INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

ESCOM

Trabajo Terminal

“MECANISMO PROGRAMABLE PARA NIÑOS”

2014-A070

Presentan

Aldama Pérez Cristopher

Zavala Ventura Miguel Ángel

Directores

M.C. Saucedo Delgado Rafael Norman

Enero 2015

INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO
SUBDIRECCIÓN ACADÉMICA

8/12/2015

Documento Técnico

“Mecanismo programable para niños”

Presentan
Aldama Pérez Cristopher
Zavala Ventura Miguel Ángel

Directores
M.C. Saucedo Delgado Rafael Norman

RESUMEN

En este reporte se muestran los alcances del proyecto de trabajo terminal, haciendo énfasis en la parte de diseño y desarrollo. Este proyecto consta de dos grandes subsistemas: Un lenguaje de programación visual llamado Olinki que permite la creación de programas simples con una interfaz de arrastrar y soltar. Y una pequeña computadora controladora que ejecutar los programas olinki y donde se conectan los sensores y motores. Con el fin de acercar temas de computación y programación a niños en edad de educación primaria.

Palabras clave: Lenguaje, pedagogía, robótica, programación, juego, juguete

ESCUELA SUPERIOR DE CÓMPUTO
SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE FORMACIÓN INTEGRAL E INSTITUCIONAL

COMISIÓN ACADÉMICA DE TRABAJO TERMINAL

México, D.F. a 18 de enero de 2016

DR. FLAVIO ARTURO SÁNCHEZ
GARFIAS PRESIDENTE DE LA
COMISIÓN ACADÉMICA DE
TRABAJO TERMINAL
P R E S E N T E

Por medio del presente, se informa los alumnos que integran el **TRABAJO TERMINAL: 2014-A070**, titulado "Mecanismo programable para niños" concluyeron satisfactoriamente su trabajo.

Los discos (DVDs) fueron revisados ampliamente por su (s) servidor (a/es/as) y corregidos, cubriendo el alcance y el objetivo planteados en el protocolo original y de acuerdo a los requisitos establecidos por la Comisión que Usted preside.

ATENTAMENTE

M. en C. Rafael Norman Saucedo Delgado

0.1. Advertencia

"Este documento contiene información desarrollada por la Escuela Superior de Cómputo del Instituto Politécnico Nacional, a partir de datos y documentos con derecho de propiedad y por lo tanto, su uso quedará restringido a las aplicaciones que explícitamente se convengan."

La aplicación no convenida exime a la escuela su responsabilidad técnica y da lugar a las consecuencias legales que para tal efecto se determinen.

Información adicional sobre este reporte técnico podrá obtenerse en:
La Subdirección Académica de la Escuela Superior de Cómputo del Instituto Politécnico Nacional, situada en Av. Juan de Dios Bátiz s/n Teléfono: 57296000, extensión 52000.

0.2. Agradecimientos

Aprovecho estas líneas para dar las gracias a mi familia, a mi niñito Santi, a mi esposa Monse que me ayudaron a mantenerme enfocado para alcanzar mis metas personales, además a El Explicador por regresarme la curiosidad hacia las ciencias.

Cristopher Aldama Pérez

A mi familia, los amigos de vida que hice en MUTEC, los ingeniokids, a todos los invisibles. Hay muchas personas de ESCOM que agradecer, hasta el sujeto de los agradecimientos de arriba. Maestra Martha, no me olvido de usted. También gracias por aparecer Brenda.

Miguel Ángel Zavala Ventura

Índice general

0.3. Introducción

En el campo de la educación básica, tanto en escuelas públicas como privadas, una de las principales preocupaciones es enseñar conceptos relacionados con la tecnología, debido a la exposición que la sociedad tiene con ella es cada vez mayor, y para ello desarrollan competencias en las que motivan el conocimiento, uso y aplicación de la computadora en las tareas de la vida diaria; sin embargo, aunque en el mercado existen diversos materiales para su enseñanza, es difícil encontrar alguno que mantenga el interés de los niños pequeños y se ajuste al ritmo en el que absorben las ideas.

Actualmente los dispositivos en los que se apoya la enseñanza de estos conceptos son circuitos básicos, que están listos para armarse, sin embargo limitan la interacción a la observación de su funcionamiento, lo cual pierde trascendencia e interés al poco tiempo. También existen sistemas más robustos, mecanismos controlados por un programa de computadora, para el que se necesita un nivel de abstracción mayor, pues requiere de la comprensión de conceptos de matemáticas y lógica.

La propuesta de este proyecto consiste en crear un dispositivo cuyo funcionamiento pueda ser aprendido de forma gradual haciendo uso del juego y para ello se debe analizar, diseñar, probar e implementar un sistema mecánico programable que sirva como material auxiliar en la enseñanza de conceptos, en el área de la lógica y la programación de sistemas de cómputo, enfocado en niños de escuelas primarias, con edad de entre 7 y 11 años de edad.

El modelado del desarrollo del proyecto se realizó tomando como referencia el ciclo de vida que tiene cualquier sistema, puesto que el funcionamiento general no se encuentra basado en un conjunto de clases u objetos, sin embargo, en el módulo de

diseño se encuentran documentadas clases que llega a ocupar la interfaz.

0.4. Resumen

Mecanismo programable para niños es un sistema que consta de un lenguaje gráfico de programación y un bloque micontrolador al que se le pueden conectar sensores (contacto, luz y temperatura) y actuadores (motores), que sirve como apoyo a la enseñanza de programación y robótica a niños de primaria de entre 7 y 11 años.

Palabras clave: Lenguaje, Lógica, Sensor, Actuador, Robótica, Programación, Instrucción, Educación, Pedagogía.

Capítulo 1

Análisis

En este apartado se plantea el problema en el que este proyecto se enfoca, así como determinar los objetivos específicos, la justificación y se enlistan los resultados esperados.

1.1. Planteamiento del problema

La tecnología va adquiriendo día a día un lugar más importante en el desempeño de las tareas diarias, que van desde las compras en el supermercado, operaciones bancarias, hasta entretenimiento y actividades lúdicas. La computadora y sus aplicaciones tienen un rol central en el desarrollo de la sociedad, es por eso que escuelas (en especial las de educación primaria) buscan herramientas que ayuden a sus alumnos a tener un conocimiento adecuado sobre las ciencias de la computación, que sirvan como base en el desarrollo de la persona.

Este trabajo terminal, se presenta como una herramienta para la educación primaria que los maestros pueden aprovechar para facilitar la enseñanza de conceptos elemen-

tales sobre el funcionamiento básico de las computadoras así como despertar el interés de los alumnos en la manipulación responsable y razonada de la tecnología.

1.2. Objetivos

En seguida se presentan los objetivos que pretende alcanzar el proyecto Olinki, desde sus aspectos generales hasta aquellos más específicos que sirvan para alcanzarlos.

1.2.1. Objetivo General

Analizar, implementar y diseñar un sistema de cómputo que funcione como herramienta de enseñanza de conceptos lógicos. Dicha herramienta será un conjunto de componentes tanto en hardware como en software, que permita a niños de entre 7 y 11 años crear programas simples usando íconos gráficos así como su compilación en una pequeña computadora de bajo costo (en cuanto recursos lógicos y físicos se refiere) que funja como controlador principal permitiendo ejecutar instrucciones para manipular sensores y motores.

1.2.2. Objetivos Específicos

- Crear un entorno de desarrollo integrado (EDI) con soporte para el circuito Olinki.
- Implementar un interprete del lenguaje de programación Olinki.
- Implementar una GUI (interfaz gráfica de usuario, por sus siglas en inglés) que funcione como lenguaje de programación inconográfico.

- Diseñar circuitos electrónicos que den soporte a los sensores de iluminación, contacto y temperatura, así como a los motores eléctricos.
- Definir ejemplos que muestren las capacidades del lenguaje de programación.
- Diseñar una carcasa que proteja los circuitos, así como el controlador principal.
- Realizar pruebas automatizadas que muestren fallas en el diseño del lenguaje de programación.

1.3. Justificación

Con la integración de las computadoras a nuestra vida diaria en forma de teléfonos celulares, relojes inteligentes, consolas de vídeo juegos, tabletas, computadoras personales, etcétera, ha surgido una corriente que propone la enseñanza de programación en escuelas de educación primaria como apoyo en el entendimiento de la manera en que funcionan las computadoras y sus aplicaciones, y que ha sido adoptada en países como Estonia (2012) e Inglaterra (2014), y otros que están haciendo planes para incluirla en su plan de estudios como Finlandia, EUA, Singapur, Dinámarca, Isreal y Australia [12]. El objetivo de exponer a los niños al uso de computadoras tan pronto como sea posible y desarrollar habilidades técnicas en ellos, es el de prepararlos en el mundo tecnológico en el que viven inmersos, además de alimentar su curiosidad en el área de ciencias de la computación con la meta de satisfacer la demanda de profesionales en el área.

Por ello se toma en cuenta que el acercamiento a la programación por niños de primaria, requiere de herramientas adecuadas, que simplifiquen el proceso de crear y usar un algoritmo para resolver un problema en específico así pues se buscan lenguajes de programación amigables con los niños pequeños como son: Alice[15], Scratch[10], Turtle[11] entre otros. Lenguajes que fueron creados para la educación y que hacen

uso de elementos gráficos para la creación de programas simples así como colores llamativos, sentencias simples, animaciones, etcétera, pero que sin embargo solo están disponibles en el idioma inglés o que no tienen manera de interactuar directamente con hardware.

Por otro lado, en nuestro país la Reforma Integral de Educación Básica (RIEB) anima a los docentes a hacer uso de la tecnología con la finalidad de reforzar las clases y consolidar los conocimientos adquiridos, enfocándose en las competencias de los alumnos, las estrategias tomadas por esta reforma son las de capacitar a los docentes en el uso de recursos multimedia, de medios de comunicación, el internet y creación de infraestructura como enciclomedia. Sin embargo no se hace mención de la enseñanza de temas o materias de las ciencias de la computación en las aulas, de tal manera que la tecnología puede ser usada como apoyo complementario en la enseñanza de las materias y cursos (aprender con tecnología) o como modelo pedagógico (aprender de la tecnología).

Se considera de gran importancia la elaboración de este proyecto ya que propone la realización de un lenguaje de programación simplificado, con elementos gráficos y en español el cual puede ser usado como herramienta en la enseñanza de los conceptos clave de las ciencias de la computación como son: creación de algoritmos y programación de computadoras, además de la experimentación incitando al usuario a diseñar, armar y mejorar sus propios diseños de software y hardware. Que complemente el modelo propuesto por la RIEB, poniendo como actor a la tecnología en este caso la computadora y sus aplicaciones.

1.3.1. Productos o Resultados Esperados

Al final el proyecto será un sistema computacional compuesto estos productos principales:

- Un circuito con soporte para entradas de valores analógicos.
- Un circuito con soporte para salida de valores analógicos y digitales (motores y leds).
- Un IDE para programar los circuitos mencionados.

1.4. Marco Teórico

La elaboración de este proyecto se apoya de la pedagogía ya que esta ciencia tiene como meta el estudio de la educación y la formación con el objeto de incorporar al individuo en la sociedad en la que se desenvuelve apoyándose de otras ciencias como son sicología, sociología, historia, entre otras. En el contexto de nuestro proyecto esta diciplina nos brinda herramientas para el análisis, diseño e implementación de nuestro proyecto en forma teorías del aprendizaje que se acomodan a los objetivos que se plantearon en el capítulo anterior.

1.5. Metodología Pedagógica

Antes de narrar el por qué seleccionamos la corriente pedagógica que escogimos valdría la pena señalar las dos opciones importantes con las que contábamos en un inicio, que son las corrientes socio-constructivista y socio-construccionista. A pesar de la semejanza de los dos nombres resulta un tanto distantes una de la otra por las sutiles diferencias que presentan en sus bases fundamentales; para comenzar, podemos decir

que el construccionismo es un paradigma que aparece posteriormente al constructivismo, y es precisamente porque deriva de éste último. A grandes rasgos, en el segundo se hace hincapié en que el individuo forma su conocimiento a partir de experiencias que tienen qué ver con el contexto social en el que se desarrolla, mientras que en el primero explica que el individuo formará su aprendizaje basado en experiencias sociales, es decir, un grupo de sujetos desarrollará el conocimiento en el que se basarán para generar el siguiente contexto 1 . En seguida detallaremos mejor sus diferencias.

El constructivismo tiene su base en la idea de que el conocimiento se escala, es decir, que una vez que se tiene aprendido algo o un acontecimiento fue significativo para una persona, ésta experiencia le servirá como fundamento para generar el conocimiento siguiente, de tal manera que se altere la estructura mental que se generó (modelo mental) para reacomodarse y fijar un nuevo concepto. El sujeto que se construye conocimiento a partir de este modelo hace que los nuevos modelos mentales adquiridos le resulten inherentes a su vida, puesto que lo que experimentó a lo largo de sus vivencias le permitió generar aprendizaje auto referenciado 2 3.

Como anticipamos, el construccionismo se basa en un sistema social. Hace referencia a que en una sociedad la retroalimentación entre sus personajes es obligada porque la comunicación entre ellos es la base para su desarrollo; dentro de uno de los conceptos que componen a esta tendencia, está la definición del establecimiento de las reglas de las dinámicas de interacción por el conjunto de sus individuos; otro idea que la compone es que en la realidad de un sujeto, su sistema de acciones es resultado de las interacciones que tiene con la sociedad en la que se desenvuelve[3]. Como vemos ahora, el paradigma socio-construccionismo antecede la relación entre personas a la individualidad y es así como cada sujeto modela su realidad. De este modo podemos analizar el aprendizaje significativo de un sujeto cuando a este se le coloca en un entorno social

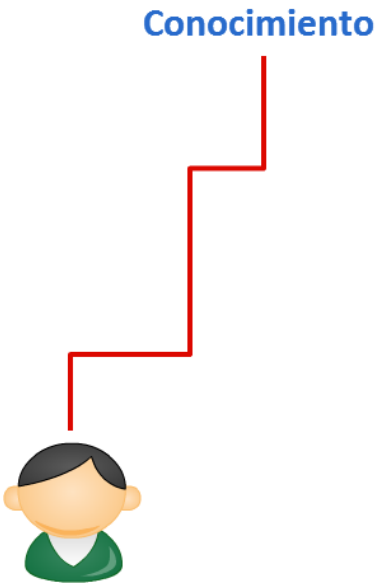

Figura 1.1: Escalación de conocimiento

con el que pueda interaccionar con otros individuos, con reglas a consensuar o las ya preestablecidas, para comenzar a experimentar la construcción de su conocimiento 2 3.

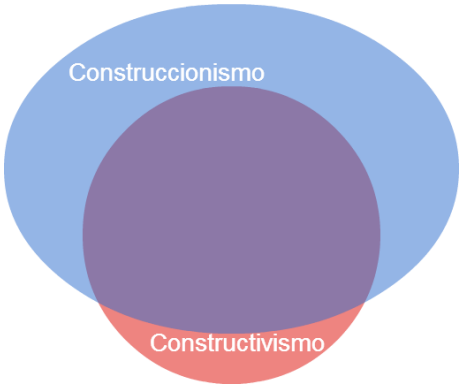

Figura 1.2: Convergencia entre construccionismo y constructivismo

Ahora retomemos la importancia del juego en el aprendizaje en edades tempranas, decidimos enfocarnos a niños por la trascendencia que queremos que tenga el juguete en el desarrollo de una persona, fundamentando el concepto de que el juego tiene mayor relevancia en un niño cuando se relaciona directamente con el objeto de apren-

dizaje. 4

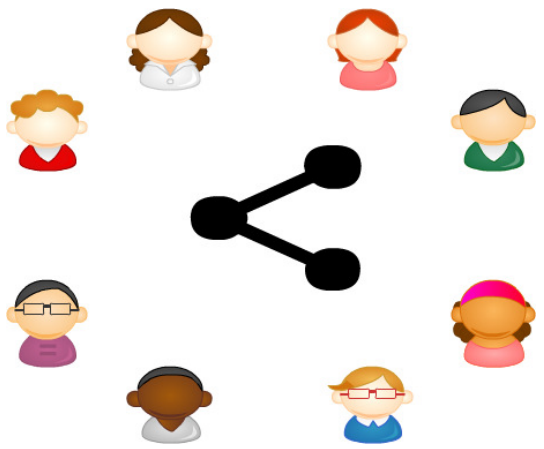

Figura 1.3: Conocimiento compartido

Lo anterior fue el motivo principal por el que seleccionamos la corriente construccionista porque pensamos fijamente que nuestro producto impactará no sólo a un niño a la vez, sino que consideramos que será utilizado por equipos de niños para que en conjunto encuentren nuevas formas de juegos y dinámicas de aprendizaje. Nos parece que de este modo también tendremos mayor repercusión con un grupo más alto de esa sección de la sociedad.

El diseño del lenguaje para programar se basa también en estos principios para la mejor relación delos niños con la programación, es decir, simplificaremos el entorno de edición al grado de que el niño pueda identificar sin complicaciones los íconos para programar el mecanismo.

1.6. Retroalimentación Pedagógica

Una vez que la investigación de la metodología pedagógica fue terminada vimos la necesidad de aclarar conceptos con alguien que sea especialista en la materia. Para esto estuvimos en constante comunicación con la Lic. En Pedagogía Mariana Jiménez Martínez. Con ella terminamos de acotar el porqué de las edades a las cuales decidimos que irá dirigido Olinki y la mejor manera de diseñar el modelo físicamente para que les resulte atractivo como juguete; así mismo el cómo y con qué colores es posible que resulte sencillo de usar el IDE Olinki. También con su apoyo en la retroalimentación decidimos que el diseño del hardware debería de ser modular para que encajara a la perfección con la metodología; siendo más explícitos, en la investigación pedagógica encontramos conceptos como la trascendencia del aprendizaje a través del juego, que en combinación con el paradigma construccionista que habla de que los niños son capaces de escalar el conocimiento a partir de conceptos que genere a partir información previa obtenida de manera colectiva, dedujimos que sería ideal que Olinki se conciba como un juguete que conjunte una máquina programable y el software necesario para programarla, esto una vez que lo platicamos a detalle con la Lic. Mariana.

Comentando con ella encontramos que sería funcional en el sentido de que sería posible que un equipo de niños (idealmente cuatro) se divida en un número par de integrantes para que una parte se dedicara a programar y la otra a ensamblar el mecanismo, esta idea encaja perfectamente en el paradigma pedagógico escogido y el concepto de la importancia del juego.

Ella también nos proporcionó los perfiles que tienen los niños en su comportamiento social dependiendo del grado es colar que se encuentren cursando:

1.6.1. Segundo grado de primaria:

En el momento en que los niños llegan a esta edad (siente años aproximadamente) se busca la aceptación de los adultos (padres, profesores, etc.) Algunos niños demandarán atención de su profesor y se puede ver afectado si no se siente especial. La mayoría de los profesores comprenden este periodo, y tratarán de hacer que todos sus alumnos se sientan importantes. Algunos niños se esfuerzan por lograr una mayor independencia. También se sienten frustrados con demasiada frecuencia. La paciencia con ellos resulta esencial. No poner demasiadas expectativas en él ayudará a que tenga un mejor desarrollo. Trate de no avergonzarlo y apoyarlo con sus tareas de la escuela cuando sea posible para que se sienta exitoso en la escuela.

Será de vital importancia fortalecer habilidades emocionales como:

- Auto conocimiento.
- Control/identificación de emociones.
- Fortaleza emocional.
- Determinación.
- Autocontrol.
- Auto estima.

Mientras desarrollamos habilidades Intelectuales como:

- Rastreo Visual.
- Ubicación espacial.
- Clasificación y comparación.

- Identificación de patrones y secuencias.
- Figuras en espejo.
- Ubicación temporal/espacial.
- Comparación y asociación de objetos y analogías.
- Situaciones de la vida cotidiana.
- Transferencia del conocimiento.
- Inferencias.

1.6.2. Tercer y cuarto año de primaria:

Este es un momento espectacular ya que se cultivan los mejores amigos. Este es también el momento en que los niños comienzan a pensar en el sexo opuesto. Es un tiempo muy inocente para ellos. ¿Les gusta las niñas (o niños)? ¿Es que quiere tener una novia (o novio)? Este es también el momento en que su apariencia empieza a importar. En esta etapa suelen volverse pueden crueles con los comentarios que hacen el uno al otro. Debemos orientarlos acerca de ser objetivos en el tema de las amistades y generar las habilidades necesarias para lograr relaciones de ganar – ganar en todo momento.

Será de vital importancia fortalecer habilidades emocionales como:

- Auto conciencia.
- Planeación.
- Organización.
- Inteligencia emocional.

- Autonomía.

Mientras desarrollamos habilidades Intelectuales como:

- Clasificación y comparación.
- Descripción.
- Interpretación.
- Comparación y asociación de objetos y analogías.
- Situaciones de la vida cotidiana.
- Transferencia del conocimiento.
- Inferencias.
- Análisis.
- Lateralidad.
- Observación.
- Retención.

1.6.3. Quinto año de primaria:

En esta etapa, el niño puede comenzar a hablar de sexo con sus compañeros. Algunos niños pueden sentirse presionados a experimentar con el sexo. Durante esta etapa, es importante que los padres estén muy abiertos a hablar de este tema, en función de sus creencias y valores. Su hijo también puede convertirse en aquel que destaque emocionalmente por su autoestima y fortaleza. Los niños pasan por una gran cantidad de

estrés emocional generado por el entorno, las cuestiones de popularidad, cuestiones personales, y cómo se ven en general. Suelen estar preocupados acerca de cómo sus cuerpos se están desarrollando. Sorprendentemente, hay más y más chicas jóvenes que empiezan su ciclo menstrual durante esta edad. Es necesario acompañarlos muy de cerca ya que se trata de un de ajuste. No escatime tiempo para hablar con sus hijos acerca de cómo se sienten y se ven de sus deseos y sus sueños etc. Por eso las habilidades de comunicación y socialización son de vital importancia en esta edad.

Haremos énfasis en habilidades como:

- Auto conciencia.
- Empatía.
- Asertividad.
- Inteligencia emocional.
- Autonomía.
- Comunicación.
- Solución eficaz de conflictos.
- Establecimiento de metas.

Mientras desarrollamos habilidades Intelectuales como:

- Razonamiento lógico.
- Decodificación e interpretación.
- Identificación y selección de información.

- Inferencia.
- Deducción.
- Metacognición (Capacidad del individuo para trascender y re-apilicar su propio conocimiento).

La información de los perfiles de comportamiento de los niños en diferentes edades, en conjunto con la información obtenida en la investigación y la aclaración de conceptos en la retroalimentación psicológica, nos sirvió como punto de partida para saber qué comportamientos observar en los niños al momento de visitar escuelas. De manera simultánea, nos brindó palabras clave para el diseño de la interfaz y la posible retroalimentación que el sistema Olinki le dará al usuario final; así mismo, la aclaración de los conceptos y paradigmas pedagógicos como el construccionismo nos ayudó para decidir de qué manera puede funcionar mejor el diseño del mecanismo Olinki encajando en las ideas de ésta corriente pedagógica.

1.7. Retroalimentación en escuelas

Como hasta ahora hemos visto, el proyecto está enfocado principalmente a niños que oscilan entre los 7 y 11 años, lo que implica que de alguna manera debemos conocer cómo es el ambiente en el que se desenvuelven para afinar detalles con el diseño del sistema Olinki. Para esto era necesario convivir con grupos de niños de diferentes edades en un ambiente en el que estén aprendiendo de manera habitual, entonces buscamos un espacio en diferentes escuelas dónde nos permitieran poder hacer una observación de campo con alumnos de segundo a quinto año, que son los grados de primaria en los que los niños se encuentran generalmente entre las edades a las que

pretendemos atender.

Las escuelas que visitamos son el instituto Atenea y el Colegio Alamilla Americano. En ellas nos permitieron acceso a los salones de computación para presenciar las clases que les imparten a los alumnos de segundo a quinto grado. En esta experiencia observamos el tipo de programas con los que practican los niños el uso de la computadora, cómo son las gráficas de los programas que utilizan para las dinámicas, los conceptos que conocen y ocupan dentro de lo que conocen de la computación; además conocimos el tipo de lenguaje que ocupan las maestras para comunicarse con los alumnos, esto también tiene importancia porque nos ayudará a definir las palabras que podrán contener el IDE Olinki en su interfaz.

1.8. Intérpretes

Se puede considerar a un intérprete como un procesador de lenguaje que analiza un programa escrito en algún lenguaje de programación, y si es correcto, lo ejecuta directamente en el lenguaje nativo de la máquina que lo está ejecutando, para cada ejecución se debe interpretar de nuevo el lenguaje a diferencia de los compiladores, que emiten código nativo una sola vez para ser ejecutado posteriormente. Existen algunas razones por las que algún lenguaje solo pueda ser interpretado pero no compilado, como son:

- El lenguaje tiene operadores difíciles de compilar, por ejemplo si se tratan las cadenas de texto como operaciones ejecutables, no es el caso de Olinki.
- Se ha eliminado la declaración de variables, que pasa a ser implícita como en el caso de Olinki.
- No hay mecanismos explícitos de control de memoria y es el intérprete quien la

gestiona, en este caso como se verá más adelante Olinki aprovecha esta característica para simplificar el manejo de variables.

- La presencia del intérprete es por razones de seguridad, no es el caso de Olinki.

El hecho de que los compiladores e intérpretes coexistan, sugieren que ambos tipos de procesadores de lenguajes tienen cada uno ventajas en ciertos aspectos, a continuación se en listan las ventajas y desventajas de los intérpretes:

1.8.1. Ventajas de los intérpretes

- Flexibilidad: Los lenguajes interpretados suelen tener más flexibilidad y permiten realizar operaciones más complejas, como la ejecución de cadenas de texto, cambiar sobre la marcha el contenido o tipo de símbolos e inclusive descartar la declaración explicita de variables.

- Gestión de Memoria: El intérprete toma por completo la tarea de tomar y liberar memoria, facilitando al programador la depuración y escritura del programa.

- Desarrollo: Como se anotó anteriormente los lenguajes interpretados facilitan la escritura de programas, automatizando las tareas de gestión de memoria, depuración, y ofreciendo mayor expresividad.

1.8.2. Desventajas de los intérpretes

- Velocidad de ejecución: Generalmente los programas interpretados son varias veces más lentos que los compilados, debido a que cada vez que se ejecutan se debe procesar el código fuente para su posterior ejecución.

- Tamaño del programa: Los programas escritos en lenguajes interpretados suelen ser más grandes que los compilados, ya que estos se encuentran en lenguajes

intermedios o de programación mas generales que el código nativo.

1.8.3. Máquina Virtual

Una máquina virtual consiste en la emulación o implementación vía software de un sistema de cómputo que ejecuta operaciones como si se tratase de una máquina real, en el caso de Olinki, se utiliza una máquina virtual de lenguaje, la cual soporta un solo proceso (el programa que se quiere ejecutar) y expone la abstracción necesaria para el manejo de variables, control de flujo y acceso a sensores que se requiera.

Este proceso inicia al arrancar el programa a ejecutar y se termina cuando se alcanza la condición de parada determinada por la instrucción “FIN”.

El propósito de crear una máquina virtual es la de proporcionar un entorno uniforme sin importar el hardware o sistema operativo en el que ésta se ejecuta, Permitiendonos desarrollar el lenguaje Olinki y sus componentes en una PC (computadora personal por sus siglas en inglés) de escritorio con Windows o Linux y trasladarlo después sin cambios a la Raspberry Pi que cuenta con un procesador ARM y un sistema operativo basado en Debian Linux. Así se agilizó el desarrollo de la plataforma y se comprobó su buen funcionamiento con las pruebas por unidad (Unit Testing). La máquina virtual tiene como interfaz instrucciones de alto nível como operaciones aritméticas entre variables multitipo (cadenas, coma flotante y booleanas), instrucciones para el control de flujo de programas (loop, while, if, ifelse) y creación y manejo de variables.

Para crear variables, la máquina virtual expone un número infinito de registros de uso general, estos registros son de tipo de datos dinámico ya que pueden expresar valores de tipo cadena de carácteres, Númericos de coma flotante y booleanos. Así cuando en el lenguaje de alto nível se crea una variable ’V’, la máquina virtual asigna uno de sus registros al valor de ’V’, y lo direcciona usando el nombre de ’V’. Este proceso se puede observar en la siguiente figura.

```
VM.registros[nombre(V)] <- valor(V)
```

Figura 1.4: Creación de Variables

El acceso a variables sigue un esquema similar al anterior; direcciona los registros por nombre y extrae su valor para su evaluación.

```
valor <- VM.registros[nombre(V)]
```

Figura 1.5: Acceso de Variables

1.8.4. Interfaz Gráfica

La interfaz gráfica de usuario (GUI por sus siglas en inglés) es el componente del sistema de software con el que el usuario interactúa directamente através de iconos gráficos e indicadores visuales, a estos artefactos gráficos se les conoce con el nombre de widgets, cada widget representa alguna acción o funcionalidad del sistema y la expone de forma gráfica al usuario. Ejemplos de widgets son: Ventanas, Botones, Campos de Texto, Menú, etc.

El objetivo de la GUI es la de crear un lenguaje visual que represente la información disponible en el sistema de software, eso ayuda a los usuarios a entender facilmente como interactuar con la computadora, generalmente se usa algún sistema apuntador como el mouse para seleccionar los diferentes elementos en pantalla y un teclado para insertar texto, a este esquema se le conoce con el nombre de WIMP (Window, Icon, Menu, Pointing: Ventana, Ícono, Menú y Apuntador). En el esquema WIMP, los comandos son ejecutados al activar los menús controlando el apuntador o puntero con el mouse.

Olinki se apoya del uso de una GUI para permitir la interacción entre los niños(nuestros

usuarios) y el lenguaje de progrmación haciendo uso de widgets que representan sentencias de éste lenguaje, haciendo la tarea de contruir un programa a base de mover un puntero para arrastrar y soltar las sentencias en el lugar deseado, el uso de menús para seleccionar opciones en las sentencias y el teclado para introducir valores alfanúmericos y lógicos.

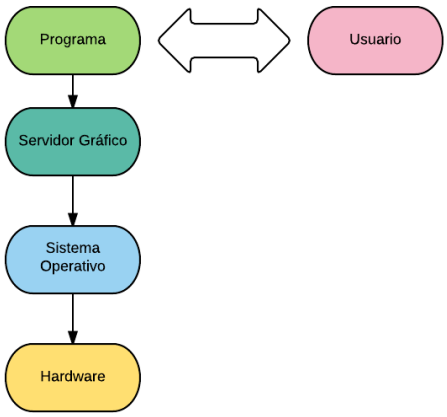

Figura 1.6: Arquitectura de Interfaz gráfica de usuario

1.9. Metodología para diseño del sistema

La mayoría de los sistemas de información están diseñados para resolver un escenario donde se presentan diferentes tipos de usuarios, manejando bases de datos, ligando redes de equipos de cómputo, manipular sistemas remotos, o procesando gran cantidad de información. Para estos es muy apropiado emplear UML (Lenguaje Unificado de modelado) que, como sabemos, permite diagramar por medio de diferentes tipos de grafos las diferentes tipos de clases y requerimientos que serán ocupados para diseñar el sistema y posterior mente darle solución [5]. Hacemos una mención especial en esto para discernir los problemas a los problemas a los que da solución UML y lo que nos puede ayudar a resolver en el desarrollo de nuestro sistema Olinki. Como ya lo he-

mos visto en el protocolo, Olinki será un proyecto que conjuntará hardware y software para funcionar como herramienta de enseñanza y aprendizaje. La parte de hardware fue pensada como un juguete que estará compuesto por una máquina programable y 6 módulos (3 sensores y 3 motores); el software, que es la parte complementaria, tendrá la única función de servir como IDE (Entorno de Desarrollo Integrado) para programar dicho juguete.

Una vez que estudiamos y tuvimos clara la naturaleza del proyecto Olinki nos dimos cuenta que con UML difícilmente es posible dar una visión de cómo debería estar estructurado el diseño del sistema, dada la cantidad de usuarios finales a los que va dirigido el proyecto y la variedad de casos de uso que tendrá. Olinki está pensado para un solo tipo de usuario para el IDE y el circuito: una o varias personas usarán el IDE para programar al circuito que ensamblen. Además de esta diferencia, el IDE Olinki es meramente un intérprete de un lenguaje de programación que diseñaremos desde cero y por esto mismo no hay un diagrama de clases, objetos o conjunto de requerimientos que podamos detallar y modelar con los diagramas del paradigma que ofrece UML. Por estos motivos medulares es que decidimos realizar el modelado del proyecto con una estructura conformada de 4 elementos secuenciales: análisis, diseño, implementación y pruebas.

1.10. Características del sistema

Con base a lo antes mencionado, el sistema Olinki deberá cumplir con un lenguaje de programación ícono gráfico apegado al paradigma construccionista en el sentido de conformarse de bloques de instrucciones que en conjunto construyan un programa. La concepción de cada programa también se apegará a este estilo de pensamiento ya que partirá de instrucciones generadas previamente, las cuales deberán tener congruencia

con las seleccionadas y acomodadas previamente. En el momento que el usuario dé por terminado el programa y lo ejecute habrá un interprete que se encargará de compilar las instrucciones generadas y enviará las señales propias a la tarjeta micro controladora para que las mande a los puertos de entrada o salida según sea el caso. La idea de que el hardware esté diseñado en módulos procura cumplir también con estos conceptos al no limitar el que se conecten a un sólo lugar o estén fijas las salidas y entradas de las señales para que el usuario tenga libre elección para conectar los dispositivos.

Capítulo 2

Diseño

Lo que a continuación describiremos será funcionamiento de la interfaz gráfica y sus componentes, detallando su proceso de desarrollo como también el del hardware desarrollado. También haremos mención de los componentes de los cuales nos auxiliamos para cargar la interfaz y la interpretación de las señales de entrada y salida.

Se seleccionó como nombre del proyecto la palabra Nahuatl “Olinki” que significa máquina, como reflejo de la parte técnica del proyecto, es decir el propio lenguaje de programación, el lenguaje que se propone para este proyecto, tiene como objetivo la construcción de programas simples que serán ejecutados por el controlador principal con la finalidad de operar motores, ensores e indicadores, haciendo uso de pictogramas o iconos para representar las estructuras de control y sentencias del lenguaje, así como una representación en forma de texto como referencia.

2.1. El lenguaje está constituido por 4 tipos de elementos

1. Variables de Entrada y Salida: Tienen como propósito alimentar al programa con datos tomados del entorno físico mediante el uso de sensores, así como entregar resultados hacia los actuadores (motores eléctricos, LEDs).
2. Variables Internas: Contienen valores alfanuméricos, usados como pasos intermedios, o valores temporales en el cálculo de expresiones.
3. Expresiones: Expresan relaciones entre los dos tipos de variables que se nombraron anteriormente, dichas expresiones son de naturaleza lógico-matemáticas (Sumas, Restas, Multiplicación, Concatenación, Negación, etcétera).
4. Bloques de Control: Afectan el flujo del programa mediante el uso de operadores lógicos.

2.2. Gramática del Lenguaje

Se propone la siguiente gramática para el lenguaje Olinki, la cual expresa la naturaleza de las expresiones y relaciones entre las variables:

2.3. Tipo de Datos

Internamente las variables pueden ser del tipo: cadena de texto, numérico y lógico, donde cadena representa una secuencia de caracteres, numérico representa enteros y números de coma flotante y los del tipo lógico representan valores que pueden tomar

```
S -> I
I -> inicio V
V -> V | F
F -> fin
V -> C
C -> si E { V } Sn
Sn -> sino { V } | epsilon
E ->      var mayor que var |
     var menor que var |
     var mayor igual que var |
     var menor igual que var |
     var igual var |
     var diferente var
E ->      E  y E |
     E o E |
     o E o E|
     no E
V -> M
M ->      var + var |
     var - var |
     var * var |
     var / var |
     var % var
C -> A
A -> nombre = M | F | E | S
```

Figura 2.1: Gramática del Lenguaje del intérprete

como valor Verdadero o Falso. La mutación de tipos dentro del lenguaje es transparente al programador, estas son las reglas que rigen dicha conversión.

Tipo	Numérico	Cadena	Lógico
Numérico	Valor	"Valor"	Valor >= 1? Verdadero: Falso
Cadena	Valor | Longitud("Valor")	Valor	"Verdadero" => Verdadero, otro Falso
Lógico	Valor? 1: 0	Valor? "Verdadero": "Falso"	Valor

Figura 2.2: Tabla de tipos de datos.

2.4. Operaciones

Las operaciones entre las variables quedan definidas en términos del tipo de dato.

Operación	Numérico V1,V2	Cadena V1,V2	Lógico
SUMA	V1 + V2	Concatenación(V1, V2)	OR(V1, V2)
RESTA	V1 - V2	NA	NA
MULTIPLICACIÓN	V1*V2	NA	AND(V1, V2)
DIVISIÓN	V1/V2	NA	NA
LONGITUD	ValorAbsoluto(V1)	Longitud(V1)	V1?1:0

Figura 2.3: Tabla de operaciones.

2.5. Palabras reservadas

El lenguaje de programación tiene un conjunto de palabras reservadas que no pueden se ocupadas o definidas por el usuario, esto es con el fin de tener identificadas las funciones que realizará el intérprete. Éstas son las palabras reservadas del sistema Olinki:

●Palabra.- INICIO: Determina inicio de programa

●Palabra.- SI: Primera Condicional

●Palabra.- SINO: Segunda Condicional

●Palabra.- REPETIR: Inicio de Ciclo de iteración

●Palabra.- ESCRIBE: Mostrar cadena en pantalla

●Palabra.- LIMPIAR: Limpiar pantalla

●Palabra.- MOTOR: Activar motor

●Palabra.- SENSOR: Leer valor de sensor

●Palabra.- FIN: Terminar Bloque de Iteración, Condicional o Programa

●Palabra.- Y: Operador lógico AND

●Palabra.- O: Operador lógico OR

●Palabra.- NO: Operador lógico de negación

2.6. Ejemplo de programa

El intérprete recibiría una secuencia de instrucciones parecida la siguiente:

```
INICIO
suma = 0
REPETIR 5
```

```
suma = suma + 1
FIN
SI suma = 5
ESCRIBE "Hola"
SINO
ESCRIBE "Adiós"
FIN
FIN
```

2.7. Requisitos funcionales (RF)

En esta sección conoceremos las acciones que podrá realizar Olinki cuando se manipulado por el usuario. Primero hablaremos del comportamiento que tendrá el EDI Olinki (Entorno de desarrollo integrado Olinki) y posteriormente la variedad de funciones que es posible realizar.

2.7.1. EDI Olinki

EDI Olinki es un sistema que puede ser manipulado desde su instalación en un equipo de cómputo o desde la máquina programable; si se programa desde éste último necesitaremos auxiliar nos de una pantalla de televisor para poder visualizar el mismo entorno gráfico y navegar con los botones del mecanismo físico. A continuación conoceremos las funcionalidades de EDI Olinki.

●RF.- El usuario podrá crear un programa con extensión de formato .oli en EDI Olinki.

●RF.- El programa que sea creado en EDI Olinki podrá ser abierto y visualizado para su edición.

●RF.- Será posible guardar en el equipo cualquier programa creado o editado en EDI Olinki siempre y cuando no exista algún otro con el mismo nombre.

●RF.- Una vez que el usuario haya terminado el programa podrá interpretarlo con EDI Olinki.

2.7.2. Mecanismo Olinki

El mecanismo Olinki es el conjunto de 2 motores y 3 sensores (de contacto, luz y temperatura) que pueden ser ensamblados y conectados en la máquina programable para que realicen alguna acción una vez que es ejecutado el programa creado en EDI Olinki. En seguida haremos una descripción de sus funciones.

●RF.- Uno o más de los sensores podrán ser ensamblados con el cuerpo de la máquina programable Olinki.

●RF.- Uno o más de los motores podrán ser ensamblados con el cuerpo de la máquina programable Olinki.

●RF.- Los sensores podrán ser activados una vez que sea ejecutado el programa cargado en la máquina programable Olinki.

●RF.- Los motores podrán ser activados una vez que sea ejecutado el programa cargado en la máquina programable Olinki

●RF.- El usuario podrá navegar en la pantalla de televisor o pantalla integrada, con los periféricos conectados a la máquina programable Olinki.

2.8. Requisitos no funcionales (RNF)

En el siguiente apartado se detallarán características que deberá cumplir el equipo de cómputo en el cual se desee instalar EDI Olinki y los aspectos generales de su interfaz.

También se hará una descripción de las características físicas que tendrá el mecanismo para conocer para conocer su posible comportamiento.

2.8.1. EDI Olinki

Comenzaremos con los requerimientos básicos para la instalación en un equipo de cómputo y especificaciones de la interfaz del EDI Olinki.

- RNF.- EDI Olinki podrá ser instalado en cualquier computadora que tenga un procesador superior a 500 MHz, memoria RAM mayor a 512 MB, almacenamiento de disco duro arriba de los 2 GB, periféricos básicos (teclado y mouse).
- RNF.- EDI Olinki podrá ser instalado en cualquier sistema operativo Linux, Windows o OSX.
- RNF.- Para usar EDI Olinki desde un televisor, éste deberá contar una entrada HDMI o RCA Compuesta.
- RNF.- El entorno de programación de EDI Olinki resultará de simple uso en particular para los usuarios de pequeñas edades.
- RNF.- Con EDI Olinki el usuario podrá escoger el directorio para ubicar cada programa creado o editado.
- RNF.- En EDI Olinki el usuario podrá visualizar los programas que han sido creados.

2.8.2. Mecanismo Olinki

En seguida presentaremos una descripción de una serie de aspectos que tendrá el mecanismo Olinki en su aspecto físico.

- RNF.- El equipo de cómputo deberá contar con uno o más puertos Ethernet.
- RNF.- El usuario no tendrá acceso al circuito interno de la máquina programable Olinki ni de sus componentes periféricos (motores y sensores).
- RNF.- Los botones de la máquina programable Olinki estarán colocados de tal manera que sea fácil la navegación durante la programación cuando se programe directamente y se quiera hacer uso de una pantalla de tv.
- RNF.- Los periféricos de Olinki no estarán conectados permanentemente a la máquina programable.
- RNF.- No será necesario que el usuario tenga conocimientos previos para que le resulte fácil el armado de la máquina programable Olinki con sus periféricos.

2.9. Interfaz gráfica

La interfaz gráfica se compone de dos partes básicas: el área de trabajo (derecha) y el área de herramientas (izquierda).

La estructura funcional de esta sección se encuentra desarrollada en HTML5, puesto que para los fines de la aplicación presenta virtudes que aprovechamos, como la capacidad semántica que tiene el lenguaje para describir el contenido las propiedades que puede ofrecerle a los contenidos 2D. Por otro lado permite auxiliarnos de CSS3 (Cascading Style Sheets, por sus siglas en inglés), ya que como su nombre lo dice, se trata de

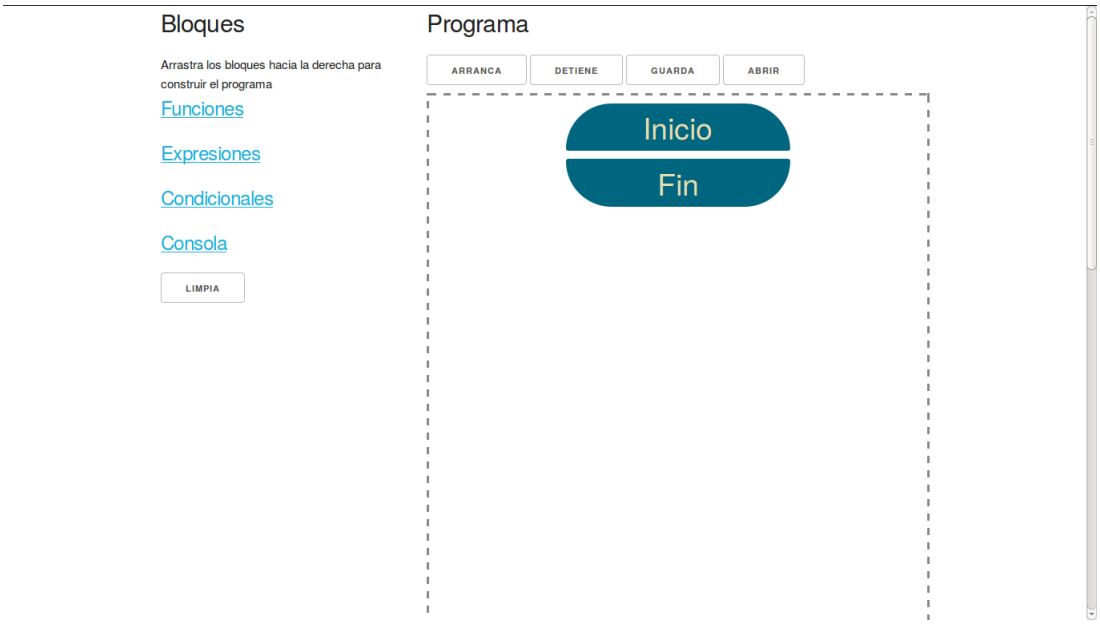

Figura 2.4: Pantalla inicial de la interfaz.

la tercera versión de las hojas de estilo en cascada que nos permitirá fijar un estándar en la estructura de los colores, formas, efectos y comportamientos que poseerán los objetos gráficos del contenido web, los cuales se detallarán más adelante.

Figura 2.5: HTML5 y CSS3, herramientas estructurales para el desarrollo.

2.10. Equivalencia Gráfica

Las palabras reservadas anteriormente listadas forman reglas gramaticales al principio de este capitulo, estas reglas tienen un equivalente gráfico que facilita la creación de programas mediante el uso de una interfaz gráfica de usuario y acciones drag AND

drop. La equivalencia entre reglas gramaticales y elementos gráficos es 1 a 1, es decir que existe un elemento gráfico por cada regla, así estos elementos gráficos son usados como piezas que al ser puestos juntos forman un nuevo programa. Se enlista la equivalencia en la figura 4.11

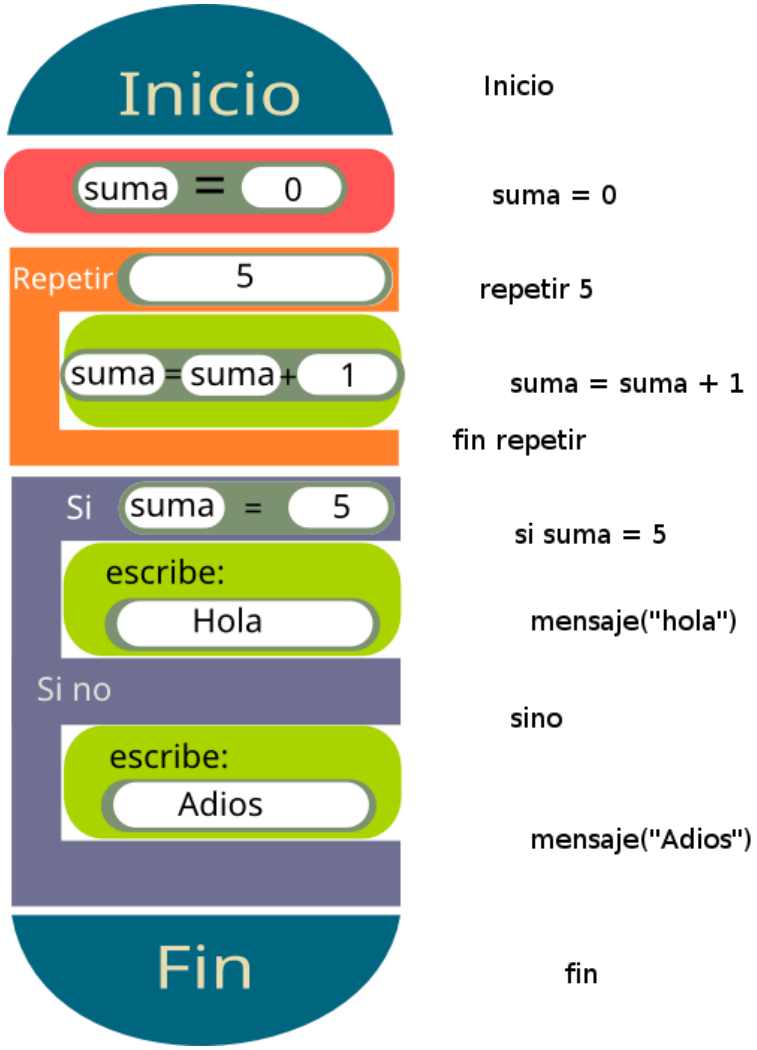

Figura 2.6: Equivalencias

2.10.1. Bloques

Aquí se encuentran todos los módulos gráficos que el usuario puede arrastrar hacia el programa. También podrá darles valores a las variables que necesite, escoger entre las diferentes opciones de condicionamiento. Además se cuenta con una pequeña consola para cuando el usuario desee mostrar mensajes o resultados en pantalla. A continuación los bloques detallados:.

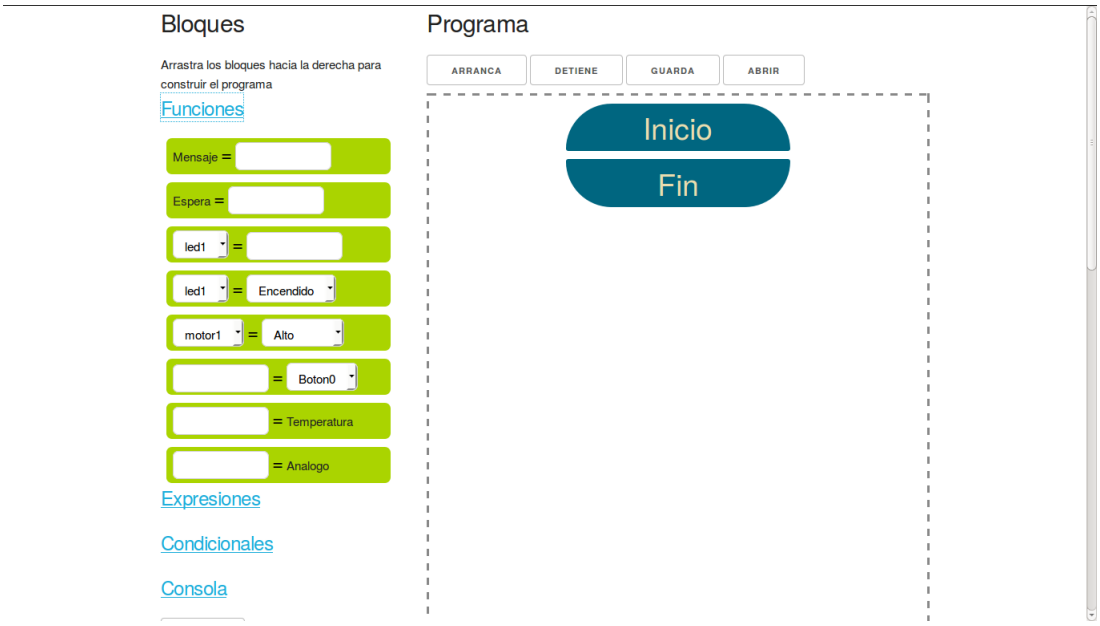

Figura 2.7: Ejemplo de arrastrar instrucciones.

2.11. Circuitos

La placa encargada

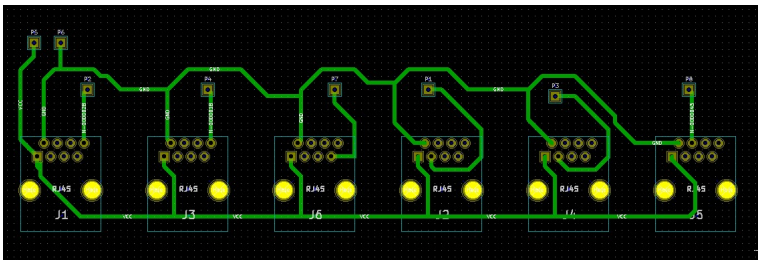

Figura 2.8: Circuitos de los sensores.

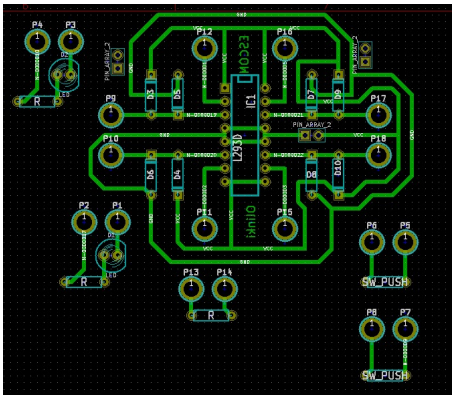

Figura 2.9: Tabla de interfaz a microcontrolador Arduino.

Capítulo 3

Implemenctacion

En este capítulo se abordarán los detalles de Implemenctación del proyecto, describiendo la integración software y hardware así como los submodulos que los conforman.

3.1. Interfaz Gráfica de Usuario

Como se ha comentado anteriormente, la interfaz gráfica de usuaio (GUI por sus siglas en inglés) está implementada usando el lenguaje de marcadores de hipertexto (HTML por sus siglas en inglés), Javascript y CSS(Cascade Style Sheet). En concreto para representar las sentencias del lenguaje se utiliza la etiqueta DIV, la cual se modifica usando reglas CSS para modificar el color, tamaño y posicionamiento de estos elementos:

```
<div id = "exp-mess" class="p-bloque p-expr c-fun" tipo="mess">
      <div id ="exp-igual-cont-m"  class="p-expr-in">
        Mensaje
       <a class="operator"> =</a>
        <input type="text" class="p-repetir-input">
      </div>
```

```
    </div>
```

Aquí se puede observar como se anidan las etiquetas div para crear un elemento que representará la sentencia Mensaje, al aplicar las siguientes Reglas CSS

```
     .p-bloque {
width: 280px;
margin-left: auto;
margin-right: auto;
margin-top: 10px;
cursor: pointer;
    }

    .p-expr {
background-color: #aad400;
height: 45px;
border-radius: 7px;
    }
```

El div se visualiza como una caja Verde con los bordes redondos, el texto “Mensaje”, un espacio para ingresar los argumentos de esta instrucción:

Figura 3.1: Sentencia Mensaje

3.1.1. Arrastrar y Soltar

El principal mecanísmo para construir programas en Olinki es arrastrar y soltar sentencias en el espacio de trabajo, es por eso que se puso especial cuidado en esta funcionalidad, la meta es proporcionar una manera simple de seleccionar las sentencias disponibles y colocarlas en el programa en construcción.

Nos apoyamos de la librería de Javascript y software libre “Dragula” la cual es soportada por la mayoría de los navegadores web y motores de renderización de Html. Su uso es simple, se requiere listar el contenedor fuente, es decir el tag padre que contenga los elementos a ser arrastrados, en el caso de la GUI, se usa el área de sentencias como fuente y el área de trabajo como contenedor destino.

```
var bloques = document.getElementById("blocks");
var programa = document.getElementById("program");


var drake = dragula([bloques, programa], {
  copy: true,
  accepts: function(el, target, source, sibling){
      //console.log("accept: " + target.className);
      if(sibling){
        //console.log("si: " + sibling.className);
        var type = sibling.getAttribute("tipo");
        if(type == "begin"){
          return false;
        }
      }
      else{
        if(target.className.indexOf("program") !== -1){
```

```
            return false;
          }
        }


        return true;
    },
    moves: dontMove
});
```

Como se puede observar en el código anterior se puede pasar tres parámetros opcionales, copy: permite clonar los elementos arrastrados, accepts: permite filtrar los elementos a ser soltados y moves: la cual permite filtrar que elementos se pueden mover.

3.2. Análisis Léxico, Sintáctico y Semántico

Se puede aprovechar el proceso de construcción del programa por el usuario, para realizar el análisis léxico y el análisis sintáctico, a cada momento que se generá una nueva sentencia dentro del lenguaje, el análisis léxico constituye en verificar que todas las palabras contenidas en el programa sean palabras que pertenecen al lenguaje, por ejemplo dado el lenguaj L = {hola, vaca, casa} y la palabra “avión” es fácil ver que existe un error léxico ya que “avión” no es un elemento de L. Aplicando esto al lenguaje Olinki, se enlistan todas las palabras que pertenecen a Olink en forma de elementos gráficos, y se evita la creación por parte del usuario de otras palabras gráficas de esta menera se evitan los errores léxicos dentro de Olinki. El análisis sintáctico se da cuando se toma la lista de sentencias y se transforman en un árbol de sintaxis abstracta en forma de JSON.

3.3. Backend - Lógica de Negocio

El desarrollo de la lógica principal o lógica de negocio, se llevó a cabo usando el lenguaje de programación C++, este lenguaje nos ofrece como beneficios la utilización eficiente de memoria en ejecución, compilación a código nativo, soporte multiplataforma, abstracción de clases, programación funcional y definición de operadores propios.

También se usaron las librerias de C++ estandard STL y las librerías de código libre Qt, las primeras nos permiten crear código eficiente multiplataforma y añadir soporte a cadenas, apuntadores inteligentes, contenedores y algoritmos matemáticos, mientras que Qt nos ofrece las funcionalidades necesarias para crear aplicaciones con interfaces de usuario mixtas permitinedo integrar un motor HTML en nuestra aplicación nativa, y facilitar la comunicación entre objetos Javascript y C++. Se aprovechó la sobrecarga de operadores en clases de C++ para implementar las operaciones entre los tipos de datos:

```
Variable operator +(const Variable &v) const { return suma(v);}
Variable operator -(const Variable &v) const { return resta(v);}
Variable operator *(const Variable &v) const { return multiplicacion(v);}
Variable operator /(const Variable &v) const { return division(v);}
Variable operator %(const Variable &v) const { return modulo(v);}
Variable operator >(const Variable &v) const { return mayor_que(v);}
Variable operator >=(const Variable &v) const { return mayor_igual_que(v);}
Variable operator <(const Variable &v) const { return menor_que(v);}
Variable operator <=(const Variable &v) const {return menor_igual_que(v);}
Variable operator == (const Variable &v) const { return igual_que(v);}
Variable operator != (const Variable &v) const { return diferente_que(v);}
Variable operator !() const { return inverso();}
Variable operacionPorNombre(const std::string &op, const Variable &v = Variable());
```

De esta manera el desarrollo del resto del motor de variables fue mas natural, permitiendo la expresión de las operaciones de una manera mas sencilla.

```
Variable Variable::suma(const Variable &v) const
{
    if(m_tipo == Cadena || v.m_tipo == Cadena){
        return valorString() + v.valorString();
    }

    switch(m_tipo){
    case Numerico:
        return Variable(valorDouble() + v.valorDouble());

    case Logico:
        return Variable(bool(valorBool() || v.valorBool()));

    case SinTipo:
    default:
        break;
    }

    return Variable();
}
```

Para implementar la tabla de variables se usó una tabla Hash donde la llave está dada por el nombre de la variable y el valor por la variable en sí. A sí pues, se crearon funciones auxiliares que simplifican la creación y búsqueda de variables por el intérprete.

```
void crearVariable(const std::string &nombre, const Valor &v, const Variable::Tipo &
void actualizar(const std::string &nombre, const Valor &v, const Variable::Tipo &tip
void actuliazaCrea(const std::string &nombre, const Variable &var);
Variable remover(const std::string &nombre);
Variable busca(const std::string &nombre);
Variable busca(const std::string &nombre, bool &exito);
```

Intérprete

El intérprete propuesto para el lenguaje de programación Olinki, fue implementado por una máquina virutal desarrollada en C++, está máquina tiene soporte para todas las intrucciones de Olinki y usa la tabla de variables como su archivo de registros así pues, se propone una Máquina virtual (VM) con arquitectura Harvard, con un número infito de registros de propósito general con un tipo de datos mutable (cadena, numérico y booleano) y tres registros especializados: contador de programa (PC), contador de Ciclo (LOOP) y apuntador a condición de parada (SP). Y una pila para guardar el estado de la máquina en caso de llamadas anidadas. La ejecución consiste en tres partes: Obtención de Instrucción (Fetching), Decodificación (Decoding) y Ejecución.

Fetching

La máquina virtual obtiene la Instrucción a ejecutar desde la memoria de código usando el contador de programa.

```
auto &last = stack.last(); //Contexto Actual
QJsonObject inst = last.ctx.at(pc).toObject(); //Instrucción a Ejecutar
```

La Instrucción se retiene en forma de un objeto JSON con el siguiente formato:

```
{
        "type": "expr-op",
        "var": variable,
        "op1": op1,
        "operation": operation,
        "op2": op2
    };
```

Decoding y Ejecución

La parte de Decodificación de la Instrucción comienza con la interpretacion del tipo de Instrucción que se está ejecutando, de esta manera se deduce que función llamar mediante un switch-case.

La ejecución se lleva a cabo dentro de cada caso de la construcción switch, en esta se extraen los operandos de la Instrucción y se llama a la funcion adecuada que realiza el trabajo de dicha Instrucción, como por ejemplo realizar una suma, resta o multiplicacion, habilitar algun sensor, esperar algún tiempo, terminar la ejecución del programa, etc.

```
  QString t_string = inst.value("type").toString("no-type"); //valor de tipo de instrucc
          Parser::Types type = types[t_string]; //tipo de instruccion

           switch (type) {
```

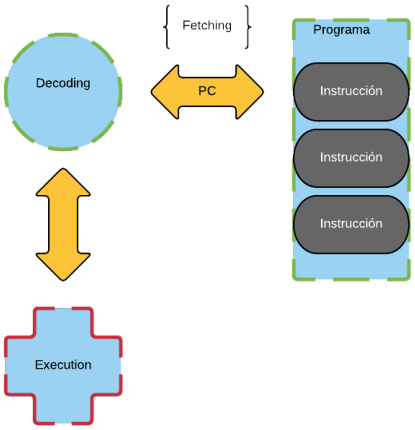

Figura 3.2: Proceso de Ejecución

```
case ASIG:{
    ...
}
    break;

case EXPROP:{
    ...
}
    break;

case LOOP:{
    ...

}
    break;
```

```
case WHILE:{
   ...

}
    break;

case IF:{
   ...
}
    break;
case IFELSE:{

    ...

}
    break;

case MESS:{
   ...
}
    break;

case LED:{
  ...
 }
    break;
```

```
    case BUTTON:{
        ...
    }
        break;

    case TEMP:{
        ...
    }
        break;

    case ANA:{
...
    }
        break;

    case DELAY:{

        ...
    }
        break;

    case MOTOR:{
        ...
    }

    case BEGIN:
    case END:
```

```
case EXPRLOG:
case NO_TYPE:
default:
    break;
}
```

El contador de programa se incrementa despues de la ejecución de la instrucción en curso para preparar la instrucción a ejecutar en el siguiente ciclo de reloj, además se checa el estado de la pila y los registros de ciclo y condición de parada para extraer el contexto en el que se ejecutará las posteriores instrucciones.

```
auto &ctx = stack.last(); //Contexto actual
pc++; //incremento del contador de programa
//Se esta ejecutando la instrucción dentro de un ciclo
if(ctx.type == LOOP){
   ...
}
//Se esta ejecutando la instrucción en un ciclo con condición de parada lógico
else if(ctx.type == WHILE){
  ...

}
//Existe una condición para ejecutar la instrucción
```

```
else if(ctx.type == IF || ctx.type == IFELSE){
  ...



}
//Se ha concluido con la ejecución del programa y la máquina virtual se detiene
else if(ctx.type == PROGRAM){
    if(pc >= ctx.ctx.size()){
        stop();
        st = FINISHED;
    }
}
```

Por último al terminar la ejecución del programa en curso, la máquina virtual entra en el estado de Parada y se da como concluida la ejecución y se emite la señal “FINISHED” para comunicar a la interfaz gráfica del cambio de estado en la máquina virtual.

Comunicación del Lenguaje con los Sensores y Actuadores

La máquina virtual anteriormente descrita cuenta ademas con un módulo de comunicación mediante puerto serie, el cual le permite mandar mensajes de lectura y escritura al microcontrolador Arduino a cargo de la placa de sensores y la placa controladora de los motores.

estos mensajes son de 3 bytes de tamamaño y tienen la siguiente forma:

```
enum Sensor {
    CONTACT = 0,
    TEMP,
```

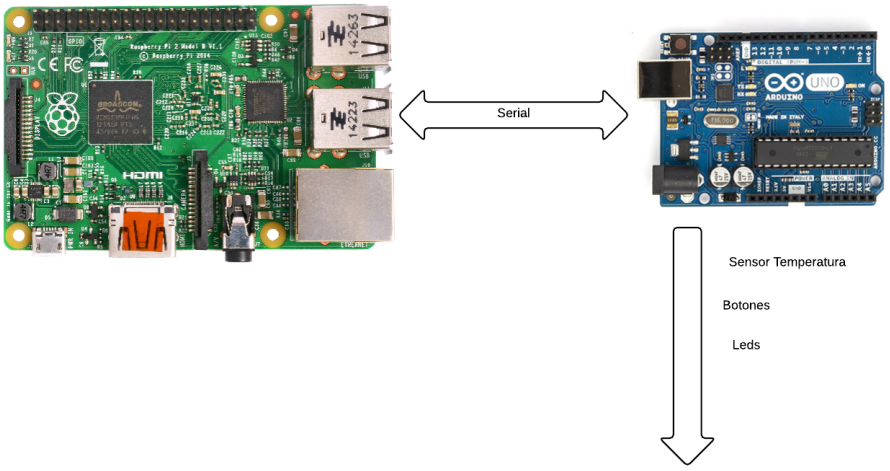

Figura 3.3: Comunicación Raspberry pi y Arduino

```
    MOTOR,
    LED,
    ANALOG

};

struct SMess {
    uint8_t sensor; //Sensor al que se quiere consultar, listados arriba
    uint8_t idx;    //indice del sensor, por ejemplo: LED1 o LED2
    uint8_t val;    //valor a escribir en caso de ser operacion de escritura
};
```

EL tipo de operación (lectura o escritura) está determinado por la naturaleza del sensor, es dicr que los sensores de contacto, temperatura, y analógico solo pueden ser leídos, y los motores y leds solo pueden ser escritos (ENCENDIDO/APAGADO). Al ter-

minar la operación el Arduino responde con un solo byte, indicando el resultado de la operación.

Microcontrolador Arduino

El microcontrolador Arduino tiene el control directo de las placas de sensores y la placa controladora de motores, ya que las diferentes terminales de estas se conectan los pines del microcontrolador como se muestra a continuación.

```
const int button_pin[2] = {0,1};
const int analog_pin = A0;
const int motor_pin[2][2] = { {2, 3},{4, 5} };
const int led_pin[2] = {6, 7};
const int temp_pin = 8;
```

Como se puede notar los pines 1 y 2 corresponden a los sensores de contacto o pushbuttons, el sensor de iluminación o analógico está conectado al pin A0, el motor izquierdo está controlado por los pines 2 y 3

Además de mantener los objectos para la lectura del sensor de temperatura, y las funciones para normalizar las lecturas de los sensores entre los valores de 0 a 255 (un byte), manteniendo uniformidad entre estos. Así mismo efectua la interpreación de los comandos ALTO, IZQUIERDA y DERECHA recibidos desde el lenguaje de programación Olinki a valores lógicos usados por el puente H para la activación de los motores.

```
const uint8_t m_values[3][2] = { {0,0}, {1,0}, {0,1} }; //ALTO, IZQUIERA y DERECHA

void decodeMotor(uint8_t idx, uint8_t cmd) //Función de interpretación de comandos p
{
```

```
uint8_t v1 = m_values[cmd][0];
uint8_t v2 = m_values[cmd][1];

digitalWrite(motor_pin[idx][0], v1);
digitalWrite(motor_pin[idx][1], v2);
}

float decodeTemp() //Interpretación del sensor de temperatura
{
sensors_event_t ev;
dht.temperature().getEvent(&ev);
if (isnan(ev.temperature)) {
return 0.0f;
}

return ev.temperature;
}
```

3.3.1. Comunicación entre Interfaz de Usuario y Backend

La interfaz gráfica de usuario está implmentada en Html, CSS y Javascript, se comunica con el Backend mediante mensajes JSON, usando el WebKit Bridge de las librerías Qt. El cual es un mecanísmo que entiende el acceso a objetos de Javascript mediante la declaración de QObjects. Esto se logra registrando los objectos C++ (derivados de QObjects) en el marco donde se está mostrando la interfaz gráfica.

```
QWebFrame *frame = myWebPage->mainFrame();
frame->addToJavaScriptWindowObject("someNameForMyObject", myObject);
```

Permitiendo acceder los métodos y propiedades de los objetos QObject desde el contexto Javascript

```
parser.message.connect(function(mess){
var c = document.getElementById("consola");
c.innerHTML += "<p>" + mess + "</p>";
var cont = c.parentElement;
cont.scrollTop = cont.scrollHeight;
});

file.programLoaded.connect(function(json){
 console.log(json);
 unparse(json);
});
```

Código Javascript haciendo uso de los siguientes objectos de C++

```
 class Parser : public QObject
{
    Q_OBJECT
public:
    explicit Parser(Comm *c, QObject *parent = 0);
```

```
Q_INVOKABLE bool load(const QString &json, bool async = true);
Q_INVOKABLE void stop();

enum Types {
    ASIG,
    EXPROP,
    EXPRLOG,
    LOOP,
    WHILE,
    IF,
    IFELSE,
    MESS,
    LED,
    BUTTON,
    TEMP,
    ANA,
    DELAY,
    MOTOR,
    BEGIN,
    END,
    PROGRAM,
    NO_TYPE

};

enum LED_V {
    ON = 1,
```

```
    OFF = 0
};

enum MOTOR_V {
    STOP = 0,
    RIGHT = 1,
    LEFT = 2

};

enum State {
    STOPPED,
    RUNNING,
    FINISHED
};

struct BPoint{
    QJsonArray ctx;     //Contexto actual
    Parser::Types type; //Tipo de Contexto
    ProgramCounter ins; //Punto de entrada de proximo ctx
    int loop_counter;
    int loop_ceiling;
    std::string op1_name;
    std::string op2_name;
    std::string op_name;

    BPoint(){
```

```
            loop_counter = 0;
            loop_ceiling = 0;
        }

    };

    State state();
    QString getSimbols();
    Variable varGuessingType(const QString &v);
    Variable varGuessingType(const std::string &v);


signals:
    void message(const QString &m);
    void led(int idx, bool v);

public slots:
       void tick();
       void disable_delay();
private:
    QTimer timer;
    QMap<QString, Types> types;
    QMap<LED_V, QString> led_s;
    QMap<MOTOR_V, QString> motor_s;
    TablaVariables simbols;
    ProgramCounter pc;
    QList<BPoint> stack;
```

```
    State st;
    Comm *comm;
    bool delay;
private slots:


};


class FileManager : public QObject
{
    Q_OBJECT
public:
    explicit FileManager(QObject *parent = 0);

signals:
    void programLoaded(const QString &file);

public slots:
    Q_INVOKABLE void saveProgram(const QString &contents);
    Q_INVOKABLE void loadProgram();
};
```

Capítulo 4

Pruebas

En este capítulo se abordaran las pruebas de lenguaje y variables. Su intención es probar el buen funcionamiento del sistema y motrar el comportamiento del mismo en diversas situaciones.

4.1. Pruebas de Lenguaje

Para probar las capacidades del lenguaje se decidió separarlas en pruebas del motor de variables y pruebas de ejecución, las primeras consisten en probar la capacidad para crear y manejar variables, además de verificar el correcto funcionamiento de las operaciones entre los distintos tipos de datos (Texto, númerico y booleano).

4.1.1. Pruebas del Motor de Variables

Para probar la capacidad del motor de variables se diseñó una prueba automatizada la cual crea números aleatorios y los somete a las operaciones de suma, resta, multiplicación y división:

```
void testArithmetic(int times = STEPS)
```

```
{
    std::default_random_engine generator;
    std::uniform_real_distribution<double> dist(0 , std::numeric_limits<double>::max()/2

    for(int i = 0; i < times; ++i){

        double val = dist(generator);
        double val2 = dist(generator);

        Variable varN(val);
        Variable varN2(val2);
        printf("Comprobando asignacion numerica var%d = %g", i, val);
        if(val != varN.valorDouble()){
            printf("Fallo al asignar valor %g a variable Numerica: %g\n", val, varN.valo
            exit(0);
        }
        printf("\tOk\n");
        printf("Comprobando Suma numerica %g + %g", val, val2);
        if(val + val2 != (varN + varN2).valorDouble()){
            std::cout << "Fallo Suma de dos variables " << val << ", " << val2 << std::e
                      << "Se tuvo: " << (varN + varN2).valorDouble() << " se esperaba:"
            exit(0);
        }
        printf("\tOk\n");
        printf("Comprobando Resta numerica %g - %g", val, val2);
        if(val - val2 != (varN - varN2).valorDouble()){
            std::cout << "Fallo Resta de dos variables " << val << ", " << val2 << std::
```

```
                    << "Se tuvo: " << (varN - varN2).valorDouble() << " se esperaba:"
            exit(0);
        }
        printf("\tOk\n");
        printf("Comprobando Multiplicacion numerica %g * %g", val, val2);
        if(val * val2 != (varN * varN2).valorDouble()){
            std::cout << "Fallo Multiplicacion de dos variables\n " << val << ", " << va
                    << "Se tuvo: " << (varN * varN2).valorDouble() << " se esperaba:"
            exit(1);
        }
        printf("\tOk\n");
        if(val2 == 0.0){
            continue;
        }
        printf("Comprobando Division Numerica %g / %g", val, val2);
        if(val/val2 != (varN / varN2).valorDouble()){
            printf("\nFallo\n");
            exit(0);
        }
        printf("\tOk\n");

    }

    printf("Prueba de variables numerica pasada\n");

}
```

Está función se probó con 10, 100 y 10,000 iteraciones, todas ellas satisfactorias, se lista parte de la salida de esta prueba.

```
Comprobando Multiplicacion numerica 1.34383e+306 * 5.77939e+307 Ok
Comprobando Division Numerica 1.34383e+306 / 5.77939e+307 Ok
Comprobando asignacion numerica var91 = 8.25004e+307 Ok
Comprobando Suma numerica 8.25004e+307 + 8.71959e+307 Ok
Comprobando Resta numerica 8.25004e+307 - 8.71959e+307 Ok
Comprobando Multiplicacion numerica 8.25004e+307 * 8.71959e+307 Ok
Comprobando Division Numerica 8.25004e+307 / 8.71959e+307 Ok
Comprobando asignacion numerica var92 = 7.58604e+307 Ok
Comprobando Suma numerica 7.58604e+307 + 4.09651e+307 Ok
Comprobando Resta numerica 7.58604e+307 - 4.09651e+307 Ok
Comprobando Multiplicacion numerica 7.58604e+307 * 4.09651e+307 Ok
Comprobando Division Numerica 7.58604e+307 / 4.09651e+307 Ok
Comprobando asignacion numerica var93 = 1.98499e+306 Ok
Comprobando Suma numerica 1.98499e+306 + 6.3544e+307 Ok
Comprobando Resta numerica 1.98499e+306 - 6.3544e+307 Ok
Comprobando Multiplicacion numerica 1.98499e+306 * 6.3544e+307 Ok
Comprobando Division Numerica 1.98499e+306 / 6.3544e+307 Ok
Comprobando asignacion numerica var94 = 3.92471e+307 Ok
Comprobando Suma numerica 3.92471e+307 + 6.75672e+307 Ok
Comprobando Resta numerica 3.92471e+307 - 6.75672e+307 Ok
Comprobando Multiplicacion numerica 3.92471e+307 * 6.75672e+307 Ok
Comprobando Division Numerica 3.92471e+307 / 6.75672e+307 Ok
Comprobando asignacion numerica var95 = 6.25309e+307 Ok
Comprobando Suma numerica 6.25309e+307 + 6.77607e+307 Ok
Comprobando Resta numerica 6.25309e+307 - 6.77607e+307 Ok
```

```
Comprobando Multiplicacion numerica 6.25309e+307 * 6.77607e+307 Ok
Comprobando Division Numerica 6.25309e+307 / 6.77607e+307 Ok
Comprobando asignacion numerica var96 = 3.82362e+305 Ok
Comprobando Suma numerica 3.82362e+305 + 7.1833e+306 Ok
Comprobando Resta numerica 3.82362e+305 - 7.1833e+306 Ok
Comprobando Multiplicacion numerica 3.82362e+305 * 7.1833e+306 Ok
Comprobando Division Numerica 3.82362e+305 / 7.1833e+306 Ok
Comprobando asignacion numerica var97 = 8.11929e+307 Ok
Comprobando Suma numerica 8.11929e+307 + 6.7115e+307 Ok
Comprobando Resta numerica 8.11929e+307 - 6.7115e+307 Ok
Comprobando Multiplicacion numerica 8.11929e+307 * 6.7115e+307 Ok
Comprobando Division Numerica 8.11929e+307 / 6.7115e+307 Ok
Comprobando asignacion numerica var98 = 5.70021e+305 Ok
Comprobando Suma numerica 5.70021e+305 + 4.66033e+307 Ok
Comprobando Resta numerica 5.70021e+305 - 4.66033e+307 Ok
Comprobando Multiplicacion numerica 5.70021e+305 * 4.66033e+307 Ok
Comprobando Division Numerica 5.70021e+305 / 4.66033e+307 Ok
Comprobando asignacion numerica var99 = 4.88993e+307 Ok
Comprobando Suma numerica 4.88993e+307 + 1.93994e+307 Ok
Comprobando Resta numerica 4.88993e+307 - 1.93994e+307 Ok
Comprobando Multiplicacion numerica 4.88993e+307 * 1.93994e+307 Ok
Comprobando Division Numerica 4.88993e+307 / 1.93994e+307 Ok
Prueba de variables numerica pasada
```

De manera analoga la prueba de variables de texto, crea variables con un texto aleatorio y despues las concatena, comprueba las conversiones Númerico <->Texto, Boolean <->Texto. De igual forma se probaron 10, 100 y 10,000 iteraciones éxitosas, se lista el código de la prueba y parte de los resultados.

```
 void testString(int times = STEPS)
{

   const size_t len = 15;
   for(int i = 0; i < times; ++i){

       std::string str1 = random_string(len);
       std::string str2 = random_string(len);
       Variable var1(str1);
       Variable var2(str2);

       printf("Comprobando Asignacion de Cadenas var%i = %s", i, str1.c_str());
       if(var1.valorString() != str1){
           printf("\nFallo asignacion de cadena %s != %s\n", var1.valorString().c_str(),
           exit(1);
       }
       printf("\tOk\n");
       printf("Comprobando concatenacion de Cadenas %s,%s", str1.c_str(), str2.c_str());
       if( (var1 + var2).valorString() != str1 + str2){
           printf("\nFallo concatenación de Cadenas %s != %s\n",
                  (var1 + var2).valorString().c_str(),
                  (str1 + str2).c_str());
           exit(0);
       }
       printf("\tOk\n");
   }
   std::default_random_engine generator;
```

```
std::uniform_int_distribution<int> dist(0 , 1000);
for(int i = 0; i < times; ++i){

    double val = dist(generator);
    std::ostringstream strs;
    strs << val;
    std::string str = strs.str();

    Variable var_str(str);
    Variable var_num(val);

    printf("Comprobando Conversion Cadena -> Numerico: %s -> %g", str.c_str(), val);
    if(var_str.valorDouble() != val){
        printf("\nFallo %g != %g\n", var_str.valorDouble(), val);
        exit(0);
    }
    printf("\tOk\n");
    printf("Comprobando Conversion Numerico -> Cadena: %g -> %s", val, str.c_str());
    if(var_num.valorString() != str){
        printf("\nFallo\n");
        exit(0);
    }
    printf("\tOk\n");

}

printf("Comprobando Conversion a Logico %s == \"Verdadero\"", "Verdadero");
```

```
    if(!Variable("Verdadero").valorBool()){
        printf("\tFallo Conversion a valos logico\n");
        exit(0);
    }
    printf("\tOk\n");
    printf("Comprobando Conversion a Logico %s == \"Falso\"", "Falso");
    if(Variable("Falso").valorBool()){
        printf("\nFallo Conversion a valos logico\n");
        exit(0);
    }
    printf("\tOk\n");
    printf("Prueba de Variables Cadena pasada\n");

}
```

```
Comprobando concatenacion de Cadenas he315yGe8xdHhyC,5SgDQ9hK9ZV0wLs Ok
Comprobando Asignacion de Cadenas var92 = z2U14azIE8FqNvo Ok
Comprobando concatenacion de Cadenas z2U14azIE8FqNvo,X0EBDeKszSPSQMn Ok
Comprobando Asignacion de Cadenas var93 = ILonKsLKAXQNNlI Ok
Comprobando concatenacion de Cadenas ILonKsLKAXQNNlI,CGGQSR3kJ0ChSd3 Ok
Comprobando Asignacion de Cadenas var94 = GvM2ghs1z0ZNNu6 Ok
Comprobando concatenacion de Cadenas GvM2ghs1z0ZNNu6,e6MuVoKYZbYjI0M Ok
Comprobando Asignacion de Cadenas var95 = LEGiGwN8wK8ThUN Ok
Comprobando concatenacion de Cadenas LEGiGwN8wK8ThUN,l8U70zuKXRw3AE3 Ok
Comprobando Asignacion de Cadenas var96 = XYGlEWhbdbvj4aD Ok
Comprobando concatenacion de Cadenas XYGlEWhbdbvj4aD,SLLuQKrKeOlYPun Ok
Comprobando Asignacion de Cadenas var97 = SRLgAZBp8oR1XVZ Ok
Comprobando concatenacion de Cadenas SRLgAZBp8oR1XVZ,lvs4pIOgd12MZRG Ok
```

```
Comprobando Asignacion de Cadenas var98 = MsffYpChdIV4H1Z Ok
Comprobando concatenacion de Cadenas MsffYpChdIV4H1Z,qkTioIzBzaC1wjR Ok
Comprobando Asignacion de Cadenas var99 = B4JojpcuXFC0JS1 Ok
Comprobando concatenacion de Cadenas B4JojpcuXFC0JS1,qIjJzWcyhZYrYSa Ok
Comprobando Conversion Numerico -> Cadena: 869 -> 869 Ok
Comprobando Conversion Cadena -> Numerico: 630 -> 630 Ok
Comprobando Conversion Numerico -> Cadena: 630 -> 630 Ok
Comprobando Conversion Cadena -> Numerico: 736 -> 736 Ok
Comprobando Conversion Numerico -> Cadena: 736 -> 736 Ok
Comprobando Conversion Cadena -> Numerico: 726 -> 726 Ok
Comprobando Conversion Numerico -> Cadena: 726 -> 726 Ok
Comprobando Conversion Cadena -> Numerico: 1000 -> 1000 Ok
Comprobando Conversion Numerico -> Cadena: 1000 -> 1000 Ok
Comprobando Conversion Cadena -> Numerico: 889 -> 889 Ok
Comprobando Conversion Numerico -> Cadena: 889 -> 889 Ok
Comprobando Conversion Cadena -> Numerico: 233 -> 233 Ok
Comprobando Conversion Numerico -> Cadena: 233 -> 233 Ok
Comprobando Conversion Cadena -> Numerico: 306 -> 306 Ok
Comprobando Conversion Numerico -> Cadena: 306 -> 306 Ok
Comprobando Conversion Cadena -> Numerico: 351 -> 351 Ok
Comprobando Conversion Numerico -> Cadena: 351 -> 351 Ok
Comprobando Conversion Cadena -> Numerico: 513 -> 513 Ok
Comprobando Conversion Numerico -> Cadena: 513 -> 513 Ok
Comprobando Conversion Cadena -> Numerico: 591 -> 591 Ok
Comprobando Conversion Numerico -> Cadena: 591 -> 591 Ok
Comprobando Conversion Cadena -> Numerico: 846 -> 846 Ok
Comprobando Conversion Numerico -> Cadena: 846 -> 846 Ok
```

```
Comprobando Conversion Cadena -> Numerico: 412 -> 412 Ok
Comprobando Conversion Numerico -> Cadena: 412 -> 412 Ok
Comprobando Conversion Cadena -> Numerico: 842 -> 842 Ok
Comprobando Conversion Numerico -> Cadena: 842 -> 842 Ok
Comprobando Conversion Cadena -> Numerico: 269 -> 269 Ok
Comprobando Conversion Numerico -> Cadena: 269 -> 269 Ok
Comprobando Conversion Cadena -> Numerico: 415 -> 415 Ok
Comprobando Conversion Numerico -> Cadena: 415 -> 415 Ok
Comprobando Conversion a Logico Verdadero == "Verdadero" Ok
Comprobando Conversion a Logico Falso == "Falso" Ok
Prueba de Variables Cadena pasada
```

Por último se comprobó el manejo de varibles, creando variables con valores aleatorios, y se probó su búsqueda, actualización y destrucción. Como en los casos anteriores, esto se hizo con 10, 100 y 10,000 iteraciones todas éxitosas. Se incluye la prueba y sus resultados.

```
 void testVariableTable(int times = STEPS)
{
    TablaVariables vars;
    const size_t len = 15;

    for(int i = 0; i < times; ++i){
        std::string str = random_string(len);
        Valor val = {.cadena = str, .numerico = 0, .logico = false};
        printf("Creando la Variable %s", str.c_str());
        vars.crearVariable(str,val,Variable::Cadena);
        printf("\tOk\n");
```

```
printf("Buscando la Variable %s ", str.c_str());
bool exito;
Variable ele = vars.busca(str, exito);
if(!exito){
    printf("\nFallo\n");
    exit(0);
}
printf("\tOk\n");
printf("Comprobando Valor de Variable %s ", str.c_str());
if(ele.valorString() != str){
    printf("\nFallo\n");
    exit(0);
}
printf("\tOk\n");
std::string str2 = random_string(len);
printf("Actualizando la variable %s a %s ", str.c_str(), str2.c_str());
vars.actualizar(str,
                (Valor){.cadena = str2, .numerico = 0, .logico = false},
                Variable::Cadena);
if(vars.busca(str).valorString() != str2){
    printf("\nFallo\n");
    exit(0);
}
printf("\tOk\n");
printf("Comprobando Destruccion de Variable %s ", str.c_str());
vars.remover(str);
```

```
        vars.busca(str, exito);
        if(exito){
            printf("\nFallo\n");
            exit(0);
        }
        printf("\tOk\n");
    }
    printf("Prueba de tabla de variables pasada\n");
}
```

```
Creando la Variable GgclqMQcHaqURxQ Ok
Buscando la Variable GgclqMQcHaqURxQ  Ok
Comprobando Valor de Variable GgclqMQcHaqURxQ  Ok
Actualizando la variable GgclqMQcHaqURxQ a utkcsUR4L5by6S5  Ok
Comprobando Destruccion de Variable GgclqMQcHaqURxQ  Ok
Creando la Variable yjlbSZxqBDOzhqw Ok
Buscando la Variable yjlbSZxqBDOzhqw  Ok
Comprobando Valor de Variable yjlbSZxqBDOzhqw  Ok
Actualizando la variable yjlbSZxqBDOzhqw a 7inpKdJmhdrJZwj  Ok
Comprobando Destruccion de Variable yjlbSZxqBDOzhqw  Ok
Creando la Variable euQPVswTj5e72Lx Ok
Buscando la Variable euQPVswTj5e72Lx  Ok
Comprobando Valor de Variable euQPVswTj5e72Lx  Ok
Actualizando la variable euQPVswTj5e72Lx a wQfkFyNXi38XKhT  Ok
Comprobando Destruccion de Variable euQPVswTj5e72Lx  Ok
Creando la Variable 3LOUitKeK1hy9jH Ok
Buscando la Variable 3LOUitKeK1hy9jH  Ok
```

```
Comprobando Valor de Variable 3LOUitKeK1hy9jH  Ok
Actualizando la variable 3LOUitKeK1hy9jH a 4fhiNvelQMmYt6D  Ok
Comprobando Destruccion de Variable 3LOUitKeK1hy9jH  Ok
Creando la Variable N9WlbEcwqxvXv4E Ok
Buscando la Variable N9WlbEcwqxvXv4E  Ok
Comprobando Valor de Variable N9WlbEcwqxvXv4E  Ok
Actualizando la variable N9WlbEcwqxvXv4E a D7tsnHlROBlmheq  Ok
Comprobando Destruccion de Variable N9WlbEcwqxvXv4E  Ok
Creando la Variable uzzQiZcLTQIOvBR Ok
Buscando la Variable uzzQiZcLTQIOvBR  Ok
Comprobando Valor de Variable uzzQiZcLTQIOvBR  Ok
Actualizando la variable uzzQiZcLTQIOvBR a 9MW1DJIyiG8T2p5  Ok
Comprobando Destruccion de Variable uzzQiZcLTQIOvBR  Ok
Creando la Variable sk3p8lMj6p9MC5W Ok
Buscando la Variable sk3p8lMj6p9MC5W  Ok
Comprobando Valor de Variable sk3p8lMj6p9MC5W  Ok
Actualizando la variable sk3p8lMj6p9MC5W a dCs9D3QT26jAXjx  Ok
Comprobando Destruccion de Variable sk3p8lMj6p9MC5W  Ok
Creando la Variable aZhdPoPlXTZeqlh Ok
Buscando la Variable aZhdPoPlXTZeqlh  Ok
Comprobando Valor de Variable aZhdPoPlXTZeqlh  Ok
Actualizando la variable aZhdPoPlXTZeqlh a MMsCT5GtZGzGOU0  Ok
Comprobando Destruccion de Variable aZhdPoPlXTZeqlh  Ok
Creando la Variable L4Z1iwn5gIYFwOy Ok
Buscando la Variable L4Z1iwn5gIYFwOy  Ok
Comprobando Valor de Variable L4Z1iwn5gIYFwOy  Ok
Actualizando la variable L4Z1iwn5gIYFwOy a ciKUtnX7e6NbLl5  Ok
```

```
Comprobando Destruccion de Variable L4Z1iwn5gIYFwOy  Ok
Prueba de tabla de variables pasada
```

4.1.2. Pruebas de Ejecución

Las pruebas de ejecución tienen como objectivo comprobar que el sistema funciona como un todo, que tanto el lenguaje, la máquina virtual y la interfaz se acoplan de manera adecuada para funcionar coordinadamente como un solo ente. Para ello se probaron dos programas: Seria de Fibonacci y División por 2.

Serie de Fibonacci

La serie de Fibonacci (0,1,1,2,3,5,8,...) aparece en el año 1202 en el libro “Liber Abaci” escrito por Leonardo de Pisa conocido mejor como Fibonacci, esta serie fue observada tras estudiar el crecimiento ideal de poblaciones de conejos, crecimiento de plantas y otros fenomenos naturales y está formulada como:

$$F_n = F_{n-1} + F_{n-2} \tag{4.1}$$

Así la implementación en el lenguaje olinki es:

Como se observa en la figura llamada “Fibonacci en Olinki”, el programa es capaz de general la secuencia de manera correcta.

División por 2

El objetivo de este programa es decidir si un número es par o impar, para ello se usa el operacor módulo % el cual da como resultado el resto de una división entre enteros así 4 % 2 da como resultado 0, mientras que 5 % 2 da resultado 1. Expresado en el lenguaje gráfico Olinki queda de la siguiente manera:

Figura 4.1: Fibonacci en Olinki

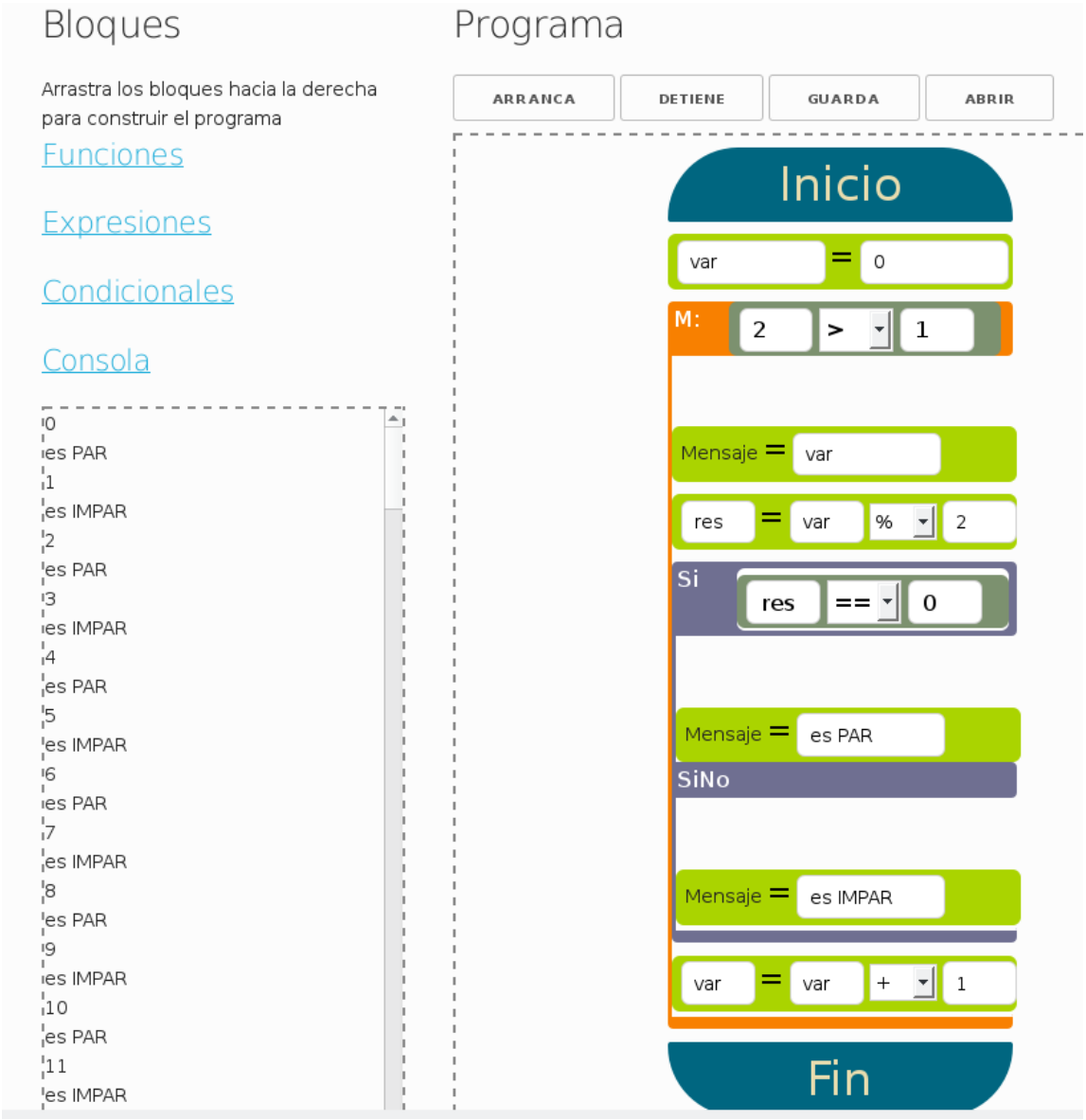

Figura 4.2: Pares e impares

Capítulo 5

Conclusiones

En la opinión de quienes elaboraron este proyecto, la computación es una ciencia que impacta de manera muy directa a la sociedad. Es también una de las que se esperan resultados con mayor fluidez y velocidad puesto que de ella dependen muchas de las competencias tecnológicas hoy en día. Actualmente presenciamos una carrera por ver quién tiene el mejor desarrollo computacional, derivada de un mercado que demanda progresivamente herramientas que les faciliten gran parte de sus tareas diarias, o compartir información, convivir con personas que se encuentran ubicadas a grandes distancias, interpretar datos para establecer pronósticos o tomar decisiones.

La mayor parte de las herramientas que utilizamos para el desarrollo de este proyecto se encuentran disponibles desde hace tiempo o han sido mejoradas constantemente para funciones más detalladas. Lo que el documento presente da conocer no es sí mismo una innovación sino una solución. Recordemos el lema de esta institución es “La técnica al servicio de la patria” y nosotros echamos mano de algunas de las tecnologías a nuestro alcance para brindar a un estrato de la población de un posible instrumento de enseñanza. Se tuvo que partir de conocer qué necesidades adolece un estrato de la población ante las competencias tecnológicas globalizadas para poder identificar de qué manera contribuir como apoyo para su atención. Como lo comentamos en la jus-

tificación, para nosotros resultó un buen mercado la población infantil porque de ella dependerá que en un futuro las decisiones que le darán rumbo a su sociedad. Y creemos que es importe alfabetizar tecnológicamente a las generaciones siguientes para que tengan presente la variedad tan diversa de herramientas con las que cuentan para dar respuesta a las demandas de una sociedad cada vez más competitiva.

Una de las conclusiones a las que llegamos fue que la innovación no siempre está condicionada al desarrollo de un producto jamás antes visto, sino que también se puede manifestar al combinar un conjunto de técnicas y tecnologías para el desarrollo de propuestas que atiendan carencias o necesidades de la sociedad. Otra muy importante es tomar en cuenta que la mayor parte de los avances tecnológicos de las civilizaciones se han dado de manera escalonada y a veces hasta recursiva, lo que implica un grado importante de aprendizaje entre los individuos que la conforman, esto, visto desde un punto de vista pedagógico, es contrsuccionismo.

Glosario

A

Arduino

Microcontrolador enfocado en la creación de proyectos de electrónica y hardware para la comunidad “hazlo tu mismo”.

ARM

Arquitectura de procesadores basados en RISC creados por la compañía ARM

B

Booleano

Tipo de variable o constante matemática que solo puede tener dos valores: Verdadero o Falso comumente representados como 0 y 1

C

C++

Lenguaje de programación orientado a objetos creado en 1983 por Bjarne Stroustrup

Cadena

Secuencia de simbolos, por ejemplo una secuencia de letras del alfabeto español

Coma flotante

Represetación de números reales, recorriendo el punto decimal según la precisión necesaria para represetar una gran variedad de números en el mismo espacio.

Computadora

Máquina electrónica que ejecuta algoritmos representados mediante programas expresados en un lenguaje afin ésta

Cómputo

Acción de transforma información de manera que se pueda aprovechar

Construccionismo

Teoría social y sicologica que considera como los fenómenos sociales se desarrollan desde un contexto social mas general

Constructivismo

Corriente pedagogica que consiste en dar al alumno las herramientas necesarias que le permitan crear sus propios procedemientos para resolver un problema

CSS

Hojas de estilo que formulan las reglas visuales de un archivo HTML

D

Debian Linux

Distribución Linux que fue creada en 1993 por Ian Murdock con el fin de crear un sistema operativo universañ capaz de funcionar en un amplio rango de compuntadoras, desde super computadoras, servidores y sistemas embebidos.

Decoding

También conocido como decodificación, es el paso encargado de tomar una instrucción codificada de forma binaria y activar los subsistemas necesarios de un procesador para su ejecución.

DIV

Etiqueta HTML que se usa para divir el documento en segmentos

DragAndDrop

Acción de arrastrar y soltar un elemento gráfico usando un dispositivo apuntador como el mouse

Dragula

Librería de código libre usada para crear elementos DragAndDrop

E

EDI

Entorno de Desarrollo Integrado, como su nombre lo dice es un sistema que proporciona todas las herramientas necesarias para el desarrollo de programas de cómputo

F

Fetching

Paso que extrae una instrucción desde la memoria de instrucción de un procesador direccionada por el contador de programa

Fibonacci

Serie númerica de crecimiento exponencial: 0,1,1,2,3,5,8, ...

G

Gramática

Conjunto de reglas de producción de un lenguaje

GUI

Graphic User Interface o Interfaz gráfica de usuario, es el componente de software con la que el usuario interactua usando elementos gráficos desplgados en una pantalla

H

HTML

Hyper Text Marking Lenguaje o Lenguaje de Marcadores de Hiper Texto, Esun conjunto de etiquetas usado para describir documentos

I

Intérprete

Programa de Cómputo que toma como entrada otro programa expresado en algún lenguaje de programación y lo ejecuta paso a paso

J

JavaScript

Lenguaje de programación multi paradigma principalmente usado en aplicaciones WEB

JSON

JavaScript Object Notation o Notación de Objectos JavaScript, es un modelo de represetar objetos del lenguaje JavaScript mediante texto plano

L

LED

Light Emitter Diode o Diodo Emisor de Luz, es un componente electrónico que al ser polarizado emit Luz de cierto color

Lenguaje

Conjunto de Palabras

Linux

Sistema operativo creado en 1991 por Linus Torvalds

Lógica

Rama de las matemáticas que tiene como estudio la expresión de sistemas formales

LOOP

Registro de la máquina virtual Olink encargado de llevar la cuenta dentro de ciclos

M

Mecanísmo

Estructura u Objeto con partes funcionales móviles

Microcontrolador

Dispositivo electrónico constituido por un procesador, memoria de datos, memoria de programa y sistemas de entrada/salida

O

Olinki

Nombre del Lenguaje de programación propuesto por este proyecto

OSX

Nombre del sistema operativo encontrado en computadoras de la marca Apple

P

PC

Program Counter o Contador de Programa, registro especualizado de la máquina virtual Olinki encargado de direccionar la instrucción a ejecutar
También hace referencia a Personal Computer, es decir una computadora de uso personal

Pedagogía

Disciplina educativa que tiene como objeto de estudio la educación y enseñanza

Programación

Acción de crear o construir un programa en algún lenguaje afín a un sistema de Cómputo

QObject

Representación de un Objecto por parte de las librerías Qt dentro del paradigma orientado a objetos

Qt

Librerías de código libre que incluyen herramientas para el desarrollo de aplicaciones gráficas

R

Raspberry Pi

Computadora de bajo costo desarrollada por la fundación Raspberry con el fin de asistir en la enseñanza en los salones de clases

RIEB

Reforma Integral de Educación Básica. Política educativa nacional que tiene como objetivo la articulación y el desarrollo de la currícula escolar

S

SP

Stop Pointer, Registro especualizado en la máquina virtual Olinki cuyo propósito es mantener presente la condición de parada de un contexto de ejecución

STL

Standar Template Library, Librería del lenguaje C++ que incluye herramientas para la manipulación y creación de contenedores

U

UML

Unified Modeling Languague o Lenguaje de Modelado Unificado, Es un proceso de modelado general usado en la descripción de relaciones entre entidades de un sistema

V

VM

Virtual Machine o Máquina Virtual, es la implementación de una computadora real o ficticia usando software

W

WebKit

Librerías para el renderizado y manejo de documentos HTML

WIMP

Window Icon Menu Pointer o Ventana Ícono Menú y Puntero, es un paradigma de interacción entre usuarios y sistemas de software usando elementos gráficos

Windows

Sistema Operativo desarrollado por la empresa Microsoft para su uso en computadoras personales

Bibliografía

[1] J. ADELL. *Redes y Educación. Nuevas Tecnologías, Comunicación audiovisual y educación*. Barcelona: Ed. Cedecs.

[2] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. *Compiladores: principios, técnicas y herramientas*. Pearson Educación, 1998.

[3] Alfred V Aho, Jeffrey D Ullman, et al. *Principles of compiler design*. Addision-Wesley Pub. Co., 1977.

[4] Manuel Alfonseca Moreno et al. Compiladores e intérpretes: teoría y práctica. *Editorial Prentice Hall. Madrid*, 2006.

[5] Hans-Georg GADAMER. *Verdad y Método. Fundamentos de Hermenéutica filosófica*. Edic. Sigueme, Salamanca, 1988.

[6] K. Gergen. *Realidades y Relaciones: aproximaciones a la construcción social*. Ediciones Paidos Iberica. Barcelona, España, 1996.

[7] Antti Herala et al. Applying qt to programming courses. 2013.

[8] Schmuller J. *Aprendiendo UML en 24 Horas*. Prentice Hall. México, DF., 2006.

[9] Andreas Krall. Efficient javavm just-in-time compilation. In *Parallel Architectures and Compilation Techniques, 1998. Proceedings. 1998 International Conference on*, pages 205–212. IEEE, 1998.

[10] Lifelong Kindergarten Group at the MIT Media Lab. Create stories, games, and animations Share with others around the world, 2013, mayo.

[11] Logo Foundation. Logo Update, (21 ed.), 2001, Diciembre.

[12] M. Mateos Cruz. Programación y educación: qué países la tienen en su plan de estudios.(1ra ed.), 2014, Septiembre, 1.

[13] Seymour P. *La máquina de los niños*. Argentina: Ediciones Paidos, 1995.

[14] M. Pulaski. *El desarrollo de la mente del niño según Piaget*. Buenos Aires: Ediciones Paidos Ibérica, 1978.

[15] Saarland University. the alice system. (14 ed.), 2014, Julio, 31.

[16] SEP. *Plan de estudios 2011 – Educación básica*. Secretaría de Educación Pública, Cuauhtémoc, México DF, 2011.

[17] Bill Venners. *Inside the Java virtual machine*. McGraw-Hill, Inc., 1996.

Índice de figuras