

# Universidad Central “Marta Abreu” de Las Villas

## Facultad de Matemática, Física y Computación



Trabajo para optar por el título de Máster en Ciencia de la  
Computación

## Plataforma de apoyo al proceso de enseñanza-aprendizaje de la algoritmización con un enfoque de Currículo Invertido

### **Autor**

Lic. José Albert Cruz Almaguer

### **Tutor**

Dr. Rafael Arturo Trujillo Rasúa

Santa Clara, marzo de 2011



*A mi madre, ese ser sacrificado al que le debo todo.*



# Resumen

El proceso de enseñanza-aprendizaje de la solución de problemas mediante computadoras es una actividad que presenta varios retos a estudiantes y a profesores. Dentro de la comunidad de investigadores de la enseñanza de las Ciencias de la Computación se realizan constantes esfuerzos para identificar las razones que hacen a la programación una habilidad difícil de adquirir. Entre los obstáculos encontrados están los medios usados para la enseñanza, en particular el lenguaje de programación y el tipo de ejercicios desarrollados.

La presente investigación propone un entorno de programación que incluye un lenguaje gráfico y un marco de trabajo para el montaje de micromundos. El orden de los contenidos y el tipo de ejercicios que permite la herramienta obedecen al enfoque de Currículo Invertido. Los resultados obtenidos ponen en manos de los profesores un medio de enseñanza personalizable que facilita el montaje de cursos introductorios de programación. Se hace, además, una exposición informal de algunos de los conceptos de la programación utilizando un caso de estudio.

**Palabras claves:** Enseñanza de la programación, lenguaje, Currículo Invertido, micromundos.





# Índice general

<b>Introducción</b>	<b>1</b>
Objetivo general . . . . .	3
Objetivos específicos . . . . .	3
Estructura del documento . . . . .	4
<b>1 Solución de problemas mediante computadoras</b>	<b>5</b>
1.1 Alfabetización computacional . . . . .	5
1.1.1 Pensamiento computacional . . . . .	6
1.2 Solución de problemas mediante computadoras . . . . .	7
1.2.1 Solucionar problemas . . . . .	7
1.3 La computación, herramienta de solución . . . . .	7
1.4 Consideraciones sobre la programación . . . . .	8
1.4.1 Conceptualizaciones erróneas . . . . .	9
1.4.2 Entornos de programación para principiantes . . . . .	10
1.4.3 Entornos gráficos de programación . . . . .	12
1.4.4 Scratch . . . . .	13



**ÍNDICE GENERAL**

1.4.5	Roles de los lenguajes de programación . . . . .	15
1.4.6	Los ejercicios a utilizar . . . . .	16
1.5	Currículo invertido . . . . .	16
1.5.1	El enfoque . . . . .	17
1.5.2	Requerimientos para aplicar el enfoque . . . . .	17
1.5.3	Variante para todos los niveles: los objetos primero . . . . .	18
1.6	Conclusiones parciales . . . . .	18
<b>2</b>	<b>Plataforma para la solución de problemas computacionales</b>	<b>19</b>
2.1	Diseño . . . . .	19
2.1.1	Finalidad y servicios que brinda . . . . .	19
2.1.2	Finalidad del entorno desarrollado . . . . .	20
2.1.3	Clientes de la plataforma . . . . .	20
2.1.4	Requerimientos que provee . . . . .	20
2.1.5	Entorno de construcción de las soluciones . . . . .	21
2.1.6	Micromundos . . . . .	25
2.2	Consideraciones sobre la implementación . . . . .	25
2.2.1	Interfaz de programación para los micromundos . . . . .	25
2.2.2	Implementación del micromundo robot . . . . .	27
2.2.3	Arquitectura . . . . .	29
2.2.4	Tecnologías usadas en la implementación . . . . .	30
2.2.5	Funcionamiento del sistema . . . . .	32
2.3	Conclusiones parciales . . . . .	35

<b>3</b>	<b>Aplicación de la plataforma a un caso de estudio</b>	<b>37</b>
3.1	Objetos . . . . .	37
3.2	Interfaces . . . . .	40
3.3	Consultas y órdenes . . . . .	41
3.4	Algoritmo . . . . .	42
3.5	Problema computacional . . . . .	43
3.6	Estado . . . . .	43
3.6.1	Entrada/salida . . . . .	44
3.7	Especificación de un problema computacional . . . . .	44
3.7.1	Instancia de un problema computacional . . . . .	45
3.7.2	Solución al problema anteriormente especificado . . . . .	45
3.8	Computadora . . . . .	46
3.9	Información y datos . . . . .	47
3.9.1	Tipo de datos . . . . .	47
3.10	Lenguaje de programación . . . . .	48
3.11	Instrucciones de computadora . . . . .	48
3.12	Programa . . . . .	48
3.12.1	Programa orientado a objetos . . . . .	49
3.13	Corrección de un programa . . . . .	49
3.14	Programar una computadora . . . . .	50
3.15	Variable . . . . .	50
3.15.1	Instrucciones para manipular variables . . . . .	50
3.16	Conclusiones parciales . . . . .	51

## ÍNDICE GENERAL

Conclusiones y recomendaciones	53
Bibliografía	55
A Uso del marco de trabajo	61
B Personalización de las estructuras de control	63
C Configuración de las aplicaciones	65
Índice de figuras	67
Índice de tablas	69
Índice de códigos	71

# Introducción

El ser humano se distingue de los demás seres por su capacidad de construir herramientas que le extiendan sus capacidades mentales. La computación es una de las principales creaciones humanas destinadas a apoyar el intelecto, su utilidad es comparable a la del lenguaje y la escritura.

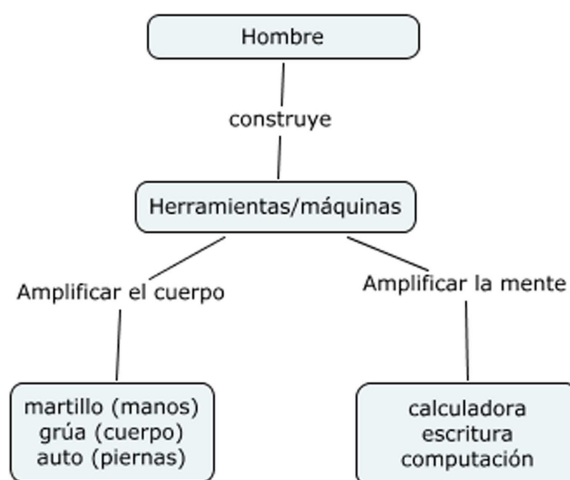


Figura 1: La computación es para la mente lo que un automóvil es para las piernas

La computación está presente en múltiples esferas de la vida del hombre: en la elaboración de productos audiovisuales, en el procesamiento de datos para la predicción del clima, en los bancos financieros, etc., pero sobre todo constituye un instrumento poderoso de resolución de problemas. Esto último hace que aprender computación aporte a la formación general un valor a un nivel similar que las Matemáticas. La principal vía de resolver problemas mediante la computación es la **programación**, y es a la enseñanza de esta habilidad, como modesto aporte, que está destinado este trabajo.

La solución de problemas mediante computadoras, enmarcada en el contexto del primer año de estudios de las carreras de Informática (Ingeniería Informática y Ciencia de la Computación), conocida como programación, es una actividad que presenta varios retos tanto a los estudiantes como a los profesores.

En la Universidad de las Ciencias Informáticas se usan, como medio de iniciar a los estudiantes en la

programación, los lenguajes Java, C# y C++. Las razones que llevan a esto son de diversa naturaleza y tienen por lo general en su esencia la creencia de que no importa el lenguaje que se use, siempre que soporte el paradigma orientado a objetos.

Cuando un estudiante comienza sus estudios de programación, se encuentra con palabras o expresiones tales como: `static`, `void`, `public`, `main` y `System.out.println` que conforman un vocabulario difícil de asimilar y que debe hacer corresponder con descripciones de un mundo artificial. Esto hace que con facilidad los estudiantes cometan errores en sus soluciones aunque los algoritmos que hayan pensado sean correctos. Además cuando los entornos de programación informan sobre los errores cometidos lo hacen utilizando mensajes especializados que no son legibles para un programador novel. En resumen, los entornos y lenguajes profesionales hacen muchas veces que el proceso de traducción de un algoritmo correcto en programa sea engorroso, y consecuentemente el estudiante demore mucho tiempo en comprobar si la solución diseñada es correcta o no.

Por otra parte, un estudiante, antes de llegar a la universidad tiene por lo general múltiples encuentros con medios virtuales tales como video-juegos y programas para el trabajo en oficinas los cuales poseen interfaz gráfica de usuario; sin embargo, cuando construye sus primeros programas los hace con interfaz de consola y sobre dominios alejados normalmente de sus intereses (tales como simples análisis aritméticos). Este tipo de interfaz es poco usado y da por lo tanto la idea de que lo producido carece de utilidad, desmotivando el aprendizaje.

Especialistas de renombre como (Meyer 1997a) y diferentes investigaciones (Kolling 1999a, Kolling 1999b) han notado la importancia del lenguaje y el entorno de desarrollo que se usen en los primeros momentos de la enseñanza. Otros (Mark Guzdial 2002) han precisado que para la *generación del Nintendo*<sup>1</sup> es necesario utilizar ambientes y realizar ejercicios con marcada riqueza visual.

Como parte de los esfuerzos por transformar la enseñanza de la programación, en los últimos años se ha llevado a la práctica un grupo de ideas que invierten el orden clásico de contenidos de un curso tradicional de programación y que recibe el nombre de *Currículo Invertido*. Este enfoque fue utilizado primeramente en el área de la Ingeniería Eléctrica (Cohen 1991) siendo vista su aplicabilidad en la Ingeniería del Software con prontitud (Meyer 1993). El mismo ha encontrado su mayor desarrollo en el diseño de cursos de Programación Orientada a Objetos (Meyer 2003), en los cuales se comienza consumiendo clases y objetos existentes en bibliotecas para luego, una vez entendidos los principios de la programación, pasar a actuar como productores.

Hasta el momento este enfoque no ha sido usado en las universidades cubanas puesto que aplicarlo demanda el cumplimiento de precondiciones muy exigentes. Sin embargo algunas de las ideas que maneja son lo suficientemente básicas y simples como para aplicarlas.

Esta situación da lugar al ***problema científico*** a resolver en este trabajo:

---

<sup>1</sup>Denominación dada por este investigador a los jóvenes de inicio de siglo con amplio acceso a video-juegos.

Las características, tanto de los entornos de programación profesionales como de los enfoques desarrollados comúnmente en los cursos iniciales de programación, hacen poco atractivo el proceso de enseñanza-aprendizaje de los conceptos básicos de esta materia.

### Preguntas de investigación

Según la problemática planteada se elaboraron las siguientes preguntas de investigación:

1. ¿Es adecuado el enfoque de Currículo Invertido para el diseño de un curso introductorio de algoritmización que exponga los conceptos de mayor relevancia de manera motivante y simple?
2. ¿Qué características debe tener un lenguaje de programación para facilitar su uso por parte de estudiantes principiantes?
3. ¿Qué arquitectura es adecuada para el desarrollo de un entorno de programación orientado a principiantes?
4. ¿Cuál sería un caso de estudio adecuado para exponer conceptos de programación usando las ideas del enfoque de Currículo Invertido?

Dado el hecho de que en los niveles de enseñanza secundario y preuniversitario se ha reducido la enseñanza de la programación, se considera que la existencia de medios de enseñanza-aprendizaje de apoyo a la misma pudieran ayudar a la inclusión de algunos temas de la materia en el currículo de dichos niveles.

Por lo anteriormente notado se considera como **hipótesis** en la investigación lo siguiente:

Usando los principios del Currículo Invertido, un lenguaje de programación gráfico que incluye construcciones sintácticas cercanas al lenguaje natural y un entorno que soporte su utilización, se puede diseñar un curso sencillo y con facilidades visuales cuyo proceso de enseñanza-aprendizaje sea motivador para los estudiantes.

## Objetivo general

---

Desarrollar una herramienta que facilite la construcción de programas para el apoyo del proceso de enseñanza-aprendizaje de la algoritmización usando el enfoque de *Currículo Invertido*.

### Objetivos específicos

- Definir un lenguaje gráfico de programación que conste de estructuras de control, invocación a métodos, asignación y declaración de variables.

- Construir un micromundo que sirva de caso de estudio.
- Establecer una arquitectura distribuida para la interacción entre el lenguaje y los micromundos.

## Estructura del documento

---

La tesis se estructura en tres capítulos. En el capítulo 1 se aborda el estado del arte de la enseñanza de la programación a principiantes, el capítulo 2 se encarga de describir el sistema propuesto y en el capítulo 3 se expone una posible manera de utilizar el entorno para explicar algunos de los conceptos básicos de la programación.

# Solución de problemas mediante computadoras

La civilización humana debe gran parte de su desarrollo a la creación de herramientas. Muchas de ellas tienen por objetivo aumentar la fuerza de nuestros brazos, otras persiguen hacer más rápidas a nuestras piernas. Además se han creado algunas capaces de extender las capacidades intelectuales: el lenguaje que permite transmitir las ideas, y la escritura que permite mantener dichas ideas en el tiempo y trasladarlas en el espacio. La computación es una de las principales herramientas capaces de extender nuestra mente: permite amplificar prácticamente cualquier actividad intelectual, y ha logrado cambiar la forma en que se resuelven muchos problemas (Evans 2010*d*).

## 1.1 Alfabetización computacional

---

Tradicionalmente se entiende por alfabetización la adquisición de las habilidades de leer, escribir y hacer cálculos aritméticos; sin embargo, en el actual siglo, cada día es más común que las personas se encuentren frente a medios digitales de diversos tipos con los cuales han de producir, consumir o entregar información. Por ello se ha llegado a definir un concepto de alfabetización computacional con el cual orientar mejor a la sociedad hacia la formación mediante los sistemas educacionales de estas habilidades. La versión más actual de este concepto se encuentra en (SooHwan Kim & Kim 2009):

**Definición 1.1 (Alfabetización computacional)** *Estar alfabetizado computacionalmente es ser capaz de expresar ideas, y de entender lo producido por otros con un acercamiento informacional; así como*



*utilizar el medio digital como herramienta de comunicación. Entendiendo acercamiento informacional como la habilidad de resolver problemas de la vida real utilizando el **pensamiento computacional**.*

En algunos círculos (Bogost 2005, Bogost 2007) se entiende que dicha alfabetización está contenida dentro de la *alfabetización procedural*, que posee posibilidades de influir en la manera en que se entienden tan disímiles áreas como el aprendizaje, la política y la historia.

### 1.1.1 Pensamiento computacional

La versión más adecuada encontrada de este término es la de la profesora Jeannette M. Wing de la universidad Carnegie Mellon de Pittsburgh en EEUU.

**Definición 1.2 (Pensamiento computacional)** *La utilización de los conceptos básicos de la Ciencia de la Computación para la solución de problemas, el diseño de sistemas y el entendimiento del comportamiento humano (Wing 2008).*

La profesora Wing identifica dos categorías: la abstracción y la automatización, con las características que se refieren en la tabla 1.1 (SooHwan Kim & Kim 2009).

Categoría	Descripción
Abstracción	<ul style="list-style-type: none"><li>• Escoger las abstracciones adecuadas.</li><li>• Actuar en varios niveles de abstracción a la vez.</li><li>• Definir las relaciones existentes entre dichos niveles.</li></ul>
Automatización	Pensar en términos de mecanizar la abstracción, las capas (niveles) y sus relaciones. Dicha mecanización se logra mediante diferentes modelos y notaciones.

Tabla 1.1: Categorías del pensamiento computacional

La habilidad más importante del pensamiento computacional es la definición de abstracciones a varios niveles. Ella constituye la herramienta mental de la computación y a la vez es la mecanización de nuestras abstracciones lográndose que las herramientas mentales sean potenciadas por ellas mismas (Wing 2008).

## 1.2 Solución de problemas mediante computadoras

---

Los problemas son obstáculos o interrogantes a los que el ser humano se enfrenta continuamente. Una vez que la respuesta a la interrogante es encontrada o el obstáculo vencido se da por resuelto y se pasa al siguiente problema (Evans 2010c) .

### 1.2.1 Solucionar problemas

Según el matemático George Polya (Polya 1957), cuando se resuelven problemas intervienen cuatro etapas mentales:

1. Entender el problema

¿Qué es desconocido? ¿Qué datos son conocidos? ¿Cuáles son las restricciones? ¿Es posible satisfacerlas? ¿Existen contradicciones o redundancias? En esta etapa se crean diagramas y se seleccionan notaciones.

2. Trazar un plan de solución

Encontrar la relación entre los datos existentes (la entrada) y lo que se quiere saber. ¿Hemos visto ya este problema o alguno parecido? ¿El resultado esperado se parece al de algún otro problema? División del problema en otros más pequeños.

3. Ejecutar el plan

Llevar a cabo cada acción decidida en la etapa 2, analizando en cada momento la pertinencia de la misma, listo para regresar al plan si se encontrara algo incorrecto.

4. Revisar

Analizar la solución, verificar que es correcta y que por lo tanto se solucionó el problema.

Elas son empleadas de manera cíclica y dinámica tal y como se muestra en la figura 1.1: en el intento de hacer el plan se puede concluir que no se ha entendido del todo el problema y por lo tanto hay necesidad de regresar a esta etapa; o cuando se va a ejecutar el plan puede que un estudiante no sepa cómo hacerlo y entonces intente con un nuevo plan (James Wilson 1993).

## 1.3 La computación, herramienta de solución

---

La computación estudia los *procesos de información*. Un proceso es una secuencia de pasos, al describirlo enumerando las etapas (pasos) que lo componen se crea un procedimiento. Aquellos que puedan ser

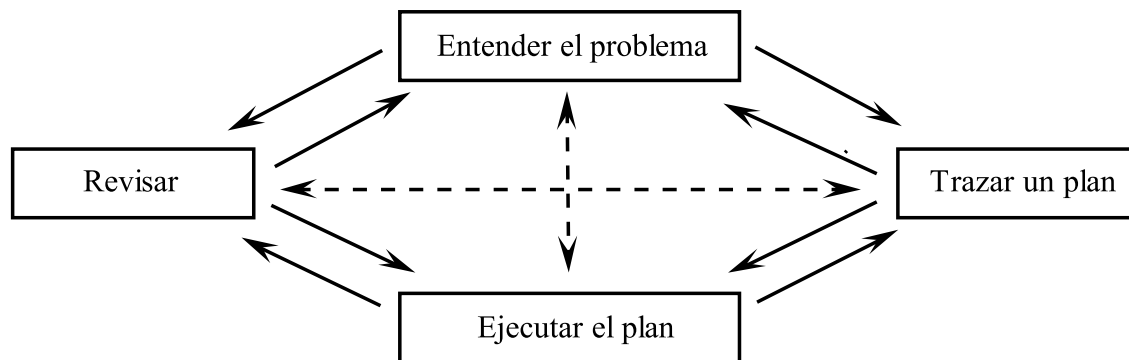


Figura 1.1: Naturaleza cíclica de las etapas para la solución de problemas planteadas por Polya

seguidos sin el empleo de ningún tipo de razonamiento son llamados *procedimientos mecánicos* (Evans 2010d).

Muchos problemas constituyen búsquedas de incógnitas a partir de datos conocidos y además poseen semejanzas entre sí. Dichos problemas abren la posibilidad de enfocarlos desde una perspectiva informacional. Al enfrentarse a un problema, un conocedor de la computación, no solamente piensa en solucionarlo sino que lo ve como una correspondencia entre la entrada y las salidas deseadas, y en función de ello desarrolla un procedimiento que lo resuelva para cada posible entrada (Evans 2010a).

Los algoritmos son procedimientos mecánicos que siempre terminan y que establecen una relación entre una información de entrada y una de salida, es por esto que constituyen una excelente herramienta de solución de problemas. Sin embargo, ellos solamente indican la solución, y necesitan por tanto de un ente que los interprete y llegue a la solución. Este agente puede ser un ser humano, pero es el uso de las computadoras el que los hace realmente útiles. La traducción de un algoritmo dado a un lenguaje que pueda ser obedecido por una computadora se conoce como programación. Aunque la acepción básica de programación sea esta (en la cual programar es traducir ideas ya existentes, *codificar*), los cursos de programación son los encargados de enseñar a construir algoritmos.

El estudio de las características de los algoritmos y de la forma de crearlos confiere una especial importancia a la alfabetización computacional. Esto hace que enseñar a programar computadoras posea valor en el contexto de la formación de profesionales del ramo, así como en la formación de la cultura básica de los seres humanos del siglo XXI.

## 1.4 Consideraciones sobre la programación

En la educación, la programación de computadoras puede ser un *medio ideal mediante el cual los estudiantes expresen su creatividad, desarrollen nuevas ideas y aprendan a comunicarlas* (A. Begel 2005). Aprender a programar puede ser una tarea difícil, además de aprender sintaxis confusas y poco

intuitivas, el estudiante se ve obligado a estructurar su pensamiento y sus soluciones, y a entender la ejecución de sus programas (Caitlin Kelleher 2005).

Diseñar procesos en detalle implica poner en concordancia ideas: conceptos, requisitos, consideraciones iniciales, etc. Esas características exigen gran coordinación mental para lograr plasmarlas en un artefacto que operará independientemente de su creador.

### 1.4.1 Conceptualizaciones erróneas

Son numerosos los casos en los que transferencias de conocimientos desde otras áreas hacen que se entiendan incorrectamente conceptos de la programación. A continuación se describen algunas.

#### El idioma

Los términos no siempre significan lo mismo en el lenguaje natural nativo de los estudiantes que en programación (transferencia lingüística) (Clancy 2005). Trabajando con estudiantes universitarios que aprendían Pascal, fueron notadas inconcordancias entre el Inglés y el vocabulario de programación (Spohrer & Soloway 1986, Bonar & Soloway 1989, Pea 1986), por ejemplo:

- La palabra **while** implica en inglés una prueba continuamente activa como es el caso en “*while the highway stays along the coast, keep following it north*”, mientras que en Pascal la iteración **while** hace una prueba por cada iteración. Un estudiante que haga una analogía entre los dos puede esperar que un ciclo termine en el momento en que la expresión de prueba sea satisfecha.
- En el caso de la estructura **if**, estudiantes de BASIC pensaban que ella “esperaba que se cumpliera” (detenía la ejecución) la condición para luego realizar el **then**.

Aunque estas conclusiones son relevantes fundamentalmente para personas de habla inglesa, sirven para hacer notar la importancia que tienen la claridad de las estructuras sintácticas que se utilicen. Además siempre es aconsejable ser coherentes con el significado que tienen los términos fuera del ámbito de la programación.

#### La notación matemática

Otra fuente de sobregeneralización es la notación algebraica. Varios autores (Bayman & Mayer 1983, R. Putnam 1986) han observado la confusión en los novicios por la sentencia de asignación  $cant = cant + 1$  que se da en lenguajes de la familia C y en Fortran: ¿cuántos estudiantes que comienzan con lenguajes como estos se han preguntado cuáles son los valores de  $cant$  para los que esta expresión tiene sentido? Y tal como lo han hecho notar Wirth y Dijkstra (Wirth 2002): ¿Por qué  $a = b$  no significa lo mismo que  $b = a$ ?

### Las analogías

Un modelo analógico “es por definición un mapeo parcial al sistema de cómputo de lo que se trata de explicar. Ninguna analogía simple es suficiente para explicar completamente la manera de operar de un sistema de cómputo debido a las grandes diferencias que posee respecto a los demás sistemas. Muchas veces esto obliga a que se usen varios modelos y para hacer efectivo su uso se lleva a los usuarios ante una gran confusión al tener que ubicarse en las inferencias adecuadas y desechar las múltiples incorrectas o irrelevantes sugeridas por la analogía” (Halasz & Moran 1982). Son varios los casos de aplicación errónea de analogías, ejemplo de ello es cuando los profesores se refieren a las variables como a cajas: dado que una caja puede almacenar más de un objeto, los estudiantes tienden a pensar que las variables también (Boulay 1989) pueden hacerlo.

Para evitar esto es importante que siempre se le presente al estudiante una argumentación simple, rigurosa y lógica del modo de trabajo de las computadoras (modelo de cómputo), de sus partes constitutivas principales así como de los conceptos asociados a la solución de problemas mediante equipos de cómputo. Por otra parte se dejará a los ejercicios que se desarrollen la tarea de simular el mundo ya conocido o imaginado.

### 1.4.2 Entornos de programación para principiantes

La creación de ambientes de programación para principiantes ha tenido especial relevancia a partir del momento en que se concluyó que la habilidad de programar es difícil tanto de adquirir como de enseñar (Boulay 1989, R Pea 1987, Martin 1986). La práctica docente e investigativa ha notado que los estudiantes tienen problemas con las construcciones iterativas, con las de selección, con el ensamblado de componentes y con el uso de otras muchas construcciones de los lenguajes de programación. Por lo que si se logra un medio de trabajo que las haga manejables se puede lograr que aprender a programar sea más fácil.

Establecer comparaciones entre ambientes para novicios es muy difícil dado que no existe una sólida base teórica con métodos de medición definitivos (Guzdial 2005). Entre otras, se desconocen las respuestas a las preguntas: ¿cómo se aprende a programar? ¿qué se entiende primero? ¿cuáles son las precondiciones necesarias para el aprendizaje? Se sabe mucho de los problemas que los estudiantes tienen al aprender a programar pero poco de las causas de esos problemas.

### Familias de ambientes de programación

Existen particularmente tres líneas de desarrollo de ambientes que han tenido un profundo impacto en todo el desarrollo subsiguiente de los entornos (Guzdial 2005):

**La familia Logo:** variante de Lisp que se ha ramificado de muchas maneras.

**Basada en reglas:** descendiente del propio Logo, de Smalltalk-72 y de Prolog.

**Lenguajes de programación tradicionales:** enmarca a las variantes que hicieron aportes en la metodología de los cursos, logrando mayor protagonismo de los estudiantes. Para estos casos no tuvo importancia la elección del lenguaje.

### Logo

Logo fue desarrollado a mediados de la década de 1960 por Wally Feurzeig y Danny Bobrow en BBN Labs junto a Seymour Papert del MIT. La divisa fundamental de Logo fue la pregunta “¿por qué un estudiante debe aprender a programar?”, que en el momento en que se crea Logo tuvo por respuesta: *para que los estudiantes analicen su pensamiento y se expresen mediante los programas, obteniendo habilidades intelectuales superiores al revisar lo que hacen.*

Los creadores de Logo lo entendieron como un medio para introducir a los estudiantes en el campo de las Matemáticas. Según ellos, la herramienta permitiría el aprendizaje de esta materia de la misma manera en que los niños aprenden una lengua extranjera cuando están en un país en el que la misma es la norma. Por otra parte, fue una firme tesis entre ellos la idea de que programar una computadora es equivalente a enseñarle a una persona a hacer algo (Harvey 1997).

Logo fue muy criticado luego de un estudio hecho por el Bank Street College en el que se planteó que no se obtenían beneficios cognitivos al usarlo. Años después se analizó que se habían utilizado muy pocos estudiantes y que no todas las variables fueron adecuadamente controladas. Dado que solamente unos pocos estudiantes habían aprendido a programar la conclusión más evidente era que aprender a programar es difícil.

### Smalltalk-72

Con Smalltalk-72 Alan Kay y otros miembros del Xerox PARC Learning Research Group extendieron el modelo de Logo de muchas maneras. Este lenguaje-entorno tuvo como propósito el aprendizaje mediante la creación y exploración de la gran variedad de medios habilitados por la computadora.

Kay, al igual que los creadores de Logo, veía en las computadoras un medio de expresión de ideas, sin embargo sentía que el poder computacional que Logo brindaba era muy débil. Este brillante investigador acabó creando un sistema de programación orientada a objetos como medio para permitir la creación de artefactos más complejos que facilitaran el acceso a dominios ricos. Al hacerlo fue creada la interfaz de usuario de escritorio que se conoce en la actualidad.

El resultado que obtuvieron fue exitoso: en estudios realizados con niños por Kay y Goldberg se vio que los mismos eran capaces de producir programas impresionantes tales como animaciones, sistemas

musicales y herramientas de dibujo. Desafortunadamente toda esta iniciativa en el campo pedagógico fue eclipsada por el desarrollo de Smalltalk como entorno profesional para la creación de software.

### Boxer

Boxer fue otra de las extensiones realizadas a Logo, en este caso no se dirigió el esfuerzo hacia una tarea en específico. Este sistema fue el resultado del análisis de qué apariencia tendría la computación si fuera entendida como una habilidad tan básica como la lectura (o la escritura).

DiSessa, su creador, estaba convencido de que en los lenguajes de programación tradicionales hay muchos hechos escondidos y abstractos. Según él, lo que hace más difícil a la programación es la interfaz de los entornos de programación y la manera en que se relacionan con el lenguaje materno de los estudiantes (diSessa 2001). La conclusión obtenida fue que la programación es una actividad desafiante pero si se aprendiera desde edades tempranas y fuera practicada por todos sería mucho más fácil de entender.

### 1.4.3 Entornos gráficos de programación

Se han creado muchos sistemas de programación gráficos para romper las barreras que existen ante la programación de computadoras, ayudando a simplificar la codificación. En ellos, en lugar de trabajar exclusivamente con texto y obligar a los usuarios a seguir complicadas reglas sintácticas, se permite manipular e interactuar con objetos visuales para construir proyectos de programación.

En estos sistemas, los usuarios arrastran bloques desde paletas hasta el área principal. Los bloques constituyen los comandos del lenguaje y los usuarios solamente pueden unir bloques que estén relacionados. Las relaciones están dadas por reglas simples, manifestadas gráficamente y fáciles de memorizar permitiendo que la sintaxis del lenguaje quede representada visualmente. Algunos sistemas gráficos de programación han eliminado totalmente la presencia de textos o etiquetas permitiendo solamente el uso de imágenes. En el sistema StageCast (Stagecast 2010), por ejemplo, el usuario debe especificar estados anteriores y posteriores mediante imágenes como las que se muestran en la figura 1.2.



Figura 1.2: Estados anteriores y posteriores de un agente en Stagecast

### Programación gráfica basada en nodos

En la programación basada en nodos los usuarios manipulan y enlazan nodos, cada uno de los cuales realizan operaciones específicas. Los enlaces representan el flujo de los datos y la estructura resultante consiste en un grafo dirigido, que brinda a los usuarios una vista sobre el flujo de los datos y del programa. Inspeccionando la entrada y salida operada sobre cada nodo se puede saber cuáles son los estados por los que pasa el nodo que se quiera analizar.

Un ejemplo de lenguajes basado en nodos es el Quartz Composer (Composer 2010). Cuando un usuario quiere aplicar a una imagen un efecto difuminador (blur) lo que debe hacer es enlazar un nodo que representa la imagen con otro que realiza dicho efecto.

### Reescritura gráfica de reglas

En lugar de obligar a los usuarios a convertir sus ideas en código, estos sistemas permiten expresar el comportamiento de los objetos mediante reglas de reescritura. La figura 1.2 muestra un caso de regla con la cual se expresa que un agente debe ir a la parte superior de una jarra si se encuentra a la izquierda. En estos sistemas es muy fácil crear animaciones y simulaciones, no siendo así para problemas de otra naturaleza.

### Programación gráfica basada en bloques

En la programación gráfica basada en bloques los usuarios construyen sus programas conectando objetos tal y como harían con piezas de rompecabezas. Las características de estos bloques son las que dictan la sintaxis del lenguaje, lográndose reducir la curva de aprendizaje.

Son muchos los lenguajes basados en bloques que se han creado: Bongo, Flogo, Mindstorms, Tangible Programming Bricks, StartLogo TNG y Scratch. A continuación se describe brevemente este último con el ánimo de mostrar con más claridad sus características.

#### 1.4.4 Scratch

Scratch (figura 1.3) es un lenguaje visual de programación desarrollado por el Lifelong Kindergarten Group en el MIT Media Lab; soporta un estilo de trabajo basado en la mezcla de imágenes, sonidos y videos bajo el control de guiones en los que se especifica la lógica de programación (Ford 2009). Nace a partir de la creencia de que muchas iniciativas de introducción de la programación a jóvenes han fracasado debido al uso de lenguajes muy difíciles de utilizar, proponiendo actividades fuera de los intereses de estos. Según sus creadores es necesario dejar de ver a la programación como una actividad





Figura 1.3: Entorno Scratch, con él se crean micromundos con gran facilidad

extremadamente técnica, apropiada solamente para un reducido grupo de personas (Mitchel Resnick & Maeda 2007).

Desde su lanzamiento en mayo de 2007 ha generado muchas expectativas, pues provee al usuario no especializado en informática de un medio de creación de software. Su origen estuvo en la idea de *lograr habilidades creativas y tecnológicas en los Computer Clubhouses* (Mitchel Resnick & Maeda 2007) sobre todo con los niños de entre 8 y 12 años. Este producto está bajo licencia de código abierto basado y desarrollado en Squeak, una versión moderna de Smalltalk. Su lema es *Imagina-Programa-Comparte* y está diseñado para estimular la creatividad de los adolescentes en los medios digitales, aún cuando su vocación profesional esté alejada de la creación de software (Mitchel Resnick & Kafai 2009).

Independiente a su enfoque informal, soporta conceptos de programación tales como: lógica condicional e iterativa, programación de eventos y el uso de variables. Ha logrado que niños de distantes lugares del mundo colaboren entre sí en la creación de disímiles historias y juegos.

### Programar usando bloques

En los lenguajes que usan el texto como medio de expresión se han de formular las sentencias siguiendo un conjunto de reglas. Fallar en el uso de estas precisas reglas invalida a los programas. Scratch utiliza un enfoque en el que, para construir los programas, se han de seleccionar y unir bloques como los

que se muestran en las figuras 1.4 y 1.5. Las reglas que rigen la unión de estos bloques impide que se cometan errores sintácticos.

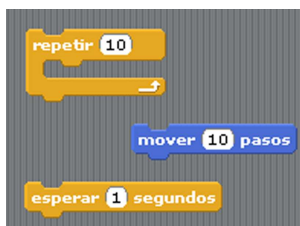


Figura 1.4: Distintos tipos de bloques disponibles en Scratch



Figura 1.5: Guión de Scratch con los bloques ensamblados

### Uso en la educación superior

Aunque su objetivo inicial han sido los niños, nada impide que se utilice en la enseñanza de programación a futuros profesionales del desarrollo de software. La Universidad de Harvard fue una de las primeras en hacerlo (David J. Malan 2007); además, desde el curso 2008-2009 se utiliza en la Universidad de Ciencias Informáticas (UCI) en un curso propedéutico de Algoritmización de apoyo a las asignaturas curriculares.

El principal inconveniente a la hora de usarlo a este nivel es que por su vocación simplificadora no posee soporte a varios de los conceptos de programación requeridos por profesionales del área. Además presenta, en su versión en español, incongruencias en el uso de varias palabras técnicas tal y como se ha manifestado en (Almaguer 2009a).

#### 1.4.5 Roles de los lenguajes de programación

El uso de los lenguajes en el aprendizaje de la programación puede verse en tres dimensiones fundamentales (Knudsen & Madsen 1990):

1. Medio para instruir a la computadora: viéndolo como un lenguaje de máquina de altísimo nivel, constituyendo una abstracción del medio de cómputo capaz de resolver muchos problemas. Con

la atención en aspectos de la ejecución de los programas, manejo de la memoria etc. (Nivel de implementación)

2. Gestión de la descripción del programa (fuentes): usándolo para entender el programa en su conjunto a partir de los recursos de administración del encapsulamiento, modularidad entre otros. (Nivel de especificación)
3. Modelado conceptual: usado para el modelado de conceptos del dominio, sus relaciones y propiedades. (Nivel conceptual)

Salvo raras excepciones (Bennedsen & Caspersen 2004), que usan el *nivel conceptual*, los cursos introductorios se basan en desarrollar las dos primeras. Para potenciar la primera es deseable que las construcciones sintácticas posean símbolos fáciles de aprender y que no colisionen con significados ya conocidos. Además, es recomendable que las estructuras de control tengan una clara traducción desde sus componentes sintácticas hasta la especificación de lo que describen como proceso (ver sección 1.4.1).

En la segunda dimensión son particularmente útiles las posibilidades declarativas de los lenguajes y en particular la posibilidad de expresar precondiciones, postcondiciones e invariantes. Se considera que mediante estas características se puede desarrollar la habilidad de leer código, que tan poca atención recibe en el diseño de los cursos iniciales.

### 1.4.6 Los ejercicios a utilizar

Dentro de las ciencias cognoscitivas se considera decisiva la participación de los estudiantes en el proceso de enseñanza-aprendizaje. Investigadores de la enseñanza de las Ciencias de la Computación consideran que la mejor manera de lograrlo es creando multimedios dado que: “los estudiantes de la generación del Nintendo preferirán aprender a manipular arreglos cuando los ejemplos consistan en producir sonido, en lugar de cuando haya que ordenar edades o hacer búsquedas sobre nombres de estudiantes” (Mark Guzdial 2002).

Se ha usado el simple “Hola mundo” durante los últimos años porque el texto fue el medio más fácil de usar con la tecnológica que se poseía. Hoy en día se pueden manipular sonidos, gráficos y video con la misma facilidad.

### Micromundo: medio para concretar la programación

Durante la ejecución de un programa la computadora realiza operaciones aritméticas sobre números y guarda los resultados en registros invisibles, esto hace que el proceso sea muy artificial y complejo para los estudiantes.

Los micromundos, inspirados por la Tortuga Logo, tienen por objetivo concretar la programación al mostrar a los estudiantes construcciones de programación que controlen el comportamiento de un actor que reside en un ambiente simple y controlado. En sí estos mundos son muy simples y el estudiante puede aprenderlos en muy poco tiempo, en general incluyen simulaciones que sirven como medio de saber cómo avanza el cómputo.

## 1.5 Currículo invertido

---

Siendo tan amplia y compleja la problemática de la enseñanza de la programación los esfuerzos no se limitan a los entornos, los métodos también se analizan y revolucionan sobre todo buscando una mayor motivación en el alumnado. Dentro de ellos existe uno que más que un método es un enfoque de la manera en que se organizan los cursos introductorios de programación.

### 1.5.1 El enfoque

El término *currículo invertido* apareció por primera vez en el campo de la Ingeniería Eléctrica (Cohen 1991), poco tiempo después fue visto como una posible manera de diseñar cursos de introducción a la Programación Orientada a Objetos. Entre otras virtudes se le vieron enormes potencialidades como método de enseñanza que permitiera ser fieles al paradigma durante la exposición de los conceptos (Meyer 1993), aunque fue ideado a inicios de los años 90 debió esperar hasta el 2002 para ser implementado (Meyer 2003).

El orden de los temas en un curso de programación tradicional es de *abajo hacia arriba* (bottom-up): comienza por los componentes básicos tales como las variables y la asignación, continúa con algunas estructuras de datos y de control yendo luego, si el tiempo lo permite, hacia el diseño modular y técnicas para estructurar programas grandes. En el caso de la implementación hecha por Meyer la hipótesis fundamental es que la manera más efectiva de aprender a construir software es usando software existente de excelente calidad (Meyer 2003).

Ventajas del enfoque:

- Acentúa principios importantes de la Ingeniería de Software, especialmente las técnicas de programación orientadas a objetos, permitiendo al mismo tiempo no descuidar la enseñanza de conceptos y destrezas de niveles más bajos.
- El uso de bibliotecas acostumbraría al estudiante a la idea de que no hay por qué saber cómo funciona todo, y además, que cuando se crean aplicaciones se reutiliza mucho software.

### Papel de la tecnología de objetos

La tecnología de objetos no es exclusiva respecto al enfoque tradicional (Meyer 2009), un programa OO es hecho con clases y su ejecución opera sobre objetos, pero las clases contienen rutinas, y los objetos contienen campos sobre los que los programas deben operar tal y como lo harían con las variables tradicionales. Por lo tanto ambos, la arquitectura estática de los programas y la estructura dinámica de los cómputos cubren los conceptos tradicionales.

#### 1.5.2 Requerimientos para aplicar el enfoque

El enfoque es muy atractivo, muchos aceptan su validez como evidente. Sin embargo, para aplicarlo se necesitan requerimientos que no todos los profesores están en condiciones de satisfacer. Los principales son enumerados a continuación.

**Ejemplos de gran calidad:** para desarrollar con calidad un curso basado en estos principios, es necesario tener programadas varias bibliotecas que soporten la realización de ejercicios de todos los temas que se vayan a tratar en el curso.

**Profesores que oculten lo necesario:** a los profesores que nunca han usado el enfoque en sus clases les resultará difícil ocultar los detalles de implementación.

**Manuales para su estudio:** las bibliotecas de código tienen un papel fundamental dentro del enfoque, sin embargo es necesaria la presencia de textos con los cuales tratar la teoría.

#### 1.5.3 Variante para todos los niveles: los objetos primero

Como *los objetos primero* (“objects first”) se entiende una variante del enfoque que se distingue básicamente en el tamaño de las aplicaciones usadas como ejemplos (Nienaltowski 2009), en el caso de *Currículo Invertido* los estudiantes están expuestos principalmente a una gran aplicación de decenas de miles de líneas de código las cuáles van explorando en la medida que avanza el curso. En esta variante, sin embargo, se tratan de usar algunos objetos (muchas veces solamente uno) para realizar los ejercicios en un contexto determinado y simple.

Este enfoque puede ser utilizado como soporte a cursos con diversos objetivos, ya sea para especialistas en Informática o para aficionados.

### 1.6 Conclusiones parciales

---

El enfoque de Currículo Invertido, al basarse en el consumo de objetos existentes, provee un marco de trabajo adecuado para la solución de problemas cuya representación sea rica en interfaz y permita

la simulación de ambientes conocidos por los estudiantes. Hasta el momento dicho enfoque se ha utilizado fundamentalmente para el desarrollo de cursos de Programación Orientada a Objetos, pero es prometedora su utilidad en cursos que tengan por principal objetivo el entendimiento de los conceptos básicos de la algoritmización.

Por otra parte, el estudio realizado ha mostrado, que para los estudiantes principiantes es importante la utilización no solo de lenguajes de programación especiales sino de entornos simples que permitan manipularlos. En particular se ha visto que los lenguajes gráficos basados en nodos son de una especial utilidad al liberar a los programadores de los errores sintácticos. Por último se ha podido notar que las construcciones que se usen en el lenguaje deben ser cercanas a las del idioma materno de los estudiantes y sobre todo no presentar significados que entren en conflicto con el mismo.



# 2

## Plataforma para la solución de problemas computacionales

En este capítulo se listan los conceptos que han sido considerados como los de mayor relevancia a la hora de entender los principios de la programación de computadoras. Luego se presenta la solución propuesta desde el punto de vista de sus diferentes usuarios: estudiantes y profesores en preparación de un curso. Para finalizar el capítulo se describe de manera breve el diseño y la implementación de la plataforma.

### 2.1 Diseño

---

#### 2.1.1 Finalidad y servicios que brinda

A partir de las características de la programación de computadoras enunciados en 1.4.1, de la naturaleza de los ejercicios descrita en 1.4.6, de los principios del enfoque anteriormente presentado (en particular en su versión *objetos primero*) así como de la importancia de usar un entorno de solución de problemas simple (y completo) se ha decidido construir una nueva herramienta que de forma integrada cumpla con estos requerimientos. Se considera que la manera en que se da respuesta a los mismos no ha sido explorada con anterioridad.



### 2.1.2 Finalidad del entorno desarrollado

Aún cuando la herramienta va destinada a diferentes grupos de usuarios, y por lo tanto para cada uno de ellos representa un tipo particular de medio de trabajo, todos rondan alrededor de un mismo objetivo:

- Proveer un medio sencillo e ilustrativo de exposición (a los profesores) y aprendizaje (a los estudiantes) de los siguientes conceptos:

objetos, métodos, datos, información, interfaces, tipos de datos, problema computacional, especificación de un problema computacional, instancia de problema computacional, algoritmo computacional, estado, entrada, salida, máquina computadora, instrucción, lenguaje de programación, programar computadoras, variable, asignación y declaración de variable.

### 2.1.3 Clientes de la plataforma

La herramienta incluye un producto acabado que puede ser usado propiamente y, sobre todo, un marco de trabajo<sup>1</sup> sobre el cual montar cursos de enseñanza de la programación. El principal grupo de usuarios serán los profesores de programación/algoritmización que quieran explorar nuevas ideas didácticas pero, por supuesto, los usuarios son los estudiantes que recibirán los cursos, y en función de ellos es que se han tomado las decisiones:

**Profesores:** diseño de cursos de programación y de algoritmización, en particular la concepción de ejercicios a partir de los micromundos disponibles.

**Estudiantes:** objeto de aprendizaje para la asimilación de los conceptos enunciados en 2.1.2.

**Profesores con conocimientos de programación en Java:** creación de micromundos sobre los cuales diseñar cursos introductorios.

### 2.1.4 Requerimientos que provee

Se ha construido una herramienta de trabajo que provee los requerimientos que necesitan los grupos de usuarios enunciados en 2.1.3, los principales son:

**Características básicas de un entorno de programación:** bloques gráficos y un espacio de trabajo para manipularlos.

---

<sup>1</sup>En idioma Inglés comúnmente llamado framework.

**Fácil de utilizar:** los componentes gráficos son pocos, sencillos y autodescriptivos.

**Trabajo en grupo:** permite que varios estudiantes trabajen a la vez sobre un mismo micromundo en ejecución.

**Personalizable:** está diseñado para acoplarle nuevos micromundos con facilidad.

La herramienta posee dos interfaces gráficas: la primera consiste en el entorno donde el estudiante podrá armar las soluciones; la otra, que será accedida solamente por la primera, reflejará el resultado de las acciones, siendo un ambiente ajustado al auditorio.

### 2.1.5 Entorno de construcción de las soluciones

El entorno constituye la aplicación que manipulará el estudiante, mediante ella realizará las actividades de edición y ejecución de los programas. Consta de dos partes fundamentales, la primera es el área donde se depositarán los bloques (etiquetada como *Programa*), en ella se pueden declarar variables y especificar las acciones que se quieren ejecutar. La segunda es la fábrica de bloques en la que se pueden encontrar cinco tipos de bloques diferentes; de ella se toman aquellos que sean necesarios para la construcción del programa (figura 2.1).

#### Acciones sobre el entorno

De entre los bloques a usar se destacan las acciones a aplicar al escenario en el que estén los objetos. Se escogerán las órdenes y preguntas a realizarle a los objetos en aras de obtener el resultado deseado (figura 2.2).

#### Estructuras para el control del flujo de ejecución

Se disponen de tres estructuras de control: condicional simple, condicional compuesta y una para la iteración (figura 2.3). Todas personalizables a través de configuraciones.

#### Tipos de datos

Siempre se dispondrá de tres tipos básicos: Número, Lógico y Cadena; además, en dependencia del micromundo que se utilice, estarán también sus tipos definidos (figura 2.4).

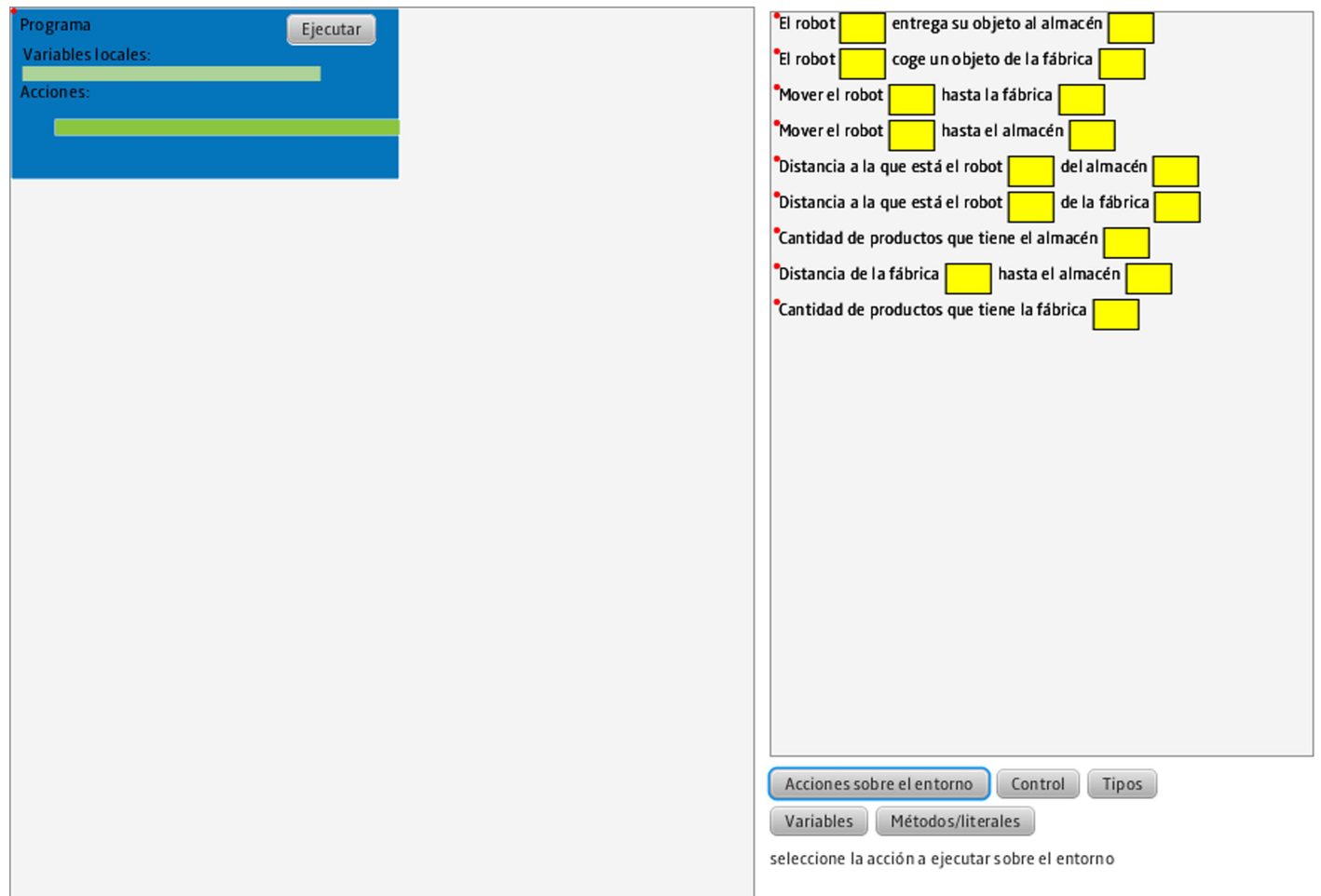


Figura 2.1: Vista inicial del entorno, mediante ella se accede a cada una de las funcionalidades

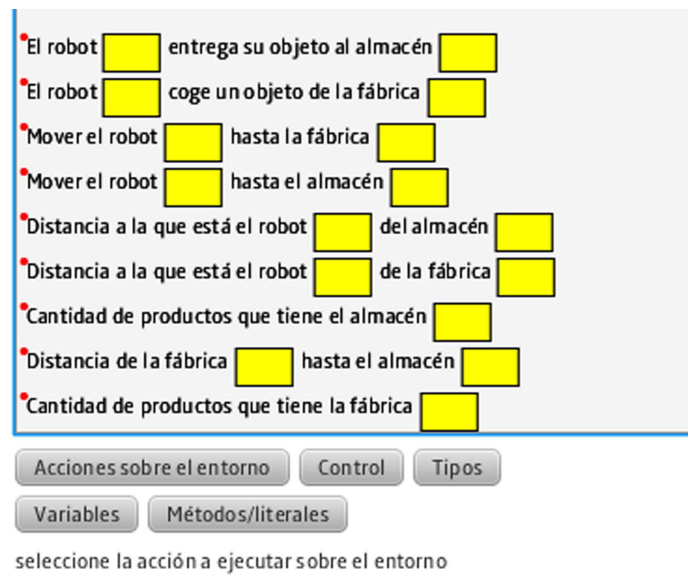


Figura 2.2: Conjunto de acciones con las que se podrá actuar sobre el entorno (micromundo) que se esté usando

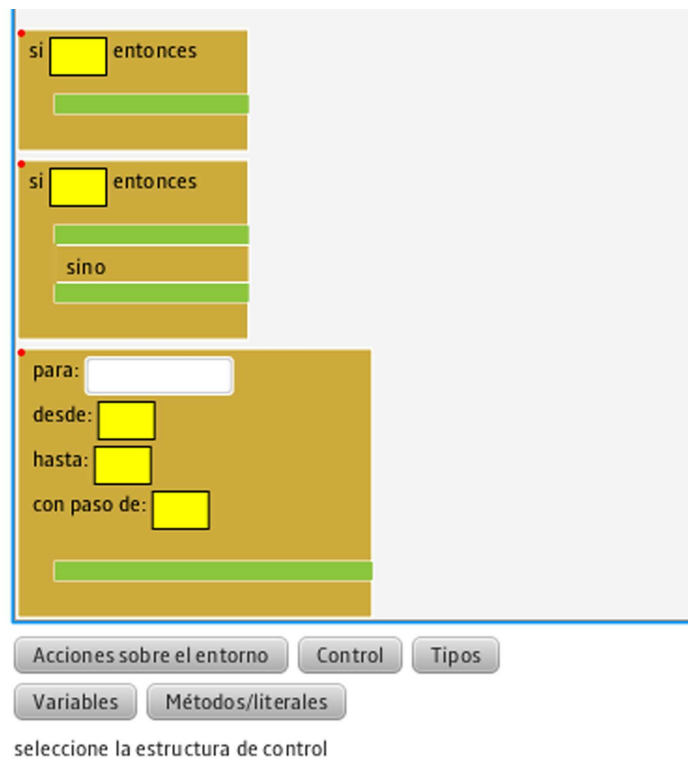


Figura 2.3: Vista de las estructuras de control de que se dispone



Figura 2.4: Conjunto de tipos de datos a utilizar



Figura 2.5: Vista con las variables que se pueden manipular en un momento dado

## Variables

Es aquí donde se encuentran las operaciones que se pueden hacer sobre las variables. En la primera posición se encuentra el bloque que permite declararlas, a continuación la operación de asignación y luego el conjunto de objetos de datos disponibles en ese momento. Para actualizar el área con nuevas variables declaradas se utilizará el botón *Actualizar* (figura 2.5).

## Métodos/Literales

En esta área se encuentran los bloques de los literales numéricos, de cadena y lógicos. Además, existe un botón por cada tipo registrado en el entorno, cuya función es mostrar el conjunto de operaciones que pueden aplicarse a objetos de dicho tipo (figura 2.6).

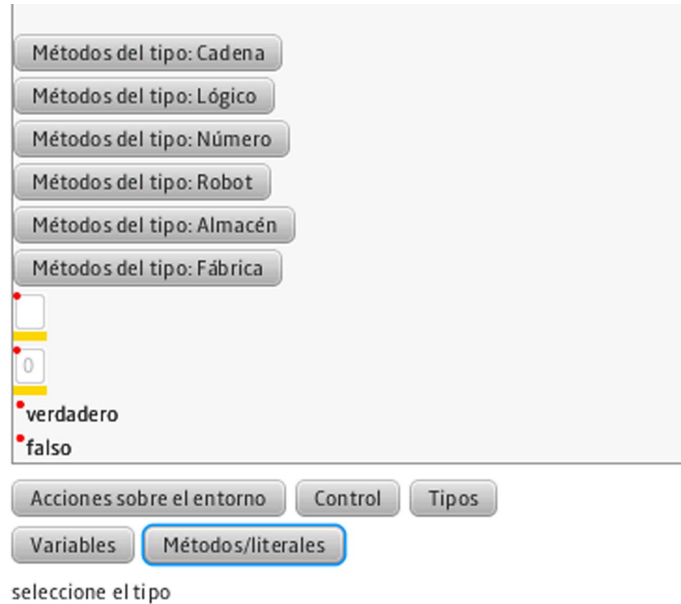


Figura 2.6: Vista para acceder a literales y a la interfaz de los tipos

### 2.1.6 Micromundos

Los programas creados manipulan objetos que pertenecen a micromundos externos al entorno de programación (en aplicaciones independientes). En el caso de la arquitectura diseñada, estas aplicaciones deberán estar en una computadora independiente especialmente dedicada a ello. Este diseño permite que varios estudiantes compartan el mismo micromundo y, por lo tanto, puedan trabajar en equipos ya sea para cooperar o para competir (figura 2.7).

## 2.2 Consideraciones sobre la implementación

---

### 2.2.1 Interfaz de programación para los micromundos

La herramienta podrá ser personalizada de dos formas: la más común consistirá en la creación de micromundos ajustados al contexto en que se vaya a desarrollar el curso (por ejemplo, de usarse en una ciudad minera como Moa no debiera faltar un entorno que simule la dinámica de extracción de minerales). Otra forma sería la extensión o modificación del sistema, actividades posibles dado que se dispone del código de la aplicación, solo que en este caso se necesitarían mayores conocimientos y requeriría más esfuerzo que la primera.

A continuación describimos el modo de proceder para desarrollar un micromundo. Por razones de claridad usaremos el mismo que será desarrollado en un próximo capítulo como caso de estudio.

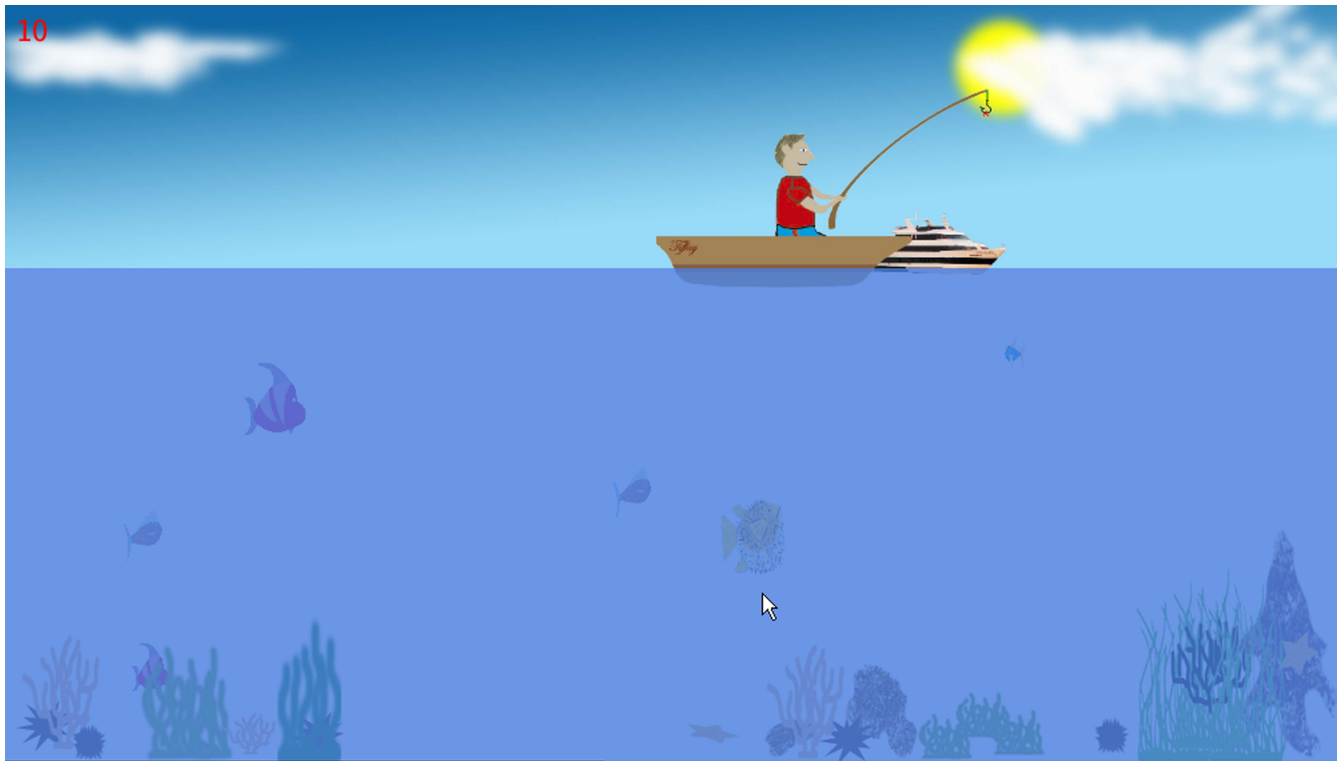


Figura 2.7: Entorno de ejemplo, micromundo que simula un pescador en el mar

### 2.2.2 Implementación del micromundo robot

Los micromundos han de ser simples, con un dominio de aplicación cercano a lo que conozcan los estudiantes y, sobre todo, con una alta riqueza visual (consideraciones en las secciones 1.4.6 y 1.5). Además estarán unidos al ambiente de programación proveyéndolo de los tipos, variables y acciones que se necesitan para resolver los problemas.

Un curso típico requiere de ejercicios con los cuales se pueda llevar a los estudiantes a muchos escenarios distintos; se necesitará lograr la creación de contextos que les reten la imaginación y los mantengan interesados. Uno de los objetivos de la solución que se plantea es que cada profesor interesado en aplicarla pueda crear sus propios ambientes con facilidad.

Para montar un ambiente se declararán e implementarán interfaces Java que representen los objetos que se quieran exportar al entorno. Durante dicha definición se expresará mediante anotaciones las características visuales deseadas para los bloques de las acciones (los métodos), así como otros detalles de la interfaz. En la tabla 2.1 se relacionan algunas de estas anotaciones y en 2.4 se muestra un fragmento del código de la interfaz *IRobot* con la que se ha modelado el objeto robot.

Anotación	Descripción
@etiqueta	Se aplica a los parámetros, consiste en el comentario que precede cada uno de ellos en los bloques (ver códigos 2.2 y 2.3).
@nombre	Se aplica a tipos, especifica el nombre con que se visualizará el tipo en la interfaz (ver código 2.1).
@objeto	Se aplica a los métodos, con él se comenta al objeto que recibe el mensaje (ver códigos 2.2 y 2.3).
@sufijo	Se aplica a los métodos, permitiendo ubicar un texto después del último parámetro (ver código 2.3).

Tabla 2.1: Semántica de las principales anotaciones usadas para configurar los micromundos

```
@nombre("Robot")
public interface IRobot { /* ... */ }
```

Código 2.1: Uso de la anotación @nombre

```
public interface IRobot {
    @objeto(" Mover el robot ")
    void moverHastaFabrica(
        @etiqueta(" hasta la fábrica ")
        IFabrica fabrica);
    //...
}
```

Código 2.2: Uso de las anotaciones @objeto y @etiqueta



```

public interface IAlmacen {
    @objeto(" Cantidad de productos que tiene el almacén ")
    @sufijo(" en estos momentos")
    Numero cantidadProductos();
    //...
}

```

Código 2.3: Uso de las anotaciones @objeto y @sufijo

```

@proxy @nombre("Robot")
public interface IRobot {
    @retornoAsincrono
    @objeto(" Mover el robot ")
    void moverHastaFabrica(
        @etiqueta(" hasta la fábrica ")
        IFabrica fabrica);
    @retornoAsincrono
    @objeto(" El robot ")
    void cogerObjeto(
        @etiqueta(" coge un objeto de la fábrica ")
        IFabrica fabrica);
    @retornoAsincrono
    @objeto(" El robot ")
    void entregarObjeto(
        @etiqueta(" entrega su objeto al almacén ")
        IAlmacen almacen);
    @retornoAsincrono
    @objeto(" Mover el robot ")
    void moverHastaAlmacen(
        @etiqueta(" hasta el almacén ")
        IAlmacen almacen);
    @objeto(" Distancia a la que está el robot ")
    Numero distanciaHastaFabrica(
        @etiqueta(" de la fábrica ")
        IFabrica fabrica);
    @objeto(" Distancia a la que está el robot ")
    Numero distanciaHastaAlmacen(
        @etiqueta(" del almacén ")
        IAlmacen almacen);
}

```

Código 2.4: Declaración de la interfaz IRobot

La implementación de las interfaces podrá hacerse en cualquiera de los lenguajes que hoy día permiten hacerlo (Java, JRuby, Scala, JavaFX Script, entre otros). En el caso de estudio mostrado, la implementación fue realizada en JavaFX Script, empleando solamente cuatro horas para el diseño y la programación. La apariencia visual puede mejorarse en muchos sentidos, sin embargo se ha decidido hacerla así con el ánimo de demostrar que la solución es totalmente viable aún cuando presupone la construcción de entornos por parte de los profesores.

### 2.2.3 Arquitectura

El sistema consta de una arquitectura cliente/servidor en la que un enrutador de mensajes provee el protocolo mediante el que los entornos de programación se comunican con el micromundo. La figura 2.8 muestra la arquitectura implementada.

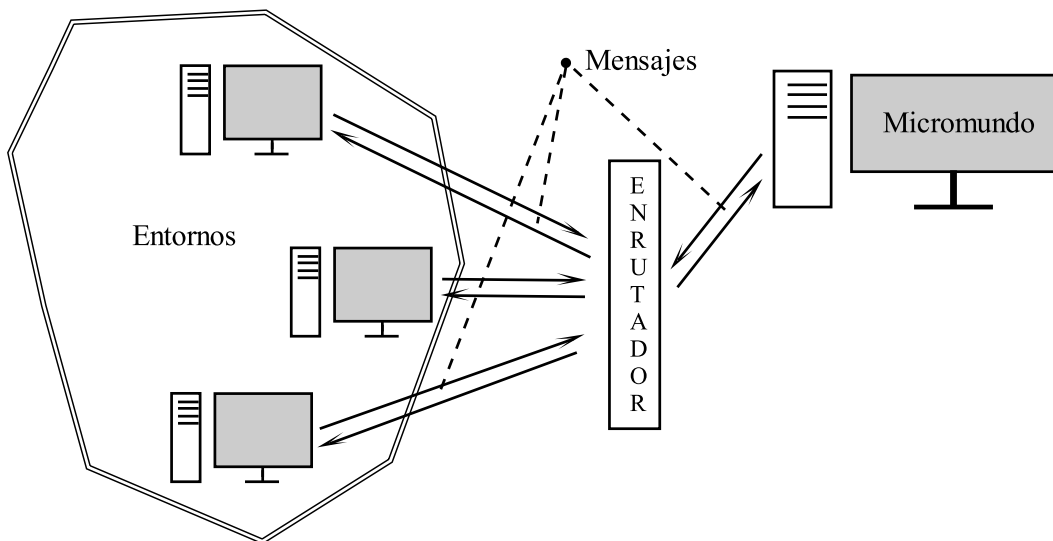


Figura 2.8: Arquitectura del sistema

Las partes constituyentes de la misma son:

**Entornos de programación:** clientes en los que trabajarán los estudiantes en la solución de problemas.

**Micromundo:** componente encargado de ejecutar los cambios de estado operados en los objetos, en cada uso del sistema estará activo un único micromundo.

**Enrutador:** medio mediante el que se establecerá la comunicación entre los otros dos componentes, transforma cada mensaje de los clientes en mensajes para el micromundo, manteniendo una misma cola para todos los clientes.

### 2.2.4 Tecnologías usadas en la implementación

En el desarrollo del marco de trabajo fueron utilizadas varias tecnologías, a continuación se nombran indicando el componente en que se utiliza y la razón:

- **Java:** descripción de las interfaces usadas en los micromundos y como plataforma al ser integradora de otros tres lenguajes (JavaFX, Scala y JRuby).
- **JavaFX:** desarrollo del micromundo y de la interfaz del entorno de programación.
- **Erlang:** implementación del enrutador (debido a su rendimiento y velocidad de respuesta).
- **Scala:** mecanismos necesarios para la comunicación entre los clientes, el servidor y el enrutador.
- **JRuby:** a este lenguaje de programación se traducen los bloques de código armados en el entorno, en caso de que hayan pasado el análisis semántico; fue tomado como intérprete del lenguaje desarrollado.

#### Java

El lenguaje de programación Java se encuentra, según el sitio TIOBE<sup>2</sup>, entre los más utilizados en el mundo. Entre sus características se destacan (Friesen 2007):

**Multiplataforma:** existen versiones de su máquina virtual (MV) para Linux, MacOS, UNIX, Windows, entre otros e incluye además versiones para dispositivos móviles.

**Soporte a varios modelos de programación:** es un lenguaje Orientado a Objetos, presenta primitivas para la sincronización de hilos de ejecución mediante monitores, se han desarrollado extensiones que soportan la programación orientada a aspectos (AspectJ) y dentro de la propia plataforma en su uso empresarial se le ha incluido este paradigma a través de anotaciones predefinidas. La presencia de anotaciones, de capacidades reflexivas y la propia arquitectura de la MV le han permitido una alta flexibilidad.

**Distribuido:** ha sido diseñado para el entorno distribuido de internet, presenta bibliotecas que permiten tratar URLs como simples ficheros y que hacen sencilla la comunicación entre aplicaciones residentes en diferentes computadoras.

Son muchos los sistemas que necesitan para su desarrollo el uso de diferentes lenguajes de programación, en particular el uso de lenguajes script facilita la integración de componentes construidos

---

<sup>2</sup><http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

con diferentes tecnologías. En la versión 6 del lenguaje se ha incluido soporte a la integración entre el lenguaje Java y muchos otros, tales como: BeanShell, Jython, JavaScript, Groovy y JRuby entre otros (*Scripting for the Java Platform* n.d., Bosanac 2008).

### JavaFX

JavaFX consiste en una familia de tecnologías cuyo fin es desarrollar aplicaciones de alta riqueza visual. Entre sus características se destacan el lenguaje de programación JavaFX Script y el *modo retenido* para manejar los gráficos.

La programación de interfaces posee características que complejizan su codificación en la mayoría de los lenguajes (Morris 2010), ejemplos:

**Presencia de estructuras de datos extensas y anidadas:** conjunto de objetos agrupados de forma jerárquica poseedores de numerosos atributos y comportamientos configurables.

**Extenso uso de la concurrencia:** las interfaces actuales aplican efectos de transición y de animación, lograr dichos comportamientos exige el diseño y la escritura de fragmentos de código extensos y complejos en la mayoría de los lenguajes actuales.

**Uso del patrón Modelo Vista Controlador:** sincronización de diferentes vistas dependientes de un modelo dado.

JavaFX Script es un lenguaje cuyo diseño responde, entre otras, a estas necesidades. Soporta conceptos de programación y un modelo de cómputo orientado a la implementación de interfaces de usuario. Sus características más destacadas son: los literales de objetos, la sentencia `bind` (para expresar la relación modelo/vista de manera declarativa) y el modelo de componentes visuales basados en nodos sobre los que se puede aplicar, entre otros, numerosos efectos de animación (James L. Weaver & Iverson 2009). Tanto la Programación Orientada a Objetos como la Funcional son soportadas por dicho lenguaje.

### Erlang

Erlang es un lenguaje de programación de propósito general que soporta los paradigmas Concurrente y Funcional (Armstrong 2007). Entre las empresas que lo usan se destacan Amazon, Yahoo, Facebook y Motorola; es ideal para el desarrollo de aplicaciones distribuidas y concurrentes, que necesiten la actualización de partes del código en ejecución y que deban siempre responder en un tiempo razonable sin importar la carga de cómputo que posea el sistema. Su modelo de cómputo concurrente se basa en el concepto de *actor*, y el éxito logrado por esta tecnología constituye la mejor demostración de cuán



Figura 2.9: Ejemplo de instrucción a ejecutar desde el entorno

adecuado es este modelo para el trabajo concurrente (Cesarini & Thompson 2009). Este lenguaje posee además facilidades para la integración con otros tales como C/C++, Java y C#; este hecho posibilita que sea usado como capa de comunicaciones en disímiles entornos.

## Scala

Scala es un lenguaje de programación multiparadigma cuyos programas se ejecutan en la misma máquina virtual de Java. Se pueden utilizar desde él las bibliotecas escritas en el lenguaje Java (y viceversa).

Una de sus ventajas reside en la facilidad con la que se pueden crear bibliotecas que al usarse funcionan como extensiones del propio lenguaje. Su flexibilidad permite que soporte la concurrencia según el mismo patrón y sintaxis que Erlang (Martin Odersky 2010).

## Ruby

El lenguaje de programación Ruby también permite el uso de más de un paradigma, es Orientado a Objetos, Funcional y posee grandes facilidades para la metaprogramación. Existen varios intérpretes de este lenguaje, entre ellos se encuentra uno escrito en Java (nombrado JRuby) que permite la interacción entre estas dos tecnologías (Charles O Nutter & Dees 2011).

### 2.2.5 Funcionamiento del sistema

Para mostrar las acciones y transformaciones de datos por las que pasa la ejecución de las instrucciones se tomará como ejemplo la ejecución de la instrucción mostrada en la figura 2.9. La primera acción que se realiza al presionarse el botón *Ejecutar* es la traducción de los bloques al código correspondiente en Ruby. Es en él que se garantiza el cumplimiento de la semántica de las estructuras de control constituyendo el intérprete del lenguaje desarrollado.

Los objetos que se manipulan en el entorno son *proxies* remotos<sup>3</sup> (Erich Gamma & Vlissides 1997) de los que se encuentran en el micromundo. Cada invocación a método corresponde a la lógica reflejada en el código 2.6. En esta capa se mantiene una línea de comunicación con el enrutador (reflejada en el código 2.7), mediante ella se realiza la comunicación con el micromundo.

<sup>3</sup>Representación local de objetos que se encuentran en otra aplicación.

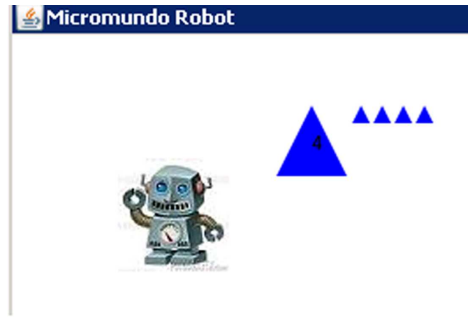


Figura 2.10: Estado inicial del micromundo

```
robot.moverHastaFabrica(fab2)
```

Código 2.5: Código Ruby obtenido tras la traducción de las instrucciones del entorno

```
def moverHastaFabrica(p0 : uci.IFabrica) {
  val p01 = getKey(p0)
  proxy.sendAndWait(
    target, 'mmoverHastaFabrica, new OtpErlangBinary(p01)
  )
}
```

Código 2.6: Fragmento del proxy del cliente que usa el entorno, escrito en Scala

```
{SName, mmoverHastaFabrica, SenderSName, Bin}
->
{iRobotServer, SName} ! {mmoverHastaFabrica, {SName, senderSName}, Bin},
loop(SName);
```

Código 2.7: Código del enrutador que lleva los mensajes del entorno hasta el micromundo

Los objetos del micromundo son controlados a través de otra capa de proxies que brinda servicios de la manera en que se muestra en el código 2.8. Luego de realizarse la ejecución del método se retorna la respuesta a través del mismo camino pero recorriéndolo a la inversa: enrutador (código 2.9), proxy cliente (código 2.10). Como resultado del proceso anteriormente descrito se logra transformar el micromundo desde el estado reflejado en la figura 2.10 hasta el estado mostrado en la figura 2.11.

```
case ('mmoverHastaFabrica, (senderSName: Symbol,
senderSNNName: String), b: OtpErlangBinary) =>
    val p1Key = b.getObject().asInstanceOf[Symbol]
    val p1 = _global(p1Key).asInstanceOf[uci.IFabrica]
    // otras instrucciones...
    objetos(senderSName) moverHastaFabrica p1
```

Código 2.8: Fragmento del proxy del servidor que usa el entorno, escrito en Scala

```
{mmoverHastaFabricaDone, {SenderSName, SenderSNNName}}
->
{iRobotClient, SenderSNNName} ! {SenderSName, mmoverHastaFabricaDone},
loop(ServiceNodeName);
```

Código 2.9: Código del enrutador que devuelve las respuestas de los mensajes

```
case (target: Symbol, 'mmoverHastaFabricaDone) =>
    llaves += ((target, 'mmoverHastaFabrica) -> (true, 0))
```

Código 2.10: Fragmento del proxy del cliente que recibe la respuesta de la ejecución

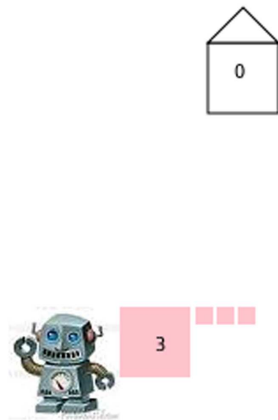


Figura 2.11: Estado del micromundo luego de la ejecución de la instrucción

### Consideraciones sobre la arquitectura y las tecnologías

Se decidió una arquitectura cliente-servidor con un mecanismo de integración buscando flexibilidad y generalidad, sus principales ventajas están dadas por:

- Posibilidad de trabajo de varios estudiantes sobre un mismo micromundo: se pueden utilizar ambientes de diferentes niveles de complejidad en los que los estudiantes puedan jugar y colaborar. Existen experiencias excelentes con este tipo de proyectos<sup>4</sup> (Achten 2008).
- Posibilidad de implementar los micromundos de manera rápida y de usar otras tecnologías para su desarrollo como puede ser Scratch (siguiendo las ideas planteadas en (Almaguer 2009a)).

## 2.3 Conclusiones parciales

---

A partir de las ideas enunciadas en la sección 1.4, en particular las referidas al lenguaje y los ejercicios a usar, se consideró que es conveniente la utilización de una arquitectura cliente-servidor. En el caso de la solución planteada se obtuvo como valor agregado el poder utilizar más de una tecnología para la implementación de los micromundos. Las características del lenguaje desarrollado están en correspondencia con lo concluido en la sección 1.6.

---

<sup>4</sup>Ver Robotcode: <http://robocode.sourceforge.net/?Open&ca=daw-prod-robocode>





# 3

## Aplicación de la plataforma a un caso de estudio

Como caso de estudio se utilizará un micromundo compuesto por tres fábricas, un almacén y un robot transportador de productos desde fábricas hasta almacenes (ver figura 3.1); las fábricas tienen 3, 4 y 5 artículos respectivamente. Se decidió la implementación de un caso de estudio de estas características debido al éxito logrado por experiencias en el uso de robots tales como Mindstorms y StartLogo TNG reportados en (Hancock 2003). Estos objetos se encontrarán en un área rectangular separados entre sí a una distancia cualquiera, las acciones que podrán realizarse aparecen descritas en la tabla 3.1.

Como aplicación de la propuesta se procede a mostrar la manera en que se puede enseñar algunos de los principales conceptos de la programación de computadoras. Las definiciones presentadas son producto de la investigación realizada o tomadas de los libros *Touch of Class* (Meyer 2009) o *Algorithmic Adventures, From Knowledge to Magic* (Hromkovic 2009). El orden escogido en las definiciones se basa en las dependencias existentes entre ellas, apareciendo primero las usadas por las demás.

### 3.1 Objetos

---

Los programas que se construyan siempre trabajarán con objetos que quedarán organizados como modelos de sistemas existentes en la realidad. Existirán dos grandes grupos: los que modelan objetos físicos (materiales) y los que constituyen colecciones de datos producto de la imaginación humana (de los que algunos reflejarán conceptos de la computación propiamente dicha).

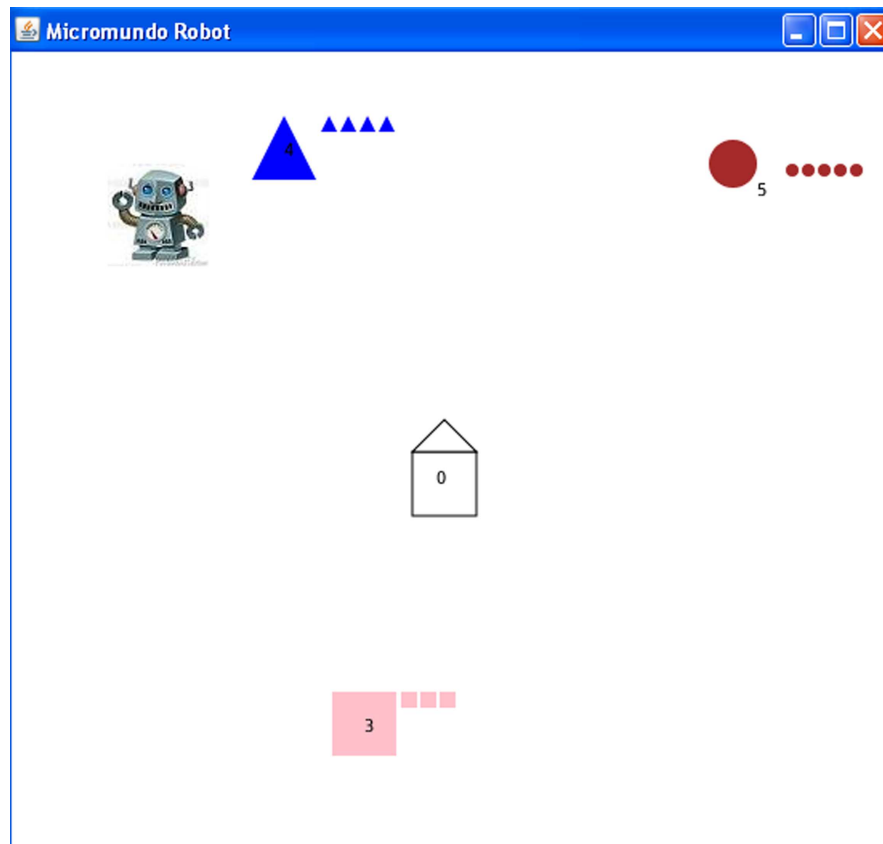


Figura 3.1: Entorno Micromundo Robot, escenario reducido en el que se desenvuelve un robot

Acción	Descripción
• Cantidad de productos que tiene la fábrica <input type="text"/>	Cantidad de productos en una fábrica.
• Cantidad de productos que tiene el almacén <input type="text"/>	Cantidad de productos en un almacén.
• Distancia de la fábrica <input type="text"/> hasta el almacén <input type="text"/>	Distancia entre una fábrica y un almacén.
• Distancia a la que está el robot <input type="text"/> del almacen <input type="text"/>	Distancia entre un robot y un almacén.
• Distancia a la que está el robot <input type="text"/> de la fábrica <input type="text"/>	Distancia entre un robot y una fábrica.
• Mover el robot <input type="text"/> hasta el almacen <input type="text"/>	Desplazar un robot hasta un almacén.
• Mover el robot <input type="text"/> hasta la fábrica <input type="text"/>	Desplazar un robot hasta una fábrica.
• El robot <input type="text"/> coge un objeto de la fábrica <input type="text"/>	Un robot toma un objeto de una fábrica.
• El robot <input type="text"/> entrega su objeto al almacen <input type="text"/>	Un robot entrega un objeto en un almacén.

Tabla 3.1: Acciones disponibles en el micromundo usado como caso de estudio



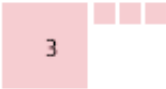


Objeto gráfico	Descripción
	Fábrica número 1.
	Fábrica número 2.
	Fábrica número 3.
	Objeto robot.
	Objeto almacén.

Tabla 3.2: Objetos disponibles en el caso de estudio

**Definición 3.1** *Un objeto es una máquina que posibilita a los programas acceder y modificar colecciones de datos.*

En este caso de estudio existen cinco objetos, todos identificables por las figuras que aparecen en la tabla 3.2. La manipulación de dichos objetos es a través de las variables que se encuentran accesibles en el entorno (figura 2.5).

### 3.2 Interfaces

Al realizar un intercambio entre dos objetos, cada uno de ellos debe respetar lo que está dispuesto a hacer o dar el otro. Las técnicas del diseño por contrato (Meyer 1997b) plantean que en la definición adecuada de interfaces puede estar el éxito de un diseño. En el enfoque de Currículo Invertido el consumo de interfaces es clave en el desarrollo de habilidades a la hora de realizar abstracciones.

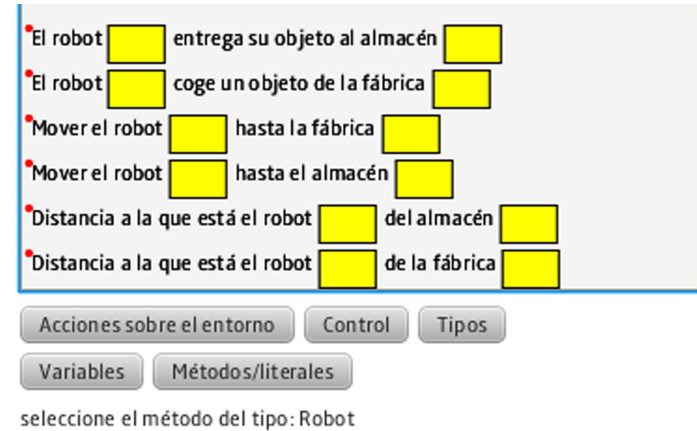


Figura 3.2: Interfaz del tipo Robot, a ella se llega por la opción Métodos/Literales + Robot

**Definición 3.2 (Interfaz)** *Un cliente de un mecanismo software (que sería por lo tanto un proveedor) es cualquier ente que lo use: una persona u otro software. Una **interfaz** es la **descripción de los medios** que permiten a los clientes usar lo brindado por el proveedor.*

### Informalmente

Una interfaz de una pieza de software es la descripción de cómo el resto del mundo podrá “hablarle”. Es importante notar que un elemento software dado puede ofrecer más de una y que se clasifican en dos grandes grupos:

**Interfaz de usuario:** para la cual el cliente es la persona que usa el sistema.

**Interfaz de programa:** para clientes que son a su vez elementos software.

Al primer grupo pertenecen los elementos gráficos con los que un cliente humano interactúa. En el segundo caso está el conjunto de acciones mediante las cuales se pueden manipular objetos de cada tipo, la figura 3.2 muestra la interfaz del tipo *Robot*.

## 3.3 Consultas y órdenes

Las interfaces estarán formadas por dos tipos de componentes que serán accesibles a través de la interfaz de un tipo en específico; los tipos de componentes son:

**Definición 3.3 (Consulta)** *Componente de la interfaz que permite pedirle un dato a un objeto.*

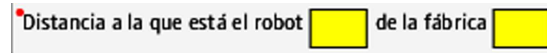


Figura 3.3: Consulta que permite conocer la distancia entre un robot y un almacén



Figura 3.4: Orden que permite desplazar un robot hasta el lugar donde esté un almacén

**Definición 3.4 (Orden)** *Componente de la interfaz que permite cambiar o transformar a un objeto.*

Las consultas permitirán encuestar a un objeto sobre su estado o alguna propiedad derivada de este. Sirva de ejemplo el bloque de la figura 3.3 con el que se puede saber una información a partir de las posiciones que ocupen un robot y un almacén. Por otra parte las órdenes son los medios mediante los cuales se altera el estado de un objeto, son las instrucciones con las que se puede guiar a la computadora hacia la solución de un problema. Un ejemplo de ellas es el bloque de la figura 3.4, esta orden logra que el robot se mueva hasta el lugar donde está el almacén.

### 3.4 Algoritmo

---

Uno de los conceptos centrales en todo curso de programación es el de algoritmo; el significado de sus características y las estrategias para crearlos serán actividades que continuamente llevarán a cabo los estudiantes.

**Definición 3.5 (Algoritmo)** *Conjunto de pasos caracterizado por:*

1. *Describir las actividades de manera que se puedan llevar a cabo incluso sin que se sepa cuál es el resultado buscado.*
2. *No existir diferentes interpretaciones de las instrucciones.*
3. *Obtener siempre el mismo resultado al obedecerlo, independientemente de quién lo ejecute y a la cantidad de veces que se haga.*

En el entorno aparecen descritas con claridad y precisión todas las actividades que pueden realizarse, la persona encargada de preparar el curso pondrá en manos del estudiante los significados de las interfaces en un lenguaje preciso. Por otra parte, se podrán sustituir a los micromundos por versiones en papel con las cuales los estudiantes hagan función de medios de cómputo. El uso de tablas para visualizar el estado de cada objeto y los cambios que se produzcan por la ejecución de los métodos puede ser la manera en que un profesor explique las propiedades de los objetos.

## 3.5 Problema computacional

---

Existe toda una rama de las Ciencias de la Computación que se dedica al estudio de los problemas con solución algorítmica. Si bien en un primer curso de programación no cabrían estos temas no es menos cierto que un estudiante necesita tener claridad respecto a la naturaleza de los problemas en computación.

Uno de los objetivos que tiene un primer curso de programación es que se aprenda a diferenciar los problemas computacionales de los que no lo son, para este fin se propone la siguiente definición:

**Definición 3.6 (Problema computacional)** *Problema cuya solución (o soluciones) pueda ser obtenida mediante el uso de un algoritmo (3.5).*

Ejemplo:

Trasladar todos los artículos de una de las fábricas hasta el almacén en el menor tiempo posible, sabiendo que el robot siempre se traslada a la misma velocidad.

## 3.6 Estado

---

Debido a las características gráficas del entorno es fácil hacer notar que la ejecución de órdenes siempre provoca cambios: se producen alteraciones de la posición, de la cantidad de artículos de los objetos, de los colores, etc. Basado en ello se pueden identificar características de los objetos que cambian en el tiempo: la posición, los colores, etc.

**Definición 3.7 (Estado)** *Situación de los objetos en un instante de tiempo dado.*

Algunos de los posibles estados del entorno en el actual caso de estudio son:

- Todas las fábricas poseen dos artículos, el almacén cinco, y el robot uno; el robot se encuentra al lado del almacén.
- El robot se encuentra junto a la fábrica # 3, el almacén está vacío y cada fábrica posee los artículos que ha producido.
- El almacén posee todos los artículos, las fábricas ninguno y el robot se encuentra junto a la fábrica # 1.



### 3.6.1 Entrada/salida

Los algoritmos establecen una relación entre una entrada y una salida. Estudios realizados (B. Haberman 2005) indican que este es un hecho poco entendido entre principiantes y por lo tanto merece un énfasis especial. A partir de una adecuada asimilación del concepto de estado basta enunciarlo como:

**Definición 3.8 (Entrada/Salida)** *Estados iniciales (entrada) y finales (salida) en la ejecución de un programa.*

En el epígrafe 3.7 aparece la especificación de un problema que incluye ejemplos de estos estados.

## 3.7 Especificación de un problema computacional

---

En el proceso de desarrollo de un sistema software se comienza siempre por una etapa de análisis, y en ella, se deberá identificar cuáles son las problemáticas a resolver. Cuando se comienza en la algoritmización es muy saludable alejar lo más posible a los estudiantes de las características de estas etapas, enfrentarlos a problemas de interpretación con las ambigüedades que estos conllevan puede confundirlos.

**Definición 3.9 (Especificación de un problema computacional)** *Descripción detallada de un problema computacional en la que se plasme de manera clara y sin ambigüedad cuál sería el estado inicial de los datos, cuál el estado final y de qué manera se relaciona el primero con el segundo.*

Es de mucha utilidad hacer notar la relación entre esta idea y el concepto de interfaz (3.2). Mediante su análisis se puede lograr un acercamiento a problemáticas del desarrollo profesional del software, ya sea desde el punto de vista de los métodos formales (Meyer 1991) o de cuestiones más “simples” como los procesos de validación de requisitos.

### Especificación del problema en el caso de estudio

Teniendo cuidado de no exagerar el nivel de formalización se puede lograr una especificación que cumpla con la definición.

#### Descripción del estado inicial:

- el almacén estará en una posición cualquiera y sin ningún artículo dentro.
- el robot estará en una posición cualquiera y no cargará con ningún objeto.
- la fábrica # 1 (*fab1*) tendrá 3 artículos, # 2 (*fab2*) tendrá 4 y la # 3 (*fab3*) 5 artículos, estarán también en cualquier posición.

**Descripción del estado final:**

- el almacén tendrá todos los objetos de una de las fábricas.
- una de las fábricas no tendrá artículos pues estos estarán en el almacén, las demás tendrán cualquier cantidad de artículos.
- el robot estará en una posición cualquiera, y podrá cargar algún objeto.

**Relación del estado inicial con el final:**

Traslado de todos los artículos de una de las fábricas hasta el almacén en el menor tiempo posible.

**Condiciones:**

- el robot siempre se traslada a la misma velocidad.
- podrán realizarse las órdenes y consultas listadas en la tabla 3.1.

**3.7.1 Instancia de un problema computacional**

Las instancias de un problema no son incluidas en la especificación, hacerlo convertiría a esta en un documento demasiado extenso. No obstante, es de mucha importancia saber cuáles son (y cuántas) para lograr la corrección de los algoritmos y entender con total precisión al propio problema.

**Definición 3.10 (Instancia de un problema computacional)** *Datos de entrada (que determinen el estado inicial) de un algoritmo que resuelve un problema computacional dado.*

Las instancias serán dadas por cada una de las combinaciones posibles de datos que se ajusten a los términos en que se describe la entrada (en la especificación), ejemplo: almacén en posición (2, 5), robot en posición (20, 15), fábrica # 1: posición (50, 50) , fábrica # 2: posición (70, 90), fábrica # 3: posición (30, 40).

**3.7.2 Solución al problema anteriormente especificado**

El problema en cuestión supone un análisis para la determinación de una solución óptima por lo que puede ser considerado como avanzado. Aunque sea un tanto compleja la problemática, tiene la ventaja de estar ajustada a la idea que ya poseen los estudiantes sobre la naturaleza de los problemas. A partir de un análisis no muy profundo se puede llegar a la conclusión que lo primero que hay que hacer es determinar la fábrica desde la que hay que mover los productos hasta el almacén, y luego, moverlos. Quedando la solución como sigue:

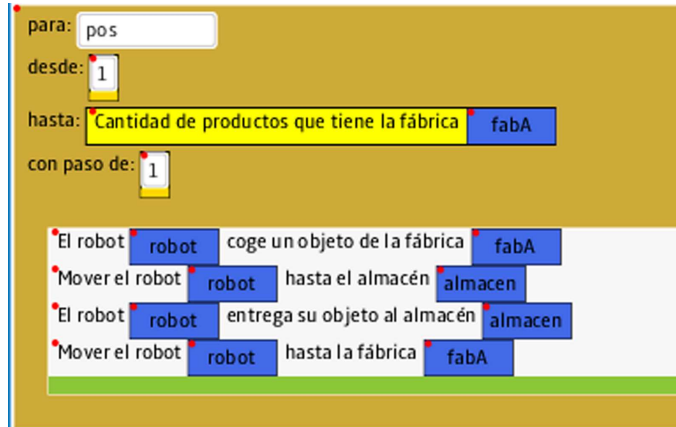


Figura 3.5: Solución al problema planteado

1. Determinación de la fábrica cuyos artículos se deben trasladar, sea ella la *fabA*.
2. Mover el robot hasta la fábrica *fabA*.
3. Repetir tantas veces como artículos haya en la fábrica *fabA*:
  - (a) Tomar artículo de la fábrica *fabA*.
  - (b) Mover el robot hasta el almacén.
  - (c) Entregar artículo en el almacén.
  - (d) Mover robot hasta fábrica *fabA*.

Quedando por solución la presentada en la figura 3.5.

### 3.8 Computadora

El hombre crea máquinas con el fin de resolver problemas que están más allá de sus posibilidades físicas o mentales, y por lo general ellas están destinadas a fines específicos: máquina de afeitar, de lavar, etc. La computadora es un tipo de máquina diferente que puede resolver una variedad de problemas en dependencia de lo que se le programe.

**Definición 3.11 (Máquina computadora)** *Máquina de propósito general que ejecuta programas y está conformada por:*

**dispositivos de entrada salida:** *medios por los que se le hacen llegar los programas y los de entrada que estos requieran.*

**memoria:** *espacio en el que residen los datos y los programas a ser procesados.*

**unidad de procesamiento:** *componente encargado de realizar los cálculos mediante los que se llevan a cabo los procesamientos.*

Este sería el único concepto relativo al hardware que se utilizará pudiéndose ver como la máquina que permite hacer toda la magia que es la computación. A la hora de concebir un micromundo para su exposición es válido notar que el ideal sería una Máquina de Turing.

## 3.9 Información y datos

---

Una palabra clave en la actualidad es **información**, para alguien de nuestro campo es saludable tener clara las diferencias entre información y datos. En general tratar estos temas puede servir para elevar la motivación pues en muchos círculos se habla de la *Era de la Información* como el futuro del desarrollo de la civilización.

**Definición 3.12 (Datos)** *Datos: colección de símbolos.*

**Definición 3.13 (Información)** *Información: interpretación humana de un dato.*

Para el enfoque escogido se puede particularizar a los datos como una parte pasiva de los objetos o como a los objetos en sí mismos. La información por su parte será el uso que se haga de los datos en el contexto de un algoritmo computacional dado. En la tabla 3.3 se pueden observar algunos ejemplos.

Datos	Información
2, 5	Coordenadas de la posición actual del robot.
1, 5	Cantidad de artículos que poseen el almacén y una fábrica respectivamente.

Tabla 3.3: Ejemplos de datos y de información, con la relación que poseen

### 3.9.1 Tipo de datos

Los tipos de datos que se utilicen permitirán el uso correcto de los objetos en un programa, mediante ellos el entorno sabrá dónde permitir la ubicación de los bloques consulta y de las variables. Con este caso de estudio se puede explicar que en la celda amarilla del bloque de la figura 3.6 solamente se pueden poner bloques que sean de tipo *Fábrica* (alguna de las variables que se muestran en la figura 3.7).

**Definición 3.14 (Tipo de datos)** *Nombre que se le da a una interfaz.*

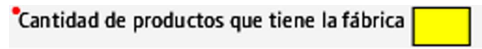


Figura 3.6: Bloque que consulta la cantidad de productos de una fábrica dada



Figura 3.7: Grupo de variables de tipo Fábrica

### 3.10 Lenguaje de programación

---

Los lenguajes de programación constituyen uno de los temas que más polémica provoca en los estudiantes, es importante saber manejar las expectativas que en ellos se crean para evitar la desmotivación. El rasgo más importante que se propone resaltar es verlo como medio de comunicación hombre-máquina y hombre-hombre.

**Definición 3.15 (Lenguaje de programación)** *Lenguaje mediante el cual se programan 3.20 (o instruyen 3.11) a las computadoras.*

En el caso del entorno, el lenguaje quedará definido por: las estructuras de control, los bloques para manipular variables y los métodos asociados a cada uno de los tipos de que se disponga. Además, formarían parte del lenguaje las reglas mediante las cuales se combinan los bloques.

### 3.11 Instrucciones de computadora

---

Cuando se programa se le da a la computadora una guía para que realice sus actividades, es decir se le **instruye** para que resuelva problemas.

**Definición 3.16 (Instrucción)** *Cada uno de los símbolos que componen a un lenguaje de programación.*

En nuestro caso serían instrucciones las estructuras de control, los bloques para manipular variables, las órdenes y las consultas asociadas a los tipos.

### 3.12 Programa

---

Este sería el concepto asociado a *software*, con él se relacionan muchos otros: algoritmo, instrucción, objeto, entrada/salida; por lo que es necesario tener especial cuidado.

**Definición 3.17 (Programa)** *Secuencia de instrucciones de computadora.*

### 3.12.1 Programa orientado a objetos

El enfoque desarrollado hace énfasis en el uso de objetos y, por lo tanto, es natural que los programas se vean en dicha dimensión.

**Definición 3.18 (Programa orientado a objetos)** *Programa conformado por la interacción de un grupo de objetos. Es decir, las instrucciones presentes hacen que un conjunto de objetos se relacionen entre sí.*

En esta propuesta se ve reflejado este principio al existir micromundos conformados por objetos que interactúan continuamente entre sí.

#### Aclaraciones sobre la definición de programa

- Un programa puede ser una secuencia de instrucciones sin sentido, no necesita ser la representación de un algoritmo.
- Un algoritmo no necesita ser escrito en forma de programa, puede ser descrito en lenguaje natural o en el lenguaje de las matemáticas.

## 3.13 Corrección de un programa

---

Algo que puede causar problemas, sobre todo a la hora de un examen, es el análisis de cuándo un programa está bien hecho, es decir ¿cuándo se considera correcto?

**Definición 3.19 (programa correcto)** *Programa obtenido al traducir un algoritmo que es solución de un problema.*

Esta definición se da en términos de las propiedades de un algoritmo con relación a un problema, por lo general no es objetivo de los cursos iniciales este tipo de análisis.

#### Aclaraciones sobre la definición de corrección

- Pueden existir estados para los que el programa realice acciones sin sentido, siempre que estén fuera de lo que se declaró en el texto de la especificación (3.9) del problema en cuestión.
- Salvo situaciones en las que un cliente lo solicite explícitamente: no es necesario demostrar la corrección de un programa.

### 3.14 Programar una computadora

---

Este es el nombre de la actividad para la que se estarían preparando los estudiantes por lo que posee una alta importancia. En función de los conceptos esbozados hasta ahora la definición queda muy sencilla:

**Definición 3.20 (Programar una computadora)** *Reescribir un algoritmo determinado en forma de programa (3.17).*

### 3.15 Variable

---

El concepto de variable es de gran importancia en la programación imperativa, ella refleja los cambios de estado que se van dando según avanza el cómputo. Definirla se encuentra principalmente con el inconveniente de que no es la primera vez que un estudiante se encuentra con ella, en Matemáticas se usan desde el inicio de la enseñanza secundaria con una connotación declarativa que poco tiene que ver con la programación<sup>1</sup>. Ejemplos de variables son las mostradas en la figura 2.5.

Se verá la palabra variable con un significado particular y dependiente al modelo de computación empleado:

**Definición 3.21 (Variable)** *Porción de memoria utilizada para almacenar en cada instante de tiempo un valor determinado, ya sea como medio de identificación de un objeto o como dato.*

#### 3.15.1 Instrucciones para manipular variables

Asociado al trabajo con variables se tendrán dos instrucciones:

**Definición 3.22 (Declaración de variable)** *Instrucción de computadora mediante la cual se da nombre y tipo (se define) a una variable.*

**Definición 3.23 (Asignación)** *Instrucción mediante la cual se reemplaza el contenido de una variable con un nuevo valor.*

Las declaraciones se harán mediante el primer bloque que aparece en la figura 2.5. Por su parte las asignaciones se llevarán a cabo con el segundo bloque de la misma figura.

---

<sup>1</sup>Se deja fuera con toda intención el concepto de variable de la programación declarativa.

## 3.16 Conclusiones parciales

---

En este capítulo se han enumerado algunos de los principales conceptos de la programación. En dependencia de las características del curso que se vaya a desarrollar se pueden modificar, eliminar o añadir algunos de ellos. A partir de lo expuesto se puede concluir que:

1. El entorno posee un lenguaje gráfico con el que se pueden expresar las operaciones básicas de la programación.
2. El micromundo escogido fue implementado en pocas horas, lo cual indica la posibilidad de hacer personalizaciones con facilidad.
3. Los conceptos escogidos fueron explicados de forma coherente y autocontenida.





# Conclusiones y recomendaciones

## Conclusiones

---

La herramienta que se propone cumple con los principios encontrados durante la investigación. Los resultados obtenidos fueron:

1. Marco de trabajo para el montaje de ejercicios que, bajo un enfoque de Currículo Invertido, permita el desarrollo de cursos introductorios de programación.
2. Lenguaje de programación gráfico basado en bloques que permite personalizar las estructuras de control.
3. Entorno de trabajo para la utilización del lenguaje.
4. Arquitectura distribuida para la interacción entre los micromundos y el entorno de programación, que permite el trabajo colaborativo entre estudiantes.
5. Utilización del marco de trabajo para el montaje de un caso de estudio.
6. Exposición de los principales conceptos asociados a la solución de problemas mediante computadoras.

## Recomendaciones

---

Son muchas las áreas y maneras en que se puede extender este trabajo. Los resultados obtenidos sirven como punto de partida a la creación de medios de aprendizaje y a la evaluación de diferentes estrategias de enseñanza, en particular se propone:

- Incluir un *debugger* en el entorno.
- Añadir soporte para la definición de los conceptos de método y clase.

### **CAPÍTULO 3. APLICACIÓN DE LA PLATAFORMA A UN CASO DE ESTUDIO**

- Montar varios casos de estudio.
- Escribir un manual que apoye la realización de un curso de programación utilizando la herramienta.
- Hacer una validación de la propuesta con estudiantes de primer año de una carrera de Informática.
- Utilizar la herramienta en un círculo de interés de la enseñanza secundaria.

# Bibliografía

- A. Begel, E. K. (2005), ‘StarLogo TNG: An Introduction to Game Development’, *Journal of E-Learning* .
- Achten, P. (2008), ‘Teaching Functional Programming with Soccer-Fun’, *Model Based System Development, Radboud University Nijmegen, The Netherlands* .
- Almaguer, J. A. C. (2009a), ‘Ambiente virtual y Lenguaje de Domino Específico para la enseñanza de la programación’, *MATECOMPU* .
- Almaguer, J. A. C. (2009b), ‘Pseudocódigo ejecutable para la enseñanza de la algoritmización’, *FIMAT XXI* .
- Andrew J. Ko, B. A. M. & Aung, H. H. (n.d.), ‘Six Learning Barriers in End-User Programming Systems’.
- Armstrong, J. (2007), *Programming Erlang Software for a Concurrent World*, The Pragmatic Bookshelf.
- B. Haberman, H. A. . D. G. (2005), ‘Is it really an algorithm? The need for explicit discourse.’, *Proceedings of the ITiCSE 2005 Conference, Lisbon, Portugal* .
- Bayman, P. & Mayer, R. E. (1983), ‘A Diagnosis of Beginning Programmers’ Misconceptions of BASIC Programming Statements’, *Commun. ACM* **26**(9), 677–679.
- Bennedsen, J. & Caspersen, M. (2004), ‘Programming in Context, A Model-First Approach to CS1’, *Proceedings of the thirty-fifth SIGCSE Technical Symposium on Computer Science Education* .
- Blackwell, A. F. (2002), First Steps in Programming: A Rationale for Attention Investment Models, *in* ‘Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments’, pp. 2–10.  
URL: <http://www.cl.cam.ac.uk/users/afb21/publications/HCC02a.pdf>
- Bogost, I. (2005), ‘Procedural Literacy: Problem Solving with Programming, Systems, & Play’, *T E L E M E D I U M* **52**(1 y 2).
- Bogost, I. (2007), *Persuasive Games, The Expressive Power of Videogames*, The MIT Press.
- Bonar, J. & Soloway, E. (1989), ‘Preprogramming knowledge: A major source of misconceptions in novice programmers’, *Studying the Novice Programmer. Hillsdale, NJ: Lawrence Erlbaum Associates* .
- Bosanac, D. (2008), *Scripting in Java Languages, Frameworks, and Patterns*, Pearson Education, Inc.

## BIBLIOGRAFÍA

- Boulay, B. D. (1989), 'Some difficulties of learning to program', *Studying the Novice Programmer*. Hillsdale, NJ: Lawrence Erlbaum Associates .
- Caitlin Kelleher, R. P. (2005), 'Lowering the Barriers to Programming: a survey of programming environments and languages for novice programmers', *Journal ACM Computing Surveys (CSUR)* **37**(2).  
URL: [www.cs.cmu.edu/~caitlin/papers/NoviceProgSurvey.pdf](http://www.cs.cmu.edu/~caitlin/papers/NoviceProgSurvey.pdf)
- Caspersen, M. & Bennedsen, J. (2007), 'Instructional Design of a Programming Course: A Learning Theoretic Approach', *Proceedings of the 3rd International Computing Education Research Workshop, ICER 2007, Atlanta, Georgia, USA* pp. 111–122.
- Caspersen, M. & Kölling, M. (2006), 'A Novice's Process of Object-Oriented Programming', *Companion to the 21st ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications* pp. 892–900.
- Caspersen, M.E., B. J. & Larsen, K. (2007), 'Mental Models and Programming Aptitude', *Proceedings of the 12th Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE, Dundee, Scotland* .
- Castrillón, F. R. (2009), 'Lexico en programación orientada a objetos'.  
URL: [http://riosur.net/Lexico\\_en\\_programacion\\_orientada\\_a\\_objetos.php](http://riosur.net/Lexico_en_programacion_orientada_a_objetos.php)
- Cesarini, F. & Thompson, S. (2009), *Erlang Programming*, O'Reilly Media, Inc.
- Charles O Nutter, Nick Sieger, T. E. O. B. & Dees, I. (2011), *Using JRuby Bringing Ruby to Java*, The Pragmatic Programmers LLC.
- Clancy, M. (2005), *Computer Science Education Research*, Taylor & Francis e-Library, capítulo 1, pp. 86–100.
- Cohen, B. (1991), The Inverted Curriculum, Technical report, National Economic Development Council, London.
- Composer, Q. (2010), 'Quartz Composer'.  
URL: [http://developer.apple.com/library/mac/#documentation/GraphicsImaging/Conceptual/QuartzComposer/qc\\_intro/qc\\_intro.html](http://developer.apple.com/library/mac/#documentation/GraphicsImaging/Conceptual/QuartzComposer/qc_intro/qc_intro.html)
- David Barnes, M. K. (2002), *Objects First with Java, A Practical Introduction using BlueJ*, Pearson Education / Prentice Hall.
- David J. Malan, H. H. L. (2007), 'Scratch for Budding Computer Scientists'.  
URL: <http://www.eecs.harvard.edu/%7Emalan/publications/fp079-malan.pdf>
- Dijkstra, E. (n.d.), 'A Short Introduction to the Art of Programming'.  
URL: <http://www.cs.utexas.edu/users/EWD/transcriptions/EWD03xx/EWD316.html>
- diSessa, A. (2001), *Changing Minds: Computers, Learning, and Literacy*, The MIT Press.
- Erich Gamma, Richard Helm, R. J. & Vlissides, J. (1997), *Design Patterns, Elements of Reuseable Object-Oriented Software*.

- Evans, D. (2010a), *Introduction to Computing: Explorations in Language, Logic, and Machines*, University of Virginia, p. 24.
- Evans, D. (2010b), 'Introduction to Computing: Explorations in Language, Logic, and Machines', p. 215.
- Evans, D. (2010c), 'Introduction to Computing: Explorations in Language, Logic, and Machines', p. 61.
- Evans, D. (2010d), 'Introduction to Computing: Explorations in Language, Logic, and Machines', p. 10.
- Fischer, G. (2005), 'Computational Literacy and Fluency: Being Independent of High-Tech Scribes', *University of Colorado, Boulder*.
- Ford, J. L. (2009), *Scratch Programming for Teens*, Course Technology PTR.
- Forte, A. & Guzdial, M. (2005), 'Motivation and nonmajors in computer science: Identifying discrete audiences for introductory courses', *IEEE Transactions on Education* pp. 248–253.
- Friesen, J. (2007), *Beginning Java SE 6 Platform From Novice to Professional*, Apress.
- Guzdial, M. (2005), *Computer Science Education Research*, Taylor & Francis e-Library, capítulo 3, pp. 127–154.
- Guzdial, M. (2008), 'Education Paving the way for computational thinking', *Communication of ACM* **51**(8), 25–27.
- Halasz, F. & Moran, T. P. (1982), 'Analogy considered harmful', *Proceedings of Human Factors in Computer Systems* pp. 383–386.
- Hancock, C. M. (2003), *Real-Time Programming and the Big Ideas of Computational Literacy*, PhD thesis, Massachusetts Institute of Technology.
- Harvey, B. (1997), *Computer Science Logo Style Vol. 3 Beyond Programming*, The MIT Press, capítulo 6, p. 294.
- Hromkovic, J., ed. (2010), *Teaching Fundamental Concepts of Informatics*, number 2009941784, LNCS, Springer Berlin Heidelberg NewYork.
- Hromkovic, P. D. J. (2009), *Algorithmic Adventures From Knowledge to Magic*, Springer-Verlag Berlin Heidelberg.
- James L. Weaver, Weiqi Gao, P. S. C. & Iverson, D. (2009), *Pro JavaFX Platform Script, Desktop and Mobile RIA with Java Technology*, Apress.
- James Wilson, M. F. . N. H. (1993), 'Technology and problem solving'.  
URL: <http://jwilson.coe.uga.edu/emt725/Pssyn/Pssyn.html>
- Jean Griffin, Mark Fickett, R. P. (2002), 'Objects-First, Algorithms-Early with BotWorld'.  
URL: <http://www.seas.upenn.edu/~botworld/files/botworld-012009.pdf>
- Jens Bennedsen, M. E. C. (n.d.), 'Teaching Object-Oriented Programming, Towards Teaching a Systematic Programming Process'.

## BIBLIOGRAFÍA

- Jens Bennedsen, M. E. C. & Kölling, M., eds (2008), *Reflections on the Teaching of Programming, Methods and Implementations*, Vol. 4821 of *LNCS*, Springer-Verlag, Transitioning to OOP/Java, A Never Ending Story, pp. 80–97.
- Jorma Sajaniemi, C. H. (n.d.), ‘Teaching Programming, beyond Objects First’.  
URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.107.3914>
- Jungwoo Ryoo, F. F. & Janzen, D. S. (2008), ‘Teaching Object-Oriented Software Engineering through Problem-Based Learning in the Context of Game Design’, *21st Conference on Software Engineering Education and Training*.
- Kessler, M. (2004), Exercise Design for Introductory Programming Learn-by-doing basic O-O-concepts using Inverted Curriculum, Master’s thesis, ETH Zurich.
- Kim H. C., Lee W. G., H. H. S. S. E. M. & H., K. S. (2008), Application strategies of Educational Programming Languages for New K-12 ‘Information’ subject curriculum, Technical Report KRF-2007-721-B00082, KRF(Korea Research Foundation Grant).
- Kölling, M. (2010), *Introduction to Programming with Greenfoot*, Pearson Higher Education, Prentice Hall.
- Knudsen, J. & Madsen, O. (1990), ‘Teaching Object-Oriented Programming is more than Teaching Object-Oriented Programming Languages’, *DAIMI-PB 251*.  
URL: <http://www.ifs.uni-linz.ac.at/~ecoop/cd/papers/0322/03220021.pdf>
- Kolling, M. (1999a), ‘The problem of teaching object-oriented programming Part I: Languages’, *Journal of Object-Oriented Programming*.
- Kolling, M. (1999b), ‘The problem of teaching object-oriented programming Part II: Environments’, *Journal of Object-Oriented Programming*.
- M., S. M. M. (2006), ‘Evaluación de las herramientas de software Lexico y CSharp como alternativas en la enseñanza y el aprendizaje de la Programación Orientada a Objetos’.  
URL: [http://riosur.net/Resumen\\_de\\_investigacion\\_Copacabana.php](http://riosur.net/Resumen_de_investigacion_Copacabana.php)
- Mark Guzdial, E. S. (2002), ‘Teaching the Nintendo Generation to Program’.  
URL: [http://coweb.cc.gatech.edu/guzdial/uploads/17/CACM\\_Nintendo\\_Generation.pdf](http://coweb.cc.gatech.edu/guzdial/uploads/17/CACM_Nintendo_Generation.pdf)
- Martin, D. P. . F. (1986), ‘Fragile knowledge and neglected strategies in novice programmers.’, *Empirical studies of programmers*. Norwood, NJ: Ablex. .
- Martin Odersky, Lex Spoon, B. V. (2010), *Programming in Scala, Second Edition*, Artima Press.
- Mateas, M. (2005), Procedural literacy: Educating the new media practitioner, in ‘in On the Horizon: Special Issue on Future Strategies for Simulations, Games and Interactive Media in Educational and Learning Contexts, forthcoming’, p. 2005.
- McEuen, S. F. (2001), ‘How Fluent with Information Technology Are Our Students?’.  
URL: <http://net.educause.edu/ir/library/pdf/EQM0140.pdf>

- Meyer, B. (1991), *Advances in Object-Oriented Software Engineering*, Prentice Hall, Englewood Cliffs, capítulo 1, pp. 1–50.
- Meyer, B. (1993), ‘Towards an O-O Curriculum’, *Technology of Object-Oriented Languages and Systems* pp. 585–594.
- Meyer, B. (1997a), *Object-Oriented Software Construction*, 2nd edn, Prentice Hall PTR, capítulo 9, pp. 938–940.
- Meyer, B. (1997b), *Object-Oriented Software Construction*, 2nd edn, Prentice Hall PTR.
- Meyer, B. (2003), The Outside-In Method of Teaching Introductory Programming, in M. Broy & A. Zamulin, eds, ‘Proceedings of fifth Andrei Ershov Memorial Conference, Akademgorodok, Novosibirsk’, Vol. Lecture Notes in Computer Science, Springer-Verlag, pp. 66–78.
- Meyer, B. (2009), *TOUCH OF CLASS: Learning to program well with objects and contracts*, Springer-Verlag.
- Michael Haungs, J. C. & Janzen, D. (2008), ‘IMPROVING ENGINEERING EDUCATION THROUGH CREATIVITY, COLLABORATION, AND CONTEXT IN A FIRST YEAR COURSE’, *American Society for Engineering Education*.
- Mitchel Resnick, John Maloney, A. M. H. N. R. E. E. K. B. A. M. E. R. J. S. B. S. & Kafai, Y. (2009), ‘Scratch: Programming for Everyone’, *Communications of the ACM*.  
URL: <http://web.media.mit.edu/~mres/scratch/scratch-cacm.pdf>
- Mitchel Resnick, Y. K. & Maeda, J. (2007), ‘A Networked, Media-Rich Programming Environment to Enhance Technological Fluency at After-School Centers in Economically-Disadvantaged Communities’.  
URL: <http://web.media.mit.edu/%7Emres/papers/scratch-proposal.pdf>
- Morris, S. (2010), *JavaFX in Action*, Manning Publications Co., capítulo 1, p. 25.
- Nienaltowski, M.-H. (2009), Automated tutoring for novices in object-oriented programming, PhD thesis, University of London.  
URL: <http://www.dcs.bbk.ac.uk/research/recentphds/mhelene.pdf>
- Niko Myller, Andrés Moreno, R. B. E. S. (2006), ‘Jeliot 3’.
- Pea, R. (1986), ‘Language-independent conceptual bugs in novice programming.’, *Journal of Educational Computing Research* **2**(1), 25–36.
- Pedroni, M. & Meyer, B. (2006), ‘The Inverted Curriculum in Practice’, *Proceedings of SIGCSE, ACM, Houston, Texas*.
- Polya, G. (1957), *How to Solve It, A New Aspect of Mathematical Method*, Princeton Univeristy Press.
- Priestley, M. (2011), *A Science of Operations*, Springer-Verlag.
- R Pea, E. S. . J. S. (1987), ‘The buggy path to the development of programming expertise.’, *Focus on Learning Problems in Mathematics* **9**(1).



## BIBLIOGRAFÍA

- R. Putnam, D. Sleeman, J. A. B. L. K. K. (1986), ‘A Summary of Misconceptions of High School Basic Programmers’, *Educational Computing Research* **2**, 459–472.
- Roque, R. V. (2007), OpenBlocks, An Extendable Framework for Graphical Programming Systems, Master’s thesis, MIT.
- Rosana Satorre, Patricia Compañ, F. L. (2004), ‘Un modo de entender la programación’, *X Jornadas de Enseñanza Universaria de la Informática, Jenui 2004* .
- Scratch Reference Guide* (2009).
- Scripting for the Java Platform* (n.d.).  
URL: <http://jcp.org/en/jsr/detail?id=223>
- SooHwan Kim, S. H. & Kim, H. (2009), ‘How Can We Teach Computational Literacy to All Levels of Students?’, *Fifth International Joint Conference on INC, IMS and IDC* .
- Spohrer, J. & Soloway, E. (1986), ‘Analyzing the high frequency bugs in novice programs.’, *Empirical Studies of Programmers. Norwood, NJ: Ablex* .
- Stagecast (2010), ‘Stagecast’.  
URL: <http://www.stagecast.com/>
- Stephenson, C., Gal-ezer, J., Phillips, M. & Verno, A. (2008), ‘Task Force Chair’.  
URL: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.170.2776>
- Thomas H. Cormen, Charles E. Leiserson, R. L. R. & Stein, C. (2009), *Introduction to Algorithms*, third edn, The MIT Press.
- Wing, J. M. (2008), ‘Computational Thinking and Thinking About Computing’.  
URL: <http://www.cs.cmu.edu/afs/cs/usr/wing/www/ct-and-tc-long.pdf>
- Wirth, N. (2002), ‘Computing Science Education: The Road not Taken’, opening address at ITiCSE conference, Aarhus, Denmark.  
URL: <http://www.inr.ac.ru/~info21/texts/2002-06-Aarhus/en.htm>



## Uso del marco de trabajo

Los profesores que deseen usar el marco de trabajo para montar sus propios micromundos deberán diseñar un conjunto de interfaces Java, luego las decorarán con las anotaciones que aparecen descritas en las tablas 2.1 y A.1.

El siguiente paso sería usar la clase *Generator* para generar los proxies necesarios para establecer la comunicación entre el entorno y el micromundo, en la tabla A.2 se encuentra descrita la interfaz de dicha clase. Una vez generadas las clases se procederá a implementar las interfaces para que quede desarrollado el micromundo.

Anotación	Descripción
@ignorar	Indica los métodos que no necesitan ser tenidos en cuenta a la hora de construir los proxies.
@implementacion	Utilizada con los tipos primitivos, indica la clase que los implementa.
@objeto	Utilizada para etiquetar el lugar donde se pondrá el objeto que recibe el mensaje.
@orden	Alterar el orden de los parámetros en el bloque respecto a cómo están en el código mediante una secuencia de enteros.
@proxy	Marca las clases a procesar.
@retornoAsincrono	Indica cuándo un método retornará segundos después de ser ejecutado, utilizado fundamentalmente con métodos que implican animaciones.

Tabla A.1: Semántica del resto de las anotaciones usadas para configurar los micromundos

Método	Descripción
generateClientProxies	Genera en el directorio especificado las clases usadas por el cliente.
generateServerProxies	Genera en el directorio especificado las clases usadas por el servidor (el micromundo).
generateInterfaces	Genera en el directorio especificado las interfaces utilizadas en los proxies del servidor.
generateErlangServer	Genera en el directorio especificado los módulos Erlang que constituyen el enrutador.

Tabla A.2: Especificación de la clase *Generator*, encargada de generar los proxies

# B

## Personalización de las estructuras de control

Entre los objetivos del trabajo se encuentra el brindar a los profesores un medio con construcciones sintácticas claras y cercanas al vocabulario de los estudiantes. Son muchas las decisiones tomadas en este sentido, una de ellas el darle al profesor la posibilidad de personalizar las etiquetas de las estructuras de control y de la asignación.

```
{  
"antesDeCondicion":"Si es cierto que",  
"despuesDeCondicion":"entonces hago",  
"sino":"sino hago"  
}
```

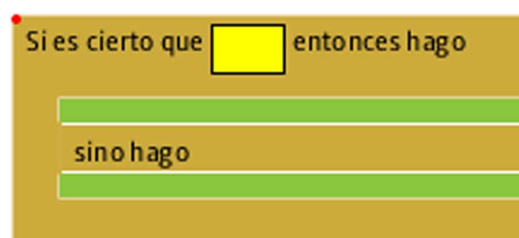


Figura B.1: Apariencia de la estructura de control Si-Entonces a partir del uso del código B.1

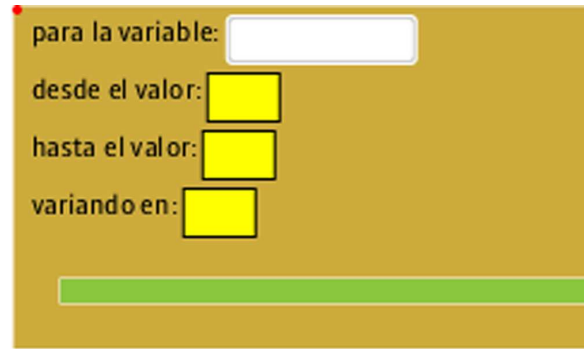


Figura B.2: Apariencia de la estructura de control iterativa a partir del uso del código B.2



Figura B.3: Apariencia de la asignación usando el código B.3

Código B.1: Configuración de las etiquetas de la estructura de control Si-Entonces

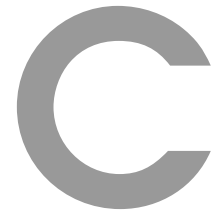
```
{
  "nombreVariable" : "para la variable: ",
  "inicio" : "desde el valor: ",
  "fin": "hasta el valor: ",
  "paso": "variando en: "
}
```

Código B.2: Configuración de las etiquetas de la estructura de control iterativa

```
{
  "etiquetaOrigen": "el valor:",
  "etiquetaDestino": "copiar en la variable:"
}
```

Código B.3: Configuración de las etiquetas de la asignación

Las etiquetas que aparecen en los bloques de la selección simple (si-entonces) y múltiple se controlan mediante el fichero EtiquetasSiEntoncesSino.json tal y como se muestra en la figura B.1. En el caso de la estructura iterativa (figura B.2) se utilizará el fichero EtiquetasDesdeHasta.json y la asignación (figura B.3) será controlada mediante EtiquetasAsignacion.json; todos estos ficheros están en el directorio *./configuracion*.



## Configuración de las aplicaciones

Una vez obtenidos el servidor, el enrutador y el cliente mediante el marco de trabajo se deberá configurar cada una de las partes. Para ello se utilizarán ficheros con extensión *.properties* con las entradas que se muestran en el código C.1.

```
# Nombre que identifique de manera única la estación de
# trabajo cliente (puesto de trabajo de un estudiante)
nombreDelCliente=cliente1
# Puerto que se usará por el sistema (tanto en el cliente
# como en el servidor)
puerto=5901
# IP del servidor
ip=127.0.0.1
# Entorno que se usará
entornoActivo=B
# IP del enrutador
enrutador=127.0.0.1
```

Código C.1: Ejemplo de configuración de las aplicaciones



## Índice de figuras

1	La computación es para la mente lo que un automóvil es para las piernas . . . . .	1
1.1	Naturaleza cíclica de las etapas para la solución de problemas planteadas por Polya . .	8
1.2	Estados anteriores y posteriores de un agente en Stagecast . . . . .	12
1.3	Entorno Scratch, con él se crean micromundos con gran facilidad . . . . .	14
1.4	Distintos tipos de bloques disponibles en Scratch . . . . .	15
1.5	Guión de Scratch con los bloques ensamblados . . . . .	15
2.1	Vista inicial del entorno, mediante ella se accede a cada una de las funcionalidades . .	22
2.2	Conjunto de acciones con las que se podrá actuar sobre el entorno (micromundo) que se esté usando . . . . .	23
2.3	Vista de las estructuras de control de que se dispone . . . . .	23
2.4	Conjunto de tipos de datos a utilizar . . . . .	24
2.5	Vista con las variables que se pueden manipular en un momento dado . . . . .	24
2.6	Vista para acceder a literales y a la interfaz de los tipos . . . . .	25
2.7	Entorno de ejemplo, micromundo que simula un pescador en el mar . . . . .	26
2.8	Arquitectura del sistema . . . . .	29
2.9	Ejemplo de instrucción a ejecutar desde el entorno . . . . .	32
2.10	Estado inicial del micromundo . . . . .	33
2.11	Estado del micromundo luego de la ejecución de la instrucción . . . . .	34
3.1	Entorno Micromundo Robot, escenario reducido en el que se desenvuelve un robot . .	38
3.2	Interfaz del tipo Robot, a ella se llega por la opción Métodos/Literales + Robot . . . .	41
3.3	Consulta que permite conocer la distancia entre un robot y un almacén . . . . .	42
3.4	Orden que permite desplazar un robot hasta el lugar donde esté un almacén . . . . .	42
3.5	Solución al problema planteado . . . . .	46
3.6	Bloque que consulta la cantidad de productos de una fábrica dada . . . . .	48
3.7	Grupo de variables de tipo Fábrica . . . . .	48



**ÍNDICE DE FIGURAS**

B.1 Apariencia de la estructura de control Si-Entonces a partir del uso del código B.1 . . . 63

B.2 Apariencia de la estructura de control iterativa a partir del uso del código B.2 . . . . 64

B.3 Apariencia de la asignación usando el código B.3 . . . . . 64

## Índice de tablas

1.1	Categorías del pensamiento computacional . . . . .	6
2.1	Semántica de las principales anotaciones usadas para configurar los micromundos . . .	27
3.1	Acciones disponibles en el micromundo usado como caso de estudio . . . . .	39
3.2	Objetos disponibles en el caso de estudio . . . . .	40
3.3	Ejemplos de datos y de información, con la relación que poseen . . . . .	47
A.1	Semántica del resto de las anotaciones usadas para configurar los micromundos . . . .	61
A.2	Especificación de la clase <i>Generator</i> , encargada de generar los proxies . . . . .	62



## Índice de códigos

2.1	Uso de la anotación @nombre . . . . .	27
2.2	Uso de las anotaciones @objeto y @etiqueta . . . . .	27
2.3	Uso de las anotaciones @objeto y @sufijo . . . . .	28
2.4	Declaración de la interfaz IRobot . . . . .	28
2.5	Código Ruby obtenido tras la traducción de las instrucciones del entorno . . . . .	33
2.6	Fragmento del proxy del cliente que usa el entorno, escrito en Scala . . . . .	33
2.7	Código del enrutador que lleva los mensajes del entorno hasta el micromundo . . . . .	33
2.8	Fragmento del proxy del servidor que usa el entorno, escrito en Scala . . . . .	34
2.9	Código del enrutador que devuelve las respuestas de los mensajes . . . . .	34
2.10	Fragmento del proxy del cliente que recibe la respuesta de la ejecución . . . . .	34
B.1	Configuración de las etiquetas de la estructura de control Si-Entonces . . . . .	63
B.2	Configuración de las etiquetas de la estructura de control iterativa . . . . .	64
B.3	Configuración de las etiquetas de la asignación . . . . .	64
C.1	Ejemplo de configuración de las aplicaciones . . . . .	65