Visualización del mercado de acciones en 3D

Juan Esteban Pemberthy

Juan Sebastian Munoz

29 de abril de 2009

Índice general

Glosario

1. **TDD (Test-driven development):** Desarrollo guiado por pruebas, es una práctica de programación que involucra otras dos prácticas: Escribir las pruebas primero (**Test First Development**) y Refactorización (**Refactoring**). Para escribir las pruebas generalmente se utilizan las pruebas unitarias (**unit test** en inglés). Primeramente se escribe una prueba y se verifica que las pruebas fallen, luego se implementa el código que haga que la prueba pase satisfactoriamente y seguidamente se refactoriza el código escrito. El propósito del desarrollo guiado por pruebas es lograr un código limpio que funcione (Del inglés: **Clean code that works**). La idea es que los requerimientos sean traducidos a pruebas, de este modo, cuando las pruebas pasen se garantizará que los requerimientos se hayan implementado correctamente.

2. **Pair Programming:** Programación en pareja, requiere que dos Ingenieros en Software participen en un esfuerzo combinado de desarrollo en un sitio de trabajo. Cada miembro realiza una acción que el otro no está haciendo actualmente: Mientras que uno codifica las pruebas de unidades el otro piensa en la clase que satisfará la prueba, por ejemplo. La persona que está haciendo la codificación se le da el nombre de controlador mientras que a la persona que está dirigiendo se le llama el navegador. Se sugiere a menudo para que a los dos socios cambien de papeles por lo menos cada media hora o después de que se haga una prueba de unidad.

3. **Mock Object:** Objeto simulado, los objetos simulados imitan el comportamiento de los objetos reales en forma controlada. Un programador generalmente escribirá un objeto simulado para probar el comportamiento de otro objeto en una forma muy similar a la que aplica un diseñador de carros al usar un muñeco para probar el comportamiento de un carro durante un accidente.

4. **RSpec:** Ruby Spec, es un framework para realizar tests en el lenguaje de programación Ruby, contiene su propio **framework** para simular objetos.

5. **ORM:** mapeo objeto-relacional, es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional.

En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

6. **MVC:** Modelo Vista Controlador, es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

7. **Daemon:** Un demonio, (siglas en inglés, **Disk And Execution Monitor**), es un tipo especial de proceso informático que se ejecuta en segundo plano en vez de ser controlado directamente por el usuario. Este tipo de programas se ejecutan de forma continua (infinita), vale decir, que aunque se intente cerrar o matar el proceso, este continuará en ejecución o se reiniciará automáticamente. Todo esto sin intervención de terceros y sin dependencia de consola alguna.

8. **Unity3D:** Motor de Juegos para crear aplicaciones 3D.

9. **Shader:** Conjunto de instrucciones gráficas destinadas para el acelerador gráfico, estas instrucciones dan el aspecto final de un objeto. Los Shaders determinan materiales, efectos, color, luz, sombra y etc.

10. **Vertex Shaders:** Actúa sobre las coordenadas, color, textura, etc. de un vértice.

11. **Geometry Shaders:** Es capaz de generar nuevas primitivas dinamicamente.

12. **Pixel Shaders:** Actúa sobre el color de cada pixel (texel para ser más preciso).

13. **Mesh:** Término que se refiere a una figura en 3D, en general que esté formada por polígonos.

14. **Triangulación de Delaunay:** Es una red de triángulos que cumple la condición de Delaunay. Esta condición dice que la circunferencia circunscrita de cada triángulo de la red no debe contener ningún vertice de otro triángulo. Se usan triangulaciones de Delaunay en geometría por ordenar, especialmente en gráficos 3D por computadora.

15. **Transform:** Objeto en Unity3D que describe propiedades de un **Mesh** como Escalamiento, Rotación, Posición, etc.

16. **Proyección Ortográfica:** Esta proyección utliza rayos paralelos para crear una imágen de la escena. El área de vision está determinada por las longitudes de los vectores **right** y **up**, que, por cierto, han de ser

especificados, ya que con este tipo de proyección no se utilizan los de la cámara por defecto. En caso de ser omitidos, se usará el segundo método de proyección ortográfico de la cámara.

17. **Proyección Perspectiva:** La palabra clave **perspective** determina la cámara perspectiva, que simula la típica cámara con objetivo. El ángulo de visión horizontal es determinado por la proporción entre la longitud del vector dirección y la longitud del vector **right**, o por la palabra clave opcional **angle**, que es el modo aconsejado. El ángulo de visión ha de ser mayor de 0 y menor de 180 grados.

18. **Framework:** En el desarrollo de software, es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

19. **Agile Software Development:** Desarrollo ágil de software, se entiende como Desarrollo ágil de software a un paradigma de Desarrollo de Software basado en procesos ágiles. Los procesos ágiles de desarrollo de software, conocidos anteriormente como metodologías livianas, intentan evitar los tortuosos y burocráticos caminos de las metodologías tradicionales enfocándose en la gente y los resultados. Es un marco de trabajo conceptual de la ingeniería de software que promueve iteraciones en el desarrollo a lo largo de todo el ciclo de vida del proyecto. Existen muchos métodos de desarrollo ágil; la mayoría minimiza riesgos desarrollando software en cortos lapsos de tiempo. El software desarrollado en una unidad de tiempo es llamado una iteración, la cual debe durar de una a cuatro semanas. Cada iteración del ciclo de vida incluye: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación. Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al mercado, pero la meta es tener un demo (sin errores) al final de cada iteración. Al final de cada iteración el equipo vuelve a evaluar las prioridades del proyecto.

20. **CoC:** Convención sobre Configuración, es un paradigma de programación de software que busca decrementar el número de decisiones que un desarrollador necesita hacer, ganando así en simplicidad pero no perdiendo flexibilidad por ello. Cuando la convención tomada es suficiente para lograr el comportamiento deseado, se hace innecesario realizar aquellas tareas para las que la convención ya ha definido un comportamiento, por ejemplo escribir archivos XML de configuración del entorno. Cuando la convención definida no es suficiente para lograr el comportamiento deseado, el desarrollador puede alterar el comportamiento por defecto y adaptarlo a sus necesidades.

21. **Don't Repeat yourself (DRY):** El principio No te repitas, es una filosofía de definición de procesos que promueve la reducción de la duplicación especialmente en computación. Según este principio ninguna pieza de información debería estar duplicada nunca debido a que la duplicación incrementa la dificultad en los cambios y evolución posterior, puede perjudicar la claridad y crea un espacio para posibles inconsistencias. Por 'pieza de información' podemos, en un sentido amplio, entender desde datos almacenados en una base de datos pasando por el código fuente de un programa de software hasta llegar a información textual o documentación. Cuando el principio DRY se aplica de forma eficiente los cambios en cualquier parte del proceso requieren cambios en un único lugar. Por el contrario, si algunas partes del proceso están repetidas por varios sitios, los cambios pueden provocar fallos con mayor facilidad si todos los sitios en los que aparece no se encuentran sincronizados.

Introducción

> *"Elegir acciones individuales sin una idea de lo que estás buscando es como atravesar una fábrica de dinamita con una cerilla encendida. Puedes sobrevivir, pero sigues siendo un idiota"*
> –Joel Greenblat[1]

En la bolsa de valores de Colombia existen cuatro mercados principales de operación, el trabajo de ésta tesis se enfoca única y exclusivamente a uno de estos mercados; El mercado de Renta Variable, donde se negocian las acciones de compañías inscritas en el mercado publico de valores.

Al tratarse de un mercado público cualquier persona esta en la libertad de comprar y/o vender acciones según lo desee, generalmente estas personas llevan cierto control sobre sus inversiones, ya sea de un modo sistematizado ó escribiendo en una agenda de apuntes sus movimientos, con esta información realizan un análisis de datos que le permite determinar cuándo una acción le genera ganancia y/o perdida que apoyen una decisión, como comprar o vender.

El análisis de datos siempre ha estado ligado a una representación gráfica de los mismos, y el mercado de acciones mencionado anteriormente no es la excepción, generalmente para representar éstos datos, los que describen el estado y movimiento en el tiempo de una acción se han utilizado gráficos en segunda dimensión como barras, tortas, entre otras.

Esta tesis introduce un nuevo paradigma de visualización en este campo, al pasar de gráficos en dos dimensiones, a una nueva propuesta de contenidos para visualizar los datos en tres dimensiones, permitiéndole a una persona visualizar de una manera innovadora el estado y movimiento de las acciones sobre las que lleve algún tipo de registro.

Para lograr éste trabajo, se desarrollo una aplicación web, en la que se integró satisfactoriamente un modelo de datos con un conjunto de contenidos 3D, sobre los cuales se estableció un protocolo de comunicación que permite el apropiado despliegue de gráficos, buscando que sea más fácil para un usuario

[1] *"The little book that beats the market"* por Joel Greenblat.

identificar el movimiento de una ó varias acciones.

La propuesta acá descrita es presentada como proyecto de grado para optar por el titulo de Ingeniería de Sistemas de la Universidad EAFIT.

Definición del problema

Durante la evolución de la computación moderna se han presenciado avances significativos en los diferentes campos que componen la misma, uno de estos campos es la visualización de aplicaciones, es decir la forma en la que una aplicación es desplegada para que una persona interactue con una máquina con el fin de cumplir una determinada tarea, el surgimiento de la computación gráfica como área de estudio le ha significado a la computación una nueva vía para desarrollar aplicaciones en las que la visualización juega un papel fundamental y que antes apenas si eran soñadas.

Con éste proyecto de grado se busca aprovechar la capacidad tecnológica actual para desarrollar contenidos en tercera dimensión que puedan ser visualizados desde cualquier lugar, el tema sobre el que se trabajó fue el movimiento del mercado de acciones Colombiano, se eligió este sector para brindarle a los propietarios de acciones que no están sumergidos de lleno en este mundo una herramienta que les permita identificar fácil y gráficamente el movimiento del mercado, pues actualmente muchas de estas personas llevan control de sus inversiones en un papel, o en un archivo plano, sobre el que construyen una tabla, y/o gráficas limitadas a dos dimensiones.

Las gráficas en segunda dimensión, son simples y fáciles de entender cuándo se están comparando acciones en una unidad de tiempo ó se esta viendo el movimiento de una sola acción en un período determinado, pero a la hora de comparar varias acciones en un período, una gráfica en 2D podría ser difícil de entender para un usuario común, mientras que una gráfica en 3D facilita el entendimiento de la misma, con el fin de apoyar el problema definido anteriormente, en esta aplicación se diseñaron un conjunto de contenidos en tercera dimensión que recrean la actividad del mercado, dichos contenidos son alimentados desde una base de datos que se actualiza constantemente, tanto datos como contenidos 3D fueron integrados en una aplicación web permitiendo que la aplicación sea visualizada desde cualquier lugar en el que se tenga acceso a Internet.

Capítulo 1

Objetivos del proyecto

1.1. Objetivo General

Brindar un mecanismo que permita visualizar y comparar el movimiento de una o varias acciones de la bolsa de valores de Colombia en el tiempo, mediante contenidos en tercera dimensión, incursionando en un campo sobre el cuál la mayoría de aplicaciones están diseñadas para desplegar sus resultados en 2D.

1.2. Objetivos Específicos

- Permitir llevar el control e historial de acciones de manera personalizada.
- Desarrollar un protocolo de comunicación que pueda ser reutilizado en aplicaciones futuras, que exploten el poder del motor gráfico para desarrollar contenidos alimentados desde una base de datos.
- Integrar el modelo de datos con la aplicación 3D para desplegar apropiadamente el movimiento del mercado y/o una determinada acción.

Capítulo 2

Alcance y productos esperados

Se entrega una aplicación Web desarrollada en **Ruby on Rails** que integra un conjunto de contenidos en 3D desarrollados en **Unity 3D** un motor gráfico que permite exportar los contenidos para que sean acoplados con la aplicación Web y consecuentemente se puedan visualizar desde un navegador. Además los contenidos se estarán actualizando constantemente en un período de tiempo razonable, a esto se le suma la posibilidad de un usuario registrarse y configurar su perfil, sobre este perfil cada usuario podrá simular o evidenciar según sea el caso (sí posee ó no acciones en la vida real) ganancias o pérdidas sobre las acciones que tenga registradas y la cantidad asociada a las mismas.

Se aclara de antemano que la aplicación no sirve como medio para comprar o vender acciones, en el perfil también se puede desplegar de forma personalizada el movimiento de las acciones que el usuario tenga registradas, permitiendo diferentes esquemas de visualización en los que por ejemplo se tendrá en cuenta información guardada en la base de datos de la aplicación, por ejemplo los periodos de tiempo en los que un usuario registró una compra o venta serán utilizados para ayudar al usuario a que se de una mejor idea del estado de su inversión.

Adicionalmente los clásicos esquemas en segunda dimensión que actualmente son utilizados para la representación del movimiento de acciones como lo son las barras, también fueron implementados en la aplicación, pero de tal manera que tales barras no muestren sólo el estado de una acción en varios periodos de tiempo, o varias acciones en un periodo de tiempo, sino ofrecer la oportunidad de ver varias acciones como se comportan en varios periodos de tiempo.

Capítulo 3

Justificación

El uso de tecnologías de visualización, como apoyo al entendimiento de datos planos en diferentes entornos, tanto de ingeniería como administrativos es un tema que día a día ha venido tomando auge en nuestra sociedad, debido a que facilitan la comprensión de como se están comportando dichos datos para consecuentemente ayudar en un momento determinado con la toma de decisiones.

Al facilitar una herramienta que permita visualizar los datos del mercado de acciones, no solo en el tiempo para una sola acción, sino también entre diferentes acciones a la vez, se abre la posibilidad de comparar rendimientos o pérdidas de una o varias acciones a través del tiempo, dentro de un mismo gráfico, para así mirar tendencias y tomar decisiones.

El planteamiento de la solución acá expuesta se basa en la necesidad de tener mas control y conocimiento sobre un portafolio que una persona determinada maneje, independientemente de donde esta se encuentre, pues gracias a la ayuda de la tecnología y mas específicamente de la Web 2.0, acceder desde cualquier computador a un mismo portafolio es completamente transparente para quien lo usa. De esta forma, los usuarios podrán tener acceso a su portafolio de acciones en tiempo real, en el que encuentran tablas y gráficos que representan el historial de sus inversiones.

Capítulo 4

Marco referencial

El Mercado Público de Valores en Colombia ha pasado por muchas etapas y ha sido manejado por diversas instituciones, las cuales nacen y desaparecen con el correr de los años según cambian las necesidades de quienes intervienen en ellas. Tuvo que pasar un siglo desde la creación de la primera bolsa en el país para que apareciera la Bolsa de Valores de Colombia, institución que hoy representa la unificación del Mercado Bursátil.[10]

Así como en otros sectores, el desarrollo tecnológico ha jugado un papel fundamental para el desarrollo de la bolsa de valores, pues desde sus inicios quienes han estado sumergidos en este mundo, se han valido de la tecnología de momento para llevar control de las acciones, es así como desde que se presentó una fiebre de especulaciones surgida por las oscilaciones de la tasa de cambio ocurridas a principio de siglo(debido a la guerra de los mil días) y que se desarrollaban todas las noches de 7:00 a 10:00 en el atrio de la catedral de la Candelaria de Medellín y dentro del parque Pedro Justo Berrio[8], posteriormente culminaban con el desarrollo de negocios en oro y los registros eran calculados con una máquina sumadora o en algunos casos eran parte de un calculo mental. Actualmente la Bolsa cuenta con avanzados sistemas de información de uso interno que permite conocer el estado en tiempo real de una acción y ver como se ha comportado en el tiempo, las personas naturales que poseen acciones y que no tienen acceso a estos sistemas pueden consultar el estado de sus inversiones en diversos sitios públicos (sitios web, revistas, periódicos), en donde se les muestra una información plana a veces acompañada de unos gráficos en dos dimensiones.

Muchas de las personas naturales mencionadas anteriormente, llevan el control de sus inversiones en libros y hojas de cálculo diseñadas por ellos mismos, algunos, los mas familiarizados con herramientas tecnológicas utilizan herramientas disponibles (Google Finance entre otras), estas ultimas herramientas son útiles a la hora de llevar el historial y permiten calcular fácilmente ganancias/pérdidas, algunos de los problemas que tienen estas aplicaciones son:

1. Los gráficos a pesar de que son claros, están limitados a presentar en dos dimensiones múltiples indicadores, que a la larga le podría representar al usuario común complicaciones a la hora de entender el gráfico.

2. La mayoría no permiten llevar el control personalizado del portafolio, y aquellas que integran esta característica rara vez incluyen a la bolsa de valores de Colombia o son de libre acceso.

3. Ninguna de estas aplicaciones ofrece un componente 3D que permita comparar varias variables que afectan el mercado accionario para apreciar mejor el comportamiento de las acciones, y que aparte de esto se puedan acceder desde un navegador web, pues la tendencia en 3D es desarrollar aplicaciones tipo **Stand-Alone**[1]

Es por esto que es pertinente el desarrollo de una aplicación que sea portable y permita ser visualizada desde cualquier lugar en cualquier plataforma, ademas de esto que permita apreciar las variables que afectan el mercado accionario Colombiano.

[1] Aquellos programas que no necesitan de un interpretador para ser ejecutados en una máquina.

Capítulo 5

Solución

5.1. ¿Por qué Ruby on Rails?

Para responder a esta pregunta, es necesario primero definir a **Ruby on Rails**, llamado tambien *Rails* ó RoR, es un *framework* que hace que sea más fácil desarrollar, implementar, y mantener aplicaciones web. Durante los meses que siguieron a su primera liberación, *Rails* paso de ser un juguete desconocido a un fenomeno mundial. Ha ganado premios, y más importante aún, se ha convertido en el framework de preferencia para la implementación de cierto conjunto de las llamadas aplicaciones Web 2.0[13].

Se eligió *Rails* para desarrollar la aplicación web en la que se comunican datos y contenidos 3D, por qué se buscaba utilizar una herramienta en la que la convención estuviera por encima de la configuración[1], que contará con una arquitectura pre definida, en éste caso MVC (Modelo Vista Controlador) Figura5.1 que *Rails* aprovecha apropiadamente, cuándo se desarrolla en *Rails*, hay un lugar para cada pieza de código[2], y todos los componentes de la aplicación interactúan según el estándar.

Los programadores profesionales, suelen escribir tests para simular el comportamiento de la aplicación que se esta desarrollando, para el presente proyecto, se buscaba que el framework tuvieran un soporte para hacer tests, una vez más Rails se ajusta a esas necesidades, pues cuenta con soporte para algunos frameworks para realizar tests, se encuentran; RSpec (utilizada en este proyecto, bajo la filosofía de TDD [3]) en donde se escriben los tests, antes de la implementación, Test Unit, Shoulda y Cucumber (BDD[4])

[1] Léase **CoC** en el glosario
[2] Léase **DRY** en el glosario
[3] Léase **TDD** en el glosario
[4] Léase **BDD** en el glosario

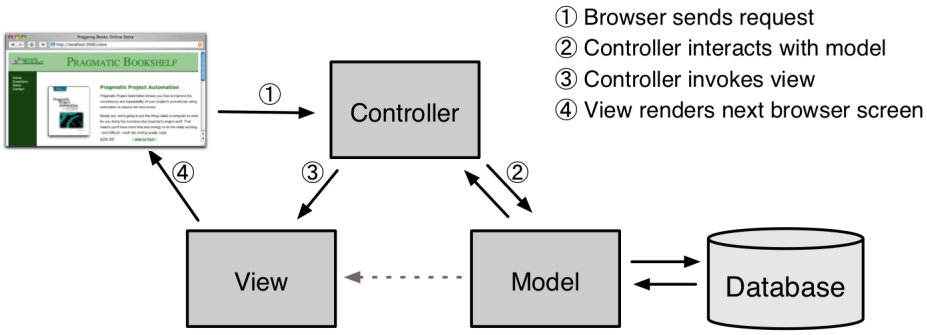


Figura 5.1: Modelo Vista Controlador

Se buscaba que la herramienta fuera escrita en un lenguaje de programación orientado a objetos e interpretado, por la necesidad de crear *scripts* que corrieran independiente de la aplicación, como tareas programadas para correr cada cierto tiempo. Las aplicaciones en *Rails* son escritas en el lenguaje de programación *Ruby*, que cumple con lo que se buscaba, y aparte introduce características favorables como metaprogramación, haciendo que el código sea más fácil de entender, leer y/o escribir, programas más cortos en términos de **LOC**[5], para evidenciar esto, a continuación se muestra una porción de código de la aplicación, en la que se define una clase de un Modelo que expresa mucha información en pocas líneas de código y es fácil de entender.

```
class Record < ActiveRecord::Base

  belongs_to :stock_action

  validates_presence_of :amount
  validates_presence_of :price
  validates_presence_of :variation

  validates_numericality_of :price, :amount, :greater_than => 0
  validates_numericality_of :variation

end
```

Finalmente, no se pretendía manipular la base de datos directamente, es decir, con código nativo del motor de bases de datos, sino sacar provecho de técnicas como ORM [6], en las que las tablas de las bases de datos son mapeadas

[5]Líneas de código, en inglés *Lines of Code*
[6]Léase **ORM** en el glosario

a clases, los registros de las tablas son mapeados como objetos, y consecuentemente los campos, se mapean como atributos de los objetos, mientras que existen metodos de clases que se encargan de realizar operaciones a nivel de tablas, dejando como opcional la utilización de código **SQL** dentro de la aplicación, una vez más *Rails* suple esta necesidad con **ActiveRecord** ó **Datamapper**.

5.2. ¿Por qué Unity3D?

Actualmente existen posibilidades para el desarrollo de interfaces gráficas de usuario tridimensionales en los navegadores de internet, **PaperVision3D**[7] (entre otros) es un motor de renderizado 3D en tiempo real, escrito en **ActionScript 3**[8], de código abierto entre otros, que posee las funcionalidades básicas de la computación gráfica tridimensional, creando una ilusión 3D en el motor de renderizado que posee el *Flash Player*.

Desafortunadamente el motor de renderizado para *Flash* esta diseñado específicamente para gráficos en 2D. Este es un punto crítico por el cual *Flash* no es técnicamente la mejor opción. La cantidad de fotogramas de contenido 3D renderizados por segundo (FPS)[9] suele ser media o baja, en casos reales. La comunidad de desarrolladores del *Flash Player* está desarrollando varios proyectos para poder renderizar 3D en el mismo, existen proyectos privados y de código abierto, entre los que se encuentran:

- *Alternativa3D*
- *Away 3D*
- *Five3D*
- *ND3D*
- *Papervision3D*
- *Sandy3D*
- *Wire Engine 3D*

[7] **Papervision** es un motor de gráficos para Flash. Aunque todavía está en versión beta, ya se han lanzado algunas aplicaciones que lo utilizan y los resultados son muy prometedores.

[8] **ActionScript** es un lenguaje de programación orientado a objetos (OOP), utilizado en especial en aplicaciones web animadas realizadas en el entorno **Adobe Flash**

[9] Las imágenes por segundo (en inglés más conocido como *frames per second, fps*) es la medida de la frecuencia a la cual un reproductor de imágenes genera distintos fotogramas (*frames*). En informática estos fotogramas están constituídos por un número determinado de pixeles que se distribuyen a lo largo de una red de texturas para determinar un fotograma por segundo.
La frecuencia es proporcional al número de pixeles que se deben generar, incidiendo en el rendimiento de la máquina.

Los cuales usualmente utilizan algoritmos de rasterizado para el proceso de generar una imagen 2D a partir de una escena 3D, consumiendo así en dicho proceso, mucha capacidad computacional.

La técnica más utilizada hoy en día para la producción de gráficos 3D en tiempo real es la rasterización. La rasterización es básicamente un proceso de transformación de datos de vectores que se convierten en un conjunto de pixeles (imágenes).

En pocas palabras, muchos de los motores de renderizado 3D escritos para trabajar con el *plugin* de *Flash* utilizan bastante lógica para producir una ilusión 3D sobre 2D debido a que es una de las técnicas mas rápidas, pero, como se mencionó anteriormente, estos motores por mas optimizados que estén, siempre van a depender del renderizado 2D para el cual fue diseñado *Flash* en un principio.

Unity3D por su parte también es un *plugin* para navegadores, especializado en la creación de contenido 3D en tiempo real. Técnicamente es la mejor opción para desarrollo 3D, pues dicho *plugin* aprovecha las capacidades de procesamiento de hardware de la tarjeta de video para desplegar contenidos en 3D, permitiendo así optimizar ciclos de procesamiento en el mejoramiento de los FPS de la escena en vez de ejecutar cálculos de conversión de imágenes 3D a 2D, repercutiendo enormemente en la experiencia del usuario.

Unity3D por su parte es un motor escrito en C/C++ que trabaja con *Mono*[10] y *PhysX*[11] para crear código multiplataforma, haciendo así practicamente transparente generar ejecutables tanto para MacOSX, Windows, widgets y exportar para Web.

Gracias a Mono, es fácil generar ejecutables multiplataforma debido a la arquitectura con la que dicho *framework* trabaja.Figura5.2

Por otro lado, *Unity3D* también se aprovecha de la capacidad del motor de física *PhysX* para llevar a cabo cálculos complejos para así alivianar la carga de procesamiento que conlleva un juego/aplicación 3D.

Cuando el Motor exporta contenido a Web, dicho contenido se puede comunicar facilmente con *Javascript*, permitiendo así entablar un puente de comunicaciones con *Rails* a través de *Javascript*, factor clave para la visualización de

[10] **Mono** es el nombre de un proyecto de código abierto iniciado por Ximian y actualmente impulsado por Novell (tras la adquisición de Ximian) para crear un grupo de herramientas libres, basadas en GNU/Linux y compatibles con .NET según lo especificado por el ECMA.

[11] **PhysX** es un chip y un kit de desarrollo diseñados para llevar a cabo cálculos físicos muy complejos. Conocido anteriormente como la SDK de NovodeX, fue originalmente diseñada por AGEIA y tras la adquisición de AGEIA, es actualmente desarrollado por Nvidia e integrado en sus chip gráficos más recientes.

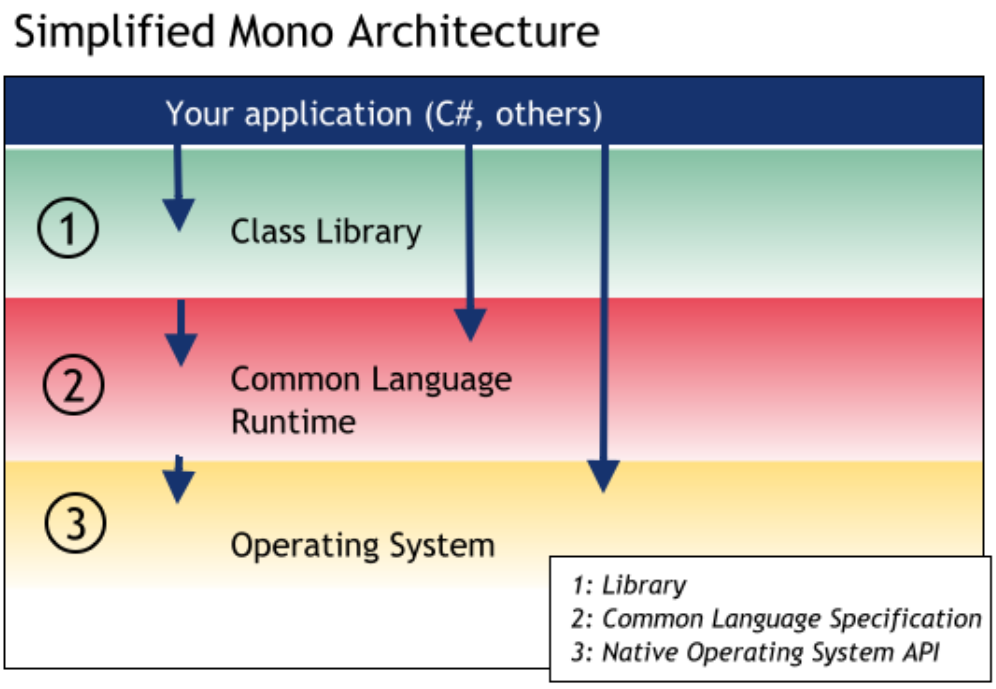


Figura 5.2: Arquitectura general del *framework* Mono

los datos que tiene la aplicación.

Capítulo 6

Objetivos

6.1. Objetivo General

6.2. Objetivos Específicos

- Entender que son los operadores disfijos.
- Implementar un lenguaje de programación prueba de concepto que simule operadores disfijos mediante el lenguaje de programación Haskell.
- Utilizar los operadores disfijos del lenguaje como mecanismo de extensibilidad para implementar un lenguaje con características objetuales. Dicha construcción se realizará en etapas.

Capítulo 7

Importancia del problema.

Muchos de los avances en el desarrollo de software han tenido su origen en el area de lenguajes de programación. Los desarrollos en lenguajes de programación han permitido:

1. Escribir software en forma portable gracias a la posibilidad de abstraer los detalles arquitectónicos de la plataforma para la cual se desarrolla el software.

2. Escribir software mas seguro, ocultando y restringiendo ciertas construcciones inseguras del lenguaje objeto.[1]

3. Reutilización de código, mediante la incorporación de abstracciones para la generalización de soluciones. [2]

De la misma manera, desarrollos en lenguajes de programación han permitido el desarrollo de lenguajes mas expresivos. Entendiendo por lenguaje expresivo es aquel que brinda construcciones sintácticas y abstracciones que permiten al programador resolver problemas en forma mas "elegante" [3].

La importancia de la elegancia en los lenguajes de programación además de tener ventajas en mantenibilidad, y legibilidad tambien parece ser pieza importante en la reducción de errores. Existen indicios [14] para afirmar que la densidad de errores por número de líneas no se ve alterada por la elección del lenguaje de programación. De esta forma, lenguajes que requieren menos líneas de código para resolver diferentes problemas (lenguajes expresivos) pueden ayudar a disminuir la cantidad de errores.

Brindar expresividad enfrenta al diseñador del lenguaje de programación al dilema de Cardelli [4]: Decidir entre dar una amplia y expresiva notación o

[1] Entendiendo lenguaje objeto como el lenguaje producido por el lenguaje de programación. Por ejemplo, **x86** es uno de los lenguajes objeto de **C**

[2] Un ejemplo de ésto es Polimorfismo. El lenguaje diseñado en esta tésis tiene características polimórficas. Más sobre esto en la sección 4.

[3] Elegante puede tener connotaciones subjetivas, sin embargo el concepto es formalizado en [5]

tener un *"core"* pequeño. Ambas propiedades son deseables en un lenguaje de programación. Los *"cores"* pequeños son mas fáciles de mantener y permiten que el lenguaje sea asimilado más fácil por los programadores. Existe un enfoque híbrido basado en lenguajes extensibles. Estos lenguajes parten de un *"core"* pequeño, pero permiten que el usuario defina su propia sintáxis.

Esfuerzos para construir lenguajes que puedan "crecer" [12] se han realizado anteriormente. Dentro de las técnicas utilizadas en este enfoque hibrido cabe mencionar las siguientes: *"Syntax Macros"* [2], *"Extensible Syntax"* [4], *"Conctypes"* [1].

Este trabajo complementa dichos esfuerzos mostrando como un subconjunto de operadores disfijos puede ser utilizado como mecanismo de extensibilidad en los lenguajes de programación.

Capítulo 8

Preliminares

8.1. Conceptos de diseño de lenguajes de programación

Esta sección presenta conceptos generales sobre construcción de lenguajes de programación. Dichas definiciones han sido basadas en [9]

8.1.1. Sintáxis concreta versus sintáxis abstracta

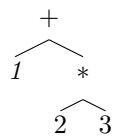
Definición 1. *Sintaxis concreta en lenguajes de programación se refiere a la representación del programa como cadenas de caracteres. Dicha representación es la que sirve de interfaz para el programador. La sintáxis concreta es el primer nivel en la definición de la sintaxis de un lenguaje.*

Ejemplo 1. *El siguiente es un ejemplo de sintaxis concreta para una expresión matemática.*

$$1 + 2 * 3$$

Definición 2. *Sintáxis Abstracta es el segundo nivel en la representación de sintáctica de un programa. Este nivel es alcanzado mediante dos procesos. Analisis léxico y parsing. El analisis léxico descompone la cadena de caracteres en "tokens" o lexemas como identificadores, literales y puntuación. El segundo paso transforma la lista de tokens en un árbol de sintaxis abstracta. Es en este paso es donde se resuelve la presedencia de operadores.*

Ejemplo 2. *El siguiente arbol representa la sintáxis abstracta del ejemplo presentado en 1*



Como se mencionó anteriormente la precedencia es resuelta en el paso anterior a la construcción del arbol sintáctico abstracto. De esta forma, la representación abstracta omite los parentesis.

8.1.2. Gramáticas libres de contexto, Notación y ambigüedad.

Una gramática libre de contexto es un formalismo desarrollado por Noam Chomsky para describir en forma recursiva la estructura de bloques de los lenguajes. Dicho formalismo es ampliamente utilizado para la descripción formal de los lenguajes de programación. Una de las razones es que dada la gramática libre de contexto, o un subconjunto de esta, es posible construir algoritmos que reconozcan el lenguaje expresado por dicha gramática en forma programática. Dos algorítmos ampliamante usados para este propósito son: *LL* y *LR*.

Formalmente, una gramática libre de contexto consta de un conjunto de símbolos terminales (V). Un conjunto de símbolos no terminales Σ, un conjunto de reglas generadoras $V \rightarrow (V \cup \Sigma)^*$. Y un $S \in V$ llamado símbolo inicial. Usualmente se utiliza convención Bakus-Naur o una de sus derivadas para definír la gramática. En este trabajo usaremos la siguiente convención:

Ejemplo 3. *Simple gramática para la definición de números.*

$\langle number \rangle \rightarrow \langle digit \rangle | \langle digit \rangle \langle number \rangle$
$\langle digit \rangle \rightarrow \mathbf{0}|\mathbf{1}|\mathbf{2}|\mathbf{3}|\mathbf{4}|\mathbf{5}|\mathbf{6}|\mathbf{7}|\mathbf{8}|\mathbf{9}$

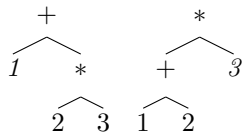
1. Símbolo inicial es el primero en la lista de "producciones'
2. Símbolos no terminales son aquellos en negrilla.
3. Símbolos no terminales están encerrados en $\langle\rangle$

Definición 3. *Una gramática libre de contexto es ambigua si existe una cadena perteneciente al lenguaje generado por dicha gramática y para esta, existe mas de un arbol sintáctico.*

Ejemplo 4. *El siguiente es un ejemplo de expresiones aritméticas ambiguas. (Por brevedad usaremos la misma definición de $\langle digit \rangle$ anteriormente mencionada)*

$\langle E \rangle \rightarrow \langle E \rangle + \langle E \rangle$
$\langle E \rangle \rightarrow \langle E \rangle * \langle E \rangle$
$\langle E \rangle \rightarrow \langle digit \rangle$

Dada la anterior gramática y la expresión $1+2*3$ *es posible construir dos árboles sintácticos.*



Esto es suficiente para concluir que la gramática es ambigua.

8.1.3. Operadores, precedencia y asociatividad

El ejemplo 4 muestra dos arboles sintácticos para la misma expresión aritmética. Solo uno de estos arboles cumple con el orden de evaluación esperado. El orden de evaluación esperado no es mas que el concepto de precedencia.

8.2. Lambda Calculo

Lambda cálculo es un formalismo matemático detras de la definición y aplicación de funciones. Dicho formalismo además representa la noción de computabilidad en forma equivalente a la maquina de Turing [3].

Lambda expresiones

El elemento constitutivo del lambda cálculo son "lambda expresiones".

Definición 4. *Una lambda expresión puede ser definida en forma recursiva de la siguiente manera:*

1. *Si* v *es una variable,* v *es una lambda expresión.*

2. *Si* v *es una variable y* M *es una lambda expresión,* $\lambda v.M$ *es una lambda expresión. A esto operación nos referiremos como abstracción de* M *sobre* v*.*

3. *Si* N *y* M *son lambda expresiones entonces* N M *también es una expresión. Esto es llamado aplicación (*M *se aplica a* N*)*

Variables Libres

La operación de abstracción liga[1] la variable asociada. Son variables libres aquellas que no están ligadas por ninguna abstracción.

Ejemplo 5. *En* $\lambda x.x\,y$ *la variable* x *en la expresión está ligada y la variable* y *está libre.*

[1] El termino usual en la literatura es *"bind"*.

α-Conversión

Dos expresiones son alfa-equivalentes (denotado por $=_\alpha$) si es posible transformar una en otra, renombrando las variables.

Ejemplo 6. $\lambda x.x\, y =_\alpha \lambda z.z\, y$

Renombrar las variables ligadas de una lambda expresión es conocido como α-Conversión. Existen dos casos especiales para tener en cuenta al realizar α-Conversión. En primer lugar, no es posible realizar α-Conversión de una expresión por una variable que se encuentre libre en dicha expresión. Esto es llamado captura de nombres[2].

Ejemplo 7. $\lambda x.x\, y \neq_\alpha \lambda y.y\, y$

El segundo caso se refiere al ámbito de las abstracciones. $\lambda x.\lambda x\, y$

Notación

Por practicidad seguiremos ciertas convenciones notacionales descritas en [11].

1. Los paréntesis más externos serán omitidos. De esa forma $(N\, M)$ será escrito: $N\, M$
2. La abstracción se extiende hasta donde sea posible. $\lambda x.N\, M$ es equivalente a $\lambda x.(N\, M)$ y no a $(\lambda x.N)M$
3. Aplicación es asociativa por izquierda. $M\, N\, O$ es equivalente a $(M\, N)\, O$ y no a $M(N\, O)$
4. Podemos abreviar la expresión $\lambda x.\lambda y.M$ de la forma: $\lambda x\, y.M$

Lambda cálculo puro versus lambda cálculo aplicado.

Existen diferentes codificaciones bien conocidas usando lambda expresiones para diferentes tipos de datos usuales en lenguajes de programación [3]. Dichas codificaciones resultan convenientes como mecanismo de exploración del lambda cálculo como formalismo de computabilidad; sin embargo, ineficiente a la hora de implementar lenguajes de programación. Debido a esto suelen añadirse construcciones al lambda cálculo "puro.[ex]plicado anteriormente. Dichos lambda cálculos extendidos suelen denominarse lambda cálculos aplicados. A continuación se ilustra la relación de ambos conceptos [4]

$$\begin{aligned} \textit{programming language} &= \textit{applied lambda calculus} \\ &= \textit{pure lambda system} + \textit{basic data types} \end{aligned}$$

Ahora agregaremos 2 nuevas reglas a la definición 4 de lambda expresión. Siguiendo la terminología usada lo resultante es un lambda cálculo aplicado.

[2] En la literatura se refieren a *"name capture"*
[3] El capítulo 3 de [11] incluye dichas codificaciones para enteros, tuplas, listas arboles
[4] Tomado de [7] Página 370.

1. Si k es un entero o k es una cadena de caracteres entonces k es una constante.
2. Si k es una constante entonces k es una lambda expresión.

8.2.1. Estrategias de Reducción

8.2.2. Sistema de Tipos

Capítulo 9

Parsing, Análisis y Evaluación.

Esta sección explica los diferentes pasos en la implementación de nuestro lenguaje para soportar operadores permisivos.

9.1. Una visión general

Inicialmente

$$Texto \rightarrow_1 Tokens \rightarrow_2 (Declaraciones, Main)$$

9.2. Lexing y Parsing

El primer paso de parsing detecta las declaraciones de funciones y la función principal main. Esta es la gramática del lenguaje siguiendo la notación del capítulo 3.

⟨Program⟩→⟨Declarations⟩⟨Main⟩

⟨Declarations⟩→⟨Declarations⟩⟨Declaration⟩|ϵ

⟨Declaration⟩→⟨Infix⟩|⟨Prefix⟩|⟨Suffix⟩|⟨Closed⟩

⟨Infix⟩→⟨Init⟩⟨InfixKeyword⟩⟨Precedence⟩⟨Id⟩⟨Id⟩⟨Id⟩=⟨Definition⟩

⟨Precedence⟩→ϵ⟨Number⟩

⟨InfixKeyword⟩→**infixr**|**infixl**

⟨Init⟩→**let**|**let rec**

⟨Prefix⟩→⟨Init⟩⟨Id⟩⟨Ids⟩=⟨Definition⟩

⟨Suffix⟩→⟨Init⟩**suffix**⟨Ids⟩⟨Id⟩=⟨Definition⟩

⟨Closed⟩→⟨Init⟩**closed**IdIdsId=⟨Definition⟩

⟨Ids⟩→⟨Id⟩⟨Ids⟩|ϵ
⟨Definition⟩→⟨ExpresionTokens⟩
⟨ExprTokens⟩→⟨ExprToken⟩⟨ExprTokens⟩|ϵ
⟨ExprToken⟩→⟨Literal⟩|⟨Id⟩
⟨Literal⟩→⟨Number⟩|⟨String⟩
⟨Number⟩→⟨Digit⟩|⟨Digit⟩⟨Number⟩
⟨Digit⟩→**0**|**1**|**2**|**3**|**4**|**5**|**6**|**7**|**8**|**9**
⟨String⟩→**"**⟨Chars⟩**"**
⟨Chars⟩→⟨Char⟩⟨Chars⟩|ϵ
⟨Char⟩→Cualquier caracter
⟨Id⟩→⟨Char⟩|⟨Char⟩⟨Id⟩
⟨Main⟩→**main =** |⟨Definition⟩

Ahora bien. Para definir una expresión añadiremos notación adicional: Op_p^a Indica que el operador tiene precedencia p asociatividad a. Usaremos un entero para denotar p y $\{l, r\}$ son los posibles valores de a. l denota asociatividad por izquierda y r asociatividad por derecha.

$$Expr := Open\, Expr\, Closed$$

$$Expr := Expr_1 Infix_n^r\, Expr_2 | \forall_{Op_m^r \in Expr_1 or Expr_2} n \leq m$$

En español... Una expresión puede estar formada por un operador infijo operando 2 subexpresiones. siempre y cuando el operador tenga la menor precedencia. En la defición falta agregar el manejo de asociatividad (en el caso de $Infix_n^r$ se debe garantizar que no exista ningún operador con la misma precedencia y misma asociatividad en $Expr_2$).

$$Expr := Expr_1 Infix_n^l\, Expr_2 | \forall_{Op_m^j \in Expr_1 or Expr_2} n \leq m$$

$$Expr := PrefixOp\, Expr$$

$$Expr := Expr\, SuffixOp$$

$$Expr := Literal$$

$$Literal := string | natural$$

9.3. Resolución de operadores

Data: Lista de ExprTokens
Result: Arbol de evaluación
initialization;
if *understand* **then**
| go to next section;
end

Algoritmo 1: Resolución de precedencia

9.4. Sistema de tipos

El lenguaje de programación es fuertemente tipado y está construido a partir de la implementación encontrada en [6]. El sistema de tipos esta conformado por los siguientes constructores en Haskell.

```
data Type    =  TVar String
             |  TInt
             |  TBool
             |  TString
             |  TList Type
             |  TProd Type Type
             |  TFun Type Type
```

El sistema de tipos cuenta con variables de tipos, enteros, booleanos, cadenas de caracteres, listas genéricas, funciones y producto de tipos.

9.5. Construcción de notación aritmética

9.6.

Capítulo 10

Conclusiones

Bibliografía

[1] Anika Aasa. *User Defined Syntax*. PhD thesis, Chalmers University of Technology, 1992.

[2] Arthur Baars and Doaitse Swierstra. Syntax macros. Unfinished Draft.

[3] Henk Barendregt and Erik Barendsen. Introduction to lambda calculus. Disponible en línea: `http://www.cs.chalmers.se/Cs/Research/Logic/TypesSS05/Extra/geuvers.pdf`, March 2000.

[4] Luca Cardelli, Florian Matthes, and Martin Abadi. Extensible syntax with lexical scoping. Technical report, Systems Research Center, 1994.

[5] G. J. Chaitin. *People and Ideas in Theoretical Computer Science*, chapter Elegant LISP Programs. Springer-Verlag, 1999. Disponible en línea: `http://www.cs.auckland.ac.nz/CDMTCS//chaitin/lisp.html`.

[6] Martin Grabmüller. Algorithm w step by step. Draft paper, September 2007.

[7] Jan Leeuwen and Jan van Leeuwen, editors. *Handbook of Theoretical Computer Science: Formal models and semantics*. MIT Press, 1994.

[8] Francisco Piedrahita. Retazos de historia. Technical Report 88416316, 1986.

[9] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.

[10] Maribel Serna Rodríguez and Andrés Mauricio Mora Cuartas. La bolsa de valores de colombia: Su historia y relación con la universidad eafit. MERCADOS FINANCIEROS 003891, GRUPO DE INVESTIGACIÓN EN FINANZAS Y BANCA - EAFIT, July 2007.

[11] Peter Selinger. Lecture notes on the lambda calculus. Disponible en línea `http://www.mathstat.dal.ca/~selinger/papers/lambdanotes.pdf`. Notas de un curso de lambda cálculo de la universidad de Ottawa de 2001.

[12] Guy L. Steele, Jr. Growing a language. *Higher Order Symbol. Comput.*, 12(3):221–236, 1999.

[13] Dave Thomas, David Hansson, Leon Breedt, Mike Clark, James Duncan Davidson, Justin Gehtland, and Andreas Schwarz. Agile web development with rails. Technical report, 2006.

[14] Ulf Wiger. Four-fold increase in productivity and quality. Technical report, Ericsson, 2001.