HiLesMaker: Diseño de Especificaciones de Alto Nivel Asistida por Computador

Gerardo Muñoz
Sergio A. Rojas
Juan C. Rodríguez

RESUMEN

El lenguaje de especificación de alto nivel (HiLeS – High LEvel Specification) [1] es una herramienta para el diseño de procesos algorítmicos implementables en hardware y en software. Dado que se trata de un lenguaje visual, se debe realizar su conversión a un código ejecutable en algún dispositivo de cómputo como una celda programable, un microprocesador o un microcontrolador. En este artículo se presenta HiLeSMaker, un primer prototipo de herramienta computacional que permite la edición de diagramas HiLeS, y que tiene capacidad para generar una representación simbólica de los programas diseñados en él.

Palabras clave: Especificación de procesos, diseño de programas, lenguajes visuales

ABSTRACT

The HiLes (High Level Specification) [1] is a tool for algorithmic process design to be running in hardware and software. Since it is a visual language, it must be converted to a executable code for a computing device like a programmable cell, a microprocessor or a micro controller. In this paper we present HiLesMaker, a first prototype of a computational tool that allows for the edition of HiLes diagrams, and generation of symbolic representation of programs that had been designed on it.

Keywords: Process specification, program design, visual languages

HiLeS es una especificación gráfica para sistemas complejos en tiempo real, mediante una descripción jerárquica de la combinación de procesos continuos y procesos por eventos.

1. INTRODUCCIÓN

El lenguaje de Especificaciones de Alto Nivel (HiLeS – High Level Specification) [1] se ha propuesto como una herramienta de alto nivel mediante la cual el ingeniero puede diseñar procesos algorítmicos (secuenciales o paralelos) que puedan ser ejecutados en plataformas de hardware como microprocesadores o arreglos de celdas programables (FPGAs). Se trata de un lenguaje inspirado en las Redes de Petri [2], que además del manejo de eventos propio de este tipo de redes, adiciona un manejo de datos, unificándolos en diagramas en donde las transiciones, las operaciones, el almacenamiento y la transmisión de datos se representan de manera gráfica.

Dada su naturaleza visual, y buscando facilitar el diseño e implementación de programas en este lenguaje, se ha propuesto el desarrollo de una herramienta asistida por computador para la creación y edición de diagramas de especificaciones, que además permita generar código ejecutable en dispositivos de cómputo. En este artículo se describe a ***HiLeSMaker***, un primer prototipo desarrollado al interior de los grupos de interés ACME y HiLeS de la Universidad Distrital. El documento se organiza así: En la siguiente sección se presenta un resumen del lenguaje HiLeS, posteriormente se discuten algunos aspectos del diseño e implementación de la herramienta y por último se plantea el trabajo que sigue en el futuro inmediato.

2. UN BREVE REPASO DEL LENGUAJE HiLeS

HiLeS es una especificación gráfica para sistemas complejos (distribuidos o paralelos) en tiempo real, mediante una descripción jerárquica de la combinación de procesos continuos (flujo de datos) y procesos por eventos (red de petri). La implementación de una especificación Hiles se puede realizar en software o hardware y este último puede ser análogo o digital (sincrónico o asincrónico).

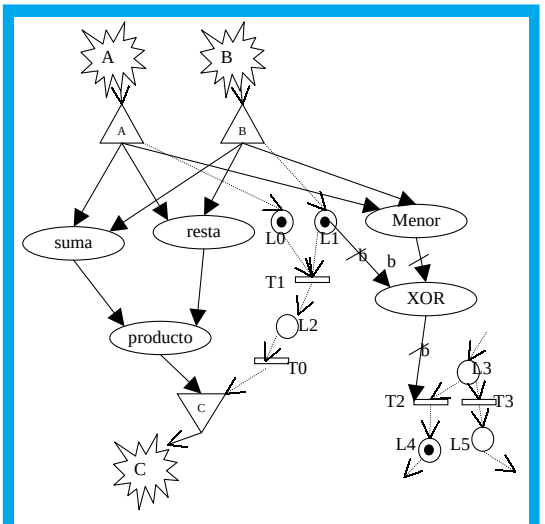

Figura 1. Ejemplo de diagrama HiLeS

En la figura 1 se observa un ejemplo de un diagrama HiLeS. La entrada y salida de información se

En principio se identificaron cuatro casos de uso de edición de diagramas en los cuales el usuario tiene la posibilidad de crear, cargar, salvar y modificar un espacio de trabajo.

representa con estrellas. La parte correspondiente a la red de petri es un grafo orientado formado por lugares (L), transiciones (T), fichas (o "tokens") y flechas de fichas. Los lugares se representan por círculos y en ellos se pueden almacenar fichas (en el caso de HiLeS se permite máximo una ficha por lugar). Las transiciones se representan líneas horizontales gruesas, y requieren para dispararse que todos los lugares que la preceden tengan ficha; por ejemplo en la figura 1, T1 se puede disparar porque L0 y L1 tienen ficha, sin embargo ni T2 ni T3 se pueden disparar, porque L3 no tiene ficha. Al dispararse una transición desaparecen las fichas de los lugares precedentes y aparecen fichas en los lugares posteriores; en el ejemplo al dispararse T1 quita las fichas de L0 y L1 y coloca una ficha en L2.Las flechas de fichas se representan con líneas punteadas y punta sin rellenar, e indican el origen y el destino del envío de una ficha.

La parte correspondiente al flujo de datos consta de ecuaciones dibujadas con óvalos y conexiones de datos representados por flechas continuas con la punta rellena. Se asume que por las flechas de datos siempre hay alguna información digital o análoga; en el caso de datos booleanos las flecha son de color verde (o están marcadas con "b"). A manera de ejemplo, obsérvese cómo en el lado izquierdo de la figura 1 está la representación de la operación C=(A+B)(A-B).

Los datos y las fichas pueden interactuar de varias formas:

- De los lugares puede salir un dato booleano que es falso si no tiene ficha y verdadero si tiene; por ejemplo en el lugar L1 hay un dato de salida.
- A las transiciones puede llegar un dato booleano, que en el caso de ser falso no permite el disparo y si es verdadero la habilita para su comportamiento normal; por ejemplo la transición T2 sólo se puede disparar si el resultado de la XOR es verdadero y hay una ficha en L3.
- El *Sampled* permite muestrear datos con la llegada de una ficha, y transmitirlo como un paquete, esto genera un nuevo tipo de flecha que agrupa las dos anteriores y se denomina flecha de canal. El *Sampled* se dibuja con un triángulo invertido. En el ejemplo de la figura 1, el *sampled* de C muestrea el valor de la salida del producto cuando se dispara T0.
- El *Hold* permite descomponer la flecha de canal nuevamente en la flecha de dato y en la flecha de ficha, en este caso la flecha de dato se convierte en un ZOH (Zero Order Hold) manteniendo el mismo valor del dato que llegó, hasta que llegue un nuevo paquete. El *hold* se representa con un triángulo sin invertir. En la figura los *Holds* de A y de B mantienen los respectivos valores mientras se realizan las operaciones.

En la figura 2 se puede ver los tipos de flechas utilizados en HiLeS y como interactúan en un *Sampled* y en un *Hold*.

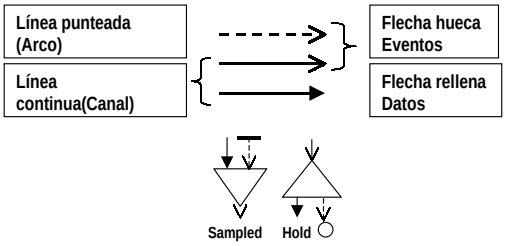

Figura 2. Tipos de flechas usadas en HiLeS

Las ecuaciones y la red de petri pueden ser agrupadas en bloques que permiten trabajar diferentes niveles de jerarquía, permitiendo modularizar los diseños; dichos bloques se grafican como rectángulos. De esta forma todo el proceso descrito en un diagrama HiLeS puede encapsularse en un bloque para ser utilizado en un diseño más complejo.

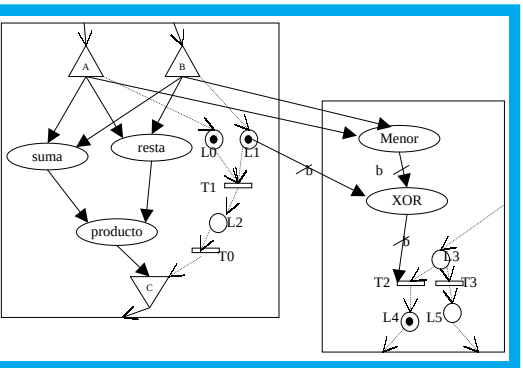

Figura 3. Bloques jerárquicos en HiLeS

3. DISEÑO DEL EDITOR DE HiLeS

El editor de HiLeS fue diseñado con UML [4], un lenguaje de modelado de software ampliamente utilizado por la industria. En principio se identificaron cuatro casos de uso de edición de diagramas (ver Figura 4), en los cuales el usuario tiene la posibilidad de crear, cargar, salvar y modificar un espacio de trabajo.

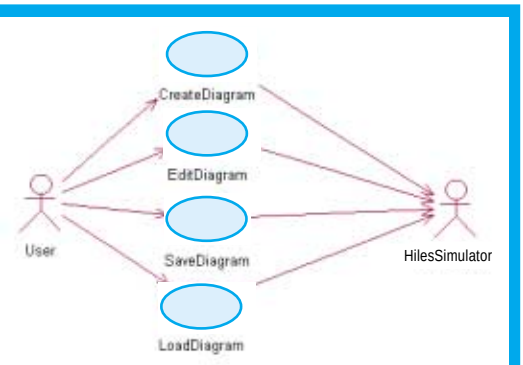

Fig. 4. Casos de uso del editor de especificaciones HiLeS

El espacio de trabajo es la hoja donde el usuario podrá dibujar, mover y borrar elementos HiLeS del diagrama de especificación de un proceso. Cada uno

de estos elementos ha sido modelado como objetos en la jerarquía de clases que se puede observar en la Fig. 5.

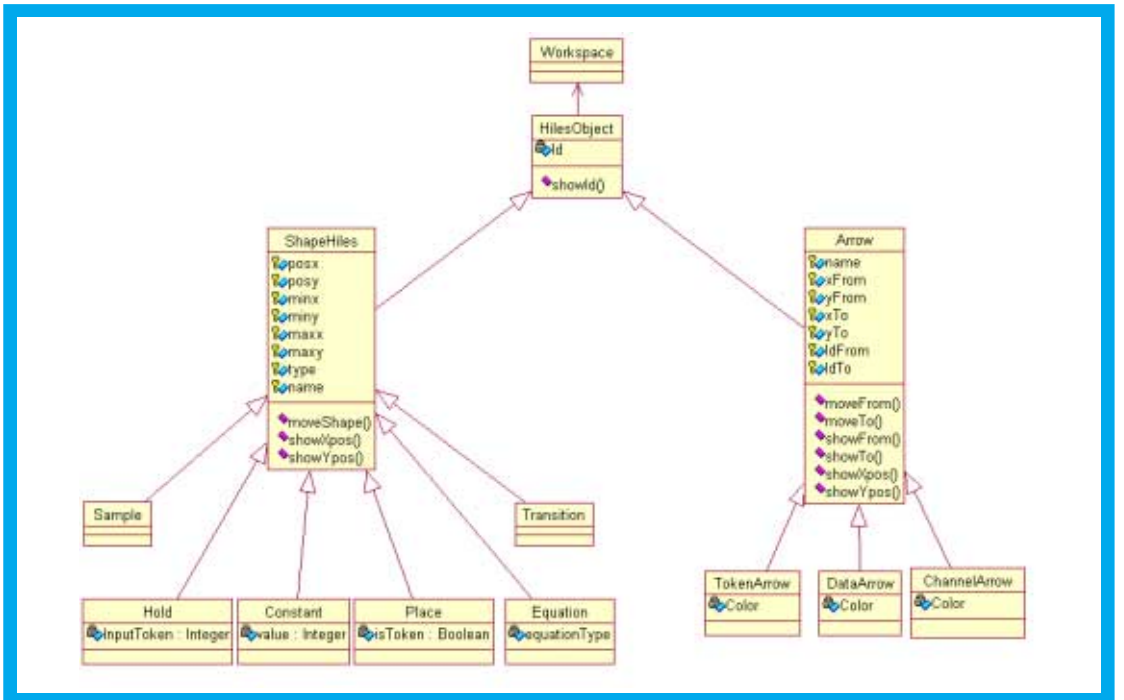

Figura 5. Diagrama de clases del editor de diagramas HiLeS

El usuario puede manipular dos clases de objetos principales: las formas y las flechas. Por una parte está el objeto Forma (*Form*) que tiene una ubicación en la pantalla y un código que lo identifica. De este objeto base se derivan las diferentes clases necesarias para representar los nodos HiLeS que se enumeran a continuación:

- *Place*: Un lugar que permite almacenar hasta una ficha. Recibe y envía flechas de fichas, también puede enviar una flecha de dato booleano.
- *Transition*: Este objeto permite el disparo de las fichas. Recibe y envía flechas de fichas, también puede recibir una flecha de dato booleano.
- *Equation*: Es un objeto que se utiliza para indicar una serie de operaciones que se ejecutan sobre un conjunto de señales de entrada, para generar una salida. Aunque inicialmente está pensado para albergar cualquier tipo de operación, en el diseño actual se ha limitado a 16 operaciones básicas. Sólo puede recibir y enviar flechas de datos.
- *Constant*: Es un caso particular de *Equation*, en el que no existen entradas. Genera una salida de valor constante. En la actualidad solamente soporta datos de tipo entero.
- *Sampled*: Es un objeto que captura una muestra de una flecha de datos cuando llega una ficha, empaqueta el dato y lo envía por una flecha de canal.
- *Hold*: Es el objeto opuesto al *Sampled*, el cual recibe una flecha de canal y envía una flecha de ficha y una flecha de dato. En la flecha de dato congela la última información que recibió por la flecha de canal.

Por otra parte, el objeto Flecha (*Arrow*) determina las conexiones entre las formas, es decir, es el encargado de transmitir información entre dos nodos del diagrama; este objeto almacena los identificadores de los elementos que está conectando. De acuerdo con el tipo de señal que transmitan, se deriva en las siguientes clases:

- *TokenArrow*: Es un tipo de flecha en el cual solo viajan fichas; representa los arcos de conexión de una red de petri. Este objeto permite conectar *Place*, *Transition*, *Sampled*, *Hold*.
- *DataArrow*: Es el tipo de flecha que representa datos enteros de un byte; determina las conexiones entre objetos *Equation*, *Sample*, y *Hold*.
- *ChannelArrow*: Esta flecha es la unión entre la conexión *DataArrow* y *TokenArrow*, es decir, empaqueta un dato con una ficha. Usualmente se utiliza para determinar canales de comunicación.

Cuando el usuario selecciona el tipo de objeto que desea utilizar y lo ubica en el área de trabajo, se modifica la matriz y se asignan las propiedades.

Para el manejo de conexiones entre objetos y para su movimiento en la hoja de trabajo, se diseñó una matriz de 64 x 48 posiciones que se actualiza siempre que se coloca un nuevo objeto Hiles o cuando cambia la posición de uno existente. Cuando el usuario selecciona el tipo de objeto que desea utilizar y lo ubica en el área de trabajo, se modifica la matriz y se asignan las propiedades. Cuando el usuario desea colocar una flecha, primero tiene que ubicarse en un espacio donde haya un objeto, y posteriormente tie-

La investigación continuará hacia convertir a HiLeSMaker en una herramienta asistida por el computador para especificar procesos de software y hardware, algo así como un "CASE inteligente" para procesos.

ne que arrastrar la flecha hasta otro objeto ubicado en el espacio de trabajo (de lo contrario se invalida la acción). Una vez finalizado el diagrama, la herramienta automáticamente genera la matriz de conexiones para ser implementada en el microcontrolador.

4. PRESENTACIÓN DE HiLeSMaker

HiLeSMaker es un producto de software desarrollado en lenguaje Java, basándose en el diseño descrito en la sección anterior, utilizando el compilador Visual J++[4]. Puesto que se ejecuta sobre una máquina virtual para Web, posee una amplia portabilidad y facilidad de acceso desde cualquier terminal destinada a la especificación de procesos.

La interacción con el usuario que desea diseñar un diagrama HiLeS, se realiza mediante una interfaz gráfica en la cual con el ratón se selecciona el objeto que se desea crear y se coloca en la hoja de trabajo. De manera similar se pueden realizar conexiones entre los objetos (cuya validez será controlada por la herramienta). Además, HiLeSMaker cuenta con facilidades de edición (para mover, eliminar y modificar objetos) y para almacenamiento y recuperación de diagramas HiLeS. En la figura 6 se puede observar la interfaz del programa HiLeSMaker, junto con un diagrama simple desarrollado en él, en donde se especifica un proceso que transmite una dato cuando su bit menos significativo (LSB) es cero.

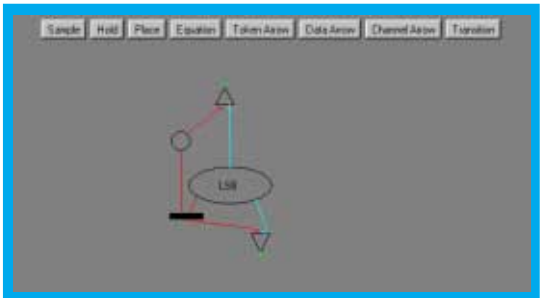

Figura 6. Interfaz gráfica de HiLeSMaker

Una vez el usuario finaliza el diagrama, puede generar una representación simbólica del proceso que ha especificado. Este código puede ser posteriormente implementado, por ejemplo, en un microcontrolador. Concretamente, HiLeS genera una matriz de comandos que pueden ser almacenados en la memoria del dispositivo y ser ejecutados de acuerdo con un programa "maestro" que revise cada una de las instrucciones generadas. Cada operación representada en el diagrama, tiene un código o comando predeterminado. En realidad estos códigos corresponden a programas (subrutinas) en código máquina elaborados de antemano, razón por la cual podrán ser ejecutados por el programa "maestro" simplemente al invocarlos.

5. CONCLUSIONES Y TRABAJO FUTURO

Utilizando el modelamiento UML y la programación orientada por objetos en Java, se ha desarrollado una herramienta visual para la creación de especificaciones de procesos en alto nivel. Aparte de las ventajas obvias de permitir almacenar, recuperar, modificar y corregir este tipo de diagramas, que de por sí son una gran utilidad para el ingeniero diseñador, HiLeSMaker ofrece la facilidad de aprovechar el computador para automatizar la implementación del proceso en una plataforma de hardware, esto es, programación en lenguaje máquina. Se planea que en futuras versiones, adicionalmente permita simular el proceso, de tal manera que el diseñador pueda realizar pruebas y correcciones antes de hacer la implementación.

Aunque la técnica de generación de código utilizada simplifica un poco el alcance del lenguaje HiLeS (por ejemplo, solo se pemiten 16 operaciones en el objeto *Equation*) se plantea también la posibilidad de más adelante generar directamente el programa máquina que implemente el proceso. Igualmente, se estudiará la forma en que la herramienta pueda hacer un análisis del diagrama para evaluar la viabilidad de su implementación en paralelo, y que automáticamente distribuya la carga en varios procesadores, inclusive en plataformas inalámbricas. En ese sentido, la investigación continuará hacia convertir a HiLeSMaker en una herramienta asistida por el computador para especificar procesos de software y hardware, algo así como un "CASE inteligente" para procesos.

REFERENCIAS

[1] MUÑOZ, G; JIMÉNEZ, F; GARCÍA, A; et al, **Tools and Models for Systems Design and Synthesis of MEMS Based on Asynchronous Circuits**, Jaico Piblishing House, 2000.

[2] DAVID, R.;ALLA, H. **Du grafcet aux Reseaux de Petri**. Hermes, 1989.

[3] JIMENEZ, F.; **Redes de Petri**, Material de Curso Control Secuencial, Universidad de los Andes, Facultad de Ingeniería, Departamento de Eléctrica.

[4] BOOCH, G.; JACOBSON, I. **Lenguaje Unificado de Modelado de Objetos**. Addison Wesley. 2000.

[5] **Visual J++ 6.0 Reference Library.** Varios. Microsoft Press. 1999.

Gerardo Muñoz
Ing. Electrónico, U. Antonio Nariño. Magister en Ingeniería Electrónica y de Computación, U. de los Andes. Profesor/investigador de la Facultad de Ingeniería, Universidad Distrital FJC. Director del Grupo HiLeS. ***gerar@atenea.udistrital.edu.co***

Sergio Andrés Rojas Galeano
Ing. de Sistemas, U. Nacional de Colombia. Especialista en Ing. de Software, U. Distrital. Profesor/investigador de la Facultad de Ingeniería, Universidad Distrital FJC. Director del Grupo de Interés en Adaptación, Computación & MEnte (ACME). ***srojas@udistrital.edu.co***

Juan Carlos Rodríguez-Paez
Estudiante de Ingeniería de Sistemas, Universidad Distrital. Integrante del Grupo de Interés en Adaptación, Computación & MEnte (ACME). ***jurodri@tutopia.com***