

# Extensión del Lenguaje y Modelo Simplesem con Soporte para Paralelismo

Lucas Leandro Diez de Medina Quintar<sup>1</sup>, Gustavo Wolfmann<sup>2</sup>, Orlando Micolini<sup>3</sup>

<sup>1</sup> Facultad de Ciencias Exactas, Físicas y Naturales, Córdoba, Argentina

<sup>2</sup> Facultad de Ciencias Exactas, Físicas y Naturales, Córdoba, Argentina

<sup>3</sup> Facultad de Ciencias Exactas, Físicas y Naturales, Córdoba, Argentina {lucaslt89, gwolfmann}@gmail.com, [omicolini@compuar.com](mailto:omicolini@compuar.com)

## Resumen

El modelo de ejecución planteado por Carlo Ghezzi y Mehdi Jazayeri, conocido como Simplesem, no contempla lenguajes con instrucciones de ejecución paralela. El objetivo es extender esta herramienta educativa incorporando al modelo la existencia de múltiples procesadores primitivos de lenguaje que permitan expresar conceptos básicos de paralelismo. Para ello, se desarrolló una herramienta que permite analizar de manera gráfica y sencilla el impacto de programas multihilo sobre instrucciones de bajo nivel, que operan directamente sobre la memoria compartida de una máquina virtual. Esta extensión permite representar la semántica operacional de lenguajes paralelos que requieren procesadores multihilo, comunes en la actualidad. Se pretende distribuir esta herramienta a la comunidad educativa, de manera tal que puedan realizarse estudios sobre los beneficios obtenidos al aplicarla durante el proceso de aprendizaje de los lenguajes de programación.

*Palabras clave:* Simplesem, Paralelismo, Modelo de ejecución, Sintaxis, Semántica, Lenguajes de Programación.

## Abstract

The execution model proposed by Carlo Ghezzi and Mehdi Jazayeri, known as Simplesem, does not cover languages with parallel execution instructions. The aim is to extend this educational tool model incorporating the existence of multiple processors primitive language that can express basic concepts of parallelism. For this, a tool to analyze graphical and easily the impact of multithreaded programs on low-level instructions that operate directly on the shared memory of a virtual machine was developed. This extension allows to represent the operational semantics of parallel languages that require multi-threaded processors, very common now a days. This tool is intended to be distributed to the educational community, so that

studies on the benefits obtained when applied during the process of learning programming languages can be made.

*Key Words:* Simplesem, Parallelism, Execution Model, Syntax, Semantics, Programming Languages.

## 1. Introducción

Las dificultades a las que se enfrenta un alumno al estudiar el proceso que atraviesa un programa escrito en un lenguaje de alto nivel, hasta poder ser ejecutado por el procesador, se deben principalmente a dos motivos: La complejidad de los lenguajes de programación de alto nivel existentes, y la del hardware sobre el cual se ejecutan.

En su libro “Programming Language Concepts”, Carlo Ghezzi y Mehdi Jazayeri [1] propusieron una alternativa para solucionar estos problemas. En primer lugar, plantearon un modelo de ejecución que consiste en una máquina formada por un procesador simple, una memoria de código y una memoria de datos. El procesador es capaz de ejecutar cuatro instrucciones elementales que conforman las primitivas del modelo Simplesem. En segundo lugar, definieron una serie de lenguajes de programación de alto nivel, de complejidad creciente, que coinciden con diversas categorizaciones de lenguajes. A partir de estas dos propuestas, lograron simplificar el proceso de aprendizaje de la relación entre los lenguajes de programación, y los recursos de la computadora.

Las memorias de datos y de código de la máquina Simplesem están formadas por celdas. En la memoria de código, las celdas almacenan instrucciones, y se utiliza un puntero para seleccionar la celda que contiene la siguiente instrucción a ejecutar. En la memoria de datos se almacena un valor por cada celda. Las operaciones se realizan directamente sobre la memoria de datos (no hay registros intermedios). Existen algunas celdas en las que se almacenan valores específicos utilizados para el funcionamiento de la máquina.

Con respecto a las instrucciones, el modelo Simplesem consta de tres instrucciones compuestas (**set**, **jump** y **jump**), y una instrucción especial para indicar la finalización del programa (**halt**). Por instrucciones compuestas se entiende aquellas instrucciones que pueden contener expresiones, y el resultado de dichas expresiones será lo que se utilice como operandos de la instrucción.

Las primitivas del modelo Simplesem permiten implementar la semántica operacional de un conjunto de lenguajes de alto nivel. Ghezzi y Jazayeri definieron lenguajes de alto nivel para demostrar el funcionamiento del modelo Simplesem.

El primer lenguaje, conocido como C1, sólo posee tipos de datos simples, e instrucciones simples (no tiene soporte para funciones). Sólo pueden utilizarse tipos de datos que permitan determinar los requerimientos de memoria de manera estática, como enteros, arreglos de tamaño fijo y estructuras. El segundo lenguaje, conocido como C2, incorpora la posibilidad declarar rutinas en el programa, que pueden contar a su vez con variables locales. Las rutinas no soportan llamadas recursivas, no pueden recibir parámetros, ni devolver valores de retorno. El tercer lenguaje, C3, agrega la posibilidad de que las rutinas devuelvan un valor, y pueden ser llamadas recursivamente.

El lenguaje de alto nivel utilizado en este trabajo es una extensión del lenguaje C3, en la cual se incorporan primitivas de ejecución paralela, y se demuestra cómo puede implementarse la semántica operacional de este nuevo lenguaje, con primitivas del modelo Simplesem. La primer sección del Artículo debe ser una Introducción que contendrá el análisis general de la problemática y las referencias principales.

### 1.1. Contribución

El presente trabajo se basó en extender el modelo de Ghezzi y Jazayeri (que sólo estudiaba lenguajes monohilo), para abarcar también lenguajes con primitivas de ejecución paralela. La principal motivación fue la predominancia de los procesadores multicore, y la importancia que ha adquirido la programación concurrente actualmente. La propuesta consiste en la definición de un nuevo lenguaje de alto nivel, y la extensión del conjunto de instrucciones Simplesem para implementar la semántica operacional de dicho lenguaje. Se desarrolló también una herramienta que representa una máquina Simplesem, con la cual se podrá escribir un programa en lenguaje de alto nivel, compilarlo a instrucciones Simplesem, y ejecutar dichas instrucciones.

El proceso de desarrollo constó de tres etapas. La primera consistió en la extensión del modelo básico monohilo, a un modelo multihilo. En la segunda etapa

se definieron las instrucciones básicas simplesem que permiten la ejecución de programas paralelos. Finalmente, se definió la gramática de un lenguaje de alto nivel, con primitivas de paralelismo, cuya semántica operacional puede ser implementada con la extensión del modelo Simplesem.

### 1.2. Importancia

Con la herramienta lograda al finalizar este proyecto, se ampliará el campo de estudio del modelo Simplesem a lenguajes multihilo de ejecución paralela, de manera tal que aprovechando la simplicidad del modelo, podrán estudiarse conceptos más complejos inherentes a la programación concurrente.

En el presente trabajo no se realizó un estudio pedagógico sobre los beneficios de utilizar el modelo Simplesem para la enseñanza de los lenguajes de programación, sino que se desarrolló la herramienta con la cual se podrán realizar dichos estudios. Nuestro principal objetivo es distribuir el presente trabajo a la comunidad educativa, dejando como trabajo a futuro el estudio pedagógico sobre los beneficios de aplicarla.

## 2. Desarrollo e Implementación

La utilización del modelo Simplesem en el ámbito educativo ha demostrado tener buenos resultados en el proceso de aprendizaje de los lenguajes de programación de alto nivel. Sin embargo, dicha propuesta se limita sólo a lenguajes secuenciales, con un único hilo de ejecución. Proponemos ahora una extensión del modelo para aplicarlo a la enseñanza de lenguajes multihilo.

### 2.1. Procesador Multihilo

La primera extensión se realiza sobre el diseño del procesador de la máquina Simplesem. Podrán crearse hasta cuatro hilos paralelos. Las memorias de código de todos los hilos contendrán las mismas instrucciones, por lo que el comportamiento específico deberá determinarse de acuerdo al identificador de cada hilo, de manera tal que cada uno ejecute sólo las instrucciones que le corresponden. La memoria de datos será compartida por todos los hilos. Cada hilo tendrá su propio puntero a instrucción.

### 2.2. Nuevas Instrucciones Simplesem

Para realizar una extensión a lenguajes paralelos, se agregaron nuevas primitivas al modelo Simplesem.

**processors n:** Esta instrucción provoca que se creen n hilos, cada uno con su propio IP, y con una memoria

de código independiente del resto. Al momento de su creación, todos los hilos estarán formados por las mismas instrucciones, y será el programador quien deba dotar de un comportamiento distinto a cada hilo.

**barrier:** Cada vez que el procesador deba ejecutar la instrucción *barrier* de un hilo, deberá verificar si todos los otros hilos están en la misma instrucción. De no ser así, la ejecución deberá bloquearse hasta que todos los hilos estén en la misma instrucción.

**wait n:** Esta instrucción es utilizada para lograr la implementación de semáforos a través de instrucciones Simplesem. Cada vez que una instrucción *wait* se ejecuta sobre la celda *n* de la memoria de datos, si el valor almacenado en la celda *n* es 0 la ejecución del hilo se bloquea. De no ser así, se decrementa el valor de *n* en 1, y se continua con la ejecución.

**numHilo:** El indicador *numHilo* se utiliza como palabra clave dentro de las instrucciones Simplesem para referirnos al hilo que actualmente se está ejecutando.

### 2.3. Lenguaje C3P - Extensión del Lenguaje C3

El enfoque abordado consiste en utilizar una variable paralela, que será distinta para cada hilo que se ejecute. El resto de las variables, tanto globales como locales, son compartidas por todos los hilos.

A continuación se presentan las incorporaciones que se hicieron al lenguaje C3, para generar un nuevo lenguaje al que llamamos C3P, con primitivas de ejecución en paralelo.

**Sentencia *par\_for*:** Suponiendo que *i* es la variable paralela (distinta en todos los hilos), la estructura de esta sentencia es:

```
par_for 4 (i = 0; i < 8; i++){
    suma = suma + i;
}
end_par_for;
```

El número 4 representa la cantidad de hilos que quieren crearse. El límite del ciclo *for* (en este caso 8) debe ser un número divisible por N (4), de manera tal que cada hilo procesará  $8 / 4 = 2$  valores de *i* y los sumará al resultado final. Las instrucciones que se encuentren dentro del bloque *par\_for* serán ejecutadas por los cuatro hilos.

**Sentencia *barrier*:** Dentro de un bloque *par\_for*, necesitamos un mecanismo para sincronizar todos los hilos. Cada vez que un hilo encuentra una sentencia *barrier*, este se bloquea hasta que todos los otros hilos lleguen a la misma sentencia. En ese momento todos los hilos se desbloquean y pueden continuar su

ejecución.

Si colocamos una sentencia *barrier* debajo de la sentencia de suma del ejemplo anterior, todos los hilos terminarán de procesar el primer valor de *i* antes de poder procesar el segundo valor.

**Sentencia *wait*:** Para abordar problemas típicos de concurrencia (en los cuales se utilizan múltiples hilos), debemos contar con construcciones como semáforos o monitores. Para ello, se incorporó la sentencia *wait* al lenguaje C3P.

Esta sentencia se ejecuta sobre una variable, por ejemplo *wait(i)*. Si el valor de la variable es mayor que cero, se disminuye en uno su valor y se incrementa en uno el contador de programa del hilo que ejecutó la instrucción *wait*. Si el valor de la variable es 0, el contador de programa no se modifica.

**Sentencia *notify*:** La sintaxis de esta sentencia es *notify(variable)*. Fue incorporada debido a que habitualmente se utiliza para incrementar en uno los permisos de un semáforo. El efecto de la sentencia *notify(i)* es el mismo que el de la sentencia *i++*. Si algún hilo se encontraba bloqueado por una instrucción *wait* sobre la variable *i*, al haberse incrementado el valor de esta última, dicho hilo se desbloqueará.

**Funciones con parámetros:** Otra de las funcionalidades con las que no contaba el lenguaje C3, eran funciones que recibieran parámetros.

El lenguaje C3P incorpora la posibilidad de declarar funciones que reciban múltiples parámetros. Los parámetros se guardan en el registro de activación al llamar a una función, antes de las variables locales de la misma. Los parámetros pueden utilizarse dentro de una función como si fueran variables locales.

**Restricciones del Lenguaje C3P:** Para simplificar el proceso de compilación del lenguaje C3P, se realizaron dos simplificaciones respecto al lenguaje C3. En primer lugar, todas las variables son de tipo entero, ya que soportar otros tipos de datos implicaba que el compilador realizara una detección del tipo de datos, para realizar la reserva de memoria en función de esto.

La segunda simplificación fue con respecto a las funciones. El lenguaje C3P sólo soporta la declaración y utilización de una única función. Esto se debe a que incorporar más de una función implica realizar un manejo de la tabla de símbolos del compilador, para conocer la ubicación en memoria de las instrucciones y el tamaño del registro de activación de cada función.

### 2.4. Implementación

La herramienta desarrollada se llevó a cabo en tres etapas: un proceso de formalización del lenguaje C3P, la implementación del compilador y la implementación de la máquina de ejecución abstracta (intérprete) de las

instrucciones Simplesem.

## 2.5. Formalización del Lenguaje C3P

La primera etapa consiste en extender el lenguaje de alto nivel agregando sentencias a la gramática del lenguaje C3. La gramática completa del lenguaje extendido puede encontrarse en [3]. Las nuevas reglas de producción del lenguaje C3P se muestran a continuación

```
statement : assign ";"
          | "read" "(" iden ")" ";"
          | "write" "(" expr ")" ";"
          | iterate
          | condition
          | "return" expr ";"
          | "wait" "(" iden ")" ";"
          | "notify" "(" iden ")" ";"
          | call ";"
          | parallel_statement
          | comment

parallel_statement : parallel_for
                  | "barrier" ";"

parallel_for      : parallel_for_beginning
statement_block parallel_for_end

parallel_for_beginning : "par_for"
numbers for_definition

parallel_for_end : "end_par_for" ";"
```

## 2.6. Implementación del Compilador

Para la implementación del compilador, se utilizó lenguaje PERL, junto con el módulo Parse::RecDescent[4], que permite implementar un parser descendente recursivo. Este módulo provee una sintaxis para describir formalmente la gramática de un lenguaje de programación. El compilador tiene como objetivo generar las instrucciones Simplesem que implementen la semántica operacional de un programa escrito en lenguaje C3P. Para tal fin, hace uso de un parser descendente recursivo que analiza el código fuente escrito en lenguaje C3P, genera elementos representables por instrucciones Simplesem, y finalmente obtiene dichas instrucciones.

El parser recibe como entrada un programa escrito en lenguaje C3P y la gramática del lenguaje.

A medida que se van encontrando coincidencias del código fuente con las reglas de producción del lenguaje C3P, se van generando las instrucciones Simplesem correspondientes.

Cada producción permite incorporar *acciones*, en las cuales tendremos acceso a los valores encontrados (es decir a los elementos de la sentencia que coincidió con la regla en cuestión). Con el uso de estas *acciones*, se generan las instrucciones Simplesem en función de cada sentencia específica.

## 2.7. Implementación del Intérprete

Para la elaboración del intérprete se utilizó lenguaje C++, junto con la biblioteca de clases Qt.

El intérprete consiste de un editor de texto integrado, donde puede escribirse código fuente, compilarlo y obtener las instrucciones Simplesem en una tabla que representa la memoria de instrucciones y la memoria de datos.

Cuenta con controles que permiten realizar distintos modos de ejecución. Podemos procesar una instrucción de un hilo específico, una instrucción de todos los hilos a la vez, o realizar una ejecución continua hasta que el intérprete encuentre una instrucción halt.

## 3. Uso de la herramienta

El intérprete cuenta con 6 ejemplos integrados que utilizan todas las primitivas del lenguaje C3P. Dichos ejemplos pueden ser cargados, modificados y compilados a instrucciones Simplesem desde la misma herramienta. Tanto el código fuente como versiones distribuidas para Windows, Linux y Mac OS pueden ser descargados desde el repositorio de la herramienta (<https://github.com/lucaslt89/SimplesemExtension>).

El siguiente ejemplo muestra la implementación del problema de los filósofos (Dijkstra, 1965) en lenguaje C3P, para cuatro filósofos, cada uno de los cuales está representado por un hilo.

```
var: tenedor range 1..4, filosofo range
1..4, j;

program{
    //Inicializacion de los semaforos
    (tenedores).
    for(j=0;j <= 3; j++){
        tenedor[j] = 1;
    }
    par_for 4 (filosofo = 0;filosofo <=
3; filosofo++){
        while(1 > 0){
            if(filosofo == 3){
                write(filosofo*10+1);
//Pensando
                wait(tenedor[0]); //Toma
los recursos
```

```

        wait(tenedor[3]);
        write(filosofo*10+0);
//Comiendo
        notify(tenedor[3]);
//Libera los recursos
        notify(tenedor[0]);
    }
    else{
        write(filosofo*10+1);
//Pensando
        wait(tenedor[filosofo]);
//Toma los recursos

wait(tenedor[filosofo+1]);

        write(filosofo*10+0);
//Comiendo

notify(tenedor[filosofo+1]);    //Libera
los recursos

notify(tenedor[filosofo]);
    }
}
end_par_for;
}

```

## Conclusiones

El modelo de ejecución presentado por Carlo Ghezzi y Mehdi Jazayeri brindó una herramienta de gran riqueza educativa, simplificando la enseñanza y el aprendizaje de conceptos fundamentales de los lenguajes de programación.

Con el trabajo aquí presentado, se logró extender el modelo para incorporar lenguajes paralelos, además de brindarse una herramienta que engloba de manera práctica, todos los conceptos planteados teóricamente, desde la definición de un lenguaje hasta la ejecución de un programa escrito en dicho lenguaje sobre una máquina Simplesem.

El lenguaje de alto nivel planteado, el parser/compilador desarrollado, y la interfaz gráfica del modelo de ejecución, conforman una herramienta completa, que permite estudiar todas las etapas por las que atraviesa un programa escrito en un determinado lenguaje, desde que comienza a compilarse, hasta que finaliza su ejecución.

La ejecución paso a paso de un programa paralelo en

una interfaz gráfica, a través de la cual se puede interactuar con la máquina Simplesem, ayuda a la comprensión de conceptos como el interbloqueo o la inanición. Permite además analizar el impacto del acceso concurrente a la memoria de datos, y la manera en que los hilos de un programa paralelo deberán sincronizarse para ejecutar un programa de la manera esperada.

En este trabajo, no se realizó un estudio sobre los beneficios a nivel educativo que la herramienta brinda, dejando esta tarea sujeta a una futura investigación. La herramienta se encuentra disponible en <http://goo.gl/pbXeUT>, de donde puede ser descargada para ser extendida, o para futuros estudios sobre las ventajas a nivel pedagógico de utilizar el modelo Simplesem.

Si bien existen muchas mejoras posibles a la extensión planteada, consideramos que el trabajo realizado es de gran utilidad en el ámbito educativo, ya que brinda una manera práctica de abordar el estudio de los lenguajes de programación y la relación que estos tienen con los recursos de la computadora.

## Referencias

- [1] Carlo Ghezzi y Mehdi Jazayeri, Programming Language Concepts, Tercera edición, 1996. Publicado el 23 de junio de 1997 por John Wiley & Sons. 448 Páginas. ISBN: 0471104264.
- [2] Sitio web oficial de Perl: [www.perl.org](http://www.perl.org) Consultado en diciembre de 2012.
- [3] Lucas Leandro Diez de Medina Quintar. Extensión del Modelo Simplesem a un Lenguaje Paralelo. Tesis de Grado disponible en <http://goo.gl/9QwTJp> -- Publicada en Julio de 2013.
- [4] Parse::RecDescent Documentation. Disponible en: <http://search.cpan.org/perl/doc?Parse::RecDescent> Consultado en diciembre de 2012.

*Dirección de Contacto del Autor:*

**Diez de Medina Quintar, Lucas Leandro**  
 Av. Velez Sarsfield 1611  
 Ciudad Universitaria  
 Córdoba  
 Argentina  
 e-mail: [lucast89@gmail.com](mailto:lucast89@gmail.com)

---

Lucas estudió ingeniería en la Facultad de Ciencias Exactas, Físicas y Naturales. Finalizó sus estudios en septiembre del año 2013, y comenzó su carrera profesional desarrollando aplicaciones para plataformas Apple.

---