Universidad Tecnológica Nacional
Facultad Regional Córdoba
Argentina

IMPLEMENTACION DE UN INTERPRETE SQL EN MANAGED CODE PARA DISPOSITIVOS MOVILES

Sebastián Mariano Salomón
Luis Rogero Lucero
Daniel Alejandro Giménez
Gabriel Mamani
Federico Martín Pascual
Ing. Cesar Ignacio Martínez Spessot
Ing. Calixto Maldonado

Ingeniería en Sistemas de Información
Facultad Regional Córdoba - Universidad Tecnológica Nacional
Maestro M. López esq. Cruz Roja Argentina - Ciudad Universitaria - C.P. (5016)
Córdoba - República Argentina

e-mail: vladisalomon@hotmail.com
e-mail: luisrogero@gmail.com
e-mail: danielalejandro347@gmail.com
e-mail: gabrielmamani@hotmail.com
e-mail: pascualfederico@gmail.com
e-mail: cspessot@gmail.com
e-mail: calixtomaldonado@hotmail.com

Palabras claves: Interprete SQL, Manager Code, Dispositivos móviles.

Resumen:
La extensión del uso de sistemas computarizados exigen a estos a cumplir mayores requisitos de seguridad ya que son constantemente expuestos a intentos de acceso no autorizados, los DBMS no están exentos de esto. Las plataformas de ejecución manejada, tales como Common Language Runtime y Java Virtual Machine proveen un conjunto de servicios de aplicación como aislamiento de aplicaciones, sandboxing security, recolección de basura, y un estructurado manejo de excepciones. Aprovechando estos servicios y adoptando la filosofía de Managed Code es que trabajamos en un intérprete de SQL que permite mayor efectividad en el control del software, eliminando una gran variedad de bugs, que desestabilizan el rendimiento y la seguridad de la aplicación. Además provee una arquitectura para ejecutarse en dispositivos móviles y con la facilidad de ampliar su funcionalidad para soportar nuevos paradigmas, como álgebra temporal y multiespacial.

1. INTRODUCCION:

En la actualidad, el software corre sobre plataformas desarrolladas durante los últimos 40 años y con el pasar del tiempo esto se nota cada vez más. El ambiente de aquel período era muy diferente del entorno de hoy. Los ordenadores eran limitados en la capacidad de memoria y la velocidad; accesible sólo por un pequeño grupo de usuarios técnicamente capacitados y no mal intencionados y raras veces eran conectados a una red o a dispositivos físicos. Pero ningunas de estas características siguen siendo verdaderas, las arquitecturas de computadoras modernas, los sistemas operativos, y los lenguajes de programación no se han desarrollado para ajustarse a los ambientes actuales.

La idea del desarrollo de un interprete SQL en Managed Code surge de un análisis de prioridades en donde se destaca la seguridad en lugar del performance, lo que no quiere decir que se descuide la confiabilidad del mismo.

2. EL INTERPRETE SQL

El intérprete dispone de una fase de análisis donde se interpretan las entradas y se generan estructuras intermedias.
Esta fase esta estructurada de la siguiente forma:

a) Análisis Léxico
b) Análisis Sintáctico
c) Análisis Semántico

2.1. Analizador léxico:

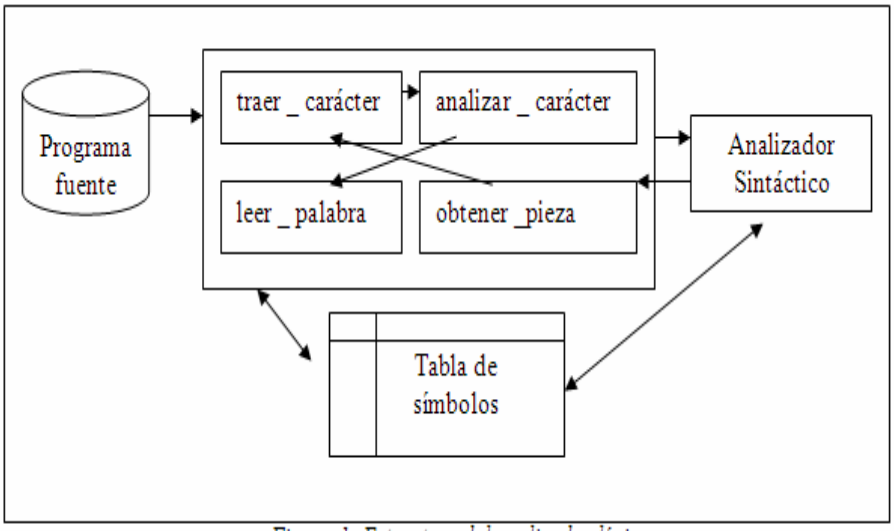

Figura 1: Estructura del analizador léxico

El analizador léxico lee los caracteres del programa fuente uno a uno, identificando las piezas declaradas en una tabla de símbolos para luego generar una secuencia de componentes léxicos que utilizará el analizador sintáctico para hacer su correspondiente análisis.
Las funciones que realiza este analizador son:

- Leer los caracteres de entrada del programa fuente.
- Generar la secuencia de componentes léxicos.
- Eliminar comentarios, espacios en blanco, retornos de carro, y todo lo que este excluido de la sintaxis del lenguaje.
- Reconocer identificadores, palabras reservadas del lenguaje, y tratarlos

correctamente con respecto a la tabla de símbolos (solo en los casos que debe de tratar con dicha tabla).

- Avisar de errores léxicos. Por ejemplo, si @ no pertenece al lenguaje, avisar de un error.

2.2. Analizador sintáctico:

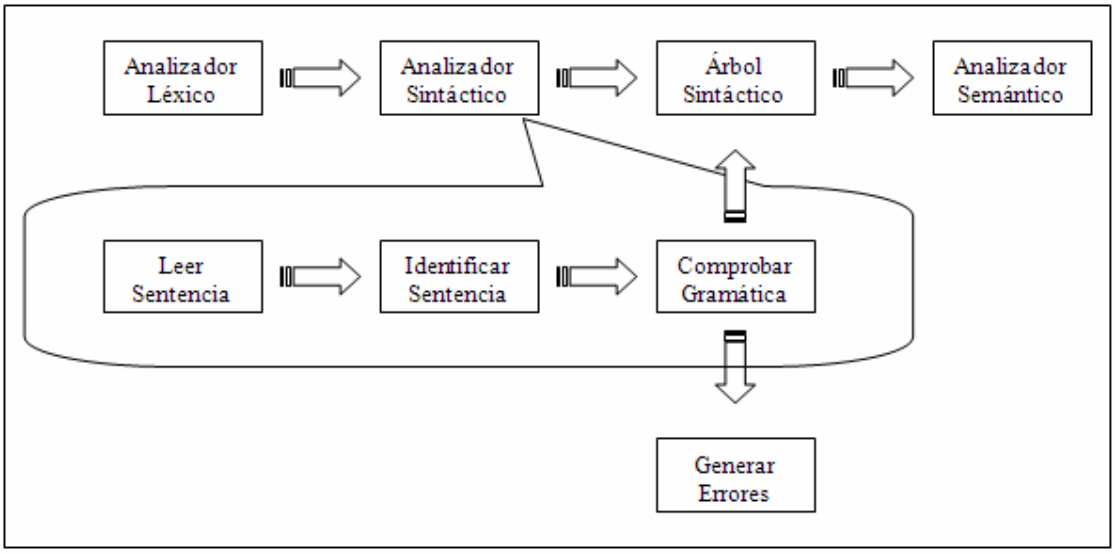

Figura 2: Estructura del analizador sintáctico

El análisis sintáctico o parser se encarga de chequear el texto de entrada, suministrado por el analizador léxico, en base a una gramática dada. En caso de que el texto corresponda a una sentencia valida, suministrará el árbol sintáctico que la reconoce. Además de determinar si la gramática establecida por el lenguaje se respeta, el analizador sintáctico realiza las siguientes funciones:

- Acceder a la tabla de símbolos para armar el árbol sintáctico.
- Chequear los tipos.
- Generar código intermedio.
- Generar errores cuando se producen.

2.3. Analizador Semántico:

La semántica describe el significado de los símbolos, palabras y frases de un lenguaje.
El analizador semántico revisa que las operaciones sean realizables al determinar que las entidades están definidas de los tipos correctos y que los operadores son capaces de manipularlos.
Las funciones que realiza son:

- Comprobación de tipos: La aplicación de los operadores y operandos deben ser compatibles.
- Comprobaciones del flujo del control: Las proposiciones que hacen que se abandone el flujo del control de una construcción deben transferirse a otro punto (break, exit).
- Comprobaciones de unicidad: Hay situaciones en las que un objeto solo puede definirse una vez exclusivamente. Las etiquetas de una sentencia case no deben repetirse, al igual que las declaraciones de objetos.
- Comprobaciones relacionadas con nombre: El mismo nombre debe aparecer dos o más veces.
- Además de comprobar que un programa cumple con las reglas de la

gramática, hay que comprobar que tenga sentido.

- Esta fase también modifica la tabla de símbolos y suele estar mezclada con la generación de código intermedio.

3. MANAGED CODE

Mediante éste código se eliminan categorías completas de errores como, por ejemplo, los daños en el heap y los errores de índice de vectores fuera de los límites. Admite requisitos actuales, tales como el código móvil seguro y los servicios Web XML.
El fundamento principal de la implementación de nuestro intérprete de SQL en Managed Code es que puede comprobarse la seguridad a través de tipos confiables de datos y la administración automática de memoria.
La compilación aplicada al intérprete en C# nos permite generar Código Intermedio (CI), al ejecutar uno de estos componentes el Common Language Runtime (CLR), el cual a su vez verifica si es la primera vez que se ejecuta, en caso de ser asi, realiza una compilación JIT para generar el código nativo, optimizado para el hardware en done debe ejecutarse. Si el componente ya se había ejecutado antes, la compilación ya se había realizado, por lo que el CLR solo debe localizar el código nativo y volver a ejecutarlo.
Este proceso hace que la primera vez que se ejecuta un componente se obtenga un tiempo de respuesta por debajo de lo normal. [1]. Sin embargo nos permite analizar las sentencias SQL (que se escriban para nuestro motor de base de datos) con más performance para una segunda etapa de ejecución.

Secuencia de ejecución utilizando código administrado:

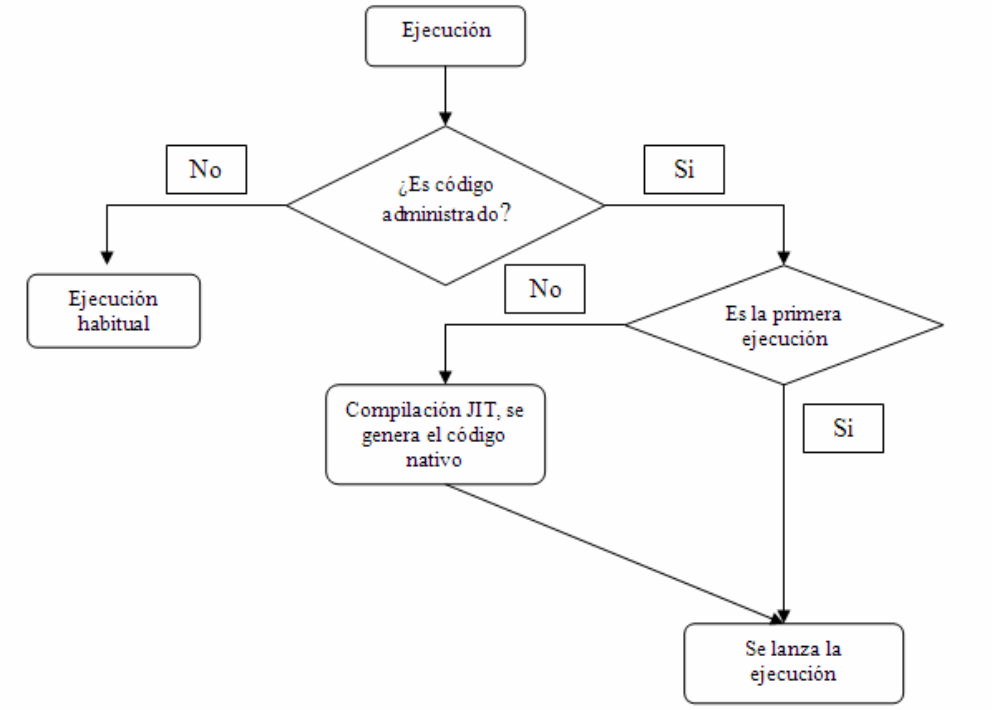

Figura 3: Proceso de ejecución.

4. ESTUDIO DE PERFORMANCE

Native Code vrs. Managed Code.

Con el objetivo de determinar la demanda de microprocesador y memoria para la ejecución del proceso del motor de consultas se han estudiado los resultados probados por Intel [2], los cuales se realizaron basándose en el hardware: DBPXA250 Intel XScale® Personal Internet Client Architecture el cual cuenta con 64MB de SDRAM, 32 MB de memoria Flash y microprocesador de 295 Mhz PXA255.

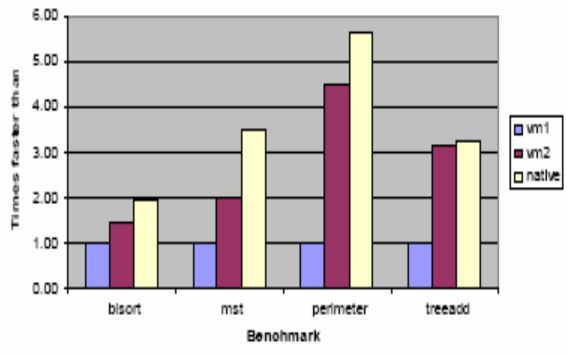

Figura 4: Al comparar código nativo contra j2me cldc se encontró que funciona más eficientemente que código de J2ME en todos los casos que se midieron.
La versión de código nativa fue compilada con la opción - O usando un compilador GCC optimizado para la micro-arquitectura de XScale®.

Las conclusiones que se obtienen de esta comparación es:

- El código nativo funciona más rápido que el código administrado, en todos los casos.
- Incluso con una buena implementación, el tiempo de ejecución (vm2) del código nativo funciona 1.7 veces más rápido.
- La diferencia es más dramática con vm1, en la cual el código nativo funciona de 2.0 a 5.5 veces más rápido.

Comparación de áreas de memoria

También se deseaba comparar la diferencia de las áreas dinámicas de memoria entre los intérpretes y los compiladores dinámicos.
Se muestreo la utilización de la memoria durante el curso de la carga de trabajo a través de /proc/<pid>/status , en la cual el kernel de Linux actualiza para cada proceso. De todas las muestras se determino la memoria residente máxima.

La figura 5 demuestra la memoria residente máxima.

Figura 5: *Al comparar el uso de memoria residente máxima, se descubrió que la memoria utilizada por JITs no es perceptiblemente diferente que la utilizada por los intérpretes.*

La figura 6 demuestra el área residente de la memoria en un cierto plazo.

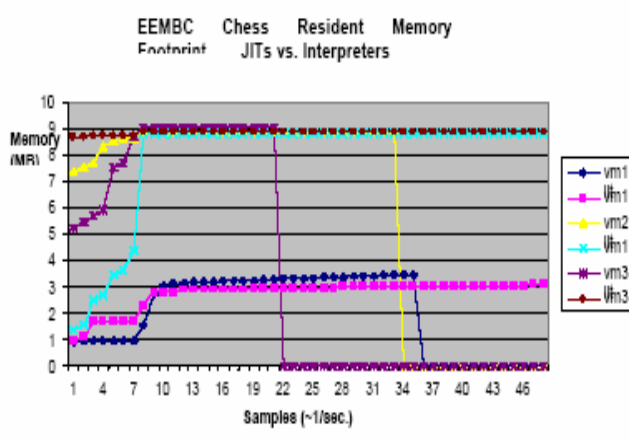

Figura 6: *La comparación del estado de la memoria residente en un lapso de tiempo demuestra que JIT y el intérprete para cada VM consume casi la misma cantidad de memoria. El tamaño del heap para todo el VMs es igual a 8MB. Las caídas abruptas significan el fin de las pruebas patrones.*

La clave que se aprendió de esta comparación es:

• La utilización dinámica de la memoria de JITs y los intérpretes son comparables. La una excepción era vm3 mientras que funcionaba la carga de trabajo del regexp.

• No todas las puestas en práctica utilizan la memoria completa proporcionada a ella. Se encontró que vm3 en intérprete y modo de JIT utiliza perceptiblemente menos memoria que vm1 o vm2.

Del estudio de performance de la arquitectura del intérprete determinamos que si bien demandará más procesamiento, el gasto en memoria no será significativo. Teniendo en cuenta que actualmente los procesadores actuales han superado la ley de Moore (observación empírica que la velocidad del microprocesador se duplica cada 18 meses), no es una restricción para nuestro desarrollo.

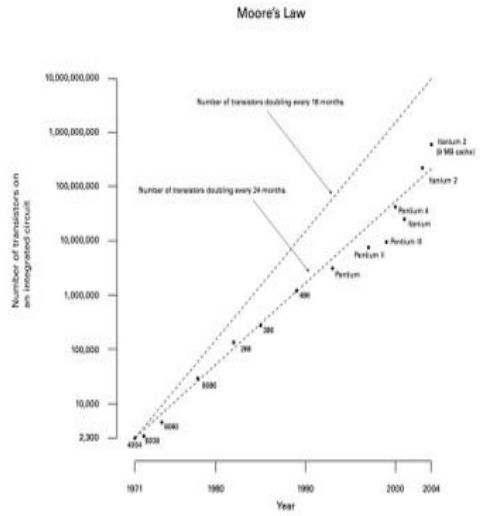

Figura 7: *Gráfico representativo de Ley de Moore.*

5. IMPLEMENTACION

Contamos con el intérprete SQL ejecutándose en un dispositivo móvil, el cual cuenta con la característica de interpretar consultas básicas del ANSI SQL, creando objetos en memoria y notificando de errores en las diferentes fases: léxica, sintáctica y semántica.

Para la implementación se ha usado el paradigma orientado a objetos desarrollado en el lenguaje C#.NET.

La interfaz actualmente se ejecuta en una Pocket PC HP con Windows Mobile 4.0 con 64 MB. El peso en memoria del archivo ejecutable es de 62,5 KB, por lo cual corre con buena performance sobre el equipo.

La interfaz de usuario es simple para facilitar el uso de tinta digital (lápiz por presión a pantalla), por lo cual posee 2 funcionalidades: Ejecutar SQL y limpiar la interfaz de consulta (ver figura 8)

Figura 8: *Interfaz del interprete ejecutándose en una Pocket PC.*

6. CONCLUSION

Nuestro trabajo actual es una aplicación innovadora de técnicas de desarrollo que se adapten a los escenarios actuales, en donde se prioriza la seguridad sobre la velocidad, esto es aceptable hasta cierto punto y es por eso que investigamos sobre cual es la verdadera diferencia de rendimiento entre el código administrado y el nativo. El resultado fue que el costo no es significativo comparado con el beneficio que brinda la seguridad en el software, nuestra intención es centrarnos en la eficiencia del producto, garantizando su funcionamiento y confiando en que las nuevas tecnologías suplirán la deficiencia de velocidad en la ejecución. Además, el desarrollo de nuestro propio interprete lo consideramos un paso indispensable para sentar bases sólidas que nos permitan evolucionar hacia nuevas tecnologías que el grupo tiene en vista.

REFERENCIAS

[1] http://www.microsoft.com/spanish/msdn/articulos/archivo/010903/voices/fastmanagedcode.asp.

[2] Comparative Performance Análisis of Mobile Runtimes on Intel XScale® Technology.

[3] http://cursosgratis.emagister.com/frame.cfm?id_user=80459550103200603666857655767516&id_centro=27422070033149505256526748524548&id_curso=52910020050250544854525157674570&url_frame=http://www.microsoft.com/spanish/msdn/articulos/archivo/010903/voices/highperfmanagedapps.asp

[4] http://studies.ac.upc.edu/FIB/CASO/seminaris/2q0304/T3annex.pdf#search=%22codigo%20administrado%20filetype%3Apdf%22

[5] Desarrollo de un Sistema de Gestión de Base de Datos Relacional.
TecnoDB de: Anabel Natalia Ruiz, Ing. César Ignacio Martínez Spessot, Ing. Juan Carlos Vázquez, Ing. Calixto Maldonado
www.investigacion.frc.utn.edu.ar/tecnodb/

[6] Traductores, compiladores e intérpretes de Maria del mar Aguilera Sierra, Sergio Gálvez Rojas.
www.lcc.uma.es

[7] Diseño de un interprete SQL de Hilmar Hinojosa L.*
http://sisbib.unmsm.edu.pe

[8] Semántica, de
http://lintiinfo.linti.unlp.edu.ar