Programación Matemática y Software (2013)
Vol. 5. No 1. ISSN: 2007-3283

Recibido: 6 de septiembre del 2012 Aceptado: 21 de abril del 2013
Publicado en línea: 25 de junio del 2013

Aprendizaje Basado en Proyectos Utilizando L-Systems en un Curso de Compiladores

J. Jesús Arellano-Pimentel[1], Omar Nieva-García[1], Ignacio Algredo-Badillo[1]

[1]Ingeniería en Computación, Universidad del Istmo, Sto. Domingo Tehuantepec, Oaxaca

{jjap@sandunga, omarng@bianni, algredobadillo@sandunga}.unistmo.edu.mx

Resumen. El presente trabajo propone el empleo de la estrategia educativa de Aprendizaje Basado en Proyectos (ABP) utilizando los L-Systems como caso de estudio en un curso de Compiladores. Dicho caso de estudio ofrece una alternativa viable para abordar los conceptos base del proceso de compilación, además de que permite la aplicación de la metodología de desarrollo incremental estudiada en cursos previos de Ingeniería de Software. En este trabajo se plantea el desarrollo de un generador de L-Systems que tenga una interfaz gráfica de usuario para compilar e interpretar especificaciones L-Systems 2D y posteriormente traducirlas a código OpenGL optimizado. Además, este generador sirve como punto de partida para otro proyecto relacionado con la programación distribuida y paralela.

Palabras Clave. ABP, L-Systems, Compiladores, Desarrollo Incremental.

1 Introducción

El estudio de los compiladores es una parte esencial dentro de los planes y programas de estudio de las carreras afines a las ciencias computacionales. En México podemos citar dos ejemplos, el caso de la carrera de Ingeniería en Computación de la UNAM [1] y la carrera de Ingeniería en Computación de las universidades del SUNEO [2] en el estado de Oaxaca. Ambas carreras cuentan en su retícula con la asignatura de compiladores teniendo un objetivo de aprendizaje y contenido temático muy similar.

En un escenario típico de esta asignatura, al finalizar el curso, se pide a los alumnos desarrollar un pequeño compilador o intérprete que traduzca de un sencillo lenguaje de alto nivel basado en una sintaxis tipo C a código en lenguaje ensamblador. Prácticamente siempre se trata del mismo proyecto: una calculadora para realizar operaciones aritméticas que también utilice estructuras secuenciales y de control. Este escenario presupone como antecedente reticular un curso de lenguaje ensamblador, aquí el problema estriba en que esto no siempre se cumple. Cuando no se tiene como antecedente un curso de lenguaje ensamblador suelen darse las siguientes situaciones:

1. El alumno lo resuelve como puede, “es su problema”.
2. El profesor omite los temas que involucren lenguaje ensamblador o sólo da una revisión superficial de estos.
3. El alumno desarrolla un traductor (compilador) fuente-fuente de alto nivel, pero sin dejar de profundizar en el conocimiento y compresión de los conceptos básicos de generación y optimización de código.

En el presente trabajo se plantea el desarrollo incremental de un generador de L-Systems [3] como caso de estudio para un curso de compiladores enmarcado en la estrategia educativa ABP [4]. Dicho generador es un traductor fuente-fuente de alto nivel que permite la edición, con resaltado

tipográfico, de una especificación dada en términos de los L-Systems para posteriormente interpretarla y producir código OpenGL optimizado.

Los siguientes apartados describen el marco teórico, el diseño y construcción del generador, los resultados y trabajo futuro, y por último las conclusiones. Dentro del marco teórico se abordan los principios del ABP, el contenido de un curso típico de compiladores y los conceptos básicos relacionados con los L-Systems. El diseño y construcción del generador detalla la etapa de planeación, la estructura del generador y cada una de las fases de compilación.

2 Marco teórico

2.1 Aprendizaje Basado en Proyectos (ABP)

La premisa básica del ABP parte de la filosofía pragmática que establece que el conocimiento se adquiere a través de las consecuencias observables y el aprendizaje implica el contacto directo con las cosas [5]. Así, el ABP es definido en [4] como la experiencia educativa donde los estudiantes han desarrollado destrezas, técnicas y preparación en un área particular del conocimiento y son retados a llevar a cabo un proyecto en esta área.

El análisis hecho en [6] sobre varios trabajos que utilizan ABP señala algunos de los beneficios esta estrategia de aprendizaje: aumento de la motivación en los estudiantes lo que propicia una participación más activa en clase, mayor retención de conocimientos al tener estudiantes comprometidos con un proyecto estimulante, incremento de la autoestima en los estudiantes al enorgullecerse de lograr algo que tenga valor fuera del aula de clase, entre otros.

Cuando se trabaja con ABP es importante que el proyecto a realizar esté enfocado a lograr el objetivo de aprendizaje del programa oficial de estudios, abarcando la gran mayoría de los temas establecidos. Al concluir el curso o asignatura los estudiantes deben demostrar sus conocimientos adquiridos a través del producto final

coherente con el objetivo del programa.

Este trabajo toma como base la propuesta del modelo ABP hecha en [6] considerando los siguientes aspectos: 1) el desarrollo del proyecto dirige el avance sobre los contenidos temáticos, 2) el producto de software final se define al inicio del curso y es la meta a alcanzar, 3) dicho producto debe ser desarrollado enteramente por los estudiantes generando productos agregados como exposiciones o presentaciones en foros de divulgación científica. En este contexto el profesor debe fungir como guía que generalmente asigna y provee las herramientas tecnológicas necesarias para el desarrollo y conclusión del proyecto. Además, el rol del profesor también incluye proporcionar el contenido temático de la asignatura antes y durante el desarrollo del proyecto. La estrategia ABP descrita anteriormente fue aplicada en la asignatura de compiladores.

2.2 Asignatura de compiladores

A grandes rasgos, un compilador es una herramienta que lee un programa escrito en algún lenguaje fuente y lo traduce a un programa equivalente en otro lenguaje objeto, durante esta traducción se deben presentar los errores del programa fuente en caso de que existan [7]. Sin embargo, en su definición tradicional un compilador se concibe como un programa escrito en un lenguaje fuente de alto nivel y genera su equivalente en código máquina.

En la literatura clásica de compiladores [8] se menciona que el desarrollo de estas herramientas involucra diversas áreas del conocimiento tales como: los lenguajes de programación, la arquitectura de computadoras, la teoría de lenguajes, los algoritmos y la ingeniería de software. No obstante, la distribución de materias en algunas retículas, por ejemplo en [1] y [2], no manejan como antecedente la arquitectura de computadoras, lo que obliga a llevar la materia de compiladores desde un

enfoque de procesadores de lenguaje o intérpretes. Independientemente de esto, los temas centrales de la asignatura de compiladores se corresponden con el modelo de análisis y síntesis de la compilación (ver Figura 1). La fase de análisis divide al programa fuente en los elementos que lo componen y crea una representación intermedia. La fase de síntesis construye el programa objeto a partir de la representación intermedia.

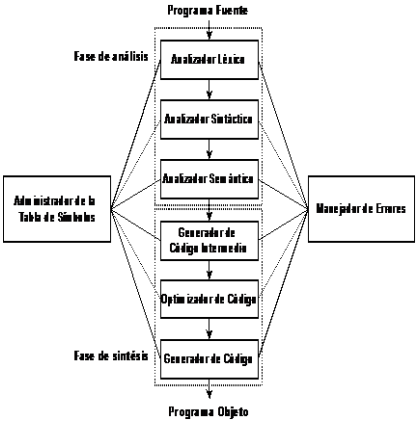


Fig. 1. Fases de un compilador.

Particularmente, el programa de estudios impartido en el la Universidad del Istmo [9] perteneciente al SUNEO contiene las siguientes unidades: 1) introducción al proceso de compilación, 2) análisis léxico, 3) análisis sintáctico, 4) análisis semántico, 5) generadores de código y 6) optimización de código. La relación que guardan estas unidades respecto a las fases del compilador, esquematizadas en la Figura 1, resulta evidente. El problema particular de este contenido temático, al igual que en el caso de [1], es que los temas referentes a la optimización y generación de código necesitan como antecedente curricular una asignatura de lenguaje ensamblador para que se pueda profundizar en ellos, ambas retículas carecen de este antecedente. Luego entonces, es necesario idear una alternativa viable para desarrollar un proyecto que involucre estas dos fases del proceso de compilación pero sin la necesidad de dominar el lenguaje ensamblador. La alternativa que proponemos en este trabajo es el desarrollo de un generador de L-Systems que involucre todas las fases del proceso de compilación.

2.3 L-Systems

Los sistemas Lindenmayer o L-Systems [3] son concebidos como

una teoría matemática de desarrollo de plantas vegetales. Los L-Systems se basan en un concepto de reescritura, que es una técnica para definir objetos complejos mediante reemplazos sucesivos de partes de un objeto inicial simple, empleando un conjunto de reglas de reescritura o producciones.

Cabe mencionar que los L-Systems y las gramáticas de Chomsky, concepto base en el estudio de compiladores, tienen cierta relación y difieren principalmente en la forma de aplicar las producciones. En las gramáticas de Chomsky las producciones son aplicadas secuencialmente, mientras que en los L-Systems son aplicadas en paralelo y simultáneamente. La relación entre las clases de lenguajes de Chomsky y las clases de lenguajes generados por los L-Systems se resume en la Figura 2. Los símbolos OL e IL denotan clases de lenguajes generados por los L-Systems los cuales son libres de contexto y sensibles al contexto respectivamente.

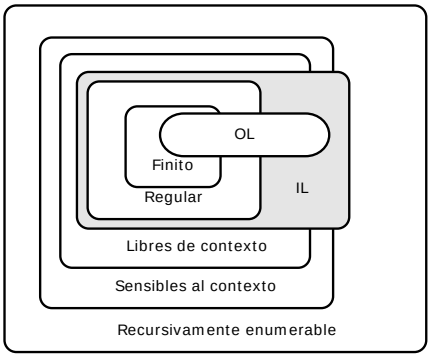


Fig. 2. Relación entre los leguajes de Chomsky y los L-Systems.

DOL-Systems. Esta clase de L-Systems son los más simples, se trata de sistemas deterministas e independientes del contexto. En estos, cada cadena es asociada con una regla de reescritura, por ejemplo la regla *p:* $F \rightarrow F + F - F - F + F$ define una producción, lo cual significa que la letra *F* debe ser remplazada por la cadena $F + F - F - F + F$. El proceso de reescritura comienza con una cadena llamada axioma inicial denotada por *w*. Considerando como axioma inicial la cadena *F*, el primer paso de reescritura reemplaza *F* por $F + F - F - F + F$, en el segundo paso cada F de esta cadena es reemplazado nuevamente por la regla de producción dando como resul-

tado: *F + F – F – F + F + F + F – F – F + F – F + F – F – F + F – F + F – F – F + F + F + F – F – F + F*, y así sucesivamente puede reemplazarse cada *F* de esta cadena resultante por la regla de producción tantas veces como se desee.

La interpretación de las cadenas o secuencias de caracteres generadas se basan en comandos gráficos al estilo del lenguaje de programación LOGO usado ampliamente con propósitos didácticos. Estos gráficos son conocidos popularmente como gráficos de tortuga donde los símbolos (letras y operadores) son interpretados como líneas rectas y ángulos de giro. En este caso particular se tienen los siguientes comandos: *F* o *f* indican a la tortuga avanzar una longitud *d* pintando o sin pintar respectivamente, el + y – indican un giro de la tortuga en un ángulo α a la izquierda o derecha respectivamente. Tomando en cuenta los comandos anteriores la interpretación del DOL-System anterior se observa en la Figura 3.

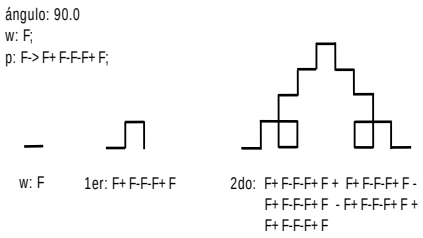


Fig. 3. Ejemplo de DOL-Systems y su interpretación.

OL-Systems. Esta clase de L-Systems se utiliza para modelar el desarrollo de estructuras ramificadas. Emplea un mecanismo de reescritura que opera directamente sobre árboles simétricos. Una regla de reescritura o producción árbol, reemplaza una lista predecesora por un árbol simétrico sucesor de tal forma que el nodo inicial del predecesor es identificado como base del sucesor y el nodo final es identificado con la copa del sucesor. Dicho en términos más simples, se introducen dos nuevos operadores (comandos) para indicar una ramificación, el operador *[* indica el inicio de una rama y el operador *]* señala el fin de la misma. La interpretación en gráficos de tortuga implica las siguientes acciones: ante un opera-

dor *[* se introduce a la pila el estado actual de la tortuga dado por su posición y orientación vigente, con un operador *]* se saca del tope de la pila el estado previo convirtiéndose en el estado actual. Aplicando estos sencillos comandos, la Figura 4 ejemplifica su ejecución en la interpretación de un OL-System.

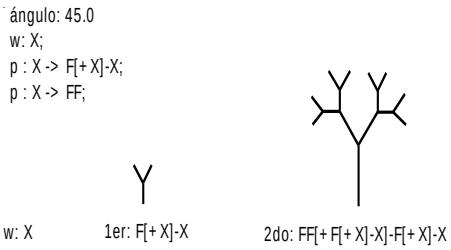


Fig. 4. Ejemplo de OL-Systems y su interpretación.

3 Construcción del generador

3.1 Etapa de planeación

El desarrollo del generador de L-Systems toma como marco de referencia al paradigma del desarrollo incremental descrito en la literatura clásica de ingeniería de software [10]. Por tanto, es muy importante planear los incrementos de software en función de avance en los contenidos temáticos de la asignatura. Desde el inicio del curso se plantea a los alumnos el proyecto a realizar y el temario haciendo énfasis en los productos (incrementos) esperados después de cada unidad.

Es importante mencionar que el incremento a entregar es establecido antes de abordar la teoría para mantener incentivados a los alumnos en el conocimiento y comprensión de los conceptos teóricos para su posterior aplicación en el desarrollo del incremento esperado, estos son:

- Primer incremento: consta de una interfaz de desarrollo integrado (IDE, por sus siglas en inglés) capaz de realizar resaltado tipográfico al momento de editar los tokens válidos.
- Segundo incremento: consiste en la validación sintáctica de la gramática propuesta, notificando de los posibles errores en la especificación del L-System.
- Tercer incremento: consiste en la validación semántica de la

especificación, lo que implica el empleo de la tabla de símbolos.

- Cuarto incremento: implementa la generación de código intermedio, en nuestro caso se trata de la aplicación de las reglas de reemplazo del L-System para generar la secuencia de caracteres que posteriormente serán interpretadas y traducidas.
- Quinto incremento: se trata de la interpretación de caracteres en la IDE desarrollada (código abstracto).
- Sexto incremento: traduce la secuencia de caracteres a código OpenGL (código concreto).
- Séptimo incremento: optimiza la cantidad de vértices generados en OpenGL.

Cada uno de los incrementos descritos anteriormente se corresponde con alguna de las fases del proceso de compilación (análisis o síntesis), por lo tanto también son coherentes con el programa de estudios de la asignatura.

3.2 Estructura y funcionamiento del sistema

El esquema de la Figura 5 muestra la estructura y funcionamiento del Generador de L-Systems. Se trata de una IDE para editar especificaciones L-Systems con resaltado tipográfico, analizarlas de forma léxica, sintáctica y semántica, reportando errores si los hubiera, y finalmente sintetizar las especificaciones generando las secuencias de caracteres e interpretándolas con gráficos de tortuga a la par de generar un archivo OpenGL equivalente que posteriormente será optimizado reduciendo la cantidad de vértices necesarios para su renderización.

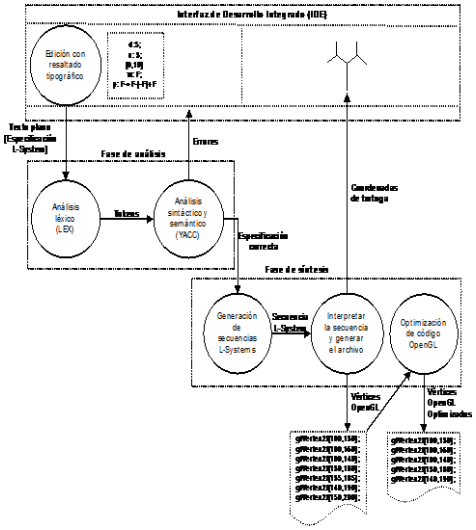


Fig. 5. Estructura y funcionamiento del Generador de L-Systems.

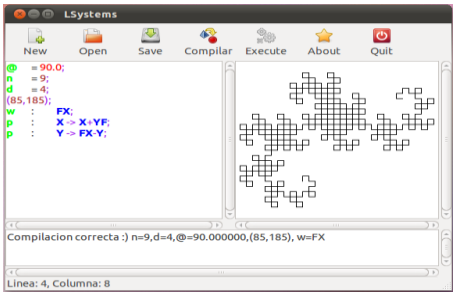
Cabe mencionar que las herramientas utilizadas en la implementación del generador son principalmente Lex&Yacc [11] para el desarrollo de la funcionalidad, y las librerías de Gnome [12] Gtk, Gdk y Pango para el desarrollo de la IDE. El resaltado tipográfico emplea un analizador léxico ad-hoc implementado en C funcionando directamente con la IDE.

La fase de análisis consta esencialmente de dos especificaciones: una en Lex y otra en Yacc. La especificación Yacc tiene inmerso código asociado para realizar las validaciones semánticas y utiliza una tabla de símbolos para verificar la coherencia de la especificación del L-System. Por otro lado, La fase de síntesis difiere en la forma clásica de un compilador ya que primero se genera el código objeto, es decir el archivo OpenGL, y posteriormente se realiza el proceso de optimización bajo la premisa de hacer menos paradas en el camino de la tortuga de tal forma que si se deben trazar *n* líneas contiguas en una misma dirección, entonces se traza solo una para reducir los *n* vértices en solo dos vértices.

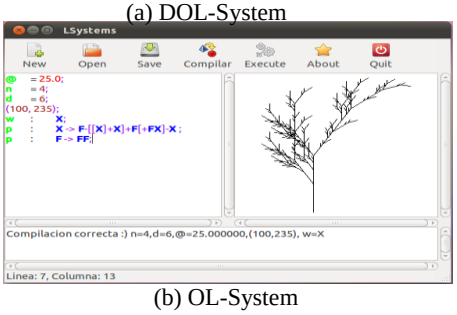
4 Resultados y trabajo futuro

La estrategia educativa ABP fue aplicada durante el Ciclo Escolar 2011-2012A en la carrera de Ingeniería en Computación de la Unistmo al total de alumnos de un grupo reducido de 4 estudiantes. Por cada incremento planeado, los alumnos completaban entre el 80% y 100% de lo esperado. Al final del curso, el 100% de los

alumnos lograron terminar el Generador de L-Systems y expresaron estar motivados en realizar mejoras o nuevos incrementos, sin embargo algunos de ellos (25%) tuvieron ciertas fallas a la hora de ejecutar su aplicación. La Figura 6 muestra el resultado final de la IDE construida, en ésta se observa el resaltado tipográfico, el resultado de la fase de análisis y las interpretaciones de un DOL-System, ver inciso (a), y de un OL-System, ver inciso (b).



(a) DOL-System



(b) OL-System

Fig. 6. Estructura y funcionamiento del Generador de L-Systems.

En la fase de optimización del código OpenGL generado se observó que según la naturaleza del L-System es el nivel de optimización que se logra alcanzar con la estrategia de simplificación de vértices contiguos. Tomando como ejemplos particulares los casos de la Figura 6 tenemos que para el inciso (a) la reducción es de apenas 3% y para el inciso (b) es del 23%. También se observó que la reducción es mayor a medida que se incrementa el número de reemplazos de tal forma que si pasamos de 4 a 6 reemplazos para el caso del inciso (b) la reducción se incrementa de un 23% a un 31%.

En lo que respecta al trabajo futuro se planea mejorar la estrategia de optimización, exportar los L-Systems a formatos como EPS, BMP o GIF y producir L-Systems en 3D. Además, el Generador de L-Systems construido será utilizado como base para implementar estrategias de algoritmos paralelos y distribuidos dada la naturaleza

de las producciones que definen un L-System, las cuales pueden aplicarse de forma paralela y simultánea. Por ejemplo, se planea lanzar varios hilos simultáneos para ejecutar las reglas de reemplazo y la simplificación de vértices en una y/o varias máquinas para medir el desempeño de sistemas distribuidos y paralelos frente a los centralizados como caso de estudio práctico en la asignatura de Sistemas Distribuidos y Paralelos que se imparte en la carrera de Ingeniería en Computación de la Universidad del Istmo. Además, el diseño de un compilador y de una arquitectura hardware tipo procesador tienen una relación muy estrecha, por lo que otras investigaciones y evaluaciones se lograrían en el marco de la materia de Arquitectura de Microcomputadoras.

Conclusiones

Tomar como caso de estudio los L-Systems en un curso de compiladores es factible cuando no se tiene como antecedente curricular una asignatura de Lenguaje Ensamblador ya que es posible abordar los conceptos principales del proceso de compilación, tanto de la fase de análisis como de la fase de síntesis, sin dejar de lado la implementación de los ejemplos del caso tradicional concerniente a una calculadora aritmética.

Para que la construcción del generador tenga éxito, es decir se concluya al 100%, es de suma importancia planear desde un inicio el desarrollo del mismo en el marco de una metodología de desarrollo de software. Para esto, la metodología de desarrollo de software incremental evaluada generó buenos resultados conjuntamente con la estrategia de Aprendizaje Basado en Proyectos, la cual estimula y motiva a aprender desarrollando un producto de software de tal forma que se logra el objetivo de del curso de Compiladores: “Obtener los elementos teórico-prácticos suficientes para analizar, diseñar e implementar compiladores”.

Además, el software obtenido al finalizar el curso puede retomarse en otras asignaturas relacionadas con la programación paralela y distribuida dada la naturaleza de los L-Systems.

Reconocimiento

Los autores agradecen al proyecto PROMEP 103.5/11/5266, UNISTMO-PTC-056 por proporcionar apoyo financiero.

Referencias

1. Universidad Nacional Autónoma de México (UNAM): Oferta educativa, Ingeniería en Computación. FES Aragón. https://www.dgae.unam.mx/planes/. Fecha de consulta: Agosto 2012.
2. Sistema de Universidades Estatales de Oaxaca (SUNEO): Oferta educativa, Ingeniería en Computación. http://www.suneo.mx/oferta_licenciaturas.html. Fecha de consulta: Agosto 2012.
3. Prusinkiewicz, P. Lindenmayer, A.: The Algoritmic Beauty of Plants. Editorial Springer-Verlag (1990).
4. Caseida, C., Dávila, E.: El aprendizaje basado en problemas y proyectos: una estrategia de integración. Universidad Interamericana de Puerto Rico (2006).
5. Instituto Tecnológico y de Estudios Superiores de Monterrey (ITESM). El aprendizaje basado en problemas como técnica didáctica. http://www.ub.edu/mercanti/abp.pdf. Fecha de consulta: Agosto 2012.
6. Galeana de la O, L.: Aprendizaje basado en proyectos. Revista CEUPROMED. Universidad de Colima (2006).
7. Gálvez, R. S., Mora, M. M. A.: "Java a Tope: Traductores Y Compiladores Con

Lex/yacc, Jflex/cup Y Javacc. Universidad de Malaga (2005)

8. Aho, A. V., Sethi, R., Ullman, J. D.: Compiladores. Principios técnicas y herramientas. Editorial Pearson (1990)
9. Universidad del Istmo (UNISTMO): Oferta educativa, Ingeniería en Computación, http://www.unistmo.edu.mx/ing_computacion.html. Fecha de consulta: Agosto 2012.
10. Pressman, R. S.: Ingeniería del software. Un enfoque práctico. Editorial Mc Graw Hill. Quinta edición (2002)
11. Levine, J., Mason, T, Brown, D.: LEX&YACC. Editorial O'Reilly Media, Inc (1995)
12. The GNOME Project, http://www.gnome.org/. Fecha de consulta: Agosto 2012.

J. Jesús Arellano Pimentel estudió y obtuvo la Ingeniería en Sistemas Computacionales por el Instituto Tecnológico de Morelia en 2002 y la Maestría en Ciencias en Ingeniería Eléctrica con opción en Sistemas Computacionales por la Universidad Michoacana de San Nicolás de Hidalgo en 2005. Es Profesor-Investigador asociado “C” de tiempo completo adscrito a la carrera de Ingeniería en Computación en la Universidad del Istmo campus Tehuantepec desde julio de 2005. Sus áreas de interés actuales incluyen: Software Educativo, Prototipos Didácticos, Compiladores, Sistemas Operativos y Robótica Móvil.

Omar Santiago Nieva García estudió y obtuvo la licenciatura en informática por el Instituto Tecnológico de Oaxaca en 1999 y la maestría en ciencias de la computación por el Laboratorio Nacional de Informática Avanzada en 2006 en Xalapa Veracruz. En 2012 concluyó la especialidad a distancia en tecnología educativa en la Universidad Autónoma del Estado de Hidalgo. Es profesor-investigador asociado "C" adscrito a la carrera de ingeniería en computación en la Universidad del Istmo desde marzo de 2007. Ha impartido cursos, talleres y seminarios en diversas instituciones sobre temas de programación desde 1999. También ha sido desarrollador de software trabajando en diversos proyectos en la iniciativa privada. Sus áreas de interés son actualmente Bases de Datos, Sistemas en Tiempo Real y Software Educativo.

Ignacio Algredo-Badillo recibió el grado en Ingeniería Electrónica del Instituto Tecnológico de Puebla (ITP) en el 2002 y los grados de Maestría y Doctorado en Ciencias Computacionales del Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE) en el 2004 y 2008, respectivamente. Desde 2009, se desempeña como profesor-investigador de tiempo completo adscrito a la carrera de Ingeniería en Computación en la Universidad del Istmo. Él es miembro del SIN desde el 2011 y sus líneas de investigación son enfocadas en el diseño y desarrollo de sistemas digitales, arquitecturas reconfigurables, plataformas basadas en el concepto radio software, seguridad, prototipos didácticos, sistemas criptográficos, implementaciones en FPGA, sistemas basados en microcontroladores y microprocesadores y aceleración en hardware para aplicaciones específicas.