EXTRACCIÓN DE PARALELISMO Y CONCURRENCIA EN AMBIENTES DE MODELADO Y SIMULACIÓN DE SISTEMAS COMPLEJOS

DIEGO ALEXANDER RUEDA PLATA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2014

EXTRACCIÓN DE PARALELISMO Y CONCURRENCIA EN AMBIENTES DE MODELADO Y SIMULACIÓN DE SISTEMAS COMPLEJOS

DIEGO ALEXANDER RUEDA PLATA

Trabajo de grado para optar al título de Ingeniero de Sistemas

Director:
Carlos Jaime Barrios Hernández
Ingeniero de Sistemas PhD.

Codirector:
Hugo Hernando Andrade Sosa
Ingeniero de Sistemas Msc.

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2014

Índice general

Índice de figuras

Índice de tablas

Índice de Anexos

RESUMEN

Título: Extracción de paralelismo y concurrencia en ambientes de modelado y simulación de sistemas complejos.*

Autor: Diego Alexander Rueda Plata**

Palabras Clave: Paralelismo, Concurrencia, Dinámica de Sistemas, Evolución, Delphi

Resumen: El grupo SIMON de Investigaciones ha desarrollado una herramienta para el modelado y simulación con Dinámica de Sistemas llamada EVOLUCIÓN, este software es un producto de diferentes proyectos de pre-grado y maestría desarrollados por integrantes del grupo SIMON.

Con el surgimiento de la computación en paralelo, es posible diseñar y desarrollar software de tal manera que este sea beneficiado por el poder computacional de todos los procesadores; sin embargo, el desarrollo en paralelo es más complejo que el desarrollo de software secuencial. Por ello, hoy dia es mas común planificar la explotación de concurrencias sobre aplicaciones ya desarrolladas, mediante estrategias y herramientas que permitan identificar las zonas potencialmente paralelizables en la aplicación.

Este proyecto ha sido enfocado en la extracción de concurrencia y paralelismo en ambientes de modelado y simulación con dinámica de sistemas, como es la herramienta Evolución 4.5. En primer lugar se llevó a cabo la extracción de la arquitectura de la aplicación para identificar las regiones funcionales que la conforman, seguidamente se realizaron evaluaciones de rendimiento en cada área para determinar el estado inicial del software.
Una vez identificadas las rutinas candidatas a paralelización, se definió una estrategia que ha permitido identificar las oportunidades de concurrencia y desarrollar soluciones que optimicen la ejecución de la herramienta eficientemente, esta estrategia ha sido validada realizando evaluaciones de rendimiento que emplean los mismos parámetros iniciales tomando como criterio el tiempo de ejecución.

Finalmente, las experiencias en la explotación de concurrencias han sido reunidas en unos lineamientos que ayudaran a la paralelización de ambientes de modelado y simulación de sistemas complejos.

---

*Trabajo de Grado en la Modalidad de Investigación.

**Facultad de Ingenierías Físico Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director PhD. Carlos Jaime Barrios Hernández. Codirector MSc. Hugo Hernando Andrade Sosa.

ABSTRACT

Title: Parallelism and concurrency extraction from software environments for modeling and simulation of complex systems.*

Author: Diego Alexander Rueda Plata**

Keywords: Parallelism, Concurrency, System Dynamics, Evolución, Delphi

Description: SIMON research group has developed a tool for modeling and simulation with system dynamics known as Evolución, this software is the product of several undergraduate and postgraduate projects developed by members of SIMON.

With the emergency of parallel computing engineers have the opportunity to design and develop software to exploit the computational power of multicore machines, however, producing code in parallel is substantially more difficult than sequential code. Therefore in order to exploit more efficiently concurrencies in existing applications strategies to discover, introduce and verify parallelism must be used.

This project focuses in the extraction of concurrency and parallelism in modeling and simulation software with system dynamics, in this case with Evolución 4.5. Firstly, the architecture of Evolución has been extracted to identify the logical components of this software, afterwards several performance tests have been done in each component to discover the current state of Evolución and find bottlenecks.

Once the procedures that are candidate to exploit its concurrencies have been found, a strategy to identify and exploit this opportunities developing solutions to optimize the software performance has been developed, these solutions are then validated with new performance evaluations under the same conditions of the first tests.

Finally, based on our experiences exploiting concurrency in this software we developed some guidelines to help parallelize modelling and simulation tools of complex systems.

*Undergraduate final project, research modality.

**Physico-Mechanical Engineering Faculty. Systems Engineering and Computer Science School. Director PhD. Carlos Jaime Barrios Hernández. Codirector MSc. Hugo Hernando Andrade Sosa.

Introducción

En el campo de la programación paralela, no hay definidos métodos o estrategias para paralelizar aplicaciones, hasta el momento se puede mencionar que la paralelización de una aplicación va muy ligada a la tarea que el aplicativo desarrolle y la forma en que fue programada. Por lo cual si fue concebida para ejecutarse en forma serial, es muy común que se deban hacer modificaciones para hacer posible que esta se ejecute de forma concurrente sobre el hardware del computador.

Este documento presenta la extracción de concurrencia y paralelismo en una herramienta para el modelado y simulación con Dinámica de Sistemas llamada Evolución, este software ha sido desarrollado por el grupo SIMON de Investigación en Modelado y Simulación de la Universidad Industrial de Santander.

Inicialmente se ha extraído la arquitectura actual del sistema, esto permite identificar las áreas funcionales de Evolución, para posteriormente realizar mediciones de rendimiento mediante un perfilador. Una vez documentadas estas pruebas se procede a identificar las áreas candidatas a paralelización. A continuación, el código fuente es estudiado y se propone una solución que permita aprovechar las opciones de paralelización del lenguaje de desarrollo Delphi. Esto ha sido validado mediante pruebas de rendimiento a cada sector de la herramienta en su versión optimizada con el fin de contrastar con los resultados anteriores, y determinar el factor de ganancia alcanzado, producto de las optimizaciones y la ejecución concurrente.

Finalmente, las experiencias en la explotacin de concurrencias han sido reunidas en unos lineamientos que ayudaran a la paralelización de ambientes de modelado y simulación de sistemas complejos.

Capítulo 1

Planteamiento del Problema

Con el surgimiento de los procesadores multinúcleo, se ha hecho posible diseñar y desarrollar aplicaciones de tal manera que estas puedan beneficiarse de la potencia computacional de todos los procesadores. Para aprovechar este hardware adicional disponible en estas arquitecturas, es necesario desarrollar software paralelo.

Existen importantes razones para trabajar en este tipo de aplicaciones, pero es claro que hay trabajo en software secuencial ya desarrollado que sería inviable descartarlo y desarrollarlo nuevamente, por lo que es necesario adquirir la habilidad de convertir el código en serie a paralelo, para el aprovechamiento de la capacidad computacional de las maquinas actuales.

La metodología para paralelizar aplicaciones, podría considerarse como un proceso iterativo donde en primer lugar se analiza el software y se detectan las oportunidades de paralelización realizando pruebas de rendimiento al software serial. Posteriormente, el paralelismo es introducido en el código fuente y se verifica la correcta transformación del código. Finalmente, se realizan pruebas de rendimiento del software ejecutado concurrentemente, para ver la ganancia de concebir una programación paralela a fin de acelerar la ejecución de una aplicación.

El enfoque de este proyecto está en la extracción de oportunidades de paralelización del software Evolución, un ambiente para el tratamiento de sistemas complejos mediante dinámica de sistemas, desarrollado por el grupo de investigación SIMON de la Universidad Industrial de Santander.
Este ambiente usualmente presenta procesos que consumen muchos recursos y su tiempo de ejecución en los procedimientos es mayor al teóricamente esperado por sus desarrolladores, pues a pesar de su robustez desde sus principios la implementación no estaba pensada para explotar las concurrencias naturales en este tipo de herramientas.

Se plantea un proyecto que permita encontrar estas concurrencias e implementarlas en el software, de modo que este aproveche mejor los recursos computacionales de la máquina.

Capítulo 2

Justificación

El grupo SIMON de Investigación perteneciente a la Escuela de Ingeniería de Sistemas de la Universidad Industrial de Santander, ha desarrollado importantes aplicaciones orientadas al aprendizaje y simulación de fenómenos complejos en diferentes áreas. El proyecto bandera de SIMON es Evolución, una herramienta software para el modelamiento y simulación con Dinámica de Sistemas. El grupo SIMON y la herramienta Evolución han obtenido el reconocimiento en este campo en el ámbito nacional e internacional.

Tanto la herramienta Evolución como los modelos desarrollados con este han sido base para el desarrollo de numerosas investigaciones en el grupo SIMON, entre ellas SIPROB 2.0[3] y EcoGranja[11], por esta razón es de gran interés el continuo mejoramiento de la herramienta la cual ha madurado desde la versión 1.0 desarrollada en 1994 hasta la versión actualmente disponible Evolución 4.5 a través de diferentes proyectos desarrollados en el grupo SIMON los cuales han sido elaborados siguiendo un proceso de análisis, dise´ no, desarrollo, evaluación y mejora continua cuyo objetivo ha sido proporcionar un software de calidad a la comunidad de dinámica de sistemas.

En el desarrollo de Evolución se han empleado diversas técnicas de ingeniería del software para sacar el máximo partido al lenguaje (Delphi) para contar con una herramienta robusta y de calidad a la comunidad, sin embargo el procesamiento de las rutinas no aprovecha de manera eficiente los recursos computacionales disponibles en los computadores actuales y sobre los cuales regularmente es ejecutada la aplicación.

Este proyecto abarca una problemática respecto al uso eficiente de los recursos hardware por parte del software Evolución, para esto se plantea la extracción de concurrencia y paralelización, para generar un prototipo de una versión paralela del aplicativo de modelado y simulación de Dinámica de Sistemas.

Capítulo 3

Objetivos

3.1. Objetivo General

Identificar y extraer las oportunidades de concurrencia en el ambiente de modelado y simulación soportado en la dinámica de sistemas Evolución, y proponer mecanismos de implementación de procesamiento en paralelo.

3.2. Objetivos Especificos

- Extraer y analizar la arquitectura de Evolución para identificar la concurrencia y las oportunidades de paralelización.
- Proponer una estrategia que permita el procesamiento eficiente y generar un prototipo implementando dicha estrategia.
- Validar el desempeño obtenido sobre una arquitectura definida de gran escala realizando análisis y evaluación de rendimiento.
- Establecer lineamientos que orienten la implementación de mecanismos de explotación de las oportunidades de concurrencia y paralelización que garanticen un alto rendimiento en aplicaciones que soporten el modelamiento y simulación de sistemas complejos usando dinámica de sistemas, observando el caso de Evolución.

Capítulo 4

Estado del Arte

4.1. Dinámica de Sistemas

La dinámica de sistemas es una metodología y técnica de modelación matemática para analizar, entender y discutir situaciones problemáticas complejas. Mediante diagramas de Forrester se puede desarrollar una representación gráfica de los sistemas dinámicos, modelando las relaciones entre las partes mediante símbolos que corresponden a una interpretación del sistema.

Los niveles corresponden a las variables de estado de la teoría de sistemas y representan las variables cuya evolución es significativa para el estudio del sistema. Estos acumulan material a través de los flujos.

Los flujos definen el comportamiento del sistema, ya que determinan la velocidad de cambio del estado del sistema de acuerdo a un conjunto de ecuaciones asociadas. Las ecuaciones dependen de la información que las válvulas reciben del sistema y del entorno. La información se transmite a través de los canales de información.

Las variables auxiliares corresponden a pasos intermedios en el cálculo de las funciones asociadas a los flujos; se utilizan para simplificar el proceso bien porque ciertos cálculos matemáticos se emplean en varias ecuaciones o bien porque tienen cierto significado o interpretación física que puede ser interesante observar.

Las variables exógenas representan la interacción del sistema con el exterior, cuya evolución se supone independiente a la del sistema.

Los retrasos pueden afectar a la transmisión del material o de información, pero en ambos casos introducen mayor capacidad descriptiva.

4.2. Ambientes de Modelado y Simulación con Dinámica de Sistemas

Los ambientes de modelado han sido popularizados desde la época de 1990, a continuación se listaran tres ambientes reconocidos ampliamente en el ámbito de los investigadores de sistemas dinámicos.

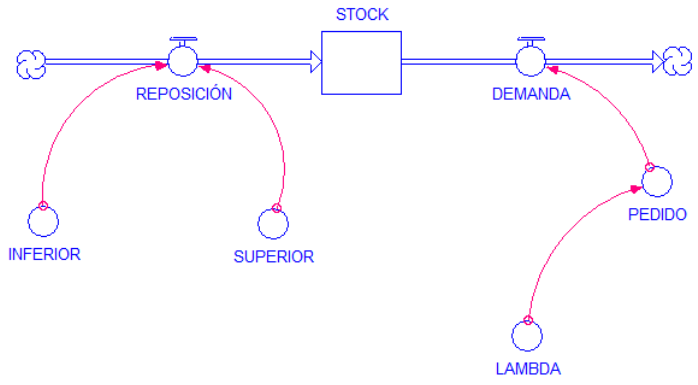
Los ambientes Stella y Vensim son desarrollados por compañías estadounidenses con diferentes versiones orientadas a empresas y a investigadores.

El ambiente Evolución ha sido desarrollado en la Universidad Industrial de Santander y ha sido ampliamente usado para investigación en el grupo de investigación en dinámica de sistemas Simon y también por otros grupos externamente.

4.2.1. STELLA

Stella es un ambiente desarrollado por isee systems, compañía fundada en 1985 por Barry Richmond fue galardonada con el premio Jay Forrester al ser la primera en introducir una herramienta de modelado y simulación basada en iconos STELLA.

Figura 1: Ejemplo de Modelo en STELLA

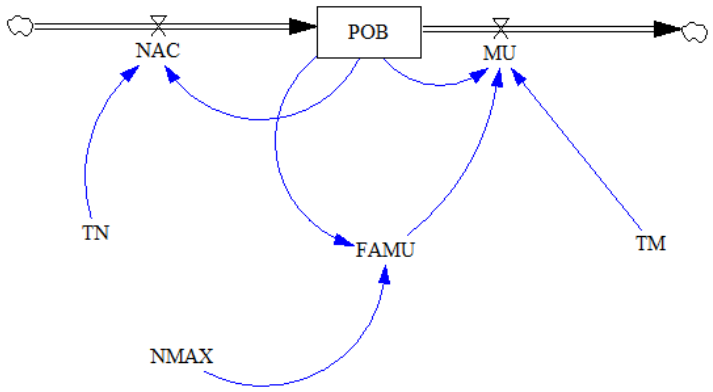


Stella es un software privativo, por lo que el acceso a los procesos de simulación implementados es restringido y no ha sido posible determinar si implementa algún tipo de explotación de concurrencias.

4.2.2. VENSIM

Vensim es un software usado para desarrollar, analizar y empaquetar modelos con realimentación dinámica. Este ha sido desarrollado por Ventana Systems, y actualmente se encuentra en la versión 6.1.

Figura 2: Ejemplo de Modelo en VENSIM



Vensim utiliza una técnica de simulación compilada. Las ecuaciones del modelo son escritas en un programa en C, el cual es luego compilado y enlazado a Vensim como una librería de enlace dinámico (DLL). No es posible determinar si el código en C usa alguna forma de paralelización.

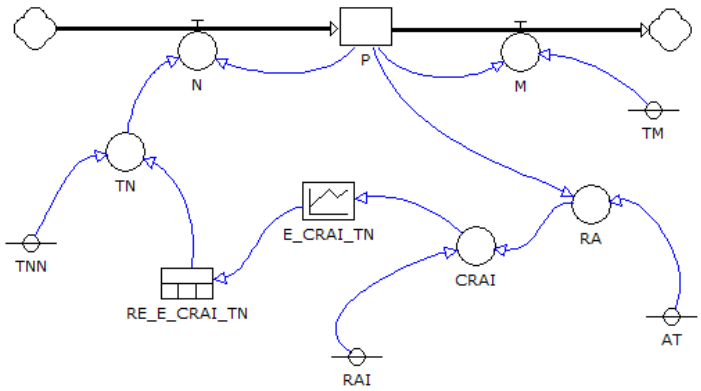
4.2.3. EVOLUCIÓN

Evolución es un software que permite modelar y simular fenómenos complejos con Dinámica de Sistemas, brinda la posibilidad de crear, editar y guardar Diagramas de Influencias y Diagramas de Flujo-Nivel (o Forrester) como parte de la documentación del modelo.

Evolución inició con una propuesta del ingeniero Hugo Hernando Andrade Sosa profesor de la Universidad Industrial de Santander, fundador del grupo SIMON de Investigación en Modelamiento y Simulación adscrito a la Escuela de Ingeniería de Sistemas e Informática.

Hasta la fecha se encuentra en la versión 4.5, esta ha sido el resultado del trabajo de proyectos de pregrado y está enmarcado dentro de los lineamientos de investigación en modelamiento y simulación del Grupo SIMON de Investigación.

Figura 3: Ejemplo de Modelo en EVOLUCIÓN



Para el desarrollo de Evolución se utilizaron técnicas de Programación Orientada a Objetos, diseño basado en componentes, patrones de diseño, todo respaldado por diagramas en Lenguaje Unificado de Modelado UML, ha sido creado en el entorno de desarrollo Delphi.

Evolución es una herramienta desarrollada durante 20 aos al interior del Grupo SIMON y durante este tiempo han habido mltiples proyectos de pre-grado que han alimentado el desarrollo de la aplicación.

A continuación se presenta una lista de los proyectos y los autores que han contribuido al desarrollo de Evolución.

TÍTULO	INTEGRANTES	AÑO
Dinámica de sistemas aplicada a la simulación de algunos fenómenos de transporte (SDS)	Hugo Hernando Andrade Sosa Luis Carlos Gómez Flórez	1990
Herramienta software para la representación solución y análisis de sistemas dinámicos en la forma espacio-estado: SIMUIS 1.0	Alberto Julio Arias Blanco Rafael Ortiz Fonseca	1995
Herramienta software para construir y analizar modelos mediante dinámica de sistemas: «Evolución 2.0»	Carlos Humberto Ardila Arango Pedro Enrique Durán Sánchez	1995
Diseño y elaboración de la interfaz gráfica y material de marketing para la herramienta software Evolución 3.0	Rosa Cecilia Martínez Beltrán Jorge Heriberto Vanegas Vega	1997
Interfaz gráfica para Evolución 2.0 y Simuis 1.0	Amolfi Hernando Pineda Gómez Blanca Inés Rueda Núñez	2000
Análisis y diseño de Evolución 32, herramienta software para la simulación con dinámica de sistemas	Dilia Ester Torres Cantillo Gerardo Solórzano Dangond	2000
Evolucin 3.0, herramienta software para el modelamiento y simulacin con dinmica de sistemas	Maria Isabel Ardila Arango William Moreno Suárez	2000
Evolucin 3.5 herramienta software para el modelamiento y simulacin con dinmica de sistemas	Mario Cuellar Yeneris Emiliano Lince Mercado	2003
Anlisis de sensibilidad y diagramas de influencias para la herramienta de modelado y simulacin, Evolucin	Angélica María Alfonso Higuera Jaqueline Calderón Ramírez	2003
Componente de sistema de inferencia difusa (FIS) para Evolucin 3.5	Gesman David Machado Mendoza César Eduardo González Pérez	2006
Mantenimiento del Software Evolucin 4.0	Alexander Elías Hernández Cuadrado Adriana Judith Monsalve Quintero	2014

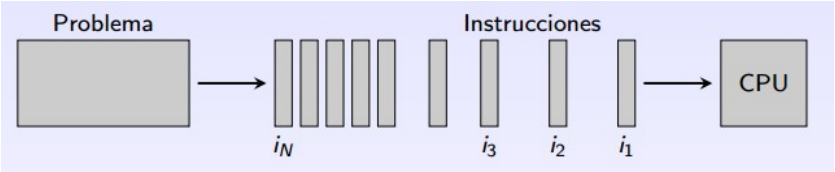
Capítulo 5

Marco Teórico

5.1. Computación en Paralelo

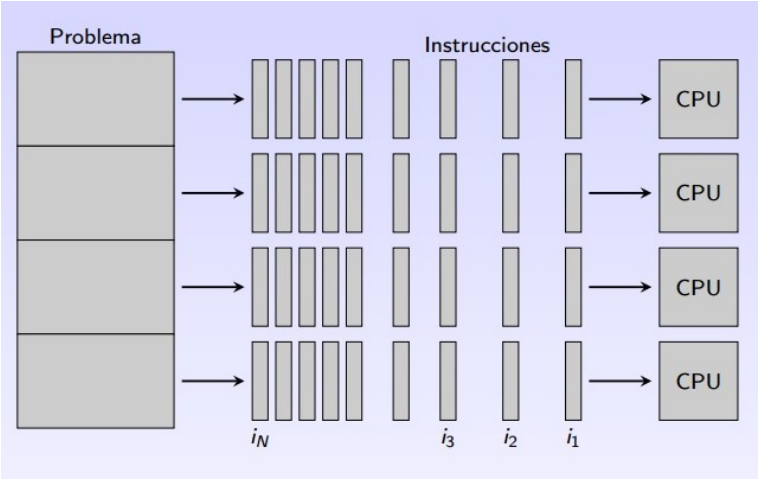
Tradicionalmente, el software ha sido desarrollado de manera serial para ser ejecutado por un solo computador con un procesador. En este estilo, los problemas son divididos en series de instrucciones y estas son ejecutadas una detrás de otra, solo una instrucción a la vez.[2]

Figura 4: Procesamiento en Serie



La computación en paralelo es el uso simultáneo de varios procesadores para resolver un problema computacional. El problema se divide en varias partes que serán resueltas concurrentemente, cada una de estas partes se subdivide en series de instrucciones.

Figura 5: Procesamiento en Paralelo



5.1.1. Ley de Amdahl

Esta ley es usada para estimar la máxima aceleración posible al total de un algoritmo cuando solo una parte de este es mejorada. Es usada en computación en paralelo para predecir teóricamente la aceleración ganada al usar múltiples procesadores. El aumento de velocidad de un programa usando múltiples procesadores en paralelo está limitado por la fracción secuencial de este.[2]

$$A = \frac{1}{(1 - P) + \frac{P}{N}} \tag{5.1}$$

Donde:

P es la proporción de un programa que puede ser paralelizada.
(1 - P) es la proporción que no puede ser paralelizada y permanece serial.
A es la maxima aceleración que puede ser conseguida al usar N procesadores.

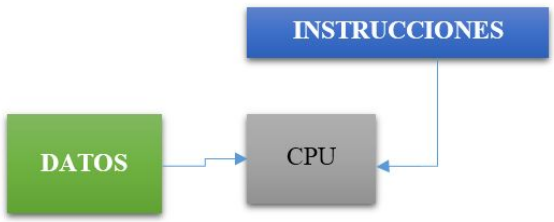
5.1.2. Clasificación de Flynn

Esta clasificación divide las arquitecturas de computadores según si estos operan un flujo o múltiples flujos de instrucciones, y si estos manejan únicos o multiples conjuntos de datos.

SISD (Flujo único de instrucciones, Flujo único de datos)

La clasificación SISD es equivalente a un programa completamente secuencial donde no se explota el paralelismo en las instrucciones o flujos de datos. Ejemplos de esta arquitectura son las maquinas tradicionales como un computador personal, aunque en la actualidad estas PC tienen múltiples procesadores.[2]

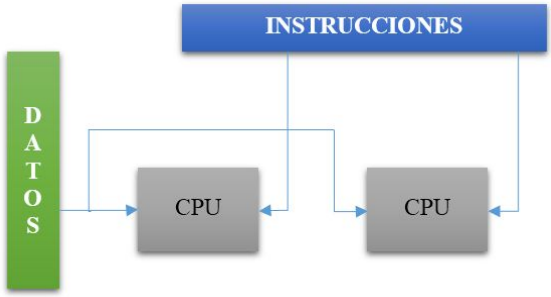
Figura 6: Flujo único de instrucciones, Flujo único de datos



MISD (Flujo múltiple de instrucciones, Flujo único de datos).

En este tipo de computador multiples instrucciones operan sobre un flujo único de datos. Esta arquitectura es poco común y es usada generalmente para tolerancia de fallos, donde sistemas heterogeneos operan sobre los mismos datos y debe coincidir en el resultado.[2]

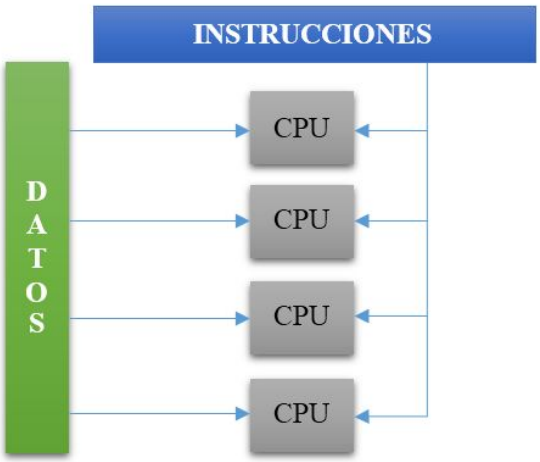
Figura 7: Flujo múltiple de instrucciones, Flujo único de datos



SIMD (Flujo único de instrucciones, Flujo múltiple de datos)

Este tipo de arquitectura explota múltiples flujos de datos que operan con un único flujo de instrucciones para realizar operaciones que pueden ser naturalmente paralelizadas. Un ejemplo común de estas arquitecturas son las GPU.[2]

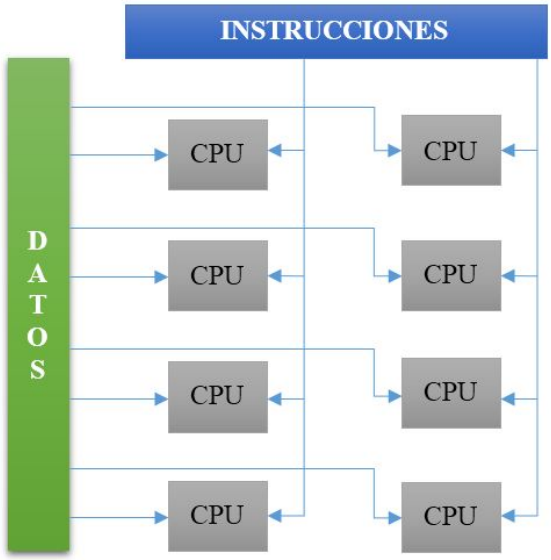
Figura 8: Flujo único de instrucciones, Flujo múltiple de datos



MIMD (Múltiples flujos de instrucciones, Múltiples flujos de datos)

Múltiples procesadores autónomos que ejecutan diferentes instrucciones en diferentes grupos de datos. Los sistemas de memoria distribuida entran en esta categoría.[2]

Figura 9: Flujo múltiple de instrucciones, Flujo múltiple de datos



5.1.3. Arquitecturas de Computación Paralela

Arquitecturas de Memoria Compartida

La memoria principal es uniformemente compartida por todos los procesadores y cada procesador tiene igual tiempo de acceso a la memoria compartida.[2]

Arquitecturas de Memoria Distribuida

Cada procesador tiene su propia memoria privada. Las operaciones son realizadas localmente y de ser necesario datos externos, los procesadores se comunican entre sí para adquirir estos datos.[2]

Arquitecturas Hibridas

El tiempo de acceso a la memoria depende de la ubicación de esta en relación al procesador que intenta accederla. Un procesador accede a la memoria local más agilmente que a una memoria externa.[2]

5.1.4. Modelos de Programación Paralela

Programación de Memoria Compartida

Con base en la arquitectura de memoria compartida, como todos los procesadores acceden al mismo espacio de memoria, estos pueden sincronizarse y trabajar juntos mediante variables compartidas. Cuando el programa se ejecuta en un hilo maestro ejecuta las porciones secuenciales de el algoritmo, cuando se requiere el paralelismo el hilo maestro crea hilos adicionales que trabajan en la sección en paralelo, al terminar de ejecutarse el control vuelve al hilo maestro.[2]

Programación de Memoria Distribuida

Este modelo se enfoca en arquitecturas de memoria distribuida, los programas se organizan en grupos de tareas independientes que se comunican entre sí mediante mensajes.[2]

5.1.5. Tipos de Paralelismo

Paralelismo a nivel de Instrucciones

Este toma ventaja de las secuencias de instrucciones que requieren diferentes unidades funcionales. Diferentes arquitecturas enfocan este tipo de manera diferente, pero la idea en general es tener las instrucciones no dependientes entre sí, ejecutándose simultáneamente para mantener las unidades funcionales trabajando continuamente.[2]

Paralelismo a nivel de Datos

Se refiere a escenarios en donde la misma operación es realizada concurrentemente sobre unos elementos de una colección. La colección es particionada y distribuida en múltiples hilos que pueden operar en diferentes segmentos simultáneamente.[2]

Paralelismo a nivel de Tareas o Hilos

Este tipo de paralelismo se enfoca en distribuir procesos de ejecución (hilos) a través de diferentes nodos de computación en paralelo. Los hilos pueden ejecutar el mismo o diferente código, estos se pueden comunicar entre si mientras están trabajando. La comunicación usualmente toma lugar al pasar datos de un hilo al otro.[2]

Capítulo 6

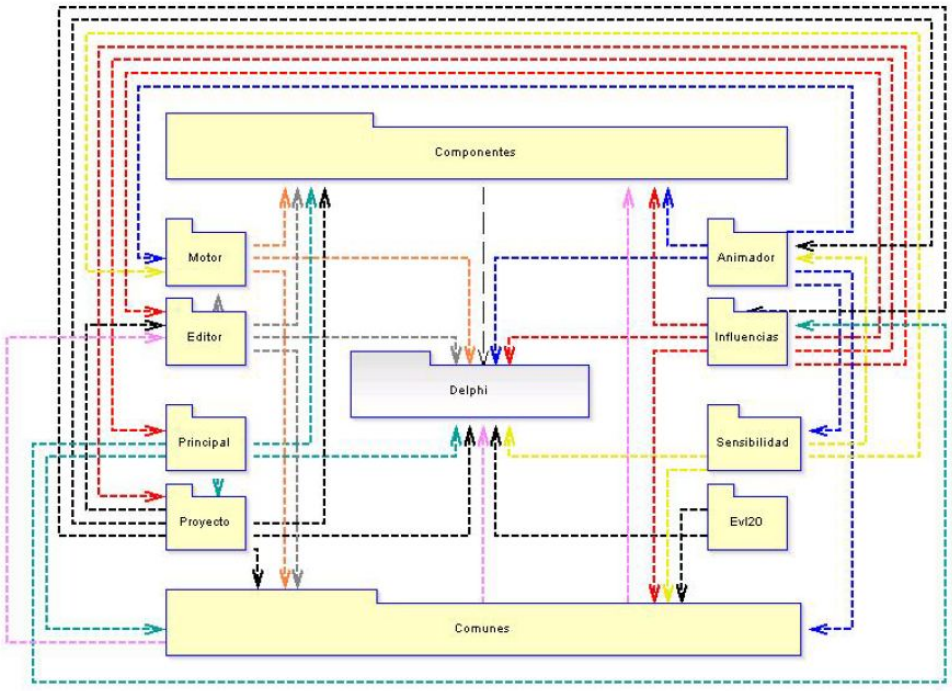
Desarrollo del Proyecto

6.1. Extración de la Arquitectura de Evolución

La arquitectura del software de un sistema es la estructura o estructuras del mismo, que comprenden los componentes del software, las propiedas externas de estos elementos y las relaciones entre ellos.[4]

Para el análisis de la arquitectura de Evolución han sido consultados un proyecto de maestría[18] desarrollado por Jair Moreno y uno de pre-grado[13] encargado del mantenimiento de la versión 4.0 los cuales han documentado la arquitectura de Evolución. La Figura Figura 10 es resultado de estos proyectos y representa la Arquitectura del Software Evolución 4.0.

Figura 10: Arquitectura de Evolución 4.0



Debido a lo anterior y a que el trabajo de mantenimiento no culmina con la presentación del proyecto de grado, y se sigue trabajando en resolver incidencias reportadas por los usuarios, se hace necesario extraer una representación arquitectónica más acorde al estado actual del software. El proceso de extracción es una labor asistida por diferentes herramientas que analizan el código fuente y permiten estudiar las relaciones entre los componentes desarrollados, adicionalmente la colaboración de los anteriores desarrolladores ha sido valiosa.

A continuación se hará una breve descripción de las herramientas utilizadas en el proceso de extracción de la arquitectura.

- EssModel: Esta herramienta permite la generación automática de diagramas de clase a partir de código fuente en Delphi. También permite la navegación y edición a través del modelo generado a partir del código fuente con el propósito de crear la documentación del mismo en formato HTML.[7]
- Model Maker: Este grupo de herramientas permite visualizar el código existente para entender la estructura y el diseño de este. Entre las funcionalidades con que cuenta el módulo Dependency Analyzer permite analizar las dependencias y usos entre unidades basado en la cláusula de usos del código fuente generado en Delphi.[26]

En primer lugar, usando EssModel se generó una documentación del sistema en html, permitiendo estudiar el código internamente lo cual es una gran ayuda para los nuevos desarrolladores de Evolución.

En segundo lugar, mediante una herramienta incluida en Model Maker se ha producido un archivo que contiene el análisis de las relaciones entre las unidades de Evolución, también incluye las dependencias circulares entre estas.

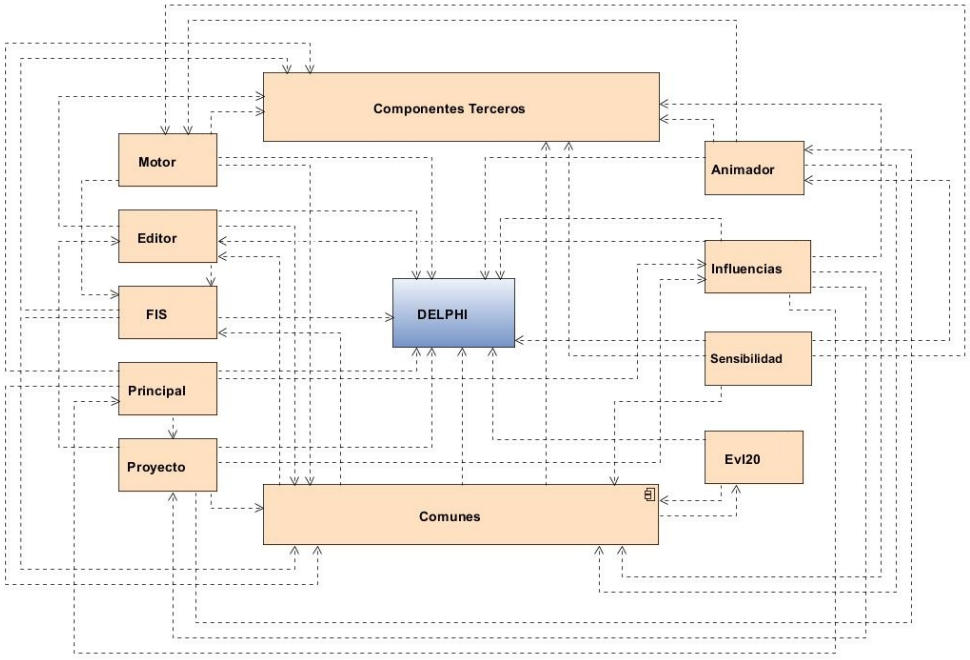
Usando estas dos herramientas ha sido posible representar, en la Figura Figura 11, la arquitectura actual del sistema.

Esta arquitectura representa los componentes de la herramienta, cada bloque está compuesto de varias clases que en permiten el funcionamiento de cada componente, y las relaciones representan la interacción entre estos para cumplir con el funcionamiento de la aplicación.

El funcionamiento de los componentes será explicado a continuación:

Motor: Es el encargado de procesar los elementos de Forrester en las diferentes iteraciones de la simulación. Es el núcleo de Evolución permite iniciar y parar la simulación a antojo, pudiendo cambiarse las condiciones iniciales de simulación y valores de variables antes y durante la simulación.

Figura 11: Arquitectura de Evolución 4.5



Editor: Provee mecanismos para la creación, edición y destrucción de elementos del diagrama de Forrester (Flujo-Nivel). Permite crear, guardar y cargar archivos de modelos y manejar los elementos del diagrama, ademas de sectores, exportar el modelo en modo texto o gráfica, imprimir el modelo, entre otras funcionalidades.]

Influencias: Provee mecanismos para la creación, edición y destrucción de elementos del diagrama de Influencias. Tiene funcionalidades similares al componente Editor.

Animador: Este componente permite la observación de datos de una corrida de simulación en modo tabla o gráfica. Ofrece mecanismos para elegir las variables a graficar, la forma, los colores de la gráfica, creación y eliminación de vistas de simulación y métodos para imprimir en diferentes formas las gráficas y tablas de cada vista.

Principal/Proyecto: Estos dos componentes se encargan del trabajo mas externo de Evolución y de las funciones de administración de los proyectos.

FIS: Este componente permite la inclusión y evaluación de elementos del modelo en trminos de un sistema de inferencia difuso y ofrece mecanismos para la evaluación y análisis del comportamiento del elemento.

Sensibilidad: Este componente permite el análisis de sensibilidad de los modelos me-

diante variación de escenarios y parámetros.

Evl20: Este componente es un legado de las primeras versiones de Evolución permite la compatibilidad de los modelos creados en la versión 2.0.

Comunes: En este componente se encuentran las clases que sirven de apoyo a los demás componentes de la aplicación, este es ampliamente usado en la aplicación.

Componentes Terceros: En este se encuentran los componentes desarrollados por terceros que brindan soporte al funcionamiento de los componentes en Evolución:

- **FormulaCompiler:** Este componente es usado por el Motor de Evolución para evaluar las formulas que vienen de las ecuaciones del modelo.
- **FlatStyle:** Este componente permite presentar los elementos de la interfaz de Evolución en un estilo plano.
- **RegExpr:** Permite al componente FIS el manejo de expresiones regulares.
- **TeeChart:** Presente en el componente Animador, permite la presentación de gráficas en Evolución.
- **SDL-suite:** Permite al componente FIS presentar gráficas en 3D y 2D.
- **Abbrevia:** Este componente permite Comprimir los datos que serán incluidos en el archivo guardado.

6.2. Evaluaciones de Rendimiento

El método usado para identificar las areas en Evolución, donde la implementación del procesamiento en paralelo resulta mas beneficioso para el sistema, es realizar un analisis de rendimiento usando un perfilador instrumental[AQtime] y de ser necesario un perfilador de muestreo[ASMprofiler], esto se debe a que el perfilador instrumental permite identificar las rutinas que emplean mas tiempo de ejecución con gran detalle al ejecutar una corrida de simulación, pero el tiempo real de la aplicación se ve afectado por la instrumentación, por otro parte, el perfilador de muestreo es menos preciso pero permite medir el tiempo de ejecución de los hilos de la aplicación .

Las herramientas escogidas para realizar el analisis de rendimiento son descritas a continuación.

- AQtime: Es un perfilador de rendimiento de tipo instrumental, el cual añade instrucciones a cada sección del código lo que permite recolectar información de las rutinas usadas en el programa, el tiempo que se demora el cuerpo de esta. Es desarrollado por SmartBear Software y se integra con la ambiente integrado de desarrollo de Delphi 7.[22]

- ASMprofiler: Es un perfilador desarrollado para Windows y permite medir aplicaciones sin cambiar el código original, tiene muy poca carga sobre la aplicación lo que permite que esta corra a velocidad normal y permite observar el tiempo de ejecución de los hilos creados.[19]

6.2.1. Características del equipo y las pruebas

Los resultados de la prueba dependen en gran medida del computador en el cual se realizan, en este caso han sido realizadas en un equipo proporcionado por el grupo SIMON con las siguientes caracteristicas.

- Procesador: AMD Athlon(tm) 64x2 Dual Core Processor 2.2GHz
- Memoria RAM: 2048MB
- Sistema Operativo: Windows 7 Professional 32bits

La prueba consiste en la simulación de ciertos modelos desarrollados en el grupo SIMON, estos modelos hacen parte de investigaciones presentadas como proyecto de grado. Para cada componente la prueba realizada puede ser diferente dependiendo de lo que se quiera evaluar.

Para los componentes Motor, Animador y Sensibilidad se realizan pruebas de ejecución de un modelo por 1000 iteraciones.

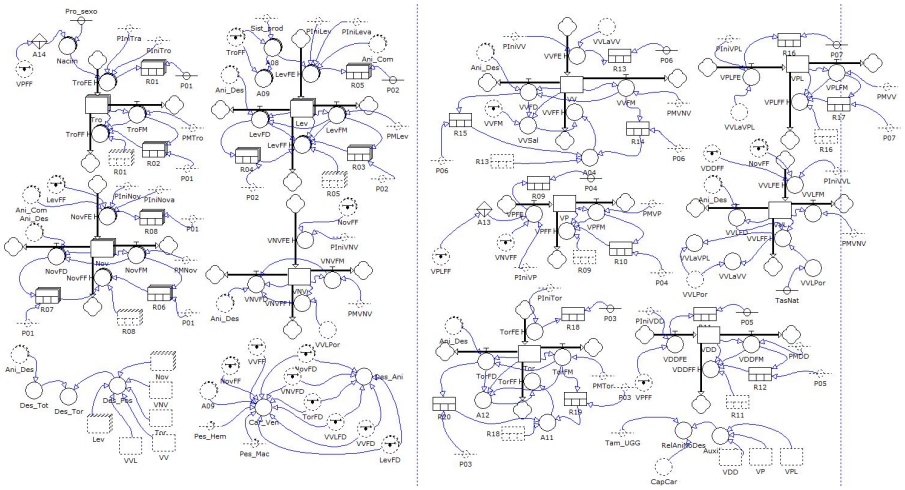
Para los componentes Editor e Influencias encargados de la creación del modelo se realizaron pruebas de carga de un modelo con varios elementos desde archivo, ya que la creación manual del modelo depende de la velocidad del modelador.

Las pruebas en los demás componentes han sido explicadas en su respectiva sección.
Los modelos y sus autores son nombrados a continuación.

- "Propuesta de un modelo de simulación de sistemas de producción de ganadería para la investigación integral, un enfoque sistémico"
  Autores: Omar Barragán y Urbano Gómez. [3]
  En la Figura Figura 12 se muestra una parte del modelo en cuestión.
- "Modelo para la toma de decisiones en la inversión pública municipal con el presupuesto de libre inversión y orientado al mejoramiento del desarrollo local, un enfoque sistémico"
  Autor: Natalia Díaz.[6]

La elección de estos modelo está dada por la alta complejidad que presentan y son ampliamente considerados en el grupo SIMON de investigación como modelos que aprovechan ampliamente las caracteristicas presentes en Evolución como son la lógica difusa, el manejo de vectores, y las diferentes presentaciones de resultados, entre otras.

Figura 12: Sección del modelo de Barragán/Gómez.



Los modelos listados anteriormente se encuentran en la base de datos de la biblioteca de la Universidad Industrial de Santander, tambien en el Grupo SIMON de Investigaciones.

6.2.2. Componente - Animador

El paquete Animador permite la observación de una corrida de simulación de datos en modo Tabla o Gráfica. Este paquete contiene 66 clases que ofrecen mecanismos para la escogencia de las variables a graficar, la forma, los colores de la gráfica, creación y eliminación de vistas de simulación y métodos para imprimir en diferentes formas las gráficas y tablas de cada vista.

Para esta evaluación se realizaron dos pruebas usando el modelo de Barragán/Gómez[3] representando todos los elementos del modelo, la simulación se realizó con 1000 iteraciones en ambos modos Gráfica y Tabla. Las rutinas con mayor tiempo se han registrado en la Tabla 3.1 para el modo Tabla y en la Tabla 3.2 para el modo Gráfica.

Prueba #1 - Modo Tabla

Tabla 1: Prueba - Animador Modo Tabla

Rutina	**Tiempo (s)**
TAnimador::PasoMotor	210,4
TAnimador::setDatos	1,6
TTabla::Animar	1,46
TManejadorResultados::EnviarMensaje	1,42
TGrafica::Animar	0,94
TFormAnimador::Cargar	0,26
TFormAnimador::ColocarValoresIniciales	0,2

Prueba #2 - Modo Gráfica

Tabla 2: Prueba - Animador Modo Gráfica

Rutina	**Tiempo (s)**
TManejadorResultados::EnviarMensaje	2726,21
TAnimador::PasoMotor	217,4
TGraficador::SetVerBarra	3,05
TGrafica::MostrarPropiedadesControl	2,17
TAnimador::setDatos	1,99
TManejadorResultados::CondicionesSimulacion	1,08
TTabla::Animar	0,61
TGrafica::Animar	0,59

El tiempo trabajado en el componente Animador en el modo tabla y gráfica difiere ámpliamente en la rutina *TManejadorResultados::EnviarMensaje* usada en el modo gráfica, por lo tanto se hace necesario estudiar el funcionamiento de esta rutina y los llamados de esta para determinar la causa de este retraso en la ejecución. También se destaca la rutina *TAnimador::PasoMotor* en los dos modos de ejecución. Todas las métricas tomadas al componente se encuentran incluidas en el Anexo A - Métricas del Componente Animador.

6.2.3. Componente - Motor

El paquete Motor contiene 7 clases principales, este paquete es el núcleo de la simulación y permite al sistema manejar la evaluación de los elementos presentes en el diagrama de Forrester.

Las Tablas 3.3 y 3.4 contienen las métricas de rendimiento para el paquete Motor, estas fueron tomadas al simular los 2 modelos de pruebas, solo se muestran los 7 mayores valores de las rutinas en la medición. Los demás valores se encuentran en el Anexo B - Metricas del Componente Motor.

Tabla 3: Métricas de rendimiento - Modelo Barragán/Gómez

Rutina	**Tiempo (s)**
TMotorEvolucion::Iteracion	104,6
TEvaluador::ReemplazarFuncion	67,62
TEvaluador::EvaluarExpresion	55,37
TElementoEvaluacion::TransformarDefiniciones	36,43
ReemplazarParaElemento	28,46
TListaElementosEvaluacion::Get	8,07
TElementoEvaluacion::Preparar	3,69

Tabla 4: Métricas de rendimiento - Modelo Natalia Díaz

Rutina	**Tiempo (s)**
TMotorEvolucion::Iteracion	103,04
TEvaluador::EvaluarExpresion	64,65
TElementoEvaluacionFis::Evaluarse	32,13
TEvaluador::ReemplazarFuncion	18,57
TElementoEvaluacion::TransformarDefiniciones	10,02
ReemplazarParaElemento	8,39
TEvaluador::ReemplazarFunciones	1,51

6.2.4. Componente - Editor

El paquete Editor es el encargado de permitir al usuario de Evolución diseñar diagramas de Flujo-Nivel, también de cargar y pintar los diagramas desde el archivo, está compuesto por 25 clases.

La prueba para el Editor, sera la carga y dibujo de los elementos desde archivo del modelo base de producción usado en la prueba del motor, los elementos del modelo son mostrados en la Figura Figura 13.

Figura 13: Estadisticas del Modelo de pruebas

Los resultados obtenidos al realizar el análisis de rendimiento, en la Tabla 3.5 se encuentran los tiempos de mayor demora, indican tiempos en general muy eficientes de carga y dibujo desde archivo, por esta razón el paquete Editor no será tenido en cuenta para optimización mediante procesamiento en paralelo. En el Anexo C se encuentran todas las métricas tomadas al componente Editor.

Tabla 5: Métricas de rendimiento - Editor

Rutina	**Tiempo (s)**
ULineas::AutoPintarse	0,47
UInterfacesDeDibujoPintarLetra	0,17
UElementosGraficos::VariablesGraficasDe	0,15
TFormForrester::EnviarMensaje	0,12
TListaElementos::VerificarPosicionNombre	0,1
UElementosGraficos::DibujarSeleccion	0,1
UElementosGraficos::CambiarPosicion	0,06
TLinea::PtInLine	0,05

6.2.5. Componente - Influencias

El paquete Influencias presenta al usuario una interfaz en la que puede crear, editar y guardar diagramas de Influencia, como parte de la documentación del modelo. Está compuesto por 14 clases.

Los rutinas que mayor tiempo consumen en la simulación son mostradas en la Tabla 3.6 y el listado completo se encuentra en el Anexo D - Métricas del Componente Influencias.

Tabla 6: Métricas de rendimiento - Influencias

Rutina	**Tiempo (s)**
TElementosInfluencia::DibujarNombre	0,24
TFormInfluencias::EnviarMensaje	0,13
TElementosInfluencia::AutoPintarse	0,11
TEditorInfluencias::Create	0,09
TFormInfluencias::CargarPropiedadesInfluencia	0,04
TEditorInfluencias::CalcularHojas	0,03
TFormInfluencias::FormCreate	0,03

Igualmente, como ocurrió en el paquete Editor, las rutinas usadas en el paquete Influencias presentan tiempos bastante eficientes, debido a esto no es necesario tenerlas en cuenta para el proceso de paralelización.

6.2.6. Componente - Comunes

El paquete Comunes es un área de soporte para las demás unidades del sistema, está compuesto por 31 clases, y sus funciones varían ampliamente.

Los rutinas que mayor tiempo consumen en la simulación son mostradas en la Tabla 3.7 y el listado completo se encuentra en el Anexo E - Métricas del Componente Comunes

En esta evaluación la rutina *UProcesos::SacarToken* genera un retardo menor pero considerable para el sistema.

Tabla 7: Métricas de rendimiento - Comunes

Rutina	**Tiempo (s)**
SacarToken	38,36
THiloEvento::Execute	12,02
TLista::Cargar	3,81
SacarPar	2,85
TEFuncion::GetNombreFuncionMayusculas	1,3
TengaNombreE	0,94
DividirVector	0,9
Agregar	0,76

6.2.7. Componente - Sensibilidad

El componente Sensibilidad es un componente especial desarrollado como proyecto de grado en el grupo SIMON de investigación, este presenta dos funcionalidades que permiten el análisis de sensibilidad mediante *Variación de Escenarios* y *Variación de Parámetros*.

Este componente para su funcionamiento hereda o sobrescribe rutinas de los componentes Motor y Animador, por esta razón ambos componentes serán tenidos en cuenta en la evaluación de rendimiento para el componente Sensibilidad.

Prueba #1 - Sector Variación de Escenarios

Tabla 8: Prueba - Variación de Escenarios

Rutina	**Tiempo (s)**
TMotorEvolucion::Iteracion	432,05
TEvaluador::ReemplazarFuncion	147,13
ReemplazarParaElemento	85
TEvaluador::EvaluarExpresion	78,5
TElementoEvaluacion::TransformarDefiniciones	77,63
TGraficaEscenarios::Animar	21,92
TEvaluador::ReemplazarFunciones	5,88

Prueba #2 - Sector Variación de Parámetros

Tabla 9: Prueba - Variación de Parámetros

Rutina	**Tiempo (s)**
TMotorEvolucion::Iteracion	387,06
TEvaluador::ReemplazarFuncion	100,89
TEvaluador::EvaluarExpresion	60,84
TTablaParametros::DatosAnalisis	50,95
ReemplazarParaElemento	49,03
TElementoEvaluacionParametros::TransformarDefiniciones	44,45
TTablaDatos::AnimarT	27,04

Las rutinas detectadas en ambas pruebas con mayor tiempo provienen del componente Motor, por lo que se espera que al optimizar estas rutinas, que ya han sido identificadas, en el componente Motor estos resultados mejoren. Igualmente, se plantea la posibilidad de evaluar los escenarios o parámetros en paralelo, esto debido a que cada evaluación está compuesta de varios modelos independientes donde cambian los valores de todo el modelo(escenarios) o el cambio es de un valor particular de este(parámetros) y pueden ser evaluado en diferentes hilos de proceso.
Las métricas tomadas del Componente Sensibilidad se encuentran en el Anexo F.

6.2.8. Componente - Principal/Proyecto

El componente Principal permite el manejo del sistema por parte del usuario, es el encargado de llamar a los demás componentes y de administrar la cara mas externa de Evolución.

El componente Proyecto se encarga de crear, cargar desde archivo y administrar los proyectos hechos en Evolución y permite manejar mas de un proyecto a la vez en el software.

Los rutinas que mayor tiempo consumen en la simulación son mostradas en la Tabla 3.10 y el listado completo se encuentra en el Anexo G - M étricas de los componentes Principal y Proyecto. Las rutinas usadas por los componentes Principal y Proyecto no generan mayor retardo en el sistema, por lo que no serán tenidas en cuenta en el proceso de paralelización.

Tabla 10: Prueba - Componente Principal/Proyecto

Rutina	**Tiempo (s)**
TProyecto::Abrir	7,42
TFormPrincipal::TBnAbrirClick	0,78
TFormPrincipal::Activarse	0,46
TProyecto::CrearDInfluencias	0,16
TProyecto::Destroy	0,12
LlenarListaConEscenario	0,11
TProyecto::CrearForrester	0,1

6.2.9. Componente - FIS

El componente de lógica difusa en Evolución fue desarrollado como proyecto de grado por [17], este componente permite al usuario de Evolución: Crear Variables lingüisticas de Entrada y de Salida, Definir las operaciones del FIS, Crear reglas, Ver gráficos que muestran el comportamiento del modelo, entre otros.

Las resultados de la prueba de rendimiento para el componente de lógica difusa son mostrados en la Tabla 3.11, y el listado completo se muestra en el Anexo H - Métricas del Componente FIS.

Tabla 11: Prueba - Variación de Parámetros

TPanelGrafica3d::dibujar3D	30,10
TRegla::EvaluarRegla	4,30
TOperador::calcular	1,62
ActualizarProgresBar	0,62
TFuncion::evaluarFuncion	0,57
Uutilidadesfis::RealACadena	0,26
Uutilidadesfis::ReemplazarVarios	0,24

6.2.10. Componentes - Evl20 y Componentes Terceros

Respecto a la evaluación del componente Evl20 y a los componentes desarrollados por terceros usados en Evolución.

En el primer caso, EVL20 es un componente desarrollado para facilitar la compatibilidad de los modelos desarrollados en la versión 2.0 de Evolución, por lo que es usado muy raramente, debido a que la mayoría de proyectos han sido desarrollados en la versión 3.5 en adelante.

Respecto a los componentes terceros, los componentes que mayor influencia tienen en el funcionamiento de Evolución son TeeChart[24] y SDL-suite[15], estos componentes son privativos, por esta razón no pueden ser modificados.

Debido a esto EVL20 y los componentes terceros no son tenidos en cuenta en el alcance de este proyecto y no se realizan pruebas de rendimiento para estos.

6.2.11. Elección de Rutinas candidatas

De la evaluación hecha a los componentes de Evolución se han escogido las rutinas cuyo tiempo de ejecución indica una ejecución ineficiente. La Tabla 3.12 contiene información de las rutinas seleccionadas, la clase a la que pertenecen y el tiempo total de ejecución.

Tabla 12: Rutinas Escogidas

Componente	**Rutina**	**Tiempo(s)**
Animador	TManejadorResultados::EnviarMensaje	2726,21
	TAnimador::PasoMotor	217,4
Motor	TMotorEvolucion::Iteracion	103,4
	TEvaluador:ReemplazarFuncion	67,62
	TEvaluador:EvaluarExpresion	64,65
	TElementoEvaluacion:TransformarDefiniciones	36,43
	TElementoEvaluacionFIS:Evaluarse	32,13
	TElementoEvaluacion:ReemplazarParaElemento	28,46
Comunes	Uprocesos:SacarToken	33,35
Sen.Escenarios	TMotorEvolucion:Iteracion	432,05
	TEvaluador:ReemplazarFuncion	147,13
	TElementosEvaluacion:ReemplazarParaElemento	85
	TEvaluador:EvaluarExpresion	78,5
	TElementosEvaluacion:TransformarDefiniciones	77,63
Sen.Parámetros	TMotorEvolucion:Iteracion	387,06
	TEvaluador:ReemplazarFuncion	100,89
	TEvaluador:EValuarExpresion	60,84
	TTablaParametros:DatosAnalisis	50,95
	TElementosEvaluacion:ReemplazarParaElemento	49,03
FIS	TPanelGrafica3D:Dibujar3D	29,17

6.3. Estrategia de Paralelización

Evolución es una herramienta cuya historia de desarrollo tiene mas de 20 años, siendo mejorada gracias a proyectos de grado desarrollados en el grupo SIMON. Debido a esto, el código fuente de Evolución se caracteriza, en primer lugar por ser extenso el desarrollo durante estos años ha hecho que el ambiente contenga mas de 100.000 lineas de código.

Para definir una estrategia que permita explotar las concurrencias de la aplicación es importante tener una familiaridad con esta y conocer el lenguaje en que ha sido desarrollada con el fin de reestructurar la rutina y exponer la concurrencia para finalmente explotarla. Sin embargo, la solución que se implemente para cada rutina puede ser diferente.

La estrategia definida a continuación consiste en una serie de pasos que sirven de base para la explotación de concurrencias en las rutinas de la herramienta.

A. Estudiar el código en las rutinas:

Este paso inicial consiste en el análisis de las rutinas candidatas identificadas de las pruebas de rendimiento realizadas, en una herramienta con varios años de trabajo y diferentes desarrolladores es de esperar problemas comunes en código heredado con poco mantenimiento, por ejemplo documentación incompleta, repetición de código, interdependencia de clases y rutinas ineficientes.

La documentación incompleta se soluciona temporalmente cuando el desarrollador se ha familiarizado con el software y el lenguaje, pero los problemas de repetición e interdependencia requieren refactorización del código y las rutinas ineficientes se solucionan al implementar optimización del código en serie.

B. Opciones de Paralelización

Delphi 7 es una interfaz de desarrollo presentada en 2002, en ese entonces no se habían desarrollado librerías de paralelización para el IDE. Sin embargo, para versiones posteriores a Delphi 7 como son la versión Embarcadero Delphi XE4 se han desarrollado librerías de paralelización que permiten un comportamiento similar al de OpenMP en C++[20] pero orientadas a aplicaciones en Delphi como son OmniThreadLibrary[10] y AsyncCalls[12].

Por esta razón, y a las diferentes herramientas de mantenimiento que pueden ser integradas a Delphi XE4, se ha tomado la decisión de migrar el código fuente de Evolución a la Versión Delphi XE4, ya que permitiran en un futuro responder más agilmente a los problemas de soporte y mantenimiento de la aplicación.
Entre los cambios mas importantes realizados a la aplicación se encuentra el soporte de Unicode, donde las cadenas de texto en Delphi XE4 cambiaron su funcionamiento interno esto será explicado en mas detalle en el Anexo I - Migración a Delphi XE4. Adicionalmente, fue necesario actualizar los componentes desarrollados por terceros que soportan el funcionamiento de Evolución.

Posterior a la migración de Evolución a Delphi XE4 existen entonces tres opciones de paralelización que pueden ser usadas en las soluciones a implementar:

- **Clase TThread en Delphi**: Es una clase abstracta incluida en Delphi que permite la creación de hilos separados de ejecución en la aplicación. Descendientes de la clase TThread representan un hilo de ejecución en una aplicación multihilos[8].

- **OmniThreadLibrary**: Es una librería de paralelización que permite mediante pocas lineas de código crear múltiples hilos. Es desarrollada por Primož - Gabrijelčič, aun se encuentra en fase de desarrollo.[10]

- **AsyncCalls**: Con esta librería es posible ejecutar multiples funciones al mismo tiempo y sincronizarlas en cualquier punto de la función o metodo que las inicio, esto permite ejecutar código cuyo resultado es necesitado mas adelante en un hilo diferente. Ha sido desarrollada por Andreas Hausladen, pero abandonada desde el 2011.[12]

C. Optimizar y/o Explotar las concurrencias

El fundamento de cualquier programa en paralelo es la concurrencia, es decir las series de instrucciones que pueden ser ejecutadas al mismo tiempo. La clave está en análizar cada rutina candidata y pensar diferentes formas de descomponer el problema para crear una concurrencia util. Este proceso inicia con la identificación de instrucciones que pueden ser ejecutadas simultaneamente, luego descomponer el problema para minimizar la ejecución en serie. Sin embargo, una optimización de la rutina en ejecución serial puede mejorar el rendimiento de la rutina, disminuyendo el tiempo de ejecución de la misma haciendo innecesaria la explotación de concurrencias en el código.

D. Verificar y Medir el rendimiento

Finalmente, cada solución desarrollada para las rutinas tendrá una evaluación de rendimiento bajo las mismas condiciones iniciales de las primeras pruebas con el objetivo de contrastar la rutina antes y después de la optimización o explotación de concurrencias.

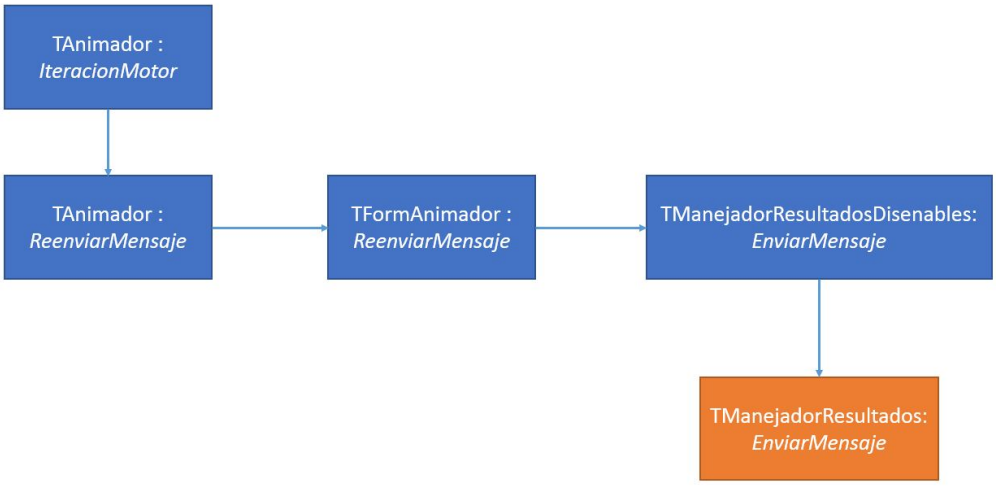
6.4. Análisis y Solución a las rutinas escogidas

6.4.1. Rutinas del Componente Animador

En el componente Animador se detectaron dos rutinas con un tiempo de ejecución considerable *TManejadorResultados::EnviarMensaje* y *TAnimador::PasoMotor*.

TManejadorResultados::EnviarMensaje La rutina *EnviarMensaje* en el componente Animador hace parte del sistema de mensajería implementado por Evolución que permite enviar unos tipos de mensajes a la aplicación principal, mostrar mensajes al usuario, mostrar mensajes de error en pantalla y registrarlo en un archivo Evolucion.log.

Figura 14: Gráfico del llamado de rutinas



Gracias a una característica del perfilador AQtime podemos ver en el gráfico 3.5 el llamado de rutinas, esta resulta especialmente útil cuando el retardo no se encuentra en el algoritmo de la rutina, sino en el llamado que se hace de esta.

El gráfico puede ser trazado hasta la rutina *IteracionMotor* en la clase TAnimador, donde la rutina es llamada para cada iteración con el fin de mostrar un mensaje de texto con el valor actual de la iteración del motor en la barra de estado de Evolución.

Esta actualización en la interfaz gráfica de Evolución es realizada en el hilo principal de la aplicación el cual se encarga también de representar la gráfica al usuario, sin embargo al ser llamada la aplicación ordena al hilo principal que pause la graficación hasta enviar el mensaje.

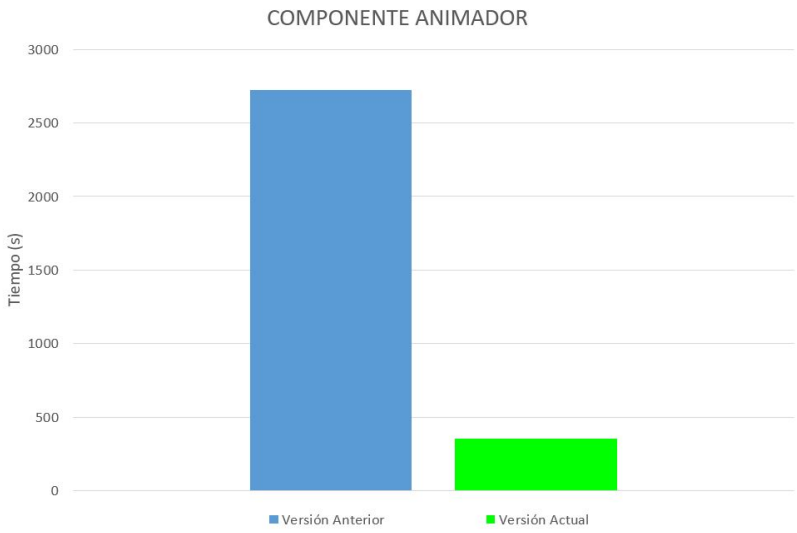
Por lo tanto, la modificación realizada a esta rutina consiste en la creación de un hilo secundario usando la clase TThread de Delphi, esto permite al hilo principal de la aplicación encargarse de realizar la graficación de los resultados, mientras el hilo secundario actualiza la iteración actual del motor en la barra de estado de Evolución.

TAnimador:PasoMotor La rutina *PasoMotor* es ejecutada cada vez que el motor realiza una iteración y evalúa los elementos del modelo, al terminar cada iteración el Motor envia un evento al animador dando la orden de Animar los elementos en pantalla añadiendo la última iteración.

En Delphi, el manejo de la interfaz gráfica es normalmente manejada únicamente por un hilo de ejecución por lo que todas las modificaciones a esta deben ser sincronizadas de ser manejadas por diferentes hilos, por esta razón no es posible paralelizar este rutina en Evolución.

Para comprobar que los cambios realizados efectivamente disminuyen el tiempo en la presentación de la simulación se realiza una prueba de rendimiento al componente Animador de Evolución, estas métricas son registradas en el Anexo F.6 y F.7 y en la Figura Figura 15 se presenta una comparación de los tiempos de ejecución del componente en la versión inicial y despues de implementar estos cambios.

Figura 15: Aceleración en el Componente Animador



6.4.2. Rutinas del Componente Motor

El modelo matemático de la estructura general de la dinámica de sistemas es un sistema de ecuaciones diferenciales, no lineales de primer orden[23]. El componente Motor es el encargado de evaluar estas ecuaciones que son obtenidas de las relaciones entre los elementos del modelo. Para esto los elementos del sistema deben ser evaluados en un orden determinado para resolver correctamente el modelo. Este orden de evaluación en el componente Motor presenta un problema de dependencia de datos que no permite implementar una paralelización en la evaluación de los elementos directamente.

Entre las posibilidades planteadas de paralelización en este componente, se consideró la creación de arboles de evaluación independientes donde los elementos no relacionados puedan ser evaluados en paralelo, sin embargo, el Motor de Evolución funciona de tal manera que cada iteración evaluada es incluida en el presentador de resultados actualizando la gráfica o tabla, esta actualización debe ser realizada por el hilo principal de la aplicación. Para lograr explotar las concurrencias en el Motor de Evolución se haría necesario entonces cambiar el funcionamiento de este componente, al ser la base de Evolución los cambios se extenderian por los demás componentes para que la presentación no sea para cada iteración sino para una corridea de simulación completa, por esta razón se ha decidido no modificar el funcionamiento del componente Motor.

6.4.3. Rutinas del Componente Comunes

La rutina *SacarToken* es la encargada de separar la ecuación de un elemento en un vector de cadenas de texto, separando los caracteres especiales como paréntesis, símbolos matemáticos y de puntuación, de las funciones o relaciones a otros elementos presentes en la función.

Estudiando esta rutina se descubre un algoritmo de ejecución ineficiente, no solo por la lógica presente en el procedimiento, sino también el manejo de vectores usado en el procedimiento en Delphi 7.

Para mejorar esta rutina se propone optimizar el algoritmo y usar las colecciones presentes en Delphi XE4, y aunque al terminar el procedimiento se debe convertir la colección a un vector para ser manejada por los demás procedimientos de Evolución las pruebas de rendimiento muestran una mejoría en el tiempo de ejecución.

Esta rutina separa la cadenas de texto que representa la ecuación de un elemento en un vector de cadenas de texto para que en una rutina diferente las cadenas que no son simbolos matemáticos sean reemplazadas por su valor correspondiente.

(Nivel/Parametro) === ['(' , 'Nivel' , '/' , 'Parametro' , ')']

La rutina sin optimizar, mostrada a continuación, consiste de un bucle inicial que separa los caracteres matemáticos de los demás valores en la ecuación, y el segundo bucle recorre el vector creado para eliminar los espacios vacios en los valores.

```
1  begin
2    NTokens:=0;
3    SetLength(Result,0);
4    //Inicia el primer bucle
5    For i := 1 to Length(Cadena) do
6      if Validar(Cadena[i]) then
7        begin
8          Inc(NTokens);
9          setlength(Result,NTokens+1);
10         Result[NTokens]:=Cadena[i];
11         Inc(NTokens);
12       end
13     else
14       begin
15         If Length(Result)<NTokens+1 then
16           begin
17             setlength(Result,NTokens+1);
18             Result[NTokens]:='';
19           end;
20         Result[NTokens]:=Result[NTokens]+Cadena[i];
21       end;
22   //Fin del primer bucle
23
24   //Segundo y Tercer bucle
25     j:=0;
26     while j < length(result) do
27     if ((trim(Result[j])='') and (not ConEspacios)) or
28        (((Result[j])='') and ConEspacios) then
29     begin
30       For i := j to Length(result)-2 do
31         Result[i]:=result[i+1];
32
33       setlength(Result,length(Result)-1);
34     end
35     else
36       inc(j);}
```

Notese como en esta rutina el uso de un array dinámico, construcción obsoleta, obliga al desarrollador a añadir mas lineas de código para modificar el tamaño de este haciendo el código mas dificil de leer y como la función **Trim** es usada en el segundo bucle para eliminar espacios vacios en los valores del vector lo cual puede ser mejorado para sencillamente no incluirlos en el primer bucle.

La siguiente rutina es la versión optimizada, en esta se usa una Lista que permite desarrollar un código mas legible y puede cambiar de tamaño facilmente, adicionalmente se controla la adición de caracteres vacios. Sin embargo, no es posible devolver una Lista al resto de la aplicación que espera un vector, por lo que se copian los valores de la lista a un vector antes de terminar la rutina.

```
begin
  valorActual := '';
  listaTokens := TStringList.Create;
  longitudCadena := Length(cadena);

  //Inicia el bucle
  for i := 1 to longitudCadena do
    begin
    if Validar(cadena[i]) then
    begin
      if (Length(valorActual) > 0) then
        listaTokens.Add(valorActual);
      if (cadena[i] <> ' ') then
        listaTokens.Add(cadena[i]);
      valorActual := '';
    end
    else
    if (cadena[i] <> ' ') then
      valorActual := valorActual + cadena[i];
    end;
   //Termina el bucle

   //Agrega el  ltimo token, si no es vacio
   if (valorActual <> '') then
     listaTokens.Add(valorActual);

  SetLength(Result, listaTokens.Count);

  //Empieza el bucle convertidor y libero listaTokens
  for i := 0 to listaTokens.Count - 1 do
    Result[i] := listaTokens[i];

  listaTokens.Destroy;
```

De esta manera se optimiza el funcionamiento de la rutina SacarToken respecto al tiempo de ejecución y no se hace necesaria la implementación del paralelismo puesto que el tiempo tomado en las pruebas de rendimiento para la nueva rutina presentan un tiempo de ejecución aceptable.

Tabla 13: Prueba 2 - Componente Comunes

Rutina	**Tiempo (s)**
Uprocesos::SacarToken	9,97
TEFuncion::GetNombreFuncionMayusculas	3,90
TLista::Cargar	1,73
Uprocesos::Validar	1,65
Umensaje::MensajePantalla	0,79
Uprocesos::SacarPar	0,69
TengaNombreE	0,60

En este componente para la rutina *SacarToken* se ha conseguido una disminución de 33,35 segundos de la versión anterior a 9,97 segundos al mejorar el algoritmo de la

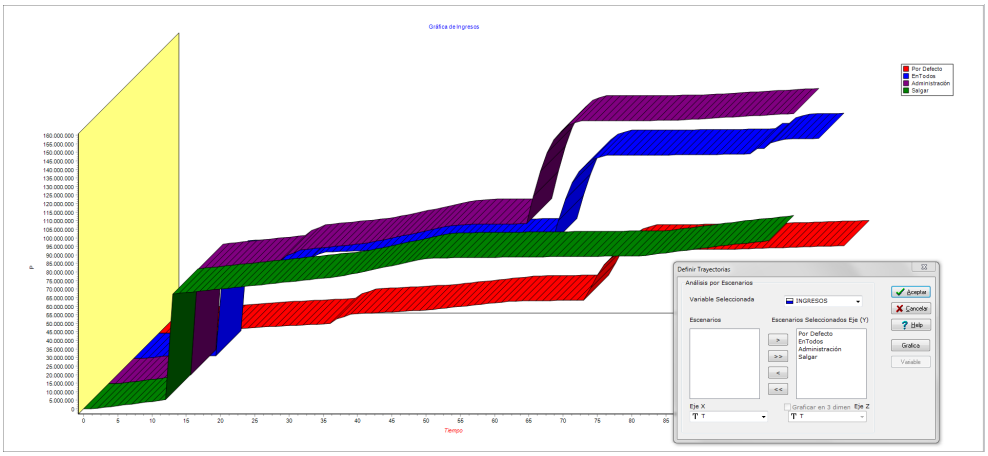
rutina y usar colecciones que han sido optimizadas para el compilador Delphi XE4, aproximadamente 3 veces mas rapido que la versión anterior.

6.4.4. Rutinas del Componente Sensibilidad

Este componente de Evolución presenta dos opciones para realizar Análisis de Sensibilidad al modelo: una permite la *Variación de Escenarios* y la otra por *Variación de Parámetros*.

El análisis por Variación de Escenarios permite generar una simulación en la cual se seleccionan los escenarios a simular del modelo y una variable a analizar, mientras el modelo mantiene su estructura cada escenario permite definir distintos valores a cada elemento del modelo de esta manera se puede estudiar el comportamiento de la variable para diferentes condiciones iniciales.
La Figura Figura 16 muestra una corrida de simulación para 4 escenarios, donde se analiza el comportamiento de una variable.

Figura 16: Gráfica de Variación de Escenarios

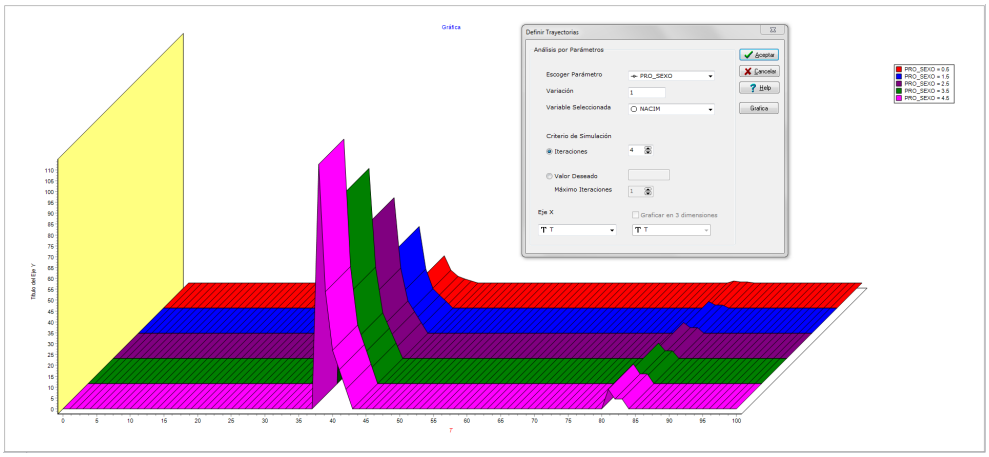


El análisis por Variación de Parámetros permite, de igual manera, generar una simulación en la cual se elige un parámetro que será modificado y una variable a análizar, el parámetro escogido es modificado en varias ocasiones para estudiar el comportamiento de la variable. Este se muestra en la Figura Figura 16

Las métricas tomadas para ambos componentes muestran los mayores tiempos en las rutinas que el Componente Sensibilidad usa del componente Motor, sin embargo como se explico anteriormente el Motor no ha sido modificado. Por esta razón se ha propuesto otra estrategia que permite explotar las concurrencias en el componente de Sensibilidad.

Estudiando el código y el funcionamiento de este componente se ha encontrado una

Figura 16: Gráfica de Variación de Parámetros



situación que se repite para ambas opciones de evaluación de escenarios y variación de parámetros. En ambas situaciones se usa un único motor para evaluar los diferentes escenarios o variaciones de parámetros, al terminar la evaluación de uno de estos el Motor es reiniciado y se le dan los datos para la siguiente iteración hasta terminar la presentación.

La solución que se ha planteado que permite explotar las concurrencias inherentes en este componente consiste en usar varias instancias del Motor, una para cada escenario o variación de parámetros, esto es posible al ejecutar cada uno de las motores en un hilo de ejecución diferente. Para esto fue necesario modificar varias rutinas del componente Sensibilidad que estaban desarrolladas para funcionar con un único motor.

Para lograr este objetivo fue necesario realizar un amplio trabajo de reestructuración y refactorización al código en el componente y sobreescribir varias rutinas que provienen del Componente Animador, el cual solo trabaja con un único motor.

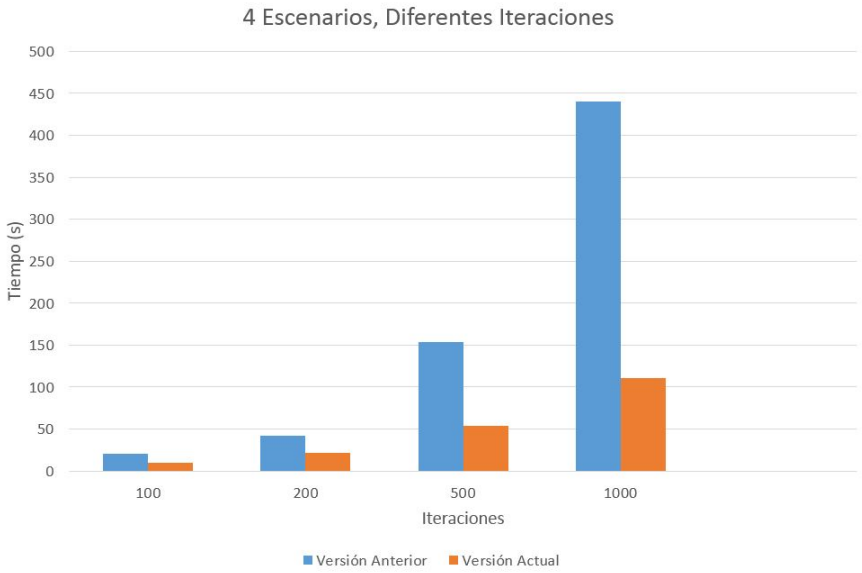
El motor funcionaba de la siguiente manera en el componente Sensibilidad, después de realizar los calculos para una iteración en la simulación se activa un evento que actualiza la presentación en pantalla, cuando termina el tiempo de simulación para un escenario o variación se reinicia para simular el siguiente.

En la versión concurrente, un motor por cada escenario o variación es iniciado, cada uno cuenta con una identificación que permite a cada evento de graficación enviado por los motores trabajar en el mismo graficador sin entrar en conflicto y actualizando cada uno su propia serie de datos.

Para comprobar la mejora obtenido al implementar el procesamiento por hilos en este componente se realizaron pruebas de rendimiento aumentando las íteraciones simuladas

de 100 a 1000 comparando los tiempos registrados de la versión anterior con la versión actual del componente de Sensibilidad para varias íteraciones.

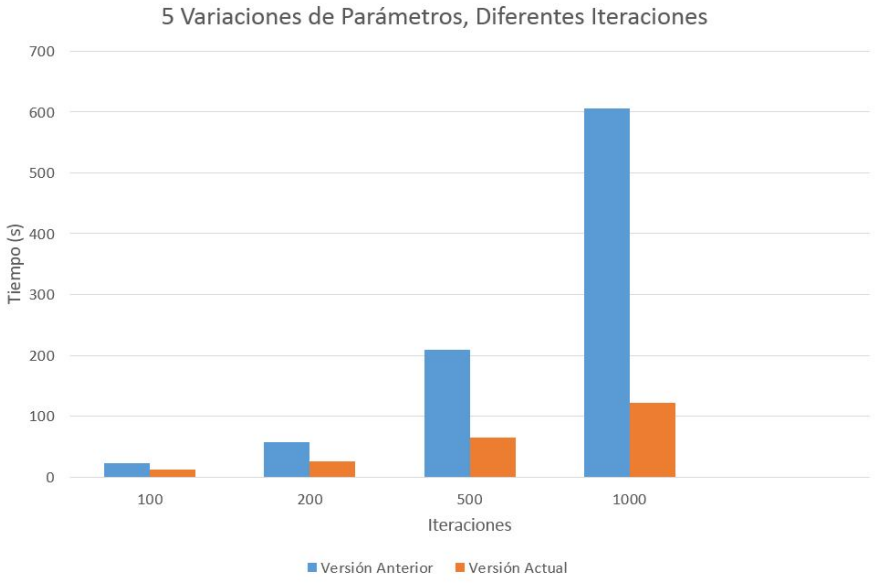
Figura 17: Comparación de Rendimiento en Variación de Escenarios



La primera prueba fue realizada a la Variación de Escenarios se mantiene constantes 4 escenarios del modelo en la Figura Figura 17 están representados los resultados obtenidos con el perfilador. Para 100 íteraciones no se nota una gran diferencia entre las dos versiones pero a medida que estas aumentan es notable la mejora en el rendimiento de la aplicación.

De igual manera se realizarón varias prueba en el modo de Variación de Escenarios, en estas se graficaron 5 variaciones a un parámetro del modelo aumentando las iteraciones desde 100 hasta 1000 íteraciones. Los resultados son satisfactorios y muestran una mejoría notable en el tiempo de ejecución.

Figura 18: Comparación de Rendimiento en Variación de Parámetros



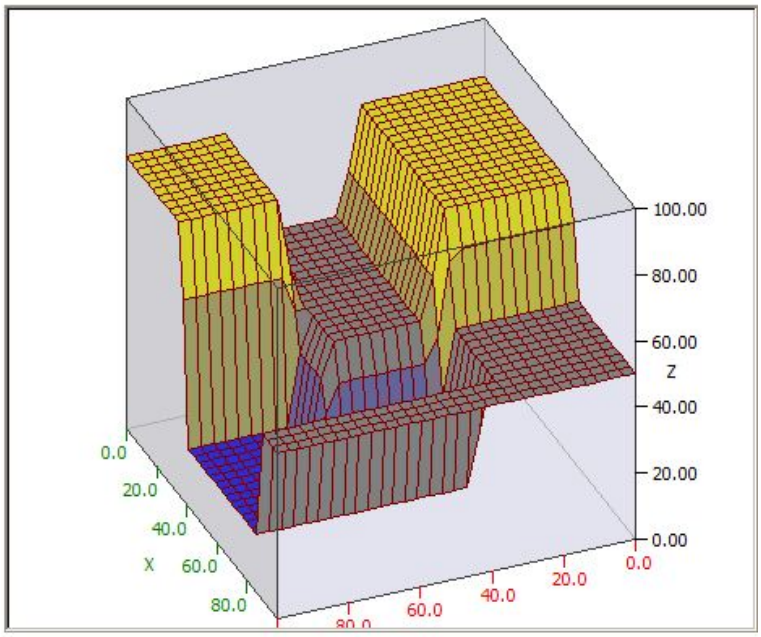
6.4.5. Rutinas del Componente FIS

El componente FIS es el resultado de desarrollar e integrar a Evolución un componente software que permitiera la implementación de Sistemas de Inferencia Difusa para la definición de variables de Dinámica de Sistemas en el diagrama de flujo-nivel. Este componente es una característica de Evolución y no está presente en otras herramientas de modelado y simulación de su tipo.[17]

Las rutinas detectadas con el perfilador son parte de una funcionalidad del componente FIS que permite al usuario ver el comportamiento del elemento creado a través de gráficas que muestran las tendencias de las salidas con respecto a las entradas bajo las condiciones que el modelador desee. En este caso en particular, se trata de una gráfica en tres dimensiones la cual se construye con dos entradas y una salida.

La solución que se ha implementado en este componente consiste, en primer lugar, en separar el calculo de los datos de la presentación en la gráfica, esto reduce las ordenes de actualización en pantalla al mínimo, y ha permitido reducir el tiempo de ejecución de cerca de 30 segundos a solo 6 segundos aproximadamente.

Figura 19: Gráfica en Tres Dimensiones en FIS



6.5. Lineamientos de Paralelización

Para conseguir la explotación de concurrencias en un ambiente de modelado y sistemas complejos, en este caso Evolución 4.5, se hizo necesario realizar cambios a los componentes de la herramienta y modificar las rutinas para permitir el acceso compartido a la interfaz de la aplicación. Nuestra experiencia ha sido recogida en unos lineamientos que ayudaran a paralelizar ambientes de modelado y sistemas complejos.

6.5.1. Perfilado de la Aplicación

Un perfilador es una herramienta que permite al desarrollador entender como el software desarrollado opera desde un punto de vista especifico que permite a este mejorar el rendimiento de la aplicación desarrollada.

El perfilado es típicamente usado cuando existe un problema evidente con el rendimiento de la aplicación, sin embargo, el uso regular de esta herramienta puede ayudar a descifrar problemas no tan notorios en el código.

Existen diferentes perfiladores tanto libres como comerciales en diferentes plataformas y lenguajes de desarrollo, para el caso de Evolución 4.5 se han usado dos tipos de perfiladores: AQtime de tipo instrumental, para medir y tener mas detalle de las rutinas de cada componente y optimizar o explotar las concurrencias, y ASMprofiler de muestreo, que permite ver los hilos en ejecución en la aplicación.

El uso de este tipo de herramientas es recomendado después de agregar alguna característca importante en la aplicación para identificar posibles problemas en la aplicación y corregirlos antes de entregar una nueva versión del producto. En Evolución 4.5 gracias al perfilado se han corregido problemas que persisten desde la versión 3.5.

6.5.2. Refactorización de Código

La refactorización es el proceso de reestructurar código existente sin cambiar su comportamient externo. Las ventajas de la refactorización incluyen mejorar la lectura del código y reducir la complejidad para mejorar la mantenibilidad del mismo. Esta refactorización es usualmente motivada por el descubrimiento de "olores" en el código, estos no necesariamente son problemas que impidan el funcionamiento de la aplicación pero indican una debilidad en el diseño que puede estar retrasando el desarrollo o aumentar el riesgo de errores o fallas a futuro.[9]

Antes de realizar la refactorización es necesario realizar pruebas de unidad a la herramienta, sin embargo, el mantenimiento realizado a la versión 4.5 de Evolución[13] en un proyecto anterior incluyó diversas pruebas de unidad que comprobaron un correcto funcionamiento individual de los rutinas.

La paralelización de Evolución requirió la refactorización de algunos componentes de la herramienta en mayor o menor medida. Aunque la refactorización tiene como objetivo en este proyecto, preparar el código para la explotación de concurrencias, se consiguieron otros beneficios que permitirán un mantenimiento mas completo en la herramienta.

El componente donde la refactorización ha sido mas notoria es el componente Sensibilidad, donde problemas como largas listas de parámetros y clases que dependen de la implementación de otras fueron encontradas.

```
1  DatosDeParametros              := Propiedades.DatosDeParametros;
2  //NumeroIteraciones            := Propiedades.NumeroIteraciones;
3  //PosicionVector               := Propiedades.PosicionVector;
4  //VariacionPorParametro        := Propiedades.VariacionPorParametro;
5  //ValorDeseado                 := Propiedades.ValorDeseado;
6  //TipoCriterio                 := Propiedades.TipoCriterio;
7  //ParametrosAnalizados         := Propiedades.ParametrosAnalizados;
8  //VariableAnalizada            := Propiedades.VariableAnalizada;
```

La sección de código anterior es un ejemplo de refactorización realizado en el componente Sensibilidad para el Análisis por Variación de Parámetros, esta lista es usada en cada iteración para manejar las condiciones especiales de la simulación, por cada variación representada era necesario pasar esta lista a los diferentes evaluadores usados repitiendola en varias clases, si en un futuro se quisiera modificar estos parámetros añadiendo o eliminando alguno se haría necesario modificar todos los lugares donde la

lista se encuentra repetida para asegurar un correcto funcionamiento, por esta razón se ha creado una estructura de datos que incluye los parámetros especiales de la lista y evita la repetición de código en las clases que la necesitan.

En general, la refactorización del código ha traído mejoras en la herramienta aun cuando las secciones modificadas solo eran aquellas candidatas a paralelización, sin embargo, es posible que en los demás componentes de la herramienta existan diferentes clases que requieran refactorización para hacer el código mas entendible y mantenible pensando en los futuros desarrolladores de Evolución.

6.5.3. Optimización de las Rutinas

La optimización del código es una serie de modificaciones que mejoran la calidad y la eficiencia. Un programa puede ser optimizado para conseguir un menor tamaño, consumir menos memoria, ejecutarse mas rapidamente, o para que realice menos operaciones de entrada y salida.[5] En el caso de Evolución, las evaluaciones de rendimiento realizadas a los componentes de la aplicación marcan a su vez el objetivo de la optimización que es una ejecución mas rápida de las rutinas.

Anteriormente se ha realizado una etapa de refactorización, sin embargo esto no significa que el código modificado sean necesariamente mas optimizado en cuanto a velocidad de ejecución, adicionalmente es posible encontrar rutinas que al ser optimizadas su mejora en el tiempo de ejecución sea importante de tal manera que no requieran de paralelización, por esta razón es importante estudiar la posibilidad de optimización en las rutinas antes de explotar las posibles concurrencias en el código.

Una optimización de este tipo se da en el componente Comunes en la rutina SacarToken de la unidad UProcesos, donde el algoritmo ha sido optimizado reduciendo la cantidad de bucles y manejando listas en lugar de vectores para facilitar la lectura del código. Con este cambio en el algoritmo se redujo el tiempo de ejecución a un punto en que no se considera necesaria la explotación de concurrencias para mejorar el rendimiento de la rutina.

Antes de explotar las concurrencias en este tipo de ambientes es importante partir de un código serial optimizado, sin embargo es importante considerar el tiempo invertido por el desarrollador en la optimización debido a que esta usualmente no añade características a la herramienta y si puede generar errores e impactar en la mantenibilidad de la aplicación.

6.5.4. Creación de Hilos y Uso de Librerías Multihilo

Los hilos son una forma relativamente sencilla para implementar multiples caminos de ejecución dentro de una aplicación. Cuando una aplicación no es concurrente tiene so-

lo un hilo de ejecución este debe responder a todos los eventos, actualizaciones de la interfaz y realizar todas las computaciones necesarias en la aplicación. En contraste, una aplicación que soporta concurrencia empieza con un hilo y adiciona mas según sea necesario para crear caminos de ejecución adicionales y cada uno de estos hilos ejecuta una rutina independientemente del hilo principal. Tener multiples hilos puede mejorar el rendimiento de la aplicación en sistemas multi-procesador.

Para explotar las concurrencias en las aplicaciones en sistemas multi-procesador se han desarrollado diferentes extensiones que permiten la adición del paralelismo en el código fuente sin tener que reescribirlo completamente. La extensión mas apmliamente usada que soporta programación en paralelo con memoria compartida es OpenMP para los lenguajes C, C++ y Fortran y es administrada por un gran grupo de dearrolladores de hardware y software.[20]

En el caso de Evolución 4.5 esta ambiente ha sido desarrollado en Delphi 7 un entorno de desarrollo de software que usa una versión moderna de Pascal llamada Object pascal. Para Delphi 7 no existe soporte por parte de OpenMP, por lo que se ha realizado una migración del código fuente a una versión actualizada llamada Delphi XE4, aunque de igual manera no es posible usar OpenMP en esta versión si hay la posibilidad de usar 2 librerías desarrolladas por terceros sin soporte oficial de Embarcadero (empresa desarrolladora de Delphi XE4), la primera librería multihilo **OmniThreadLibrary** se encuentra en un estado alpha de desarrollo[10], la segunda librería es **AsyncCalls** ha sido desatendida desde el 2011 por su desarrollador.

Para aprovechar las concurrencias en Evolución 4.5 inicialmente la estrategia consistió en introducir el paralelismo en la aplicación mediante las construcciones ofrecidas por OmniThreadLibrary y AsyncCalls. Desafortunadamente, este esfuerzo no dio los resultados esperados debido en parte a problemas de localidad en los procedimientos en Evolución y a problemas introducidos por la librería que resultan en errores inesperados en la ejecución. Adicionalmente, el soporte de OmniThreadLibrary es realizado por un solo desarrollador, Primož Gabrijelčič, por lo que la corrección de errores y mantenimiento a la librería será mas lento en comparación con OpenMP en C, C++ Y Fortran.

En bucles sencillos la librería funciona correctamente, pero cuando los objetos a manejar presentan herencias extensas y son compuestos de otros objetos se presentan caidas del sistema por lo que no es posible usar las librerías de paralelización desarrolladas para Delphi y es necesario usar la clase Thread encargada de la creación de hilos en el lenguaje para explotar las concurrencias en la aplicación.

Es sabido que muchos lenguajes soportan la creación de hilos, en el caso de Delphi una aplicación normal maneja un solo hilo de esta manera todos los objetos de la interfaz acceden a sus propiedades y ejecutan sus métodos en este hilo principal. Por lo tanto, para acelerar la velocidad de procesamiento en la aplicación es posible crear uno

o mas hilos «secundarios».

La explotación de concurrencias en el componente Sensiblidad requirió el uso de varios hilos, donde en cada hilo trabaja un Motor de simulación independiente, sin embargo todos los hilos deben actualizar la interfaz gráfica por lo que deben ser sincronizados al momento de acceder a los recursos comunes como eran las gráficas y tablas.

6.5.5. Separación de Simulación y Presentación de Resultados

En general, los ambientes de modelado y simulación con dinámica de sistemas poseen un Motor que permite la evaluación del modelo cuyos resultados son usados para presentar mediante gráficas y tablas el comportamiento de los elementos que componen el modelo. Herramientas de este tipo son Vensim[27] y Evolución[1].

La Simulación del modelo en Vensim es realizada inicialmente para generar un «dataset» o conjunto de datos que es usado por las gráficas y tablas para generar la representación del comportamiento de los elementos en el modelo. En Evolución el Motor tiene la misma función que permite evaluar el modelo, pero por cada iteración el Motor actualiza la interfaz gráfica, esto se puede entender como una «falsa» generación de datos en tiempo real.

El enfoque de Evolución desde el punto de vista de la explotación de concurrencias, no es muy eficiente, debido a que la actualización de la interfaz debe ser realizada por el hilo principal de la aplicación, por esta restricción el Motor debe detener el calculo de las iteraciones siguientes mientras actualiza la gráfica o tabla, de igual manera cuando varios motores trabajan simultáneamente estos deben sincronizarse para actualizar la interfaz. Sin embargo, este tipo de Motor en Evolución permite que la simulación sea pausada para realizar cambios al modelo y ver los efectos de esta modificación en la misma presentación.

Es recomendable entonces un ambiente que separe la simulación del modelo de la presentación de resultados, para explotar las concurrencias presentes en los modelos y reducir las actualizaciones de la interfaz; pero que de igual manera conserve la capacidad de análisis de resultados y modificación de los mismos que presenta Evolución.

Capítulo 7

Conclusiones

Evolución es un ambiente desarrollado por más de 20 años, mejorado mediante trabajos de pregrado y maestría realizados en el Grupo SIMON de Investigación. Durante estos años de desarrollo nuevas características y funcionalidades han sido añadidas al ambiente por diferentes programadores, sin embargo, el paradigma de implementación no consideró la explotación de concurrencias inherentes en Evolución, por esta razón una evaluación de rendimiento asistida por herramientas de perfilado ha sido realizada para identificar las posibilidades de aceleración manteniendo la experiencia del usuario y la coherencia en las simulaciones con dinámica de sistemas.

La primera etapa del proyecto consiste en la Extracción de la Arquitectura de la Aplicación, esto fue posible siguiendo el proceso de extracción documentado en versiones anteriores de Evolución y el uso de herramientas de análisis de código estático que han permitido observar las relaciones de dependencia entre unidades de código y diagramas de clases de módulos escasamente documentados. Este proceso ha permitido la identificación de los componentes y sus múltiples relaciones para representar una arquitectura del software de la versión 4.5 de Evolución.

En la segunda etapa han sido realizadas evaluaciones de rendimiento a la aplicación para encontrar las zonas críticas que puedan ser candidatas a explotación de concurrencias, para lograr este objetivo se ha usado un perfilador instrumental que proporciona métricas de gran detalle de las rutinas evaluadas con un grado de error pues debe modificar el código para la toma de tiempos. Para reducir este margen las evaluaciones han sido realizadas a los componentes de manera individual y escogiendo como candidatas a explotación las rutinas con un mayor tiempo de ejecución en cada componente.

Terminada la etapa de evaluación a los componentes de Evolución, se ha generado una estrategia para llevar a cabo la optimización y explotación de concurrencias en las rutinas. Con la intención de ampliar las facilidades de paralelización se ha realizado una migración de Delphi 7 a una versión superior Delphi XE4, brindando la posibilidad de usar librerías de paralelización creadas por desarrolladores independientes al no existir un soporte al paralelismo diferente a la creación manual de hilos por parte de Embarcadero*. Aunque la versión Delphi XE4 presenta características ausentes para la versión Delphi 7, persisten problemas como son la dependencia al framework y la ausencia de desarrollo multiplataforma.

Las soluciones implementadas en las diferentes rutinas candidatas han sido desarrolladas siguiendo la estrategia propuesta, a continuación se da una breve explicación del trabajo realizado en los diferentes componentes.

- **Animador**: Un hilo externo ha sido implemento para realizar un trabajo asíncrono de actualización de la interfaz permitiendo al hilo principal continuar la simulación del modelo y realizar una presentación de resultados más fluida.
- **Comunes**: El algoritmo serial en la rutina de mayor tiempo de ejecución ha sido optimizado, reduciendo la cantidad de bucles necesarios para su funcionamiento.
- **Motor**: No ha sido posible modificar y explotar las concurrencias en este componente, por la implementación del mismo y su interdependencia con los demás componentes.
- **Sensibilidad**: Han sido utilizadas múltiples instancias del Motor cada una sobre un hilo de ejecución independiente para ejecutar las diferentes variaciones de escenarios o de parámetros, para lograrlo fue necesario un proceso de refactorización y modificación de procedimientos claves.
- **FIS**: La grafica en tres dimensiones realizada por el componente ha mejorado su tiempo de ejecución al separar este proceso en una etapa de obtención de datos y una de presentación, reduciendo de esta manera las actualizaciones a la interfaz gráfica.

Con el fin de demostrar que las modificaciones introducidas a los diferentes componentes y rutinas candidatas de Evolución han mejorado el rendimiento de la aplicación por lo que se han realizado pruebas de rendimiento bajo las mismas condiciones iniciales comprobando de esta manera una disminución en los tiempos registrados inicialmente. La tabla a continuación muestra el tiempo y porcentaje de mejora respecto a las métricas iniciales.

Tabla 14: Comparación General

	Tiempo Anterior (s)	**Tiempo Actual (s)**	**Mejora**
Animador	2954,4	354,21	8x
Comunes	62,14	20,99	3x
Sen - Escenarios	439,826	110,489	4x
Sen Parmetros	605,682	121,374	5x
FIS	38,58	8,54	4.5x

La explotacin de concurrencias en este tipo de aplicaciones ha demostrado ser una medida exitosa para mejorar el rendimiento, esto se traduce en una mejor experiencia y comodidad para el usuario modelador.

Finalmente, basados en las experiencias con Evolución 4.5 se han desarrollado una serie de lineamientos para la explotación de concurrencias para conseguir de esta manera un mejor rendimiento en ambientes de modelado y simulación con dinámica de sistemas en general.

- Perfilar la aplicación, para identificar las secciones del código que más se beneficiarían de la explotación de concurrencias.
- Refactorización por simplicidad, la paralelización es propensa a errores por lo que entre más complejo sea el código más difícil se hace encontrar y prevenir el uso de recursos compartidos y llamadas a funciones que no son seguras cuando se llaman desde los hilos.
- Optimización en tiempo de ejecución del código serial, de esta manera podemos mejorar el rendimiento de la aplicación y utilizar la explotación de concurrencias donde realmente sea útil y necesario.
- Creación de hilos, cuando la única opción para explotar las concurrencias en el lenguaje de desarrollo es la creación manual de hilos se deben identificar los recursos compartidos y el código que puede ser ejecutado en los hilos.
- Separación de simulación y presentación de resultados, debido a que la presentación en pantalla debe realizarse únicamente bajo el hilo principal, es recomendable entonces reducir las interacciones con la interfaz y simular el modelo de manera independiente para permitir mayores oportunidades de explotación de concurrencias.

Sobre el trabajo realizado en Evolución 4.5 es posible concluir que el desarrollo a través de proyectos de grado ha permitido otorgar una herramienta robusta y funcional a la comunidad de dinámica de sistemas, y gracias a la identificación y explotación de concurrencias desarrolladas en este proyecto ha conseguido la ejecución eficiente de los procedimientos de la herramienta, sin embargo, se han identificado problemas en la herramienta como son la variedad de desarrolladores sin una metodología de desarrollo definida, la poca documentación realizada a los procedimientos y la ausencia de una arquitectura estructurada y modular que guie el desarrollo de nuevas características en la herramienta, han hecho que el código de Evolución sea complejo, difícil de mantener y trazar para encontrar errores.

De continuar esta forma de desarrollo los problemas en la herramienta se harán más notorios y difíciles de solucionar, por lo que se hace necesario cambiar la forma de desarrollo de Evolución de una contribución académica a un producto software cuya experiencia de usuario sea satisfactoria y su coste de mantenimiento y desarrollo no se salga de lo estimado. Para lograr esto es recomendable desarrollar la herramienta en un lenguaje de desarrollo multiplataforma y ampliamente usado como C++ y definir

una arquitectura modular que permita la inclusión de nuevas funcionalidades y defina claramente las relaciones entre los componentes del software, así como generar políticas de aprobación de nuevas características y un manejo apropiado del código fuente y su documentación.

Cabe señalar que la creación de esta herramienta multiplataforma y sus políticas de desarrollo conllevan un tiempo considerable, por lo que se hace necesario no dejar de lado el mantenimiento del ambiente actual Evolución 4.5 de forma que se pueda tener una herramienta disponible entre la comunidad de dinámica de sistemas y mantener la base de usuarios.

Capítulo 8

Recomendaciones

Recomendaciones para Evolución 4.5

- Actualizar la documentación del código fuente a la versión 4.5 y realizar comentarios que describan el funcionamiento de las rutinas.
- Realizar un trabajo de refactorización y optimización en todos los componentes de Evolución y realizar evaluaciones estáticas de código.
- Eliminar de la aplicación el componente FlatStyle y otros componentes terceros sin soporte y abandonados por sus desarrolladores, y reemplazar sus funciones con construcciones internas que son incluidas en Delphi XE4.
- Generar un repositorio privado que permita controlar los cambios y nuevas características introducidas a la herramienta por los diferentes .
- Realizar un mantenimiento periódico a Evolución independiente del desarrollo mediante proyectos de grado, en especial al Componente FIS.

Recomendaciones para la nueva Versión de Evolución

- Desarrollar una arquitectura modular que guíe el desarrollo, defina una estructura para la aplicación y relaciones claras entre sus componentes.
- Utilizar métricas de calidad para el desarrollo y aceptación de nuevas características en la herramienta.
- Tener en cuenta los logros obtenidos por Evolución hasta la fecha y las sugerencias de los modeladores para proporcionar una herramienta que compita con los ambientes de pago que se encuentran en el mercado.
- Consideran la multiplataforma y el uso eficiente de los recursos computacionales a la hora de escoger un lenguaje de desarrollo, por lo cual se recomienda C++ para la implementación de la nueva versión.

Referencias

1. Andrade, H. Lince, E. Hernandez, A. Monsalve, A. Evolución: Herramienta software para modelado y simulación con Dinámica de Sistemas. Bucaramanga : Universidad Industrial de Santander, Grupo SIMON.

2. Barney, Blaise. Introducción a la computación en paralelo. Introduction to Parallel Computing. [Online] 2013. [Cited: 08 03, 2013.] `https://computing.llnl.gov/tutorials/parallel_comp/`.

3. Barragán, O. Gómez, U. Andrade, H. García, C. Propuesta de un modelo de simulación de sistemas de producción de ganadería bovina para la investigación integral, un enfoque sistémico. Bucaramanga : Universidad Industrial de Santander, Grupo SIMON, 2002.

4. Bass, L. Clements, P. Kazman, R. Arquitectura del Software en Practica, "Software Architecture in Practice". s.l. : Addison Wesley, 2003. 0-321-15495-9.

5. Bentley, Jon. Escribiendo Programas Eficientes "Writing Efficient Programs". s.l. : Prentice-Hall Software Series, 2008.

6. Diaz, N. Andrade, H. Serrano, G. Osorio, P. Modelo para la toma de decisiones en la inversión pública municipal con el presupuesto de libre inversión y orientado al mejoramiento del desarrollo local, un enfoque sistémico ADMONSOFT 2.0. Bucaramanga : Universidad Industrial de Santander, 2012.

7. Eldean. ESS-Model. ESS-Model: UML reversing tool. [Online] Eldean. [Cited: 06 30, 2014.] `http://essmodel.sourceforge.net/index.html`.

8. Embarcadero. TThread Class. Documentación Oficial de Embarcadero Delphi. [Online] Embarcadero. [Cited: 06 30, 2014.] `http://docwiki.embarcadero.com/Libraries/XE6/en/System.Classes.TThread`.

9. Fowler, M. Beck, K. Brant, J. Opdyke, W. Refactorización: Mejorar el diseño de código existente. Refactoring: Improving the Design of Existing Code". s.l. : Object Technology, 2002.

10. Gabrijelčič, Primož. OmniThreadLibrary: The ultimate Delphi threading library. [Online] Primož Gabrijelčič, 2014. [Cited: 06 30, 2014.] `http://otl.17slon.com`.

11. Garavito, J. Andrade, H. Gomez, U. Vásquez, C. Ecogranja, Heramienta software para el diseño de granjas integrales agropecuarias, un enfoque dinámico sistémico. Bucaramanga : Universidad Industrial de Santander, Grupo SIMON, 2012.

12. Hausladen, Andreas. AsyncCalls. [Online] Andreas Hausladen, 2011. [Cited: 06 30, 2014.] http://andy.jgknet.de/blog/bugfix-units/.

13. Hernandez, A. Monsalve, A. Lince, E. Andrade, H. Mantenimiento del Software Evolución 4.0. Bucaramanga : Universidad Industrial de Santander, Grupo SIMON 2014.

14. Interface, Message Passing. Interfaz de Paso de Mensajes. [Online] MPI, 2013. [Cited: 08 23, 2013.] http://www.mcs.anl.gov/research/projects/mpi.

15. Labs, Epina Software. SDL Delphi Component Suite. [Online] Epina Software Labs, 2014. [Cited: 06 30, 2014.] http://www.lohninger.com/sdlindex.html.

16. Lanusse, Andreano. Razones para migrar a Delphi XE. Reasons to Migrate to Delphi XE". s.l. : Embarcadero, Embarcadero, 2010.

17. Machado, G. Gonzalez, C. Andrade, H. Componente de Sistema de Inferencia Difusa (FIS) para Evolución 3.5. Bucaramanga : Universidad Industrial de Santander, Grupo SIMON, 2006.

18. Moreno, J. Andrade, H. Diseño de una Arquitectura para un Entorno de Modelamiento - Simulación y Creación de un proceso para su desarrollo por una comunidad. Bucaramanga : Universidad Industrial de Santander, Grupo SIMON, 2006.

19. Mussche, Andre. AsmProfiler. AsmProfiler is a free and open source profiler for 32 bit Windows applications. [Online] AsmProfiler, 2014. [Cited: 06 30, 2014.] https://code.google.com/p/asmprofiler/.

20. OpenMP. Introduction to OpenMP. [Online] 2013. [Cited: 08 23, 2013.] http://openmp.org/wp/.

21. Ruiz Garcia, F. Silva Olivares, R. Andrade Sosa, H. Vasquez, C. Herramienta software para el aprendizake del manejo de un sistema de producción de ganadería bovina mediante simulación con dinámica de sistemas, un enfoque sistémico SIPROB 2.0. Bucaramanga : Universidad Industrial de Santander, Grupo SIMON, 2008.

22. SmartBear. Application Performance - AQtime. AQtime. [Online] SmartBear, 2014. [Cited: 06 30, 2014.] http://smartbear.com/products/.

23. Society, System Dynamics. What is System Dynamics? Qué es la Dinámica de Sistemas? [Online] System Dynamics Society, 2013. [Cited: 08 29, 2013.]

24. Software, Steema. TeeChart for VCL. [Online] Steema Software, 2014. [Cited: 06 30, 2014.] http://www.steema.com.

25. Systems, ISEE. STELLA Modeling & Simulation Software. [Online] ISEE Systems, 2013. [Cited: 08 29, 2013.] http://www.iseesystems.com/softwares/Education/StellaSoftware.aspx.

26. Tools, Model Maker. ModelMaker Tools. ModelMaker Tools: Delphi and C# .Net Refactoring & UML modeling. [Online] Model Maker Tools, 2014. [Cited: 06 30, 2014.] http://www.modelmakertools.com.

27. Vensim. Industrial Strength Simulation Software. [Online] Vensim, 2013. [Cited: 08 29, 2013.] http://vensim.com/.

Anexos

Métricas del Componente Animador

Este anexo contiene las métricas tomadas para el componente Animador en una simulación normal de un modelo de prueba con 1000 iteraciones. Se incluyen las métricas para el modo Gráfica y para el modo Tabla tomadas para la primera fase del proyecto y finalmente las métricas después de la explotación de concurrencias.

Tabla 15: Métricas Componente Animador - Modo Gráfica - Antes 1/2

Nombre de la Rutina	Tiempo	Tiempo con Hijos
TManejadorResultados::EnviarMensaje	2726,21	2726,22
TAnimador::PasoMotor	217,4	232,65
TGraficador::SetVerBarra	3,05	3,05
TGrafica::MostrarPropiedadesControl	2,17	2,23
TAnimador::setDatos	1,99	1,99
TManejadorResultados::CondicionesSimulacion	1,08	1,08
TTabla::Animar	0,61	0,66
TGrafica::Animar	0,59	0,64
TFormAnimador::Cargar	0,29	0,57
TFormAnimador::ColocarValoresIniciales	0,2	0,32
TTabla::Destroy	0,11	0,11
TListaPropiedadesSerie::Get	0,09	0,09
TFormAnimador::Destroy	0,07	0,28
TFormAnimador::Create	0,06	0,41
TVistaGraficador::Destroy	0,05	0,2
TTabla::Cargar	0,05	0,08
TGraficador::Cargar	0,05	0,28
TGraficador::CrearVista	0,04	0,05
TGrafica::Destroy	0,03	0,03
TFormTrayectoriasGrafica::Create	0,03	0,03
TAnimador::IteracionMotor	0,03	2719,97
CargarDatosUbicacion	0,03	0,03
TGraficador::SetParent	0,02	0,02
TFormAnimador::ColocarNombreEnBarraTitulo	0,02	0,02

Tabla 16: Métricas Componente Animador - Modo Gráfica - Antes 2/2

Nombre de la Rutina	Tiempo	Tiempo con Hijos
TVistaGraficador::WParent	0,01	0,01
TFormAnimador::SetDatos	0,01	2,01
TGrafica::Cargar	0,01	0,02
TGraficador::Create	0,01	0,01
TGrafica::CrearSerie	0,01	0,01
ColocarInformacionAnimador	0	0,01
TVistaGraficador::CambiarModo	0	0
TGrafica::MouseMove	0	0
TAnimador::Create	0	0,01
ParaCadaControlQueImplementeCustomAnim	0	4,35
TFormGraficador::CrearObjetos	0	0,01
TGrafica::Create	0	0
TManejadorResultados::GetFormAnimadorActivo	0	0
EstablecerPropiedadesAEje	0	0
TFCondiciones::FormCreate	0	0
TFCondiciones::FormDestroy	0	0
hacer	0	1,3
TManejadorResultados::CargarDesdeEspacio	0	2,86
TVistaGraficador::Animar	0	1,3
TManejadorResultados::ActivarFormAnimador	0	0
TManejadorResultados::VerificarVentana	0	0
TPropiedadesSerie::Create	0	0
TManejadorResultadosDisenables::VerificarVentana	0	0
TGraficador::Animar	0	1,3
TFormAnimador::ReenviarMensaje	0	2726,14
TManejadorResultadosDisenables::EnviarMensaje	0	2726,22
TVistaGraficador::Create	0	0,02
TAnimador::GetControl	0	0
TAnimador::ReenviarMensaje	0	2726,11
TTabla::Iniciar	0	0,01
TAnimador::CrearObjetos	0	0

Tabla 17: Métricas tomadas del Componente Animador - Modo Tabla - Antes

Nombre de la Rutina	Tiempo	Tiempo con Hijos
TAnimador::PasoMotor	210,4	232,65
TAnimador::setDatos	1,6	1,6
TTabla::Animar	1,46	1,51
TManejadorResultados::EnviarMensaje	1,42	1,42
TGrafica::Animar	0,94	0,97
TFormAnimador::Cargar	0,26	0,5
TFormAnimador::ColocarValoresIniciales	0,2	0,33
TTabla::Destroy	0,1	0,1
TListaPropiedadesSerie::Get	0,07	0,07
TGraficador::SetVerBarra	0,06	0,06
TFormAnimador::Destroy	0,06	0,27
TFormAnimador::Create	0,05	0,41
TVistaGraficador::Destroy	0,05	0,19
TTabla::Cargar	0,04	0,05
TGraficador::CrearVista	0,04	0,05
TGrafica::Destroy	0,04	0,04
TGraficador::SetParent	0,04	0,04
TGraficador::Cargar	0,03	0,24
CargarDatosUbicacion	0,02	0,02
TTabla::getListaVariables	0,01	0,01
TFormAnimador::ColocarNombreEnBarraTitulo	0,01	0,01
TVistaGraficador::CambiarModo	0,01	0,01
TVistaGraficador::WParent	0,01	0,01
TTabla::Iniciar	0,01	0,02
TGrafica::Cargar	0,01	0,02
TGraficador::Create	0,01	0,01
IndiceDe	0,01	0,01
TPropiedadesSerie::Cargar	0,01	0,01
TAnimador::Destroy	0,01	0,15
TGrafica::CrearSerie	0,01	0,01
TFormAnimador::SetDatos	0,01	1,62
TGrafica::CompactarPropiedades	0,01	0,01

Tabla 18: Métricas del Componente Animador - Modo Gráfica - Después

Nombre de la Rutina	Tiempo	Tiempo con Hijos
TAnimador::PasoMotor	218,52	232,66
TAnimador::MostrarIteracion	54,00	54,00
TAnimador::IteracionMotor	19,99	19,99
TFormAnimador::SetDatos	18,51	20,01
TManejadorResultados::EnviarMensaje	9,77	9,77
TGrafica::Animar	6,81	8,43
TFormTrayectoriasGrafica::Seleccionar	5,80	5,85
TTabla::Animar	2,78	5,70
TManejadorResultados::CondicionesSimulacion	2,72	2,73
TGrafica::MostrarPropiedadesControl	2,14	8,02
TTabla::getListaVariables	1,80	1,80
TGrafica::MostrarPropiedadesAnimacion	1,62	1,63
TListaPropiedadesSerie::Get	1,54	1,54
TAnimador::setDatos	1,49	1,49
TGraficador::SetVerBarra	1,48	1,48
TCustomAnim::setNombre	1,45	1,45
TGrafica::getListaVariables	1,22	1,22
TManejadorResultadosDisenables::EnviarMensaje	0,76	10,53
TListaFormAnimador::CargarElemento	0,46	19,16
TAnimador::MostrarIteracion	0,37	9,85
TFormAnimador::Cargar	0,25	1,84
TFormAnimador::ColocarValoresIniciales	0,22	1,35
TFormAnimador::TFormAnimador	0,06	1,44
TTabla:: TTabla	0,04	0,04
TFormAnimador:: TFormAnimador	0,03	0,11
TGrafica:: TGrafica	0,03	0,03
TVistaGraficador:: TVistaGraficador	0,03	0,10
Utiposgraficastablas::Colores	0,02	0,02
Ubuscador::IndiceDe	0,02	0,02
TFormAnimador::ColocarNombreEnBarraTitulo	0,02	0,02
TTabla::Cargar	0,02	0,03
Uprocesosestandaranim::CargarDatosUbicacion	0,02	0,02
TGraficador::CrearVista	0,02	0,02
TGrafica::CompactarPropiedades	0,02	0,02
Utiposgraficastablas::Indices	0,02	0,04
TGraficador::SetParent	0,02	0,02
TGraficador::Cargar	0,02	1,59
TFormTrayectoriasGrafica::TFormTrayectoriasGrafica	0,01	0,01
TAnimador::TAnimador	0,01	0,03

Métricas del Componente Motor

Este anexo contiene las métricas tomadas para el componente Motor en este se encuentran todas las rutinas que han sido tomadas por el perfilador en una simulación normal del modelo con 1000 iteraciones.
Se han registrado las métricas tomadas al simular los dos modelos disponibles para evaluación.

Tabla 19: Mediciones tomadas al Motor con el Modelo de Natalia Diaz

Nombre de la Rutina	Tiempo	Tiempo con Hijos
TMotorEvolucion::Iteracion	103,04	240,04
TEvaluador::EvaluarExpresion	64,65	64,97
TElementoEvaluacionFis::Evaluarse	32,13	69,11
TEvaluador::ReemplazarFuncion	18,57	35,02
TElementoEvaluacion::TransformarDefiniciones	10,02	19,09
ReemplazarParaElemento	8,39	8,92
TEvaluador::ReemplazarFunciones	1,51	36,02
TElementoEvaluacionConVariasSalidas::SetDefinicion	1,07	1,07
TEvaluador::ResolverEcuacion	0,48	85,77
TMotor::SetEstadoSimulacion	0,39	0,39
EliminarPos	0,32	0,32
TEvaluador::FCFunction	0,32	0,32
TElementoEvaluacion::GetDimension	0,26	0,26
TEvaluador::FSumaRetardo	0,23	0,23
TElementoEvaluacion::Evaluarse	0,17	68,46
TListaElementosEvaluacion::Get	0,15	0,15
TEvaluador::TEvaluador	0,15	0,15
TManejadorEvaluacion::Evaluar	0,11	137,93
TElementoEvaluacion::GetValor	0,10	0,10
TManejadorEvaluacion::CrearElementoEvaluarCon	0,07	1,16
TElementoEvaluacionTabla::EvaluarTabla	0,06	0,07
TElementoEvaluacionNivel::EulerCalcularFlujos	0,06	0,11
TElementoEvaluacion::SetValor	0,05	0,05

Otras rutinas con tiempos menores a 0,01 segundos no fueron incluidas en el anexo.

Tabla 20: Mediciones tomadas al Motor con el Modelo de Barragán/Gómez

Nombre de la Rutina	Tiempo	Tiempo con Hijos
TMotorEvolucion::Iteracion	104,6	245,5
TEvaluador::ReemplazarFuncion	67,62	97,41
TEvaluador::EvaluarExpresion	55,37	55,38
TElementoEvaluacion::TransformarDefiniciones	36,43	67,13
ReemplazarParaElemento	28,46	30,14
TListaElementosEvaluacion::Get	8,07	8,07
TElementoEvaluacion::Preparar	3,69	7,34
TEvaluador::ReemplazarFunciones	1,46	98,2
TMotorEvolucion::CrearListaOrdenDeEvaluacion	1,31	1,31
AgregarAlimentador	1,11	1,11
TElementoEvaluacionNivel::Preparar	1,1	2,74
TElementoEvaluacion::SetDimension	1,1	1,46
TEvaluador::ResolverEcuacion	0,75	125,4
TElementoEvaluacion::ArreglarDefinicion	0,73	0,73
EliminarPos	0,72	0,72
TElementoEvaluacion::GetValor	0,4	0,4
TElementoEvaluacion::GetDimension	0,4	0,4
TElementoEvaluacionRetardo::PrepararRetardo	0,39	3,04
TElementoEvaluacion::Destroy	0,38	0,38
TCustomListaElementosEvaluacion::Get	0,38	0,38
TCustomListaElementosEvaluacion::Create	0,37	0,37
TCustomElementoEvaluacion::SetDimension	0,37	0,37
TElementoEvaluacionParametro::Evaluarse	0,36	3,35
TElementoEvaluacionRetardo::Evaluarse	0,25	3,11
TElementoEvaluacion::Evaluarse	0,21	188,41
TEvaluador::FSumaRetardo	0,2	0,2
TElementoEvaluacionRetardo::GetValor	0,19	0,19
TManejadorElementos::VerificarElemento	0,19	7,18
TElementoEvaluacion::Asignar	0,19	3,31
TElementoEvaluacion::SetNombre	0,18	0,18
TElementoEvaluacion::Resolver	0,18	0,76
TElementoEvaluacionNivel::Destroy	0,18	0,18
TodosSubmodelosEstanDefinidos	0,18	0,18
TElementoEvaluacion::SetDefinicion	0,18	0,91
TCustomElementoEvaluacion::QuitarValorInicial	0,18	0,18
TMotor::SetEstadoSimulacion	0,06	0,06
TManejadorElementos::CargarDesdeEspacio	0,02	0,02
TElementoEvaluacionNivel::EulerCalcularFlujos	0,01	0,02
TElementoEvaluacionNivel::GetValor	0,01	0,01
TElementoEvaluacion::GetDefiniciones	0,01	0,01

Métricas del Componente Editor

Este anexo contiene las métricas tomadas para el componente Editor usando el perfilador AQtime, solo se registran las rutinas cuyo tiempo o tiempo con hijos son mayor o igual a 0,01 segundos. La prueba consistió en cargar un Modelo de Flujo-Nivel contenido en el modelo de Barragán/Gómez con mas de 300 elementos desde archivo.

Tabla 21: Métricas tomadas del Componente Editor

Nombre de la Rutina	Tiempo	Tiempo con Hijos
ULineas::AutoPintarse	0,47	0,47
UInterfacesDeDibujoPintarLetra	0,17	0,17
UElementosGraficos::VariablesGraficasDe	0,15	0,23
TFormForrester::EnviarMensaje	0,12	0,13
TListaElementos::VerificarPosicionNombre	0,1	0,33
UElementosGraficos::DibujarSeleccion	0,1	0,1
UElementosGraficos::CambiarPosicion	0,06	0,25
TLinea::PtInLine	0,05	0,05
TPropiedadesGraficas::_AddRef	0,05	0,05
DibujaNube	0,04	0,04
TPropiedadesGraficas::_Release	0,04	0,04
TFormForrester::Destroy	0,04	0,04
TPintor::Create	0,03	0,03
dibujar_parametro	0,02	0,05
TEditor::CalcularHojas	0,02	0,02
CargarVariablesGraficas	0,02	0,02
DibujarNombre	0,01	0,17
TFlujo::Pintar	0,01	0,14
TNube::Pintar	0,01	0,06
TParametro::Pintar	0,01	0,04
TFormForrester::FormCreate	0,01	0,04
TRetardo::Pintar	0,01	0,04
Dibujar_auxiliar	0,01	0,03
Dibujar_Flujo	0,01	0,02
TPintor::BorrarLienzo	0,01	0,01
DentroNube	0,01	0,01

Métricas del Componente Influencias

Este anexo contiene las métricas tomadas para el componente Influencias usando el perfilador AQtime, solo se registran las rutinas cuyo tiempo o tiempo con hijos son mayor o igual a 0,01 segundos. La prueba consistió en cargar un Modelo de Influencias que se encuentra en el Modelo de Natalia Diaz desde archivo.

Tabla 22: Métricas tomadas al Componente Influencias

Nombre de la Rutina	Tiempo	Tiempo con Hijos
DibujarNombre	0,24	0,24
TFormInfluencias::EnviarMensaje	0,13	0,13
AutoPintarse	0,11	0,12
TEditorInfluencias::Create	0,09	0,09
TFormInfluencias::CargarPropiedadesInfluencia	0,04	0,21
TEditorInfluencias::CalcularHojas	0,03	0,03
TFormInfluencias::FormCreate	0,03	0,12
TFormInfluencias::Destroy	0,02	0,03
TEditorInfluencias::BorrarLienzo	0,02	0,02
TPintorInfluencias::DibujarDiagramaI	0,01	0,37
TEditorInfluencias::Destroy	0,01	0,01
TFormInfluencias::SetZoomValor	0,01	0,09
TElementoInfluencia::Pintar	0	0,24
DibujarSignoMenos	0	0
DibujarSignoMas	0	0,01
TFormInfluencias::ColocarPropiedades	0	0
CalcularTamano	0	0
CargarDesdeEspacioSoloInfluencias	0	0
TElementoInfluencia::Cargar	0	0
TLineaInfluencia::Cargar	0	0
TLineaInfluencia::Pintar	0	0,12
TEditorInfluencias::PBInfluenciasMouseUp	0	0,09
TDatosProyectoInfluencia::ActualizarPunterosLinea	0	0
CrearMenuLinea	0	0

Métricas del Componente Comunes

Este anexo contiene las métricas tomadas para el componente Comunes usando el perfilador AQtime, estas fueron tomadas para una ejecución de un modelo con 1000 iteraciones.

Tabla 23: Métricas tomadas del Componente Comunes - Antes

Nombre de la Rutina	Tiempo	Tiempo con Hijos
SacarToken	38,36	38,55
THiloEvento::Execute	12,02	54,26
TLista::Cargar	3,81	6,47
SacarPar	2,85	2,85
TEFuncion::GetNombreFuncionMayusculas	1,3	1,3
TengaNombreE	0,94	0,94
DividirVector	0,9	1,84
Agregar	0,76	0,76
TColeccion::ForEach	0,42	0,86
EsEspecial	0,19	0,19
Establecer	0,16	0,18
THiloEvento::Destroy	0,09	0,09
LeerCadenaDeStream	0,06	0,06
TColeccion::FirstThat	0,05	1
FormatoEvolucionDelPortapapeles	0,05	0,05
TDatosEnMemoria::LeerPuntero	0,04	0,04
TCompresor::SacarEspacio	0,03	0,03
TListaElementosDinamico::Get	0,03	0,03
TDatosEnMemoria::LeerCadena	0,02	0,03
NuevoVector	0,02	0,02
TFrameUtilidades::Create	0,01	0,01
TCompresor::Create	0,01	0,01
TListaConecciones::Get	0,01	0,01
CheckEsc	0,01	0,01
TListaElementosDinamico::Add	0	0,99
TListaElementosDinamico::VerificarPosicionNombre	0	0,99
TListaElementosDinamico::ElementoConNombre	0	0,99
TDatosProyectoEvolucion::CargarDesdeEspacio	0	0,04

Tabla 24: Métricas tomadas del Componente Comunes - Después

Nombre de la Rutina	Tiempo	Tiempo con Hijos
Uprocesos::SacarToken	9,97	11,63
TEFuncion::GetNombreFuncionMayusculas	3,90	3,90
TLista::Cargar	1,73	3,83
Uprocesos::Validar	1,65	1,65
Umensaje::MensajePantalla	0,79	0,79
Uprocesos::SacarPar	0,69	0,69
TengaNombreE	0,60	0,60
Uprocesos::DividirVector	0,50	0,85
TColeccion::ForEach	0,40	0,46
Establecer	0,18	0,18
Agregar	0,16	0,16
TListaElementosDinamico::Get	0,16	0,16
TColeccion::FirstThat	0,10	0,72
Udatosmemoria::FormatoEvolucionDelPortapapeles	0,02	0,02
Umensaje::ImprimirMensajeEnArchivo	0,02	0,02
CheckEsc	0,01	0,01
TDatosEnMemoria::LeerPuntero	0,01	0,01
TDatosEnMemoria::LeerCadena	0,01	0,02
NuevoVector	0,01	0,01
TCompresor::SacarEspacio	0,01	0,01
Uutilidades::LeerCadenaDeStream	0,01	0,01
TListaConecciones::Get	0,01	0,01

Métricas del Componente Sensibilidad

Este anexo contiene las métricas del Componente Sensibilidad para la opción de Variación de Parámetros y de Variación de Escenarios, estos valores fueron tomados con el perfilador AQtime para una simulación con 1000 iteraciones.

Tabla 25: Métricas tomadas del Componente Sensibilidad - Modo Escenarios

Nombre de la Rutina	Tiempo	Tiempo con Hijos
TMotorEvolucion::Iteracion	432,05	629,19
TEvaluador::ReemplazarFuncion	147,13	187,84
ReemplazarParaElemento	85	93,19
TEvaluador::EvaluarExpresion	78,5	78,56
TElementoEvaluacion::TransformarDefiniciones	77,63	164,18
TGraficaEscenarios::Animar	21,92	21,98
TEvaluador::ReemplazarFunciones	5,88	179,49
EliminarPos	5,31	5,31
TElementoEvaluacionRetardo::Evaluarse	4,34	5,38
TListaElementosEvaluacion::Get	3,58	3,58
TAnimadorEscenarios::PasoMotor	3,09	20,34
TMotorEvolucion::CrearListaOrdenDeEvaluacion	2,25	2,25
TElementoEvaluacion::GetDimension	1,92	1,92
TElementoEvaluacion::Preparar	1,74	2,56
TGraficaEscenarios::MostrarPropiedadesControl	1,66	1,75
TManejadorEvaluacion::CrearElementoEvaluarCon	1,59	2,83
TElementoEvaluacion::Evaluarse	1,46	363,52
TElementoEvaluacion::GetValor	1,03	1,03
TElementoEvaluacionRetardo::GetValor	1	1
TManejadorEvaluacion::Evaluar	0,79	346,24
TEvaluador::ResolverEcuacion	0,77	214,45
TElementoEvaluacionRetardo::PrepararRetardo	0,52	1,19
TManejadorEvaluacion::EvaluarTodosElementos	0,45	346,47
TElementoEvaluacionNivel::EulerCalcularFlujos	0,38	0,64
TEvaluador::FSumaRetardo	0,36	0,36

Tabla 26: Métricas tomadas del Componente Sensibilidad - Modo Escenarios

TElementoEvaluacionNivel::GetValor	0,35	0,35
TElementoEvaluacionNivel::Preparar	0,34	0,76
TManejadorEvaluacionParametros::CrearElementoEvaluarCon	0,33	0,57
TGraficaEscenarios::Cargar	0,24	0,24
TManejadorEvaluacion::Evaluar	0,23	1,93
TTablaEscenarios::Animar	0,2	0,26
TGraficadorEscenarios::SetVerBarra	0,18	0,18
ReemplazarParaElemento	0,17	0,18
TMotor::SetEstadoSimulacion	0,15	0,32
TElementoEvaluacion::SetDimension	0,15	0,19
TElementoEvaluacionParametro::Evaluarse	0,14	0,5
TElementoEvaluacion::Destroy	0,12	0,12
TCustomListaElementosEvaluacion::Create	0,11	0,11
TFormAnimadorEscenarios::ColocarNombreEnBarraTitulo	0,1	0,1
TElementoEvaluacionParametros::TransformarDefiniciones	0,1	0,28
TFormGraficadorEscenarios::CrearObjetos	0,09	0,12
TGraficaParametros::Cargar	0,09	0,09
TElementoEvaluacionNivel::Evaluarse	0,09	0,99
TElementoEvaluacionRetardoParametros::TransformarDefiniciones	0,09	0,09
TGraficadorEscenarios::CrearVistaEscenarios	0,09	0,09
TFormTrayectoriasEscenenarios::Cargar	0,07	0,08
TElementoEvaluacion::GetDefiniciones	0,07	0,07
TVistaGraficadorEscenarios::CambiarModo	0,06	0,06
TGraficaEscenarios::Destroy	0,06	0,06
TFormTrayectoriasEscenenarios::ActualizarListas	0,06	0,06
TTablaEscenarios::GraficarEscenario	0,06	0,06
TElementoEvaluacion::Asignar	0,06	0,37
TManejadorEvaluacion::AgregarElementos	0,05	8,26
TElementoEvaluacion::Create	0,05	0,12
TGraficaEscenarios::GraficarEscenario	0,05	0,06
TGraficaParametros::Destroy	0,05	0,05
TGraficadorEscenarios::setEscenarioEnSimulacion	0,05	0,06
TElementoEvaluacion::SetValor	0,05	0,05
TVistaGraficadorEscenarios::Cargar	0,05	0,26
TElementoEvaluacionNivel::EvaluarNivel	0,05	0,69
TEvaluador::FCVariable	0,05	0,05
ExisteAlimentador	0,05	0,05
TCustomElementoEvaluacion::SetDimension	0,05	0,05
TManejadorEvaluacion::GetVT	0,05	0,05
TTablaEscenarios::Iniciar	0,05	0,05

Tabla 27: Métricas tomadas del Componente Sensibilidad - Modo Escenarios

TFormTrayectoriasEscenenarios::Destroy	0,05	0,05
TManejadorEvaluacion::SetMetodo	0,04	0,09
TElementoEvaluacionNivel::SetValor	0,04	0,04
AgregarAlimentador	0,04	0,09
TVistaGraficadorParametros::Cargar	0,03	0,14
TVistaGraficadorEscenarios::Destroy	0,02	0,08
TGraficadorEscenarios::Destroy	0,02	0,02
TElementoEvaluacion::ArreglarDefinicion	0,02	0,02
TGraficadorParametros::CrearVistaParametros	0,02	0,02
TVistaGraficadorParametros::CambiarModo	0,02	0,02
TTablaEscenarios::DatosAnalisis	0,01	0,01
TVistaGraficadorParametros::Destroy	0,01	0,03
TEvaluador::FCFunction	0,01	0,01
TElementoEvaluacionInicial::GetValor	0,01	0,01
TElementoEvaluacionNivel::Destroy	0,01	0,01
TMotorEscenarios::HacerIteracion	0,01	594,88
TElementoEvaluacionInicial::Evaluarse	0,01	0,02
TFormAnimadorParametros::ColocarNombreEnBarraTitulo	0,01	0,01
TGraficadorEscenarios::Create	0,01	0,01
TElementoEvaluacion::Reiniciar	0,01	0,01
TElementoEvaluacion::SetNombre	0,01	0,01
TVistaGraficadorEscenarios::Animar	0,01	22,09
TodosSubmodelosEstanDefinidos	0,01	0,01

Tabla 28: Métricas tomadas del Componente Sensibilidad - Modo Parámetros

Nombre de la Rutina	Tiempo	Tiempo con Hijos
TMotorEvolucion::Iteracion	387,06	629,22
TEvaluador::ReemplazarFuncion	100,89	135,92
TEvaluador::EvaluarExpresion	60,84	60,92
TTablaParametros::DatosAnalisis	50,95	50,95
ReemplazarParaElemento	49,03	53,04
TransformarDefiniciones	44,45	95,49
TTablaDatos::AnimarT	27,04	27,04
TGraficaParametros::MostrarPropiedadesControl	4,83	4,87
TTablaParametros::GraficarSerie	3,97	4,48
TVistaGraficadorParametros::Animar	3,53	85,96
TElementoEvaluacionRetardoParametros::Evaluarse	3,53	5,82
TGraficaEscenarios::MostrarPropiedadesControl	2,94	3,09
TListaElementosEvaluacion::Get	2,87	2,87
TTablaParametros::Animar	2,57	7,04
TEvaluador::ReemplazarFunciones	2,49	134,26
EliminarPos	2,33	2,33
TMotorEvolucion::CrearListaOrdenDeEvaluacion	2,01	2,02
ReemplazarParaElemento	1,96	2,2
TElementoEvaluacion::TransformarDefiniciones	1,91	3,95
TGraficaParametros::GraficarSerie	1,58	2,24
TElementoEvaluacion::Preparar	1,54	2,56
TManejadorEvaluacion::CrearElementoEvaluarCon	1,48	2,27
TGraficaParametros::Animar	1,26	3,37
TAnimadorParametros::PasoMotor	1,19	64,46
TCustomListaElementosEvaluacion::Get	1,17	1,17
TElementoEvaluacion::GetDimension	1,08	1,08
TElementoEvaluacionRetardo::GetTiempoDeAjuste	1,02	1,02
TElementoEvaluacionRetardo::GetOrden	1,01	1,01
TManejadorEvaluacionParametros::Evaluar	0,85	248,94
TElementoEvaluacionParametros::Evaluarse	0,58	247,06
TElementoEvaluacionRetardo::GetValor	0,55	0,55
TElementoEvaluacionNivel::Preparar	0,55	0,7
TElementoEvaluacion::GetValor	0,48	0,48
TEvaluador::ResolverEcuacion	0,43	160,39
CrearElementoEvaluarCon	0,4	0,81
TGraficaEscenarios::Cargar	0,37	0,37
TElementoEvaluacionRetardo::PrepararRetardo	0,31	0,99
TEvaluador::FSumaRetardo	0,28	0,28
TManejadorEvaluacion::EvaluarTodosElementos	0,23	259,45

Tabla 29: Métricas tomadas del Componente Sensibilidad - Modo Parámetros

TGraficadorParametros::SetVerBarra	0,23	0,23
TManejadorEvaluacion::Evaluar	0,22	2,69
TElementoEvaluacion::Destroy	0,19	0,19
TElementoEvaluacion::Evaluarse	0,16	9,46
TMotor::SetEstadoSimulacion	0,13	0,36
EliminarPos	0,13	0,13
TFormTrayectoriasParamametros::Cargar	0,13	0,14
TFormTrayectoriasEscenenarios::Cargar	0,13	0,14
TElementoEvaluacion::SetDimension	0,12	0,13
TElementoEvaluacionParametro::Evaluarse	0,12	0,73
TFormTrayectoriasParamametros::ActualizarListas	0,11	0,11
TElementoEvaluacion::Create	0,11	0,19
TManejadorEvaluacion::AgregarElementos	0,11	8,68
TGraficaEscenarios::Destroy	0,11	0,11
TElementoEvaluacionNivelParametros::EulerCalcularFlujos	0,1	0,22
AgregarAlimentador	0,1	0,1
TGraficadorEscenarios::CrearVistaEscenarios	0,09	0,1
TElementoEvaluacionNivel::GetValor	0,09	0,09
TVistaGraficadorParametros::Cargar	0,08	0,21
TElementoEvaluacion::GetDefiniciones	0,08	0,08
TCustomListaElementosEvaluacion::Create	0,07	0,07
TVistaGraficadorEscenarios::CambiarModo	0,07	0,07
TFormGraficadorEscenarios::CrearObjetos	0,07	0,15
TGraficadorParametros::CrearVistaParametros	0,07	0,07
TVistaGraficadorParametros::CambiarModo	0,07	0,07
TElementoEvaluacion::SetValor	0,06	0,06
TFormTrayectoriasEscenenarios::ActualizarListas	0,06	0,06
TEvaluador::FCVariable	0,06	0,06
TGraficaParametros::Cargar	0,06	0,06
TManejadorEvaluacion::BorrarListaElementos	0,06	0,26
TGraficaParametros::Iniciar	0,05	0,05
TFormTrayectoriasEscenenarios::LBEscModeloDrawItem	0,05	0,05
TFormTrayectoriasEscenenarios::SeleccionarEscen	0,05	0,05
TElementoEvaluacionNivel::Asignar	0,05	0,06
TElementoEvaluacionParametro::Asignar	0,05	0,17
TVistaGraficadorEscenarios::Cargar	0,04	0,29
TVistaGraficadorParametros::Destroy	0,02	0,03
TElementoEvaluacion::ArreglarDefinicion	0,02	0,02
TElementoEvaluacion::Asignar	0,02	0,28
TElementoEvaluacionInicialParametros::Evaluarse	0,02	0,02
TFormAnimadorEscenarios::ColocarNombreEnBarraTitulo	0,01	0,01
TEvaluador::FCFunction	0,01	0,01

Tabla 30: Métricas de Variación de Parámetros - Antes y Después

Iteraciones	Versión Serial	Versión MultiHilo
1000	605,682	117,168
		122,054
		115,443
		121,374
		118,772
500	208,632	52,991
		63,07
		53,553
		64,508
		51,989
200	56,82	24,261
		19,644
		25,112
		20,303
		24,322
100	22,985	10,62
		12,114
		10,012
		12,391
		10,352

Tabla 31: Métricas de Variación de Escenarios - Antes y Después

Iteraciones	Versión Serial	Versión MultiHilo
1000	439,826	107,186
		91,711
		110,489
		94,855
500	153,29	52,918
		44,538
		54,086
		46,677
200	42,324	17,999
		21,413
		18,987
		20,635
100	20,332	7,779
		10,166
		8,689
		9,925

Métricas de los Componentes Principal y Proyecto

Este anexo contiene las métricas tomadas de los componentes Principal y Proyecto al cargar un proyecto desde archivo, solo se registran los datos mayores a 0.1 segundos.

Tabla 32: Métricas del Componente Principal/Proyecto

Nombre de la Rutina	Tiempo	Tiempo con Hijos
TProyecto::Abrir	7,42	7,97
TFormPrincipal::TBnAbrirClick	0,78	8,74
TFormPrincipal::Activarse	0,46	0,61
TProyecto::CrearDInfluencias	0,16	0,29
TProyecto::Destroy	0,12	0,16
LlenarListaConEscenario	0,11	0,12
TProyecto::CrearForrester	0,1	0,26
TProyecto::Cerrar	0,08	0,25
TFormPrincipal::FormCreate	0,08	0,13
TFormPrincipal::LlenarEscenarios	0,03	0,15
TFormPrincipal::CambiarTamanoBarra	0,01	0,01
TFormPrincipal::ActivacionSector	0,01	0,01
TProyecto::ExisteAlgunElementoSeleccionado	0,01	0,01

Métricas del Componente FIS

Tabla 33: Métricas del Componente FIS - Antes

Nombre de la Rutina	Tiempo	Tiempo con Hijos
TPanelGrafica3d::dibujar3D	30,10	36,44
TRegla::EvaluarRegla	4,30	4,37
TOperador::calcular	1,62	2,01
ActualizarProgresBar	0,62	0,62
TFuncion::evaluarFuncion	0,57	0,67
Uutilidadesfis::RealACadena	0,26	0,26
Uutilidadesfis::ReemplazarVarios	0,24	0,40
PrepararCambioVariable	0,12	0,12
TRegla::TransformarRegla	0,12	0,12
TListaVariableConjuntos::setEsHabilitado	0,07	0,07
Uutilidadesfis::ObtenerParametros	0,07	0,07
TListaVariableConjuntos::EliminarItem	0,04	0,04
TPanelGrafica3d::SetParent	0,03	0,03
TFormContenedorEditorFis::cargarTipoFuncion	0,03	0,03
TFuncion::TFuncion	0,02	0,02
TPertenencia::getFnombreConjunto	0,02	0,02
TListaPertenencia::getElemento	0,02	0,02
TFuncionMembresia::evaluarFuncion	0,02	0,71
TListaVariableConjuntos	0,02	0,09
TPertenencia::getFnombreVaribleLinguistica	0,02	0,02
TOperadorFIS::calcular	0,02	2,02
TBorrosificador::borrosificar	0,02	1,66
llenarVariablesHojaGraficas	0,02	0,02
TFuncion::EstaEnIntervalo	0,01	0,02
TDominio::estaEnIntervalo	0,01	0,01
TConjunto::agregarModificador	0,01	0,01
llenarListaFuncionesMembrecia	0,01	0,01
cagarDeborrosificadores	0,01	0,01
TTFormEstablecerDifusor::cargarTipoFuncion	0,01	0,01

Tabla 34: Métricas del Componente FIS - Después

Nombre de la Rutina	Tiempo	Tiempo con Hijos
TRegla::EvaluarRegla	4,54	4,64
TOperador::calcular	1,64	2,06
TFuncion::evaluarFuncion	0,55	0,67
ActualizarProgresBar	0,52	0,52
Uutilidadesfis::RealACadena	0,30	0,30
Uutilidadesfis::ReemplazarVarios	0,25	0,43
TRegla::TransformarRegla	0,11	0,11
Uutilidadesfis::ObtenerParametros	0,07	0,09
PrepararCambioVariable	0,04	0,04
TBaseDeReglas::TBaseDeReglas	0,04	0,17
TListaPertenencia::getElemento	0,04	0,04
TPanelGrafica3d::SetParent	0,03	0,03
TListaVariableConjuntos::setEsHabilitado	0,03	0,03
TPertenencia::getFnombreVaribleLinguistica	0,02	0,02
TFuncion::TFuncion	0,02	0,02
TTFormEstablecerDifusor::cargarTipoFuncion	0,02	0,02
TPertenencia::getFnombreConjunto	0,02	0,02
TConjunto::agregarModificador	0,02	0,02
llenarVariablesHojaGraficas	0,02	0,02
Uutilidadesfis::ReemplazarVarios	0,01	0,01
TOperadorFIS::calcular	0,01	2,07
TFuncionMembresia::evaluarFuncion	0,01	0,70
TDominio::estaEnIntervalo	0,01	0,01
TBorrosificador::borrosificar	0,01	1,67
TFuncion::EstaEnIntervalo	0,01	0,01
TGraficasFis::pintarLinea	0,01	0,01
cargarTipoFuncion	0,01	0,01
TListaVariableConjuntos::EliminarItem	0,01	0,01
TGraficasFis::agregarFuncion	0,01	0,12
llenarListaFuncionesMembrecia	0,01	0,01
cargarOperadoresImplicacion	0,01	0,01
TOperador::setDefinicionOperador	0,01	0,01
cagarDeborrosificadores	0,01	0,01
cargarOperadoresAgregacion	0,01	0,01

Migración a Delphi XE4

En primer lugar se han listado una descripción en general de las nuevas características incluidas en Delphi XE4 que no se encuentran en Delphi 7. Esta se encuentra mas complementada en el documento «Razones para Migrar a Delphi XE»[16]

Delphi 2005

- Bucles for...in...do, funciones lineales y otras optimizaciones de código.
- Acceso heterogeneo a base de datos.
- Refactorización y Vista histórica del código fuente.
- Pruebas de Unidad

Delphi 2006

- Modelado UML, Auditorias, Métrics y Generación de Documentos
- Patrones de Diseño

Delphi 20007

- VCL compatible con AJAX y VISTA.
- Diseño Vista y XP para aplicaciones.
- dbExpress . nuevo framework, drivers y soporte Unicode en base de datos.

Delphi 2009

- Soporte UNICODE en el lenguaje, librerias e IDE.
- Genéricos y Métodos Anónimos.
- Editor de Recursos y Explorador de Clases
- Nuevos Componentes VCL
- Localización del lenguaje.

Delphi 2010

- Soporte para Windows 7, Gestos y Multi-Touch, Direct-2D
- Formateador de Código Fuente

- Compilación de Fondo
- RTTI optimizado
- Puntos de parada en Hilos y congelación de hilos.
- DataSnap - Soporte para el protocolo HTTP.

Delphi XE

- DataSnap - Soporta de HTTPS, Javascript y REST.
- Integración con Subversion.
- Librería de Expresiones Regulares.
- AQtime, CodeSite, Beyond Compare y Final Builder incluidos.

Para aprovechar estas nuevas características de la interfaz de desarrollo Delphi XE4 es necesario entender y preparar el código para ser compatible con el soporte a Unicode introducido desde Delphi 2009.

En Delphi 7 se usan los siguientes tipos: STRING (para cadenas de texto), CHAR (para caracteres) y PCHAR (como apuntador) estos son usados frecuentemente en el código fuente de Evolución aunque en el código sean escritos de esa manera internamente Delphi transforma esas declaraciones tal como se ve en la Tabla 4.1. Para hacer mas claro el cambio hay que decir que el conjunto ANSI es un grupo de 217 caracteres, mientras que el conjunto de caracteres Unicode virtualmente permite que todos los diccionarios puedan ser codificados en una cadena de texto.

Tabla 35: Tabla de Conversión en Delphi

	String	Char	Pchar
Delphi 7	AnsiString	AnsiChar	PAnsiChar
Delphi XE4	UnicodeString	WideChar	PWideChar

En Delphi XE4 para el soporte a Unicode todos los CHAR pasan de 8 bits a 16 bits para representar los demás caracteres internacionales, y todos los apuntadores PChar pasan de ser un apuntador de 8 bits a tener 16 bits. Esto representa un problema para Evolución que usa ampliamente el manejo de punteros, lo cual es una practica que se desalienta en lenguajes modernos, en este caso la migración mas sencilla del código a Delphi XE4 está en convertir todas las declaraciones de estos tipos mencionados a AnsiString, AnsiChar y PAnsiChar de esta manera el código compila de igual manera que en la versión Delphi 7.

Esta migración permite la compilación de Evolución en Delphi XE4 pero trae consigo ciertos costo, en primer lugar limita la internacionalización de la herramienta a los caracteres soportados por esta codificación. En segundo lugar, todo el manejo de la interfaz es UnicodeString por lo que el compilador realiza múltiples conversiones implícitas que ocasionan fugas de memoria.

Existen ventajas para el desarrollo de Evolución en Delphi XE4, aunque como se ve en el desarrollo del proyecto el soporte para la explotación de concurrencias mediante librerías no es recomendable al no ser apoyado por Embarcadero (empresa desarrolladora de Delphi XE4)