Estado del Arte

IDE Android para la Programación de Comportamientos Robóticos

Andrés Nebel - Renzo Rozza

Tutores
Andres Aguirre, Gonzalo Tejera

Cotutores
Rafael Sisto

30 de Julio del 2013

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

Índice General

Índice de Figuras

Introducción

Motivación y objetivo del proyecto

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

Desde el año 2009 el grupo MINA del INCO trabaja en el desarrollo de una plataforma robótica educativa llamada Robot Butiá. Este robot está diseñado como una plataforma genérica y abierta, que soporta distintas arquitecturas. Actualmente se dispone de un despliegue de 50 robots Butiá ubicados en el territorio nacional, principalmente en liceos, existiendo de todas formas algunas escuelas donde también se trabaja con la plataforma.

En el contexto de la robótica móvil autónoma, existen un conjunto de arquitecturas de control pertenecientes al paradigma reactivo que se distinguen por ser bioinspiradas. Éstas intentan reflejar la conducta animal mediante módulos que implementan comportamientos sencillos (sensar y actuar) o la combinación de ellos para lograr comportamientos más complejos. Es por esto que dos de las grandes cualidades que presentan estas arquitecturas son la escalabilidad y la reutilización de módulos implementados.

Dentro de este contexto, durante el 2011 y 2012 un grupo de la asignatura proyecto de grado desarrolló un IDE, llamado eButiá, basado en el lenguaje de programación visual Etoys. Dicho IDE utiliza la arquitectura reactiva Subsumption para permitir el desarrollo de comportamientos. Para este proyecto de grado se propone la investigación e implementación de un IDE para el sistema operativo Android que permita el desarrollo de comportamientos para el robot Butiá. Dicha investigación tendrá como objetivo buscar la mejor solución

para lograr esto, analizando la adaptación de alguno de los IDEs existentes como TortuBots o eButiá, así como también la implementación entera de un nuevo IDE que logre objetivos similares a los anteriormente creados para computadoras.

Paradigmas Robóticos

Arquitectura y Paradigmas de Sistemas Robóticos

Llamamos arquitectura de un robot autónomo a *la organización de sus capacidades sensoriales, de procesamiento y de acción para conseguir un repertorio de comportamientos inteligentes interactuando con un cierto entorno* [1]. 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 resolver el problema cuando múltiples comportamientos se activan al mismo tiempo.

Los robots son sistemas con alto grado de complejidad; requiriendo un completo repertorio de comportamientos, una organización interna clara, accesible y fácilmente modificable. 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. Entonces, hablamos de arquitectura cuando incluimos la tarea de construir o seleccionar esas señales de entrada a partir de una cantidad desbordante de información no específica que sensores ofrecen. También cuando la naturaleza de los objetivos puede variar enormemente o cuando el robot tiene varios objetivos, 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.

Paradigma Reactivo

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

Figura 1: Rodney Brooks, profesor del MIT.

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

la representación simbólica, ni el razonamiento sobre símbolos para generar comportamiento.

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

Se trata de enfocarse más en los razonamientos simples que comparte todo el reino animal en vez de particularmente los de alto nivel del ser humano. Tiene como beneficio su sencillez, pero como contrapartida una incapacidad de realizar tareas con un nivel de complejidad alta, tales como por ejemplo efectuar cálculos óptimos para toma de decisiones. Mostramos a continuación un esquema a modo de ejemplo de una posible arquitectura reactiva en comparación a una deliberativa [2]:

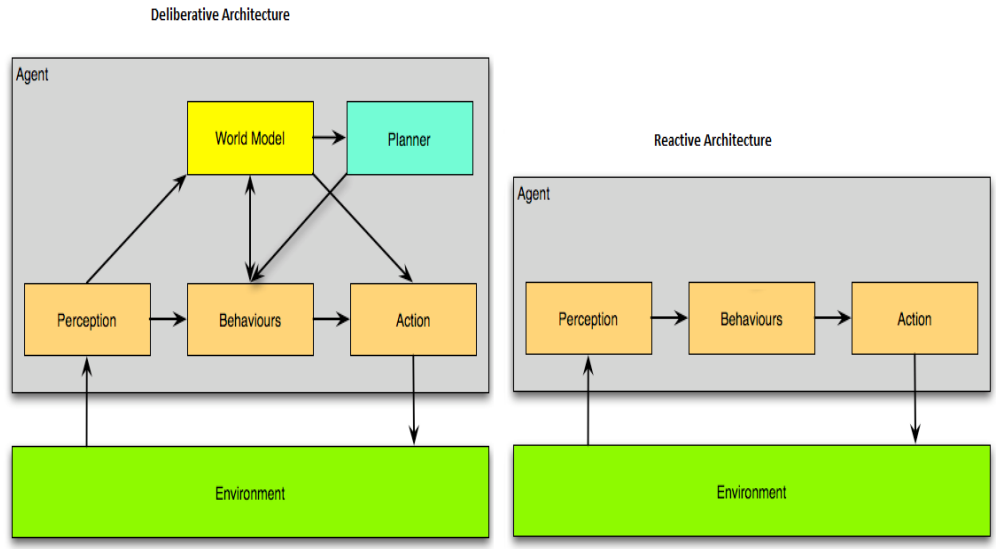

Figura 2: Comparación de dos ejemplos de arquitecturas, una deliberativa y otra reactiva.

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

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

Arquitectura Subsumption

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

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

Los módulos de capas superiores pueden alterar la entrada o salida de las tareas de capas inferiores, lo cual les da cierto control de manipulación sobre estas. Subsumption describe dos formas de realizar lo anterior: inhibir, el cual apagar 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.

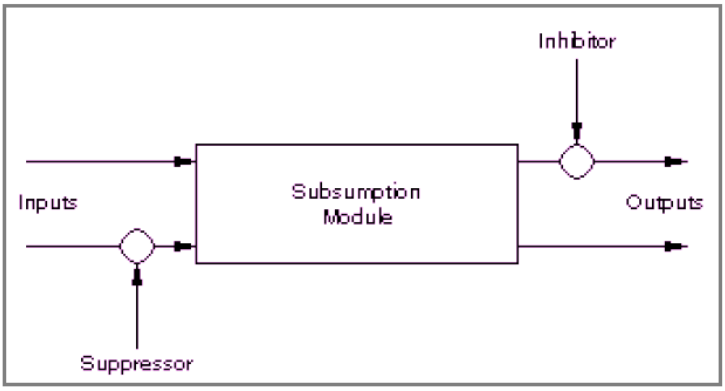

Figura 3: Módulo de subsumption con inhibir y suprimir.

Arquitectura basada en prioridades

Detallaremos aquí otro tipo de arquitectura del paradigma reactivo diferente a subsumption, y la cual 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. 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.

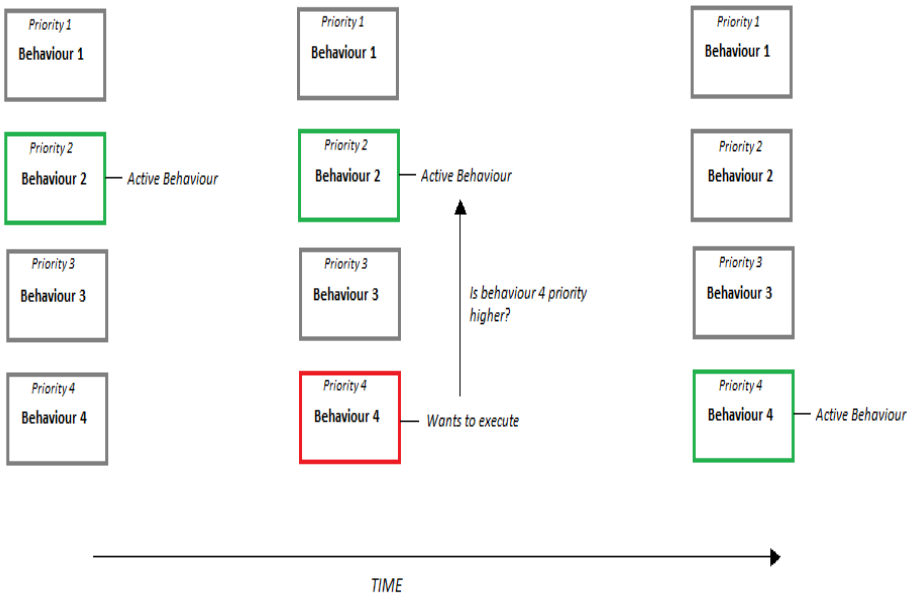

Figura 4: Activación y ejecución de una tarea de mayor prioridad que la tarea activa.

Esta arquitectura elimina el sistema a de inhibir y subsumir que se describió en la arquitectura Subsumption. Además se enfoca en la importancia de las tareas en vez de hacerlo en el nivel de abstracción de la tarea. En nuestra opinión dichas modificaciones producen una arquitectura un poco más intuitiva desde el punto de vista de la enseñanza, lo cual es interesante para los objetivos del proyecto.

Enseñanza de la Robótica

Constructivismo e influencias en la enseñanza robótica

El constructivismo es una corriente pedagógica creada por Ernst von Glasersfeld y que tuvo como gran referente a Jean Piaget, que se basa en la teoría del conocimiento constructivista, que postula la necesidad de entregar al alumno herramientas que le permitan crear sus propios procedimientos para resolver una situación problemática, lo cual implica que sus ideas se modifiquen y siga aprendiendo.

Para esta corriente, el conocimiento de un alumno no es una copia fiel de la realidad, sino una reconstrucción del individuo. El constructivismo educativo propone un paradigma en donde el proceso de enseñanza se percibe y se lleva a cabo como un proceso dinámico, participativo e interactivo del sujeto, de modo que el conocimiento sea una auténtica construcción operada por la persona que aprende. El constructivismo en pedagogía se aplica como concepto didáctico en la enseñanza orientada a la acción [54].

Seymour Papert es un pionero de la inteligencia artificial, científico informático, matemático y educador. Fue uno de los más notables discípulos de Piaget, quien llegó a mencionar en una ocasión que nadie comprendía mejor sus ideas que Papert [56]. 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, siendo pionero también en este ámbito.

Figura 5: Robot tortuga de Papert.

La idea de Logo probó ser una forma efectiva de lograr que los estudiantes comprendan los conceptos de programación, además de que es una herramienta para acercar a las matemáticas a aquellos que no se sienten motivados por ella, ya que con la tortuga se realizan cálculos para definir el avance y los movimientos. También se destaca la sencillez para aquellas personas con capacidades diferentes. En base a esta filosofía llegó a desarrollar un plan de estudios que contenía dentro del programa grandes áreas como matemáticas, lenguaje y ciencia [55].

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

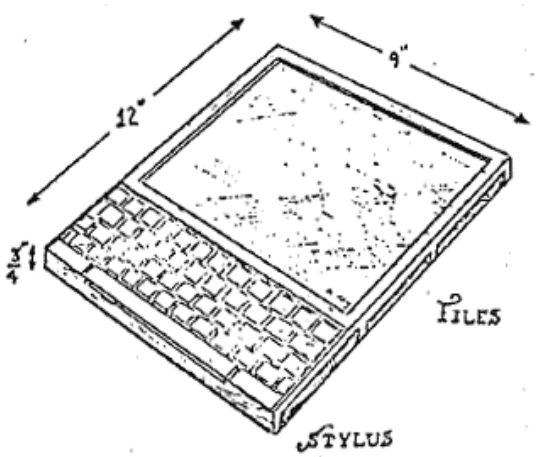

Figura 6: Ilustración de la Dynabook como fue descrita por Kay en su paper.

Lenguajes de Programación Visual

En los lenguajes de programación visual (LPV) más de una dimensión se utiliza para transmitir la semántica, donde tales dimensiones adicionales son adquiridas con el uso de objetos (un ejemplo son los bloques) y donde cada uno de estos objetos potencialmente significativos son símbolos, al igual que en los lenguajes de programación tradicionales cada palabra es un símbolo [14].

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

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

Papert, quien propuso desarrollar una nueva forma de ver el uso de la computadora para la educación [15].

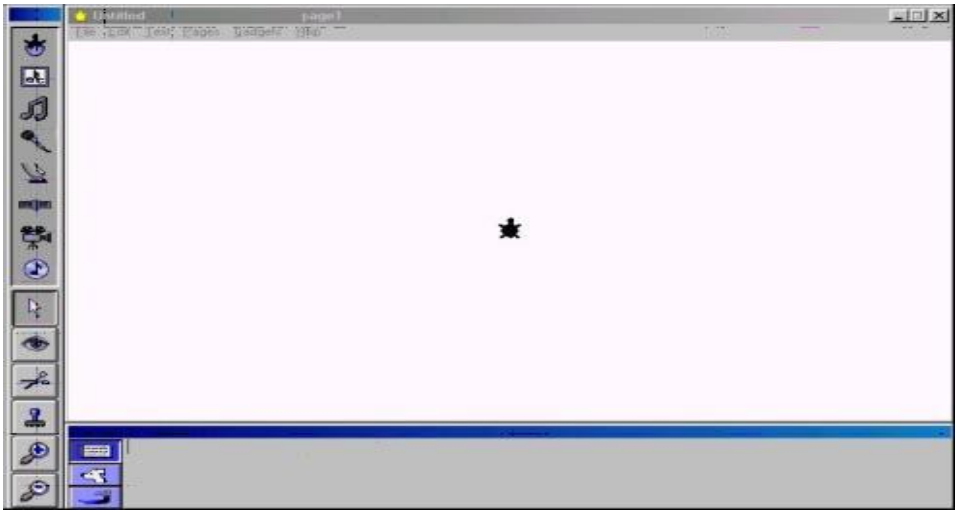

Figura 7: Interfaz gráfica del sistema Logo.

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

Así, el estudiante aprende a definir un problema y definir los pasos necesarios para resolverlo, lo cual pone al descubierto la presencia fuerte de los principios del constructivismo de Jean Piaget [47] aplicados a la enseñanza de informática y robótica. Fundamentalmente los niños enfrentan retos intelectuales que pueden ser resueltos mediante el desarrollo de programas en Logo. El proceso de revisión manual de los errores contribuye a que los niños desarrollen habilidades metacognitivas al poner en práctica el concepto de autocorrección. Esto se ve fuertemente beneficiado con el uso de *gráficos tortuga* [46], es decir, poder dar instrucciones a una tortuga virtual (ver Figura 7) o cursor usado para crear dibujos mediante palabras que representan instrucciones.

Citamos la siguiente frase de Papert que resulta apropiada [16]: “Al enseñarle a pensar al ordenador, los chicos se embarcan en una exploración del modo en

que ellos mismos piensan". Actualmente, existen variados LPVs desarrollados con fines educativos los cuales persiguen los objetivos planteados para el desarrollo de este IDE, abriendo posibilidades para lograr una interfaz gráfica con un LPV basado en bloques de tal manera que conserve y posea las ventajas pedagógicas que plantea dicho paradigma.

Experiencias anteriores

Experiencias regionales e internacionales

En el contexto de la robótica educativa se intenta profundizar en los conceptos de la física, matemática, dibujo y programación, generando una construcción propia del conocimiento por parte de los alumnos que integran las distintas disciplinas en una síntesis particular para el desarrollo de los proyectos. Así se pone al alcance de los alumnos las herramientas necesarias para que desarrollen dispositivos físicos, externos a la computadora, que serán controlados por ésta [34].

Desde 1975, en la Universidad Du Maine, en Le Mans, Francia, aparece una primera utilización con fines educativos de la robótica, con el desarrollo de una sistema de control automatizado para la administración de experiencias en el campo de la psicología. De estas investigaciones surge el concepto de encargado-robot, siendo el mismo un sistema que tiene por objetivo hacer trivial la percepción de experiencias de laboratorio en el dominio de la psicología experimental. A través de este sistema, el alumno puede plantearse múltiples preguntas sobre el campo de estudio, establecer estrategias para responder a cada una de las preguntas, experimentar e interpretar los resultados visualizados en la pantalla del computador. Luego en 1990, Martial Vivet del Laboratorio de Informática de la misma universidad, define la *microrobótica pedagógica* como "una actividad de concepción, creación, puesta en práctica, con fines pedagógicos, de objetivos técnicos físicos que son reducciones bastante fiables y significativas de procedimientos y herramientas realmente utilizadas en la vida cotidiana, particularmente en el medio industrial". Es a partir de estas definiciones, que se han realizado muchas investigaciones y trabajos que pretenden contribuir al desarrollo de un marco teórico y conceptual en la educación para la robótica pedagógica, así como para la construcción de entornos de aprendizaje en distintos medios y niveles (págs. 112, 113 del libro "La educatrónica: innovación en el aprendizaje de las ciencias y la tecnología" [41]).

Numerosas investigaciones ya demuestran el interés global por la inserción de herramientas robóticas en las aulas de clase. Siendo implementadas diversas experiencias robóticas cuyos énfasis son muy variados, como por ejemplo: en

1998 se inició el proyecto “Robótica y Aprendizaje por Diseño”, realizado conjuntamente por el Centro de Innovación Educativa de la Fundación Omar Dengo y el Ministerio de Educación Pública de Costa Rica. Se pretende concretar ciertos desempeños y habilidades relacionadas al diseño tecnológico, brindando cursos y capacitación tanto a alumnos como docentes de escuelas públicas.

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

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

A partir del nuevo milenio se expande fuertemente la robótica educativa impulsada principalmente por el proyecto mundial OLPC, impulsado por una organización sin ánimos de lucro creada por catedráticos del Laboratorio de Multimedia del MIT para diseñar, fabricar y distribuir una laptop por niño. Y a

raíz de estas experiencias, hoy en día existe una cantidad creciente de proyectos relacionados a la robótica educativa en Latinoamérica, incluyendo países como México, Argentina, Chile, Venezuela.

Ámbito Local

En el 2007 en el ámbito local, a partir del Plan Ceibal creado *"con el fin de realizar estudios, evaluaciones y acciones, necesarios para proporcionar un computador portátil a cada niño en edad escolar y a cada maestro de la escuela pública, así como también capacitar a los docentes en el uso de dicha herramienta, y promover la elaboración de propuestas educativas acordes con las mismas"* [36] y con el complemento en 2009 del proyecto Butiá (detallado posteriormente) que intenta ampliar las capacidades sensoriales y de actuación de la computadora portátil del Plan Ceibal, transformándola en una plataforma robótica móvil [3], se tienen los primeros indicios de robótica educativa a nivel masivo. Además en el 2010, con la aparición del plugin Butiá para TortuBots comienza a manejarse el concepto de un IDE para el control del robot Butiá. TortuBots es un programa inspirado por Logo que persigue sus mismos fines pero posee interfaces gráficas modernas, continuando los lineamientos de Sugar [7]. Con este complemento se permite a los alumnos programar desde dicho entorno de desarrollo el comportamiento del robot Butiá.

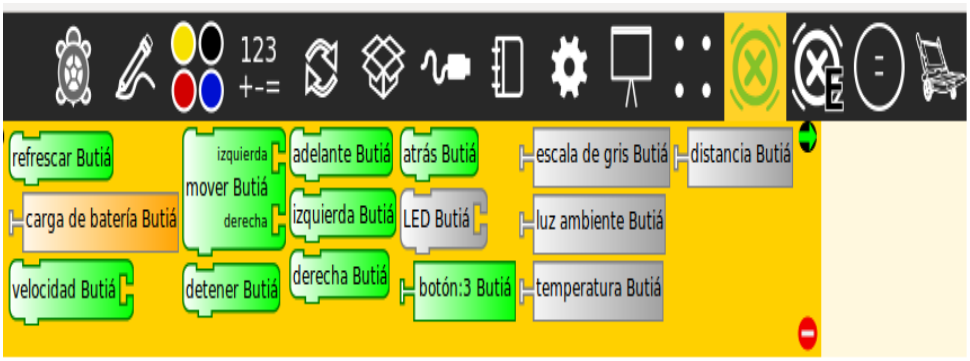

Figura 9: Interfaz gráfica del sistema TortuBots (plugin Butiá).

Vinculando concretamente las experiencias anteriores con los fines de este proyecto; desarrollar un IDE para el sistema operativo Android que pueda manejar comportamientos del robot Butiá, se debe detallar el proyecto de grado introducido al comienzo de este documento. Este fue desarrollado entre el 2011 y 2012, sobre la versión de escritorio de Sugar con el objetivo de ser ejecutado

en una computadora portátil XO. Este entorno de desarrollo de programación, orientado por arquitecturas robóticas basadas en comportamientos, pretende actuar como herramienta para crear programas que definan la forma de actuar del robot Butiá siguiendo una determinada estructura [37]. Este proyecto intenta mejorar las prestaciones de TortuBots, quien presenta como limitaciones la baja visibilidad global del código en la aplicaciones de mediano o gran porte y además no está orientado a programación de comportamientos robóticos. Proponiendo como solución, un IDE orientado a comportamientos usando como base el lenguaje de programación visual Etoys, estructurado por una arquitectura Subsumption, y una comunicación con Bobot (detallado en la próxima sección) por medio de sockets TCP/IP.

Por otro lado, y entrando en aspectos más específicos, es importante mencionar la experimentación realizada en el 2013 por los creadores del robot Butiá en la cual se agregó una SBC Raspberry Pi al kit original de este robot, donde fue configurado Sugar para poder ejecutar TortuBots. Luego, desde una tablet por medio de SVC [40] se logró ejecutar el plugin Butiá ofrecido por TortuBots y así controlar el robot Butiá desde una tablet [41]. Sin embargo, dado los problemas de performance que implica el graphical desktop sharing, se percibió que la calidad de la experiencia fue muy distinta que la original desde un ordenador de escritorio. Por un lado la velocidad de respuesta no es la misma. Por el otro, la calidad de la imagen renderizada es usualmente peor que la original y tiene como problema final que tanto la interfaz gráfica como el software en general están pensados para su uso en un ordenador y no en un dispositivo móvil, dejando de lado un montón de aspectos conocidos que mejoran potencialmente la experiencia en un dispositivo móvil cuyas características de manejo y de tamaño de pantalla, distan mucho de las de un ordenador de escritorio. Estas cuestiones son uno más de los motivos que impulsan el desarrollo de este proyecto.

Proyecto Butiá y Plataforma Robótica

Acerca del Proyecto

El proyecto Butiá fue lanzado en el año 2009 como complemento al plan Ceibal 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. Actualmente la implementación 1.0 del proyecto está siendo utilizado en formato de kit, distribuido a 28 liceos públicos del Uruguay, con un set de sensores y piezas que permiten cambiar la ubicación de los sensores en la plataforma móvil donde se coloca la computadora XO [3].

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

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

Hardware

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

Si bien posee soporte para una variedad de controladoras distintas, en general se utilizó la Arduino Mega en las primeras versiones del proyecto, y actualmente la USB4Butiá, adaptación para el Butiá de la placa USB4All [4]. Esta placa posee 6 puertos RJ45 (Ethernet) con soporte Plug&Play y Hotplug y un bus para motores AX-12.

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

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

Software

Dado que Butiá no maneja una SBC esta no posee un sistema operativo. El único software que forma parte íntegra de Butiá es el firmware de la placa USB4Butiá. Este se encarga de resolver la interacción con los sensores y

actuadores, exponiendo una interfaz que permite controlarlos de manera sencilla a través del uso de un protocolo llamado User Protocol.

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

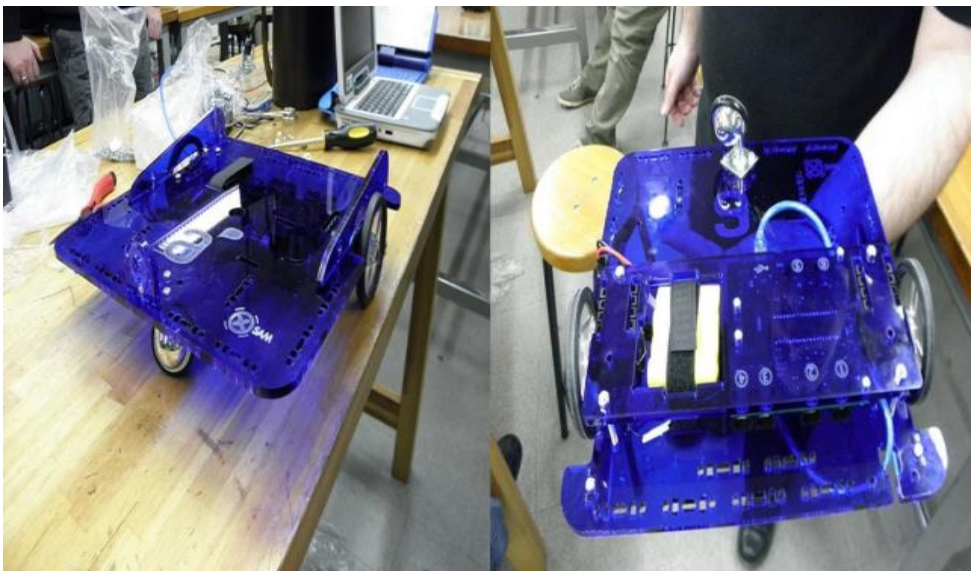

Figura 11: Robot Butiá 2.0 parte superior (izquierda) y parte inferior (derecha).

Bobot

Para utilizar el Butiá desde un computador, se creó una aplicación demonio, Bobot Server [33], que recibe funciones en formato de texto plano a través del puerto 2009, o a través de una interfaz web. Este es el encargado de realizar la comunicación a bajo nivel con la placa USB4Butiá y de exponer una interfaz a través de una conexión TCP/IP. A través de ella se pueden listar las funcionalidades ofrecidas por la interfaz e invocar cada una de ellas. De esta forma se independiza a las aplicaciones que utilizan la placa, de la implementación de la misma.

Los programas que controlan el robot pueden estar entonces implementados en cualquier lenguaje y plataforma de desarrollo capaz de interactuar a través de

una conexión TCP/IP. A continuación se describe los comandos proporcionados por Bobot para interactuar con la USB4Butiá:

1. LIST
    - Lista los módulos actualmente disponibles en la placa USB4Butiá.
2. DESCRIBE *moduleName*
    - Describe las operaciones implementadas por un componente específico.
3. OPEN *moduleName*
    - Abre un cierto componente para ejecutar sus operaciones.
4. CALL *moduleName operation param1, param2, ...,paramN*
    - Llama a una operación de un módulo pasándole una serie de parámetros.
5. CLOSEALL
    - Cierra la conexión sin devolver ningún parámetro.

Complementos

Para complementar las funcionalidades del Robot Butiá últimamente se maneja la posibilidad de agregar a su kit una Raspberry Pi [10], o en primera instancia hacer uso de esta placa de forma experimental para proyectos como este. Siendo necesario introducir esta SBC desarrollada por Raspberry Pi Foundation, organización sin ánimos de lucro iniciada en el año 2008, la cual da origen a la Raspberry Pi como ordenador de bajo coste para facilitar la enseñanza de la informática en los centros educativos.

En el 2012 comenzó a fabricarse con colaboraciones del laboratorio de informática de la Universidad de Cambridge y de Boardcom, siendo diseñada con fin de ser lo más barato posible y llegar a la mayor cantidad de usuarios. Desde el punto de vista del Hardware la placa Raspberry Pi cuenta con unas dimensiones de 8.5 por 3.5 cm, donde se ubica un chip integrado Boardcom BCM2835, que contiene un procesador ARM11 con varias frecuencias de funcionamiento y la posibilidad de realizar overclocking hasta 1 GHz sin perder la garantía, un procesador gráfico VideoCore IV capaz de obtener vídeo a 1080p

y audio de alta calidad a través de su conector HDMI, y memoria RAM entre 256 y 512 MB. La misma puede ser dotada con una sistema operativo y programas por medio de una tarjeta SD con por lo menos 1 GB y clase 4 (velocidad de lectura/escritura) o mejor. Posee una conexión Ethernet 10/100 y permite conexión Wi-Fi por medio de una interfaz USB Wi-Fi gracias a dos puertos USB incluidos.

Contando con esta placa en el kit del Robot Butiá se abre el abanico de posibilidades para realizar una infinidad de cosas; 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.

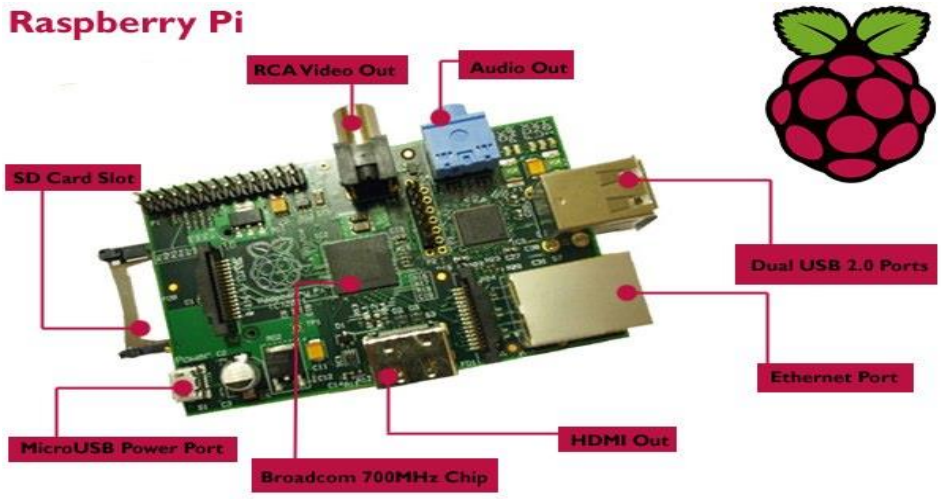

Figura 12: Placa Raspberry Pi [19].

Aspectos básicos de arquitectura

Introducción

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

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

Arquitecturas

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

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

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

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

HTML5 y librerías útiles

Ventajas de HTML5

La iniciativa de realizar el IDE para el Butiá en la Web surge de los beneficios que este sistema engloba: potencia la portabilidad en cualquier ordenador, tablet o dispositivo móvil, otorga independencia del sistema operativo, disminuye la cantidad de procesamiento que debe hacer el dispositivo, etc. En contrapartida limita a desarrollar el IDE en un lenguaje soportado por los navegadores web y es aquí donde entra en juego HTML5, tecnología aún emergente que ha sido prevista como estándar para el desarrollo web a partir del 2014 según la W3C (organismo internacional encargado de la creación de estándares para la web).

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

Si bien no obviamos la importancia de conceptos interesantes que introduce HTML5 como nuevos tags y corrección de errores, inclusión del DOM en el estándar (antes la estructuración del DOM era a criterio de los browsers), o contenido multimedia, vale la pena resaltar como muy interesante alguna de las nuevas APIs que el mismo incorpora. Donde se destaca la interfaz para WebSockets, los cuales habilitan la comunicación bidireccional en 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.

Librerías

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

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

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

Este código es sencillo, YepNope provee las propiedades “test” para testear una condición booleana, “yep” específica lo que será cargado en caso que la condición se cumpla, y “nope” lo que será cargado en caso negativo. Además, existen más propiedades opcionales como “callback” para realizar acciones luego del cargado, entre otras. Modernizr pretende poner fin a la práctica de UA sniffing presentando múltiples tests (más de 40) para características de navegadores de última generación creando un objeto JavaScript “Modernizr” que contiene los resultados booleanos de estas pruebas [28].

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

Análisis y estudio de soluciones viables

Identificación de problemas

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

Comunicación y arquitectura

Dentro del campo de la robótica es frecuente la utilización de interfaces de control (llamados también controladoras) las cuales reúnen todos los sistemas de conversión y acondicionamiento que necesita un ordenador para actuar como "cerebro" del robot [5]. En el marco del proyecto Butiá esta tarea es realizada por la placa USB4Butiá [6], la cual concentra y controla actuadores y sensores del robot. Para lograr interactuar con actuadores y sensores (y por ende el robot), es un factor determinante a resolver el medio para lograr la comunicación entre la controladora USB4Butiá y los dispositivos Android a los cuales apunta este proyecto.

Por lo tanto identificamos como uno de los problemas más grandes a resolver, la comunicación entre los dispositivos Android y la interfaz de control. Notemos que este problema implica también resolver la arquitectura de nuestro sistema, a

nivel tanto de hardware como de software. Lo primero se desprende del hecho de que para lograr la comunicación entre USB4Butiá y el dispositivo Android podrían o no haber dispositivos intermediarios que resuelvan ese problema en particular, así como también nuevos elementos de Hardware que expandan el kit que actualmente posee el robot Butiá. Lo segundo se deduce de que el software que permita generar comportamientos desde un dispositivo Android podría estar autocontenido en el dispositivo o estar distribuido entre este y otro elemento intermediario de Hardware (por ejemplo un placa SBC).

Interfaz gráfica e interacción con el usuario

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

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

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

Concurrencia y orientación a comportamientos

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

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

Estudio de comunicación con USB4Butiá

Introducción

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

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

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

Conexión USB y USB Hosting

El Universal Serial Bus (USB) es un estándar industrial desarrollado en los años ‘90 que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer alimentación eléctrica entre ordenadores, periféricos y dispositivos electrónicos [13]. Este dispone de cuatro líneas, una para datos, una para corriente y otra de toma de tierra [8].

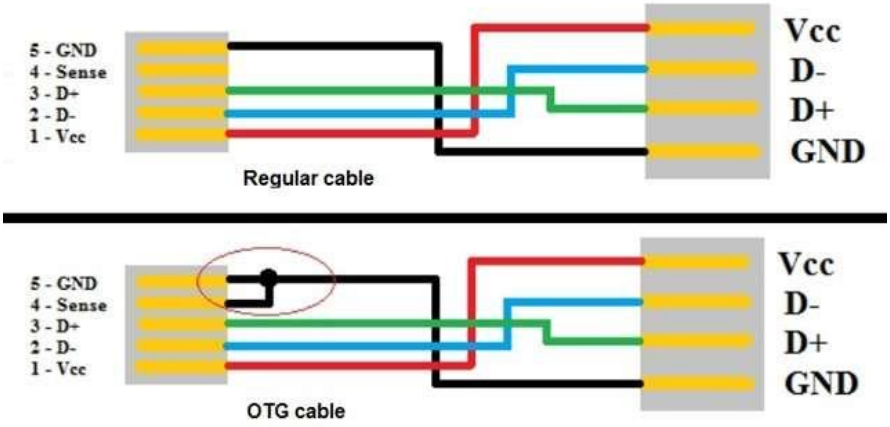

Figura 13: Cable USB OTG y USB regular.

El USB On-The-Go conocido también por USB OTG y como USB Host, 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 [9]. Los periféricos compatibles OTG disponen de un conector de tipo mini AB (o micro AB), es decir, capaz de aceptar un conector A (maestro) o B (esclavo). Gracias a que no es obligatorio que ambos dispositivos sean compatibles con OTG para comunicarse entre ellos, podemos utilizar este medio para conectar punto a punto el robot con el dispositivo Android, siendo este último quien actúa como maestro en la conexión.

Cuando un dispositivo Android se encuentra en modo USB Host, actúa precisamente como el host o master (maestro) de la conexión, alimentando el Bus e identificando a los dispositivos que estén conectados a él. La idea principal de esto es entonces que mediante USB Hosting podamos conectar dispositivos USB a dispositivos Android y lograr que estos actúen como maestros de la conexión así como alimentar la placa. Esto sera de nuestro interés

ya que nos permitirá hablar con la USB4Butiá de manera directa usando User Protocol a través de USB.

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

Soporte de Android a USB Host

No todas las versiones del sistema operativo Android soportan el modo de USB Host. Este es una funcionalidad nueva de Android agregada a partir de la versión 3.1 (Honeycomb) [17]. Este feature permite conectar dispositivos USB al dispositivo Android y que este lo reconozca. El sistema operativo ya incorpora ciertas funcionalidades básicas que permite utilizar los dispositivos más comunes de USB como ser los de almacenamiento de datos. Sin embargo lo anteriormente mencionado no es suficiente para los objetivos perseguidos, ya que una cosa es que Android de soporte para USB Host y que tenga funcionalidades básicas para dispositivos comunes, y otra muy distinta es que aplicaciones desarrolladas para Android y no incluidas en su Kernel (llamadas usualmente 3rd party Apps) tengan acceso a comunicarse con el dispositivo USB que estamos hosteando.

Para realmente poder lograr que una aplicación desarrollada para Android pueda comunicarse con el dispositivo USB conectado se debe utilizar la API de USB Host incluida en el package **android.hardware.usb** del Android SDK. Esta API nos otorga funcionalidades básicas para lograr la comunicación con el dispositivo USB y está disponible a partir de la API level 12 que coincide con la versiones de Android 3.1 en adelante [17].

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

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

Prototipo de prueba de disponibilidad de la API USB Host

Para probar lo anterior se desarrolló un prototipo en java usando Android SDK que listara los dispositivos conectados al cable USB OTG, solicitando primero los permisos especiales necesarios al usuario (de que requerimos el acceso al recurso de USB Host) y retornando un mensaje de error controlado si

no se lograba acceso a los dispositivos. Cabe comentar que los emuladores de Android que fueron utilizados como el de Eclipse no permiten emular la conexión de un dispositivo USB, por lo cual se hicieron pruebas en dispositivos reales.

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

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

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

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

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

Estadísticas de compatibilidad de USB Host

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

Para esto utilizamos precisamente la herramienta descrita en la sección anterior: USB Host Diagnostics, la cual sube a una pagina web [18] los resultados del diagnóstico (que también se describió en la sección anterior) para todo usuario que lo consienta. Se forma así una base de datos que expone para los distintos dispositivos donde se hizo el test, los resultados de este acerca de las capacidades de USB Host. Recorriendo distintas tiendas on-line locales buscamos tablets en venta en Uruguay que posean resultados de tests de la herramienta para llenar la estadística, incluyendo también algunos resultados de tablets personales testeadas por nosotros a las cuales pudimos acceder.

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

Modelo	API USB Host presente para 3rd party apps	Porcentaje de positivos
Samsung n8000	Si	55,2 % (42/76)
Ledstar AQ8	Solo rooted	(2/2)
ACER B1	No	0% (1/1)
Asus TF300T	Si	70% (46/65)
Lenovo A2107	No	0% (1/1)
Samsung PT5100	Si	44,7% (30/67)
Google Nexus 7	Si	50% (2/4 los demás solo rooted)

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

Modo Root en dispositivos Android

Como se especificó en las secciones anteriores, el problema de compatibilidad USB Host se podría reducir o disipar un poco si se considera acceder en el dispositivo a los permisos de root de este (conocido como "Rooting"). Si esto se logra, entonces se podría copiar o modificar el archivo necesario del Kernel para que la API de USB Host se vuelva funcional, reduciendo así la lista de

dispositivos incompatibles. Sin embargo este grupo descarta esta idea basándose en las siguientes consideraciones que asume como inaceptables:

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

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

Comunicación vía Wi-Fi

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

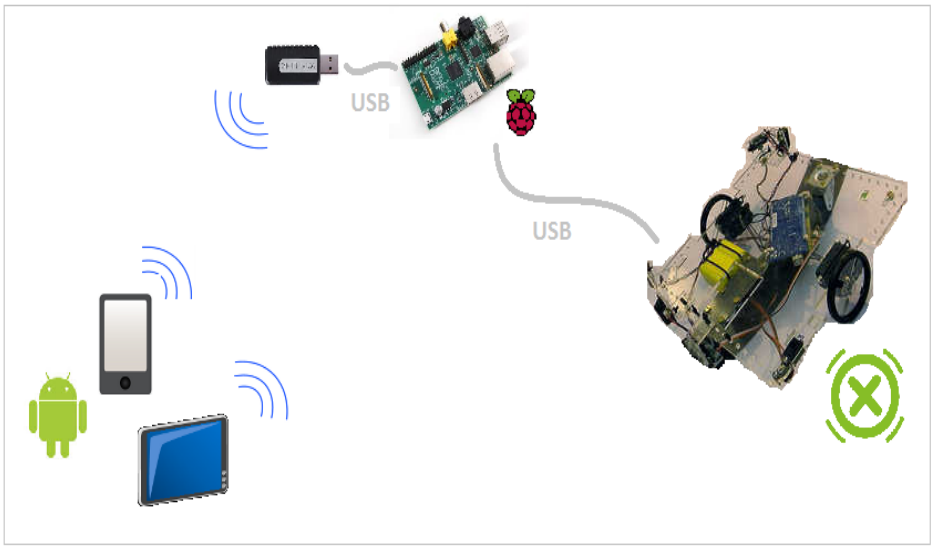

Figura 16: Diagrama de componentes asumiendo conexión Wi-Fi con Raspberry Pi.

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

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

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

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

En el marco de este proyecto como ya se identificó en múltiples ocasiones existirán dos actores principales: el dispositivo Android cliente y el robot Butiá. El primero se conoce por los mercados tecnológicos actuales y por las características de Android que dispone de soporte para conexión Wi-Fi. El segundo (robot Butiá) no lo soporta actualmente, sin embargo se tomará como cierta la hipótesis que se asume en múltiples ocasiones de este documento de que

el kit robot Butiá se expandirá para incluir una SBC que soporte USB y tenga un sistema operativo de propósito general (como por ejemplo una Raspberry pi aunque el modelo a efectos prácticos no interesa). Asumiendo esto, se incluirá un dispositivo dongle Wi-Fi que permite a la SBC establecer una conexión Wi-Fi con el dispositivo Android.

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

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

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

Interfaz Gráfica

Introducción

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

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

Scratch

Scratch, tiene un importante respaldo en la comunidad con alrededor de 37 mil visitas a su página Web en el último mes [23] y con una cantidad de 1500 nuevos proyectos subidos a la Web por dia [24]. Scratch ha influenciado fuertemente a gran parte de los proyectos posteriores de este tipo, posicionándose como una relevante referencia para los propósitos de este proyecto.

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

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

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

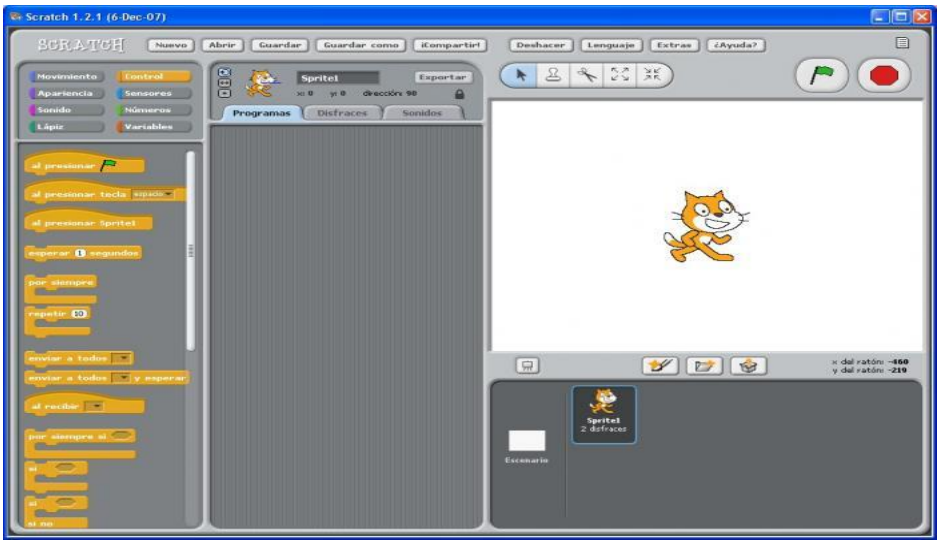

Figura 17: Sistema Scratch 1.2.1.

Opciones para HTML5

En esta sección se describirán las librerías o proyectos (de código abierto) consideradas que cumplen las características pedidas para la aplicación Web, están basados en lenguajes de programación visual con las características mencionadas anteriormente. Además, se analizará la potencial portabilidad de estos sistemas en dispositivos móviles tanto como en tablets y cuán adaptables son estos a las necesidades del IDE para el robot Butiá.

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

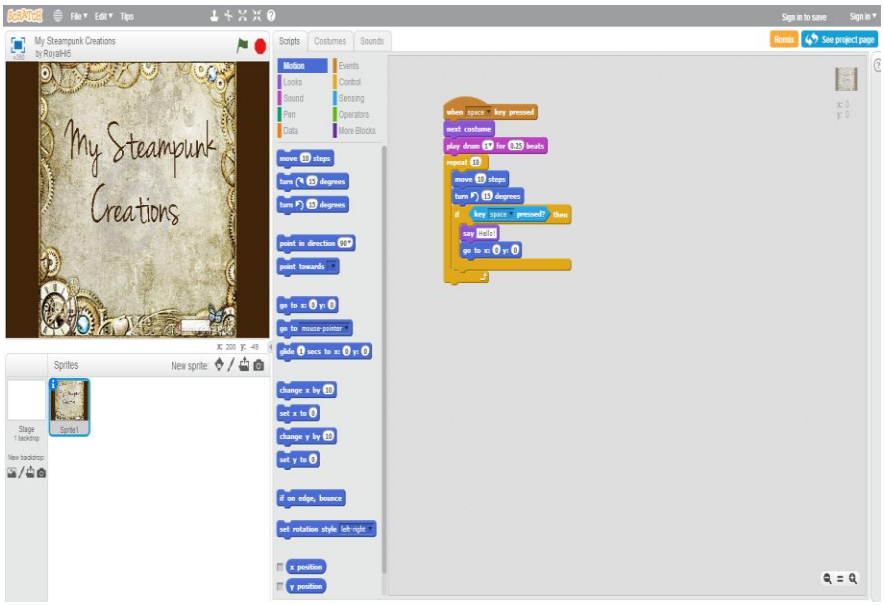

Figura 18: Interfaz de Scratch 2.0.

La versión original de este sistema forma parte del software de las XO y es compatible con la mayoría de los sistemas operativos, siendo reconocida y aceptada a nivel educativo, como fue mencionado anteriormente. Esta nueva versión permite crear y mezclar proyectos en forma colectiva, se da la posibilidad de crear nuevos bloques (lo cual equivale a crear procedimientos), también permite cierto control de gestos y hacer uso de la webcam para interactuar con los proyectos moviendo las manos o el cuerpo. Además se pueden subir datos a la

nube, fortaleciendo características en términos de la comunidad; añadiendo página de perfil donde un usuario puede mostrar sus proyectos y comentar al resto de los usuarios sobre que está trabajando actualmente. Por estas razones, a priori, Scratch 2.0 se posicionó como una excelente alternativa. Pero Scratch 2.0 está desarrollada en Adobe Flash lo cual implica una gran desventaja ya que muchos dispositivos móviles y tablet no soportan este software a nivel de usuario o necesitan instalarse APIs para poder ejecutarlo siendo incluso necesario en algunos casos rootear el dispositivo para poder instalarlo.

- **Blockly:** es un LPV open source creado 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 [12].

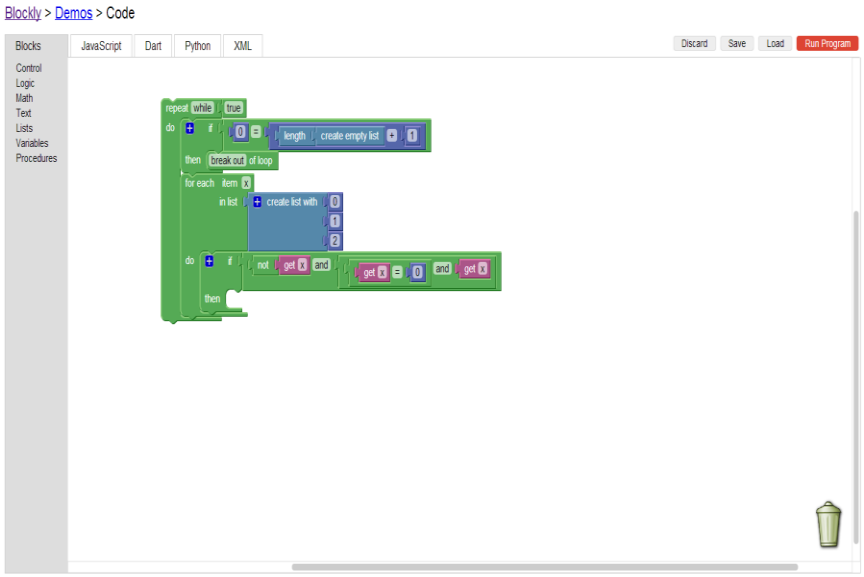

Figura 19: Interfaz de Blockly.

Está implementado en JavaScript y como tal esta 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. A pesar del buen rendimiento

en ordenadores, al experimentar con esta herramienta en dispositivos móviles y tablets no se obtuvieron buenos resultados, la herramienta no capturaba eventos de deslizamiento del dedo y por lo tanto en celulares, tablets y dispositivos con capacidad touch no funcionaba el drag and drop de los bloques. Además de eso presentó ciertas inestabilidades en el comportamiento gráfico (se probó un prototipo hosteado en Jboss en dispositivos con Android 2.3 y 4.0.4) lo cual hizo que fuera descartado como posible herramienta a utilizar ya que extender sus capacidades para manejar drag and drop y eventos touch de dispositivos móviles tomaría un tiempo considerable que escapa a los objetivos de este trabajo.

- **Snap!:** es un LPV open source desarrollado por Jens Mönig de MioSoft Corporation, con ayuda en el diseño y documentación de Brian Harvey y con colaboraciones de estudiantes de Berkeley. Snap! surge como sucesor de BYOB (Build Your Own Blocks) una avanzada modificación realizada sobre el código fuente de Scratch, enfocada en extender las funcionalidades del mismo dando, entre las mejoras más destacadas, la posibilidad de crear nuevos bloques o utilizar recursión. BYOB intenta incrementar el rango de edad de los usuarios consumidores, está orientado para estudiantes de computación o alumnos liceales. Snap! [31] es una re implementación extendida de Scratch para la Web en JavaScript, diseñada para soportar las limitaciones de los software basados en el navegador.

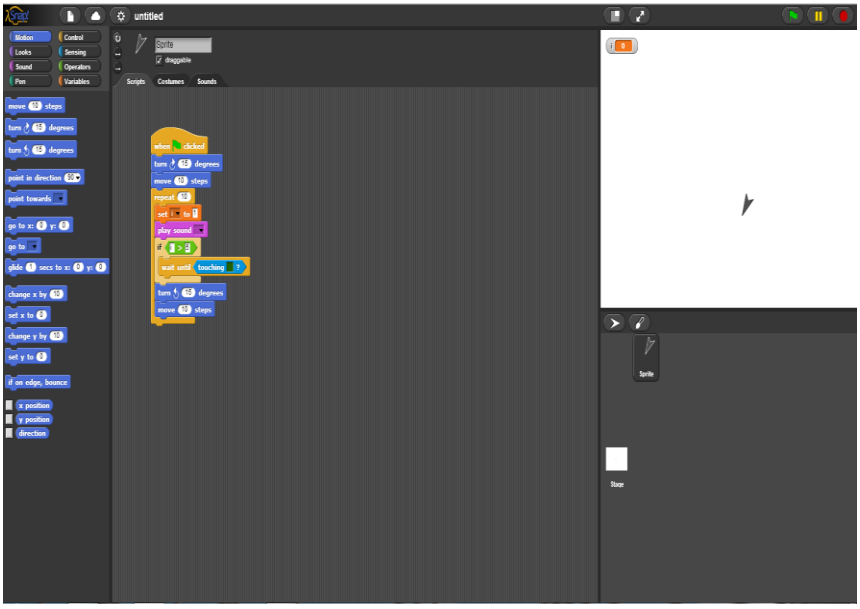

Figura 20: Interfaz de Snap!.

Snap! incorpora elementos muy interesantes desde el punto de vista de la programación pero no vale la pena entrar en detalle sobre ellos porque escapa de los fines de este proyecto, dado que los usuarios a los que se apunta serán principalmente niños con nivel escolar y difícilmente hagan uso de elementos avanzados de programación. Además posee, al igual que TortuBots, un escenario donde se simulan las acciones al ejecutar los distintos scripts y una sintaxis donde los movimientos son medidos con distancias. Estas funcionalidades no resultan acordes al paradigma elegido para la arquitectura robótica el cual se basa en comportamientos. En primer lugar, el escenario de simulación no contempla los posibles obstáculos con los que el robot puede encontrarse y resulta muy costoso realizar una simulación fiel a los movimientos del robot, luego se desea definir bloques que se mapeen a comportamientos robóticos como por ejemplo "mover robot hacia delante" en vez de definir bloques de la forma "mover robot 1 metro hacia delante" para evitar errores de precisión en los comportamientos robóticos. Al realizar pruebas en dispositivos móviles y tablets de este sistema, que cabe mencionar está en pleno desarrollo (actualmente en fase de testeo de una versión beta), tuvimos resultados positivos desde la portabilidad, pudiendo llegar a ejecutarlo, sin embargo resultó bastante lento en el renderizado y se mostró inestable, ya que se presentaron en varias ocasiones bugs bloqueantes que cerraron el navegador.

- **Waterbear:** es un LPV open source creado por Dethe Elza inspirado en el lenguaje Scratch. Destinado a principiantes en especial niños, considerado una herramienta educativa que permite crear programas con tan solo arrastrar y montar bloques sobre una superficie de diseño. Waterbear está actualmente en desarrollo y se construyó utilizando HTML5, CSS3 y JavaScript. Contiene a modo de plugins módulos con bloques que permiten generar código JavaScript o también bloques para Arduino, siendo posible también la expansión del proyecto para agregar bloques personalizados cuyo código generado sea también custom. Se encuentra actualmente en versión pre-alpha [11].

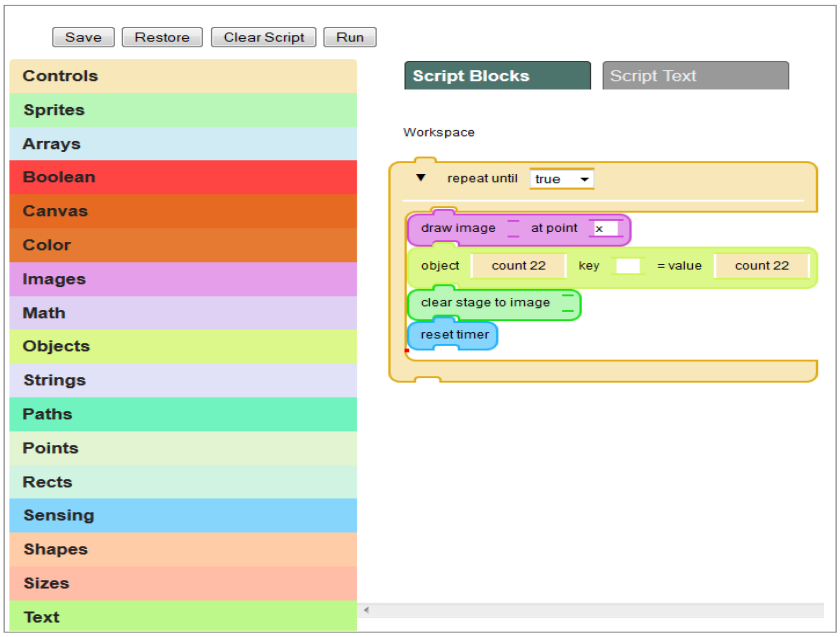

Figura 21: Interfaz de Waterbear.

Considerando el estado en construcción de esta herramienta vale la pena mencionar, sin entrar en detalle, que para poder experimentar con ella se realizaron ciertas modificaciones. Entre ellas se destaca que la versión más reciente al momento de la prueba poseía problemas en el drag and drop, a lo cual se tomaron códigos de versiones anteriores al proyecto que si tenían estabilidad (la que se podía percibir en waterbearlang.com), además se incluyeron bloques de prueba customizados a modo de extensión (como plugin) y se editó el HTML del playground original para forzar la carga de plugins a solo el plugin custom que creamos. Estas modificaciones, fueron a modo de prueba, destacando que no serán necesarias (salvo la extensión que es inherente al código de waterbear) cuando la aplicación salga de la fase alpha y se vuelva estable. Se adiciono también a modo de prueba las librerías Modernizr y YepNope para probar el comportamiento del sistema en dispositivos con distinta resolución de pantalla, construyendo unos CSSs específicos para dispositivos móviles (creados a partir de los CSSs originales) ya que la interfaz por defecto estaba pensada para ordenadores de escritorio. Además, utilizando JBoss se realizó deploy de la aplicación, permitiendo acceder a la página Web con los cambios correspondientes desde dispositivos móviles y tablets, obteniendo resultados positivos. En un principio se notó una velocidad de respuesta un poco lenta, pero esto se pudo mejorar ajustando los CSS para eliminar las transparencia de bloques al hacer drag.

Por último queda mencionar aquellos sistemas que también forman parte de la investigación pero que han sido descartados por incompatibilidades ya analizadas o que presentan menor cantidad de prestaciones, siendo el caso de Designblocks [25] quien al igual que Scratch 2.0 está desarrollado en ActionScript3, phpBlocks [26] el cual no está orientado principalmente al desarrollo educativo o LilyApp [27] el cual es un LPV pero que no está basado en bloques.

Opciones para Java con Android SDK

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

Catroid

Catroid es un sistema open source de programación visual on-device para dispositivos Android que está inspirado en Scratch. Corre en dispositivos móviles y tablets, dando la posibilidad a niños o personas sin conocimiento en programación de codificar arrastrando bloques con los dedos. Este sistema, a diferencia de Scratch o App Inventor, no necesita el uso de un ordenador ni teclado por lo que intenta aprovechar los beneficios de las pantallas más pequeñas y completamente táctiles.

Los creadores denominan al conjunto de tools, y al "lenguaje" que se genera con los bloques de Catroid, como "Catrobat" [20] o "lenguaje Catrobat". Existen actualmente versiones para iOS y versiones de Catrobat similares a Catroid para HTML5 que están en desarrollo y por lo tanto no están estables o completas. Así mismo la expansión del denominado proyecto Catroid (que incluye sistemas para distintas plataformas que manejen Catrobat y sean similares a Catroid) se encuentra actualmente planteada en el "Summer Of Code" de Google [21].

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

Figura 22: Catroid en un celular con Android.

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

limitar sus funcionalidades. Dada la estructuración, modularización y fácil aprendizaje del código, se considera a Catroid como opción perfectamente viable. Aunque tenga algunos puntos en contra, como que el desarrollar el sistema planteado en este proyecto de grado provoca desviarse mucho del proyecto original perdiendo un poco las ventajas de posibles mejoras de Catroid a futuro.

Concurrencia y orientación a comportamientos

Introducción

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

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

Semáforos y Threads en Android

El SDK de Android posee diversas herramientas para la creación y administración de Threads [29]. Si se considera que la solución está autocontenida en el dispositivo Android, se puede manejar los comportamientos en forma de threads sincronizados mediante semáforos utilizando las herramientas brindadas por Android SDK. Una idea sería por ejemplo tener una tarea “Manager” que se encargue de las tareas de sensado del robot y la administración de la ejecución de las tareas creadas por el usuario tomando en cuenta las prioridades de este.

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

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

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

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

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

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

Toribio y Lumen

Lumen es un entorno simple creado por Jorge Visca para ejecuciones multitarea basadas en corutinas. Consiste básicamente en un scheduler y fue inspirada en la descripción del scheduler de Sierra. Lumen no posee dependencias ni código de C y corre en Lua sin modificaciones, (funciona con Lua 5.1, 5.2 y LuaJIT) [30]. Posee además herramientas de logging, remote shell y web server que nos serán de mucha ayuda y que comentaremos más adelante.

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

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

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

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

sched.go()
```

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

Toribio [32] es un entorno de desarrollo de aplicaciones de robótica para plataformas embebidas. Está construido alrededor de Lumen y provee un entorno que simplifica el desarrollo y despliegue de aplicaciones robóticas. Entre los servicios que provee están:

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

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

Cada tarea a ejecutarse con Toribio se declara en un archivo de configuración en conjunto con variables o atributos propios (globales) de la tarea que también

pueden declararse allí. Toribio incluye a Bobot (ver sección de Bobot) como uno de los devices por defecto creados, lo cual permite que cualquier tarea creada pueda comunicarse con Bobot de manera sencilla y obtener fácilmente información de los sensores así como manipular los actuadores. Vemos aquí un ejemplo de esto:

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

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

end
```

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

Servidor Web y Remote Shell de Lumen

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

Cabe notar que uno de los requisitos básicos que tiene la aplicación a desarrollar es la funcionalidad de debugging. Esto significa que la aplicación debe mostrar al usuario de alguna forma los bloques que están siendo ejecutados actualmente. Para esto se necesita dar feedback a la aplicación Web de lo que el robot Butiá esta ejecutando. Si se realiza una implementación web, se podría hacer un POST inicial con los comportamientos desarrollados por el usuario

(considerados como las tareas a ejecutar por el robot) y luego al ejecutar, mediante polling de Ajax hacer sucesivos GETs al servidor web para mostrar un feedback de lo que ha sido ejecutado por Toribio. A primera vista, esta solución resulta costosa por la cantidad de mensajes HTTP que manejaría la aplicación, pero existen métodos para mejorar el polling. Entre ellos se encuentra el long polling, que propone por medio de Ajax Push [44] evitar los sucesivos GETs al servidor, reteniendo la respuesta del lado del mismo mientras no exista un nuevo mensaje para el cliente. De esta manera, además de reducir la cantidad de pedidos, se logra mejorar el rendimiento ya que la información desde el lado del servidor será enviada inmediatamente al cliente.

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

Websockets en browsers para Android

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

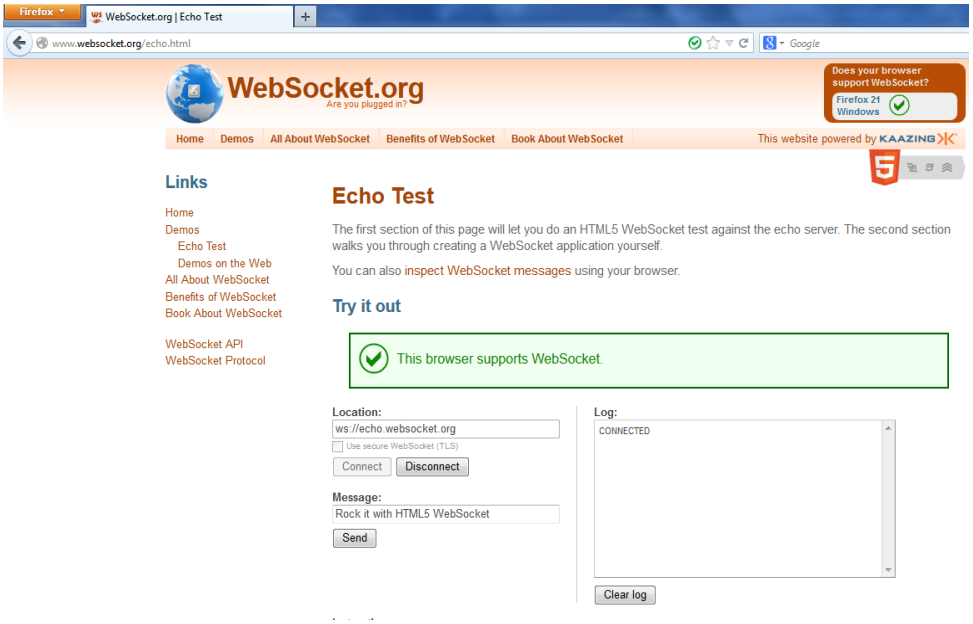

Figura 27: Echo Test con websocket.org para Firefox en un ordenador.

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

Wikipedia [42] nos muestra la siguiente tabla de compatibilidad de browsers para websockets (notar que las entradas en la tabla indican el número de versión para el cual se soporta el protocolo de websocket indicado):

Implementation status

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

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

Portar eButiá en Android

Introducción

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

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

IDE eButiá

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

El IDE eButiá fue desarrollado sobre el entorno de desarrollo Etoys, una herramienta de software libre, multiplataforma y de código abierto que viene integrada en las computadoras XO. Etoys es un entorno de programación visual

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

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

Estado actual de portabilidad ofrecida a Smalltalk

El proyecto que dispara la idea de portar sistemas Smalltalk a Android surge en el año 2009 y es el proyecto Squeak-On-Android creado por Andreas Raab, que se encuentra (incluso actualmente) disponible en el market de Android. Como se menciona en la presentación del proyecto [45], Raab, participante de la comunidad de Squeak, desarrolló una primera versión de un sistema que conseguía portar, un poco más en sentido visual que funcional, Squeak en Android. Esta versión es de código abierto y como el propio Raab lo definió, es tan solo un primer esfuerzo que tiene muchos problemas pendientes de resolver, entre ellos los más importantes el soporte a la entrada de teclado de Android y el soporte a las funcionalidades de socket y network en general.

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

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

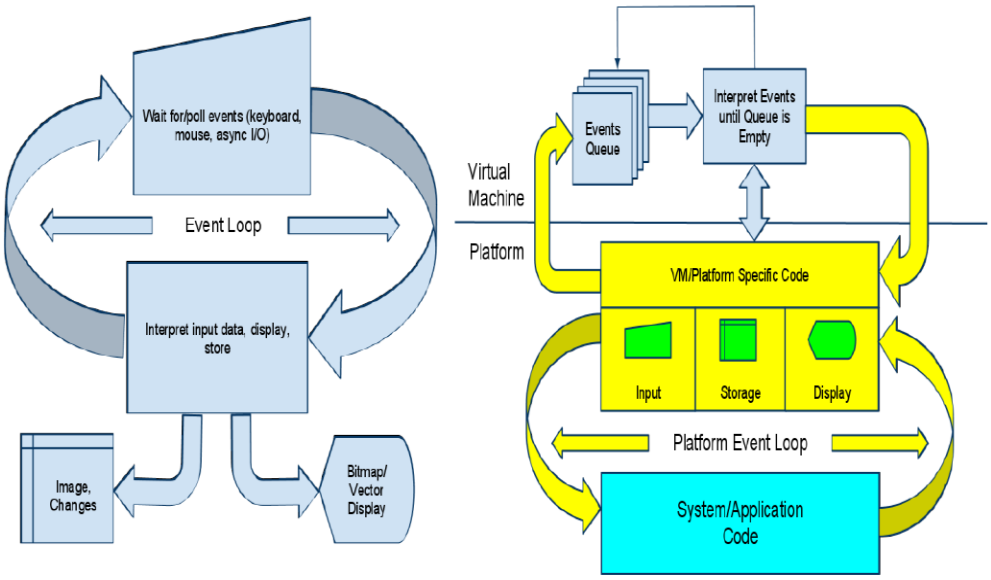

Figura 29: VM tradicional (derecha), Event-Driven VM (izquierda).

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

Dado que la interfaz de Squeak está pensada originalmente para ordenadores cuya pantalla es mayor a la que tienen la mayoría de los smartphones y otros dispositivos Android de pantalla pequeña, fue creado en el 2011 una nueva branch [48] del proyecto principal de Raab que tenía como objetivo apuntar la portabilidad de Squeak a dispositivos Android con mayor tamaño y resolución de pantalla, como son las Tablets. Este proyecto, también de código abierto al igual que el original de Andreas Raab, logró mejorar varios de los aspectos del proyecto original, entre ellos lograr una integración estable con los inputs como el teclado touch, etc.

El trabajo continuó hasta principios del 2012, en donde la página declara las releases como obsoletas ante la aparición de un nuevo proyecto (que involucra en realidad casi a la misma gente) que orientó el rumbo de la portabilidad de Squeak no hacia la máquina virtual original de Squeak sino a la conocida Cog VM [49], una máquina virtual basada en la de Squeak pero con optimizaciones y funcionalidades agregadas. Este nuevo proyecto, de código abierto como sus antecesores, tuvo dos ramas bien diferenciadas: CogDroid y PharoDroid.

CogDroid ha mantenido actividad de desarrollo hasta Octubre del 2012. Lo que ofrece es una versión estable para Android de la Cog VM. Su interfaz es simplemente una pantalla que ofrece los distintos archivos con extensión ".image" que se encontraron en el file system del dispositivo, y si se selecciona una imagen procede a cargarla con la máquina virtual. También ofrece un código precompilado de la máquina virtual y un instructivo para poder agregar una imagen (.image) y compilar todo en un único instalable de Android ".apk". Por ejemplo, podemos tomar una imagen de Squeak y compilarla con la versión precompilada de CogDroid para generar un único installer que combine ambos elementos. En principio, la VM de CogDroid no está limitada solamente a la imagen de Squeak, si bien el desarrollo de la VM tuvo a Squeak como objetivo principal. Un ejemplo de lo anterior fueron los trabajos de Hilaire Fernandes para portar Dr. Geo en Android [50]. Quien construyó un único package, reduciendo la dificultad de la instalación del sistema.

El proyecto de Pharodroid está en realidad basado en la misma idea. Pharodroid consiste en la VM pre-compilada de Cogdroid a la que le fue agregada una imagen de Pharo (la release actual tiene la versión Pharo 1.4) [51]. Por esta razón, centraremos nuestras pruebas solo en CogDroid ya que es el único que permite intentar cargar una imagen y por tanto intentar cargar eButiá. Se efectuaron las siguientes pruebas con CogDroid:

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

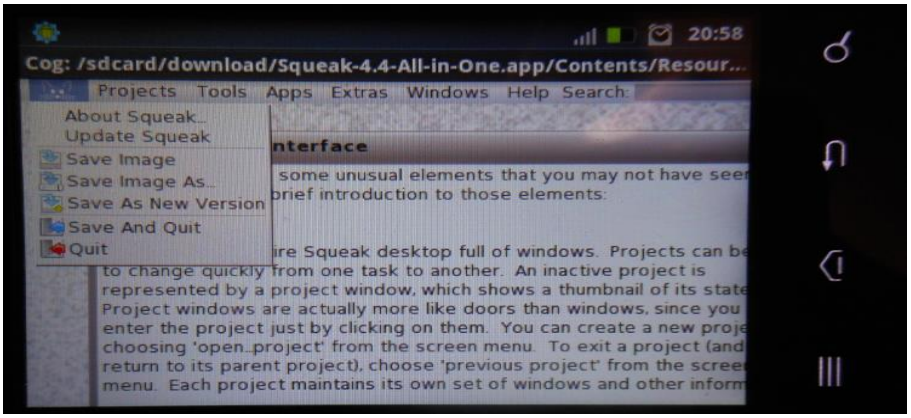

Figura 30: Squeak image con CogDroid en Android 2.3 (Motorola Defy).

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

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

Comentarios finales y conclusiones de esta sección:

Luego de las pruebas realizadas se descargó el código vigente de CogDroid (el cual resulta ser bastante grande, pesando 100 mb solo el código sin compilar) para observar sus características. El proyecto no está implementado simplemente en Java con Android SDK puro, sino que gran parte del código está almacenado en módulos escritos en C y C++ integrándose a las soluciones de Java mediante el uso de Android NDK [53], el cual es un toolset que permite implementar ciertas porciones del código de un proyecto para Android en lenguajes nativos como C y C++.

Observando el proyecto de la máquina virtual no fue fácil determinar un posible problema que repercuta en la imposibilidad de traducir la imagen de Etoys. Dado que es un proyecto poco documentado y complejo y que se desconocen las características de los componentes de Etoys a fondo, así como de

la portabilidad de estos, la curva de aprendizaje para hacer ingeniería inversa de CogDroid estimamos que tomaría un tiempo considerable. A esto se añade la problemática incertidumbre de que dicha curva de aprendizaje es solamente para determinar la causa del problema de cargar la imagen de Etoys y eButiá. El hecho de realmente lograr una implementación que solucione el problema puede quizas no ser realizable con la VM de CogDroid o tomar un tiempo no viable para el marco de este proyecto de grado.

Cabe agregar como analisis final, que en el supuesto caso de poder ejecutar la imagen de eButiá en Android, esta va a sufrir varios de los problemas detectados en el proyecto actual de CogDroid al portar Squeak, lo cual hace aún más cuestionable el esfuerzo descripto en el párrafo anterior. El problema mayor es que la interfaz gráfica tanto de Squeak como de Etoys y eButiá está pensada para ordenadores de escritorio como las computadoras XO y para dispositivos de entrada como el teclado o apuntadores indirectos como el mouse. El llevar esta interfaz a dispositivos que hacen uso de la tecnología touch para la interacción con el usuario reduce enormemente la usabilidad y la dinámica del sistema original. Otro problema actual de CogDroid es que su máquina virtual es pesada para los recursos de procesamiento y memoria que suelen tener los dispositivos Android de hoy, por lo cual puede presentar cierta lentitud la máquina virtual quitando más dinamismo al sistema. La VM ocupa al menos 50 mb iniciales de memoria, más lo que consuma la imagen en si al ejecutarse lo cual se incrementa en el tiempo si se comienza a utilizar Squeak con normalidad, mientras que en el mercado muchos dispositivos Android de gamma media poseen tan solo 256 mb de RAM.

Comentarios finales y conclusiones

Al introducirse en el mundo de la robótica educativa hemos encontrado grandes contribuciones que desde mediados del siglo XX han aportado los fundamentos para el desarrollo y evolución de esta disciplina. Su masificación en muchos países de Sudamérica está principalmente ligada al proyecto OLPC que brindó una infraestructura para poder pensar en la robótica educativa a nivel escolar y liceal. Esta realidad fue complementada por proyectos enfocados en esta rama, como lo es Butiá en el ámbito local, que pretenden otorgar instancias donde los alumnos puedan fomentar su interés en áreas como la robótica, informática, entre otras. Siguiendo esta línea, este proyecto en concreto propone el desarrollo de un IDE basado en comportamientos robóticos que permita controlar al robot Butiá desde tablets o dispositivos móviles con sistema operativo Android. En este documento se ha analizado el estado del arte con el fin de brindar a los alumnos de pequeña y mediana edad una experiencia satisfactoria, y así aportar al crecimiento de la robótica educativa en nuestro país.

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

Siendo este un sistema compuesto por dos componentes físicas, la plataforma robótica y los dispositivos móviles, es fundamental tomar decisiones respecto al medio de comunicación entre ellos. Fueron manejadas dos grandes alternativas, por un lado se investigó la capacidad de los dispositivos Android para soportar USB Host, con la intención de lograr una comunicación serial entre las

componentes. Se Encontraron inconvenientes dependiendo de las versiones del sistema operativo Android para ejecutar aplicaciones de terceros que hagan uso de la API de USB Host. Por otro lado, se asume el uso de una SBC (Raspberry Pi) como complemento en el kit del robot Butiá, asegurando la posibilidad de mantener una comunicación vía Wi-Fi entre las componentes, por medio de un dispositivo USB Wi-Fi. Obteniendo con esta última alternativa la certeza de que el sistema no depende de las capacidades del dispositivo móvil que posea el usuario, y dando además mayor flexibilidad y alcance, además de contar con una comunicación sin cables.

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

Por último, se investigaron distintos sistemas que modelan un LPV basado en bloques y respetan las características deseadas para la interfaz de usuario, siendo agrupados a partir de las tecnologías con la que fueron desarrollados. Como detalle final, en la tabla que se presenta a continuación se resume de forma sencilla las ventajas y desventajas de cada una de las alternativas estudiadas. HTML5 y el uso de Lumen y Toribio se presentan como las tecnologías con mayores prestaciones, y respecto a la interfaz gráfica el sistema Waterbear posee las condiciones más adecuadas, del punto de vista de interfaz gráfica basada en bloques, a los objetivos de este proyecto.

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

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

Referencias

A continuación se mencionan las referencias Web de este documento:

- [1] http://oa.upm.es/345/1/JOSE_MARIA_CANAS_PLAZA.pdf
- [2] http://cgi.cse.unsw.edu.au/~aishare/index.php
- [3] http://www.fing.edu.uy/inco/proyectos/Butia/mediawiki/index.php
- [4] http://www.fing.edu.uy/inco/proyectos/Butia/Usb4all
- [5] http://es.wikipedia.org/wiki/Rob%C3%B3tica_educativa
- [6] http://www.fing.edu.uy/inco/proyectos/Butia/USB4butia
- [7] http://wiki.sugarlabs.org/go/Activities/TurtleArt
- [8] http://riunet.upv.es/bitstream/Memoria.pdf?sequence=1 (pag. 36)
- [9] http://es.wikipedia.org/wiki/USB_On-The-Go
- [10] http://es.wikipedia.org/wiki/Raspberry_Pi
- [11] http://waterbearlang.com/
- [12] https://code.google.com/p/blockly/
- [13] http://es.wikipedia.org/wiki/Universal_Serial_Bus
- [14] http://www.cs.auckland.ac.nz
- [15] http://usablehack.com/amiva/AMIVA.pdf (pag. 36)
- [16] http://www.slideshare.net/ctepay/tortugarte-xo
- [17] http://developer.android.com/guide/topics/connectivity/usb/host.html
- [18] http://usbhost.chainfire.eu/
- [19] http://www.xataka.com/componentes-de-pc
- [20] https://github.com/Catrobat/Catroid
- [21] http://www.google-melange.com/catroid_project
- [22] http://arxiv.org/ftp/arxiv/papers/1204/1204.6411.pdf
- [23] https://www.quantcast.com/scratch.mit.edu
- [24] http://web.media.mit.edu/ScratchLangAndEnvironment.pdf
- [25] http://www.designblocks.net/
- [26] http://www.freegroup.de/software/phpBlocks/demo.html
- [27] http://www.lilyapp.org/
- [28] http://modernizr.com/docs/
- [29] http://developer.android.com/processes-and-threads.html
- [30] https://github.com/xopxe/Lumen
- [31] http://snap.berkeley.edu/SnapManual.pdf
- [32] https://github.com/xopxe/Toribio
- [33] http://www.fing.edu.uy/~pgiderob/EstadoDelArte-1.1.pdf (pag. 15)
- [34] http://snap.berkeley.edu/
- [35] http://www.ceibal.org.uy/docs/robotica11.pdf
- [36] http://www.ceibal.org.uy/index.php
- [37] http://www.fing.edu.uy/~pgiderob/PresentacionSqueakFest.pdf

- [38] http://virtual.uptc.edu.co/revistas/index.php/ingenieria_sogamoso
- [39] http://www.fing.edu.uy/inco/proyectos/Butia/TortuBots
- [40] http://es.wikipedia.org/wiki/VNC
- [41] http://www.fing.edu.uy/inco/proyectos/Butia/mediawiki/index.php
- [42] http://en.wikipedia.org/wiki/WebSocket
- [43] http://books.google.com.uy/books?id=cFcZadBx2C8C
- [44] http://en.wikipedia.org/wiki/Comet_%28programming%29
- [45] http://wiki.squeakvm-tablet.com/Squeak_on_Android.pdf
- [46] http://en.wikipedia.org/wiki/Turtle_graphics
- [47] http://en.wikipedia.org/wiki/Jean_Piaget
- [48] https://code.google.com/p/squeakvm-tablet/
- [49] http://www.mirandabanda.org/cog/
- [50] http://www.drgeo.eu/home
- [51] http://code.google.com/p/squeakvm-tablet/wiki/JenkinsBuilds
- [52] https://gitorious.org/cogvm/dmg-blessed
- [53] http://developer.android.com/tools/sdk/ndk/index.html
- [54] http://es.wikipedia.org/wiki/Constructivismo
- [55] http://www.fing.edu.uy/~pgiderob/EstadoDelArte-1.1.pdf (pag. 7)
- [56] http://en.wikipedia.org/wiki/Seymour_Papert