Proyecto de Grado

9 de octubre de 2013

IDE de programación orientado al desarrollo de arquitecturas robóticas basadas en comportamientos

Informe Final

Alejandro Achkar - Andrés Margalef

Tutores: Andrés Aguirre, Santiago Margni

Usuario Responsable: Gonzalo Tejera

Instituto de Computación - Facultad de Ingeniería

Universidad de la República Oriental del Uruguay

Resumen

Existen dentro de la educación distintos entornos para implementar comportamientos robóticos para el robot Butiá. Sin embargo, ninguno de estos promueve la estructuración de los programas desarrollados utilizando una arquitectura del paradigma reactivo. Partiendo de esta motivación, en este proyecto se realizó una investigación de distintas alternativas dentro de las arquitecturas reactivas más utilizadas, para orientar a los estudiantes en el desarrollo de sus comportamientos robóticos.

Con el fin de potenciar los beneficios de la enseñanza robótica como complemento de la educación, se estudiaron distintas experiencias, tanto a nivel local como a nivel mundial. Dentro de estas se pudo observar distintos beneficios. Los estudiantes mostraron una mayor facilidad de comprensión de distintas áreas de la educación, como puede ser la topología, el cálculo numérico o la física entre otras. Estos beneficios, fueron vistos también, en los alumnos de peor comportamiento y con más ausencia a las aulas de enseñanza. Quienes fueron llevados por la motivación del trabajo en la robótica, y fueron capaces de estructurar razonamientos complejos, integrando distintos algoritmos físicos y matemáticos.

Finalmente, fueron estudiados distintos entornos de desarrollo existentes dentro de la educación. Entre estos se encontraron Squeak, Etoys y TurtleBots, entre otros. Considerando los beneficios brindados por cada uno, así como sus deficiencias, este proyecto se realizó como una extensión del entorno de desarrollo existente Etoys orientado a la arquitectura reactiva Subsumption.

Palabras claves: Robótica Educativa, Robot Butiá, Subsumption, Etoys, Smalltalk, Squeak.

Índice

Índice de figuras

Índice de cuadros

Parte I
Generalidades

1. Introducción

Luego de la evaluación de distintos proyectos propuestos por diferentes docentes de la Facultad de Ingeniería de la Universidad de la República, los integrantes de este equipo, motivados a realizar un proyecto con aplicación práctica real dentro de la robótica y relacionado con la enseñanza, optaron por la implementación de un entorno de desarrollo para ser utilizado en los distintos liceos del país por estudiantes y docentes de educación primaria y secundaria.

1.1. Motivación

En 2007 con el fin, entre otros, de conseguir un alcance igualitario tanto a un computador como de internet por todos los ciudadanos del Uruguay, comenzó el plan Ceibal [25]. En él, se entregó un computador portátil a distintos alumnos de escuelas y liceos de todo el país.

Apoyándose en este plan y con el fin de que alumnos de escuelas y liceos públicos, en coordinación con docentes e inspectores de Enseñanza Secundaria de todo el país, puedan interiorizarse con la programación de comportamientos robóticos, en 2010 surge el proyecto Butiá.[28][27]. En este marco se entregó a estudiantes de primaria y secundaria de distintas escuelas y liceos del país, una plataforma robótica orientada al desarrollo de comportamientos robóticos desde la XO proporcionada por el plan Ceibal.

Dentro del proyecto Butiá se desarrollo, entre otros, TurtleBots, una herramienta basada en TurtleArt que permite la implementación de comportamientos robóticos sencillos en la plataforma. Partiendo de la motivación de los alumnos tanto por la plataforma robótica, como por los distintos entornos de desarrollo, en este proyecto se busca avanzar en el aprendizaje, permitiendo a los alumnos crear comportamientos más extensos y complejos, utilizando una herramienta que los oriente en la estructuración de sus programas mediante una arquitectura del paradigma reactivo. Es así que se busca motivar la adopción de buenas prácticas de programación, como la modularización, que les facilite la verificación y les aporte una estrategia para resolver problemas complejos mediante la separación en otros más simples.

De esta forma los estudiantes serán motivados a continuar desarrollando y aprendiendo tanto de programación, como de todas aquellas temáticas necesarias para implementar sus aplicaciones.

1.2. ¿De qué se trata el proyecto?

Es una solución a la necesidad generada por los estudiantes que actualmente utilizan la plataforma Butiá, de dar el siguiente paso en el aprendizaje de la robótica y así poder desarrollar comportamientos más complejos. El sistema que permite controlar la plataforma robótica Butiá de forma autónoma, está compuesto físicamente por dos componentes, la plataforma robótica y la XO que se conecta con esta. La plataforma, por su parte tiene una placa de control y distintos módulos de sensores y actuadores. Por el otro lado, la computadora tiene la aplicación que resuelve la comunicación con la plataforma y expone todas sus funcionalidades a través de una interfaz por socket.

La solución propuesta como componente más del sistema es, al igual que TurtleBots, un entorno de desarrollo que permite crear y ejecutar comportamientos robóticos y que mandan comandos al bobot-server para controlar la plataforma Butiá. Pero además, el entorno debe orientar al alumno a desarrollar sus comportamientos considerando una arquitectura robótica perteneciente al paradigma reactivo. Esto les dará herramientas a los estudiantes para resolver problemas complejos y adoptar prácticas de buena programación.

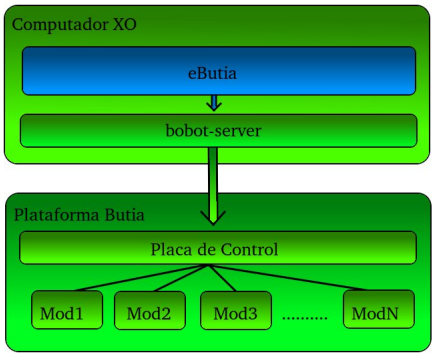


Figura 1: Componentes del sistema robótico

1.3. Objetivos

El objetivo principal de este proyecto es la creación de un entorno de desarrollo que permita a los estudiantes desarrollar comportamientos robóticos orientados a una arquitectura perteneciente del paradigma reactivo. A continuación se listan una serie de objetivos más específicos que componen el objetivo principal previamente descripto:

- Desarrollar un Entorno de Desarrollo (IDE) que permita programar comportamientos del robot Butiá, para posteriormente ejecutarlos sobre la plataforma. También debe permitir calibrar el robot (ver más adelante calibración del robot).
- El estudio de distintas arquitecturas que sigan el paradigma reactivo y la elección de una de ellas. Esta se utilizará para estructurar los comportamientos creados. La implementación de comportamientos siguiendo esta arquitectura debe realizarse de forma intuitiva, pues estará diseñada para fines didácticos.
- Debe permitir escalar los programas creados consiguiendo programas de mayor complejidad sin perder una visión global de la aplicación.
- La herramienta estará destinada a implantarse en las computadoras XO distribuidas con el plan Ceibal, por lo tanto debe ser capaz de ejecutar sobre una de ellas.

1.4. Organización del documento

El documento fue estructurado este documento conteniendo 3 grandes partes: *Generalidades*, *Desarrollo del Sistema* y *Experimentos y Resultados*. Con esta distribución se pretende realizar una introducción al proyecto, para luego adentrarnos en el mismo explicando aspectos específicos del desarrollo del sistema y finalmente mostrar las pruebas realizadas junto con las conclusiones y el trabajo a futuro. También se incluyen dos anexos, el primero es un glosario de términos y en el segundo se detallan las herramientas utilizadas para llevar a cabo el mismo.

A continuación se describen los capítulos incluidos en cada una de las partes:

Generalidades:

1 Introducción:
Se realiza una introducción al proyecto y a este documento.

2 Estado del Arte:
Aquí se incluye un resumen del documento generado en este proyecto y titulado Estado del Arte[5]. En este se realiza un estudio de distintas experiencias de enseñanza de programación robótica como complemento de la educación, también el estudio de distintas arquitecturas del paradigma reactivo, así como también distintos entornos de desarrollo para extender con el fin de desarrollar la nueva herramienta.

Desarrollo del Sistema:

3 Requerimientos Identificados:
Basada en el documento de Especificación de Requerimientos[3], en esta sección se especifican los requerimientos funcionales y no funcionales no identificados en la etapa de estudios previos.

4 Prototipos:
Es un resumen extendido del documento de Prototipos[4], donde se listan los principales prototipos desarrollados para mitigar los riesgos del desarrollo del sistema en el entorno seleccionado para su extensión.

5 Arquitectura del sistema:
Se realiza una explicación de los principales componentes del entorno de desarrollo implementado, así como también la interacción entre los mismos.

6 Diseño del Algoritmo:
Esta sección está basada en el documento Diseño - Subsumption[2], y detalla el diseño realizado para que los programas desarrollados ejecuten el algoritmo elegido para la implementación, Subsumption.

7 Verificación del Sistema:
Se detalla la metodología utilizada para la verificación del sistema así como también los resultados de la misma.

Experimentos y Resultados:

8 Estimación del tiempo de ejecución:
En esta sección se detalla la metodología utilizada para estimar el tiempo destinado a la ejecución de cada capa del algoritmo de Subsumption. Así como también, algunas de las pruebas realizadas con los tiempos de ejecución de cada una y el resultado de la estimación final.

9 Conclusiones y trabajo a futuro:
Aquí se detallan las conclusiones del trabajo realizado, junto con el listado de distintos desarrollos identificados como trabajo a futuro. Finalmente se muestra un proyecto realizado a partir de este.

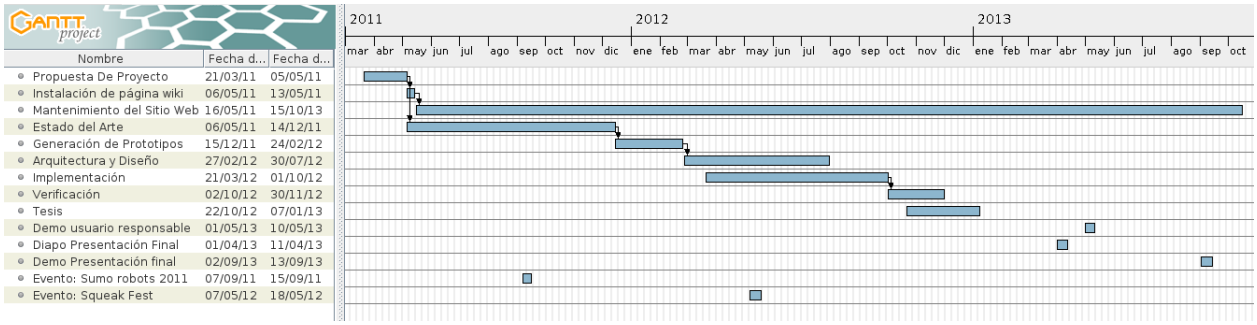
1.5. Etapas del proyecto

Si bien por distintos motivos los integrantes del proyecto han variado la cantidad de horas semanales dedicadas a lo largo del mismo, se puede estimar una dedicación horaria promedio de 22 horas semanales. El siguiente cuadro muestra la distribución de estas horas a lo largo de la semana:

Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
					9:00	8:00
						12:00
					17:00	
18:30	18:30	18:30	18:30	18:30		
21:30	21:30	21:30	21:30	21:30		

Cuadro 1: Distribución semanal de la dedicación horaria por integrante

A lo largo del proyecto se han desempeñado distintas tareas para lograr los objetivos del mismo. El siguiente cuadro muestra un diagrama con las principales tareas llevadas a cabo durante el proyecto:



Cuadro 2: Distribución temporal de las tareas a lo largo del proyecto

2. Estado del Arte

La siguiente sección es un resumen del documento generado durante el proyecto "Estado del Arte"[5]. En esta se realiza un estudio de distintas experiencias de programación robótica como complemento a la educación, distintos algoritmos del paradigma reactivo y distintos entornos de desarrollo existentes, enfocados al desarrollo de programas por parte de estudiantes que están dando sus primeros pasos en la programación.

Si bien la robótica es una de las ciencias más recientes, hoy en día existen diversos estudios y cursos que plantean su enseñanza como un complemento de la educación para todas las edades.

En está sección se realizará un breve estudio de algunas de estas experiencias. Se considerarán únicamente las aplicadas en estudios pre-universitarios. También se tomará principal enfoque en las motivaciones que llevaron a su aplicación, las conclusiones observadas por los principales actores, así como también el contexto en que fueron aplicadas.

Comprendiendo mejor estos puntos se espera realizar un entorno de desarrollo que ayude a docentes y alumnos a maximizar los beneficios obtenidos de enseñar programación robótica en estudios pre-universitarios.

2.1. Experiencias educativas

2.1.1. LOGO - Dr. Seymour Papert

El Dr. Papert[9] fue uno de los pioneros en la introducción de la informática y la tecnología en la educación. El entiende que la tecnología disminuye el trauma generado en los niños por la transición del aprendizaje por exploración al aprendizaje dependiente, ya que sirve como herramienta para que el niño pueda extender su aprendizaje por experimentación.

[31]En 1967 crea LOGO basado en sus estudios con Piaget. Lo definió como una filosofía de educación. Dentro de este contexto, observó que es una forma efectiva de lograr que los estudiantes comprendan los conceptos de programación y acerca a las matemáticas a aquellos que no se sienten motivados por ella. También destaca la sencillez para personas con capacidades diferentes.

[32]En 1980 dio un discurso dirigido a una audiencia de educadores de Costa Rica. Allí menciona ventajas para la educación de la programación aleatoria, con resultados inesperados, sobre la lineal. Considera que si uno programa para que se generen movimientos aleatorios y cambios de comportamiento basados en eventos es más interesante para los estudiantes que el programar un que robot realice una acción concreta. Esto se debe a la expectativa e incertidumbre generadas sobre si el robot va a alcanzar su objetivo.

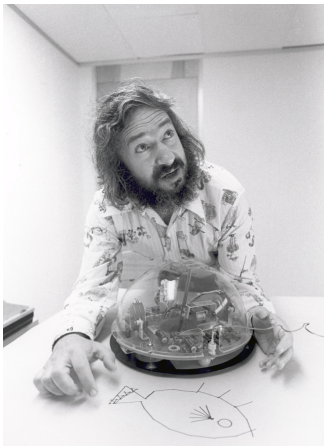


Figura 2: Dr. Papert

Observaciones: Tomando en cuenta lo mencionado por el Dr. Papert, podemos sacar las siguientes observaciones:

- Dentro de los paradigmas arquitectónicos de la robótica el más apropiado es el reactivo, ya que genera más expectativa e incertidumbre.
- Es importante la inclusión de la tecnología robótica en todas las áreas de la educación. Ya sea para evitar traumas en los cambios de métodos de

enseñanza en edades tempranas, como para motivar a los estudiantes por ciencias más abstractas como la matemáticas.

2.1.2. "Learning while Teaching Robotics" - Nueva York, Estados Unidos

[33]Esta es la denominación de un estudio realizado por el Departamento de Ciencias de la Computación de la Universidad de Columbia en los meses de Julio y Agosto de 2003. Se evaluó la experiencia de enseñar robótica en curso de verano en un liceo de Harlem. En él participaron aproximadamente 50 jóvenes liceales de entre 14 y 16 años en dos grupos. Una de las principales motivaciones perseguidas por este estudio fue el comprobar si la enseñanza de robótica alentaba a los estudiantes en el área de las matemáticas.

Figura 3: Learning while Teaching Robotics: Participantes del curso de verano

Observaciones: En lo que concierne a este documento se destacan a continuación diversas observaciones y conclusiones presentadas por las autoras en el paper publicado.

- Darles a los alumnos un modelo físico a emular era mucho más educativo que proveerles mecanismos de elaboración paso a paso.
- Los alumnos aprendían más fácil utilizando código ya escrito, modificándolo según sus necesidades y realizando reiterados testeos.
- En la sección del curso "Programación Avanzada", los alumnos aplicaron conceptos de matemática y física aprendidos en el liceo, como: sistemas discretos y continuos, geometría y álgebra, entre otros.

2.1.3. "Fundación Omar Dengo" - Costa Rica

Esta institución se ha planteado como objetivo mejorar la calidad de la educación en Costa Rica mediante de ideas pedagógicas innovadoras tomando como base la inserción de la enseñanza robótica en la educación. Esta meta es buscada desde dos enfoques distintos.

Figura 4: Fundación Omar Dengo: estudiantes de primaria diseñando modelos de robots.

Cursos dictados desde la institución: [10]La fundación ofrece cursos de dos formas distintas divididos por franjas de edades. Comienzan con diseños de robots con los kits de Lego, familiarizándose con los distintos tipos de motores y sensores, y programándolos desde la computadora. Tratan conceptos como robot autónomo y robots inteligentes que son aplicados en distintas implementaciones por los alumnos a casos reales. Para la siguiente franja, adultos a partir de 17 años, se realizan talleres orientados para profesionales educativos. Se inculcan tanto aspectos tecnológicos como pedagógicos.

Proyectos en educación pública [11]Desde 1998, junto con el Ministerio de Educación Publica de Costa Rica, lleva adelante proyectos de robótica en la educación pública de ese país en el marco del Programa Nacional de Informática Educativa (MEP-FOD).

En primaria se persigue el objetivo de sensibilizar a los estudiantes en el desarrollo de la ciencia así como también concientizarlos de su potencial creativo. En secundaria se busca involucrar a los estudiantes en su comunidad, detectando problemáticas de su entorno y elaborando soluciones robóticas a estos problemas.

Capacitación docente en educación publica [11]Además se ofrece capacitación para docentes. Comienzan con la familiarización con RoboLabs[14] a través de la programación. Se les enseña a documentar y publicar sus proyectos, y se los capacita en aspectos administrativos de la ejecución de este tipo de recursos pedagógicos en las instituciones. Luego se les enseña a identificar proyectos que acompañen la propuesta educativa.

Observaciones: De esta fundación se puede destacar observaciones de gran utilidad ya que se realiza un proyecto aplicado a la educación pública. Lo que determina un ambiente similar al del Proyecto Butiá pero con un rango de edades inferior.

- Se realiza fuerte hincapié en la inclusión y capacitación de docentes en las herramientas utilizadas.
- Los estudiantes son motivados e involucrados mediante la elaboración de proyectos aplicables en su realidad.
- Para la familiarización de los estudiantes con la programación, la herramienta utilizada es de programación visual, utilizando iconos y relaciones entre estos.
- En todos los cursos se enseñan conceptos teóricos e implementaciones de los mismos. Como robots autónomos y/o inteligentes.

2.1.4. Jardín de Infantes de la Asociación de Empleados Bancarios del Uruguay (A.E.B.U.):

A principio de la década de los noventa, profesores de informática y maestros de la institución estudiaron la posibilidad de incluir un proyecto de informática y de iniciación a la robótica al proyecto educativo institucional. Se buscó incluir a la tecnología no solo como una área en si misma, sino también, como una herramienta más para el aprendizaje.

Para la iniciación a la robótica se utiliza el robot Jeulin T-3. Este tiene la capacidad de desplazarse en una superficie lo suficientemente lisa y dibujar, alzando y bajando un marcador. Posee también un par de pinzas que son capases de agarrar y levantar objetos. Se utiliza un control remoto que es comandado con tarjetas perforadas correspondientes a las distintas acciones que es capaz de realizar para comandar el robot. Las edades de los niños dividen el curso en dos etapas que describiremos a continuación.

Actividades para alumnos de 4 años: En las primeras clases se busca que el alumno conozca tanto a sí mismo como a su entorno mediante actividades lúdicas. Luego, se les presenta el robot y se los da para que lo examinen. Los alumnos divisan el botón de encendido, una batería que lo alimenta y algunos de los actuadores como los motores que mueven las ruedas. Posteriormente se les entrega el control remoto junto con las tarjetas perforadas que lo comandan para que las prueben. Luego se les añade el marcador, se coloca el robot sobre una lámina de papel y encuentran la relación del dibujo con respecto a la posición del lápiz. En esta primera etapa se introducen implícitamente, entre otros, conceptos como: topología, geometría, cálculo y proyección. Finalmente intentan dibujar alguna figura concreta, comprobando sus hipótesis sobre como debía ser realizado y estudiando el resultado.

Actividades para alumnos de 5 años: Comienzan la actividad estudiando y comprendiendo mejor el almacenamiento de la energía. Luego se introduce el concepto de que el robot es programable. Es decir, se le pueden introducir una secuencia de comandos al robot para que luego las ejecute secuencialmente. Los alumnos primero plantean sus ideas en papel introduciendo grandes conceptos de escritura y simbología. Luego los alumnos ejecutan sus programas sobre el robot y observan los resultados.

Figura 5: Robot Jeulin T-3 y Tarjetas de control

Observaciones: Este proyecto es muy interesante para poder realizar observaciones de una experiencia en etapas tempranas de la educación "formal" del individuo.

- Se puede apreciar claramente una aplicación enfocada a minimizar el trauma generado por el cambio de métodos de aprendizaje, así como también la evolución con el mismo, mencionado por el Dr. Papert.
- También se observa nuevamente la inclusión de conceptos matemáticos y topológicos, entre otros, que generalmente son dificultosos de adquirir por el estudiante, de forma sencilla y lúdica.

2.1.5. Proyecto Butiá

En 2007 Se lanza el Plan Ceibal, una iniciativa que busca insertar la tecnología digital en la educación primaria y secundaria. En una primera etapa se entregó a niños de las escuelas una computadora portátil XO del proyecto One Laptop Per Child (OLPC). Luego se extendió entregándoles a alumnos liceales un ordenador de potencia un poco mayor, en su mayoría XO 1.5[25].

Sobre este proyecto a principios de 2010 se lanza el Proyecto Butiá. Este busca acercar a alumnos escolares y liceales a la programación de comportamientos robóticos. Con este fin se lanza el robot Butiá, una plataforma comandada por una XO colocada sobre ella conectada a través de un cable USB. Esta plataforma es proporcionada a diversos liceos para que puedan ser utilizadas por los alumnos en laboratorios. Actualmente se han entregado en 50 liceos[28]. Posteriormente se les invita a participar del Campeonato de Sumo Robótico[36].

La plataforma Butiá posee distintos sensores y actuadores, que son reconocidos por la placa de control [26]. Para utilizar el Butiá desde la portátil, se creó una aplicación demonio, el Servidor Lua (Bobot Server), que recibe funciones en formato de texto plano a través del puerto 2009, o a través de una interfaz web.

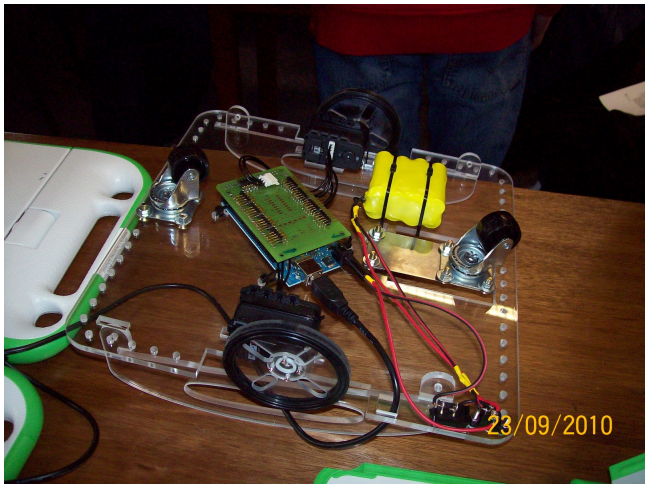


Figura 6: Plataforma Butiá

En agosto de 2010 se crea una extensión a Tortugarte que permite controlar el robot Butiá mediante la programación de comportamientos. Al ejecutar el programa, este se comunica con la aplicación demonio que controla el robot, que es levantada automáticamente por el entorno. Otra aplicación desarrollada como plugin para Tortugarte es Follow Me, creada con el fin de aprovechar la cámara web de la pc como un sensor más[27].

También se ha creado otra aplicación llamada Butialo (derivada del entorno de desarrollo Pippy), que permite programar comportamientos para el Robot Butiá en lenguaje Lua. Dentro de las principales ventajas ofrecidas, se encuentra

la auto-detección de módulos conectados al robot.

Dentro del marco del proyecto se ofrece también, un curso en Facultad de Ingeniería llamado Butiá Robótica Educativa. Este permite a los estudiantes familiarizarse con la robótica desde un punto de vista educativo, así como también con el robot Butiá. Con este fin se brinda la experiencia de realizar, junto a los docentes, talleres en distintos liceos del país. Finalmente los estudiantes deben realizar un proyecto con los conocimientos adquiridos.

2.2. Arquitecturas reactivas

El paradigma reactivo o también denominado paradigma PA (Percepción-Acción) surgió en la década de los '80 y se trataba de enfocarse más en los razonamientos del reino animal en vez de los del ser humano, tiene como beneficio su sencillez pero como contrapartida la incapacidad de realizar tareas con un nivel de complejidad alta.

Toda acción llevada a cabo se realiza en la forma de comportamientos, estos son vistos como acoplamientos mínimos y funcionan como transformación o mapeo entre las percepciones percibidas y las acciones realizadas a los actuadores.

El paradigma define que toda acción de sensar se realice de manera local a cada comportamiento, y además cada comportamiento puede sensar el mismo valor por separado como indica la figura 7.

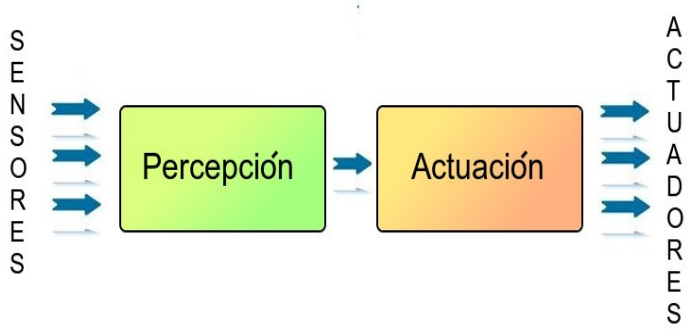


Figura 7: Estructura del paradigma reactivo

Un detalle de los sistemas basados en este paradigma es que no tienen memoria. En la práctica existen algunos comportamientos que se mantienen durante un lapso de tiempo sin presencia de percepción alguna.

Este tipo de arquitectura poseen como ventaja el ser de bajo acoplamiento y de alta cohesión. Debido a la separación de los comportamientos, logran funcionar de manera bastante independiente teniendo únicamente una leve conexión entre ellos. La cohesión se logra porque cada comportamiento encapsula de forma bien definida solo las operaciones que sirven para el propósito de ese módulo.

A continuación se muestran las distintas arquitecturas estudiadas en el documento de Estado del Arte[5].

2.2.1. Arquitectura basada en Campos de potencial

Campos de Potencial es una arquitectura del paradigma reactivo, ya que no posee memoria y los resultados de sus acciones son consecuencias directas de los datos obtenidos de los sensores.

Esta arquitectura define el mundo sobre una cuadrícula. En el mundo existen distintos objetos. Cada uno ejerce una fuerza definida por un par {dirección, módulo}, dentro de un área de influencia. Esto resulta en un conjunto de fuerzas ejercidas sobre cada campo de la cuadrilla. La fuerza resultante de la suma vectorial de las fuerzas ejercidas en un determinado campo de la cuadrilla determinará el comportamiento del robot en esa posición.

La aplicación más intuitiva es la resolución del problema de movimiento. De ésta se desprenden rápidamente tres problemas de fácil solución:

- No colisionar con el objetivo: como el objetivo tiene una fuerza atrayente, el colisionar con los objetivos parece un problema desprendido de la arquitectura. Este se puede solucionar fácilmente aumentando la tolerancia de arribo al objetivo. De esta forma el robot se detiene antes de llegar al objetivo, ya que detecta que ha llegado, y se detiene evitando la colisión.
- Limitaciones para seguir el movimiento: dado que se requiere que el robot se mueva en dirección del vector, el hardware del robot es determinante. Si se tiene un robot construido con la capacidad de girar sobre su mismo eje, puede tomar la dirección del vector en el lugar donde se encuentra. Sin embargo si el robot tiene el método de direccionamiento de un automóvil, este precisa desplazarse para cambiar la dirección ya que posee un ángulo máximo de rotación de sus ruedas.
- Valles donde el campo de potencial sume cero: es posible que la suma de los vectores en distintos puntos del plano sume cero, generando mínimos donde el robot se quedaría estancado. Una solución sencilla a este problema es tener un generador de ruido, que siempre genere un pequeño ruido que se le añada al vector resultante. De esta forma se puede salir de los valles problemáticos.

Esta arquitectura posee las siguientes ventajas:

- Resuelve muy eficazmente y de forma muy sencilla el problema del movimiento, generando trayectorias suaves.
- Se puede implementar de modo que el aumento o disminución de la cantidad de sensores del mismo tipo tenga un impacto mínimo en el código.
- Velocidad de respuesta a los estímulos
- Simplicidad computacional

También posee ciertas limitaciones:

- Su aplicación se basa en mapear los problemas con vectores de fuerza. Esto lo hace más dificultoso para resolver problemas de forma intuitiva.

- No resuelve eficientemente el problema del movimiento, ya que la trayectoria generada no necesariamente es la del camino más corto o de menor tiempo.
- Los agentes necesitan tener en el entorno local la información suficiente sobre que tarea realizar
- La arquitectura no contempla el construir agentes con comportamientos de distintas índoles.
- Ausencia parcial o total de la capacidad de aprendizaje. Las decisiones se basan en el mundo actual sin considerar el pasado. Esto provoca que ante una situación similar a una ya ocurrida, se realice la misma acción, aunque esta no haya resuelto el problema.

2.2.2. Arquitectura Subsumption

Subsumption es una arquitectura basada en el paradigma reactivo que define la siguiente serie de tareas que se deben realizar dentro de cada comportamiento:

- Tarea de sensar,
- Tarea de transformación (complejidad baja)
- Tarea de actuar

Básicamente la inteligencia del robot se puede particionar en comportamientos, donde cada uno de ellos tiene como objetivo llevar a cabo determinada tarea. Cada comportamiento está compuesto de una determinada topología en red fija, de maquinas de estados finitas. En otras palabras es una red de módulos de sensores y actuadores conectados de manera tal que se pueda realizar la tarea asignada.

Esta arquitectura tiene las siguientes características:

- Agrupación en capas de los comportamientos, de esta forma se puede establecer una jerarquía en las tareas que componen el robot, donde las tareas más bajas pueden ser tareas de supervivencia por ejemplo.
- Capas superiores pueden sobrescribir o realizar subsunción a la salida de la capa inmediatamente inferior.
- Ausencia de estado persistente, de esta forma no se mantiene información que represente el mundo o algún otro modelo. En casos excepcionales resulta necesario trasgredir este punto, pero siempre se trata de minimizar estos casos.

La figura 8 es una ilustración de como seria la estructura de capas:

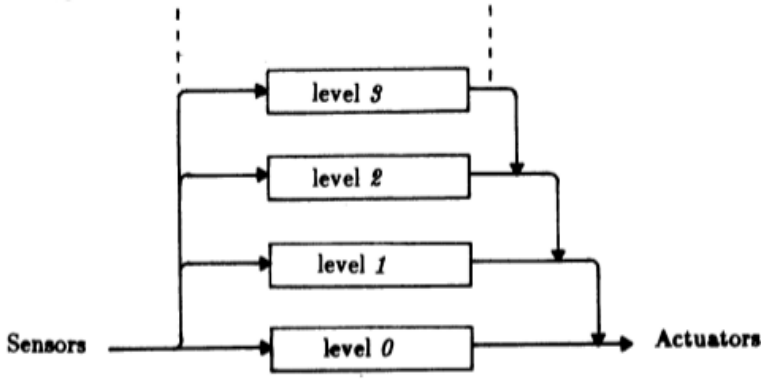


Figura 8: Diseño en capas de la arquitectura Subsumption

Las capas superiores puedan alterar el camino de los datos de las capas inferiores así como también alimentarse de información que provean las mismas.

Las dos formas en que se pueden realizar Subsumption son las siguientes:

- Inhibición, la salida de un módulo A en una capa superior se conecta a la salida de otro módulo B en una capa inferior, en caso de que exista salida en el A, se bloquea la salida del módulo B.
- Supresión, la salida de un módulo A en una capa superior se conecta a la entrada de un módulo B de una capa inferior, en caso de que haya salida en el módulo A, se reemplaza la entrada del módulo B por la del A, en otro caso no se tiene en cuenta la salida de A.

Existen variantes en la arquitectura, modificaciones que se hicieron a través del tiempo, una de ellas se plantea en "A Robot that Walks; Emergent Behaviors from a Carefully Evolved Network"[7] que es la supresión de un mensaje por un periodo corto de tiempo, haciendo que si se quiere suprimir por tiempos prolongados se debe mantener un suministro de mensajes de supresión. Otra modificación se plantea en "A Colony Architecture for an Artificial Creature"[8] donde se cambia la estructura de capas en forma de pila a forma de árbol.

Esta arquitectura posee ciertas limitaciones

- Los agentes necesitan tener en el entorno local la información suficiente sobre que tarea realizar
- Ausencia parcial o total de la capacidad de aprendizaje
- Las decisiones se basan en el mundo actual sin considerar el pasado.

y ciertas ventajas

- Velocidad de respuesta a los estímulos
- Simplicidad computacional
- Sencillez (si no se utilizan gran cantidad de comportamientos)

2.2.3. Teoría de Esquemas

El concepto de esquema es utilizado desde principios del siglo XX por los psicólogos, que hacía referencia a una unidad básica de actividad. En 1981 Michael Arbib refina este concepto definiendo como esquema la función que va desde los conocimientos de saber como percibir y/o actuar hasta el proceso computacional utilizado para llevar acabo la acción.

Comportamientos:

De esta forma podemos definir un comportamiento reactivo como un esquema con un estímulo definido de liberación y compuesto a su vez por dos esquemas: uno de percepción y uno motor. El estímulo es quien libera la acción de todo el comportamiento. El esquema de percepción es quien interpreta la información de los sensores y la procesa en un modelo interpretable por el esquema motor. Finalmente el esquema motor es el encargado de interpretar la salida procesada por el esquema de percepción y calcular la acción a tomar por los actuadores para ejecutar la acción deseada.

Programación Orientada a Objetos:

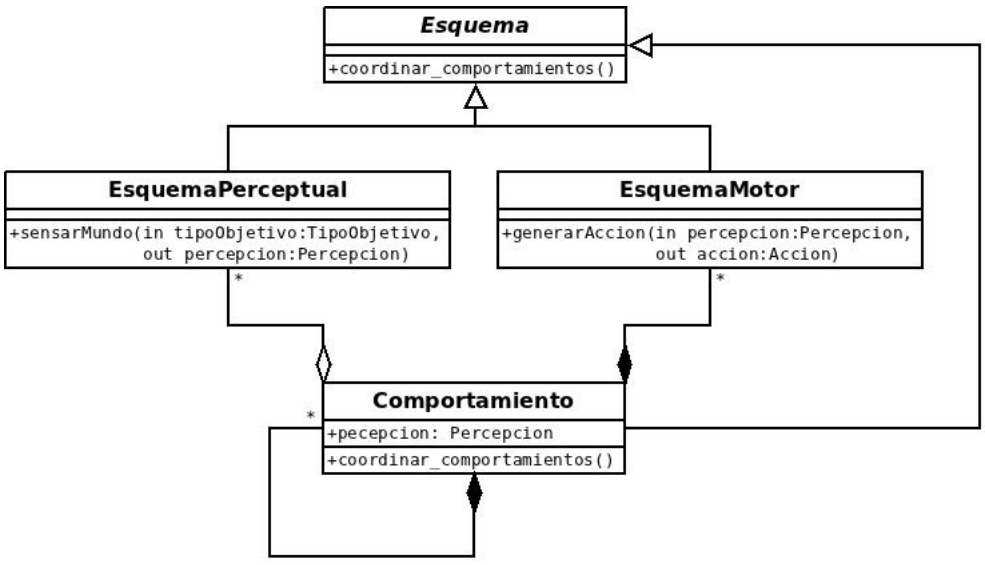


Figura 9: Diagrama de clases de esquemas y comportamientos

Un esquema puede verse, desde el punto de vista de la programación orientada a objetos, como una plantilla que contiene atributos, estructura de datos y métodos. Dentro de este dominio, podemos ver a un esquema como una clase abstracta que tiene un método opcional para coordinar los esquemas. Este

método es el encargado de combinar lo retornado por el o los esquemas motores y generar una salida combinándolos. La clase es implementada por los esquemas motores, perceptuales y los comportamientos. Los comportamientos implementan el método de coordinación.

La clase de esquema de percepción esta directamente vinculada con los sensores así como la clase motor está directamente vinculada con los actuadores.

Existen al menos, dos formas de implementar los habilitadores. La primera es utilizar clases procesadas por el programa principal, una por cada comportamiento. En caso de dar positivo, instancia el comportamiento correspondiente. La segunda, es tener los comportamientos habilitados permanentemente y tener una clase de habilitación dentro de cada uno. En caso de no estar habilitado, el comportamiento devuelve una acción neutra que corresponde a que no está habilitado.

2.3. Entornos de programación existentes

Se realizo un estudio sobre los diferentes entornos existentes con fines educativos, con el fin de usar como base para la realización del entorno de desarrollo que comprende este proyecto.

Los entornos estudiados son los siguientes:

- Squeak Smalltalk
- Etoys
- Physical Etoys
- Scratch
- Turtle Blocks

Los aspectos que se usaron para evaluar cada uno de ellos se listan a continuación

- ***Respaldo de la Comunidad que mantiene y utiliza el software***
- ***Facilidad de Extensión***
- ***Restricción en licencias***
- ***Funcionalidades ofrecidas***
- ***Curva de aprendizaje de la herramienta***
- ***Restricciones del contexto de ejecución***

  Son restricciones en el contexto o entorno de ejecución por ejemplo sistemas operativos y requisitos de hardware necesarios para ejecutarse.
- ***Sincronización de cambios en tiempo de desarrollo***

  Facilidad para compartir cambios en tiempo de desarrollo de la aplicación entre los desarrolladores.

2.3.1. Squeak Smalltalk

Smalltalk es la conjunción de tres grandes componentes

- Lenguaje
- Librería de clases

  Practicamente es el núcleo del sistema, provee de todas las funcionalidades básicas que uno espera tener en un lenguaje además de una colección de clases utilitarias multiplataformas que se pueden usar para el desarrollo de todo tipo de aplicaciones.

- Entorno de desarrollo

  Permite la implementación de aplicaciones orientadas a objetos. Toda implementación hace uso del lenguaje y de las propias herramientas brindadas.

Dado que este proyecto esta centrado en la implementación de un entorno de desarrollo, se enfocara más en este aspecto.

El lenguaje Smalltalk Smalltalk es un lenguaje de programación orientado a objetos y es considerado como el lenguaje que se ajusta más a este paradigma desde el punto de vista purista.

En comparación con otros lenguajes de programación, Smalltalk resulta ser muy simple, y no es costoso llegar a conocerlo de forma completa.

Su modelo soporta herencia, clases, instancias concretas de clases, asociaciones dinámicos, pasaje de mensajes y garbage collection.

El entorno de desarrollo Brinda los artefactos necesarios para la visualización del código fuente (inclusive todo el código del propio entorno), herramientas para debugging y ejecución de código de manera dinámica como se vera más adelante.

- Modelo de objetos

  El propio entorno ofrece un extenso framework de desarrollo que abarca desde componentes visuales a librerías utilitarias.

- Herramientas de desarrollo

  Son las encargadas de brindar al programador las funcionalidades básicas como avanzadas, por ejemplo creación, borrado y modificación de clases, debugging del código fuente, inspección de datos y alteración de los valores de los mismos, etc.

- Ambiente de ejecución

  Squeak consiste en dos partes, una máquina virtual que interpreta código binario, y el código en si que está alojado en un archivo de extensión .image.

El archivo .image es el ejecutable, ejecutando la máquina virtual desde un terminal con parámetro la ruta al .image, se consigue iniciar la aplicación, como se muestra a continuación.

zeus@pc:~$ Squeak ~/Squeak/Squeak4.1.image

También existen dos archivos que acompañan al binario:

- archivo de cambios (Squeak4.1.changes)
- archivos de fuentes (SqueakV41.sources)

El binario contiene todos los objetos, clases, diccionarios, etc que se crean dentro de la aplicación, así como también el último estado que se salvo la imagen. Cuando se inicia la aplicación todo permanece igual al último estado.

Cada cambio en la imagen también se ve reflejado en el archivo de cambios. El código base de la imagen se encuentra en el archivo .sources, este es el código fuente de la imagen sin ningún cambio. La modificación de métodos así como la inclusión de otros nuevos, como nuevas clases, etc son agregados al archivo .changes en orden cronológico en que se efectuó cada cambio.

La organización del código realizada por Squeak consta de 4 partes:

- Categorías de sistema

  Son simples agrupaciones de clases.

- Clases
- Protocolos

  Los protocolos son agrupaciones de métodos.

- Métodos

Las clases y los métodos tienen la misma connotación que en los demás lenguajes orientados a objetos.

Squeak ofrece la funcionalidades denominadas *FileOut* y *FileIn* que sirve para exportar e importar respectivamente cualquiera de las 4 partes definidas anteriormente. Es decir que un desarrollador esta implementando con una imagen y quiere compartir con otro desarrollador que está trabajando con otra imagen diferente, algunos métodos, o clases o grupo de ambos, se puede realizar el FileOut respectivo a la parte en cuestión generándose un archivo externo. Luego el desarrollador interesado en recibir la información realiza el FileIn seleccionando el archivo anteriormente generado, incluyendo toda la información exportada.

Funcionalidades ofrecidas Squeak tiene una componente denominado *Flap,* estas son barras de herramientas donde se pueden ubicar objetos para dejarlos disponibles al uso del usuario.

Generalmente la utilización de estos objetos se realiza al estilo Drag&Drop soltándolos en la Área de trabajo, habiendo muchos componentes visuales disponibles

para su uso en el armado del proyecto, estos pueden ser tan diversos como rectángulos, elipses, labels, botones, etc.

Browser

Squeak ofrece herramientas útiles para la administración de código fuente mientras se esta desarrollando, para esto existe la ventana de exploración denominada Browser, como se muestra en la figura 10.

Los cuadrantes superiores ubicados de izquierda a derecha hacen referencia a *Categorías de sistema*, *Clases*, *Protocolos* y *Métodos* respectivamente.

El cuadrante inferior es el área habilitada para la programación en lenguaje Smalltalk.

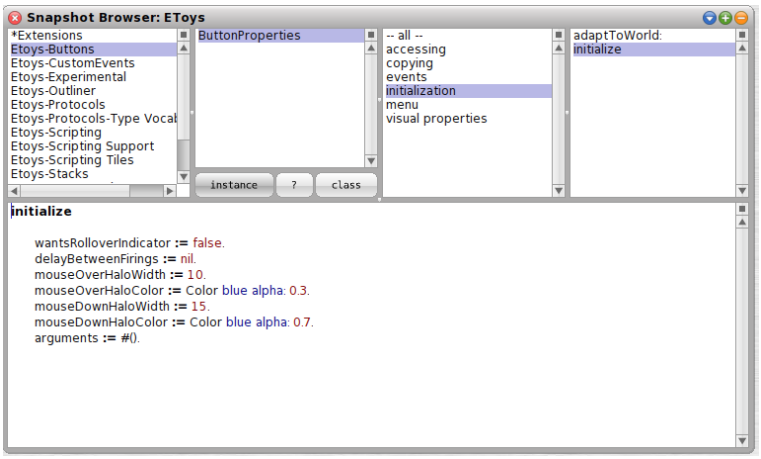


Figura 10: Explorador de paquetes

Workspace

Es usado para la ejecución de código de forma dinámica, ya sea para realizar pruebas concretas sobre objetos o para realizar ejecuciones de scripts de pocas lineas y examinar su comportamiento. Consiste en un cuadro de texto con la posibilidad de ejecutarlo.

Catalogo de paquetes

Sirve para conectarse a varios repositorios en la web permitiendo la instalación de múltiples utilidades extras.

Estas nuevas herramientas que se pueden instalar pueden ser desde plugins para compartir código fuente (cvs o svn) como también aplicaciones o juegos completos.

Respaldo de la Comunidad que mantiene y utiliza el software Tiene una comunidad bastante activa donde participan usuarios diversos como maestros, entusiastas, usuarios académicos.

Cuenta con una "Mailing List" [35] con listas orientadas a aquellos usuarios que se inician en la programación con Smalltalk, como para aquellos más avanzados

También existen varios sitios apoyando a la comunidad como lo son

- Squeak Board Blog

  Blog donde se publican anuncios y decisiones, y es el encargado de coordinar el desarrollo open-source de la comunidad Smalltalk.

- Weekly Squeak

  Es un blog de novedades donde se publican noticias, informes y entrevistas sobre temas relacionados al mundo de Squeak y su comunidad.

- PlanetSqueak

  RSS que tiene algunas entradas de desarrolladores que pueden resultar de interés.

- ESUG (European Smalltalk User Group)

  Asociación sin fines de lucro que reúne a los usuarios europeos de Smalltalk. Promociona el uso de esta plataforma y la comunicación entre los usuarios de ellas, usando cualquier medio como periódicos, reuniones, y también organizando eventos como *el Smalltalk Yearly Conference*.

- ClubSmalltalk

  ClubSmalltalk es un sitio para la comunidad de Smalltalk que inicialmente creció siendo una Mailing List colaborativa entre programadores. Actualmente se pueden encontrar aplicaciones, noticias de eventos, artículos, y otros tipos de publicaciones.

Restricción en licencias Squeak hace uso de dos licencias [34] cada una de ellas aplica en ciertas partes del código.

Las licencias que se utilizan son:

- MIT License [17]

  Esta licencia es muy permisiva, permite el uso, copia, modificación, publicación, distribución, sublicenciar y/o vender el software.

  La única restricción es que el copyright se incluya en todas las partes o partes sustanciales del software.

  La licencia es la siguiente:

  Copyright (c) The individual, corporate, and institutional contributors who have collectively contributed elements to this software ("The Squeak Community"), 1996-2010 All rights reserved.

  Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

- Apache License, Version 2.0 [6]

  Esta licencia es tan permisiva como la del MIT, dándole la posibilidad de uso, copia, modificación y redistribución de la obra protegida, resultando ser una licencia de software libre sin tener clausulas que obliguen a usar aplicar esta licencia al producto que haga uso de uno licenciado por la misma. Se exige que halla dos archivos en el directorio principal de los paquetes de software redistribuidos:

  - LICENSE - Copia de la licencia
  - NOTICE - Documento de texto, con los "avisos" obligatorias del software presente en la distribución.

  Además es obligatorio mantener el aviso de copyright y el disclaimer.

  La licencia es la siguiente:

  Copyright (c) Xerox Corp. 1981, 1982 All rights reserved. Copyright (c) Apple Computer, Inc. 1985-1996 All rights reserved.

  Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Facilidad de Extensión Squeak es un ambiente totalmente abierto, debido a que su licencias son lo suficientemente libres como para hacer cualquier tipo de uso sobre el código, por lo que se puede modificar el código del entorno tanto como se quiera.

Así mismo el entorno viene incluido con todo el código fuente, y además proporciona una variedad de componentes muy diversos para hacer uso de los

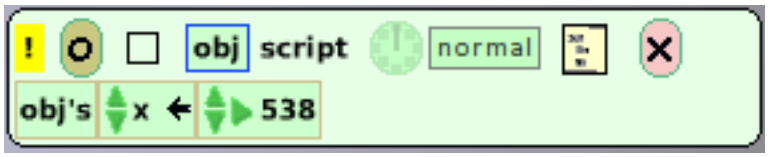


Figura 11: bloque que asigna a la propiedad x del objeto *obj* el valor *538*

mismos, como por ejemplo los componentes visuales pueden ser extendidos y modificados sin costo alguno ya que se cuenta con una herramienta visual para este motivo.

Curva de aprendizaje de la herramienta La curva de aprendizaje del entorno es sencilla, basta con conocer algunas pocas de las utilidades brindadas por el IDE para poder comenzar a desarrollar aplicaciones de pruebas. Conociendo el Browser y el workspace se esta habilitado a crear toda la estructura de objetos necesaria, y poder ejecutar el código de forma dinámica para analizar el comportamiento del mismo. Referente al aspecto visual, también es sencillo ya que para la creación de los objetos visibles en pantalla Squeak nos brinda una herramienta dedicada para esto.

No así, el lenguaje Smalltalk puede ser costoso al comienzo pues difiere de como se manejan otros lenguajes de programación. Consiste en un paradigma de pasaje de mensajes con una sintaxis diferente a lenguajes orientados a objetos como lo son c ++, java, .net, etc.

Como aspecto positivo se tiene que no es un lenguaje con muchas características o aspectos particulares, esto lleva a que aprenderlo de forma completa lleve una cantidad de tiempo menor que otros lenguajes.

Curva de aprendizaje para el código usado por el usuario final (código orientado a bloques) El código final que los usuarios del sistema podrán crear es orientado a bloques como se puede ver en la figura 11

Además Squeak brinda la opción de poder ver el código Smalltalk generado por los bloques como se muestra a continuación.

Restricciones del contexto de ejecución Smalltalk se ejecuta sobre una máquina virtual que esta desarrollada para las siguientes plataformas

- casi todas las versiones de Windows incluyendo CE/PocketPC
- MacOS
- Linux/Unix
- OS/2 Warp
- Acorn RiscOS

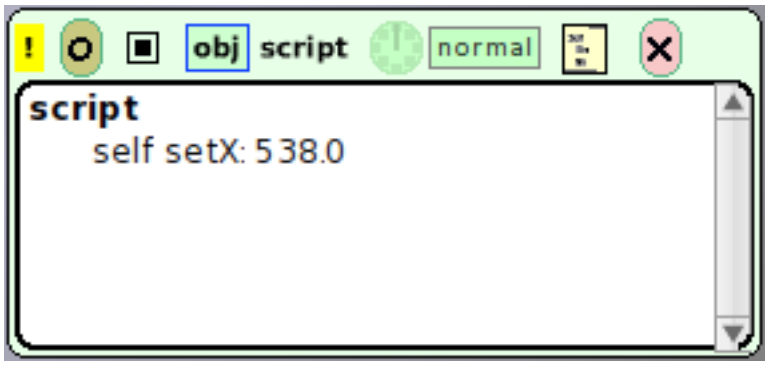


Figura 12: Código en Smalltalk generado

- incluso en el chip M32R/D (Mitsubishi).

Por lo tanto solo se necesita la máquina virtual adecuada según la plataforma, el archivo .image junto con el .changes y el .sources, estos últimos tres sirven para cualquiera de las plataformas.

Sincronización de cambios en tiempo de desarrollo Squeak incluye una herramienta llamada Monticello. Es un sistema de versionado concurrente.

Monticello trabaja con snapshots, que incluye todos los elementos de un paquete, estos elementos se listan a continuación:

- aquellas categorías del sistema que tienen el nombre del paquete
- aquellas categorías que comienzan con el nombre del paquete y siguiendo con un '-'
- aquellos protocolos que comienzan con '*' y siguen con el nombre en minúsculas del paquete.

Monticello soporta 8 tipos de repositorios, algunos de ellos pueden ser de solo lectura, solo escritura o ambos:

- HTTP

  Consiste en un repositorio de tipo lectura-escritura que el servidor puede configurarse como solo lectura.

- FTP

  Similar al repositorio HTTP, pero utiliza el protocolo FTP.

- GOODS

  Repositorio de lectura-escritura que guarda las versiones en una base de datos GOODS.

- Directorio

  Guarda las versiones en un directorio del sistema de archivos local.

- SMTP

  Usado para el envío de las versiones por mail, cada versión es enviada a la dirección configurada para el repositorio.

- SqueakMap Release

  Repositorio de escritura y lectura para la publicación de releases de paquetes a SqueakMap. Es necesario configurarlo con el nombre de paquete en SqueakMap, las iniciales y la contraseña SM. Cada versión salvada es subida hacia la cuenta SM y registrada como release en SqueakMap.

- SqueakMap Cache

  Repositorio donde se almacenan los archivos en casos que se este instalando un paquete desde SqueakMap. En caso de que no exista este repositorio, se crea la primera ves que se va a usar. Por lo general son archivos de versiones para ser usados por Monticello.

- package-cache

  Repositorio usado por la herramienta Monticello para usar de cache de los paquetes que se cargan en la imagen (archivo *.image). Básicamente se mantiene este repositorio para disminuir las ocasiones de no encontrar disponibles los archivos de versiones.

2.3.2. Etoys

Introducción Es una herramienta de software que se encuentra integrada en las XO y está orientada a fomentar el aprendizaje en los niños.

Es muy usada en la enseñanza y consiste en un entorno de desarrollo visualmente rico que funciona en casi todas las plataformas personales.

Al estar basado en Squeak Smalltalk, hereda todas sus funcionalidades y además de agregar otras nuevas.

Su pantalla inicial es como se muestra en la figura 13

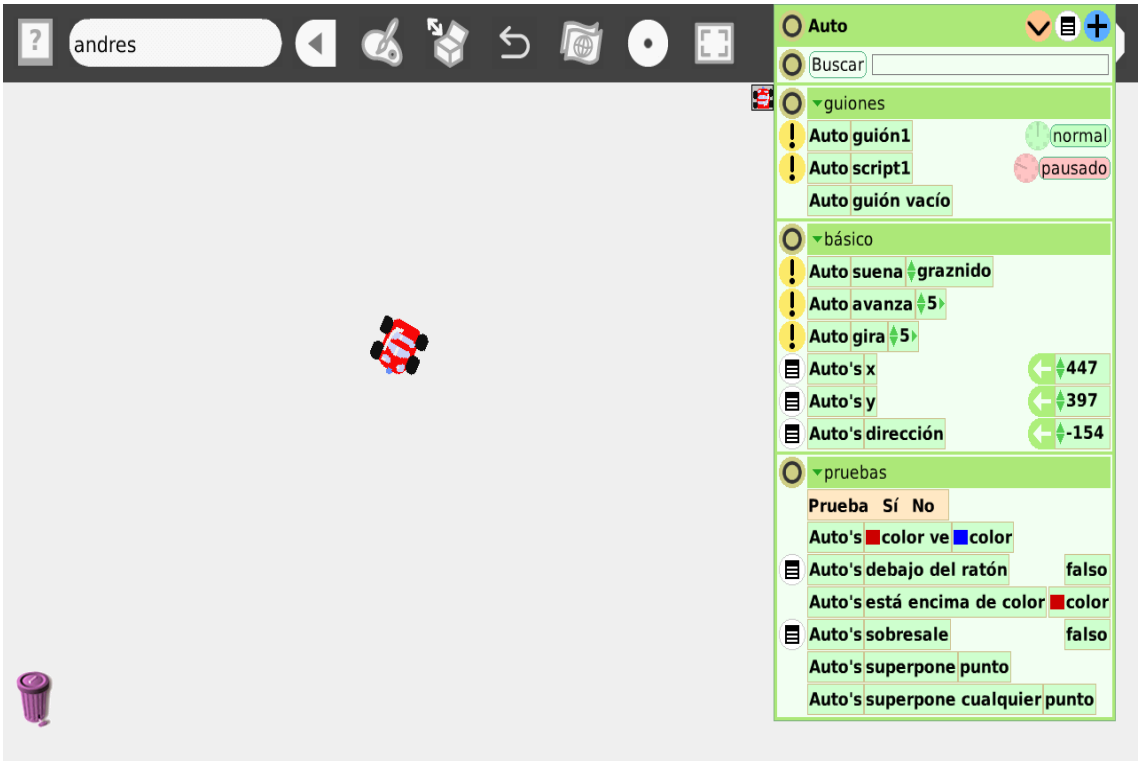


Figura 13: Pantalla principal de Etoys

Donde se distinguen

- Una barra de menú principal
- Un área de trabajo
- Un objeto Morph (imagen de un vehículo)
- Papelera
- Un cuadro de propiedades que aplican sobre el vehículo, denominado Visor.

Funcionalidades ofrecidas ***Barra de menú principal***

Formada por 12 componentes que cumplen una funcionalidad concreta, esta adaptada para un manejo más simple del entorno dejando a disposición algunas de las herramientas que son más usadas por los usuarios.

Con el fin de crear un entorno más amigable, la barra de herramientas brindada por defecto en Etoys es más simple que la de Squeak.

Figura 14: Barra principal

1. Botón de ayuda

   Explica los componentes básicos de Etoys como la barra de navegación y otras propiedades que se comentarán más adelante

2. Cuadro de texto para edición del nombre del proyecto

   Sirve para identificar un proyecto de otros realizados o por realizar.

3. Botón de navegación atrás

   Se utiliza cuando existen varios proyectos abiertos o cuando se profundiza en la estructura de uno, permitiendo la navegación a un estado anterior en la pantalla.

4. Botón de navegación adelante

   Ídem a la navegación atrás pero con orientación opuesta en el avance de los estados en pantalla.

5. Botón de provisiones

   Nos provee de herramientas ya incluidas en la aplicación que pueden resultar muy útiles a la hora de realizar proyectos. Por ejemplo el manejo de múltiples guiones de forma simultanea para la creación de comportamientos concurrentes.

6. Botón de idiomas

   Selector de idioma con el que se puede trabajar Etoys.

7. Botón de Pantalla completa

   Permite maximizar la aplicación al tamaño máximo de la pantalla.

8. Botón abrir de proyectos

   Permite la búsquedas de proyectos y cargarlos en memoria para su manejo

9. Botón guardar de proyectos

   Realiza el guardado a disco del proyecto que se esta editando actualmente.

10. Botón de cierre de la aplicación

    Cierra Etoys, pidiendo confirmación siempre de esta acción.

11. Botón de ocultar la propia barra.

Área de trabajo

Consiste en un espacio en blanco donde se pueden crear objetos que luego se le pueden programar comportamientos. Este área es también conocida como Mundo, y es una suerte de contenedor de otros objetos. El mundo tiene un menú de gestión el cual se puede utilizar para abrir ventanas útiles a la hora de programar. Presionando la combinación de teclas Shift + Alt + W se puede traer a pantalla el menú de la figura 15

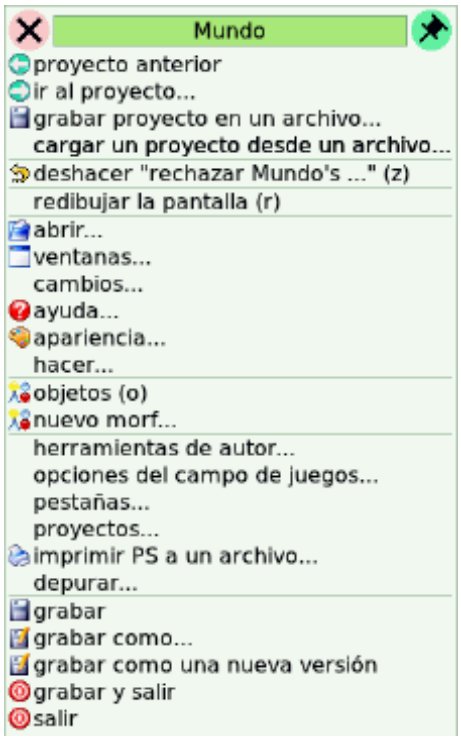


Figura 15: Menú "contextual" del mundo o área de trabajo

Mediante la opción Abrir del menú se pueden obtener las diferentes ventanas que pueden ser visualizar en pantalla, algunas sirven para la ejecución de código de forma dinámica, otra para explorar las clases existente en el entorno de Etoys, y otras herramientas útiles para la programación en Smalltalk que es el lenguaje-entorno en el que se basa Etoys.

Papelera

Sirve como contenedor temporal de aquellos artefactos que fuimos eliminando en todo el proceso de creación de una aplicación. Se puede obtener un objeto borrado y eliminarlo definitivamente.

Morph

En Smalltalk los Objetos visuales son denominados Morph. Existe la posibilidad de poder dibujar uno mediante el punto 4 de la barra principal, como se muestra a continuación en la figura 16

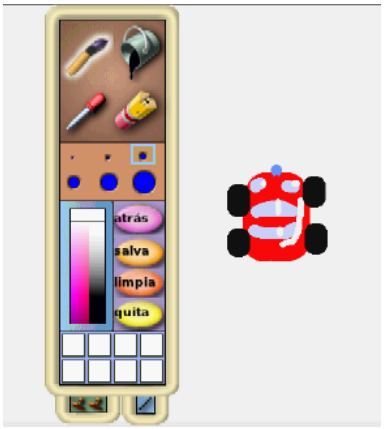


Figura 16: Edición de un Morph

Cada morph tiene habilitado un conjunto de funcionalidades que se activan presionando click derecho sobre el mismo. Estas funcionalidades se agrupan en el denominado Halo.

Figura 17: Halo con sus funciones

Las funcionalidades provistas se aplican sobre el Morph y son las siguientes:

1. Borrarlo

2. Menú contextual

   Permite la edición de algunas propiedades del morph

3. Levantar el objeto

4. Mover el objeto

5. Duplicarlo

6. Colapsarlo

7. Debug

   Muestra las opciones de debug, algunas de ellas son inspeccionar el objeto, abrir visualizador entre otras opciones

8. Visor

   Muestra las propiedades de manera detallada y funciones que se pueden aplicar.

9. Edición visual

   Se invoca el cuadro de edición visto en la figura 16

10. Crea un mosaico

    Este mosaico es un recuadro simple y minimalista que representa al objeto, para poder hacer referencia a el en los scripts.

11. Girarlo

12. Dimensionar el objeto

Visor

Consiste en un cuadro de propiedades que permite la alteración de valores inherentes al morph, como por ejemplo la ubicación en pantalla en base a los eje X e Y. También existen otras funcionalidades más complejas y muy útiles como los guiones, estos son scripts que implementan un comportamiento sobre el objeto.

Un ejemplo de visor es el de la figura 18

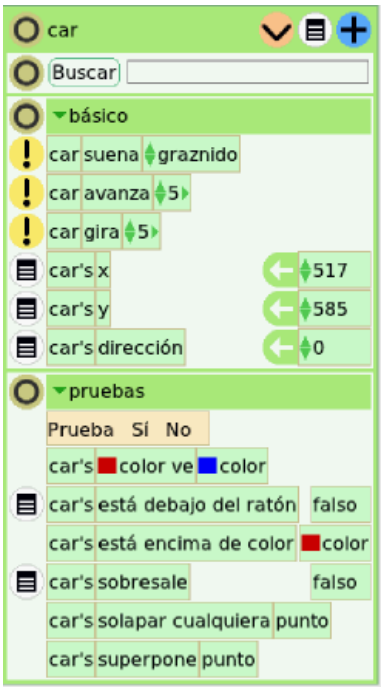


Figura 18: Visor del morph

Los scripts (guiones) son contenedores de otras piezas o bloques alojados en el visor, ejemplos de estas piezas son las clásicas clausulas "if" (Prueba en Etoys) encontrados en otros lenguajes como se ve en la figura 19 junto con los bloques "Car avanza 5", "Car gira 1", "Car rebota silencio".

Figura 19: Ejemplo de bloques en un guión

La programación en Etoys está basada fuertemente en bloques, por tal motivo la anidación de bloque resulta necesario, como se observa en la Figura 19. Aquí el bloque Prueba (resaltado con color naranja pálido) tiene dos bloques dentro. El primero es el encargado de evaluar la condición sobre el objeto Car, que alguna parte de color azul de dicho objeto halla tenido contacto con otro objeto de color gris. Este bloque que evalúa a su vez esta compuesto por 3

bloques, el primero es el objeto en cuestión, el segundo es el bloque condición *color ve* y el tercero es el bloque *color*.

Respaldo de la Comunidad que mantiene y utiliza el software La comunidad de Etoys es bastante variada, algunos de los integrantes de esta comunidad se listan a continuación

- EtoysIllinois

  El objetivo de EtoysIllinois es ayudar a niños a usar Etoys y a alentarlos a expresar sus ideas usando el lenguaje de la computadora.

- USeIT(Using Squeak to Infuse Information Technology)

  Página Squeak Etoys del profesor Prof. Randall Caton que ofrece numerosos proyectos

- Emergent.de

  Proyectos Etoys de nivel avanzado por Markus Gaelli.

- Squeak Alemania

  Una asociación sin fines de lucro de Squeakers/usuarios de Etoys que hablan alemán, enfocada hacia la educación.

- Etoys Brasil

  Sitio en portugués dedicado a difundir Etoys en Brasil.

- Squeakland.jp

  Sitio en japonés creado por la comunidad, para usuarios y programadores de Etoys.

- Niños Super Ciencia de HP

  Sitio creado por un equipo patrocinado por Hewlett-Packard para fomentar el aprendizaje de los niños usando la ciencia.

- Proyecto ALAN-K en la Ciudad de Kyoto

  Sitio en japonés con artículos que describen experiencias y conclusiones de un programa piloto de tres años.

- Squeakapolis

  Sitio mantenido por la comunidad Etoys de España.

- Kusasa

  Nuevo y dinámico sistema de aprendizaje creado por programadores en Sudáfrica.

- OFSET

  Wiki comunitario que promueve el uso de programas libres en educación entre ellos Etoys.

- Waveplace

  Usa Etoys para enseñar a niños del Caribe el uso de medios digitales.

- SqueakLândia

  Una comunidad portuguesa de Etoys.

Restricción en licencias Cuenta con la misma licencia que tiene Squeak, es decir partes del código con licencia MIT y otras parte con licencia Apache

Facilidad de Extensión Dificultad similar a la de Squeak para la extensión de la herramienta.

Curva de aprendizaje de la herramienta Igual a la de Smalltalk ya que es una extensión de este.

Curva de aprendizaje para el código usado por el usuario final (código, u orientado a bloques) Visualmente, la herramienta esta diseñada para que sea usada con fines didácticos en una primera instancia, esto influye directamente en que el uso no implique un alto costo. También el código del usuario final es igual al del Squeak Smalltalk, osea orientado a bloques.

Restricciones del contexto de ejecución Etoys al igual que Smalltalk se ejecuta sobre una máquina virtual

- Windows
- MacOS
- Linux/Unix
- OS/2 Warp

En resumen seria la máquina virtual particular para la plataforma y luego los archivos .image, .changes y el .sources.

Sincronización de cambios en tiempo de desarrollo Igual a los soportados por Squeak Smalltalk

2.3.3. Physical Etoys

Introducción Physical Etoys es un entorno visual como los anteriores que conecta el mundo virtual de la computadora con objetos del mundo real. Con esta herramienta se pueden programar los objetos del mundo real, o poder sensar valores del mundo real y realizar acciones en el mundo virtual.

Consiste en una extensión de Etoys con los agregados necesario para conectarse e interactuar con los elementos reales.

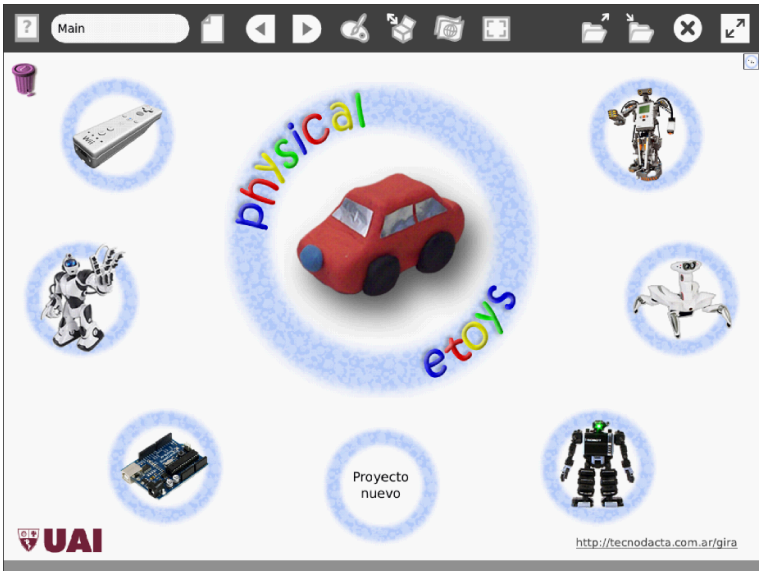


Figura 20: Pantalla principal de Physical Etoys

Funcionalidades ofrecidas Permite la conexión con el hardware listado a continuación:

- Arduino

  Plataforma libre basada en una placa con microcontrolador, que además incluye un IDE propio para facilitar el uso. La placa posee una interfaz USB por la que Physical Etoys utiliza para comunicares.

- I-Sobot

  Uno de los robots humanoides de menor tamaño existentes, capaz de realizar múltiples movimientos complejos. Physical Etoys hace uso del control infrarrojo conectado en el puerto Serial para la comunicación con el hardware

- Roboquad

  Robot de 4 piernas el cual tiene varios modos de operación, dos de los más importantes son el permitir su control de forma remota, o funcionar

totalmente de forma autónoma. La comunicación se lleva a cabo usando un transmisor infrarrojo conectado al puerto Serial.

- RoboSapien

  Robot humanoide que permite su utilización por medio de control remoto, la comunicación es a través de un transmisor infrarrojo sobre el puerto Serial.

- SqueakNxt

  Sistema el cual permite controlar en tiempo real, robots de Lego Mindstorms Nxt usando conexión Bluetooth.

- Wiimote

  Control remoto para la consola de Nintendo Wii.

Respaldo de la Comunidad que mantiene y utiliza el software La comunidad de Physical Etoys es bastante limitada. Es mantenido por el el Grupo de Investigación en Robótica Autónoma (GIRA) del Centro de Altos Estudios en Tecnología informática de Argentina (CAETI) [12].

Facilidad de Extensión Similar a la de Squeak y Etoys debido a que todos pertenecen a la misma familia de aplicaciones.

Restricción en licencias El proyecto Physical Etoys esta desarrollado bajo la licencia del MIT tal como Squeak y Etoys. También puede incluir componentes de software de terceras partes con licencias diferentes a las del MIT.

Curva de aprendizaje de la herramienta Similar a Squeak Smalltalk y Etoys. Se hace uso de las mismas herramientas para el desarrollo del IDE.

Curva de aprendizaje para el código usado por el usuario final (código, u orientado a bloques) Curva relativamente sencilla, ya que como los entornos anteriores esta basado en la codificación orientada a Bloques.

Restricciones del contexto de ejecución En una primera instancia, los requisitos de ejecución son los mismos que Etoys, salvo algunas excepciones.

La ultima versión a la fecha (versión 1.8.1) está desarrollada para que funcione de forma completa bajo el sistema operativo Windows.

Sin embargo la ultima versión para Linux (versión 1.7) tiene soporte únicamente para los proyectos Arduino y LegoNxt.

Sincronización de cambios en tiempo de desarrollo Sigue la misma línea de Squeak Smalltalk, utiliza las funcionalidades ofrecidas por Monticello para el versionado del código en tiempo de desarrollo.

2.3.4. Scratch

Introducción Scratch es una aplicación desarrollada por un grupo de investigadores del MIT especializados en la implementación de software educativo. Los usuarios a los que está destinado son adolescentes, permitiéndoles crear sus propias animaciones, juegos, y otros tipos de aplicaciones con contenido multimedia.

Su implementación esta basada en la versión 2.8 de Squeak Smalltalk, con una interfaz totalmente reformada con el objetivo de que sea más sencillo el uso de esta herramienta en parte por el público objetivo de la misma.

Es una herramienta que permite la creación de aplicaciones utilizando la misma metodología que los entornos anteriores, osea orientada a bloques.

Existen otras aplicaciones que están basadas en Scratch como por ejemplo App Inventor[13]desarrollada en los laboratorios de Google (Google labs). Su objetivo reside en la creación de aplicaciones para el Sistema Operativo Android.

La imagen 21 a continuación muestra la pantalla principal de la aplicación:

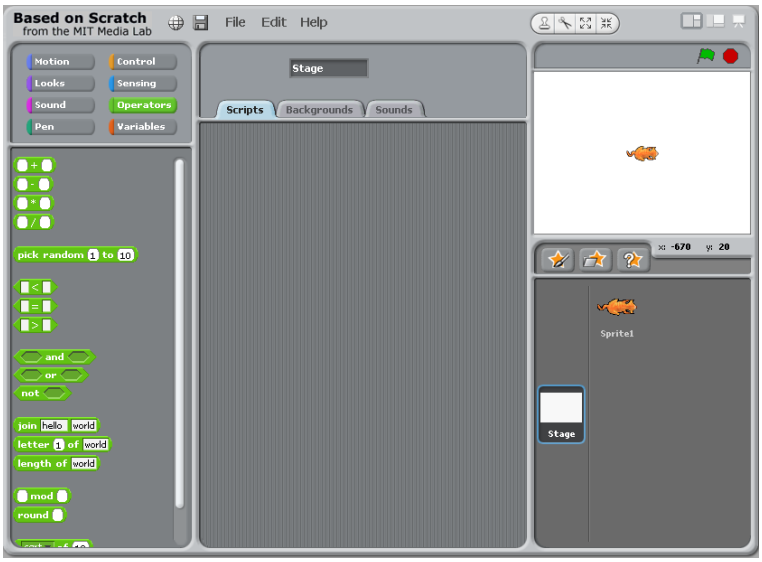


Figura 21: Pantalla inicial de Scratch

La pantalla inicial puede dividirse en varios componentes más sencillos, los más importantes son los siguientes:

- Barra de Menús
- Barra de herramientas de bloques
- Área de trabajo
- Área de presentación
- Área de Objetos

Funcionalidades ofrecidas ***Barra de Menús***

Brinda las funcionalidades necesarias para la creación de proyectos nuevos, así como la recuperación de proyectos ya creados y abrirlos para su modificación. También permite la importación de un proyecto y la exportación de objetos creados (Sprites).

La siguiente imagen 22 muestra la barra de menús

Figura 22: Barra de menú

También tiene la opción de cambio de idioma ya que es una aplicación internacionalizada.

Barra de herramientas de bloques

Esta barra reúne todos los bloques que pueden utilizarse organizados y agrupados por tipo de funcionalidad.

Las categorías por defecto que ya vienen instaladas en la aplicación son las siguientes

- Movimiento

  Bloques de movimiento como por ejemplo, avanzar, girar, entre otros. Permiten la modificación de la posición del objeto sobre el que se esta trabajando. El cambio en la posición se puede apreciar en el área de presentación.

- Apariencia

  Permite cambiar de color, cambiar de imagen, ocultar o mostrar el Objeto sobre el que se esta trabajando. También permite cambiar otras propiedades como el tamaño del mismo.

- Control

  Bloques de herramientas útiles para la implementación del código, implementan primitivas básicas de control como condicionales (bloque if, if else), iteradores (forever, repeat, forever if), etc.

- Sensores

  Bloques comunes usados para el testeo de determinadas situaciones dadas, por ejemplo si un elemento de un color esta tocando otro elemento de otro color, o si un elemento esta a cierta distancia de otro.

- Operadores

  Permiten aplicar operaciones matemáticas como también otro tipo de operaciones, como calcular el largo de una cadena de caracteres.

- Sonidos

  Bloques básicos para la manipulación de sonidos, dan la posibilidad de reproducirlos, cambiar atributos de la reproducción, como también gestionar el volumen del mismo.

- Variables

  Únicamente permiten la creación de variables, y de listas. A partir de la creación de cada uno de estos objetos se agregan bloques extras básicos para la manipulación de variables, como lo son los bloques que asignan, ocultan y muestran las variables.

- Escritura

  Herramientas para creación de dibujos en pantalla, por medio de la utilización de comandos como comenzar a dibujar (Pen down), seleccionar color con el que escribirá el lápiz, y el trazado se realiza a medida que el objeto se mueve utilizando los bloques de movimiento.

Figura 23: Barra de bloques

Área de trabajo

El área de trabajo es donde se ubican las piezas del programa a desarrollar, aquí se organizan los bloques de tal forma de que el resultado final cumpla con lo necesitado. En el siguiente ejemplo 24 la aplicación realiza la creación de un hexágono, donde el largo de cada arista es de 10 pasos.

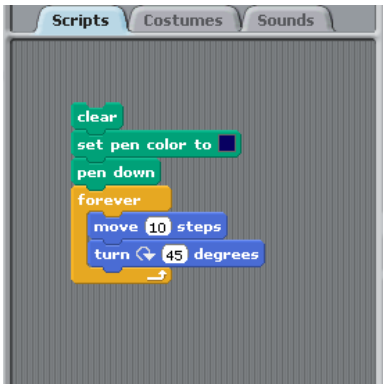


Figura 24: Área de trabajo de Scratch

Área de presentación

Es el área utilizada para visualizar el comportamiento del programa desarrollado, usando el ejemplo anterior en el cual se dibuja un hexágono, el área de presentación luego de la ejecución de unos segundos queda como se muestra en 25, de manera implícita la imagen es la que posee el lápiz y la que se encarga de realizar el trazado de la línea.

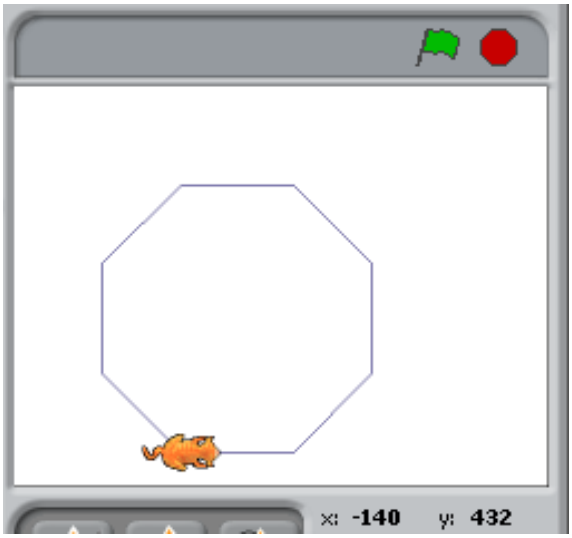


Figura 25: Área de presentación

Área de Objetos

Lugar donde se alojan los objetos sobre los cuales se aplican algunos de los bloques, por ejemplo, los bloques de movimientos aplican sobre un objeto en particular, en la siguiente imagen 26 se pueden ver dos tipos de objetos, el

Stage, y un Sprite que es usado para realizar dibujos como el que se ve en 25.

Figura 26: Área de objetos

Respaldo de la Comunidad que mantiene y utiliza el software La comunidad de Scratch [16] cuenta con una amplia cantidad de miembros registrados, por mes se reciben en promedio cerca de 650000 visitas al sitio web según [30]. A partir de 2009, en promedio se comparten por mes una cifra superior a 40000 proyectos [18].

Uno de los aspectos más interesantes es la participación de adolescentes en la mayoría de los proyectos creados, en total existen más de 240000 proyectos creados por personas de 13 años [19].

También la comunidad esta en constante aumento debido a que en promedio se registran más de 20000 usuarios mensualmente.

Restricción en licencias Scratch tiene dos licencias, una para el código y otra para el software.

***Licencia de software** [22]*

Esta licencia es conocida como la Scratch licence 1.4, que permite sin cargo alguno el derecho a uso, copia, publicación y distribución del software y su documentación respetando las siguientes condiciones:

- El copyright debe estar incluido en cada copia del software.
- Si el software es distribuido, debe mantenerse la siguiente frase ya sea en la página web o el medio en que se distribuya por ejemplo CDs.

  Scratch is developed by the Lifelong Kindergarten group at the MIT Media Lab.

See http://Scratch.mit.edu

Licencia del código [20]

La licencia permite la distribución de trabajo derivado basado en el código fuente de Scratch para uso no comercial sujeto a las siguientes restricciones entre otras más:

- No se debe usar la palabra "Scratch" para referirse al trabajo derivado excepto cuando se haga referencia al origen del código
- No se debe usar el logo de Scratch o la imagen oficial de Scratch en trabajos derivados.
- No se debe implementar la habilidad de subir proyectos a cualquiera de los sitios web de Scratch del MIT.
- Copias y trabajos derivados deben retener el copyright notice y el copyright license además de mantener habilitado el código fuente para trabajos derivados

Facilidad de Extensión Similar a Squeak Smalltalk y Etoys. Se hace uso de las mismas herramientas para el desarrollo del IDE.

Curva de aprendizaje de la herramienta Similar a Squeak Smalltalk y Etoys. Se hace uso de las mismas herramientas para el desarrollo del IDE.

Curva de aprendizaje para el código usado por el usuario final (código, u orientado a bloques) Curva relativamente sencilla, ya que como los entornos anteriores esta basado en la codificación orientada a Bloques.

Restricciones del contexto de ejecución [21]Squeak permite ejecutar en múltiples plataformas, algunas de ellas son las siguientes

- Windows XP
- Windows 2000
- Windows Vista
- Windows 7
- Mac OS X 10.4 o posteriores [24]
- Linux, en sus variadas distribuciones como Debian/Ubuntu, o Fedora. [23]

Sincronización de cambios en tiempo de desarrollo A la fecha no esta funcional la herramienta Monticello en Scratch, la única forma de compartir código es realizando los *FileOut* y *FileIn* descriptos en la sección de Squeak para la exportación de código y la importación del mismo.

2.3.5. Turtle Blocks

Introducción Tortugarte es una de las aplicaciones incluidas en Sugar. Esta inspirada en Logo aunque sigue fuertemente los lineamientos de Scratch en la programación visual para la elaboración de aplicaciones.

Fue escrita por Brian Silverman y actualmente es mantenido por Walter Bender.

Consiste en agrupar las piezas que representan funcionalidades, como operaciones aritméticas, lógicas entre otras, de forma tal que el resultado conformado por la conexión de estas piezas se comporte realizando una tarea especifica.

La versión de Tortugarte que se muestra a continuación es una versión modificada que tiene agregados para poder trabajar con la plataforma Butiá.

La pantalla inicial de la aplicación es la indicada en la figura 27

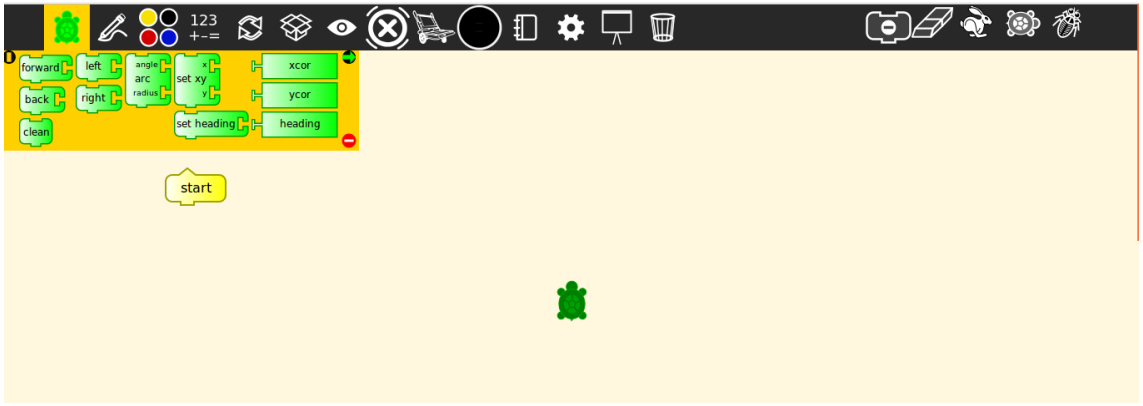


Figura 27: Pantalla principal de Turtle Blocks con el plugin Butiá

Funcionalidades ofrecidas Entre las herramientas que Tortugarte nos brinda para el armado de un programa se encuentran la paleta de bloques, y la barra de ejecución.

Paleta de bloques

Consiste en una barra que brinda las funcionalidades de las barras de bloques. Las barras de bloques son un conjunto de piezas que realizan una determinada funcionalidad. Entre los diferentes tipos ofrecidos por esta paleta están aquellos destinados a funcionar como bloques de control, bloques de movimiento, bloques de dibujado. La extensión realizada para Butiá permite interactuar con dicha plataforma realizando las acciones básicas de movimientos del robot, como también realizar el censado del ambiente ya sea luminosidad, temperatura, en presencia de campo magnético, nivel de carga de la batería, etc.

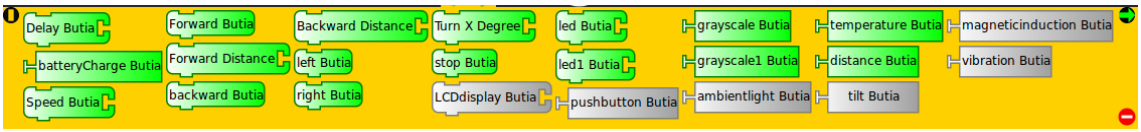


Figura 28: Instancia posible de la paleta de control para Butiá

Barra de ejecución

Consiste en varias funcionalidades enfocadas a la forma en como se ejecuta la aplicación creada. Se detallan a continuación estas funcionalidades:

- Ocultar paleta de bloques
- Limpiado de la ejecución

  Reinicia el estado de la pantalla al instante antes de haber ejecutado. Reubica la tortuga y limpia lo dibujado por la misma.
- Ejecución rápida

  Ejecuta todo el código de forma instantánea
- Ejecución lenta

  Ejecuta todo el código de forma progresiva dando la posibilidad de ver que se esta ejecutando en un instante determinado
- Ejecución en modo debug
- Detener ejecución

  Detiene la ejecución de la aplicación sin importar el modo en que se encuentre.

Respaldo de la Comunidad que mantiene y utiliza el software La comunidad de Tortugarte esta conformada por usuarios de varios países, aunque mucha de la información sobre esta aplicación es ofrecida en sus sitios oficiales.

Algunos de los sitios web que abordan temáticas directas sobre Tortugarte son los siguientes

- http://tonyforster.blogspot.com (visitado en 12/12/2012)

  Blog donde se documentan distintos ejemplos hechos en Turtle Blocks
- http://wiki.sugarlabs.org/go/Activities/Turtle_Art (visitado en 12/12/2012)

  Wiki oficial de la aplicación
- http://wiki.laptop.org/go/Turtle_Art (visitado en 12/12/2012)

  Página que divulga información útil para aquellos que quieran iniciarse en el uso de la herramienta.

- https://sites.google.com/site/solymar1fisica/fisica-con-xo-investigacion- (visitado en 12/12/2012)

  Sitio web del liceo de Solymar de Uruguay que hacen uso de la xo utilizándola como instrumento de medida y movimiento.

Restricción en licencias Tortugarte hace uso de una única licencia descripta anteriormente que es la licencia MIT [17].

Facilidad de Extensión Tortugarte brinda la posibilidad de crear plugins [1]. De esta forma no es necesario modificar el código fuente de la herramienta, sino que basta con desarrollar un plugin y es la propia herramienta la encargada de cargarlo al momento de iniciar la aplicación.

Para desarrollar un plugin basta con seguir determinados lineamientos:

- Los plugins se deben alojar en un directorio denominado plugins.
- El nombre del archivo debe finalizar de la forma _plugin.py
- La clase base debe ser descendiente de la clase Plugin definida plugin.py.
- La clase debe llamarse como el archivo pero con la primer letra en mayúsculas

También, a modo de extensión, permite insertar código python dentro de nuestras aplicaciones.

Curva de aprendizaje de la herramienta La herramienta esta enteramente desarrollada en phyton [29], que es un lenguaje muy popular y que puede ser muy simple orientado a objetos. Si bien esto no implica directamente que sea sencillo aprender como esta creada la herramienta es un buen punto de partida.

Además la estructura del código de la aplicación esta formada en su conjunto por varios archivos, a diferencia de Squeak Smalltalk que todo su código se encuentra en único archivo. De esta forma resulta más sencillo determinar los diferentes componentes que forman a la herramienta

Curva de aprendizaje para el código usado por el usuario final (código, u orientado a bloques) La herramienta tiene una curva de aprendizaje leve, esto es así pues fue desarrollada para que sea lo más intuitiva posible debido a que se enfocó a un público de menor edad.

Al ser enfocada a bloques no se tiene problemas de escrituras en el código y todas las opciones que se pueden utilizar están a la vista.

Restricciones del contexto de ejecución Tortugarte puede ejecutarse sobre la plataforma GNU Linux. Para realizar pruebas es muy útil la utilización del emulador de Sugar, de esta forma se puede aproximar al entorno que se puede tener en una XO.

Sincronización de cambios en tiempo de desarrollo Para la sincronización de cambios en el código se puede utilizar cualquiera de las herramientas de versionados que existe en el mercado, para nombrar algunas podrían ser las listadas a continuación:

- cvs

  Concurrent Versions System, usado para compartir código de forma concurrente, esta orientado a cliente servidor.

- svn

  Sistema de control de versiones, creado para reemplazar al cvs. Posee ciertas funcionalidades que son imposibles de realizar con cvs.

- git

  Sistema de control de versiones inicialmente desarrollado por Linus Torvalds enfocado a la eficiencia. Consiste en un repositorio de versiones distribuidos

2.4. Decisiones tomadas en base al estudio previo

2.4.1. Comparación de las Arquitecturas

La arquitectura de Campos de Potencial se destaca por ofrecer una solución sencilla a un problema complejo, el movimiento en busca de un objetivo. Este es uno de los problemas más comunes a enfrentar al abordar la implementación de un software de control de un robot. Ésta característica es esencial a la hora de la enseñanza, ya que los estudiantes pueden tomar sus primeros pasos resolviendo un problema común de una forma rápida, sencilla y eficaz.

Por otro lado, si se observa Teoría de los Esquemas, encontramos una arquitectura que puede resolver sencillamente distintos comportamientos complejos mediante la fragmentación en comportamientos más sencillos.

Finalmente, con Subsumption, se pueden implementar sencillamente el resto de las arquitecturas. De esta forma hereda las ventajas de cada una. Es más intuitiva que Campos de Potencial, ya que para implementar comportamientos complejos en Subsumption se dividen en comportamientos más sencillos simplificando el problema. También resuelve el problema de coordinación entre comportamientos de Teoría de Esquemas. En ésta, dicha implementación queda en manos del programador, que carecen de experiencia en el área ya que son estudiantes dando sus primeros pasos.

Finalmente, considerando los argumentos aquí expuestos, concluimos que la arquitectura más apropiada para abordar este problema es Subsumption.

2.4.2. Comparación de Entornos de Desarrollo

Para la decisión de que plataforma utilizar como punto inicial se baso en cada uno de los aspectos descriptos en cada herramienta

- Funcionalidades ofrecidas

  Squeak Smalltalk ofrece un entorno absolutamente completo para el desarrollo de una aplicación. Por lo tanto Etoys, Physical Etoys y Scratch tienen en este aspecto las mismas cualidades. Además brindan la posibilidad de instalar agregados ya creados que se encuentran en los repositorios de Squeak por ejemplo.

- Respaldo de la Comunidad que mantiene y utiliza el software

  La comunidad de Physical Etoys es bastante limitada debido a que no es un entorno muy difundido a nivel global. Sin embargo Squeak, Etoys y Scratch poseen comunidades bastante más amplias y activas, Tortugarte por su parte quizás es más moderna con una comunidad menor a las de las anteriores pero muestra signos de crecimiento.

- Facilidad de Extensión

  Squeak, Etoys, Physical Etoys y Scratch cuenta con las mismas funcionalidades y posibilidades para la extensión de los mismos, Tortugarte posee un mecanismo de plugin bastante interesante y todos brindan en su totalidad el código.

- Restricción en licencias

  Squeak, Etoys, y Tortugarte son las aplicaciones que brindan menores restricciones en licencias, Scratch por su parte posee dos licencias una para la aplicación, y otra para el código, siendo este último bastante restrictivo en lo referente a algunas de las funcionalidades que si están presentes en la aplicación. Physical Etoys además de depender de las licencias que de Etoys, y Smalltalk, depende de licencias de terceros.

- Curva de aprendizaje de la herramienta

  La curva de aprendizaje de las aplicaciones Squeak y todas las que lo extienden tienen una dificultad mayor que Tortugarte, sin embargo el potencial brindado por este tipo de aplicaciones es mayor.

- Restricciones del contexto de ejecución

  Si bien Smalltalk esta en varias plataforma, Etoys es la aplicación que lo extiende que se encuentra en casi la misma variedad de plataformas. Physical Etoys tiene ciertas restricciones, ya que para poder usar algunas de sus funcionalidades es necesario que se este ejecutando sobre una plataforma Windows. Tortugarte solo corre en entornos Linux.

- Sincronización de cambios en tiempo de desarrollo

  Tortugarte es la aplicación que brinda la opción de utilizar muchos de los diferentes mecanismos de versionado y sincronización de cambios como los son svn, cvs, o git entre otros. Squeak y Etoys poseen un mecanismo bastante particular de sincronización, sin embargo Scratch no posee ningún método sencillo y viable para versionar y compartir código mientras se esta en tiempo de desarrollo.

Etoys pertenece a las familias de aplicaciones con un potencial mayor, además esta diseñado para ser usado por personas de poca edad, permitiendo compartir código de manera ágil en tiempo de desarrollo, posee pocas restricciones a nivel de licencias y es multiplataforma.

Por todo esto se decidió a utilizar Etoys como base para implementar el desarrollo del nuevo IDE.

Parte II

Desarrollo del Sistema

3. Requerimientos identificados

En esta sección se detallan los principales requerimientos relevados en el documento de Especificación de Requerimientos[3] luego de la realización del estudio de Estado del Arte. Dentro de este se realiza un estudio de los usuarios y posteriormente se detallan los requerimientos divididos en dos categorías: Funcionales y no Funcionales. Finalmente se detallan los principales requerimientos de documentación que deben acompañar al entorno.

3.1. Usuarios

El conjunto de usuarios será dividido, según sus características en dos subconjuntos principales, estudiantes y docentes. Los estudiantes ya tuvieron sus primeras experiencias de desarrollo con TurtleBots, por lo que se puede asumir que conocen al robot y los principios básicos de la programación. Suelen desarrollar sus aplicaciones en grupos, pensando en conjunto, desde el armado del robot, hasta la programación del comportamiento que resuelve la problemática.

Por otro lado, los docentes guiarán a los estudiantes pero en general la experiencia de los mismos no es mayor a la de los estudiantes. Esto es debido a que también dieron sus primeros pasos tanto en programación, como en robótica, con TurtleBots.

Ambos tipos de usuarios utilizarán el sistema de la misma forma sin existir vistas o privilegios distintos.

Si bien pueden existir usuarios que sobrepasen los conocimientos descriptos a continuación, podemos asumir que la mayoría de los usuarios cumplen con estas características:

- Poseen conocimientos básicos de la programación y de la robótica adquiridos a través de la experiencia con la herramienta TurtleBots.
- Conocen el robot, así como también, la forma de trabajar con los módulos conectados al mismo; averiguar su propósito, funcionalidades brindadas y el tipo de valores que reciben u ofrecen.
- No poseen experiencia ni conocimiento en la estructuración de aplicaciones a través de una cierta arquitectura.
- No son usuarios informáticos avanzados, por lo tanto no tienen conocimientos básicos de comunicación entre aplicaciones.

3.2. No funcionales

Tiempo de Reacción
Las ordenes ejecutadas en el sistema deben llegar al robot en un tiempo menor a 1 segundo.

Facilidad de Distribución
La aplicación debe ser distribuida al menos en formato de archivo .xo para facilitar su instalación en las maquinas XO.

Costo
Debe ser sin costo para que pueda ser utilizado por todos los estudiantes.

La escalabilidad no debe afectar la accesibilidad
Los componentes de la aplicación que muestran conjuntos de objetos deben permitir que todos sean accesibles sin importar la cantidad de los mismos. Es decir, que por ejemplo si existiera una cantidad de módulos o capas que no entre en la pantalla, estos deben de quedar accesibles a través de algún mecanismo, como puede ser el scroll visual.

Concurrencia
El sistema debe ser capaz de ejecutar paralelamente las distintas capas que ensamblan el comportamiento robótico en la arquitectura Subsumption.

Mantenibilidad respecto a Etoys
El código debe ser fácilmente adaptable ante mejoras y cambios de Etoys.

Mantenibilidad respecto a Módulos
Nuevos módulos, con nuevas funciones no deben afectar la programación del proyecto, ya que para esto se utilizará la metadata ofrecida por el bobot-server.

Internacionalización y lenguaje
La información de lectura en pantalla debe ser internacionalizable.

Acceso al robot a través de la red
La aplicación debe poder manipular cualquier robot Butiá que se encuentre accesible través de una red tcp/ip.

3.3. Funcionales

3.3.1. Entorno

Entorno de ejecución
Debe funcionar de forma aceptable en las maquinas XO distribuidas en el plan Ceibal.

Sistema a Extender
El sistema debe implementarse como una extensión de Etoys para tomar todas las ventajas brindadas en materia de metodología de programación y concurrencia.

Barra de navegación
Se debe mantener la barra de herramientas de navegación entre los proyectos.

Calibración del robot
La aplicación debe mostrar una interfaz que permita la calibración de los módulos del Butiá.

3.3.2. Proyectos

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

Pantalla del proyecto
La pantalla de proyecto debe permitir seleccionar un bobot, elegir los módulos a utilizar y gestionar las capas.

Persistencia de proyectos
Debe poder almacenarse un proyecto con el código generado en cada una de sus capas.

3.3.3. Bobots-Servers

Gestión de bobot-server
Deben presentarse herramientas para la administración de bobots-server: creación, y configuraciones de IP y puerto.

Información del bobot-server
De cada bobot-server asociable a un proyecto debe conocerse al menos la dirección IP del equipo en el que corre y el puerto.

3.3.4. Capas

Gestión de capas

Dentro de la gestión de capas se consideran 4 funciones fundamentales:

1. Construcción

   El sistema debe permitir la creación y eliminación de capas.

2. Ordenamiento

   Debe permitir reordenar las capas de manera sencilla.

3. Habilitación

   Las capas deben poder habilitarse y des-habilitarse, de forma tal que las capas deshabilitadas son ignoradas a la hora de ejecutar el comportamiento.

4. Nombre

   Las capas deben poseer un nombre y deben poder ser renombradas de forma sencilla.

Acceso a Capas

Debe permitirse navegar entre capas de forma sencilla. Para esto, debe de poder listar las capas del sistema con el fin de tener una visión global de la aplicación para poder navegar de manera ágil y sencilla a las capas creadas por el usuario.

3.3.5. Módulos

Gestión de Módulos

Debe poderse eliminar aquellos módulos que no estén activos, así como también forzar la verificación de su disponibilidad.

Verificación de disponibilidad de módulos

Cada proyecto debe poder tener un bobot-server asociado, este se utilizará, además de la ejecución, para verificar la disponibilidad de los módulos al abrir el proyecto, en caso que alguno no lo esté, se marca como no conectado.

3.4. Requerimientos de documentación

Manual de usuario

Debe proveerse un manual de usuario orientado a los estudiantes. En el mismo debe explicarse como y por qué calibrar un robot. También como crear un proyecto y una explicación simple del funcionamiento de Subsumption.

Manual de instalación

Se incluirá un manual de instalación que explicará paso a paso las distintas alternativas de instalación del sistema.

Página web

Se debe implementar una página web en forma de wiki donde se irán actualizando toda la información relacionada con la elaboración del sistema.

4. Prototipos

En esta sección pretende registrar, al igual que en el documento de Prototipos[4], los distintos prototipos realizados con el fin de minimizar los riesgos de las decisiones tomadas respecto al entorno. Estos se listan detallando objetivos buscados, motivaciones que llevaron a realizar cada uno, el alcance que posee, sobre que supuestos se trabaja así como los resultados obtenidos.

4.1. Comunicación por sockets con servidor tonto

Objetivos:

Estudiar la viabilidad del entorno de comunicarse por intermedio de sockets con un servidor que exponga una interfaz en texto plano.

Motivación:

La comunicación con el robot Butiá se realiza a través del servidor bobot-server. Dicho servidor expone una interfaz de comunicación en texto plano a través de sockets utilizando un protocolo definido. Este prototipo busca minimizar los riesgos de la elección del entorno de desarrollo detectando la capacidad del mismo de establecer dicha comunicación, en etapas tempranas del proyecto.

Alcance:

Se desarrollará un prototipo que permita al usuario comunicarse con un servidor que exponga una interfaz de texto plano vía sockets.

Supuestos:

Se utilizará el entorno Etoys para la implementación del prototipo. Se realizará sobre el entorno gráfico de Sugar sobre un GNU Linux Fedora (versión XO).

Resultados:

Se obtuvo de forma exitosa un prototipo que permite la comunicación a través de sockets con un servidor tonto.

4.2. Interacción con servidor del Butiá

Objetivos:

Crear un prototipo que permita la comunicación con el servidor que se comunica con el robot Butiá. Este prototipo debe poder listar los distintos módulos conectados, levantar como metadata las operaciones expuestas por los mismos y ejecutarlas.

La consulta de las operaciones expuestas se realiza usando las primitivas:

- LIST

  Lista los módulos actualmente disponibles en la placa Arduino.

- DESCRIBE moduleName

  Describe las operaciones implementadas por un componente específico.

Motivación:

Para interactuar con la placa de entrada/salida que comanda el robot Butiá, existe un software llamado bobot-server. El mismo expone una interfaz de comunicación a través de sockets en texto plano con los módulos conectados y sus funciones, así como también la capacidad de ejecutarlas. El prototipo busca lograr una comunicación con el robot Butiá a través del servidor y utilizando el entorno Etoys. Esta se utilizará en un futuro para poder comandar el robot.

Alcance:

Se desarrollará un prototipo que exponga de forma sencilla todas las funcionalidades del bobot-server. Para esto se implementará una metadata que represente las funciones de dichos módulos.

Supuestos:

Se encuentra corriendo en el equipo un servidor que se conecta con la placa de entrada/salida y expone una interfaz de comunicación. Para su desarrollo se utilizará lo implementado en el prototipo 1.

Resultados:

Se obtuvo un prototipo que expone de forma sencilla las funcionalidades del bobot-server y brinda una metadata para la interacción con el mismo. Este prototipo permite listar los módulos existentes así como también ejecutar sus funcionalidades. Se observó la incapacidad de distinguir a simple vista los sensores de los actuadores.

4.3. Lectura y escritura del filesystem

Objetivos:

Construir un prototipo que permita escribir y leer archivos del filesystem para poder almacenar y consultar propiedades de configuración del entorno.

Motivación:

Familiarizarse con el entorno de desarrollo así como también con el lenguaje. Minimizar los riesgos de la elección del entorno de desarrollo. Poder almacenar configuraciones particulares de la extensión del entorno de una forma aislada y poco invasiva al entorno original.

Alcance:

La generación de un prototipo reutilizable, con una interfaz sencilla, que permita, de forma estable almacenar y consultar propiedades de un archivo en el filesystem. Estas pueden estar categorizadas y pueden a su vez tener sub atributos.

Supuestos:

Se utilizará el entorno Etoys para la implementación del prototipo. Se realizará sobre el entorno gráfico de Sugar sobre un GNU/Linux OLPC OS 13.1.0 para XO-1.5 (build 30)

Resultados:

Serialización de objetos: Se consiguió serializar objetos dentro de un archivo binario para luego levantarlos nuevamente. La desventaja de este procedimiento es la incapacidad de manipular el archivo desde fuera del entorno gráfico.

Almacenamiento en archivo XML: Se construyó un prototipo que permite consultar y almacenar propiedades categorizadas y con atributos en un archivo XML. Además, se pueden consultar las propiedades a través de una interfaz sencilla que filtra por categoría y atributo.

4.4. Generación de componentes nuestros de interfaz - Creación de ventanas de un componente

Objetivos:

Generar componentes de interfaz que representen las distintas capas de Sumsumption. Dentro de cada uno de estos se podrá desarrollar el comportamiento de dicha capa.

Motivación:

Poder implementar el concepto de modularización incluido en sumsumption gráficamente. Además, los códigos de programación de comportamientos suelen ser demasiado extensos para ser representados con programación gráfica, de aquí la posibilidad de visualizar cada capa en una ventana diferente.

Alcance:

Desarrollar ventanas de código que representen cada una de las capas, para desarrollar en una pantalla distinta cada una.

Supuestos:

Se utilizará el entorno de desarrollo de Etoys.
Existen proyectos de Etoys que se abren sobre una nueva ventana.
Los proyectos de Etoys pueden tener sub-proyectos.

Resultados:

Se desarrollaron proyectos propios que representan cada capa. Cada proyecto es un sub-proyecto del proyecto principal que representa una capa. En el proyecto principal se muestra cada capa utilizando la previsualización por defecto de los sub-proyectos. Al presionar sobre la misma se ingresa al sub-proyecto para desarrollar el comportamiento sobre una nueva ventana.

4.5. Generación de componentes nuestros de interfaz - Toolbar con objetos propios

Objetivos:

Desarrollar barras de herramientas que contengan objetos propios, así como botones e imágenes de los distintos módulos.

Motivación:

Para mejorar la usabilidad del sistema se intentará crear una barra de herramientas que posea las distintas utilidades proporcionadas por el nuevo entorno de desarrollo.

Alcance:

Programar una barra de herramientas que sea expuesta en los proyectos del nuevo sistema.

Supuestos:

Se posee una clase propia de proyectos que será instanciada por cada proyecto que se elabore en el nuevo IDE.

Esta clase de proyecto hereda todas las funcionalidades de los proyectos Etoys.

Resultados:

Se desarrolló una barra de herramientas que se instancia con cada proyecto creado. Esta barra de herramientas posee la capacidad de contener objetos Morph, clase de la cual heredan todos los objetos gráficos en Etoys.

4.6. Generación de componentes nuestros de interfaz - Creación de objetos que representen hilos

Objetivos:

Cada capa representa un comportamiento que corre en un hilo independiente, se busca la representación de dichos hilos en un objeto que permita su programación.

Motivación:

Desarrollar componentes que permitan la programación gráfica de los comportamientos dentro de cada capa.

Alcance:

Creación de un componente gráfico incluido en los proyectos que representan capas, que permita la programación gráfica de un componente.

Supuestos:

Se poseen proyectos que representan una capa dentro de la arquitectura de sumsumption. Dichos proyectos poseen todas las propiedades de los proyectos Etoys.

Resultados:

Se instancia un objeto thread de Etoys con su correspondiente morph para cada sub-proyecto capa que se cree. Dicho thread queda asociado al proyecto capa como el comportamiento de la capa.

4.7. Testeo de guardado de proyectos generados y exportación e importación de secciones de código

Objetivos:

Poder almacenar los proyectos generados, así como también exportar secciones de código.

Motivación:

Los proyectos realizados poseen a su vez sub-proyectos que representan cada una de las capas. El prototipo pretende testear la capacidad del entorno de almacenar los proyectos, así como sus capas y poder levantar capas elaboradas en otros proyectos. Con esto se busca que los alumnos puedan compartir el código generado motivando el trabajo en equipo y buenas técnicas de programación modularizada.

Alcance:

Se pretende crear proyectos que posean a su vez sub-proyectos. Al almacenar el proyecto principal deben almacenarse a su vez los sub-proyectos. Además, se deben poder almacenar cada sub-proyecto independientemente y poder importarlos dentro de proyectos que no sea el original donde fue desarrollado.

Supuestos:

Cada pantalla de Etoys es una instancia de la clase Project del framework Etoys.

Para el caso del prototipo, los proyectos elaborados son subclase de la clase Project de Etoys.

Los proyectos internos a los anteriores pertenecen a una clase distinta pero igualmente subclase de Project.

Resultados:

Se elaboraron proyectos con sub-proyectos y pueden ser almacenados y recuperados, junto con sus atributos dentro del filesystem.

En el desarrollo de la aplicación eButiá se utilizará un proyecto para almacenar en orden todas las capas que conforman la aplicación. Además cada una de estas capas se representa en un proyecto interno por separado. Se dejará para un desarrollo posterior el guardado y recuperación de capas independientemente del proyecto que las almacena.

4.8. Generación de componentes nuestros de interfaz - Creación de objetos programables que representen sensores y actuadores

Objetivos:

Desarrollar componentes que permitan acceder a los sensores y actuadores a través de la programación gráfica.

Motivación:

Etoys posee la cualidad de ser un entorno de desarrollo mediante programación con bloques. Para continuar con el mismo método de desarrollo hay que desarrollar componentes gráficos que representen el código para acceder a los sensores y actuadores.

Alcance:

Creación de componentes gráficos que permitan el acceso a sensores y actuadores mediante la programación gráfica. Estos componentes deben tener la capacidad de retornar valores y de recibir parámetros. Estos componentes deben ser creados dinámicamente a partir de metadata descriptiva de los distintos módulos y sus funciones.

Supuestos:

Etoys es un ambiente de programación que permite la programación en base a bloques de código.

Se posee una metadata que representa los distintos módulos y funcionalidades de los mismos.

Resultados:

Se construyeron bloques que representan el código de acceso a los sensores y actuadores. Estos bloques son creados dinámicamente a través de la metadata desarrollada en un prototipo anterior. Se realizó una verificación de lo programado con un robot Butiá obteniendo con éxito la comunicación y la reacción esperada en el robot.

Un ejemplo de los bloques que se generaron se puede ver en la figura 29

Figura 29: Visor con las operaciones del modulo de motores

4.9. Pruebas de programación concurrente

Objetivos:

Estudiar la capacidad del entorno de desarrollar programación concurrente.

Motivación:

La eficiencia de la arquitectura sumsumption se basa en la ejecución paralela de las distintas capas.

Alcance:

Lograr ejecutar múltiples threads, componentes gráficos que representan las distintas porciones de código, paralelamente.

Supuestos:

Etoys contiene componentes threads que almacenan porciones de código desarrollado gráficamente a través de bloques.

Resultados:

Se ha conseguido con éxito la ejecución concurrente de porciones de código y sincronización a través de semáforos entre los mismos.

4.10. Conclusiones

Se han identificado distintos riesgos sobre la elección del entorno, a partir de los requerimientos relevados. En base a estos, se plantearon distintos prototipos con el fin de verificar la viabilidad en la implementación en Etoys. Luego de haberlos realizado todos exitosamente, se concluye que el camino elegido al seleccionar como entorno a extender Etoys, no posee riesgos cuya gravedad impida la implementación del sistema.

5. Arquitectura del sistema

En esta sección se describe la arquitectura de la extensión implementada. Se detallan componentes, responsabilidades de los mismos e interacción entre estos.

5.1. Estructura general y comunicación entre componentes

El sistema está estructurado de forma que tiene 4 componentes principales:

1. **Capa de Presentación**

   Encargada de mostrar los objetos en la pantalla con los cuales interactúa el usuario

2. **Capa de Negocio**

   Responsable de que los pedidos de los usuarios se lleven a cabo. Por ejemplo, abrir proyectos, calibrar un Bobot, entre otros.

3. **Capa de Comunicación**

   Componente que se comunica con el Bobot.

4. **Capa de Persistencia**

   Brinda las funcionalidades para guardado en disco de información.

Estos componentes interactúan entre si siguiendo el diagrama de componentes encontrado en la siguiente figura:

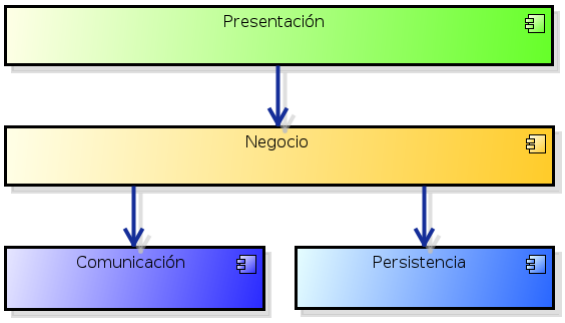


Figura 30: Diagrama de Componentes de la Arquitectura

5.2. Componentes del sistema

En esta sección se describen las 4 componentes del sistema. De cada una se detallan sus responsabilidades e interfaces.

5.2.1. Presentación

La capa de presentación fue estructurada siguiendo el patrón de diseño Model View Controller (MVC) [15]. Este define 3 componentes principales:

- **Modelo:** encargado de almacenar y obtener los datos, y presentarlos de modo que su visualización en pantalla se realice de forma sencilla.
- **Vista:** encargada de mostrar la información en pantalla.
- **Controlador:** maneja los eventos de cambios realizados por el usuario y le comunica al modelo.

Estos componentes interactúan entre sí como muestra la siguiente figura:

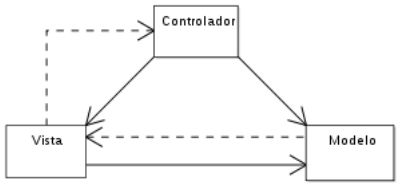


Figura 31: Patrón MVC

Para la aplicación, se utilizará este patrón para modelar cada una de las pantallas que componen al sistema. Con este fin se definió el siguiente conjunto de componentes:

- **Modelo:** información contenida dentro del proyecto y con formato fácilmente renderizable por la vista.
- **Vista:** es un morph que contiene a su vez distintos submorphs conformando la pantalla. Esta se encarga de mostrar correctamente la información proporcionada así como también la posibilidad de modificarla.
- **Controlador:** el proyecto que representa la pantalla va a oficiar de controlador dentro de este modelo, escuchando los eventos del usuario en la interfaz.

De esta forma, el diagrama de interacción queda como lo indica la imagen 32.

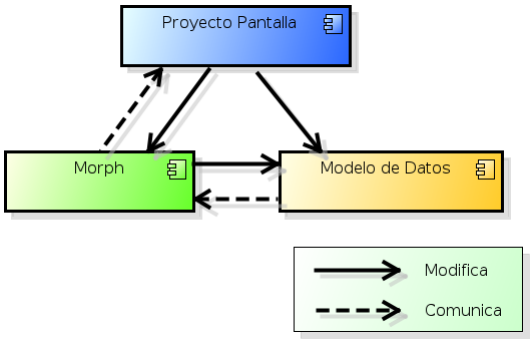


Figura 32: Patrón MVC aplicado a la presentación.

Estructura general de las Pantallas

Cada una de las pantallas que se encuentran en el sistema siguen las lineas de estructuración como se definió previamente, es decir un Proyecto, un Morph global para el proyecto y el modelos de datos. Como norma general, todos los componentes visuales están contenidos dentro del Morph global.

Proyectos: Cada proyecto cumple el rol de controlador de pantalla, por lo tanto es el responsable de encapsular la lógica de la propia pantalla, cediendo únicamente la responsabilidad de mostrar visualmente la información, a los componentes visuales contenidos en el Morph global.

Todos los proyectos contenidos en el sistema son extensiones de un proyecto base del sistema eButiá, que a su vez es una extensión de un proyecto proporcionado por el propio Etoys.

La creación de un proyecto por pantalla fue el resultado obtenido del estudio que se compone de los siguientes puntos:

- Componente ideal para encapsular funciones como variables, facilitando su modificación sin afectar al resto de los proyectos
- Permite una serialización completa del propio proyecto. Es decir, que permite guardar en disco rígido una "imagen" binaria de métodos existentes y las variables con sus valores definidos en el proyecto, así como también, el estado visual del mismo.
- Además Etoys incluye un sistema de navegación que permite navegar de un proyecto a otro de forma sencilla. Facilitando su extensión para futuras modificaciones que pudieran llegar a ser necesarias.

La jerarquía de proyectos se puede apreciar en la siguiente imagen,

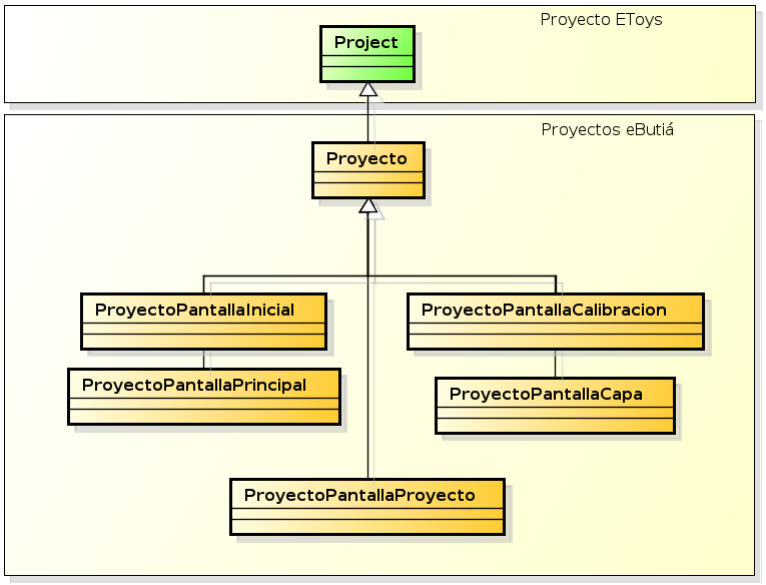


Figura 33: Jerarquías de proyectos del sistema.

Cada proyecto tiene asociado su propio Morph global que despliega al usuario la información correspondiente a la pantalla.

Morphs Globales: Los morph globales son los objetos visuales de cada proyecto que contienen dentro todos los restantes componentes visuales que renderizan la información. Al igual que con los proyectos, se tiene una jerarquías de morphs de proyectos.

eButiá tiene un ProyectoMorph, que es una clase base del cual extienden los demás. A su vez este extiende un morph proporcionado por EToys denominado PasteUpMorph.

Se utilizó PasteUpMorph como la base de todo debido a las funcionalidades que proporciona por defecto, algunas se numeran a continuación,

- Permite el pintado de nuevos objetos,
- rellenados de color en forma de gradientes,
- customización avanzada del background, y
- soporte para pestañas, entre otras.

Los Morphs globales que se encuentran en el sistema se muestran a continuación,

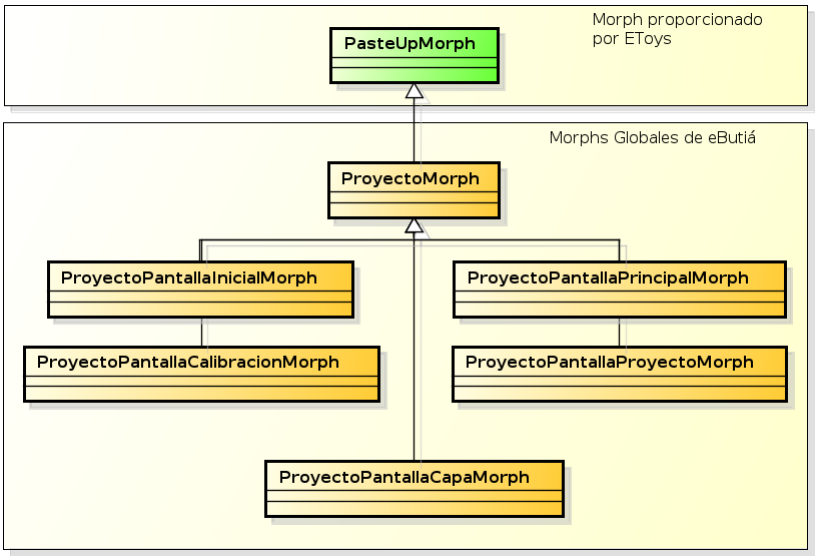


Figura 34: Morphs globales del sistema.

Las pantallas están desarrolladas recurriendo a componentes visuales reutilizables de forma que estos componentes se abstraen de los datos en cierta medida.

Entre estos tipos de componentes se encuentran los "Selectores", estos son componentes que permiten realizar Scroll sobre su contenido por lo que resultan muy útiles a la hora de mostrar listados y colecciones de objetos en pantalla.

Su estructura esta representada en la figura 35, el Selector esta representado por *SelectorPanel*, y asociado a una colección de datos (dataprovider). También está asociado a una colección de otro tipo de objetos llamados ItemRenderer, a un SelectorCommandBar que cumple el rol de barra de comandos.

El ItemRenderer es el responsable de la visualización de un único elemento del listado.

De esta forma la visualización del listado es el conjunto de visualizaciones generada por cada ItemRenderer de cada objeto del listado.

El patrón de diseño utilizado para el dibujado de cada uno de los objetos del listado es el Strategy, donde cada ItemRenderer es la estrategia usada para la renderización en pantalla de cada uno de esos objetos.

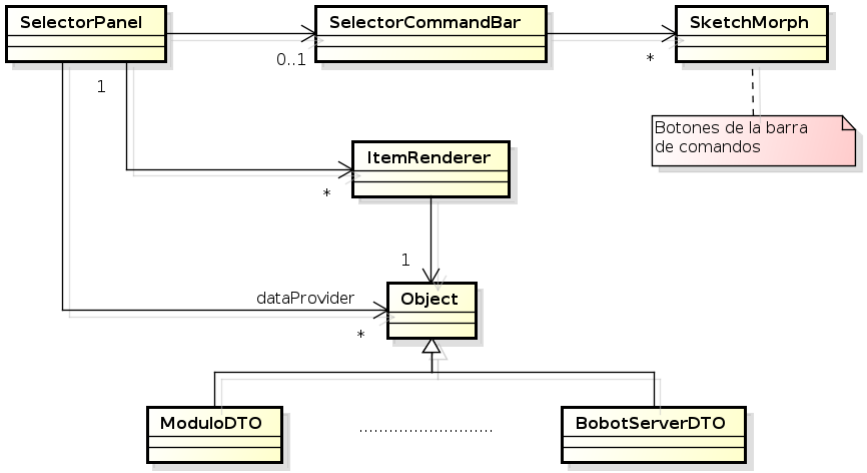


Figura 35: Diseño de Selector.

Existen varios tipos de Selectores como también varios tipos de ItemRenderer dependiendo de la información que se quiere mostrar al usuario.

A continuación se muestran los usados en el sistema.

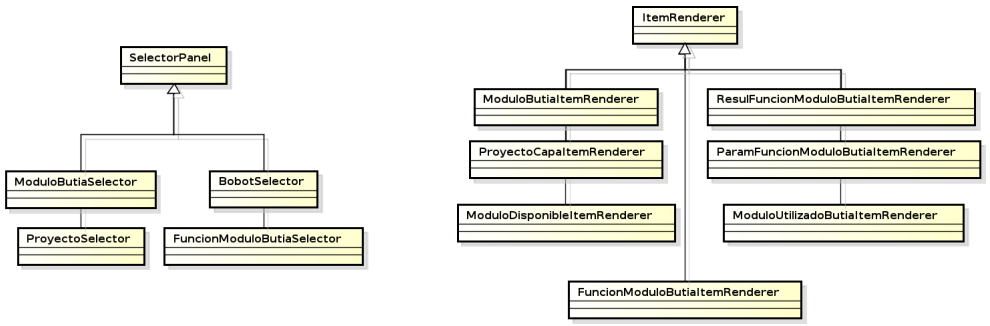


Figura 36: Selectores e ItemRenderers utilizados en el sistema.

La barra de comandos es una botonera vertical de botones representados visualmente por objetos de tipos SketchMorph. Sirven para poder realizar cierta acciones sobre determinados objetos del listado.

Modelo de datos: Como modelo de datos puede ser usado cualquier tipo objetos, generalmente se modela la información en forma de listados de objetos simples que solo almacenan información pero con lógica escasa o nula.

Pantallas

Las diferentes pantalla contempladas en la capa de presentación se numeran a continuación:

- Pantalla de inicio
- Pantalla principal
- Pantalla de calibración
- Pantalla de proyecto
- Pantalla de capa

Todas las pantallas excepto la Pantalla de proyecto y Pantalla de capa son únicas en el sistema, debido a que nunca será necesario llegar a tener más de un proyecto de estos, por lo tanto se aplicó el patrón Singleton para este propósito.

A continuación se muestra un diagrama de navegación entre las diferentes pantallas.

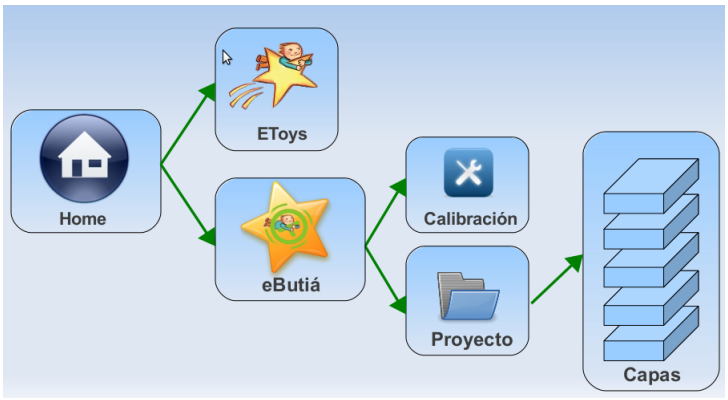


Figura 37: Navegación entre pantallas.

Cada una de las pantallas esta formada por una barra de navegación con las herramientas necesarias para llevar a cabo la navegación, entre otras funcionalidades.

El sistema de navegación en pantalla esta compuesto por un Manejador de navegación (NavigationManager) que tiene las pantallas únicas del sistema para la navegación entre ellas.

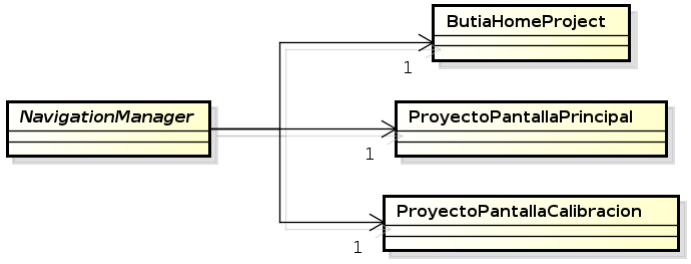


Figura 38: Manejador de navegación con los proyectos únicos en el sistema.

Pantalla de inicio

Es la pantalla de comienzo de la aplicación, desde este punto se permite acceder a la administración de eButiá (Pantalla principal) o a la pantalla inicial de Etoys.

Esta pantalla esta formada como lo indica la figura40.

Los componentes son:

- ButiaHomeProject

  Proyecto usado en la pantalla como controlador de la misma.

- ButiaHomeProjectMorph

  Morph global de la pantalla donde se guardan todos los componentes visuales.

- ButiaNavigationMorph

  Barra de navegación, en esta pantalla solo cuenta con el nombre de la pantalla, el botón de cambio de modo (pantalla simple o pantalla completa), y el botón para salir de la aplicación.

Cumple con los lineamientos planteados por el patrón MVC, con la salvedad que no existe un modelo de datos ya que no es necesario. Todas las dependencias entre el Morph y el proyecto se realizan mediante Binding basado en el uso del patrón de diseño observer.

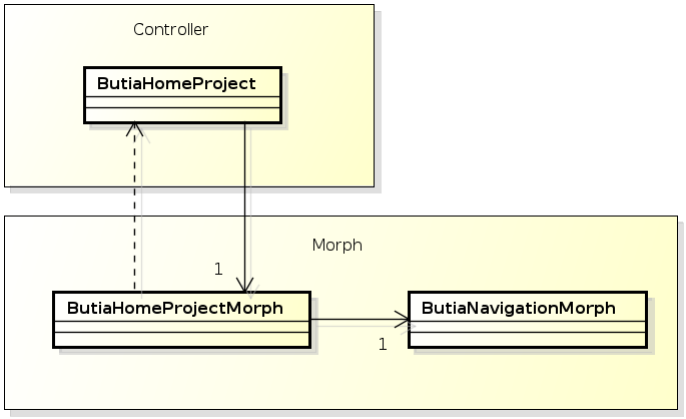


Figura 39: Diseño de pantalla inicial.

Pantalla principal

Es el punto de partida para la creación de proyectos así como también la apertura y borrado de los mismos.

Para la gestión de los proyectos se dispone de un selector de proyectos, el cual brinda las funcionalidades de crear, borrar y abrir proyectos eButiá del disco

rígido, es decir, aquellos guardados por la extensión desarrollada en el marco de este proyecto. A su vez el ItemRenderer destinado para desplegar en pantalla los proyecto tiene la funcionalidad de ingresar al proyecto directamente, además de permitir seleccionar el proyecto para borrarlo.

También se gestionan los diferentes Bobots que pueden ser utilizados por los proyectos, para esto se utiliza otro selector, que permite agregar y remover del listado.

El itemRenderer destinado a mostrar cada bobot consta de tres campos obligatorios para que estos puedan ser usados de manera correcta, los campos son la ip de máquina que se encuentra el bobot, puerto en el que se encuentra disponible, y un campo nombre para ser usado como identificación.

Los componentes que intervienen en esta pantalla son los siguientes,

- BobotItemRenderer

  Item renderer para renderizar el contenido de un bobot.

- BobotSelector

  Listado de bobots disponibles en el sistema.

- BobotServerDTO

  Objeto del listado asociado al selector de Bobots.

- ButiaNavigationMorph

  Barra de navegación de la pantalla.

- ProyectoDTO

  Objeto del listado de proyectos.

- ProyectoItemRenderer

  Item renderer para renderizar el contenido de proyecto.

- ProyectoPantallaPrincipal

  Proyecto controlador de la pantalla.

- ProyectoPantallaPrincipalMorph

  Morph global de la pantalla.

- ProyectoSelector

  Listado de proyectos del sistema.

Su diseño se muestra a continuación,

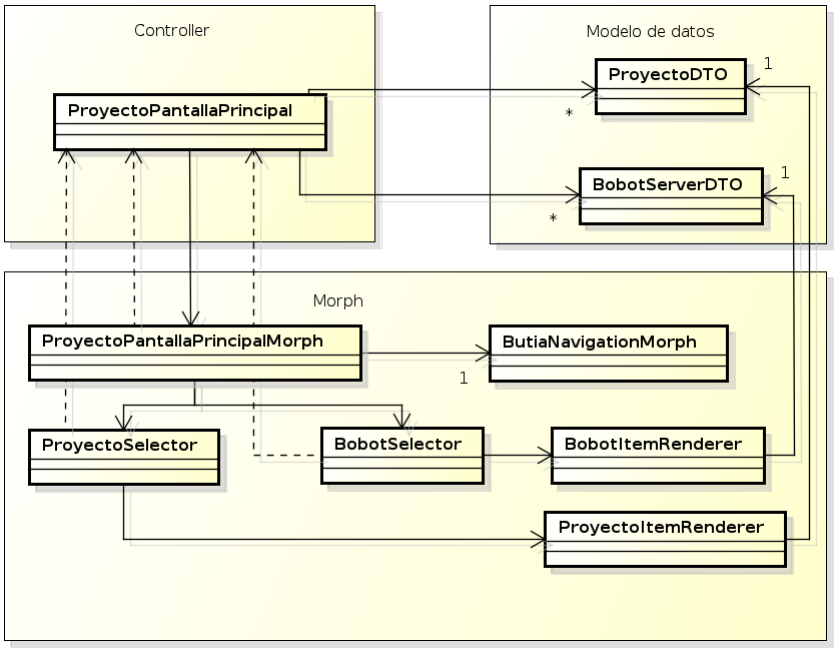


Figura 40: Diseño de pantalla principal.

Los proyectos y bobots listados en los componentes son leídos desde un archivo en el FileSystem en formato XML.

Con este fin, se utiliza un servicio de la capa de negocios para el acceso a la información almacenada en disco rígido.

Cada bobot puede calibrarse de forma independiente, la navegación a la pantalla de comunicación se realiza mediante un componente visual para este propósito, el componente se encuentra contenido en el ItemRenderer del Bobot.

El ingreso a la pantalla de un nuevo proyecto se realiza mediante la opción de la barra de selectores del *ProyectoSelector*.

Pantalla Calibración Permite la ejecución de consulta sobre sensores, como también ejecución de funciones sobre actuadores.

Consiste en dos selectores, uno para los módulos disponibles en el bobot que se esta calibrando, y otro selector con las funciones ofrecidas por algún módulo seleccionado en el primer selector.

Esta pantalla esta formada por los siguientes componentes:

- ProyectoPantallaCalibracion

  Proyecto controlador de la pantalla.

- ProyectoPantallaCalibracionMorph

  Morph global del proyecto.

- ModuloButiaSelector

  Selector de módulos del bobot que brinda las funcionalidades a usar por las aplicaciones.

- FuncionModuloButiaSelector

  Selector de funciones de un módulo seleccionado.

- ModuloButiaItemRenderer

  ItemRenderer del módulo.

- FuncionModuloButiaItemRenderer

  ItemRenderer de la función del módulo seleccionado.

- ResulFuncionModuloButiaItemRenderer

  ItemRenderer del resultado de los valores resultados de ejecutar una función.

- ParamFuncionModuloButiaItemRenderer

  ItemRenderer del parámetro de una función.

La relación de estos componentes se muestra en la siguiente figura.

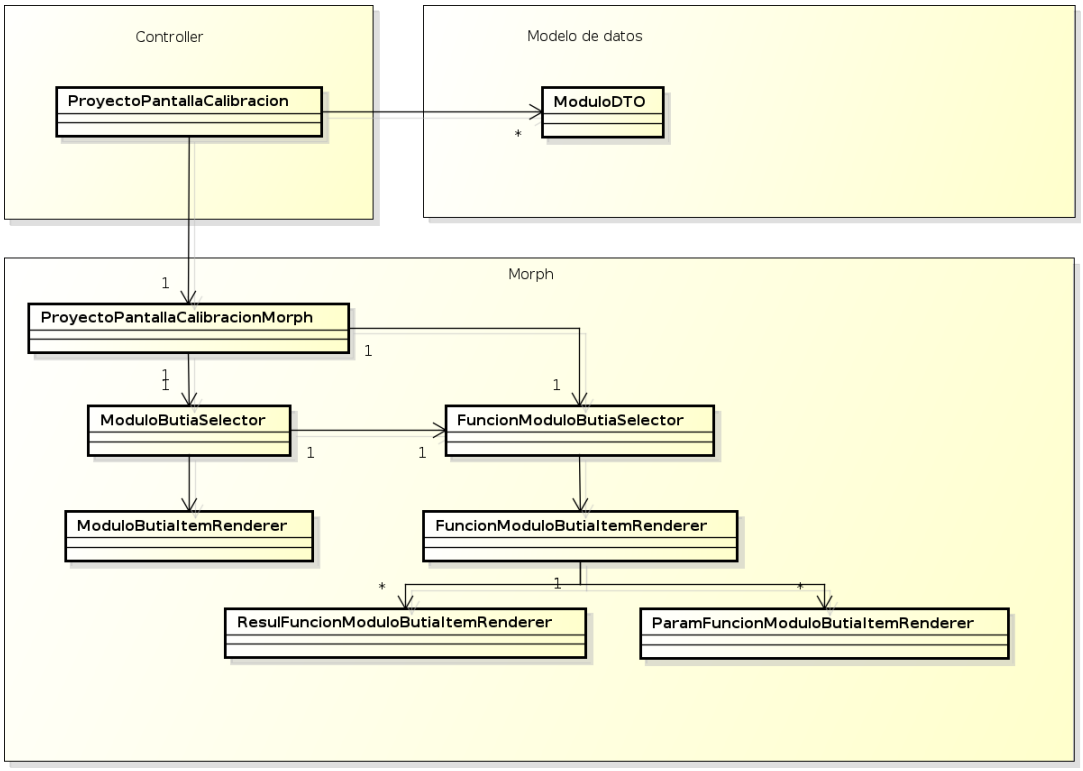


Figura 41: Diseño de pantalla de calibración de Bobots.

El ProyectoPantallaCalibracion se encarga del listado de los módulos para un Bobot dado.

Para esto utiliza el servicio de la capa de negocio que se encarga de la conexión con el Bobot, además, este es el responsable de listar sus módulos.

Pantalla de Proyecto La pantalla de proyecto es una de las pantallas que no son únicas en el sistema. Por cada diferente proyecto existirá una pantalla de proyecto distinta. Su responsabilidad es de mantener toda la información de una aplicación eButiá que los usuarios del sistema codifiquen. Por lo tanto, se guarda la información de cada capa que componen el proyecto, los módulos usados, la imagen del Butiá con los sensores y actuadores, y además el bobot que se utilizará en la ejecución de la aplicación.

Esta compuesta por los siguientes componentes:

- ButiaWindow

  Clase básica para mostrar en pantalla paneles con un header y un cuerpo, usado para los selectores.

- ImgModuloSelector

  Selector de imágenes y módulos que extiende InnerTabSelector, este cuenta con dos menús, uno para mostrar la imagen del Butiá y otro menú para mostrar el listado de los módulos disponibles para usar.

- ImgModuloSelectorContainer

  Es un agrupador de componentes que además de contener el ImgModuloSelector, también agrega la barra de comandos la realización de ciertas funciones.

- InnerTabSelector

  Selector con un submenu incluido, el submenu se utiliza para desplegar diferente contenido en el cuerpo del componente.

- ModulosProvider

  Componente usado como objeto observado por diferentes componentes visuales para la escuchar el cambio de módulos usados por el sistema, este componente es compartido por el proyecto y todos los proyectos capas.

- ProyectoCapaItemRenderer

  ItemRenderer destinado al despliegue en pantalla de una proyecto capa.

- ProyectoCapaSelector

  Selector de proyectos con la lógica de cada capa. Cuenta con una barra de comandos que brinda las funcionalidades de ordenar la capa (subir o bajar en el listado), habilitar y deshabilitar capas, agregar nuevo proyecto capa y eliminar las mismas.

- ProyectoPantallaProyecto

  Proyecto controlador de la pantalla.

- ProyectoPantallaProyectoMorph

  Morph global de la pantalla.

- SubMenuPantallaProyectoMorph

  Componente visual con las opciones de selección de Bobot a usar y los módulos disponibles de ese Bobot. También consta con los botones para ejecutar y detener la aplicación desarrollada.

Su estructura esta definida como lo muestra la siguiente figura

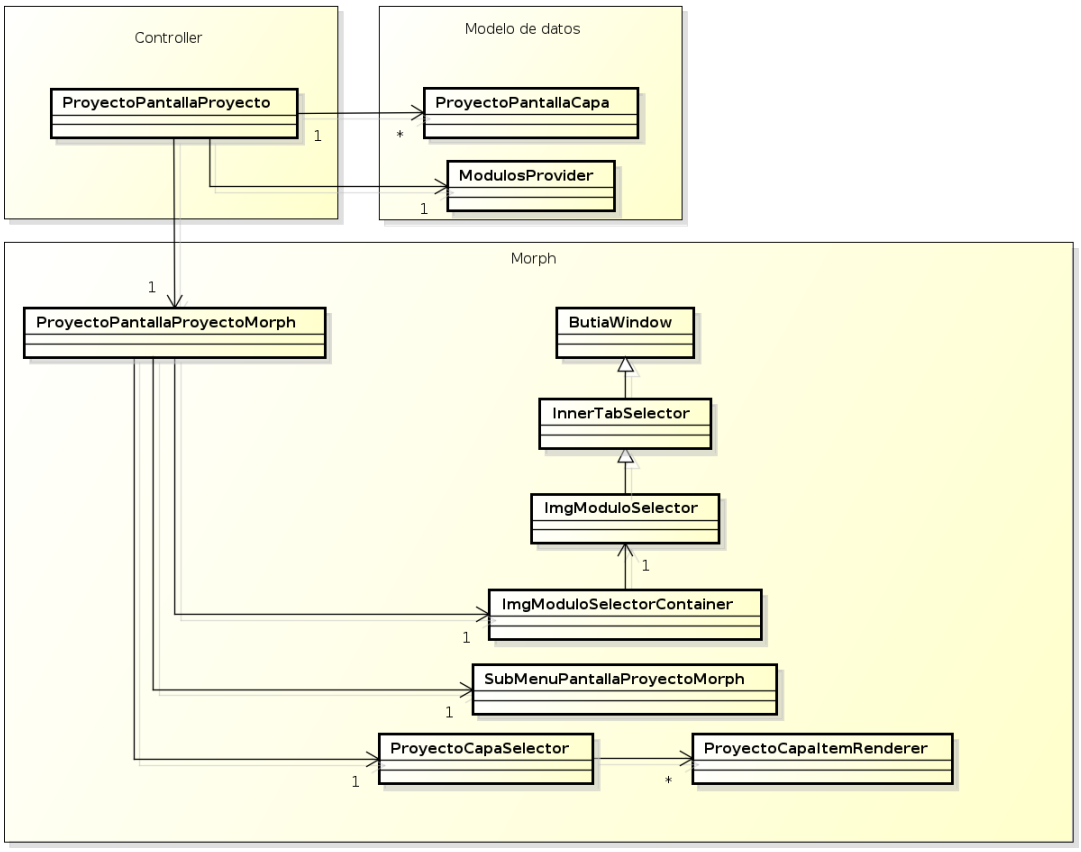


Figura 42: Estructura de la pantalla de proyecto eButiá

En el desarrollo de una aplicación eButiá es necesario seleccionar un Bobot y los las funcionalidades del mismo que serán usadas por las capas, como se describió anteriormente, el submenu de esta pantalla es el encargado de ofrecerle al usuario lo necesario para esto.

La Ejecución de la aplicación se resuelve ejecutando en paralelo la lógica de cada una de las capas habilitadas.

La imagen del Butiá que se utilice se almacenara internamente en el proyecto y se guardara incluida en el propio archivo del proyecto en formato binario durante el proceso de guardado a disco.

Los módulos usados cuentan con un estado, es decir que si por alguna razón el Bobot del proyecto no cuenta con algunos de los módulos seleccionados, estos módulos serán resaltados de forma de informar al usuario de esta situación. La ejecución de la aplicación cuando existe un módulo deshabilitado puede realizar comportamiento no esperados. Para actualizar los estados de los módulos, existe un botón para esto en la barra de comando del ImgModuloSelectorContainer. Quedará como trabajo a futuro la creación de un thread que actualice cuáles son los módulos actualmente conectados.

Las funcionalidades del selector de capas permiten cambiar la lógica a nivel macro, es decir cambiar las prioridades de las capas sobre las demás, para esto, se mantiene un orden implícito de las mismas. Las capas superiores, osea aquellas apiladas sobre las restantes son las que tienen mayor prioridad, por lo tanto se benefician sobre las capas inferiores al momento de reservar un actuador o bien cambiar el valor de los resultados de las consultas realizadas a los sensores por capas inferiores.

También se permite la habilitación de capas, esto sirve para probar capas de forma individual deshabilitando el resto. El propósito es facilitar al usuario la prueba de su aplicación parte por parte, para luego si probar todas las capas juntas, ya que la aplicación reactiva es altamente concurrente y probar todo junto a la vez es tiene una complejidad muy alta.

Pantalla de Proyecto Capa Consiste en una pantalla única para la lógica de una capa, pueden existir varias de esta en el marco de un proyecto eButiá. La información que se muestra en esta pantalla consiste tanto en los módulos seleccionados en la Pantalla Proyecto, como la imagen del eButiá, además de un área de trabajo donde los usuarios codifiquen su comportamiento.

Esta pantalla esta compuesta por los siguientes componentes:

- ButiaParameterTileMorph

  Componente visual el cual permite ingresar valores en un parámetro de una función de las proporcionadas por los módulos del bobot seleccionado.

- ButiaParametersTileMorph

  Conjunto de ButiaParameterTileMorph usado para representar los parámetros de las funciones de los módulos.

- ButiaTileMorph

  Componente visual con el nombre de la función y los parámetros de la misma.

- ButiaPhraseTileMorph

Bloque visual que representa una función del módulo, cuenta con el nombre del módulo, y además el ButiaTileMorph con los parámetros representados por el ButiaParametersTileMorph.

- ScriptEditorMorph

  Componente visual donde se posicionan los bloques de código para armar la lógica de la capa.

- ButiaPlayer

  Es el punto de comienzo en la ejecución de las funcionalidades que proporciona el Bobot. Cada bloque indica al player que función debe ejecutarse ya sea para una ejecución normal, como la supresión de la salida de un sensor, de que módulo y los parámetros que deben usarse. El player es quien resuelve toda la lógica de ejecución al bobot.

- ImgModuloSelectorContainer

  Selector de módulos usado en la pantalla proyecto.

- ModuloDisponibleItemRenderer

  Item renderer de los módulos, permite abrir el cuadro de funciones.

- ProyectoPantallaCapa

  Proyecto controlador de la pantalla.

- ProyectoPantallaCapaMorph

  Morph global de la pantalla.

La siguiente figura muestra como se relacionan estos componentes,

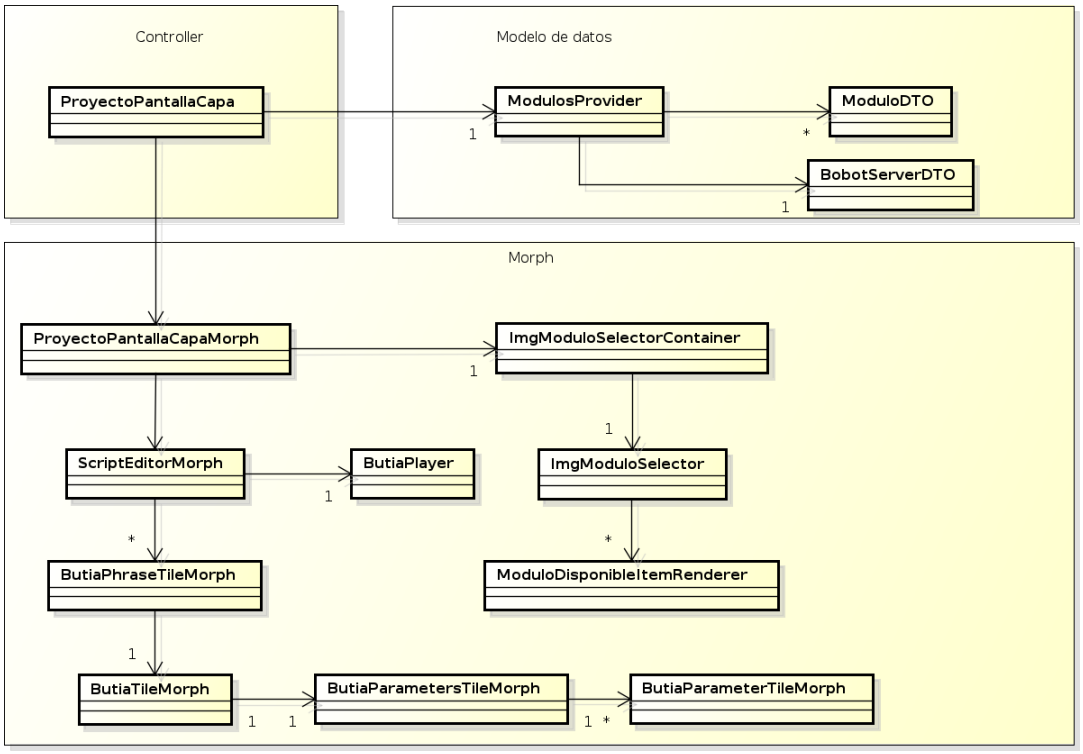


Figura 43: Estructura de la pantalla de Capa eButiá

Los bloques disponibles para utilizar en la implementación de una capa se listan en lo que se denomina un visor, este visor es desplegado cuando se selecciona un módulo del listado mostrado en pantalla. El visor es diferente para cada módulo, mostrando las funcionalidades especificas de cada uno.

Para aquellas funciones que son consideradas como consultas a sensores, se agrega un bloque extra que permite suprimir un valor y cambiarlo por otra para las capas inferiores.

El código en bloques visuales que se encuentran en el ScriptMorphEditor, se compila en el momento que el usuario altera el código ya sea agregando un nuevo bloque o bien cambiando bloques de lugar o cambiando valores de parámetros en la ejecución.

El código compilado utiliza funcionalidades de Etoys como comparación de valores, funciones matemáticas además de utilizar los servicios proporcionados por el ButiaPlayer para la utilización de los módulos del Bobot.

Los servicios implementados son:

- Ejecución de método

  Invoca acciones sobre actuadores o consultas sobre los sensores.

- Sustitución en el valor de retorno de un método

Usado para cambiar el valor de retorno de un método de consulta de un sensor. Aquellas capas con menor prioridad se verán afectadas al consultar el sensor.

A su vez, el ButiaPlayer utiliza un servicio de la componente de negocio para la ejecución de las operaciones a los módulos, este componente es el RobotProxy que se describirá más adelante, en el apartado para el diseño del algoritmo de ejecución.

5.2.2. Negocio

Parte de la capa de negocio se detallara a continuación y el resto más adelante en el apartado para el diseño del algoritmo.

En esta sección se comentara el funcionamiento del Launcher y el Manejador de idiomas.

Launcher El Launcher es la clase que se ejecuta al iniciar cada vez la aplicación de eButiá.

Durante la creación de esta clase, al momento de instalar la extensión eButiá, se realiza la configuración total del sistema, se define la nueva pantalla inicial y se instalan el resto de las pantallas, además también se agenda su ejecución para cada inicio de la aplicación.

Cada vez que inicia el programa, este componente es el responsable de iniciar el componente de navegación con las pantallas que son únicas en el sistema (Pantalla inicial, pantalla principal y pantalla de calibración).

Manejador de idiomas El manejador de idiomas es el encargado de administrar todos los textos visibles para el usuario en la aplicación. Su administración abarca desde el valor del texto según su idioma hasta su almacenamiento en disco.

Cada uno de los textos que se muestran tienen un valor por defecto grabados en el código fuente. Este valor es usado cuando no existe uno almacenado en disco.

El acceso a disco se realiza haciendo uso del componente de persistencia *Manejador de propiedades*.

El archivo donde se almacenan los textos internacionalizables es el Idioma.xml. Su estructura tiene la siguiente forma:

```
<es>
<bobot>bobot</bobot>
<módulos>módulos</módulos>
<configuracionDeModulos>Config. de módulos</configuracionDeModulos>
<cargarImagen>Cargar Imagen</cargarImagen>
<disenio>diseño</disenio>
<debeSeleccionarBobot>Debe seleccionar un Bobot.</debeSeleccionarBobot>
<capas>capas</capas>
<errorDeConeccionAlBobot>No se pudo conectar al Bobot en {1} al puerto {2}.</errorDeConeccionAlBobot> <clickAquiParaIniciar>Click aqui para iniciar</clickAquiParaIniciar>
<irAEtoys>irAEtoys</irAEtoys>
<proyectos>Proyectos</proyectos>
<bobots>Bobots</bobots>
<funciones>funciones</funciones>
<correr>correr</correr>
<bienvenidosAeButia>Bienvenidos a eButiá</bienvenidosAeButia>
<detener>detener</detener>
</es>
```

En este caso el lenguaje de los textos es el Español.

Cada componente visual es el encargado de obtener su texto que pretende mostrar al usuario haciendo uso del propio Manejador de idiomas.

5.2.3. Persistencia

Se encarga del almacenamiento en disco. La información almacenada va desde el idioma hasta la configuración de proyectos abiertos o los distintos bobot-servers configurados.

Manejador de propiedades El manejador de propiedades esta disponible para toda la capa de negocio. La información que se gestiona haciendo uso de este componente se almacena en varios archivos. En el caso anterior, los idiomas se almacenan en el archivo idioma.xml, y las propiedades de la aplicación, en propiedades-butia.xml.

Las propiedades que se guardan comprenden los proyectos existentes o agregados al sistema, su formato se almacena de la siguiente forma:

```
<proyectos maxId="170">
    <proyecto id="169">
        <name>Proyecto2</name>
        <path>/tmp/Proyecto2.pr</path>
    </proyecto>
</proyectos>
```

El atributo maxId es usado para generar identificadores nuevos para cada proyecto que se agregue.

Además también los Bobots se almacenan junto con las restantes propiedades, el formato se muestra a continuación:

```
<bobot-server maxId="34">
    <Butiá id="15">
        <puerto>2011</puerto>
        <nombre>Butiá_1</nombre>
        <ip>localhost</ip>
    </Butiá>
</bobot-server>
```

5.2.4. Comunicación

Este componente es el encargado de la comunicación de la aplicación eButiá con los Bobot que se usan. Actualmente la comunicación se realiza a través de Sockets TCP.

Los clases involucradas desde la ejecución de comandos al Bobot ya sea para realizar acciones de los actuadores o lecturas de sensores, son las siguientes:

BobotAdapter Esta clase se encarga del envío de mensajes a través de la red mediante sockets. Los mensajes son traducidos a comandos, los cuales se detallan a continuación:

- LIST

  Lista los módulos actualmente disponibles en la placa de entrada/salida.

- DESCRIBE moduleName

  Describe las operaciones implementadas por un componente específico.

- OPEN moduleName

  Abre un cierto componente para ejecutar sus operaciones.

- CALL moduleName operation param1, param2, ..., paramN

  Llama a una operación de un determinado módulo pasándole una serie de parámetros.

- CLOSEALL

  Cierra la conexión sin devolver ningún parámetro.

Como entrada al método de ejecución de comandos se requieren el nombre del módulo al que se quiere invocar la función, el nombre de la función en cuestión, y los parámetros que sean necesarios.

Esta función es bloqueante, y luego de obtener el resultado se retorna para que sea parseado por el Bobot.

Bobot Es el punto de entrada para la ejecución de las funciones, más adelante en el diseño del algoritmo se detallara como ciertos componentes de la lógica utilizan esta Clase para enviar comandos a los sensores/actuadores.

Tiene la responsabilidad de interpretar los resultados de los comandos ejecutados al bobot y crear los objetos del modelo que representa. Por lo tanto es quien crea a partir del resultado de los comandos usados por el BobotAdapter, la metadata de los Módulos, Métodos, Parámetros y Resultados.

Por ejemplo del mensaje parseado que se utiliza para describir un módulo, se muestra a continuación, donde se describe el módulo *dis* para obtener la metadata de los métodos que se le pueden aplicar junto con los tipos de los parámetros y resultados,

-> DESCRIBE dist
<- {getDistancia={ parameters={}, returns={[1]={['rname']='par1',['rtype']='int',},}},}

Cuadro 3: Ejemplo de mensaje enviado y recibido desde la aplicación.

ModuloDriver Esta clase que representa el componente conectado a la placa de entrada/salida. Esta compuesto el nombre y además los métodos ofrecidos para la ejecución.

Modulo Subclase particular de ModuloDriver especifica del tipo de comunicación realizada con el robot usado por el actual Butiá.

Metodo Metadata de un método disponible de un módulo conectado al Butiá, en el ejemplo 3 se puede ver que existe un solo método, el getDistancia que no recibe parámetros pero si retorna un valor de tipo entero.

Parametro El parámetro no es más que una representación de los parámetros ofrecidos por el módulo, consiste en un nombre de parámetro, un tipo de valor, y un orden en el método del módulo.

Resultado El parámetro no es más que una representación de los parámetros ofrecidos por el módulo, consiste en un nombre de parámetro, un tipo de valor, y un orden en el método del módulo.

RobotDriver Como se detallara más adelante en el diseño del algoritmo, debe ser extendida para crear los drivers diferente para cada tipo de robot distinto.

A continuación se muestra el diseño de como se relacionan estos componentes.

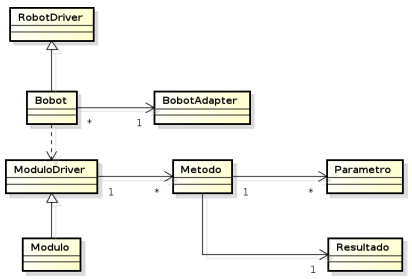


Figura 44: Diseño de la capa de comunicación.

6. Diseño del algoritmo

En esta sección se detalla el diseñó de la arquitectura Subsumption dentro del sistema, la estructura y la forma de ejecución de las capas siguiendo las especificaciones de esta arquitectura reactiva. Está fuertemente basado en, y para más información se debe acudir a, el documento de Diseño-Subsumption[2].

Se recuerda que según lo visto en el Documento de Estado del Arte [5] las principales características de Subsumption son:

1. Diseñar comportamientos en forma de capas.
2. Cada capa debe ejecutarse paralelamente.
3. Las capas deben tener un orden.
4. Las capas superiores pueden inhibir los comportamientos de las inferiores. Es decir que las capas superiores pueden reescribir la información obtenida a través de la lectura de sensores de capas inferiores.
5. Las capas superiores pueden suprimir los comportamientos de las capas inferiores. Implica que si las capas superiores están ejecutando un actuador las capas inferiores no pueden ejecutarlo.

La solución propuesta consiste en ejecutar paralelamente cada una de las capas repetidamente, es decir que cada capa al finalizar su ejecución, comience nuevamente.

De esta forma el desarrollador deberá crear comportamientos ágiles (tiempo de ejecución relativamente corto), la forma de garantizar esto es con comportamientos sencillos, como plantea la arquitectura bajo un paradigma de concurrencia cooperativa. En caso contrario, aquellos que tarden demasiado tiempo, pueden superar el tiempo de reserva de recursos, dejando libre estos recursos por una ventana de tiempo en la cual los procesos de menor prioridad puedan acceder al recurso, provocando un funcionamiento no deseado ocasionando que la lógica de una capa muchas veces no sea completada de la mejor forma.

Dicho anteriormente cada capa puede suprimir o inhibir algún recurso para las capas inferiores por un intervalo de tiempo **n**. Es recomendable que la ejecución de una capa sea más rápida que el tiempo de reserva usado para los módulos utilizado en dicha capa, de esta forma se consigue que ninguna capa inferior consiga obtener el recurso libre en la ejecución de dicha capa.

A continuación se presentan los diagramas de clases y los distintos diagramas de comunicación que componen la solución.

6.1. Diagrama de Clases

En la figura 45 se expone a continuación el diagrama de clases compuesto únicamente por las clases más relevantes que intervienen en el algoritmo.

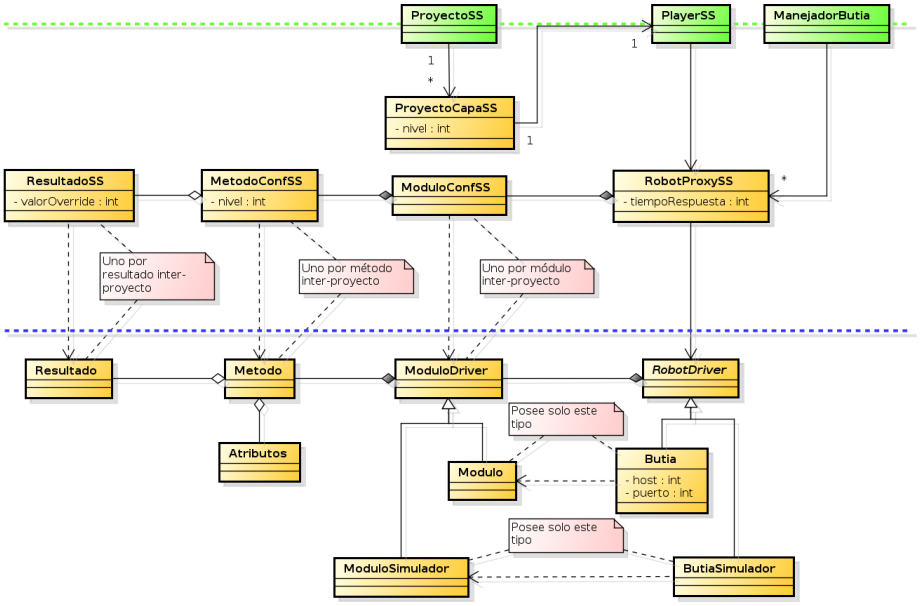


Figura 45: Diagrama de clases

Este diagrama muestra claramente las clases pertenecientes a los distintos componentes de la arquitectura. Se muestran únicamente clases incluidas en capa de Negocio y capa de Comunicación. También se puede observar cuales de estas capas están directamente relacionadas con las capas superiores. De esta forma aquellas clases incluidas por debajo de la línea azul pertenecen a la capa de comunicación y aquellas capas por sobre la línea pertenecen a la capa de Negocio.

Siguiendo este razonamiento se puede observar que la interfaz tiene vínculos directos con la capa de negocio a través de las clases ProyectoSS, PlayerSS y ManejadorButia. De la misma forma la capa de Negocio tiene vínculo directo con la clase RobotDriver.

6.1.1. Descripción de clases y responsabilidades

1. ProyectoSS

   Esta clase representa un proyecto de Subsumption. Esta formado por las capas pertenecientes al algoritmo.

2. PlayerSS

   Se encarga de procesar los pedidos generados a través de la programación con bloques y enviarlos en un formato correcto al proxy.

3. ManejadorButia

   Contiene la lista de todos los robots configurados en eButiá, estos podrán ser utilizados desde cualquier proyecto.

4. ProyectoCapaSS

   Representa una capa de la arquitectura Subsumption. Contiene el código de un comportamiento simple.

5. RobotProxy

   Oficia de proxy ante el Driver. Esto evita que las llamadas desde código se ejecuten directamente en el robot y se puedan realizar supresiones de las salidas e inhibiciones de las entradas.

6. ModuloConfSS, MetodoConfSS y ResultadoSS

   Son componentes del proxy y representan la configuración de cada resultado, método y módulo del Driver respectivamente. El Resultado guarda el valor que se obtiene por capas inferiores cuando se realiza la supresión de las entradas por una capa superior. El atributo nivel en MetodoConfSS representa o bien el nivel de la capa que quiere inhibir a las inferiores, o bien el nivel de la capa que esta haciendo uso de un recurso (por ejemplo ejecutando un actuador).

7. RobotDriver, ModuloDriver, Metodo, Resultado y Parametro

   Componen la metadata de los procedimientos y módulos de un determinado robot. Las clases RobotDriver y ModuloDriver, deben ser extendidas para crear los drivers de los distintos tipos de robots a conectar. Es importante observar que dos robots Butiá con distintos sensores y actuadores corresponden al mismo driver ya que el driver en este caso representa el Bobot Server.

Se puede ver como las clases que componen la solución respetan las 5 especificaciones expuestas en Subsumption:

1. **Comportamientos en capas**

   Cada objeto de la clase ProyectoCapaSS posee el código de una capa.

2. **Paralelismo**

   El diagrama no muestra ninguna restricción al respecto.

3. **Orden**

   Cada objeto de la clase ProyectoCapaSS posee un atributo nivel que representa la posición de la capa en el stack.

4. **Supresión**

   La clase MetodoConfSS tiene un atributo nivel y un atributo resultado que representa la capa que desea hacer la supresión y cual es el resultado a retornar para las capas inferiores.

5. **Inhibición**

   La clase MetodoConfSS tiene un atributo nivel que representa el nivel de la clase que la ha ejecutado si es un actuador.

6.2. Diagramas de comunicación

Esta sección muestra los diagramas de comunicación de los principales métodos que componen el algoritmo de Subsumption.

En la solución propuesta primero se ejecutan todas las capas repetidas veces (luego de finalizar comienza su ejecución nuevamente). Luego, como cada capa ejecuta comportamientos, obtiene valores de los sensores y realiza acciones sobre los actuadores. Para esto, cada vez que realiza acciones sobre los actuadores, si no está reservado se lo reserva por un tiempo **n**. Finalmente, desde cada capa se puede suprimir / remplazar las entradas de los sensores de las capas inferiores.

Se destacan 3 métodos principales que componen la solución.

- ejecutar

  Realizado sobre el proyecto principal para ejecutar todas las capas.

- ejecutarMetodo

  Invocado por cada capa únicamente en el momento de ejecución, para realizar acciones sobre actuadores o consultas sobre sensores.

- suprimirSensor

  Invocado por las capas superiores para suprimir los resultados devueltos a capas inferiores.

6.2.1. Método: ejecutar

Este método es llamado para comenzar la ejecución el comportamiento. Se toma como precondición que la clase proyecto posee los siguientes 3 valores:

1. RobotSeleccionado

   Driver del robot que está seleccionado para realizar la ejecución.

2. Capas

   Lista de capas que representan todos los sub-comportamientos del algoritmo. Éstas pueden estar desactivadas para no utilizarse por diversas razones que van desde testing del comportamiento a obtención de comportamientos distintos.

3. TiempoReserva

   Representa el tiempo **n** en que una determinada capa superior tiene en su "poder" un método determinado. Esto permite que una capa ejecute diversos métodos sin tener que liberar los recursos de los mismos.

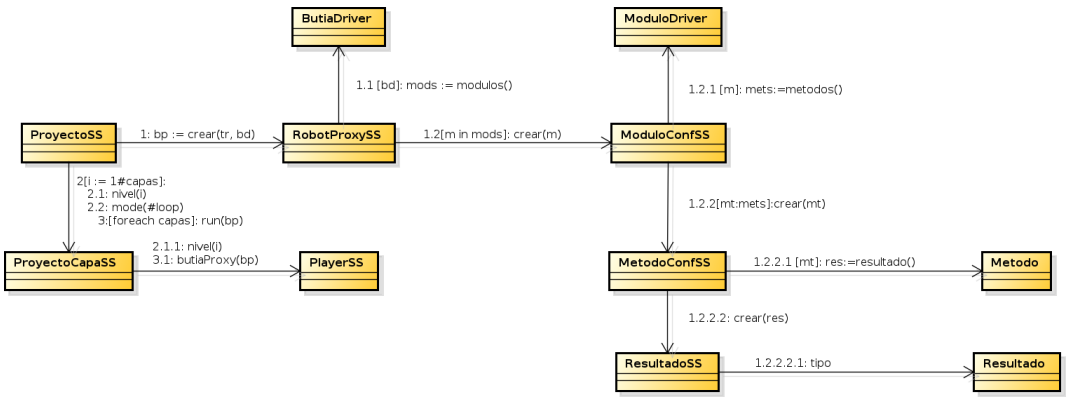


Figura 46: Diagrama de Comunicación: "ejecutar"

Se puede observar que el método implementa las primeras dos condiciones de Subsumption y permite la implementación de la tercera:

1. **Comportamientos en capas**

   Se ejecutan todas las capas que representen comportamientos y estén activas.

2. **Paralelismo**

   Las capas son ejecutadas de forma paralela.

3. **Orden**

   Se setea a cada objeto de la clase ProyectoCapaSS un atributo nivel que representa la posición de la capa en el stack.

6.2.2. Método: ejecutarMetodo

Luego de comenzada la ejecución de todas las capas, se pueden ejecutar bloques de códigos que internamente representan las funcionalidades ofrecidas por el robot. Este método se encarga de ejecutar dichas funciones, como también de controlar la ejecución de aquellas que no deben ser ejecutadas. En definitiva, aquí se aplica efectivamente la inhibición y la supresión, y se implementa la primera de ellas. Para esto, existen dos casos, distinguidos sobre el método a ejecutar:

1. Método Sensor

   Si el método que se va a ejecutar es un sensor, se siguen los siguientes pasos:

   a) Chequear sobre el método si se puede ejecutar o no.

   b) Si se puede ejecutar, ejecutarlo sobre el Driver.

   c) Si no se puede ejecutar (Supresión) se obtiene el valor almacenado por la capa superior.

2. Método Actuador

   Si el método es un actuador, se siguen los siguientes pasos:

   a) Chequear sobre el módulo si se puede ejecutar o no.

   b) Si se puede ejecutar:

      1) Ejecutarlo sobre el Driver.
      2) Reservar el módulo (implementando la inhibición)

A continuación se muestra un diagrama de comunicación que expresa lo descripto anteriormente.

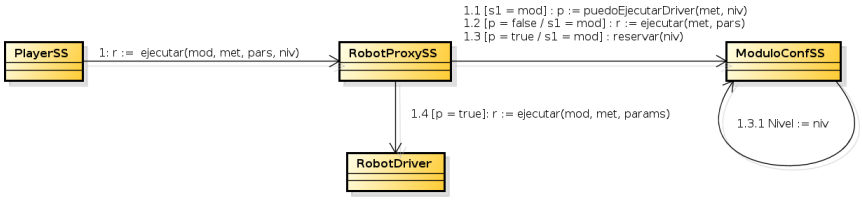


Figura 47: Diagrama Comunicación: "ejecutarMetodo"

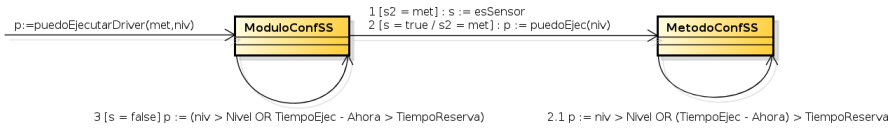


Figura 48: Diagrama Comunicación: "puedoEjecutarDriver(método,nivel)"

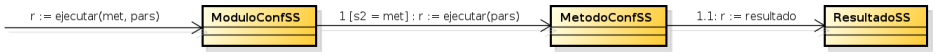


Figura 49: "ejecutar(método,parámetros)"

Observar que el método de la Figura 49 tiene como precondición puedoEjecutar = false.

Este método utiliza el orden para determinar si un método está suprimido y para consultar si un método está inhibido e inhibirlo:

1. **Orden**

   Se considera el atributo nivel para determinar si la capa superior tiene control sobre un módulo actuador o sobre una función de un sensor.

2. **Supresión**

   Cada vez que se ejecuta un método de un actuador se corrobora que se pueda realizar y sino se lo suprime es decir, se ignora la ejecución.

3. **Inhibición**

   Cuando una capa superior inhibe a las inferiores, estas al momento de consultar van a obtener un valor que fue asignado por la capa superior, de esta forma la capa superior puede sobrescribir los resultados de consultas para las capas inferiores.

6.2.3. Método: suprimirSensor

La inhibición se realiza de forma inmediata, al ejecutar un método actuador desde una capa de nivel superior. Para esto el Player tiene a disposición un nuevo método que permite al usuario la posibilidad de setear un determinado resultado a una función. Por lo tanto es el responsable de implementar la Inhibición.

En la figura 50 se muestra un diagrama de comunicación que describe el procedimiento para implementar el método.

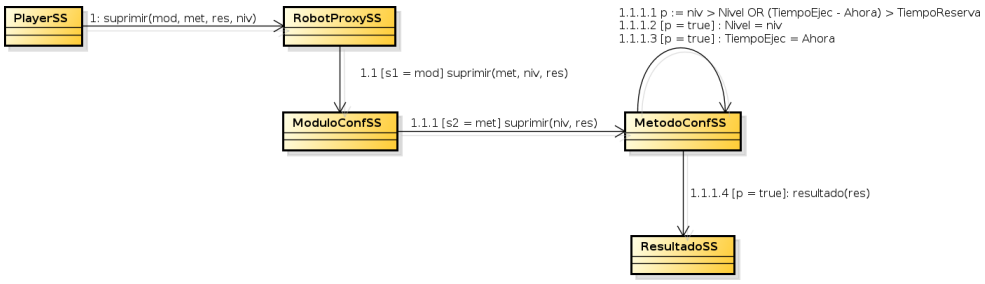


Figura 50: Diagrama Comunicación: "suprimirSensor"

Este método implementa efectivamente la supresión expuesta por la arquitectura:

1. **Orden**

   Se actualiza el nivel del método con la capa que desea suprimir a las inferiores.

2. **Supresión**

   Es implementada desde las capas superiores que deseen suprimir las entradas de las inferiores.

6.3. Observaciones

Es importante observar que en el caso de los sensores se reserva el método y en el caso de los actuadores se reserva todo el módulo. Esto es para evitar llegar al actuador de cualquier manera, manteniendo la versatilidad que ofrece tener distintas formas de acceder a él.

También podemos observar que la solución propuesta, con el conjunto de métodos descriptos en la parte 3, implementa efectivamente las características principales del algoritmo de Subsumption.

7. Verificación del sistema

En esta sección se detallan todas las pruebas realizadas junto con los errores identificados. Estos luego se dividirán en dos categorías, Errores Corregidos y Errores Conocidos. Los primeros son aquellos errores que fueron identificados y han sido corregidos; y los segundos son aquellos que, si bien fueron identificados, no han podido ser corregidos dentro del plazo delimitado por este proyecto.

7.1. Pruebas realizadas

En esta sección se detallan las distintas pruebas realizadas para verificar el correcto funcionamiento del sistema, así como también la correctitud de los documentos de ayuda generados, Manual de Instalación y Manual de Usuario.

Para realizar las pruebas se utilizó tanto el robot Butiá como un simulador eButiá-Simulador.

7.1.1. eButiá-Simulador

Es una aplicación que, motivada y creada a partir de los conocimientos adquiridos en este proyecto, implementa un simulador del robot Butiá. Este fue implementado, al igual que eButiá, como una extensión a Etoys. Expone la misma interfaz que el bobot-server, por lo que la interacción se realiza de la misma forma, siendo este cambio indetectable por eButiá o cualquier otra aplicación que lo utilice.

Módulos Implementados Esta aplicación expone tres módulos distintos, uno de actuadores y dos de sensores. Estos módulos exponen las mismas funciones que las que exponen con los drivers de bobot-server. Si bien esto no es importante en eButiá, sirve para que pueda utilizarse con otras aplicaciones que utilizan el mismo intermediario para comunicarse con el robot y no poseen carga dinámica de módulos. Estos módulos son los listados a continuación:

- Módulo de motores
- Sensor de Distancia
- Sensor de Escala de grises

Con estos módulos, el robot simulado es capaz de moverse según las velocidades ingresadas en los motores y colisionar con los objetos con altura de la escena. Además el sensor de distancia tiene la capacidad de devolver valores de distancia a objetos con altura que se encuentren en frente de él. Finalmente, el sensor de grises devuelve valores de escala de grises de objetos sin altura que se encuentren debajo de la parte delantera del mismo.

Para construir la escena con los objetos con los que interactúa el simulador, se utilizan todos los objetos de Etoys que se encuentren en la pantalla, es decir todo objeto Morph o extensión de este. Para lograr esto se le añadió una propiedad al Halo de los Morphs que permite añadirle altura.

Vista del Simulador Ofrece una vista superior del robot y la capacidad de construir entornos en los cuales actúe utilizando las herramientas de Etoys. A continuación se muestra una captura de pantalla del simulador.

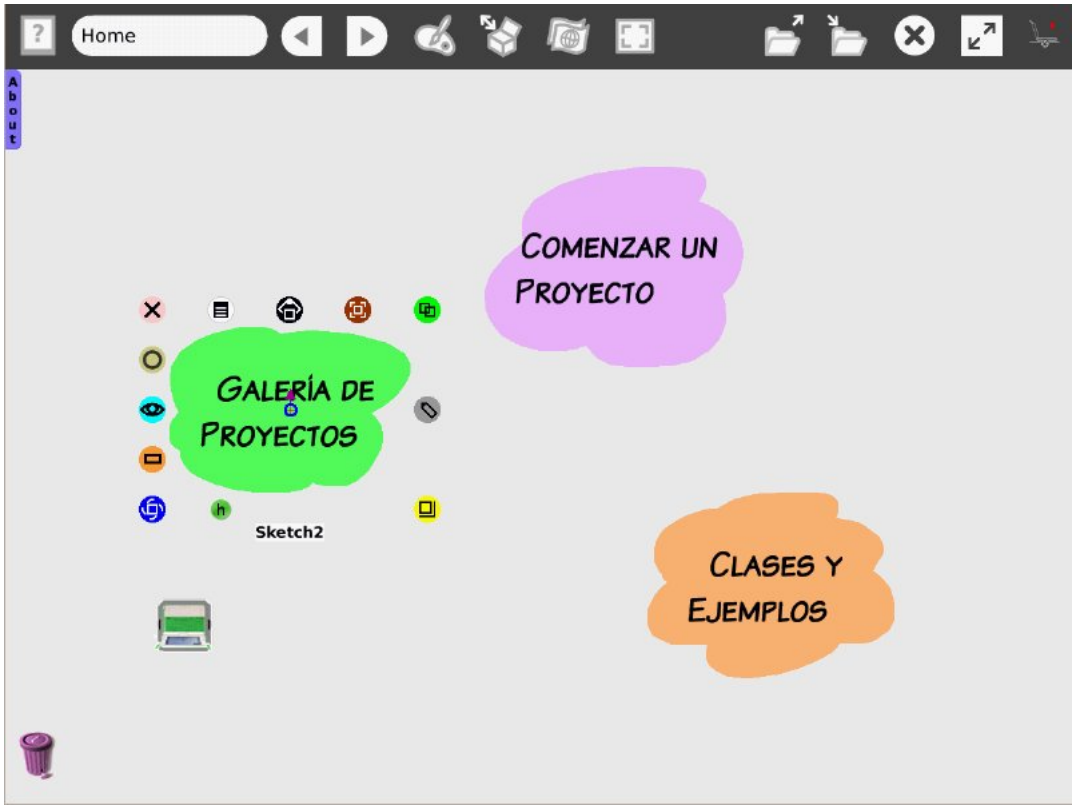


Figura 51: eButiá-Simulador

En esta se ven claramente los tres elementos gráficos principales añadidos por el simulador:

- Botón de play: en la barra de herramientas principal a la derecha, un dibujo del robot Butiá que nos permite arrancar y detener la simulación.
- Robot Simulado: una vista superior del robot donde se distingue claramente el computador XO.
- El botón de altura: un botón verde con una hache dentro del halo que nos permite añadir la altura a cualquier objeto del entorno, en este caso a la imagen de "Galería de Proyectos".

7.2. Niveles de error

Con el fin de determinar cuáles errores con mayor prioridad a corregir, se han definido distintos niveles de error. Estos son los descriptos a continuación:

- Catastrófico: la presencia de este tipo de errores impide el uso del sistema.
- Crítico: el error no permite ejecutar una funcionalidad del sistema, no satisfaciendo los requerimientos.
- Media: produce un daño menor, ya que su existencia no impide la ejecución de una funcionalidad, solo dificulta su utilización.
- Menor: no causa perjuicio al sistema, ya que no elimina la ejecución de las funcionalidades de forma sencilla, pero requiere de mantenimiento o reparación.

7.3. Componentes del sistema

Con el fin de diferenciar donde se encuentran los distintos errores, se detallan a continuación los distintos componentes del sistema. Además para cada uno, se agrega un sufijo al código de error que hace referencia al componente y permite identificar mediante el código el componente y el error.

Componente	**Prefijo**
Documento <NombreArchivo>	doc_na_
Imagen eButiá	img_eb_
Instalador XO	ins_xo

7.4. Errores Identificados

A continuación se muestra una lista de los errores identificados con una descripción detallada de cada uno.

Código	Componente	Error	Descripción	Gravedad	Versión
img_eb_001	Imagen eButiá	Falta mensaje con servidor "Desconectado"	Cuando se entra a la pantalla de calibración con el servidor no levantado, no muestra ningún mensaje de error.	Menor	106
ins_xo_001	Instalador XO	No logra ejecutar la aplicación	Cuando se ejecuta el archivo.xo desde el pendrive, se instala, pero no ejecuta la aplicación.	Catastrófico	–
ins_xo_002	Instalador XO	Aviso al ingresar a la aplicación.	Luego de instalar la aplicación, se ejecuta eButiá, pero al iniciar aparece un mensaje de error que dice "No content to install". Al presionar "Abandon" sigue ejecutando normalmente.	Media	–
img_eb_002	Imagen eButiá	Error al abrir por primera vez	Al abrir da un error "UndefinedObject" en PresentacionUtils provocado por la clase ButiaHomeProjectMorph	Catastrófico	110
img_eb_003	Imagen eButiá	Botón de play en Pantalla de Proyecto	El botón de play para ejecutar los proyectos no cambia a botón de stop cuando se está ejecutando.	Medio	110
img_eb_004	Imagen eButiá	No se pueden crear múltiples capas muy rápidamente	Cuando se crean dos capas seguidas los scripts internos quedan con el mismo nombre.	Medio	110

Código	Componente	Error	Descripción	Gravedad	Versión
img_eb_005	Imagen eButiá	Selección de módulo en Pantalla de Calibración	En la calibración solo se selecciona un módulo al presionar sobre el botón azul	Menor	110
img_eb_006	Imagen eButiá	Error ortográfico de la palabra “disenio”	La palabra “diseño” está escrita “disenio” en la Pantalla de Proyecto	Menor	110
img_eb_007	Imagen eButiá	Error ortográfico de la palabra “modulo”	La palabra “módulo” está escrita como “modulo” en la pantalla de Proyecto	Menor	110
img_eb_008	Imagen eButiá	Error en el cambio de nombre de una capa.	Al cambiarle el nombre a una capa, el mismo no se refresca automáticamente en la Pantalla de Proyecto	Menor	110
img_eb_009	Imagen eButiá	Error al ejecutar un proyecto de más de una capa.	El sistema da error en el resultado obtenido del socket al ejecutar un proyecto de más de una capa.	Crítico	113
img_eb_010	Imagen eButiá	Agregado de capa al principio de la lista	No se esta modelando un stack de capas, agregando cada nueva capa arriba de todas	Menor	113
img_eb_011	Imagen eButiá	Chequeo de conexión con bobots	No se controla la disponibilidad a los bobots durante cada consulta efectuada a ellos.	Medio	113
img_eb_012	Imagen eButiá	Manejo de concurrencia en Proxy	Las ejecuciones en el proxy realizadas por distintas capas no están excluidas mutuamente	Crítico	113
img_eb_013	Imagen eButiá	Bloqueo de morphs	Existen morph en la pantalla Home que pueden ser arrastrados cambiando la apariencia de la pantalla	Menor	113

Código	Componente	Error	Descripción	Gravedad	Versión
img_eb_014	Imagen eButiá	Tamaño de Phrase de código	Los bloques de código no tienen un tamaño adaptado al contenido sino que el bloque se expande al tamaño del script	Medio	113
img_eb_015	Imagen eButiá	Texto de función para los bloques suprimir	No aparece el texto suprimir en el bloque dentro del código de la capa cuando se utiliza el bloque	Medio	113
img_eb_016	Imagen eButiá	Actualización del estado del módulo	Al desconectar el bobot, no se deshabilitan los módulos al actualizar el listado	Medio	130
img_eb_017	Imagen eButiá	Duplicación entrada de proyectos guardados	Al sobre-escribir un proyecto se esta duplicando el proyecto en el listado de la pantalla principal	Medio	130
img_eb_018	Imagen eButiá	Quitar botón de quitar módulo	Existe el botón de quitar módulo en el visualizador de módulos de cada capa	Medio	130
img_eb_019	Imagen eButiá	Orden de tabs en los selectores de módulos	El tab por defecto que debe aparecer es el de módulos y no el de diseño	Medio	130
img_eb_020	Imagen eButiá	Error al utilizar el Suprimir	No se puede suprimir los sensores de las capas inferiores.	Catastrófico	136
img_eb_021	Imagen eButiá	Error al Suprimir capas superiores	El suprimir también suprime los sensores para las capas superiores.	Crítico	138
img_eb_022	Imagen eButiá	Duplicación entrada de proyectos guardados con el mismo nombre	Al crear dos proyectos con el mismo nombre y guardados dentro del mismo path, quedan dos entradas en la lista de proyectos apuntando al mismo archivo físico	Medio	141

Código	Componente	Error	Descripción	Gravedad	Versión
img_eb_023	Imagen eButiá	Error al abrir un proyecto del listado cuando se borro el archivo	Debería de mostrar error de no existencia del archivo	Menor	144
img_eb_024	Imagen eButiá	Categoría eButiá esta duplicada en el visor de módulo	Debería de mostrar una única vez la categoría eButiá	Menor	146
ins_xo_003	Instalador XO	No se instala el bobot-server con el instalador.	Al obtener eButiá e instalarlo, se debe tener instalado por otro medio el bobot server.	Menor	–
ins_xo_004	Instalador XO	El bobot-server ejecutado no se detiene.	Se inicia el bobot-server junto con la aplicación, pero este no se detiene al finalizar.	Menor	–
doc_na_001	Documento <Manual de Usuario>	El combo box de los módulos es el de la derecha.	El combo box de los módulos está mal señalado.	Menor	1.1, pag. 10
doc_na_002	Documento <Manual de Usuario>	Error en término “chequeado”	Se menciona la palabra "chequeado" en lugar de "verificado"	Menor	1.1, pag. 13

7.5. Errores Corregidos

A continuación se listan aquellos errores dentro de los errores que fueron identificados que fueron corregidos.

Código	Componente	Error	Gravedad	Versión De Correcci
ins_xo_001	Instalador XO	No logra ejecutar la aplicación	Crítico	–
ins_xo_002	Instalador XO	Aviso al ingresar a la aplicación.	Media	–
img_eb_002	Imagen eButiá	Error al abrir por primera vez	Catastrófico	113
img_eb_008	Imagen eButiá	Error en el cambio de nombre de una capa.	Menor	113
img_eb_004	Imagen eButiá	No se pueden crear múltiples capas muy rápidamente	Medio	120
img_eb_005	Imagen eButiá	Selección de módulo en Pantalla de Calibración	Menor	120
img_eb_009	Imagen eButiá	Error al ejecutar un proyecto de más de una capa.	Crítico	120
img_eb_006	Imagen eButiá	Error ortográfico de la palabra “disenio”	Menor	130
img_eb_007	Imagen eButiá	Error ortográfico de la palabra “modulo”	Menor	130
img_eb_010	Imagen eButiá	Agregado de capa al principio de la lista	Menor	130
img_eb_001	Imagen eButiá	Falta mensaje con servidor “Desconectado”	Menor	131
img_eb_011	Imagen eButiá	Chequeo de conexión con bobots	Medio	131
img_eb_003	Imagen eButiá	Botón de play en Pantalla de Proyecto	Medio	134
img_eb_016	Imagen eButiá	Actualización del estado del módulo	Medio	136
img_eb_019	Imagen eButiá	Orden de tabs en los selectores de módulos	Medio	136

Código	Componente	Error	Gravedad	Versión De Correcci
img_eb_020	Imagen eButiá	Error al utilizar el Suprimir	Catastrófico	138
img_eb_021	Imagen eButiá	Error al Suprimir capas superiores	Crítico	139
img_eb_012	Imagen eButiá	Manejo de concurrencia en Proxy	Crítico	140
img_eb_018	Imagen eButiá	Quitar botón de quitar módulo	Medio	141
img_eb_017	Imagen eButiá	Duplicación entrada de proyectos guardados	Medio	142
img_eb_022	Imagen eButiá	Duplicación entrada de proyectos guardados con el mismo nombre	Medio	142
ins_xo_003	Instalador XO	No se instala el bobot-server con el instalador.	Menor	–
ins_xo_004	Instalador XO	El bobot-server ejecutado no se detiene.	Menor	–
doc_na_001	Documento <Manual de Usuario>	El combo box de los módulos es el de la derecha.	Menor	1.2, pag. 10
doc_na_002	Documento <Manual de Usuario>	Error en término “chequeado”	Menor	1.2, pag. 13
img_eb_013	Imagen eButiá	Bloqueo de morphs	Menor	113
img_eb_015	Imagen eButiá	Texto de función para los bloques suprimir	Medio	113

7.6. Errores Conocidos

En esta sección se listan aquellos errores que no pudieron ser corregidos dentro de los plazos de este proyecto y quedarán para versiones posteriores del sistema.

Código	Error	Gravedad
img_eb_013	Bloqueo de morphs	Menor
img_eb_023	Error al abrir un proyecto del listado cuando se borro el archivo	Menor
img_eb_024	Categoría eButiá esta duplicada en el visor de módulo	Menor

Como se puede observar, los errores no corregidos son todos de una gravedad menor, que no causa un mayor perjuicio al sistema.

Parte III
Experimentos y Resultados

8. Estimación del tiempo de ejecución

La variación de subsumption implementada requiere que una capa, en lugar de liberar un recurso (ya sea actuador o sensor), reafirme su acción al menos cada un determinado tiempo. En caso de no hacerlo, puede perder el control sobre este con una capa de menor prioridad. Con el fin de estimar el tiempo máximo que tiene una aplicación para realizar esta acción, se utiliza el código mostrado en el siguiente cuadro, que permite calcular el tiempo de ejecución de un determinado bloque de código:

```
Time millisecondsToRun: [<<comportamiento>>]
```

Cuadro 7: Cálculo del tiempo de un bloque de código.

Para poder realizar las pruebas más fehacientemente, se creará un comportamiento con más de una capa programados mediante bloques de código. Luego se editaran los bloques para que cada uno de ellos muestre en una terminal el tiempo de ejecución de la siguiente manera:

```
Transcript
    show:
        'capaN',
        (
          Time millisecondsToRun: [<<comportamiento generado por bloques>>]
        ) asString;
    cr.
```

Cuadro 8: Cálculo del tiempo de un bloque de código.

Luego se estudiarán los resultados obtenidos.

8.1. Prueba realizada

Para realizar la prueba se realizó un comportamiento que resuelve un seguidor de líneas que se detiene al encontrarse en un círculo negro. Con este fin se implementaron 4 capas descriptas a continuación, donde números más pequeños refieren a capas de menor prioridad.

1. Ir para adelante: esta capa tiene el comportamiento básico de ir únicamente para adelante.

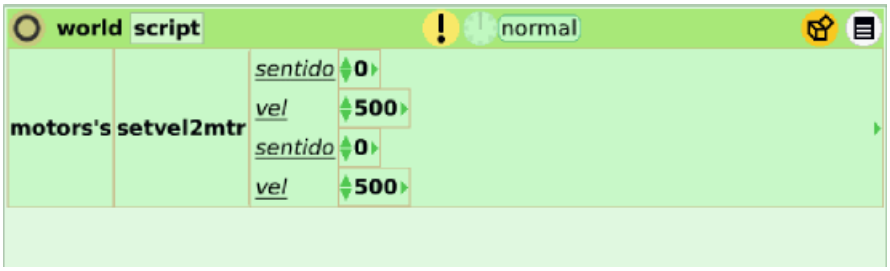


Figura 52: Código de capa Ir Para Adelante.

2. Girar a la derecha: esta capa se activa si el sensor derecho detecta la línea negra, y gira el robot a la derecha para volver a su recorrido.

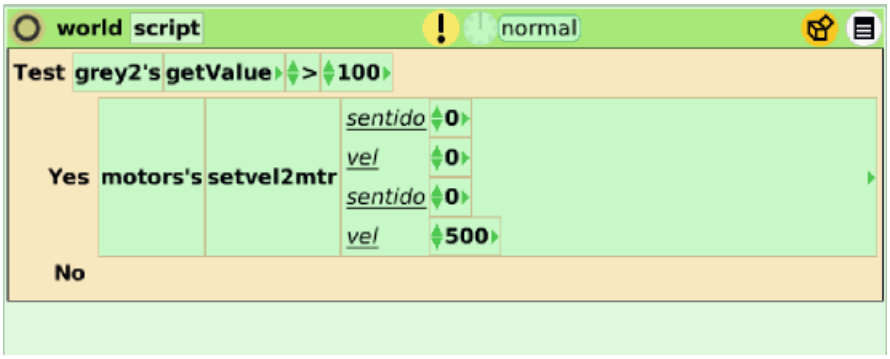


Figura 53: Código de capa Girar a la Derecha.

3. Girar a la izquierda: esta capa se activa si el sensor izquierdo detecta la línea negra, y gira el robot a la izquierda para volver a su recorrido.

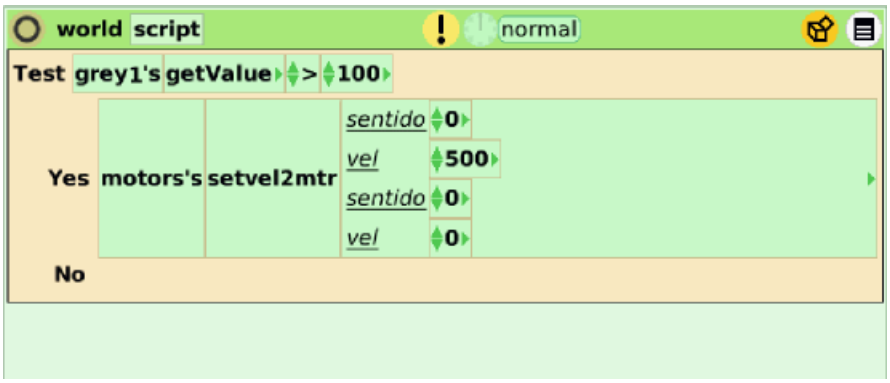


Figura 54: Código de capa Girar a la Izquierda.

4. Detenerse: esta capa se activa si ambos sensores (izquierdo y derecho) detectan la línea negra, determinando que el robot llegó al final del recorrido, y detiene completamente el robot. Observar que este comportamiento como es de mayor prioridad suprime a los anteriores, evitando que se ejecuten.

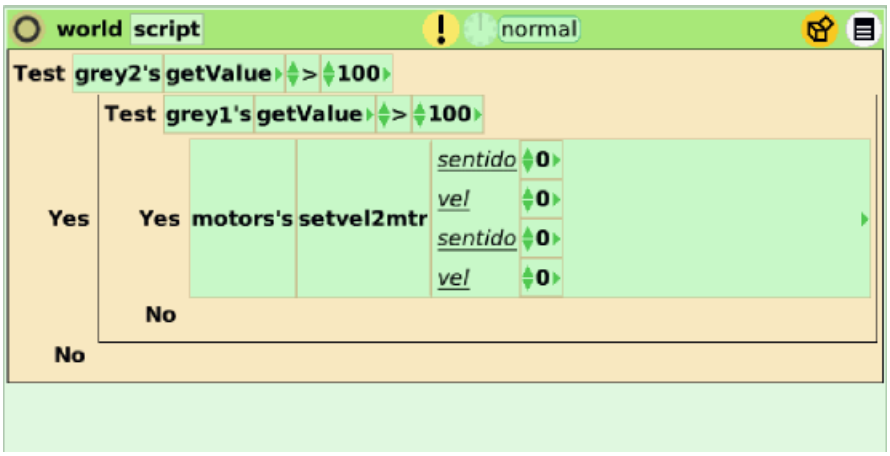


Figura 55: Código de capa Detenerse.

8.2. Resultados obtenidos

Aclaraciones sobre los resultados obtenidos:

- Las medidas tomadas refieren a todos los ciclos realizados por cada comportamiento. Se descartaron de cada uno, aquellos ciclos en los que los comportamientos no estaban "activos". Es decir que, si tomamos como ejemplo "Girar a la derecha", este verifica primero si el sensor se encuentra sobre la línea para realizar o no la acción de girar el robot. En caso de que la verificación de negativa, no se realizará ninguna acción en este ciclo. Esto evita que se realice la comunicación con la plataforma provocando que el tiempo de ejecución sea considerablemente menor. Como el tiempo calculado refiere a las reservas de módulos y estas se realizan sólo cuando se ejecuta la acción, se descartaran para el cálculo los menores tiempos obtenidos.
- Además, como se escogió un sistema que tiene capas sin alta complejidad, habría que sumarle a cada uno de los resultados un pequeño valor para asegurarnos que se ejecuten dentro del tiempo recomendado, comportamientos con más capas ejecutando en paralelo o con capas más complejas, pero que respeten el comportamiento reactivo. Sin embargo, como se está utilizando la consola Transcript para mostrar los resultados, esta ya añade un margen mayor de ejecución para comportamientos normales, por lo tanto se toman los resultados obtenidos como válidos.
- Las pruebas se realizaron en una computadora portátil XO-1.5 con el sistema operativo GNU/Linux OLPC OS 13.1.0 (build 30).

Los resultados obtenidos fueron los mostrados en el siguiente cuadro:

Capa	Tiempo mínimo	Tiempo máximo	Tiempo promedio
1. Ir para adelante:	50ms	60ms	52ms
2. Girar a la derecha:	67ms	90ms	85ms
3. Girar a la izquierda:	70ms	89ms	86ms
4. Detenerse:	73ms	95ms	95ms

Cuadro 9: Tiempos obtenidos en la estimación del tiempo de reserva

Como todos los tiempos máximos obtenidos son cercanos al tiempo promedio, habilita a tomar una aproximación al máximo de estos, como valor válido de cota de reserva de un módulo actuador o sensor por una determinada capa. Es por este motivo que se fijó el tiempo de reserva en 100ms.

Realizando pruebas con valores considerablemente superiores, del orden de los 500ms, el robot tenía una reacción lenta. Tomando como ejemplo una capa superior como una de las del control de giro, se podía observar como tomaba rápidamente el control por su nivel de capa, pero su tiempo de reserva provocaba que el robot se mantenga girando por demasiado tiempo perdiendo la línea.

Luego se probó con valores menores, con los cuales su comportamiento se hacía más inestable. Se podía observar que el robot, detectaba rápidamente

cuando había que girar; pero, como la capa "Ir para adelante" tenía un tiempo menor, en medio del giro tomaba posesión de los motores. Esto provocaba que se moviera en línea recta para adelante antes de terminar de girar.

Finalmente, se realizaron pruebas con el valor obtenido para el tiempo de reserva. Con éste, el robot tuvo un comportamiento reactivo como el esperado por el programa realizado.

9. Conclusiones y trabajo a futuro

9.1. Conclusiones

Este proyecto planteó una serie de desafíos que van desde la comprensión de las distintas arquitecturas reactivas a su implementación de forma fácilmente comprensible para estudiantes liceales.

Una de las características particulares de este proyecto es la elaboración de un software con fines didácticos, orientado a usuarios cuya experiencia con la computación y más aún con la programación es limitada o nula. Este software debía permitir a estudiantes liceales con el apoyo de docentes, el desarrollo de un software de control de forma autónoma del robot Butiá siguiendo una arquitectura del paradigma reactivo.

Esta particularidad llevó a realizar un extenso estudio de las distintas experiencias que existían de inserción de la robótica como complemento de la educación. Luego se estudiaron distintas arquitecturas del paradigma reactivo con el objetivo que los estudiantes puedan desarrollar una gran variedad de comportamientos, comprendiendo la estructura utilizada. Finalmente, con el fin de aprovechar las ventajas proveídas actualmente por distintos entornos de desarrollo orientados a la educación, se estudiaron una gran variedad de estos seleccionando Etoys para realizar una extensión que permita cumplir con los objetivos planteados. Debido a los motivos aquí explicitados el estudio del estado del arte, llevó un tiempo más extenso que el previsto inicialmente, pero con el cual se ha podido reafirmar la orientación a arquitecturas reactivas, así como también definir más claramente los objetivos del proyecto.

Se utilizaron distintas herramientas para poder llevar a cabo el proyecto. Para poder versionar el sistema y trabajar de forma paralela en el desarrollo, se utilizó Montichello Browser; y con el fin de llevar registro de las distintas actividades realizadas durante el proyecto, así como también dar difusión al mismo, se utilizó Media Wiki. Estas herramientas facilitaron al equipo el desarrollo del sistema.

9.1.1. Características de la Aplicación

A continuación se describen las distintas propiedades que caracterizan al sistema:

Multi-Plataforma:

- El sistema desarrollado ejecuta sobre una máquina virtual llamada SqueakVM. Esta característica la hace independiente del sistema operativo, siendo ejecutable en cualquier sistema sobre el cual se ejecute la plataforma. A su vez, como la máquina virtual es de código abierto y con una licencia poco restrictiva, permite su compilación a distintos sistemas operativos ampliando las posibilidades.

Gratuito:

- El sistema desarrollado es completamente gratuito desde la máquina virtual, pasando por la aplicación Etoys y la extensión desarrollada. Esto facilita la distribución y adquisición por parte de todos los estudiantes del país, sin importar su poder adquisitivo, ya que tanto los PCs como el acceso a internet para obtener la aplicación les fue otorgado por el Plan Ceibal.

Código Abierto:

- El código generado se mantiene dentro de la imagen que corre en la máquina virtual. Sin embargo este queda accesible para ser extendido y, en caso de ser necesario, corregido por toda aquella persona que lo crea necesario; sin existir restricciones en el licenciamiento que lo prohíban.

Desarrollo guiado:

- El desarrollo de los comportamientos es intuitivamente guiado para utilizar correctamente la arquitectura seleccionada, pudiendo agregar, quitar y ordenar capas, así como también inhibir o suprimir comportamientos inferiores.

Constructivo:

- La arquitectura seleccionada tiene la propiedad de generar comportamientos complejos a partir de comportamientos más simples. Siendo sencilla la extensión de estos, para generar otros más complejos.

Facilita la verificación:

- Otro aporte brindado por el conjunto de la arquitectura seleccionada junto con la implementación realizada, es facilitar la verificación. Se debe a que se fragmenta el comportamiento complejo en otros más simples, y la aplicación permite desactivar capas, dejando activas solo aquellas que se desea utilizar. Esto permite comprobar más fácilmente si se activan los comportamientos cuando deben, y si realizan la función deseada.

No invasivo:

- La aplicación no genera código invasivo, es decir que el complemento creado no modifica funcionalidades básicas del entorno. Además, se provee un acceso al sistema Etoys para poder continuar desarrollando aplicaciones con este.

Amigable y usable:

- Al ser desarrollado sobre Etoys, si bien agrega componentes visuales, no quita amigabilidad al momento del desarrollo, ya que los componentes añadidos únicamente guían al usuario al desarrollo del software sin quitar funcionalidades, permitiendo además la programación orientada a bloques

Didáctico:

- Una característica de Etoys que es mantenida por eButiá, es la posibilidad de ver el código desarrollado mediante programación con bloques, así como también modificarlo o desarrollar directamente escribiéndolo. Esto permite que la transición de la programación por bloques a la programación por código escrito sea facilitada considerablemente.

Módulos Cargados Dinámicamente:

- Los módulos conectados al robot, así como sus funciones son detectados automáticamente por la aplicación a través del bobot-server. Esto implica que nuevos módulos o funcionalidades cuyos parámetros o resultados sean similares a los descriptos anteriormente no requieren de modificaciones en el sistema.

Independencia del Hardware:

- El utilizar el robot a través de la interfaz ofrecida por el bobot-server, permite que el sistema se independice del hardware utilizado. Es decir que si se implementa una interfaz similar para otros robots, el sistema sería capaz de controlarlos. De esta forma se puede conectar a simuladores de software que implementen la interfaz, como el eButiá-Simulador sin notarlo.

Independencia de la conexión:

- Debido a la metadata utilizada, el algoritmo implementado para ejecutar la arquitectura es fácilmente reutilizable para ejecutar otros tipos de robots o con otros tipos de conexiones, implementando las clases correspondientes. Esto permite independizar más aún el sistema del robot utilizado.

Motivadora:

- La aplicación fue desarrollada utilizando el paradigma reactivo. Está fuertemente basada en la reacción a estímulos, que según Papert, genera una mayor incertidumbre en el resultado, que es acompañada por una mayor motivación llevada por la curiosidad.

9.1.2. Aportes de la aplicación

Cumpliendo con los requerimientos observados, esta aplicación añade al conjunto de las aplicaciones ya existentes distintos aportes, manteniendo muchas de las ventajas ya contenidas en otras aplicaciones, como la programación orientada a bloques, o el cargado dinámico de módulos.

Los principales aportes de la aplicación son mostrados a continuación:

Escalable:

- Permite generar aplicación a través de programación orientada a bloques fácilmente escalable de forma estructurada utilizando una arquitectura probada como Subsumption.

Didáctico:

- Aprovecha las ventajas provistas por Etoys para eliminar la complejidad inducida por el cambio de la programación mediante bloques a la escritura de código.

Orienta a la programación robótica:

- La aplicación sigue una de las arquitecturas del paradigma reactivo para generar comportamientos robóticos. Esto permite a los estudiantes aprender un método de estructuración de programas utilizado frecuentemente en el desarrollo de este tipo de aplicaciones. De esta forma los estudiantes se familiarizan con las buenas prácticas de programación utilizando complementos distintos para encapsular las distintas responsabilidades.

9.2. Trabajo a Futuro

9.2.1. Mejoras:

Como trabajo a futuro, existen distintas extensiones que se pueden hacer a la aplicación, algunos de ellos son mostrados a continuación:

Creación de pantalla de preferencias:

- Un interesante complemento a esta aplicación sería una pantalla donde se pueda cambiar distintas configuraciones personalizadas al usuario. Como pueden ser el directorio por defecto donde se almacenan los proyectos, el tiempo de reserva utilizado por el algoritmo, entre otros.

Apertura de capas de forma independiente:

- Una de las mejoras que puede realizar un gran aporte a la aplicación es el poder exportar e importar capas de forma independiente de forma sencilla. Esto permitiría a los estudiantes compartir las distintas capas. Además, los estudiantes podrían, dividirse en grupos para resolver los distintos problemas, realizando de forma simple y distribuida aplicaciones más complejas.

Creación de drivers para otros robots:

- Otra mejora de considerable aporte, es el implementar otros drivers, con sus correspondientes componentes de interfaz, para poder utilizar los comportamientos desarrollados en robots de distintas características.

Implementación de otras Arquitecturas:

- Se podría extender la aplicación para que a la hora de crear los proyectos, se elija el tipo de arquitectura a implementar y se desarrolle el comportamiento en base a esta.

Verificación permanente de módulos conectados:

- Se podría agregar a la aplicación la capacidad de detectar cuáles son los módulos que están conectados al robot en cada momento. Con este fin, se podría crear un thread que mediante polling realice esta acción y actualice el estado de los mismos. De esta forma se detectaría en poco tiempo nuevos módulos conectados y módulos que han dejado de estar disponibles.

Estimación dinámica del tiempo de ejecución:

- El tiempo de ejecución, utilizado para el bloqueo de sensores y reserva de actuadores se podría calcular dinámicamente a medida que ejecuta la aplicación. De esta forma, el tiempo para una determinada aplicación, en

un determinado instante, se aproximaría al óptimo para dicha situación. Es decir, que no se exceda el tiempo de ejecución por un margen muy grande, de la capa de mayor duración; ni que pierdan rápidamente el control de sensores y actuadores con las capas inferiores.

9.2.2. Aplicaciones derivadas:

La aplicación desarrollada ha resuelto, por intermedio de los distintos prototipos, distintas dificultades que se habían identificado. Haciendo uso de estas soluciones y fuera del alcance del proyecto, se ha desarrollado una aplicación que permite simular la actuación del robot, utilizando una interfaz similar a la del bobot-server, llamada eButiá-Simulador.

Dentro de las soluciones provistas por esta aplicación respecto a la utilización de Smalltalk y de Etoys podemos encontrar tres claros ejemplos que sirvieron de base para la implementación del simulador. Los mismos son descriptos a continuación:

- **Creación de threads:** para realizar la comunicación de forma paralela al control del robot en la interfaz, se utilizó el manejo de threads investigado en este proyecto para realizar el paralelismo en la ejecución de las distintas capas.
- **Comunicación por Sockets:** para exponer la misma interfaz que el bobot-server, se utilizó la solución obtenida en este proyecto para comunicarse con el bobot-server.
- **Modificación dinámica de la interfaz al cargar la extensión:** al momento de cargar la actualización, debía modificarse la interfaz. La solución utilizada es la proveída por este proyecto, que utiliza el método initialize que poseen los objetos clase y que es ejecutado al momento de crear la clase. Con este fin se creó, al igual que en este proyecto, una clase Launcher, que modifica la interfaz con los complementos deseados al momento de la carga.

Referencias

[1] Plugins tortugarte. http://wiki.sugarlabs.org/go/Activities/TurtleArt/Plugins. Ultima visita: 22/05/2013.

[2] Alejandro Achkar, Andrés Margalef. Documento de diseño-subsumption. 2012. Proyecto 'IDE de programación orientado al desarrollo de arquitecturas robóticas basadas en comportamientos'.

[3] Alejandro Achkar, Andrés Margalef. Documento de especificación de requerimientos. 2012. Proyecto 'IDE de programación orientado al desarrollo de arquitecturas robóticas basadas en comportamientos'.

[4] Alejandro Achkar, Andrés Margalef. Documento de prototipos. 2012. Proyecto 'IDE de programación orientado al desarrollo de arquitecturas robóticas basadas en comportamientos'.

[5] Alejandro Achkar, Andrés Margalef. Documento del estado del arte. 2012. Proyecto 'IDE de programación orientado al desarrollo de arquitecturas robóticas basadas en comportamientos'.

[6] Apache. Licencia apache 2.0. http://www.apache.org/licenses/LICENSE-2.0. Ultima visita: 22/05/2013.

[7] Rodney A. Brooks. A robot that walks; emergent behaviors from a carefully evolved network. *Neural Computation*, 1:253–262, 1989.

[8] Jonathan Connell. A colony architecture for an artificial creature. 1989.

[9] Dan Schwartz. Ghost in the machine. http://www.papert.org/articles/GhostInTheMachine.html, 1999. Ultima visita: 22/05/2013.

[10] Fundación Omar Dengo. Robotica Educativa. http://www.fod.ac.cr/robotica/index.htm, June 2011. Ultima visita: 28/10/2012.

[11] Fundación Omar Dengo and Ministerio de Educación Publica de Costa Rica. Robot-Blogs. http://www.fod.ac.cr/robotblogs, June 2011. Ultima visita: 22/05/2013.

[12] GIRA. Physical etoys. http://tecnodacta.com.ar/gira/. Ultima visita: 22/05/2013.

[13] Google. App inventor. http://www.appinventorbeta.com/, 2010. Ultima visita: 28/10/2012.

[14] LEGO Company. RoboLab. http://www.robolabonline.com/home, June 2011. Ultima visita: 22/05/2013.

[15] Microsoft. Patron mvc. http://msdn.microsoft.com/en-us/library/ff649643.aspx. Ultima visita: 17/05/2013.

[16] MIT. Comunidad scratch. http://wiki.scratch.mit.edu/wiki/Scratch_Community. Ultima visita: 22/05/2013.

[17] MIT. Licencia mit. http://www.opensource.org/licenses/mit-license.php. Ultima visita: 22/05/2013.

[18] MIT. Proyectos compartidos por mes. http://stats.scratch.mit.edu/community/activitybytime.html. Ultima visita: 22/05/2013.

[19] MIT. Proyectos creados por edad. http://stats.scratch.mit.edu/community/activitybyage.html. Ultima visita: 22/05/2013.

[20] MIT. Scratch code. Ultima visita: 22/05/2013.

[21] MIT. Scratch download. http://info.scratch.mit.edu/es/Scratch_1.4_Download. Ultima visita: 22/05/2013.

[22] MIT. Scratch license. http://info.scratch.mit.edu/Scratch_License/1.4License. Ultima visita: 22/05/2013.

[23] MIT. Scratch on linux. http://info.scratch.mit.edu/Scratch_on_Linux. Ultima visita: 22/05/2013.

[24] MIT. Scratch reference guide. http://info.scratch.mit.edu/es/Support/Reference_Guide_1.4. Ultima visita: 28/10/2012.

[25] Plan Ceibal. Portal plan ceibal. http://ceibal.edu.uy/Portal.Base/Web/verContenido.aspx?PVW=1&ID=204269, 2011. Ultima visita: 28/10/2012.

[26] Proyecto Butia. Butia manual de usuario. http://www.fing.edu.uy/inco/proyectos/butia/files/butia_manual_usuario.pdf, 2011. Ultima visita: 22/05/2013.

[27] Proyecto Butiá. Sitio Oficial del Proyecto Butiá. http://www.fing.edu.uy/inco/proyectos/butia/, 2011. Ultima visita: 22/05/2013.

[28] Proyecto Butiá. Wiki proyecto butiá. http://www.fing.edu.uy/inco/proyectos/butia/mediawiki/index.php/P%C3%A1gina_principal, 2011. Ultima visita: 22/05/2013.

[29] Python. Python. http://python.org. Ultima visita: 22/05/2013.

[30] quantcast. Visitas del sitio scratch. http://www.quantcast.com/scratch.mit.edu. Ultima visita: 22/05/2013.

[31] Seymour Papert. An evaluative study of modern technology. http://www.papert.org/articles/AnEvaluativeStudyofModernTechnology.html, June 1976. Ultima visita: 22/05/2013.

[32] Seymour Papert. Event programming part 1. http://www.papert.org/articles/costa_rica/EventProgrammingPart1.html, 1980. Ultima visita: 22/05/2013.

[33] Elizabeth Sklar and Amy Eguchi. Learning while teaching robotics. 2003.

[34] Squeak. Licencia squeak. http://squeak.org/SqueakLicense/. Ultima visita: 22/05/2013.

[35] Squeak. Squeak mailing list. http://wiki.squeak.org/squeak/608. Ultima visita: 22/05/2013.

[36] Sumo Fing. Desafío sumo 2011. http://www.fing.edu.uy/inco/proyectos/butia/files/desafio_basico_sumo_2011.pdf, 2011. Ultima visita: 22/05/2013.

A. Glosario

A.1. Actuador

Módulo conectado a un robot que permite realizar distintas modificaciones el ambiente, considerando al robot y su ubicación como parte de este.

A.2. Actuar

Acción realizada por un Actuador

A.3. Agente

Es una entidad que toma información de su entorno y la procesa de forma que pueda actuar de manera racional.

A.4. Clase

Definición de propiedades y comportamiento un objeto.

A.5. Comportamientos Robóticos

Es el conjunto de reglas que determinan las acciones que toma el robot en base a la información que posee.

A.6. Comunidad

En el contexto de una aplicación, se le llama comunidad al conjunto de usuarios, desarrolladores y toda persona relacionada con éste que pueda brindar ayuda técnica sobre el mismo.

A.7. Debugging

Acción realizada para corregir un determinado error en un sistema, en la cual se permite la ejecución paso a paso de este, verificando el contenido de las variables y objetos involucrados.

A.8. Disco Rígido

Unidad de almacenamiento físico de un computador.

A.9. EButiá

Extensión a Etoys desarrollada en este proyecto.

A.10. Entorno de Desarrollo

En el contexto del software, es una herramienta que facilita la generación en un determinado lenguaje de programación para desarrollar distintas aplicaciones.

A.11. Etoys

Entorno educativo de desarrollo de software basado en la programación por bloques.

A.12. Extensión

En el contexto del desarrollo de software, se dice extensión de un sistema X, a una modificación que se realiza en el mismo con el fin de añadir nuevas funcionalidades.

A.13. File System

Estructura de archivos y directorios compuesto por los datos almacenados en un PC.

A.14. Halo

Botones con propiedades modificables de los Morphs que se muestran circundantes al mismo al realizar click derecho sobre estos.

A.15. IDE

Abreviatura del inglés "**I**ntegrated **D**evelopment **E**nvironment" refiere a Entorno de Desarrollo.

A.16. Inhibir

En el contexto de Subsumption, se llama inhibir a la acción que realiza una capa superior al sobrescribir el valor devuelto por una función de un sensor para que capas inferiores no puedan acceder al valor real de este.

A.17. Interfaz

En el contexto de objetos de software, es la definición común a distintos objetos que permite mantener comportamientos distintos accedidos desde métodos con la misma definición.

A.18. Licencia de Software

Contrato entre el titular o autor del software con el destinatario del mismo, que define los privilegios de este último sobre la aplicación.

A.19. Método

Funciones provistas por los objetos que almacenan el comportamiento de los mismos.

A.20. Módulo

Termino genérico con el que se hace referencia a actuadores y sensores indistintamente.

A.21. Model View Controller

Patrón de estructuración de software que presenta tres componentes que interactúan entre sí para desplegar la información en pantalla: Modelo, Vista y Controlador.

A.22. Morph

Clase básica que extienden todos los objetos visibles en pantalla de Etoys.

A.23. MVC

Abreviación de **M**odel **V**iew **C**ontroller.

A.24. Objeto

Entidad instanciada de una clase que posee tanto estado (propiedades) como comportamiento (métodos).

A.25. Observer

Patrón de diseño que resuelve la problemática en la que un objeto debe notificar a un conjunto de objetos "observadores" sobre sus cambios.

A.26. OLPC

Abreviación del proyecto One Laptop Per Child, que busca brindar a cada niño un computador portátil de bajo costo y con conexión a internet.

A.27. Paradigma Reactivo

Es una propuesta de desarrollo de comportamientos robóticos, que pretende generarlos simulando el comportamiento animal, es decir en base a reacciones y no a deliberación.

A.28. Patrón de Diseño

Guía de solución para un determinado problema conocido, a resolver en el diseño de un software.

A.29. Pippy

Entorno de desarrollo de software que permite programar código python y viene incluido en la XO.

A.30. Plan Ceibal

Una iniciativa que busca insertar la tecnología digital en la educación primaria y secundaria del Uruguay.

A.31. Plataforma Butiá

Plataforma acrílica con una placa de control que permite la conexión de distintos módulos sensores y actuadores, así como también exponer las funcionalidades de estos a través de una conexión con una pc. Generalmente tiene dos motores que permiten controlar dos ruedas laterales, pero existen versiones con más.

A.32. Proyecto Butiá

Proyecto lanzado en el año 2010 que tiene como fin acercar a alumnos escolares y liceales a la programación de comportamientos robóticos.

A.33. Python

Lenguaje de programación de scripting.

A.34. Renderizar

Acción de procesar componentes lógicos para crear una imagen visual.

A.35. Robot Butiá

Conjunto de componentes que conforman un robot, compuesto por una plataforma Butiá, una XO y el software de control generado.

A.36. Scroll

Elemento de interfaz gráfica mostrado como una barra lateral a un componente, utilizado frecuentemente para visualizar una parte oculta del contenido del mismo.

A.37. Selector

Componente de interfaz gráfica que permite el listado y selección de distintos objetos.

A.38. Sensar

Acción realizada por un Sensor

A.39. Sensor

Módulo conectado a un robot que permite obtener información del ambiente, considerando al robot y su ubicación como parte de este.

A.40. Singleton

Patrón de diseño en el cual existe una única instancia objeto, de una determinada clase en todo el sistema.

A.41. Smalltalk

Es un conjunto de herramientas para el desarrollo de software compuesto de un lenguaje (del mismo nombre), un entorno de desarrollo y distintas librerías con funcionalidades.

A.42. Strategy

Patrón de diseño en el que un objeto tiene la capacidad de elegir un determinado algoritmo, de un conjunto, para resolver una tarea concreta. Al algoritmo elegido se le conoce como estrategia.

A.43. Subsumption

Es una arquitectura de software perteneciente al paradigma reactivo que divide el comportamiento complejo en subcomportamientos en forma de capas donde las capas superiores tienen prioridad sobre las inferiores; pudiendo suprimir las acciones de estas o inhibir sus entradas.

A.44. Sugar

Ambiente de escritorio educativo, creado por SugarLabs, orientado a los niños.

A.45. Suprimir

En el contexto de Subsumption, se llama suprimir a la acción que realiza una capa superior al reclamar un actuador evitando que las capas de menor prioridad puedan hacer uso de este.

A.46. Testing

Acción de verificar el correcto funcionamiento de un sistema.

A.47. UDELAR

Abreviatura que referencia a la Universidad de la República del Uruguay.

A.48. Variable

Propiedades tanto de objetos como de clases que almacenan datos modificables y definen el estado de los mismos.

A.49. XML

Formato de almacenamiento de datos en un archivo, organizados en forma de árbol n-ario mediante tags.

A.50. XO

Computador portátil de bajo costo, diseñado por OLPC, distribuido con el Plan Ceibal.

B. Herramientas para la gestión del proyecto

B.1. Astah

Es una herramienta para generar diagramas de distintos tipos, entre ellos UML. Fue utilizada para generar algunos de los diagramas del sistema.

B.2. Dia

Es otra herramienta para generar diagramas de distintos tipos, entre ellos UML. Fue utilizada para generar algunos de los diagramas del sistema.

B.3. DropBox

Herramienta de almacenamiento de archivos en línea. Posee la capacidad de ser montado como una carpeta más del sistema. Los cambios realizados localmente a los archivos en dicha carpeta son subidos automáticamente al servidor. Esta herramienta se utilizó para almacenar tanto el código generado por el Montichello Browser como los documentos y sus componentes.

B.4. Gantt Project

Permite gestionar recursos de proyectos mediante diagramas de gantt. Fue utilizada para seguir los tiempos utilizados en las distintas etapas del proyecto.

B.5. JabRef

Gestor de referencias bibliográficas que permite añadir a un archivo de bibliografía distintas referencias. Es capás de referenciar distintos tipos de documentos, como pueden ser artículos, libros y sitios web entre otros. Se utilizó para generar y administrar los distintos archivos de bibliografía que acompañan a los documentos generados.

B.6. Lyx

Procesador de texto sencillo que genera código latex. Este posee distintas herramientas comunes a los procesadores de texto avanzados, como puede ser el corrector ortográfico, inserción de distintos tipos de índices, así como también la facilidad de insertar imágenes y figuras entre otros. Además, posee la capacidad de insertar referencias en base a un archivo de bibliografía, y generar un índice bibliográfico con este. Esta herramienta fue utilizada para generar los distintos documentos.

B.7. Media Wiki

Es una wiki que permite ser instalada y personalizada fácilmente. Se instaló en el servidor de la facultad y se utilizó para mantener actualizadas las novedades y documentos durante el proyecto.

B.8. Montichello Browser

Es un gestor de versiones de código de smalltalk. Está incluido dentro de Etoys y permite almacenar las modificaciones hechas a las clases de la imagen original, así como también las nuevas clases generadas siguiendo un patrón de generación de código. Para poder diferenciar que código deseamos que administre, las nuevas categorías de sistema añadidas deben comenzar con el nombre del repositorio. Si deseamos que se administre código de una clase que ya venía incluida debemos cambiar el método modificado (o añadido) a un protocolo que tenga por nombre * seguido por el nombre del repositorio. Esta fue la herramienta utilizada para gestionar las versiones del código generado.

B.9. Pinta

Es una herramienta para generar dibujos teniendo la capacidad de poner nuestra imagen en distintas capas en un lienzo. Fue utilizado para generar algunos de los dibujos mostrados en los proyectos.

B.10. Gimp

Herramienta para generar dibujos más compleja que Pinta y fue utilizada para editar de forma más compleja algunas de las imágenes incluidas en el documento.