Enseñanza de programación en el Politécnico Grancolombiano. Situación actual y aplicación de TIC como alternativa de mejora

Nicolás Malaver*, Camilo Rey**, Julián Rodríguez***

Politécnico Grancolombiano

Fecha de recepción: 23 de noviembre de 2010
Fecha de aprobación: 15 de diciembre de 2010

Resumen La enseñanza de programación resulta central en los procesos de formación de los programas relacionados con las ciencias de la computación, la ingeniería de sistemas y la ingeniería de *software*. La formación en esta área particular del conocimiento en dichos programas tiene una particular importancia, porque de su éxito depende buena parte del desempeño académico de los estudiantes en asignaturas futuras del programa. Sin embargo, durante los últimos 20 años, la mayor parte de los procesos educativos han mantenido el mismo enfoque con resultados mezclados. En este trabajo se presenta una breve reseña de los enfoques tanto conceptuales como metodológicos aplicados históricamente, y se contextualiza la situación presente en el Politécnico Grancolombiano en torno a la formación en programación con énfasis en las dificultades actuales; y se presenta una alternativa de aplicación de TIC bajo la forma de una herramienta de *software* para la enseñanza de conceptos básicos de programación que puede facilitar los procesos de enseñanza – aprendizaje en las asignaturas asociadas.

Abstract The teaching of programming is a key point within the educative processes of the programs related to computing, systems engineering, and software engineering. Education in this particular area of knowledge in the aforementioned programs has specific importance because its success depends greatly on the academic performance that students have in future subjects belonging to the program. However, during the last 20 years, most of the educative processes have kept the same approach, with mixed results. In this work, we present a brief description of the conceptual and methodological approaches historically applied. We also contextualize the current situation at the Politécnico Grancolombiano regarding education n programming, with emphasis on the difficulties experienced nowadays, and present an alternate application of ICTs as a software tool for teaching the basic concepts of programming that can facilitate the teaching – learning processes in the related subjects.

* Ingeniero de Sistemas. Universidad Nacional de Colombia. nmalaver@poli.edu.co
** Matemático. Universidad de los Andes. creytorres@poli.edu.co
*** Ingeniero de Sistemas. Universidad Nacional de Colombia. jerodrig@poli.edu.co

Palabras Clave: educación, programación, algoritmos, aprendizaje, enseñanza, TIC.

Keywords: education, programming, algorithms, learning, teaching, ICTs.

1. Introducción

El concepto de programación, en múltiples sentidos, precede la existencia de los computadores. Artefactos cuyo comportamiento seguía un patrón definido por un conjunto de instrucciones codificadas utilizando diferentes medios, existieron desde épocas antiguas. Ejemplos de ello pueden encontrarse en múltiples escenarios desde los autómatas programables de al-Jazari [1], hasta el uso de tarjetas perforadas para controlar comportamientos y almacenar información promovidas por Charles Babbage y Herman Hollerith en el siglo XIX [2]. Incluso, si la inclinación existe, puede argumentarse que mecanismos aún más antiguos demuestran el interés, la capacidad y el trabajo orientados a la definición programática de comportamientos complejos a través de la modificación de la estructura física de las máquinas en cuestión (el mecanismo de Antikythera [3] es un ejemplo que se trae a colación de manera regular).

Sin embargo, el contexto en que se ubica este documento es la programación de computadores bajo la acepción más moderna del término (la construcción de secuencias de instrucciones en un lenguaje formal específico, concebidas para lograr la solución de un problema bien definido, mediante el aprovechamiento de la potencia de cálculo de un computador). En este sentido, las primeras aproximaciones formales al desarrollo de programas pueden trazarse a la década de 1950, momento en el cual los procesos de codificación estaban basados en el uso de lenguaje de máquina y código binario, con las evidentes dificultades que esto implicaba. La aparición subsecuente de lenguajes de programación de alto nivel (FORTRAN en 1954 y COBOL en 1959 fueron dos de los más conocidos y exitosos) implicó un movimiento muy importante hacia adelante en términos de los paradigmas, estrategias, herramientas y potencial de la programación como campo de acción [4,5]. A partir de ese momento, la evolución se desarrolló en proporción geométrica. Nuevos lenguajes aparecieron y fueron olvidados o reemplazados por mejores alternativas, y nuevos paradigmas fueron construidos con el fin de mejorar los procesos de desarrollo de programas de software.

Entre los paradigmas mencionados, uno de los más importantes en el panorama histórico fue el de la programación estructurada que surgió a finales de la década de 1960. La programación estructurada es una aproximación a la solución de problemas de programación, utilizando tres estructuras de flujo del programa: la secuencia, la selección y la repetición [6]. Durante las siguientes décadas, este enfoque sería uno de los preferidos para el desarrollo de *software* y evidentemente, la enseñanza de estrategias de desarrollo de *software*.

Sin embargo, y como resulta evidente, el campo de las aproximaciones al desarrollo de *software*, así como el de los mecanismos utilizados para impartir conocimiento alrededor de este tema no son campos en los que el crecimiento

sea lento o poco constante. A lo largo de las últimas décadas, cambios de paradigma han sido comunes y bienvenidos. La aparición de diferentes y novedosas aproximaciones (la programación orientada por objetos, la programación orientada a agentes, y la programación orientada a aspectos son sólo algunos ejemplos), así como el creciente énfasis en procesos organizados de desarrollo de *software*, han sido eventos de una relevancia muy alta en el escenario del desarrollo de *software*.

Es este, un escenario en constante crecimiento, y uno que reviste una importancia muy alta. Dada la actual ubicuidad de las soluciones de *software*, resulta evidente que buenas prácticas y enfoques adecuados redunden en un beneficio global, que se sale de la esfera del desarrollo abstracto de programas correctos y entre en la escena cotidiana de las soluciones prácticas a problemas físicos. En un mundo con estas características, se necesitan personas capacitadas para desarrollar el tipo de soluciones requeridas. Y resulta por tanto imprescindible ofrecer mecanismos de educación que cultiven este tipo de personas.

2. Educación y programación

El aprendizaje de la programación es un área de la enseñanza en la que no se aprecia un consenso real. Existen diferentes enfoques y posibilidades al momento de enfrentar el proceso de transmisión del conocimiento, y dichos enfoques generan, por definición, diferentes resultados.

Una de las líneas de pensamiento que ha permeado el medio académico es la noción de que la enseñanza de la programación trasciende la mera transmisión de los mecanismos y conceptos técnicos necesarios para escribir un código formalmente correcto. En muchos escenarios educativos se considera la programación "el latín del sílabo escolar" [7], queriendo implicar que el proceso de aprender a programar es uno que desarrolla y fortalece habilidades que van más allá de lo que usualmente se considera, robusteciendo particularmente aquellas de comprensión, descomposición y resolución de problemas, y no solamente creando capacidades técnicas. En ese sentido, resulta necesario separar dos espacios conceptuales fundamentales que en muchas ocasiones se consideraron no solamente adyacentes, sino incluso intercambiables: la habilidad conceptual de resolución de problemas y la habilidad técnica para la escritura de código, es decir, resulta necesario comprender que, como diría Elliot Solloway, saber dónde poner un punto y coma en un programa no lleva forzosamente a una mejor resolución de problemas [8].

Sin embargo esta propuesta que suena sólida y natural en su concepto teórico, se enfrenta con problemas al momento de ser puesta en práctica. Y esto sucede por varios motivos. En primer lugar, a pesar de que efectivamente puede argumentarse que los programadores expertos poseen habilidades de descomposición y solución de problemas que no están disponibles de manera natural para los programadores novicios, no es clara la ruta a través de la cual se llega a ese estado. En la mayor parte de los casos, los programadores no pueden explicar de una manera no ambigua el camino que los llevó al nivel de abstracción y comprensión en el que se encuentran. La transmisión de herramientas y habilidades para descomponer un problema, plantear una solución al mismo, escribirla en un lenguaje determinado

y finalmente identificar y resolver los errores que puedan presentarse durante la implementación es un proceso que ha demostrado no ser para nada simple, y en muchas ocasiones difícilmente repetible. Por otro lado, la separación de la solución del problema y su posterior implementación en un lenguaje específico no resulta tan simple como se desearía, pues muchas de las alternativas existentes para resolver una situación dependen de las herramientas y opciones que ofrezca el lenguaje a utilizar [7].

Es claro entonces que la enseñanza de las habilidades y competencias asociadas con la programación presentan retos que deben ser superados, si se pretende obtener un nivel de aprendizaje adecuado y parejo en los estudiantes. Buscando soluciones en este sentido, se han planteado un número de enfoques que abordan la cuestión desde diferentes perspectivas. Sin embargo, una aproximación resulta evidente en esta área de conocimiento, y es apoyada en diversos espacios y escenarios educativos: el uso de herramientas tecnológicas para superar la brecha cognitiva que existe entre los estudiantes y el tema a aprender. El trabajo en este sentido ha sido abundante y ha producido un sinnúmero de herramientas que pueden clasificarse en varias categorías [9]:

Herramientas de organización semántica: son herramientas que apoyan a los educandos para que apropien el conocimiento a través de un proceso de estructuración del mismo. En esta categoría se encuentran los motores de bases de datos y las herramientas para la generación de mapas conceptuales.

Herramientas de modelado dinámico: este tipo de herramientas permite encontrar y representar los vínculos que existen entre las ideas que se trabajan. Aquí están herramientas tales como hojas electrónicas, los sistemas expertos, y los micromundos.

Herramientas de interpretación de información: en esta categoría se encuentran herramientas que permiten organizar y procesar grandes volúmenes de información. Herramientas tales como las de visualización de datos están en este grupo.

Herramientas de conversación y construcción: se agrupan aquí herramientas que permiten la construcción de conocimiento a través del intercambio de información. Ejemplos de este tipo de herramienta son los chats y las teleconferencias.

De particular interés en esta taxonomía es el grupo de los micromundos. Las aplicaciones que pueden ser catalogadas en esta división ofrecen al usuario un conjunto de herramientas con las que puede manipular el comportamiento de elementos que existen dentro del ambiente modelado por el *software*, de acuerdo con un conjunto de reglas previamente definidas. Esta aproximación experimental y que permite la modificación a voluntad del comportamiento de un escenario, se presta de manera natural y muy efectiva para la enseñanza de las habilidades cognitivas relacionadas con la programación. La mayor parte de las aplicaciones de *software* en esta categoría, se adhieren a una metáfora conceptual

que encapsula las abstracciones de comportamiento, características y secuencia que resultan centrales al proceso de programación. Ejemplos de este tipo de herramientas de *software* son: Scratch [10], Phrogram [11], JKarel [12], Alice [13], CeeBot-3 [14]. Algunas de estas herramientas asumen una aproximación que utilizan estructuras gráficas reminiscentes de los diagramas de flujo (Scratch por ejemplo emplea una representación estructural que genera una secuencia visual de ejecución que, a pesar de no ser un diagrama de flujo en el sentido tradicional del concepto, guía visualmente el comportamiento del algoritmo diseñado), sin embargo los autores no encontraron herramientas que utilicen de manera completa el concepto ortodoxo de diagrama de flujo. Existen desde luego plataformas integradas de *software*/*hardware*, la más conocida de las cuales es probablemente, LEGO Mindstorms [15]. En general, todas estas herramientas comparten una aproximación que genera un lenguaje bien sea visual o conceptual, utilizado para describir el universo, sus reglas, y los elementos del algoritmo que pueden ser implementados en el escenario que dicho universo y sus reglas describen.

3. Contexto local

En el escenario de formación del Politécnico Grancolombiano pueden encontrarse dos asignaturas que buscan ofrecer al estudiante las herramientas básicas para enfrentarse a problemas de programación de computadores: Pensamiento Algorítmico y Programación de Computadores[1]. La asignatura Pensamiento Algorítmico es una adición más o menos reciente al pensum que busca introducir de manera informal conceptos asociados con estrategias de resolución de problemas y con la utilización de conceptos matemáticos, geométricos, algebraicos y lógicos en dichas estrategias. La metodología utilizada en las clases se apoya muy fuertemente en la resolución de problemas de tipo lógico – matemático, en primera instancia mediante una aproximación intuitiva por parte de los estudiantes, para luego mostrarles patrones básicos de solución de problemas que siembren las semillas de métodos más formales como "dividir y vencer", o que utilicen conceptos más rigurosos como la inducción.

Luego de aprobar esta asignatura, los estudiantes pasan a cursar Programación de Computadores, donde tienen su primer encuentro con los elementos formales del proceso de construcción de algoritmos y su posterior implementación en un lenguaje de programación. En esa asignatura se ofrecen las bases conceptuales necesarias para la programación de computadores, y se dispone de los espacios prácticos para poner a prueba dichas bases. En términos de paradigma y alcance, la asignatura promueve el estudio de la programación estructurada, por cuanto este enfoque muestra los componentes fundamentales para la implementación de

[1] Evidentemente, existen otras asignaturas que tocan temas más avanzados, pero en el contexto de este documento no se pretende abordar estos temas. El foco principal de atención del trabajo aquí presentado es la construcción de habilidades básicas en programación que puedan servir de base a procesos de enseñanza – aprendizaje posteriores.

un algoritmo sin complicar el proceso más allá de lo necesario para esta etapa básica de formación. Como lenguaje de programación, actualmente se utiliza Java para ofrecer a los estudiantes el contacto con uno de los lenguajes con más penetración en el mercado laboral en la actualidad, al mismo tiempo que tienen contacto con un lenguaje abierto, con una gran comunidad de usuarios y un vasto conjunto de recursos de aprendizaje fuera del aula. Como metodología, se cuenta con tres sesiones semanales, una de las cuales está dedicada a la introducción teórica de conceptos nuevos ilustrados por ejemplos y talleres elaborados con apoyo del docente, mientras que las otras dos están enfocadas en trabajo práctico autónomo por parte de los estudiantes con asistencia del docente cuando el estudiante encuantra dificultades.

A continuación se relacionan las dificultades más relevantes que se han encontrado al momento de articular el proceso de enseñanza – aprendizaje con los estudiantes:

- A pesar de la inclusión de Pensamiento Algorítmico, los estudiantes llegan a la asignatura Programación de Computadores con deficiencias conceptuales en términos de la aplicación de conceptos matemáticos adquiridos previamente en su educación media. Se aprecian deficiencias para identificar dichos conocimientos como herramientas válidas de solución a problemas prácticos, así como para aplicarlos de manera correcta una vez son identificados como una opción teórica de solución.
- La decisión de utilizar Java como lenguaje de programación para la asignatura ha tenido resultados mixtos. Mientras por un lado es evidente la ventaja que presenta el tener contacto temprano con un lenguaje que está siendo utilizado actualmente en la mayor parte de los escenarios productivos, y que es fruto de un proceso de maduración riguroso, también resulta evidente que las herramientas de desarrollo e incluso los conceptos mismos inherentes al lenguaje resultan en ocasiones ser contraproducentes al tratarse de una asignatura básica con estudiantes de primeros semestres. Concretamente, el IDE[2] utilizado en las clases (actualmente Eclipse (16), aunque se han hecho pruebas utilizando NetBeans [17], y BlueJ [18]) puede ser intimidante para los alumnos, y el hecho de enseñar programación estructurada utilizando un lenguaje orientado por objetos genera la necesidad de ignorar ciertos elementos y características del código (clases, constructores, etc.), lo que no resulta ideal.
- A pesar de que las herramientas utilizadas ofrecen opciones como la ejecución paso a paso del código, este proceso no resulta intuitivo para los alumnos. Esto resulta infortunado por cuanto una de las dificultades más notorias es la de "visualizar" el funcionamiento de un algoritmo en el tiempo. Para los estudiantes resulta complejo entender cómo una serie estática de instrucciones se comporta en tiempo de ejecución; particularmente las sentencias condicionales y los ciclos resultan confusos en términos de su interpretación en el tiempo. Las pruebas de escritorio ayudan mucho en este sentido, pero en muchos de los casos no son prácticas en términos de tiempo.

[2] *Integrated Development Environment*, Entorno Integrado de Desarrollo.

- La sintaxis del lenguaje se convierte en un obstáculo adicional a la complejidad que presenta la conceptualización de la solución. Existen escenarios en los que los alumnos tienen claridad respecto al proceso a seguir para solucionar un problema, pero presentan inconvenientes a la hora de escribir la solución que diseñan en el lenguaje de programación. La apropiación del conjunto de reglas inherentes a un lenguaje de alto nivel añade otro nivel de dificultad al aprendizaje de programación.

A pesar de ser claro que la mayor parte de las dificultades aquí expresadas son naturales y deben ser manejadas y eliminadas a lo largo del proceso de aprendizaje, este proceso resultaría mucho más fluido si fuera posible eliminarlas o reducirlas en un momento temprano de la dinámica enseñanza-aprendizaje.

4. Solución propuesta

Con los conceptos y situaciones presentados anteriormente, y teniendo en cuenta el escenario local esbozado, es posible ahora presentar la propuesta de solución a la que se refiere este documento.

En aras de solventar los inconvenientes experimentados en el proceso de enseñanza de programación en el Politécnico Grancolombiano, se propone la implementación de una herramienta de *software* que apoye la dinámica en el aula y sirva, a la vez, como herramienta de estudio y solución de problemas de programación para los estudiantes de la asignatura. Como características generales, a través de las cuales se busca paliar los inconvenientes presentados previamente, se tienen:

- **Utilización del enfoque de programación estructurada.** Temprano en el proceso de definición de la solución a proponer se decidió seguir utilizando la programación estructurada como aproximación. Esto se debió al hecho de que sin duda no es la aproximación más utilizada en términos profesionales, sí es una que ofrece un compromiso muy adecuado e interesante en términos de facilidad de aprendizaje *versus* flexibilidad y potencia de aplicación. Además de esto, con los conceptos claros de programación estructurada es posible migrar a otras aproximaciones más complejas de una manera más sencilla.
- **Abstracción del lenguaje de programación.** Dado que uno de los inconvenientes mencionados se debe a la dificultad añadida de apropiar la sintaxis de un lenguaje de programación de alto nivel, se sugiere enfocar el proceso de enseñanza-aprendizaje de una forma tal que no sea necesario involucrar este proceso extra. Por ello, resulta evidente que abordar el aprendizaje de un lenguaje de programación particular es necesario para un proceso completo de aprendizaje de programación, y además se considera importante que en esta fase inicial de dicho proceso se cimienten las bases conceptuales y se apropien los constructos básicos del área disciplinar.
- **Representación visual del proceso.** De la mano del punto anterior, y como consecuencia del mismo, se propone una herramienta que coloque la base de la solución, así como la validación de la misma, en representaciones

visuales de fácil comprensión. En este sentido, y luego de explorar otras alternativas, se decidió optar por la utilización de diagramas de flujo; ya que por definición, un diagrama de flujo es la representación gráfica de un proceso, donde el control fluye de manera explícita a través de un conjunto de figuras interconectadas [19], porque resulta apenas natural utilizar este tipo de representación, por demás estándar, en la aproximación propuesta.

- **Facilidad en el proceso de construcción de la solución.** Un elemento absolutamente fundamental a tener en cuenta durante la conceptualización e implementación de la herramienta fue el hecho de que ésta no debía convertirse en un obstáculo para el proceso. El foco debía estar en la solución del problema, no en el aprendizaje alrededor de la herramienta que permitiría implementar dicha solución. Luego de enfrentar dificultades en este sentido con las herramientas de programación utilizadas hasta ese momento, resultaba imperativo que ese no fuera el caso.
- **Facilidad en el proceso de seguimiento durante la ejecución.** Teniendo en cuenta que una de las mayores dificultades a la que se enfrentan los estudiantes es la comprensión del proceso de ejecución en la dimensión "tiempo", un factor decisivo al momento de considerar una solución de *software* al problema fue la facilidad con la que el usuario pudiera verificar el estado de la ejecución del algoritmo construido; así como la fácil comprensión de la ruta seleccionada en los puntos de decisión y repetición (sentencias condicionales y ciclos). Asociado a este punto, fue necesario considerar un esquema de gestión de variables, valores y tipos que permitiera una fácil adición y eliminación, así como la evaluación clara de su valor durante la ejecución del algoritmo.
- **Cercanía a los paradigmas técnicos asociados con la programación.** No obstante la tentación que existió de abordar el proceso utilizando una metáfora "blanda" para la construcción de los algoritmos (ambientes virtuales, como en Alice [13], o elementos móviles animados como en Scratch [10]), se decidió que teniendo en cuenta el nivel académico de los estudiantes, y en aras de ofrecer una transición lo más suave posible hacia aproximaciones más formales de desarrollo, tanto la interfaz como la metodología de construcción de algoritmos y soluciones debería ser lo más "espartana" y "técnica" posible, sin dejar de lado en ningún momento la facilidad de uso[3]. Este fue otro motivo por el cual se utilizó el concepto de diagrama de flujo y la utilización de variables con tipos formales. Se propendió sin embargo, por la eliminación de elementos gráficos innecesarios, y por la limpieza en la interfaz, de tal manera que, de nuevo, el foco se encontrara en la solución, y no en la herramienta.
- **Generación de código.** Como característica que facilitara la transición a ambientes más formales de desarrollo, se consideró necesario que la herramienta tuviera la capacidad de generar código a partir del diagrama de flujo

[3] No se pretende en ningún momento desvirtuar la utilidad de este tipo de enfoques "blandos". Es indiscutible el aporte que hacen a la comprensión de algunos conceptos; y de hecho, la integración de este tipo de herramientas se contempla como un desarrollo futuro para la herramienta. Sin embargo, en este punto del desarrollo y el proceso general de aplicación, se consideró de mayor importancia el acercarse de una manera más formal al proceso de conceptualización e implementación de algoritmos.

que representa el algoritmo diseñado. Inicialmente, y de acuerdo con la orientación de la formación en programación en el Politécnico Grancolombiano, se decidió que la generación de código Java era lo más recomendable.

Teniendo en cuenta los aspectos mencionados previamente, y buscando atacar las debilidades encontradas en el proceso de formación en programación en el Politécnico Grancolombiano, se diseñó e implementó **Ariadna**, una herramienta de *software* basada en la construcción de diagramas de flujo con elementos propios de la programación estructurada, que puede ser utilizada en el aula en cursos de programación, y que tiene como foco esencial el soporte de la construcción de soluciones algorítmicas a través de la simplificación de las herramientas y procesos asociados con esta dinámica, sin perder de vista el formalismo esencial en un proceso de educación superior. Una descripción detallada de **Ariadna** y el flujo de trabajo asociado con su uso están fuera del alcance del presente documento, y se reservan para un documento posterior. Sin embargo, se presentan a continuación algunas imágenes de su interfaz y elementos básicos.

En primera instancia se muestra en la figura 1, la interfaz de configuración de variables de algoritmo. En esta ventana es posible adicionar y eliminar las variables que estarán disponibles en **Ariadna** durante la construcción y ejecución del algoritmo. Se muestra aquí, pues es un ejemplo de la aproximación simple y directa a la gestión de información y procesos dentro de la herramienta. Como nota adicional, se mantienen los tipos de datos en su notación formal, para facilitar el paso desde una notación simbólica estándar, y hacia una notación técnica formal.

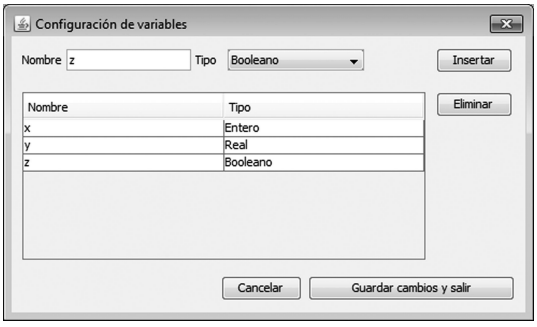

Figura 1. Interfaz de configuración de variables de algoritmo.

En la figura 2, se muestra el área de trabajo de la aplicación, con un algoritmo de prueba desplegado en ella. Nótese que los elementos utilizados para representar los bloques constructivos del diagrama, que son a su vez la representación del algoritmo construido, son los utilizados de manera estándar en la generación de diagramas de flujo. Es de resaltar también que restricciones a lo largo del proceso de construcción del algoritmo hacen que la estructura del diagrama resultante sea

no sólo formalmente correcta (a través de la prohibición de colocar elementos en posiciones no válidas), sino fácil de leer y seguir (el "crecimiento" del diagrama siempre se da de izquierda a derecha y de arriba hacia abajo, garantizando de esta forma que no sea posible incurrir en errores estructurales de programación, como por ejemplo retornar a puntos previos en el código).

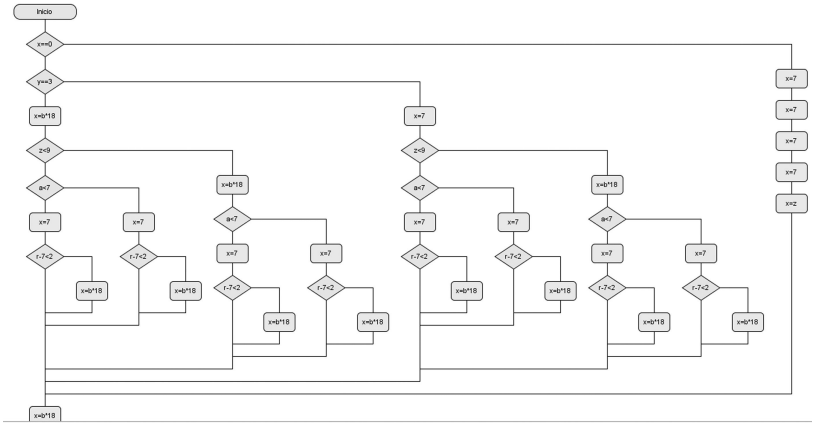

Figura 2. Área de trabajo de **Ariadna**, con un diagrama de prueba.

5. Trabajo futuro

El siguiente paso en el proceso de aplicación de la solución es extender el escenario de pruebas a las aulas de programación en el Politécnico Grancolombiano. Hasta el momento se han realizado pruebas limitadas con resultados alentadores, pero debido a la dinámica semestral de los cursos, y que es necesario integrar la herramienta al principio de los mismos, las pruebas formales comenzarán en el primer semestre de 2011.

En términos de características del *software*, se pretende integrar un ambiente de simulación en el que se pueda verificar de una manera visual la ejecución del algoritmo diseñado, más allá de las herramientas de seguimiento que ofrece la herramienta actualmente, y que están orientadas a seguir la ejecución sobre el diagrama de flujo. La posibilidad de controlar la actividad de un elemento visual animado, como por ejemplo un robot u otra representación de ese estilo podría, en conjunto con las características ya expuestas de **Ariadna**, facilitar aún más el proceso de apropiación de los conceptos básicos de programación.

Gracias a la arquitectura de diseño de **Ariadna**, la idea de adicionar lenguajes para los cuales se genere código, es de simple implementación. Esto ofrecerá flexibilidad a la herramienta a la vez que da la posibilidad al estudiante de comparar la sintaxis de varios lenguajes en su proceso de aprendizaje.

En general, se pretende que el proceso de prueba de **Ariadna** en ambientes reales de formación, con un número grande de estudiantes, ofrezca información y una perspectiva más amplia de las posibilidades que ofrece el *software* como herramienta educativa. Con base en este proceso de retroalimentación se pretende seguir adelante con el desarrollo del *software*, ampliando cada vez más sus capacidades y su alcance en el contexto de la enseñanza de programación.

Referencias

1. Rosheim, M. E.: Robot Evolution: The Development of Anthrobotics. s.l: Wiley-Interscience. (2008)
2. Heide, L.: Punched-Card Systems and the Early Information Explosion, 1880–1945 (Studies in Industry and Society). s.l: The Johns Hopkins University Press. (2009)
3. Wilford, J. N.: Discovering How Greeks Computed in 100 B.C. The New York Times. 31 de Julio de 2008. (2008)
4. Wexelblat, R. L.: History of Programming Languages (Acm Monograph Series). s.l: Academic Press, (1981)
5. Bergin, T. J., Gibson, R. G.: History of Programming Languages, Volume 2. s.l: Addison-Wesley Professional, (1996)
6. Dijkstra, E. W., Hoare, C. A. R. y Dahl, OJ.: Structured Programming (A.P.I.C. Studies in Data Processing, No. 8). s.l: Academic Press, (1972)
7. Sleeman, D.: The Challenges of Teaching Computer Programming. Communications of the ACM. Septiembre de 1986.
8. Solloway, Elliot.: Learning to program = learning to construct mechanisms and explanations. Communications of the ACM. Septiembre de 1986.
9. Jonassen, David H. y Reeves , Thomas C.: Association for Educational Communications and Technology. Learning With Technology: Using Computers As Cognitive Tools. [En línea] [Citado el: 21 de Agosto de 2010.] `http://www.aect.org/edtech/ed1/24/index.html`.
10. *Scratch*. [En línea] MIT. [Citado el: 22 de Agosto de 2010.] `http://scratch.mit.edu/`.
11. *Phrogram*. [En línea] The Phrogram Company. [Citado el: 22 de Agosto de 2010.] `http://phrogram.com/`.
12. *JKarel*. [En línea] [Citado el: 22 de Agosto de 2010.] `http://www.cs.tufts.edu/comp/10F/JKarel.htm`.
13. *Alice*. [En línea] Carnegie Mellon University. [Citado el: 22 de Agosto de 2010.] `http://www.alice.org/`.
14. *CeeBot-3*. [En línea] Epsitec. [Citado el: 22 de Agosto de 2010.] `http://www.ceebot.com/ceebot/3/3-e.php`.
15. *LEGO Mindstorms*. [En línea] LEGO. [Citado el: 22 de Agosto de 2010.] `http://mindstorms.lego.com/en-us/overview/NXT_Software.aspx`.
16. *Eclipse*. [En línea] The Eclipse Foundation. [Citado el: 23 de Agosto de 2010.] `http://www.eclipse.org/`.
17. *NetBeans*. [En línea] Oracle Corporation. [Citado el: 23 de Agosto de 2010.] `http://netbeans.org/`.
18. *BlueJ*. [En línea] La Trobe University, University of Kent at Canterbury, Sun Microsystems. [Citado el: 23 de Agosto de 2010.] `http://www.bluej.org/`.
19. Fitter, M., Green, T.: When do diagrams make good computer languages?. International Journal of Man-Machine Studies, Vol. 11. (1979)