SAMTool, una herramienta para el diseño de bucles

José Luis Fernández Alemán
Departamento de Informática y Sistemas
Universidad de Murcia
Campus de Espinardo
30100 Murcia
aleman@um.es

Youssef Oufaska
Departamento de Informática y Sistemas
Universidad de Murcia
Campus de Espinardo
30100 Murcia
oufaska@um.es

Resumen

Este trabajo presenta una propuesta docente que tiene como objetivo el aprendizaje activo del diseño iterativo en un curso introductorio de programación. Fundamentalmente, se realizan dos contribuciones: (1) un enfoque para abordar problemas algorítmicos de diseño iterativo basados en el razonamiento inductivo, en esquemas algorítmicos y en cuatro modelos de acceso secuencial; (2) una herramienta web llamada SAMTool (Sequential Access Model Tool) para soportar este enfoque. Los estudiantes pueden utilizar SAMTool en tres etapas: (1) obtener el código dependiente del problema para resolver un problema iterativo; (2) definir un esquema algorítmico en un modelo de acceso secuencial, actualmente SAMTool trabaja con esquemas algorítmicos de búsqueda y recorrido; (3) generar un programa a partir de un problema, un esquema algorítmico, un lenguaje de programación y una secuencia en el lenguaje de programación elegido.

1. Introducción

En la literatura científica, la mayoría de autores llegan a la misma conclusión: aprender a programar es difícil, en especial las estructuras de control, los análisis de casos, los *arrays* y la recursión. La mayoría de los estudiantes tienen importantes carencias en el diseño y resolución de problemas, y por tanto, encuentran grandes dificultades en llegar a construir un programa [11]. Numerosos estudios muestran que los estudiantes de cursos introductorios de programación no entienden adecuadamente los bucles [15]. Aunque esta abstracción de control es aparentemente simple, los programadores noveles, e incluso aquellos que son algo más experimentados, cometen muchos errores. En este sentido, se han publicado varios trabajos sobre el alto porcentaje de errores que tienen los programadores en bucles e instrucciones condicionales frente a otras construcciones de los lenguajes de programación [11].

La teoría y la investigación empírica han mostrado que un programador experimentado, cuando diseña un programa, inconscientemente categoriza los problemas aplicando patrones que los programadores principiantes desconocen [5][15]. Como señaló Pratt [9], la mayoría de los programas en programación imperativa siguen unos pocos patrones estándard, en su mayoría relacionados con el procesamiento secuencial de elementos en determinadas estructuras de datos. En un enseñanza basada en patrones, los estudiantes adquieren la capacidad de reconocer qué patrón pueden aplicar para resolver un problema [5][8]. Desafortunadamente, en la mayoría de los cursos de introducción a la programación, el aprendizaje de estructuras iterativas no viene acompañada de un estudio de los modelos de acceso secuencial y su relación con diversos patrones de bucle básicos. Los bucles suelen introducirse mediante ejemplos, en lugar de utilizar un proceso disciplinado basado en los modelos de acceso secuencial, esquemas algorítmicos y razonamiento inductivo. SAMTool es una herramienta que soporta el enfoque propuesto en [1], donde se describe un método para deducir e implementar patrones de bucle a partir de cuatro modelos de acceso secuencial.

Este trabajo se organiza en las siguientes secciones: después de esta introducción, la Sección 2 justifica la importancia de este recurso docente mediante la revisión de trabajos relacionados. La Sección 3 presenta SAMTool, una herramienta que soporta la construcción de esquemas algorítmicos iterativos y la generación de programas iterativos adaptados por el programador. Finalmente, la Sección 4 expone las conclusiones y los trabajos futuros.

2. Trabajos relacionados

Cuando se habla de patrones, existe cierta confusión en su definición y terminología [11]. Los términos ingleses *chunk*, *plan*, *schema*, *frame*, *script* y *pattern* suelen utilizarse como sinónimos en este contexto. En esta sección se seguirá el término adoptado por cada autor y nosotros utilizaremos el término esquema para referirnos a esta representación.

Como señala Pratt [9], después de hacer un estudio sobre los bucles en programas escritos en Pascal, gran parte del código que utiliza estructuras de control se

ajusta a unos patrones generales, en su mayoría relacionados con el procesamiento secuencial o la búsqueda secuencial de elementos en estructuras de datos. Así mismo, este autor identifica y analiza los cuatro componentes fundamentales de la estructura de un bucle: flujo de control, condición de terminación, inicialización y actualización de las variables de control. Finalmente, ofrece algunas recomendaciones a los diseñadores de los lenguajes de programación sobre cómo mejorar esta estructura de control.

Waters [14] presenta una propuesta para concebir bucles a partir de cuatro métodos de construcción de planes: *augmentation*, *filtering*, *basic loop* e *interleaving*. La idea de Waters es construir un bucle a partir del análisis y descomposición del problema a resolver. A continuación, las partes son combinadas utilizando uno o más planes para obtener el plan final. Esta combinación de fragmentos de bucle se puede entender en términos de su secuencia temporal de estados. Un plan abstracto permite aplicar el método en cualquier lenguaje de programación. El método propuesto por el autor también se puede utilizar en el análisis, comprensión y demostración de la corrección de un bucle.

En un estudio expuesto por Soloway et al. [13], se prueba la relación entre las construcciones del lenguaje de programación y la estrategia cognitiva preferida de los programadores. Los autores identifican dos estrategias de recorrido: procesar/leer y leer/procesar. Los resultados del estudio mostraron que los estudiantes encuentran más fácil construir bucles correctamente cuando el lenguaje de programación proporciona el marco adecuado para elegir su estrategia preferida. East et al. [5] presenta un modelo de aprendizaje basado en patrones. Los autores afirman que los programadores cuando abordan la resolución de problemas, comparan cada nueva situación con experiencias anteriores. En este trabajo se identificaron cinco patrones para desarrollar pequeños programas: *process-one-item*, *guarded-action*, *alternative-actions*, *process-all-items* y *process-items-until-done*.

En [12], Shackelford y Badre informan sobre los resultados de un estudio comparativo en relación al uso de varios patrones de bucle para construir programas en Pascal. Se dividieron los estudiantes en dos grupos. Cada grupo recibió un conjunto de patrones de bucle, escritos en notación algorítmica, utilizando bucles *for*, *while* y *repeat*. A un grupo de estudiantes se le entregó, junto a los patrones, una serie de reglas descriptivas sobre el número de iteraciones del bucle. Por ejemplo: "R1. *Si se conoce el número de iteraciones del bucle, utiliza un bucle for*". El otro grupo de estudiantes recibió reglas constructivas basadas en el cálculo de la variable de control del bucle. Por ejemplo: "R1. *Si el valor de la variable de control sólo cuenta el número de iteraciones, utiliza un bucle for*". los estudiantes que recibieron las reglas constructivas obtuvieron mejores resultados.

En [2], Astrachan y Wallingford describen algunos patrones de bucle básicos. Estos autores clasifican los patrones de bucle según diferentes tipos de recorrido y búsqueda. Proulx [10] presenta un curso introductorio de informática basado en programación elemental y 11 colecciones de patrones de diseño. *name use*, *reading data*, *read-process-write*, *encapsulation*, *repetition*, *selection*, *traversal*, *cumulative result*, *conversion*, *indirect reference* y *other patterns*. El trabajo incluye una breve descripción textual de cada patrón. Zant [16] presenta un enfoque basado en la técnica *Action Table*, para estudiantes de programación noveles, que transforma un enunciado de un problema a un diseño de programa. Para aplicar la técnica se emplearon dos patrones de bucle: *Input-Process-Output* e *Initialization-Loop-Termination*.

Existen pocas herramientas que soporten aprendizaje basado patrones de bucle. El sistema PROUST [7] utiliza patrones para enseñar programación en Pascal. Los patrones de programación se utilizan internamente y por tanto los estudiantes no pueden acceder a ellos. En [4] se presenta una herramienta llamada PROPAT para aprender a programar patrones pedagógicos en C. El sistema PROPAT permite a los profesores añadir y eliminar tanto patrones como problemas en la herramienta. Los estudiantes pueden resolver los ejercicios propuestos utilizandos los patrones previamente introducidos por el profesor. Nosotros tomamos como punto de partida estos trabajos y vamos un paso más allá al permitir a los usuarios seleccionar el lenguaje de programación utilizado para implementar los patrones de bucle.

En todos los artículos citados, los patrones de bucle se escriben en un modelo de acceso secuencial particular. Cada patrón se enseña como un receta, sin un proceso de descubrimiento y reflexión. En nuestro enfoque, se ofrece material instructivo para enseñar a los estudiantes a tratar la variabilidad de los esquemas algorítmicos según las instrucciones "desenrrolladas" del bucle, tales como el procesamiento de un elemento, la verificación de cierta propiedad o la lectrua de un nuevo elemento de la secuencia. El objetivo es alcanzar un aprendizaje activo basado en la exploración donde los estudiantes creen sus propios modelos mentales en lugar de permanecer como sujetos pasivos que simple-

mente reciben y almacenan el conocimiento trasmitido por el profesor.

3. SAMTool

El sistema SAMTool es una aplicación cliente-servidor que se implementó utilizando JSP/Servlets. En el cliente se construyen consultas y se visualizan los correspondientes resultados utilizando JavaScript y AJAX. El servidor web Tomcat que soporta servlets y JSPs, se utiliza para proporcionar servicios HTTP. Los datos se almacenan y se recuperan de memoria mediante Hibernate utilizando una base de datos MySQL. Ya existe un prototipo del entorno de SAMTool disponible en http://dis.um.es:8180/SAMTool.

SAMTool se emplea como una herramienta web para deducir e implementar algoritmos iterativos. Para ilustrar la propuesta, esta sección describe cómo interactúan los estudiantes con la herramienta para obtener esquemas algorítmicos de recorrido en un modelo de acceso secuencial, y generar código a partir de un problema, un esquema algorítmico, un lenguaje de programación y un tipo de secuencia en el lenguaje de programación elegido. Las actividades docentes propuestas en un curso de introducción a la programación que utilice SAMTool pueden diseñarse para cubrir el dominio cognitivo de la taxonomía de Bloom [3]. El Cuadro 1 muestra una posible correspondencia entre estas categorías y algunas de las actividades educativas propuestas con SAMTool. La herramienta, por tanto, ayuda a los estudiantes a alcanzar estos principios pedagógicos.

SAMTool tiene una interfaz web diferente para usuarios (estudiantes) y administradores del sistema (profesores). Por ejemplo, un estudiante puede construir esquemas algorítmicos y generar programas. Por el contrario, un profesor puede introducir nuevos lenguajes de programación, incluir sus construcciones de bucle e instrucciones condicionales, así como crear y asociar tipos de secuencia a un lenguaje de programación. Tanto estudiantes como profesores pueden seleccionar el lenguaje de la interfaz (actualmente inglés o español).

Para construir un programa en SAMTool se siguen los siguientes pasos:

1. Configurar SAMTool. Introducir las construcciones de control y las instrucciones condicionales de los lenguajes de programación utilizados. El profesor también introduce varios tipos de secuencia con sus operaciones (perfil del profesor).
2. Resolver el ejercicio propuesto (problema iterativo) utilizando razonamiento inductivo. Los estudiantes eligen un tipo de secuencia *S* en el lenguaje de programación *L* utilizado en los laboratorios de prácticas. A continuación, los estudiantes escriben por sí mismos los tratamientos del caso base y del caso general en *L*, e introducen el código resultante en SAMTool (perfil del estudiante).
3. Crear un esquema algorítmico de acuerdo al modelo de acceso secuencial asociado a *S* empleando SAMTool (perfil del estudiante). El esquema algorítmico resultante que puede almacenarse en la base de datos de SAMTool, se muestra en notación algoritmica.
4. Producir un programa. SAMTool permite a los estudiantes implementar automáticamente una solución a cualquier problema propuesto *instanciando* y *transformando* el esquema algorítmico (obtenido en el Paso 3) de notación algorítmica a *L*. El código (en *L*) independiente y dependiente del problema, se genera a partir de los datos obtenidos en los Pasos 1 y 2, respectivamente.

Cuadro 1: Actividades educativas en el nivel cognitivo de la Taxonomía de Bloom

Categoría y actividades educativas
Conocimiento. Memorizar conceptos tales como tipo, variable, constante, algoritmo, esquema algorítmico.
Comprensión. Entender la técnica empleada para obtener esquemas algorítmicos de recorrido y búsqueda de acuerdo a un modelo de acceso secuencial.
Aplicación. Traducir un algoritmo escrito en notación algorítmica a un lenguaje de programación.
Análisis. Utilizar razonamiento inductivo para abordar la complejidad de un problema algorítmico iterativo.
Síntesis. Implementar esquemas iterativos a partir de cuatro elementos: inicialización, terminación, tratamiento en el cuerpo del bucle y tratamiento final.
Evaluación. Elegir el mejor esquema algorítmico para un problema según el número de elementos que deben tratarse fuera del bucle.

Para ilustrar la resolución de problemas algorítmicos utilizando SAMTool, vamos a considerar el siguiente problema: "*Problema 1. A partir de una secuencia de enteros representada mediante una lista lineal enlazada con memoria dinámica en C, contar el número de enteros mayores que 0*".

3.1. Preparando el camino para implementar patrones de bucle (Paso 1)

Uno de los puntos fuertes de SAMTool es su flexibilidad. La herramienta se puede adaptar a varios lenguajes de programación según las necesidades en las prácticas de laboratorio. Desde el punto de vista del administrador, los profesores pueden introducir estructuras de control e instrucciones condicionales de cualquier lenguaje de programación con la única restricción de que la estructura de estas instrucciones esté definida en términos de una gramática abstracta común. La Figura 1 muestra el interfaz del profesor utilizado para insertar construcciones de bucle, en particular, se observa la inserción de un bucle *while* en C.

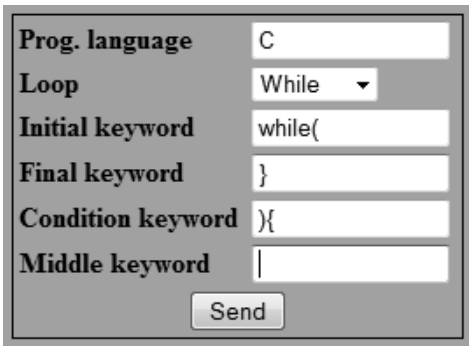

Figura 1: Bucle *while* en C.

El concepto de secuencia es fundamental en un curso introductorio de programación. Una secuencia se puede encontrar en los valores que toman las variables (secuencias implícitas), el *buffer* del teclado o en tipos de datos como archivos y cadenas. Además, las secuencias se pueden representar mediante otros tipos de datos como *arrays*, o implementarse mediante punteros.

Para acceder a una secuencia, se utilizan dos operaciones. Cuando la secuencia es un tipo de datos, estas operaciones pertenecen a la interfaz del tipo de datos. La primera operación se utiliza para comenzar el acceso secuencial. Según el modelo subyacente, podemos encontrar a su vez dos operaciones: (1) *Start* que obtiene el primer elemento de la secuencia siempre que la secuencia no esté vacía; y (2) *Initiate* que prepara el acceso secuencial pero no obtiene ningún elemento. La segunda operación de acceso, denominada *Advance* en la herramienta, se utiliza para continuar el recorrido secuencial. Su ejecución permite acceder al siguiente elemento de la secuencia. De acuerdo al modelo subyacente, después de ejecutar una operación *Advance*, el final de la secuencia se detecta cuando se llega al último elemento o cuando se sobrepasa el último elemento, utilizando la función de consulta denominada *IsLast* o *IsEnd*, respectivamente. Como resultado de combinar las dos formas de iniciar y acabar el recorrido secuencial, emergen cuatro modelos de acceso secuencial [1]. Los esquemas algorítmicos de recorrido de SAMTool surgen a partir de estos cuatro modelos de control abstractos. En la mayoría de los casos, uno de los modelos anteriores estará presente en cualquier recorrido secuencial, de ahí la importancía de su aprendizaje. La primera decisión de un programador es determinar qué modelo es el más adecuado para la secuencia del problema que pretende resolver.

Un programador sólo puede utilizar los tipos de secuencia disponibles en el lenguaje de programación que esté utilizando. Por tanto, sus programas vendrán reglados por los modelos subyacentes en los tipos de datos secuenciales. El Cuadro 2) presenta algunos tipos de secuencia de los lenguajes de programación Java, C/C++, Ada, Modula y Pascal, y sus correspondientes modelos de acceso secuencial. A partir de la comparación de los lenguajes estudiados, se observa que la mayoría siguen el primer y el segundo modelo de acceso secuencial.

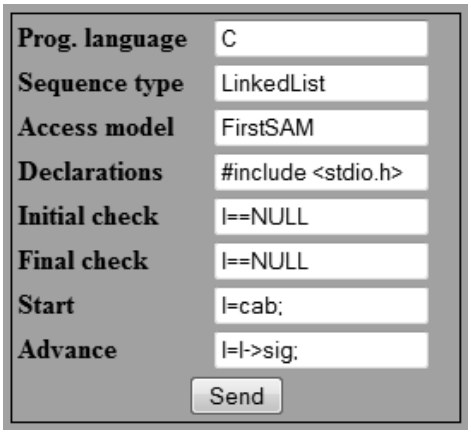

Figura 2: Lista enlazada en C.

Para completar el Paso 1, el profesor debe definir los tipos de secuencia, con sus operaciones, para cada lenguaje de programación previamente introducido en la herramienta. La Figura 2 muestra la interfaz utilizada por el profesor para añadir en SAMTool las instrucciones y declaraciones en C (Cuadro 3) requeridas para utilizar una lista enlazada con memoría dinámica (*Pro-*

Java	C/C++	Ada	Modula	Pascal
SB (M1)	Streams (M1)	F (M1)	F (M1)	F (M2)
BT (M2)	BT (M1)	BT (M3)	KT (M1)	KT (M1)

Cuadro 2: Modelos de acceso secuencial en Java, C/C++, Ada, Pascal y Modula-2. BT: *Buffer* del teclado; Mx: Modelo de acceso secuencial x; F: Ficheros; SB: StringBuffer

blema 1), según el primer modelo de acceso secuencial. El Cuadro 4 muestra la correspondencia implícita entre el modelo abstracto (encabezado *Secuencia de op.*) y las operaciones de la lista enlazada en C (encabezado *Código en C*). Obsérvese que, si se utiliza la operación *IsLast* en lugar de la operación *IsEnd*, entonces también se puede definir una lista enlazada del tercer modelo de acceso secuencial.

```
#include<stdlib>
typedef struct n {
      int value;
      struct n *sig;
} node;
typedef node *Pnode;
Pnode l, cab;
```

Cuadro 3: Representación de la secuencia S mediante punteros en C

Secuencia de op.	Código en C
CheckIsEmpty	l==NULL
CheckIsEnd	l==NULL
CheckIsLast	l->sig==NULL
Start	l=cab;
Advance	l=l->sig;
Current element	l->value

Cuadro 4: Correspondencia entre la secuencia abstracta y la secuencia concreta del primer modelo de acceso secuencial

3.2. Problemas iterativos (Paso 2)

El *Problema 1* es un problema de recorrido, esto es, su resolución requiere el uso de un esquema algorítmico que recorre y procesa todos los elementos de la secuencia de entrada. Los esquemas de recorrido secuencial pueden variar según el modelo de acceso secuencial subyacente, pero los tratamientos de cada elemento de la secuencia son independientes del modelo. En el *Problema 1*, el caso base (o tratamiento trivial) consiste en dar el valor inicial 0 a la variable *counter*, mientras que el caso general (o tratamiento no trivial) consiste en incrementar la variable *counter* en uno si el elemento actual es mayor que 0, utilizando un razonamiento inductivo. Este problema es muy sencillo, sin embargo, otros problemas como el *Segmento de mayor peso*, o la *Máxima inversión* [6], puede resolverse de manera elegante y eficiente aplicando este razonamiento. La Figura 3 muestra la interfaz que utiliza el estudiante para añadir los tratamientos inicial y general del *Problema 1* en C.

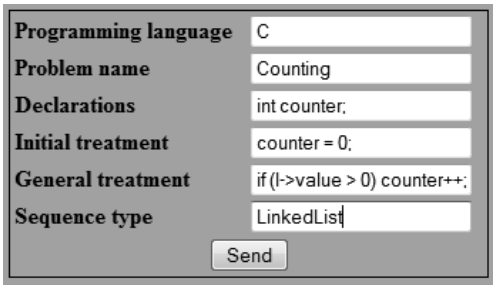

Figura 3: Código dependiente del problema.

3.3. Construir esquemas algorítmicos (Paso 3)

En SAMTool, los esquemas algorítmicos se construyen a través del perfil del estudiante. Mediante un grupo de botones de *radio* llamado "*Sequential access model*" (parte superior izquierda de la Figura 4) se elige el modelo de acceso secuencial. En algunos tipos de datos o estructuras de datos construidas en un lenguaje de programación, podrían existir coincidencias sintácticas en parejas de estas operaciones tales como (*IsEnd*, *IsEmpty*) o (*Start*, *Advance*). Por ejemplo, las operaciones *IsEnd* e *IsEmpty* corresponden con la misma operación *l==NULL* en la lista enlazada utilizada en el *Problema 1*. Estas opciones se pueden seleccionar a través de un grupo de cajas de selección llamado "*Syntactic equality*" (parte superior derecha de la Figura 4).

Para explicar a los estudiantes la construcción de un esquema algorítmico de recorrido, se utiliza el despliegue secuencial de operaciones implícitas en el bucle. En la parte inferior izquierda de la Figura 4 se muestra cómo se visualiza este despliegue en SAMTool. Las

acciones de “enrrollar” y “desenrrollar” se realizan presionando los botones “*Roll*” y “*Unroll*”, respectivamente. La acción de “enrrollar” permite a los estudiantes reducir la longitud del esquema algorítmico. Por el contrario, la acción “desenrrollar” se emplea cuando uno o más elementos de la secuencia necesariamente se deben tratar fuera del bucle. Es responsabilidad del estudiante tomar una decisión u otra para adaptar el esquema algorítmico a las necesidades del problema. En cualquier despliegue secuencial de un bucle, las operaciones se repiten a partir de un punto. Las últimas tres operaciones del despliegue mostrado por SAMTool (*Checking*, *Treatment* and *Advance* en el ejemplo de la Figura 4) se utilizan para construir el bucle del esquema algorítmico (parte inferior derecha de la Figura 4). Según sea la posición de la operación que comprueba la terminación del bucle (operación *Checking*), la herramienta elige la estructura de control correcta: while-endwhile (si la comprobación se realiza al principio), repeat-until (si la comprobación se realiza al final) o loop-exit-endloop (si la comprobación se realiza en un punto intermedio). La parte inferior derecha de la Figura 4 muestra el esquema algorítmico generado para el *Problema 1* al pinchar en el botón “*Create As*”.

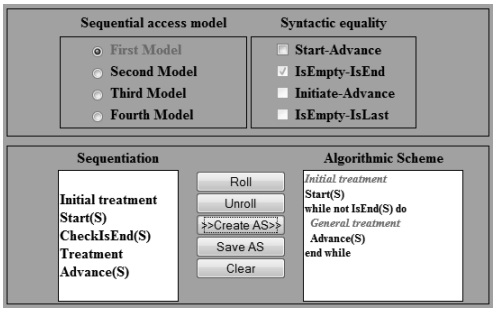

Figura 4: Generador de esquemas en SAMTool.

3.4. Generar programas (Paso 4)

Creemos que la mejor forma de aprender programación de computadores es precisamente programando. Los estudiantes no solo tienen que reflexionar sobre los problemas algorítmicos, sino también crear sus propios programas. En esta etapa, encontramos fundamentalmente dos beneficios para los estudiantes. Primero, escribir programas *instanciando* esquemas algorítmicos y adaptándolos a un contexto particular muestra a los estudiantes las ventajas de pensar sobre un problema antes de escribir código. Segundo, codificar y ejecutar programas en un ordenador anima y motiva a los estudiantes.

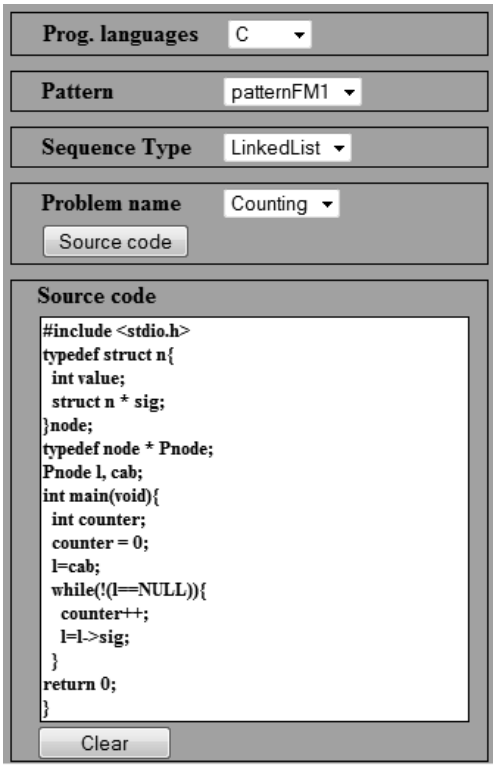

Figura 5: Generación de un programa (visión del estudiante).

La Figura 5 muestra un esquema de recorrido secuencial *instanciado* para el *Problema 1*. La solución se ha obtenido a partir de un esquema algorítmico del primer modelo de acceso secuencial (Figura 4), el código dependiente del problema (Figura 3) y el código independiente del problema (Figuras 1 y 2).

4. Conclusiones y trabajos futuros

En este trabajo hemos presentado una herramienta para diseñar bucles. SAMTool ofrece soporte para construir esquemas algorítmicos de búsqueda y recorrido, en cuatro modelos de acceso secuencial, y generar programas a partir de los esquemas iterativos adaptados por el programador para un tipo de secuencia y un lenguaje de programación concretos.

Actualmente, pretendemos extender SAMTool para incorporar nuevos esquemas algorítmicos y adaptarla a estándares para tecnologías educativas como el estándar de *e-learning* SCORM. Otra área de trabajo futuro es la retroalimentación a los estudiantes y los datos estadísticos de uso de la herramienta. La retroalimentación tiene el potencial de mejorar el aprendizaje activo en horas no lectivas. Por otra parte, los datos estadísticos recogidos en la herramienta permitirían a los docentes analizar la dificultad de los problemas y realizar un seguimiento de la evolución de los discentes.

Referencias

[1] J. L. F. Alemán. Deducing loop patterns in CS1: A comparative study. In *ICALT '09*, pages 247–248, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[2] O. Astrachan and E. Wallingford. Loop patterns. In *Proceedings of PLPC'98*, Illinois, USA, 1998.

[3] B. Bloom, E. Furst, W. Hill, and D. Krathwohl. *Taxonomy of Educational Objectives: Handbook I, The Cognitive Domain*. Addison-Wesley, 1956.

[4] L.Ñ. de Barros, A. P. dos Santos Mota, K. V. Delgado, and P. M. Matsumoto. A tool for programming learning with pedagogical patterns. In *eclipse '05: Proceedings of OOPSLA'05 workshop on eTC*, pages 125–129, USA, 2005. ACM.

[5] J. P. East, R. Thomas, E. Wallingford, W. Beck, and J. Drake. Pattern-based programming instruction. *Proceedings of the ASEE'96*, June 1996.

[6] J. García-Molina, F. Montoya-Dato, J. Fernández-Alemán, and M. Majado-Rosales. *Una Introducción a la Programación. Un Enfoque Algorítmico (in Spanish)*. Thomson, 2005.

[7] W. L. Johnson and E. Soloway. Proust: Knowledge-based program understanding. In *ICSE '84*, pages 369–380, USA, 1984. IEEE Press.

[8] O. Muller, B. Haberman, and H. Averbuch. (an almost) pedagogical pattern for pattern-based problem-solving instruction. In *ITiCSE'04*, pages 102–106, New York, NY, USA, 2004. ACM.

[9] T. W. Pratt. Control computations and the design of loop control structures. *IEEE TSE*, 4(2):81–89, 1978.

[10] V. K. Proulx. Programming patterns and design patterns in the introductory computer science course. In *SIGCSE '00*, pages 80–84, USA, 2000. ACM.

[11] A. Robins, J. Rountree, and N. Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003.

[12] R. L. Shackelford and A. Badre. Why can not smart students solve simple programming problems? *International Journal of Man-Machine Studies*, 64(38):985–997, 1997.

[13] E. Soloway, J. Bonar, and K. Ehrlich. Cognitive strategies and looping constructs: An empirical study. *Commun. ACM*, 26(11):853–860, 1983.

[14] R. C. Waters. A method for analyzing loop programs. *IEEE TSE*, 5(3):237–247, 1979.

[15] L. E. Winslow. Programming pedagogy—a psychological overview. *SIGCSE B.*, 28(3):17–22, 1996.

[16] R. F. Zant. Fundamental patterns for logic design. *Information Systems Education Journal*, 19(1), 2003.