Proyecto de Grado

Yatay

(IDE Android para la Programación de Comportamientos Robóticos)

Informe Final

Andrés Nebel - Renzo Rozza

Tutores
Andrés Aguirre, 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

Índice de Figuras

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 inclusión de la robótica educativa 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 y mejoren tanto su cooperación como su capacidad de formular hipótesis, investigar soluciones factibles y obtener conclusiones. [13][41]

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 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 que se distinguen por ser bioinspiradas. Éstas pueden ser enfocadas para promover el desarrollo del educando, ya que intentan reflejar la conducta animal mediante módulos que implementan comportamientos sencillos (sensar y actuar) o la combinación de ellos para lograr comportamientos más complejos. Dando lugar para que los procesos de enseñanza y aprendizaje se lleven a cabo como un proceso dinámico, participativo e interactivo por parte del sujeto.

Por otro lado, los avances tecnológicos en los últimos años han fomentado una evolución de los dispositivos móviles tan excepcional como veloz. Las Tablets y teléfonos inteligentes se han convertido en dispositivos con capacidades de procesamiento y recursos inimaginados, reemplazando en algunos casos el uso de computadoras portátiles o de escritorio. Siendo un

ejemplo el Plan Ceibal, que en este año (2014) entregó 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, es realmente una necesidad que los métodos de enseñanza se mantengan a la par de 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 sección 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 sección 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 sección 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

ACTIVIDAD	Mes Inicio	Mes Fin
Propuesta del proyecto	03/2013	03/2013
Estado del Arte	04/2013	08/2013
Prototipos	05/2013	10/2013
Arquitectura y diseño	07/2013	08/2013
Mantenimiento de la web	08/2013	04/2014
Implementación	09/2013	02/2014
Verificación	02/2014	03/2014
Documentación Final	03/2014	04/2014

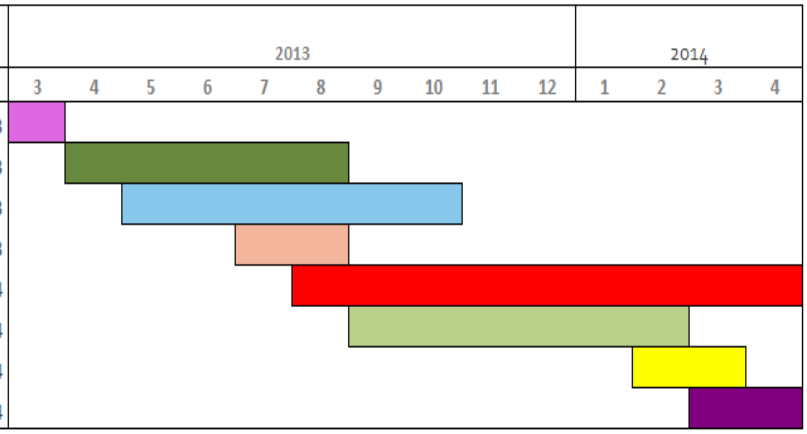


Figura 1: Etapas del proyecto, dedicación en meses por actividad.

Capítulo 2

Resumen del estado del arte

En este capítulo se brinda al lector una introducción sobre los conceptos fundamentales dentro del proyecto que permiten comprender el vocabulario de este documento. Se puede obtener más información sobre los temas tratados en el documento Estado del arte [4].

2.1. Paradigmas y arquitecturas robóticas

Llamamos arquitectura de un robot autónomo a la organización de sus capacidades sensoriales, de procesamiento y de acción para conseguir una serie de comportamientos interactuando con un cierto entorno [22]. De modo informal, las arquitecturas proporcionan la manera general de organizar un sistema de control describiendo un conjunto de componentes y la forma en que interactúan. Esta determina los comportamientos que exhibe un robot autónomo y cómo interactúan entre ellos, definiendo también como 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 una cantidad desbordante 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.

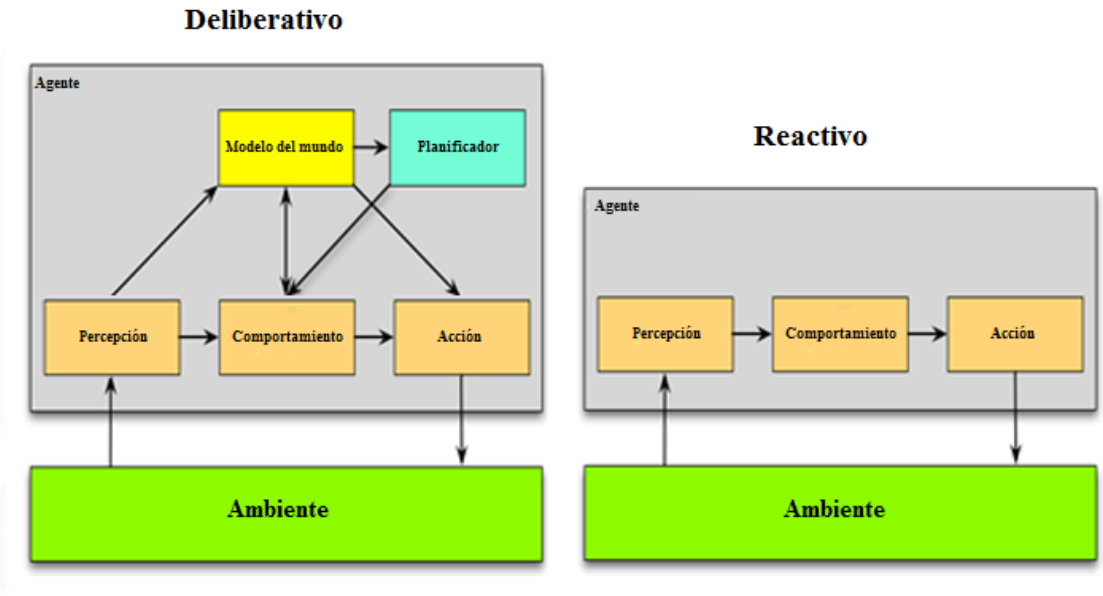


Figura 2: Diagrama comparativo entre arquitecturas deliberativas y reactivas. [16]

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 facilidad a la hora de enseñar, permitiendo apuntar las experiencias educativas a un rango de edades mayor, incluyendo así a niños en sus primeros años escolares.

2.1.2. Arquitecturas robóticas

Dentro del paradigma reactivo existen varias arquitecturas robóticas conocidas que implementan de distinta forma el concepto principal de sensar-actuar del paradigma.

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.

2.1.2.2 Basada en prioridades

Otro tipo de arquitectura del paradigma reactivo se basa en el establecimiento de prioridades. En esta arquitectura cada comportamiento se implementa en forma de máquina de estados (al igual que en Subsumption) 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 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 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. [26]

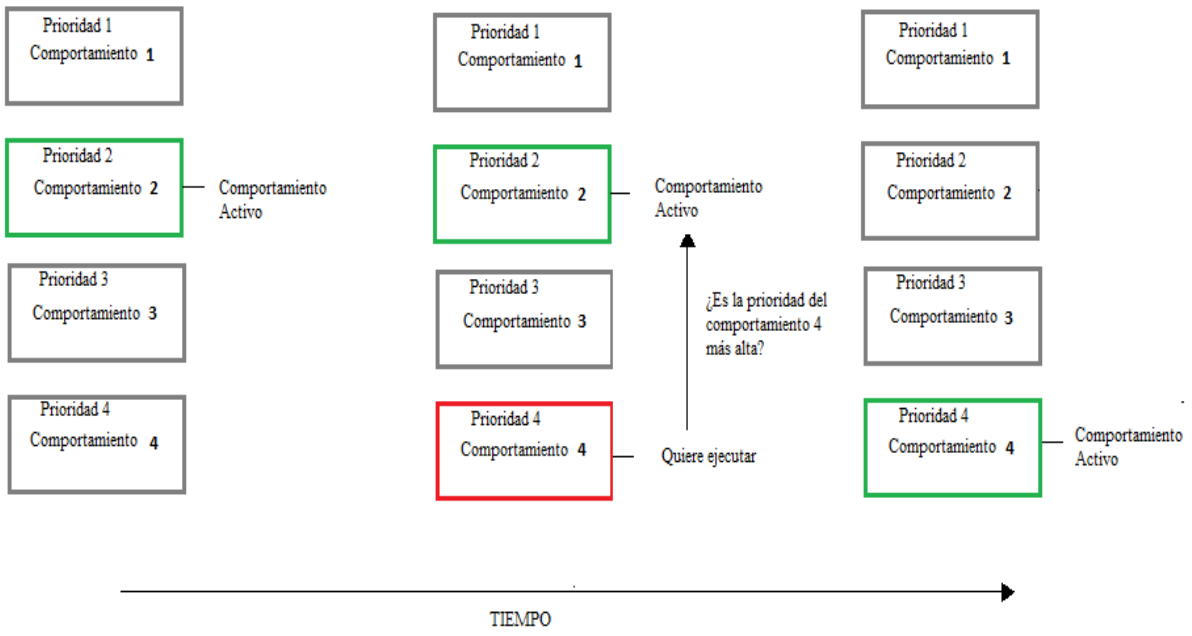


Figura 3: Diagrama representativo para una arquitectura basada en prioridades.

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 [40]. 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.

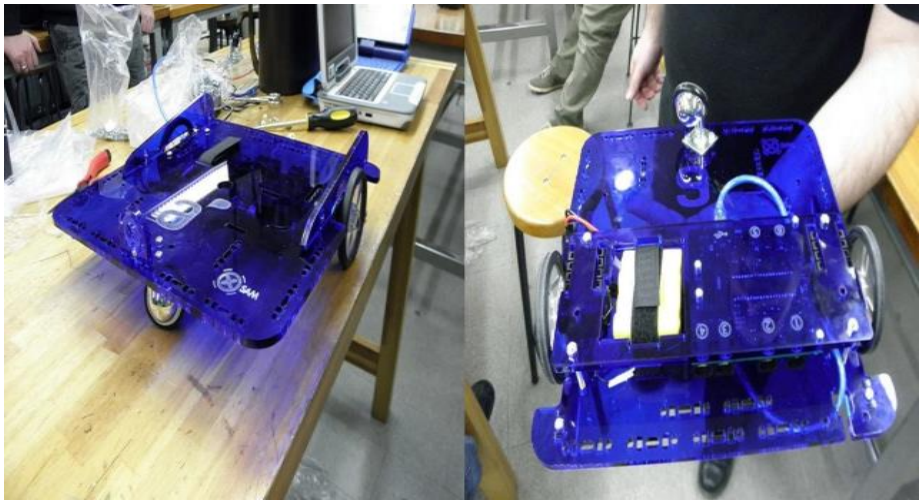


Figura 5: Robot Butiá 2.0 parte superior (izquierda) y parte inferior (derecha).

2.2.2. Complementos (Raspberry Pi)

Para complementar las funcionalidades del robot Butiá últimamente se maneja la posibilidad de agregar a su kit una Raspberry Pi [18], 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.

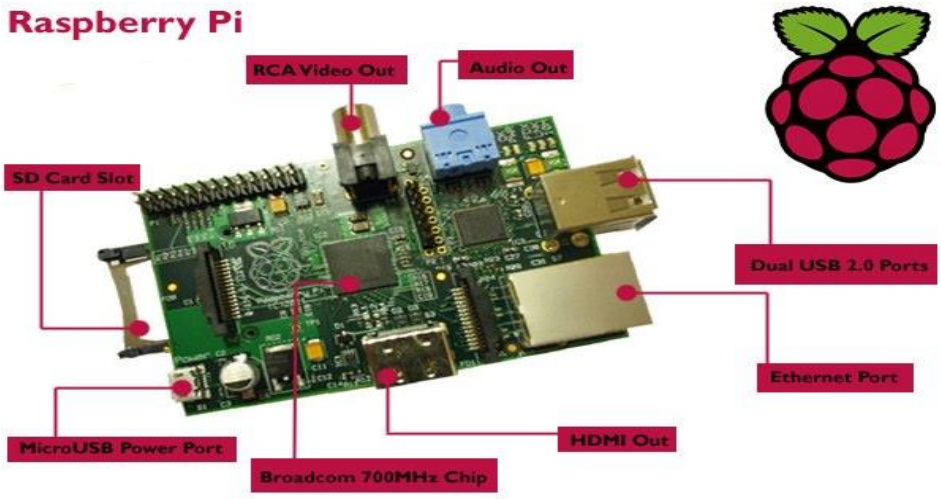


Figura 6: Componentes de la placa Raspberry Pi.

Contando con esta placa en el kit del robot Butiá se abre el abanico de posibilidades para realizar una infinidad de actividades; entre estas se pueden resaltar aquellas que benefician al desarrollo del IDE en cuestión, como la posibilidad de lograr 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.

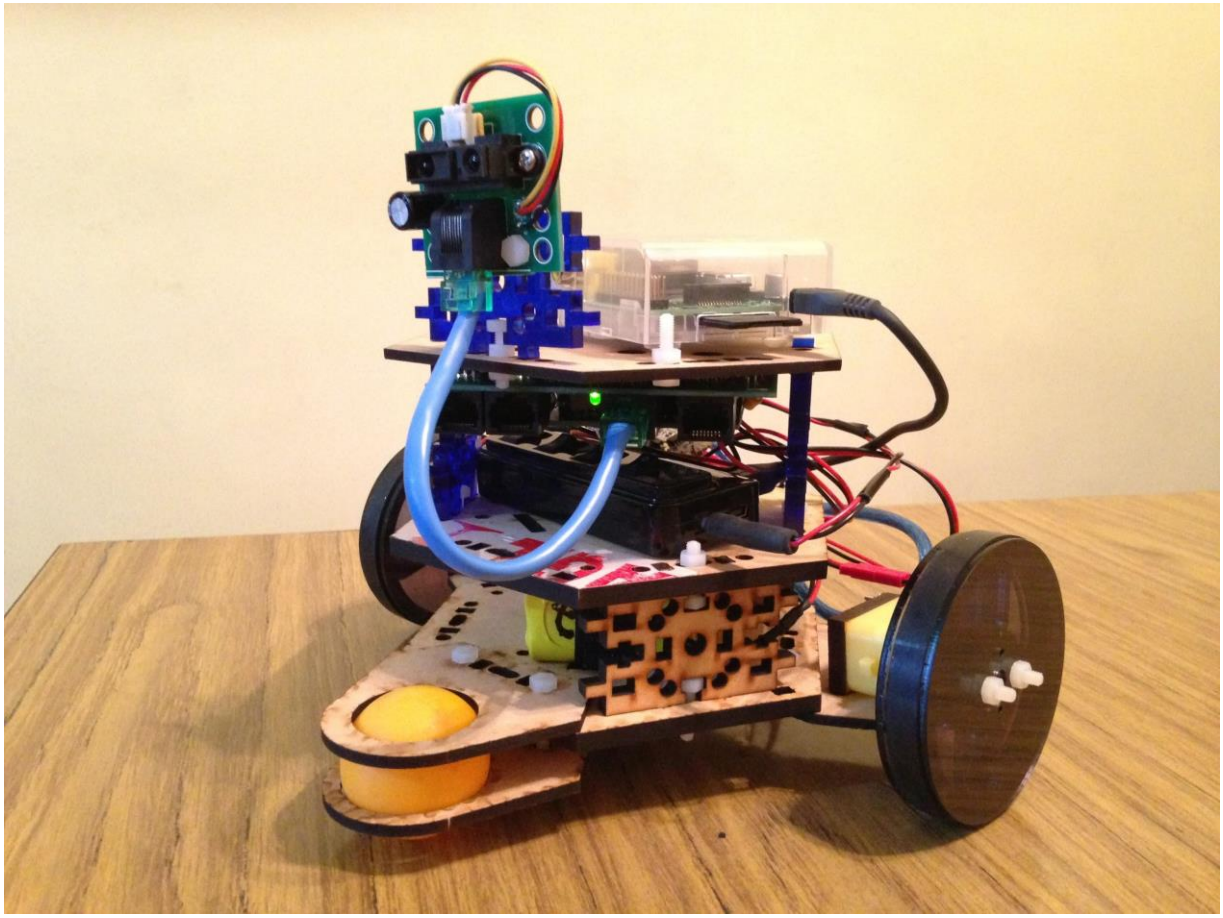


Figura 7: Robot Butiá/Yatay.

2.3 Constructivismo y enseñanza de la robótica

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* " (Mario Carretero, 1997 [8]).

Para esta corriente, el conocimiento de un alumno "*no es una copia fiel de la realidad, sino una construcción del ser humano"* (Mario Carretero, 1997 [8]), a partir de sus conocimientos

previos. Esta construcción es un proceso activo del estudiante, realizado de forma dinámica y participativa. [8][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 [34]. 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].

La "Robótica Pedagógica" 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 (Ruiz 2007) [28]. 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 el documento Estado del arte [4]. En 1975 en Francia en la Universidad Du Maine, en Le Mans, la universidad Autónoma de México en 1989, 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 (Fundación Omar Dengo, 2004) [28], también el Proyecto Butiá en el ámbito local desde el 2009.

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 construir 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 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 han influenciado las herramientas educativas de hoy en día. Alan Kay cabe destacar, continuó sus trabajos y actualmente trabaja en el proyecto One Laptop Per Child (OLPC) de manera activa.

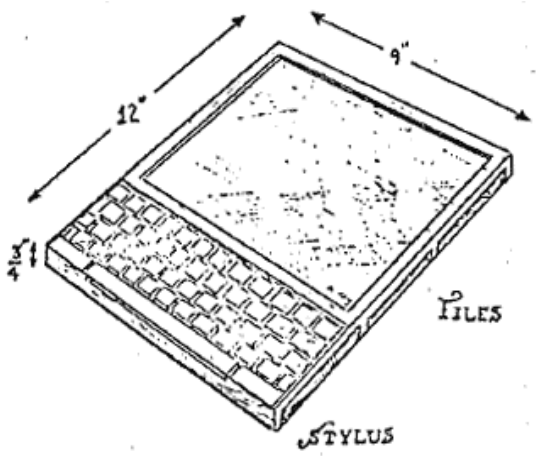


Figura 9: Ilustración de la Dynabook descrita por Alan Kay en su paper.

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.

24.1. 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, yendo contra de 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.

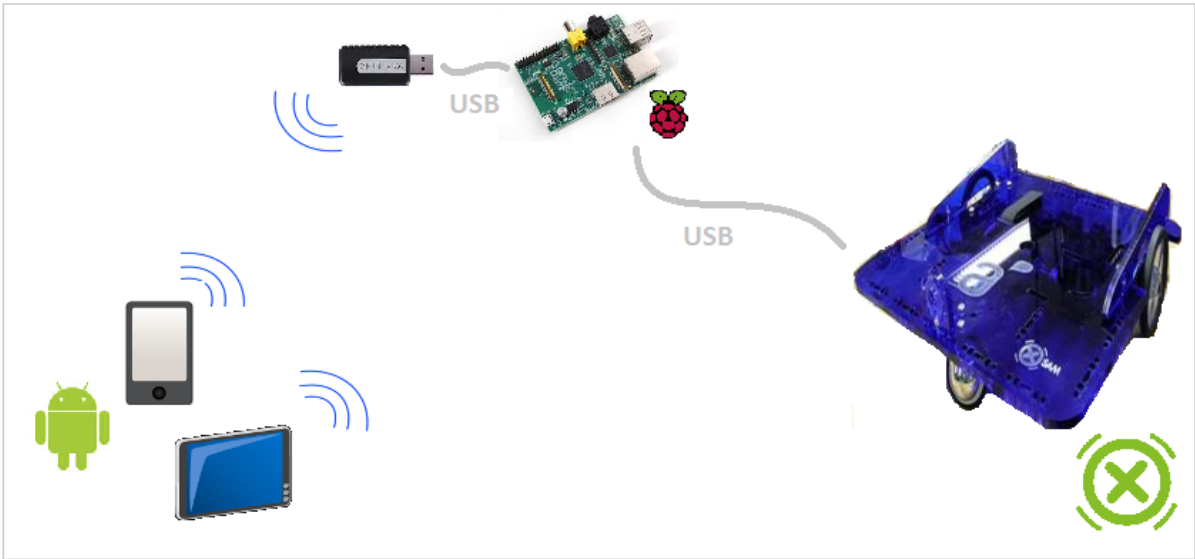


Figura 11: Diagrama de componentes haciendo uso de la placa Raspberry Pi.

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.

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. Similar para el caso de subsumption. 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 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 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 [23] 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.

```
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 [24] 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 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 [33] 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 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.

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 concreto 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 [27]. 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 [25]. 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) [36][39]. Con este complemento se permite a los alumnos programar usando un LPV distintos comportamientos para controlar al robot Butiá.

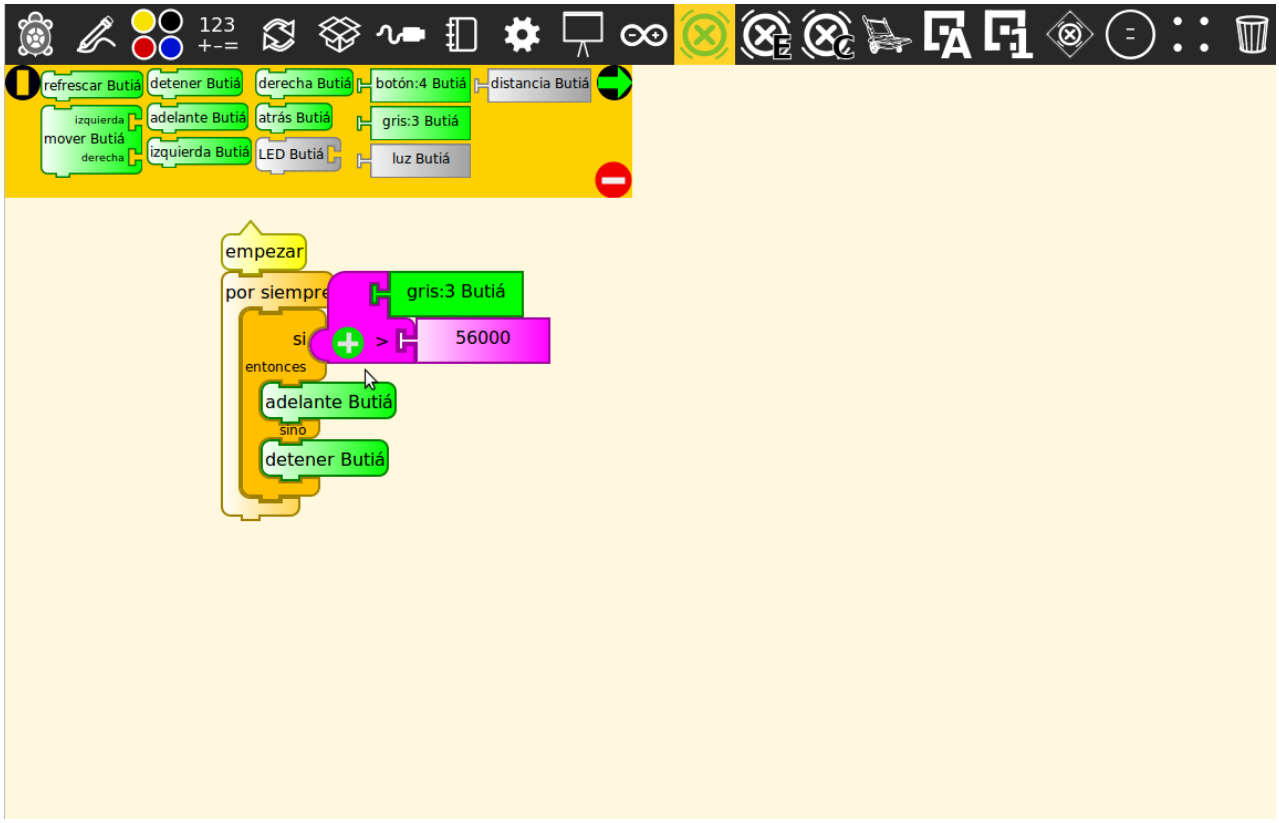


Figura 16: Ejemplo de código en Tortubots.

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 [4]).

2.7.1. Blockly

Blockly [9] es un LPV open source creado 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.

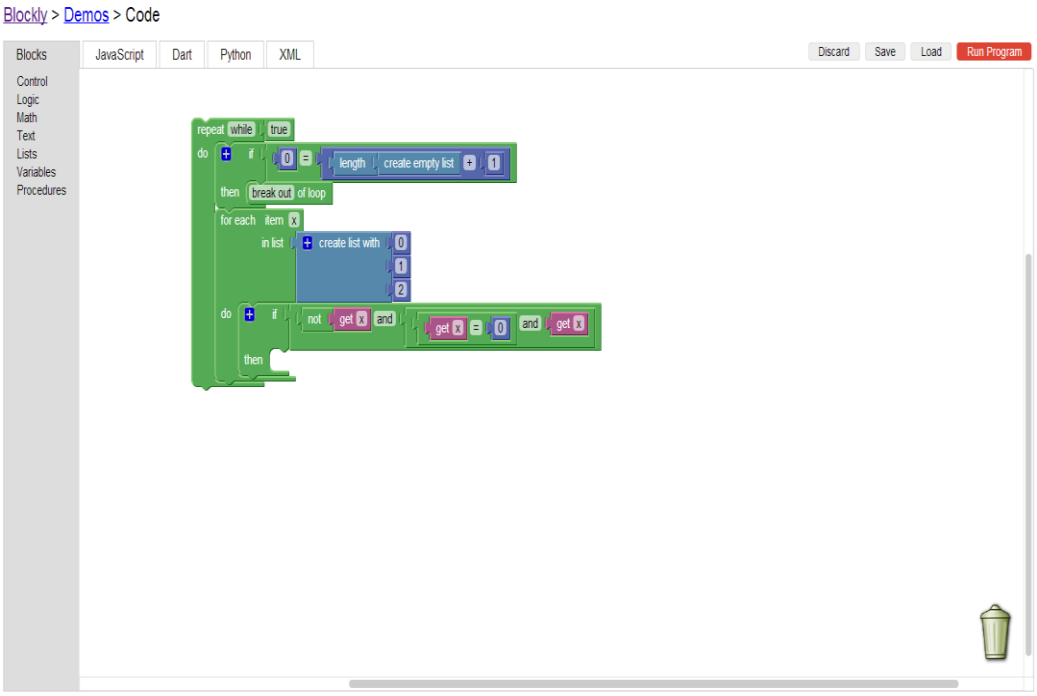


Figura 17: Interfaz de Blockly.

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. Tiene la característica también de ser muy cambiante actualmente, con de 1 a 3 commits diarios. Esto 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 [4] 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 (al ser bioinspirado) y es fácil predecir el comportamiento 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 rango enorme 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. Conociendo las limitadas capacidades de procesamiento de los dispositivos móviles esto significa una mejor utilización de los recursos. 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 [4], 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 bastante 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 [35]. 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 [37]. 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"*** [3]. 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

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 un sistema operativo GNU/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: 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 [4]. No será un requisito trabajar de manera cooperativa sobre el mismo comportamiento o la misma paleta.

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

3.2.2. 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 Estado del Arte [4].

Kit Robótico: El robot al que estará orientado este sistema, es principalmente el robot Butiá definido en el documento Estado del Arte [4]. Sin embargo, se debe mantener una cierta modularización que permita extender el sistema a otros kit robóticos.

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.

3.2.3. 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 [4]. 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 está definido gráficamente en el documento ***"Pautas para la Interfaz de Usuario"*** [6].

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

3.3. Requerimientos no funcionales

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 deben estar por debajo de esa cantidad.

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.

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 [4].

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*"** *[2]*. 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.

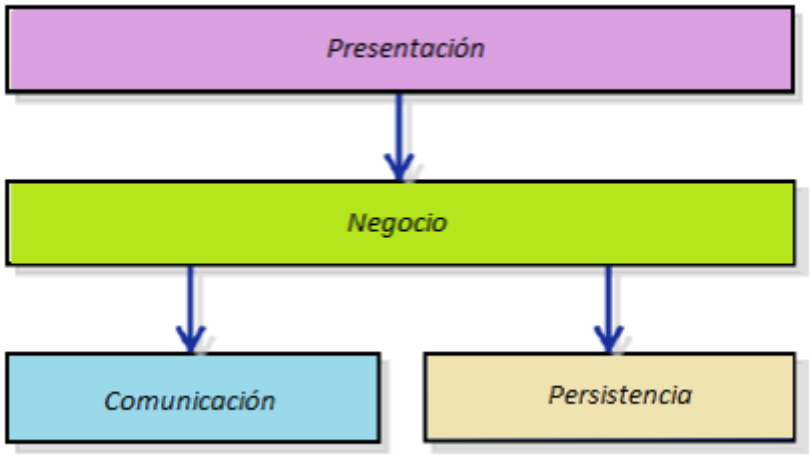


Figura 18: Diagrama de estilo arquitectónico.

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.

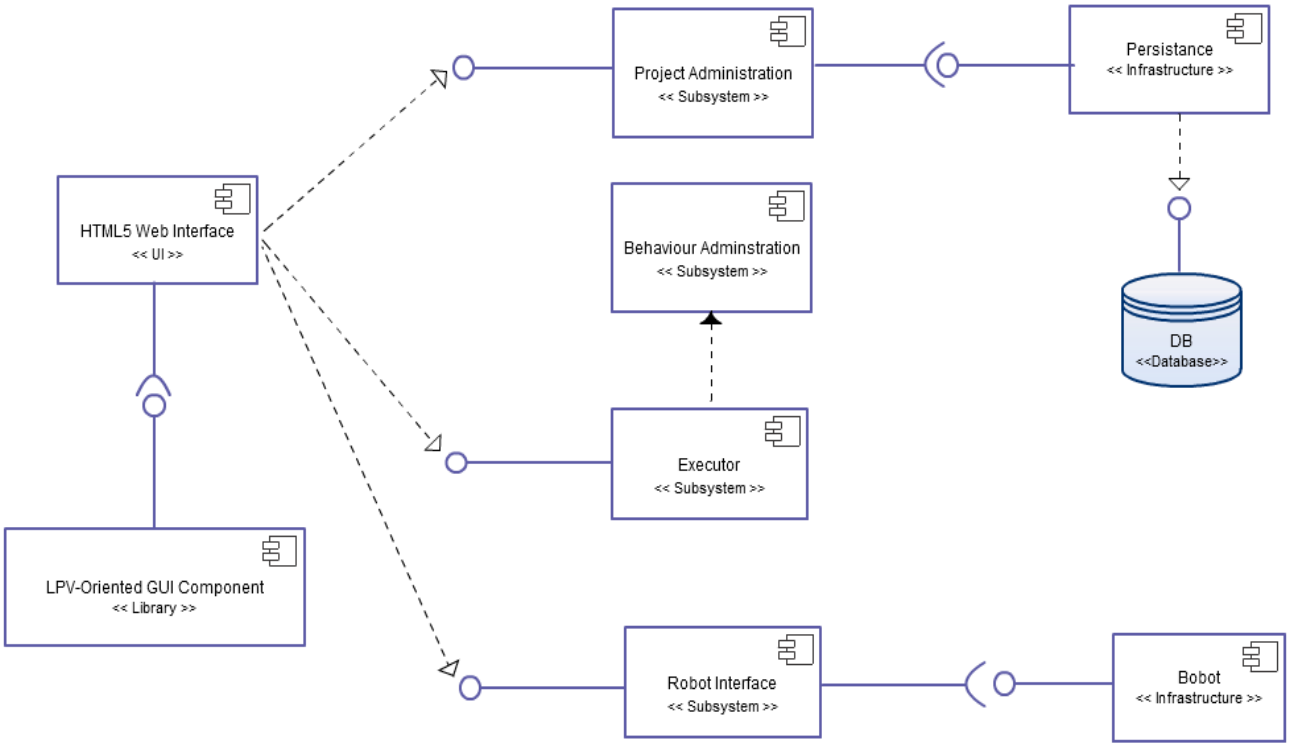


Figura 19: Diagrama de subsistema estáticos.

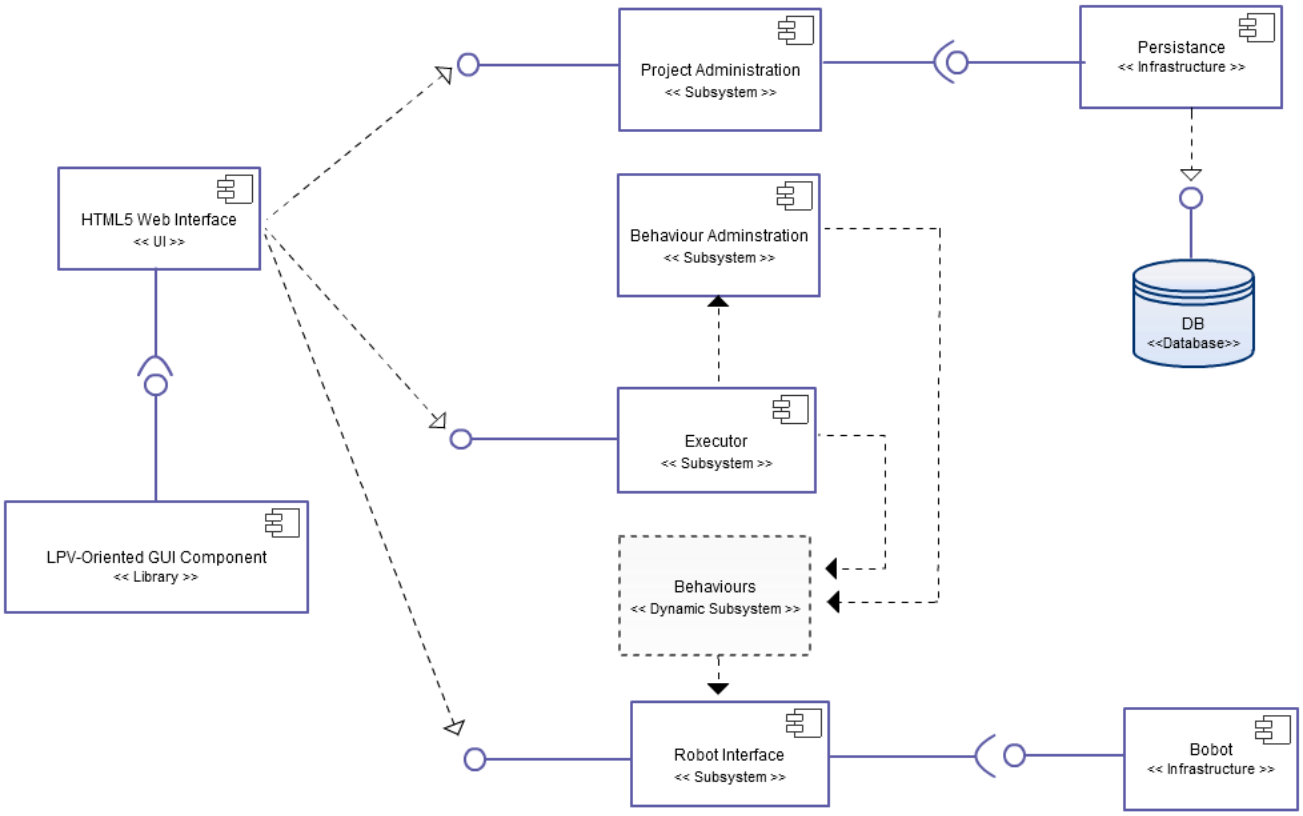


Figura 20: Diagrama de subsistemas estáticos y dinámicos.

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.

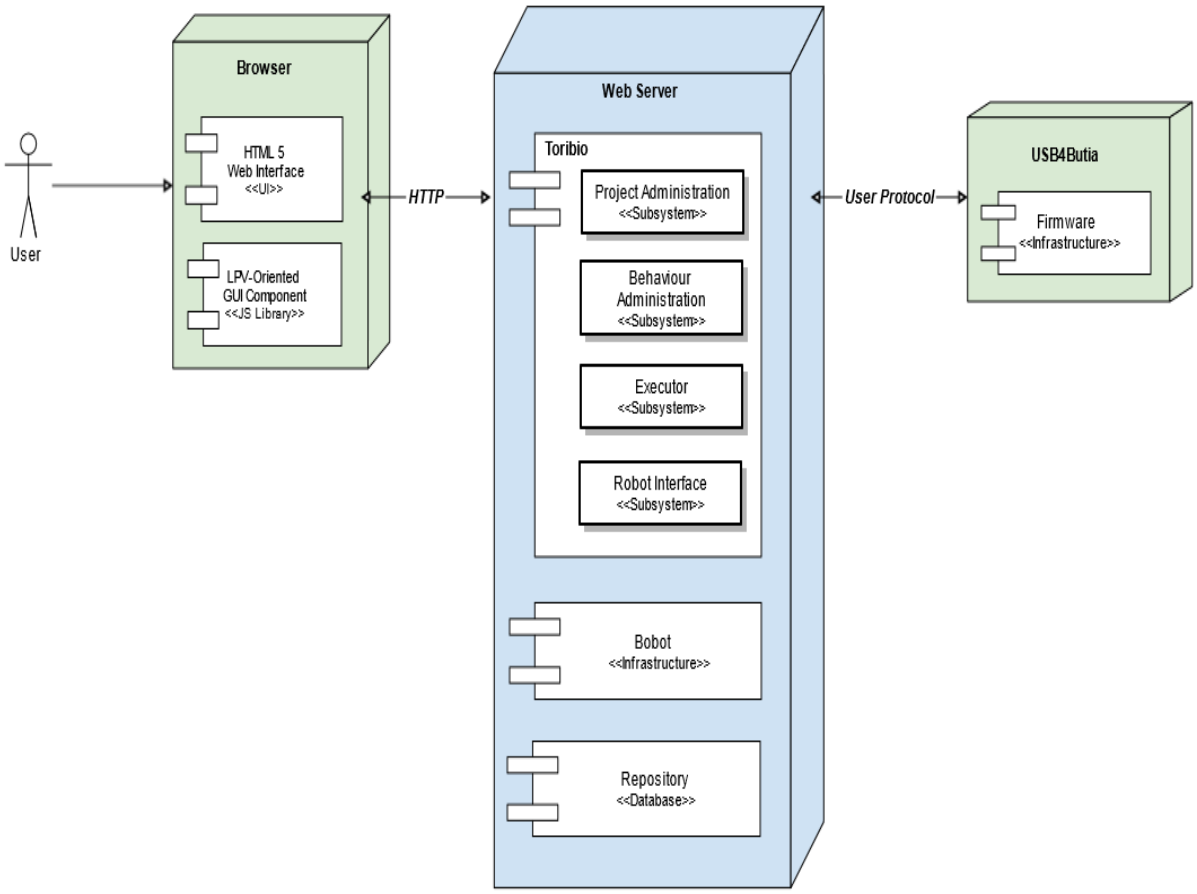


Figura 21: Diagrama de distribución.

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.

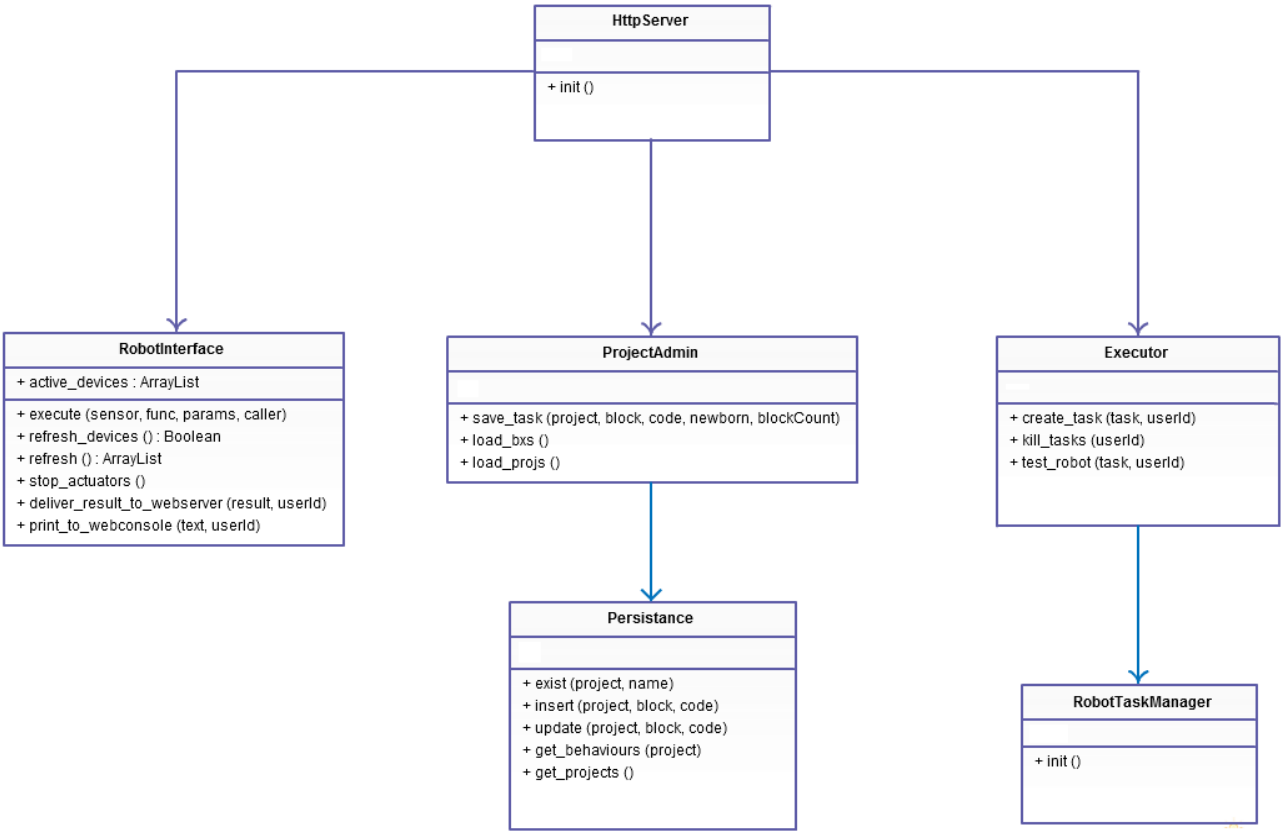


Figura 22: Diagrama de clases server-side.

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.

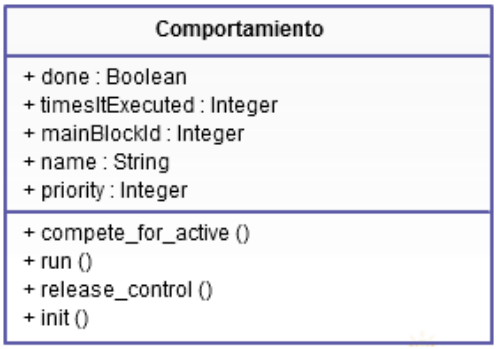


Figura 23: Diagrama de clase de un comportamiento genérico.

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.

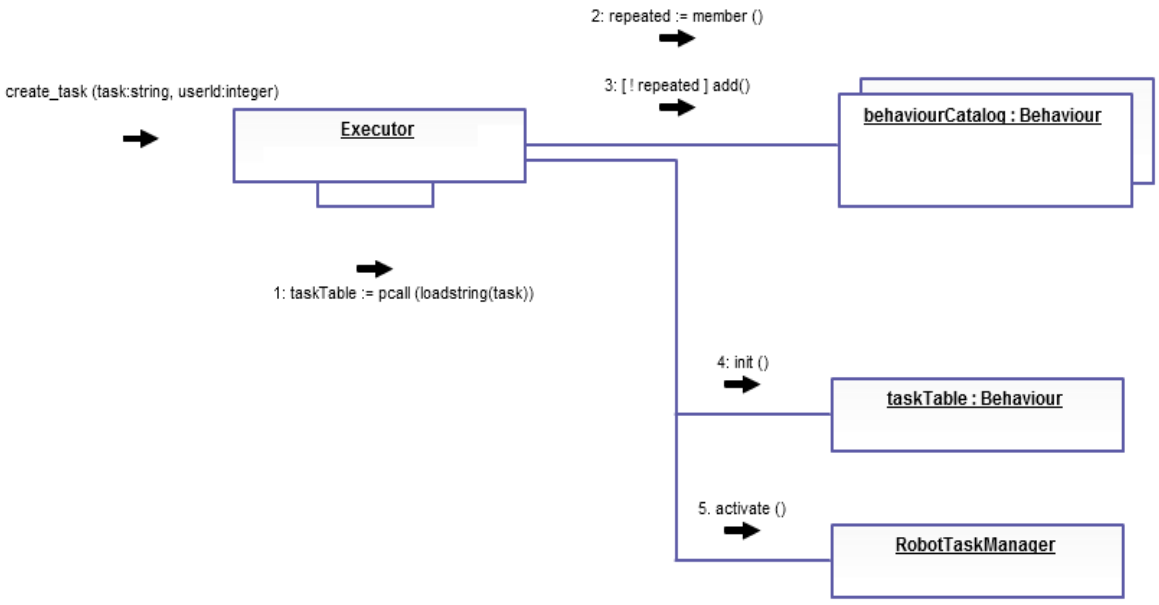


Figura 24: Diagrama de comunicación para ejecutar comportamiento.

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 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).

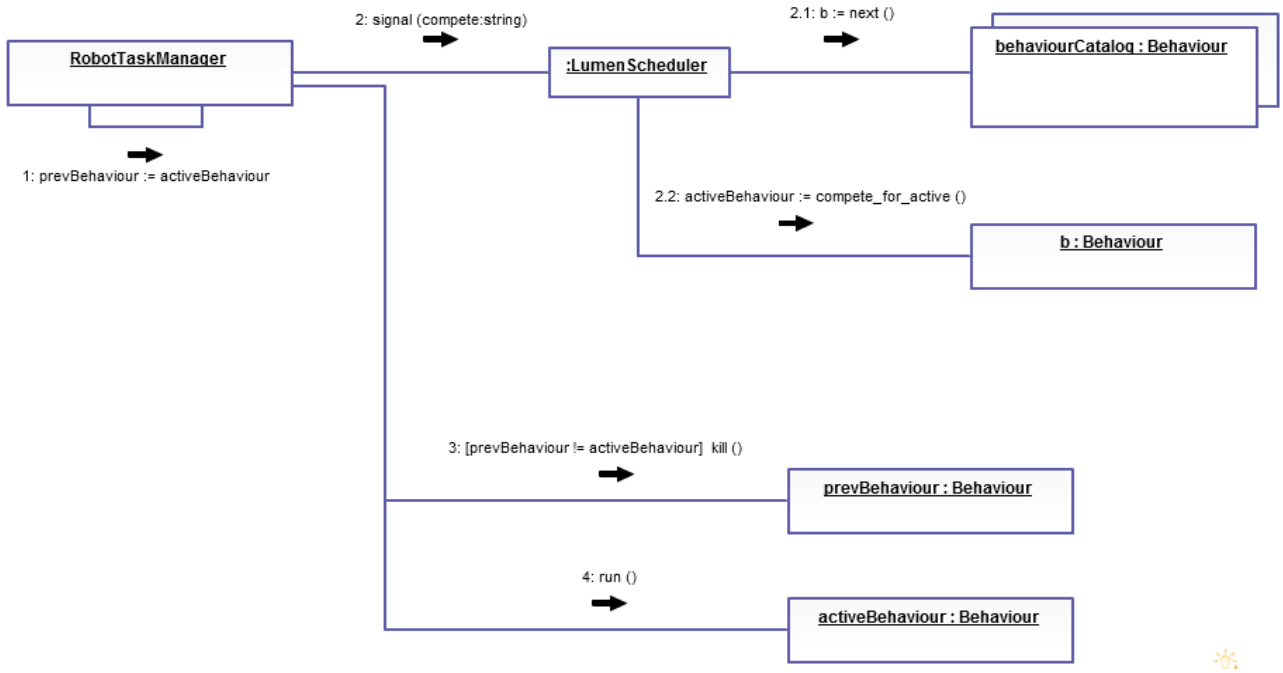


Figura 25: Diagrama de comunicación para el task switching.

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.

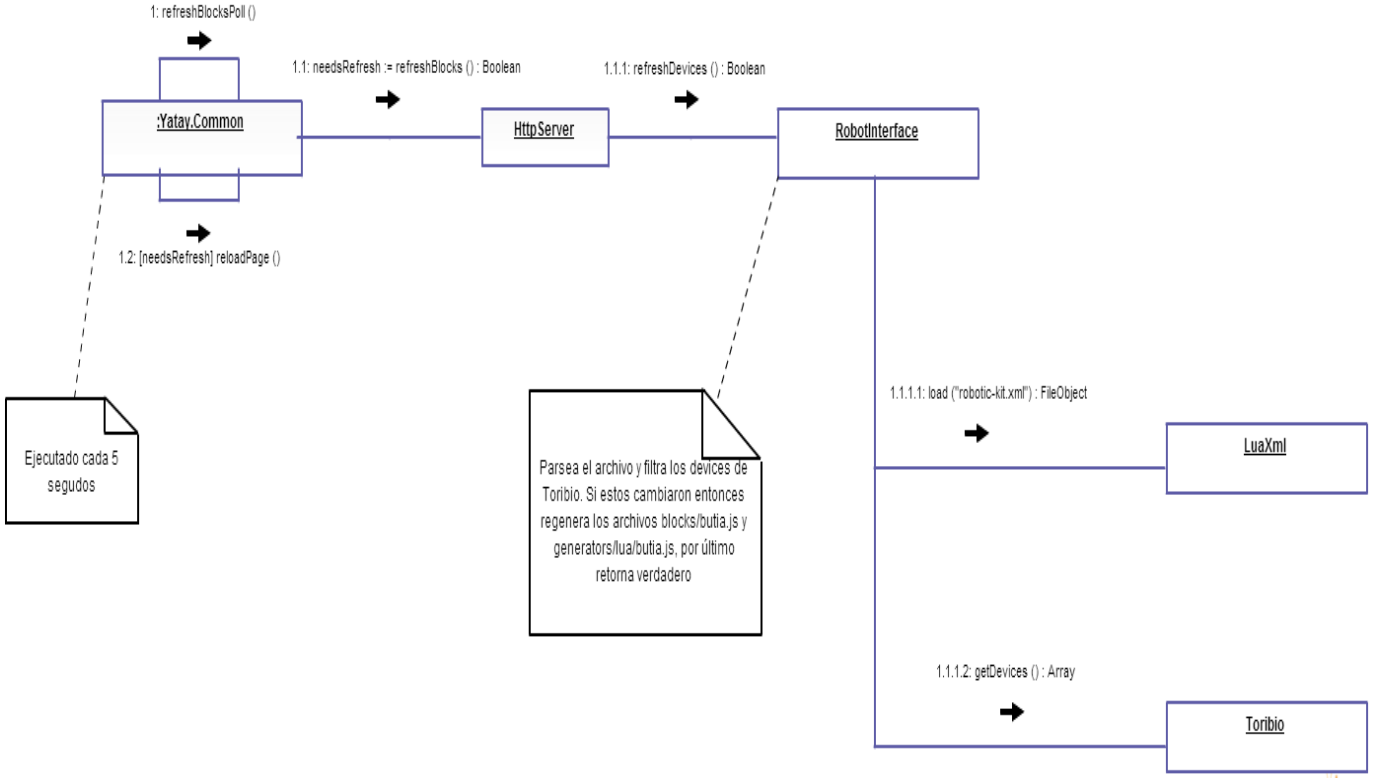


Figura 26: Diagrama de comunicación para refrescar los bloques de dispositivos conectados.

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 detallada en profundidad en la sección *Paradigmas Robóticos* del documento Estado del Arte [4].

- 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 (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 [38], 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 [20], 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 [4] 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 [29]. Por otro lado, la biblioteca YepNope funciona como un cargador condicional permitiendo cargar a demanda los Scripts o CSS requeridos [43]. 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 linealmente 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.

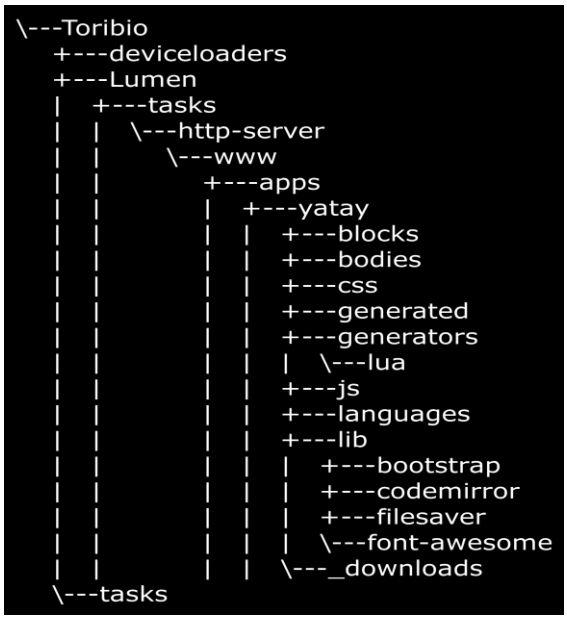


Figura 28: Árbol de directorios del framework de Yatay.

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 que 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.

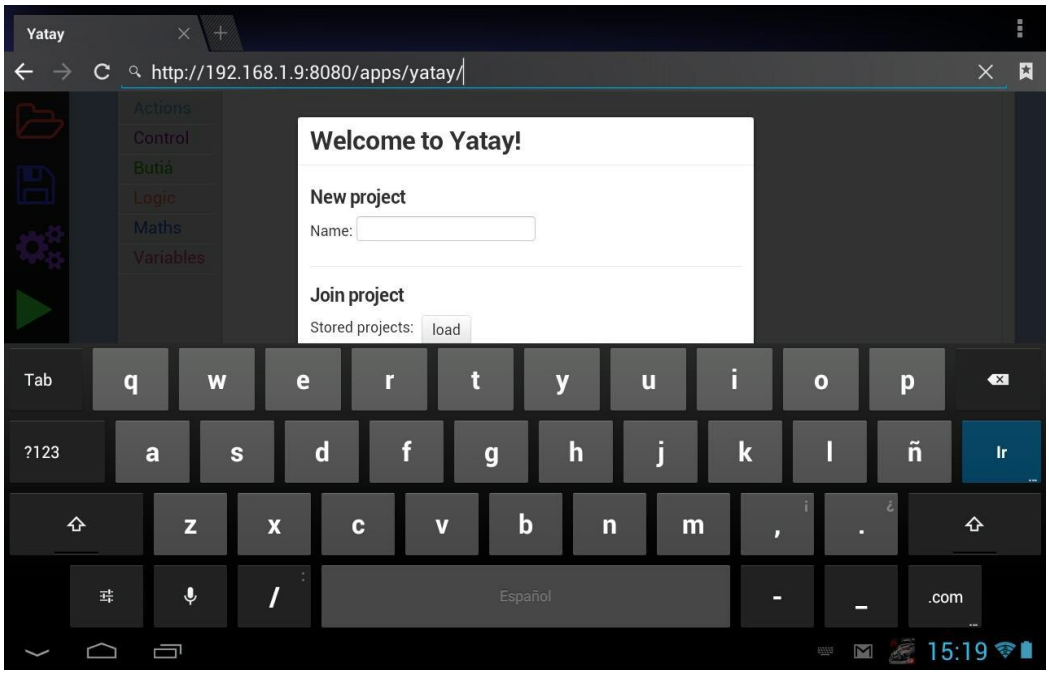


Figura 29: Pantalla inicial de Yatay en Tablet de 10’’.

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 [7].

°	Funcionalidad	Entrada	Salida esperada
.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.
.2	Ingreso (1era vez)	El usuario ingresa un nombre de proyecto y presiona "Comenzar".	El mensaje de bienvenida desaparece y el tutorial comienza.
.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.
	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.
	Visibilidad	Ingreso al sistema desde	La interfaz gráfica puede visualizarse de forma

	(Tablet)	Tablets de 10" y de 7".	correcta, siendo ésta similar a la especificada para Tablets en el documento ***"Manual de Usuario"*** [5].
	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.
.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.
.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.
.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.
.4	Creación de comportamientos	1) Se crea un comportamiento de nombre "Detener" con prioridad 3.	1) El comportamiento es creado correctamente. 2) Se enlista como comportamiento listo vaciando el área de trabajo o pizarra.

		Como disparador tiene una condición 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.	
	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.
.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.
.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.
	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.
0	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.
1	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. 3) El sistema recupera y marca como listos los comportamientos existentes en el archivo.
2	Ejecución	1) Teniendo creados los comportamientos del caso de prueba 6, se selecciona el	1) Desaparecen todos los botones quedando disponibles solo los botones de "Ejecución sin debug", "Debug" y "Detener".

		boton de ejecución (ver manual de usuario). 2) Se selecciona el boton de ejecución sin debug (ver manual de usuario).	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.
3	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.
4	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.
5.1	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.
5.2	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.
5.3	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.
6	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.
7	Funcionamiento de bloques	Se prueba la ejecución de cada uno de los bloques existentes, dentro de un	Al ejecutarse, el bloque presenta el comportamiento descrito para dicho bloque.

		contexto de bloques que permita dicha prueba.	
8	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.
9	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.
0	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.
1	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.
2	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).
3	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.
4	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.
5	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.
6	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.

7	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.
8	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 de que parte del código que se está ejecutando (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.

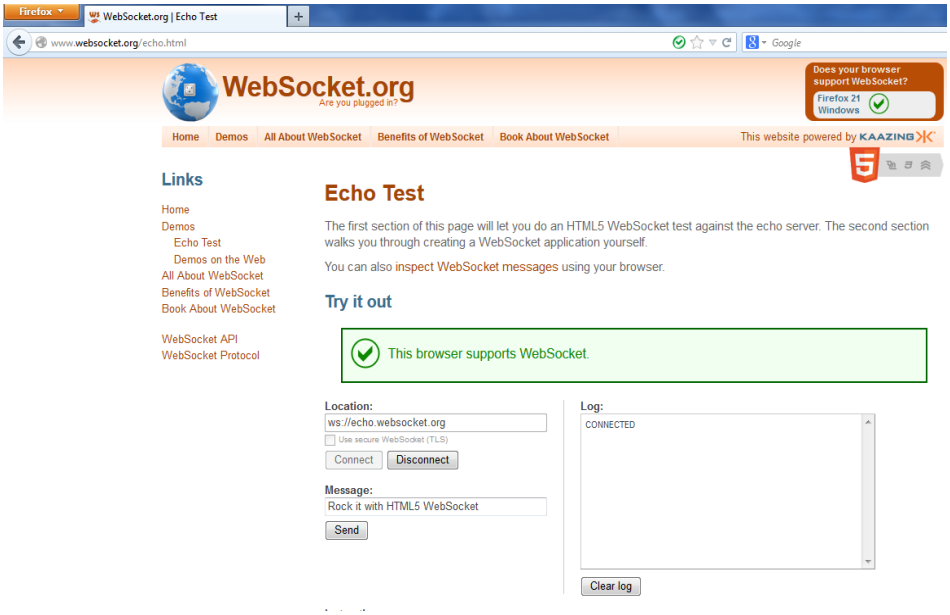


Figura 30: Echo Test con websocket.org para Firefox en un ordenador.

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 [42]). 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 [17] 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) [11]. 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 [11].

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 [12]. 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%

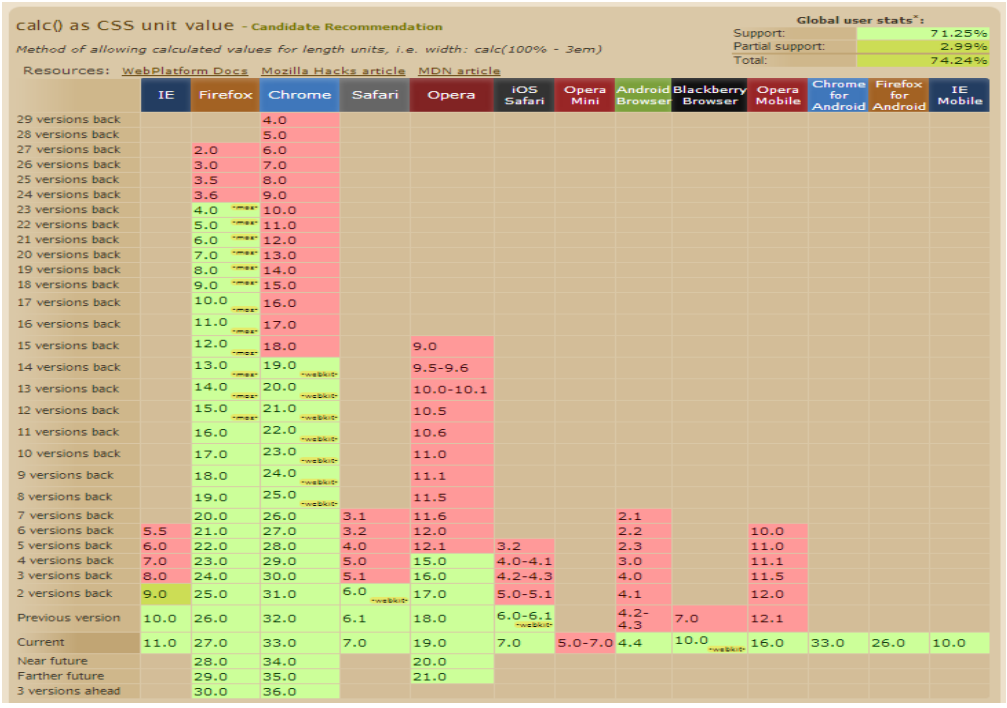
	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android	IE Mobile
29 versions back			4.0										
28 versions back			5.0										
27 versions back		2.0	6.0										
26 versions back		3.0	7.0										
25 versions back		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 [12].

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 [10]. 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 [10].

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 presionamiento 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 [14]. 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 [19] 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 necesitará 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 [30].

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 supera los requerimientos funcionales y no funcionales marcados como objetivo, 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 solo 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] Alejandro Achkar. Andrés Margalef. Estado del Arte. 2012. Disponible en Web: http://www.fing.edu.uy/~pgiderob/EstadoDelArte-1.1.pdf, último acceso julio 2013.

[2] Andres Nebel. Renzo Rozza. IDE Android para la Programación de Comportamientos Robóticos. Arquitectura y Diseño del Sistema. 2014.

[3] Andres Nebel. Renzo Rozza. IDE Android para la Programación de Comportamientos Robóticos. Especificación de Requerimientos. 2013.

[4] Andres Nebel. Renzo Rozza. IDE Android para la Programación de Comportamientos Robóticos. Estado del Arte. 2013.

[5] Andres Nebel. Renzo Rozza. IDE Android para la Programación de Comportamientos Robóticos. Manual de Usuario. 2014.

[6] Andres Nebel. Renzo Rozza. IDE Android para la Programación de Comportamientos Robóticos. Pautas para la interfaz de usuario. 2014.

[7] Andres Nebel. Renzo Rozza. IDE Android para la Programación de Comportamientos Robóticos. Plan de pruebas. 2014.

[8] Antonio Ramírez Toledo. El constructivismo pedagógico. Disponible en Web: http://ww2.educarchile.cl/UserFiles/P0001/File/El%20Constructivismo%20Pedag%C3%B3gico.pdf, último acceso julio 2013.

[9] Blockly. A visual programming editor, https://code.google.com/p/blockly/, último acceso julio 2013.

[10] Can I use calc() as CSS unit value? Compatibility table for support of calc() as CSS unit value in desktop and mobile browsers, http://caniuse.com/calc, ultimo acceso enero 2014.

[11] Can I use download attribute? Compatibility table for support of Download attribute in desktop and mobile browsers, http://caniuse.com/download, último acceso enero 2014.

[12] Can I use SVG? Compatibility table for support of SVG in desktop and mobile browsers, http://caniuse.com/SVG, último acceso enero 2014.

[13] Castro. Acuña. Propuesta comunitaria con robótica educativa: Valoración y resultados de aprendizaje. Disponible en Web: http://www.redalyc.org/pdf/2010/201024390006.pdf, último acceso abril 2014.

[14] CodeMirror. Issues. Android Chrome Backspace and Enter do not fire keydown events, https://github.com/marijnh/CodeMirror/issues/2234, último acceso enero 2014.

[15] Crómulo Gallego, Royman Pérez, Adriana P. Gallego y John F. Pascuas. Didáctica Constructivista: Aportes y Perspectivas, http://www.saber.ula.ve/bitstream/123456789/19856/2/articulo14.pdf, último acceso julio 2013.

[16] Deliberative Agents. Notes, http://cgi.cse.unsw.edu.au/~aishare/index.php, último acceso julio 2013.

[17] Eligrey, GitHub FileSaver Repository, https://github.com/eligrey/FileSaver.js/, último acceso febrero 2014.

[18] eLinux. The Raspberry Pi Wiki, http://elinux.org/RaspberryPiBoard, último acceso julio 2013.

[19] FontAwesome. The iconic font designed for Bootstrap, http://fortawesome.github.io/Font-Awesome/, último acceso enero 2014.

[20] HiddenTao, JavaScript client-side file generation and download, https://www.hiddentao.com/archives/2011/07/04/javascript-client-side-file-generation-and-download/, último acceso febrero 2014.

[21] John Maloney. Mitchel Resnick. Natalie Rusk. Brian Silverman. Evelyn Eastmond. The Scratch Programming Language and Environment. MIT. 2010. Disponible en Web: http://web.media.mit.edu/ScratchLangAndEnvironment.pdf, último acceso julio 2013.

[22] José María Cañas Plaza. Jerarquía dinámica de esquemas para la generación de comportamiento autónomo. 2003. Disponible en Web: http://oa.upm.es/345/1/JOSE_MARIA_CANAS_PLAZA.pdf, último acceso julio 2013.

[23] Jorge Visca. Lumen, https://github.com/xopxe/Lumen, último acceso julio 2013.

[24] Jorge Visca. Toribio, https://github.com/xopxe/Toribio, último acceso julio 2013.

[25] Juan Pablo Casares Charles. Ambiente para la instrucción visual de algoritmos. 1999. Disponible en Web: http://usablehack.com/amiva/AMIVA.pdf, último acceso julio 2013.

[26] LEJOS. Java for LEGO Mindstorms. Behavior programming, http://www.lejos.org/nxt/nxj/tutorial/Behaviors/BehaviorProgramming.htm, último acceso julio 2013.

[27] Margaret M. Burnett. Visual Programming. 1999. Disponible en Web: http://www.cs.auckland.ac.nz, último acceso julio 2013.

[28] María Luisa Pinto Salamanca. Nelson Barrera Lombana. Wilson Javier Pérez Holguín. Uso de la robótica educativa como herramienta en los procesos de enseñanza. 2010. Disponible en Web: http://virtual.uptc.edu.co/revistas2013f/index.php/ingenieria_sogamoso/article/download/912/912, último acceso marzo 2014.

[29] Modernizr. Documentation, http://modernizr.com/docs/, último acceso julio 2013.

[30] Open Source Initiative. MIT License, http://opensource.org/licenses/mit-license.html, último acceso febrero 2014.

[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°, http://www.presidencia.gub.uy/comunicacion/comunicacionnoticias/brechner-plan-ceibal-Tablets-alumnos-primer-ano, último acceso marzo 2014

[32] Processes and Threads. Android for Developers. Official Website, http://developer.android.com/processes-and-threads.html, último acceso julio 2013.

[33] Rob Gravelle. Comet Programming: Using Ajax to Simulate Server Push. http://www.webreference.com/programming/javascript/rg28/index.html, último acceso julio 2013.

[34] Seymour Papert. FactoriaHistorica, http://factoriahistorica.wordpress.com/2013/11/19/seymour-papert/, último acceso julio 2013.

[35] SK Planet. Python vs Lua, http://www.slideshare.net/cybrshin/lua-vs-python, último acceso julio 2013.

[36] Sugar Labs. Wiki Sugar Labs. Turtle Art, http://wiki.sugarlabs.org/go/Activities/TurtleArt, último acceso julio 2013.

[37] The Computer Language Benchmarks Game, http://benchmarksgame.alioth.debian.org/u64/benchmark.php?test=all&lang=lua&lang2=python3&data=u64, último acceso julio 2013.

[38] Tiago Donizio. Doug Currie. LuaSQLite3. Documentation, https://github.com/marijnh/CodeMirror/issues/2234, último acceso enero 2014.

[39] TortuBots. Wiki Proyecto Butiá, http://www.fing.edu.uy/inco/proyectos/Butiá/TortuBots, último acceso julio 2013.

[40] USB4All. Wiki Proyecto Butiá, http://www.fing.edu.uy/inco/proyectos/Butiá/Usb4all, último acceso julio 2013.

[41] Vavassori, Vahldick, Urban, Krueger, Halma. Experimentação com Robótica Educativa no Ensino Médio: ambiente, atividades e resultados. Disponible en Web: http://robolab.inf.furb.br/robolab/artigos/robolab/wie2009.pdf, último acceso abril 2014.

[42] WebSocket. Wikipedia, http://en.wikipedia.org/wiki/WebSocket, último acceso julio 2013.

[43] Yepnope. Home page, http://yepnopejs.com/, último acceso julio 2013.

Apéndice A

Glosario

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

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 funcionan como ruedas en el robot Butiá.

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.

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.