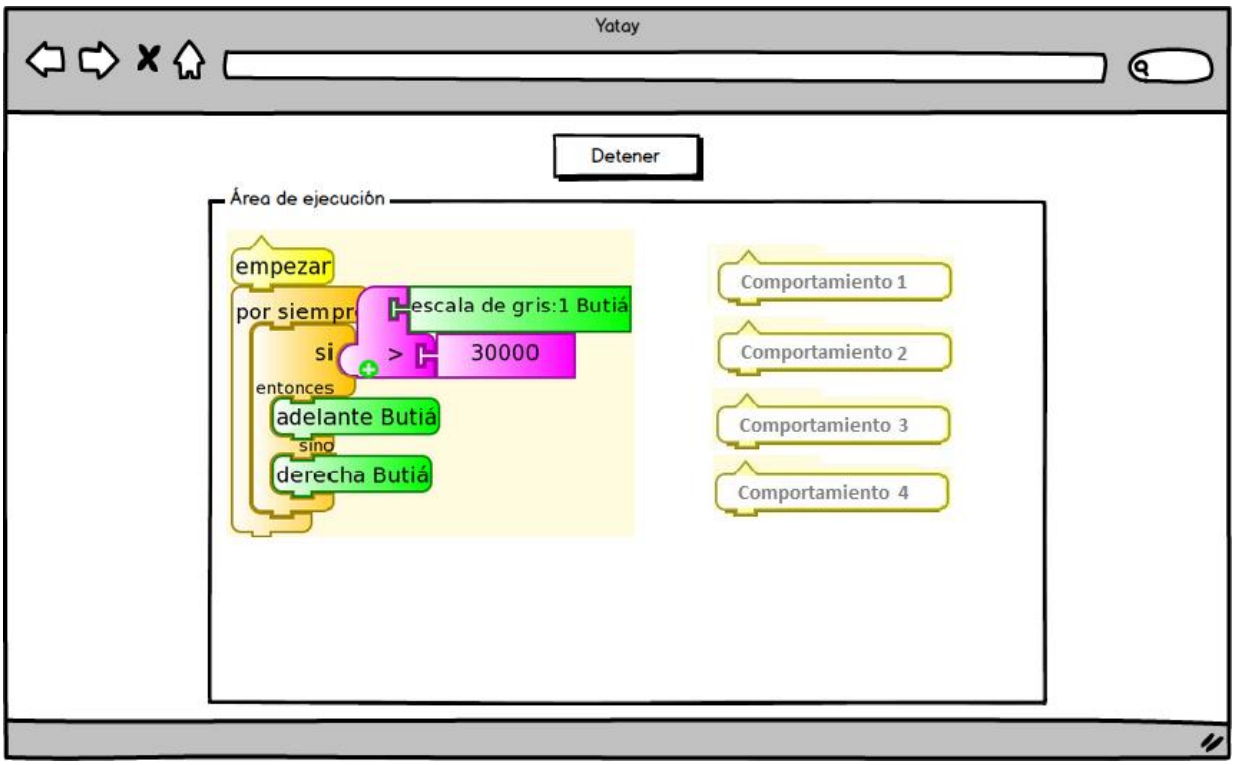


Imagen 4: Vista de ejecución de los comportamientos. Si se encuentra en modo depuración se iluminaran los comportamientos que se están ejecutando así como sus bloques.

4.2. Bocetos de dispositivos de menor pantalla

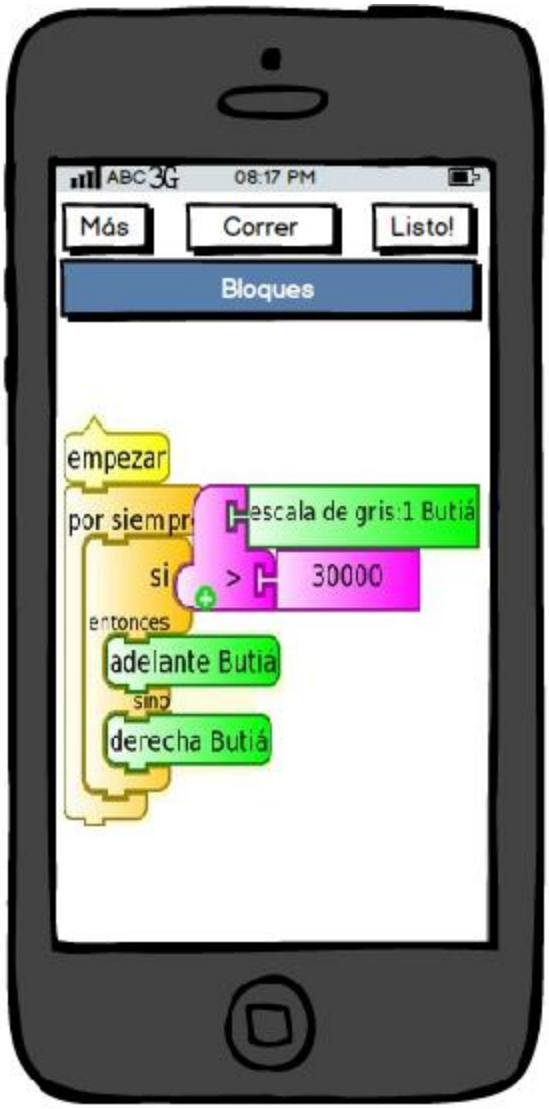


Imagen 1: Pantalla principal, entorno de trabajo. En el centro se encuentra el comportamiento actualmente en edición. Si se presiona bloques se despliega la imagen que se mostrará a continuación. Si se presiona Listo! se mueve el comportamiento al conjunto de comportamientos listos.

Imagen 2: Pantalla de bloques. Se disponibiliza todos los bloques que pueden ser agregados al comportamiento en edición.

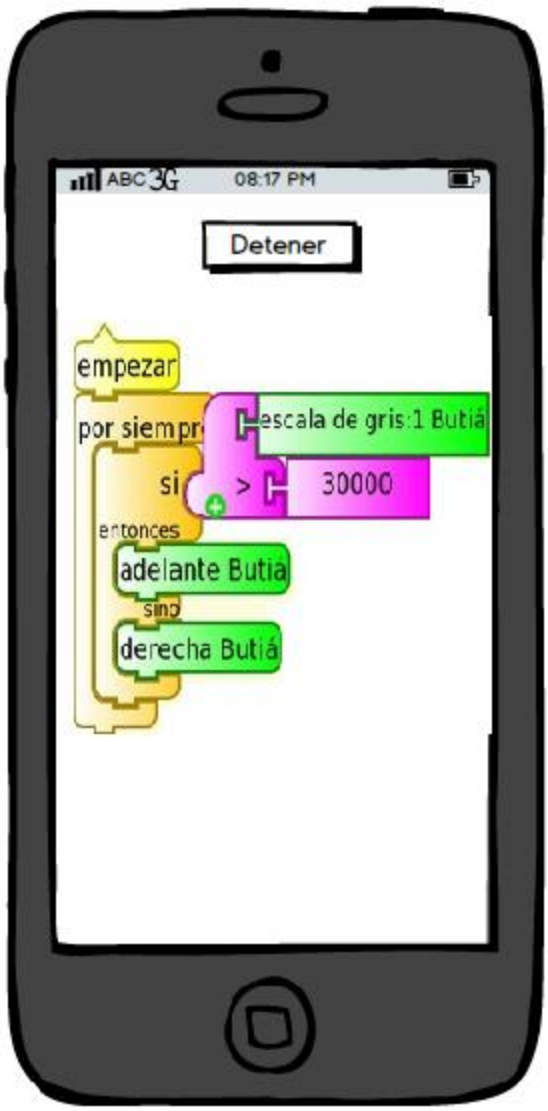


Imagen 3: Pantalla de ejecución, si se encuentra en depuración se iluminaran los bloques que están siendo ejecutados.

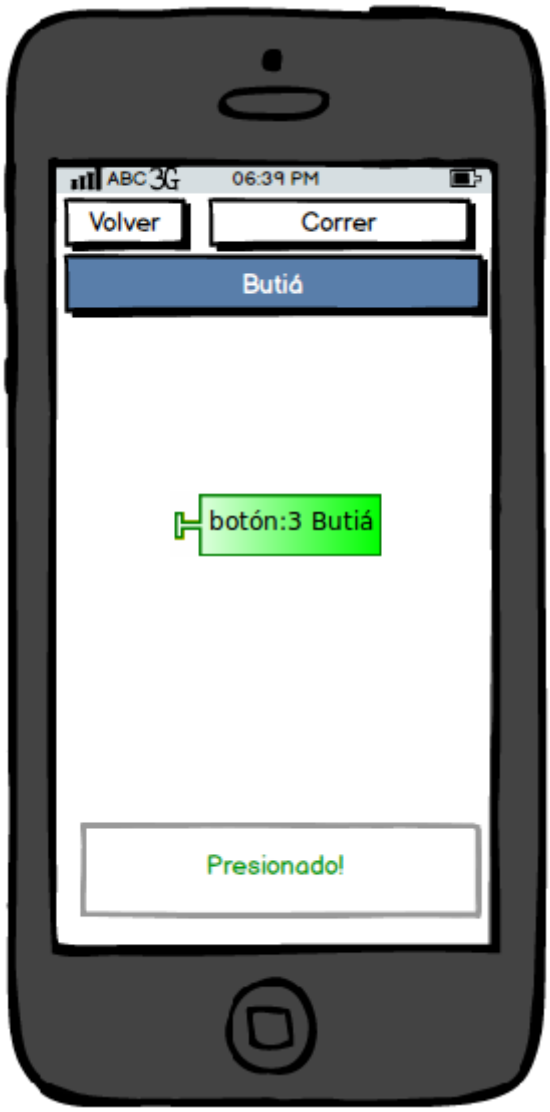


Imagen 4: Pantalla de ejecución rápida. Permite probar los sensores y actuadores del robot sin la necesidad de crear un comportamiento que lo contenga. Es útil a los efectos de la calibración del robot. En el centro muestra el bloque que se ejecutó. Arriba, el menú Butiá permite acceder a los bloques de los sensores y actuadores disponibles para ejecutar. Abajo muestra el resultado de la ejecución del bloque.

Referencias

[1] Nebel, Andres; Rozza, Renzo. IDE Android para la Programación de Comportamientos Robóticos: Especificación de Requerimientos. Proyecto de grado. Universidad de la República, Facultad de Ingeniería, Montevideo, 2013.

[2] Nebel, Andres; Rozza, Renzo. IDE Android para la Programación de Comportamientos Robóticos: Estado del Arte. Proyecto de grado. Universidad de la República, Facultad de Ingeniería, Montevideo, 2013.

[3] TortuBots. Wiki Proyecto Butiá [en línea]. Actualizada: junio de 2013 [Fecha de consulta: julio de 2013]. Disponible en Web: <http://www.fing.edu.uy/inco/proyectos/Butiá/TortuBots>

[4] Scratch [en línea]. Actualizada: julio de 2013. [Fecha de consulta: julio de 2013]. Disponible en Web: <https://scratch.mit.edu>

Proyecto de Grado

IDE Android para la Programación de Comportamientos Robóticos

Plan de pruebas

Andrés Nebel - Renzo Rozza

Tutores
Ing. Andrés Aguirre, Ing. Rafael Sisto

24 de Abril del 2014

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay

Índice

1. Objetivos 3
2. Estrategia 3
3. Casos de Prueba 3

1. Objetivos

Verificar que las funcionalidades desarrolladas cumplan con su especificación y detectar la presencia de fallas a la hora de utilizar el Sistema.

2. Estrategia

Se realizarán casos de pruebas para las funcionalidades principales del sistema especificando la entrada y la salida esperada así como cuales funcionalidades se están testeando en el caso. Se diseñan además pruebas para verificar el cumplimiento de requerimientos no funcionales. Los casos de prueba deben ser probados en cada uno de los dispositivos a los cuales apunta esta aplicación, estos son: Tablets de 7" y 10", celulares con resolución mayor a 320 x 480 y computadoras de escritorio.

3. Casos de Prueba

N°	Funcionalidad	Entrada	Salida esperada
1.1	Ingreso (1era vez)	El usuario ingresa por primera vez a la aplicación.	Se despliega un mensaje de bienvenida solicitando al usuario identificar el proyecto de trabajo. Dicho mensaje no puede saltarse sin identificar el proyecto.
1.2	Ingreso (1era vez)	El usuario ingresa un nombre de proyecto y presiona "Comenzar".	El mensaje de bienvenida desaparece y el tutorial comienza.
1.3	Ingreso (1era vez)	1) El usuario presiona cargar 2) Selecciona un proyecto y presiona "Comenzar".	1) El sistema disponibiliza los proyectos existentes en la base de datos. 2) El mensaje de bienvenida desaparece y el tutorial comienza.
2	Ingreso (No 1era vez)	El usuario ingresa al sistema luego de haber ingresado previamente a este.	El sistema mantiene el identificador de usuario y su nombre de proyecto. Si había comportamientos en edición la última vez que ingresó, estos se disponibilizan al usuario, pero solo estos y ninguno más. La paleta de Butiá muestra los sensores y actuadores disponibles actualmente. Y deshabilitados los que están configurados

			en el archivo XML pero no están conectados.
3	Visibilidad (Tablet)	Ingreso al sistema desde Tablets de 10" y de 7".	La interfaz gráfica puede visualizarse de forma correcta, siendo ésta similar a la especificada para Tablets en el documento Manual de Usuario.
4	Visibilidad (Celular)	Ingreso al sistema desde Celulares con pantalla igual o mayor a 320x480px.	La interfaz gráfica puede visualizarse de forma correcta, siendo ésta similar a la especificada para Celulares en el documento Manual de Usuario.
5	Drag and drop	1) Se ingresan mediante drag and drop dos bloques conectables. 2) Se realiza drag and drop para conectarlos entre sí. 3) Se realiza drag and drop para moverlos como conjunto.	1) El drag and drop es fluido y manejable. 2) Los bloques se conectan entre sí. 3) Los bloques se mueven de forma fluida en conjunto.
6.1	Creación de comportamientos	1) Se crea un comportamiento de nombre "Adelante" que no tenga disparador, con prioridad 1 y con un bloque de mover Butiá hacia adelante conectado. 2) Se lo marca como listo.	1) El comportamiento es creado correctamente. 2) Se enlista como comportamiento listo vaciando el área de trabajo o pizarra.
6.2	Creación de comportamientos	1) Se crea un comportamiento de nombre "GrisIzq" con prioridad 2. Como disparador tiene una condición sensor gris 1 mayor a cierto valor identificado con el color negro en el ambiente de prueba. Como acción debe tener un bloque de girar el Butiá a la derecha. 2) Se lo marca como listo.	1) El comportamiento es creado correctamente. 2) Se enlista como comportamiento listo vaciando el área de trabajo o pizarra.
6.3	Creación de comportamientos	1) Se crea un comportamiento de nombre "GrisDer" con prioridad 2. Como disparador tiene una condición sensor gris 2 mayor a cierto valor identificado con el color negro en el ambiente de prueba. Como acción debe tener un bloque de girar el Butiá a la izquierda. 2) Se lo marca como listo.	1) El comportamiento es creado correctamente. 2) Se enlista como comportamiento listo vaciando el área de trabajo o pizarra.
6.4	Creación de comportamientos	1) Se crea un comportamiento de nombre "Detener" con prioridad 3. Como disparador tiene una condición	1) El comportamiento es creado correctamente.

		sensor gris 1 y sensor gris 2 mayor a cierto valor identificado con el color negro en el ambiente de prueba. Como acción debe tener un bloque de detener el Butiá. 2) Se lo marca como listo.	2) Se enlista como comportamiento listo vaciando el área de trabajo o pizarra.
7	Comportamientos listos	Se seleccionan al azar comportamientos marcados como listos.	El sistema despliega los bloques correspondientes al comportamiento seleccionado marcando previamente como listo, si existía, el comportamiento en edición en la pizarra o área de trabajo.
8.1	Eliminar	1) Se presiona el botón eliminar. 2) Se selecciona “Solo pizarra”.	1) El sistema despliega un modal para seleccionar “Solo pizarra” o “todo”. 2) Al presionar “Solo pizarra” se eliminan todos los bloques que están en el área de trabajo. Los comportamientos listos no se eliminan.
8.2	Eliminar	1) Se presiona el botón eliminar. 2) Se selecciona “todo”.	1) El sistema despliega un modal para seleccionar “Solo pizarra” o “todo”. 2) Al presionar “todo” se eliminan todos los bloques que están en el área de trabajo y los marcados como listos.
9	AutoSave y Cargar	1) Se selecciona el botón de cargar (ver manual de usuario). 2) Se presiona el botón obtener. 3) Se seleccionan todos los comportamientos del proyecto actual. (Si se hacen en orden las pruebas, son los eliminados en la prueba 8) y se presiona “abrir”.	1) El sistema despliega el modal de cargar comportamientos. 2) El sistema despliega para cada nombre de proyecto en la base de datos, todos los comportamientos de estos. (Si se hacen en orden las pruebas, son los eliminados en la prueba 8) 3) Se recuperan y marcan como listos los comportamientos seleccionados.
10	Guardar Archivo	Se selecciona el botón de guardar (ver manual de usuario).	El sistema brinda un diálogo de descarga de un archivo con los bloques con los que trabajaba el usuario.
11	Cargar desde Archivo	1) Se selecciona el botón de cargar (ver manual de usuario). 2) Se presiona el botón “Seleccionar Archivo”. 3) Se presiona “abrir”.	1) El sistema despliega el modal de cargar comportamientos. 2) El sistema muestra un diálogo de selección de archivo en el file system. 3) El sistema recupera y marca como listos los comportamientos existentes en el archivo.

12	Ejecución	1) Teniendo creados los comportamientos del caso de prueba 6, se selecciona el botón de ejecución (ver manual de usuario). 2) Se selecciona el botón de ejecución sin debug (ver manual de usuario).	1) Desaparecen todos los botones quedando disponibles solo los botones de “Ejecución sin debug”, “Debug” y “Detener”. 2) Se ejecutan los comportamientos en el robot, siguiendo una ejecución acorde a la arquitectura de prioridades. Se muestran los resultados de los sensores que se están midiendo. Los bloques quedan inhabilitados para editar.
13	Debug	1) Teniendo creados los comportamientos del caso de prueba 6, se selecciona el botón de ejecución (ver manual de usuario). 2) Se selecciona el botón de Debug (ver manual de usuario).	1) Desaparecen todos los botones quedando disponibles solo los botones de “Ejecución sin debug”, “Debug” y “Detener”. 2) Se ejecutan los comportamientos en el robot, siguiendo una ejecución acorde a la arquitectura de prioridades. Se muestran los resultados de los sensores que se están midiendo. Se muestra en el área de trabajo el comportamiento que se está ejecutando, se ilumina el bloque que está actualmente en ejecución. Los bloques quedan inhabilitados para editar.
14	Detener y reiterar ejecución.	1) Mientras se está en el caso de prueba 12 o 13 se presiona detener. 2) Se comienza otra vez la ejecución del caso de prueba 12 o 13.	1) Se detienen los actuadores del robot y se restauran los botones iniciales. Se oculta el área de resultados. 2) Se producen los mismos resultados esperados de la sección 12 o 13.
15.1	Ver Código	Se presiona el botón de ver código (ver manual de usuario).	Se abre un modal con pestañas por cada comportamiento, en donde para cada pestaña se muestra el código Lua de dicho comportamiento.
15.2	Ver y Editar Código	1) Se edita el código de alguno de los comportamientos y se selecciona “Probar”. 2) Se detiene la ejecución.	1) Se ejecuta el código editado. 2) Se detiene la ejecución y se vuelve a mostrar el modal de ver código.
15.3	Guardar código editado	Dentro del modal de ver comportamiento se presiona “Guardar”.	El sistema brinda un diálogo de descarga de un archivo con el código lua editado.
16	Test Butiá	1) Se presiona el botón de calibrar (ver manual de usuario).	1) El sistema disponibiliza solo los botones de ejecución, detener y borrar. Se disponibiliza solo los bloques de “Butiá”.

		2) Se selecciona un bloque de actuador o sensor del Butiá y se presiona "Ejecutar". 3) Se presiona detener.	2) Se ejecuta el bloque de actuador o sensor y se disponibiliza en tiempo real la información del sensor seleccionado si corresponde. 3) Se restauran los botones iniciales y los demás bloques. Si el usuario tenía comportamientos guardados estos aparecen disponibles como comportamientos listos.
17	Funcionamiento de bloques	Se prueba la ejecución de cada uno de los bloques existentes, dentro de un contexto de bloques que permita dicha prueba.	Al ejecutarse, el bloque presenta el comportamiento descrito para dicho bloque.
18	Tiempo de ejecución.	Se ejecutan el caso de prueba 12.	Para cualquiera de los comportamientos, el sistema demora menos de un segundo en detectar la condición que lo dispara.
19	Tiempo de reacción del sistema.	Cualquier acción de drag and drop o presionado de botones.	El sistema no demora más de un segundo y medio en responder a la acción.
20	Test Butiá con múltiples usuarios	Repetir el caso de prueba 16 pero con 2 más usuarios al mismo tiempo.	El sistema devuelve a cada usuario el valor de los sensores correspondiente a lo que ejecutó dicho usuario. Sobre los actuadores, la ejecución de acciones distintas de varios usuarios al mismo tiempo no garantiza la ejecución deseada por dicho usuario por ejecutarse estos concurrentemente.
21	Ejecución con múltiples usuarios	Repetir el caso de prueba 12, pero siendo cada comportamiento del caso de prueba 6, creado por un usuario distinto.	Los comportamientos creados por los distintos usuarios se ejecutan de la misma forma esperada en el caso de prueba 12 para un único usuario.
22	Debug con múltiples usuarios	Repetir el caso de prueba 13, pero siendo cada comportamiento del caso de prueba 6, creado por un usuario distinto.	Los comportamientos creados por los distintos usuarios se ejecutan de la misma forma esperada en el caso de prueba 12 para un único usuario. Solo se iluminan los bloques del comportamiento de cada usuario cuando dicho comportamiento se disparó y está en ejecución. El resto de los casos igual mostrará los resultados de sensores y consola pero no iluminando bloques de comportamientos (pues no se están ejecutando).
23	Multilenguaje	Se presiona el botón de multilenguaje y se cambia el lenguaje.	Todos los textos desplegados por la aplicación se encuentran en el lenguaje seleccionado.

24	Comportamientos largos	Se crea un comportamiento cuyo largo y ancho superen ampliamente el tamaño predeterminado del área de trabajo.	El sistema se autoajusta al tamaño del comportamiento y permite a través de los scrollbars la correcta visualización del comportamiento.
25	Estabilidad mantenida en el tiempo	Se utiliza la aplicación durante un tiempo aproximado de una hora.	El sistema continúa respondiendo bien a los casos de prueba.
26	Bloques de sensores que no están conectados	Se cargan o se tiene en edición comportamientos que poseen sensores que no se encuentran más conectados en el robot.	Se muestran dichos bloques deshabilitados y su generación de código no genera ningún código.
27	Alteración de sensores conectados al robot Butiá	Se desconectan o conectan sensores al robot Butiá,	Si no se está en ejecución, entonces en un periodo de tiempo de hasta 10 segundos, el sistema refresca los bloques de la paleta de Butiá mostrando los sensores disponibles actualmente.
28	XML Devices	1. No existe el archivo. 2. Se configura un archivo con el atributo “from” en “bobot”, y se definen devices para los sensores. 2.1. Se agrega el atributo “except” con devices conocidos que no son usados. 2.2. Se agregan devices con alias para los sensores y actuadores (probando el atributo “values” para valores por defecto). 2.3. Se define un device genérico para excluir funciones. 3. Se configura un archivo con el atributo “from” en “XML”, y se definen devices para los sensores. 3.1-3. Igual a 2.1-3.	1. Se crean los bloques parseando bobot y se muestran con los nombres (hardcode para Butiá) y puertos. 2.1. Se muestran los bloques de la categoría Butiá con el nombre y puerto, no se muestran los devices descartados. 2.2. Se actualizan los nombres para los aliases agregados, y se despliegan nuevos bloques para las funciones por defecto de los actuadores. 2.3. Se eliminan las funciones que fueron deshabilitadas para el device genérico. 3. Se debe cumplir lo mencionado en el punto 2, considerando únicamente los datos del XML.