UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CIENCIAS

ANIMACIÓN DE ALGORITMOS EN LA PROGRAMACIÓN ORIENTADA A OBJETOS

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A :

SERGIO PADILLA REYNAUD

DRA. AMPARO LÓPEZ GAONA

2008

FACULTAD DE CIENCIAS
UNAM

Hoja de Datos del Jurado

1.Datos del alumno Apellido paterno Apellido materno Nombre(s) Teléfono Universidad Facultad o escuela Carrera Número de cuenta	1. Datos del alumno Padilla Reynaud Sergio 56 96 81 76 Universidad Nacional Autonoma de Mexico Facultad de Ciencias Ciencias de la Computación 094361654
2. Datos del tutor Grado Nombre(s) Apellido paterno Apellido materno	2. Datos del tutor Dra Amparo Lopéz Gaona
3. Datos del sinodal 1 Grado Nombre(s) Apellido paterno Apellido materno	3. Datos del sinodal 1 M en C María Guadalupe Elena Ibargüengoitia González
4. Datos del sinodal 2 Grado Nombre(s) Apellido paterno Apellido materno	4. Datos del sinodal 2 Mat Salvador López Mendoza
5. Datos del sinodal 3 Grado Nombre(s) Apellido paterno Apellido materno	5. Datos del sinodal 3 M en C Gustavo Arturo Márquez Flores
6. Datos del sinodal 4 Grado Nombre(s) Apellido paterno Apellido materno	6. Datos del sinodal 4 M en C Egar Arturo García Cárdenas
7.Datos del trabajo escrito. Título Subtitulo Número de páginas Año	7. Datos del trabajo escrito Animación de Algoritmos en la programación orientada a objetos 69 p 2008

Índice

Introducción.

La creación de un programa de computadora generalmente involucra un conjunto de conocimientos difíciles de comprender y abstraer. Al inicio, uno se encuentra con una gran cantidad de conceptos que aprender, que van desde la lógica de la programación hasta el diseño del mismo o su eficiencia, y desde luego, el conocimiento del lenguaje de programación, así como el paradigma de programación que se desee emplear. En muchas ocasiones la curva de aprendizaje resulta muy alta, lo cual desemboca en mucho tiempo de aprendizaje y poca asimilación de los conceptos más profundos de la programación [6].

El paradigma de programación en ocasiones resulta ser uno de los pasos más difíciles de superar, ya que implica pensar de una forma especial los programas para que se apeguen al paradigma a emplear [6].

En el caso particular de la programación orientada a objetos, la abstracción podría parecer un poco más sencilla al poder encontrar similitudes entre objetos reales y objetos del paradigma de programación, sin embargo, esto no siempre es aplicable, ya que no siempre se puede hacer dicha abstracción, debido a que los objetos que queremos modelar, muchas veces también son abstracciones. Este tipo de dificultades son las que presentan más dudas o conflictos para aprender este paradigma.

El intentar explicar esta forma de programación no siempre resulta fácil, en ocasiones la bibliografía resulta obscura o necesita de otros conocimientos previos. Con la práctica se mejoran habilidades, pero no siempre se resuelven todas las dudas o no se alcanzan a comprender cabalmente todos los conceptos. Sin embargo, también es cierto que hay una

carencia de herramientas que ayuden a comprender mejor estos conceptos, que ayuden a representar de una forma ágil y gráfica lo que se pretende explicar con solo líneas de código y teoría.

Estas herramientas deberían de ser capaces de mostrar cómo es que un programa se ejecuta y cómo se relacionan todas sus partes, sus estados, operaciones y resultados.

En el caso particular de la programación orientada a objetos, interesaría conocer el momento en que se crea (instancía) un objeto, el estado de sus atributos según la ejecución del programa y cómo afectan los métodos de las clases de dichos objetos a estos atributos, junto con las relaciones que puedan tener entre objetos. Con una representación gráfica síncrona de lo que ocurre en el código de un programa y lo que pasa con los objetos de dicho programa, se facilita explicar lo que ocurre con un programa al ejecutarse, y se comprende más fácilmente lo que está ocurriendo y puede resultar en la disminución de la curva de aprendizaje [1].

Objetivos.

La presente tesis tiene como principal objetivo el desarrollar una herramienta sencilla e intuitiva que ayude a tener una mejor comprensión de los conceptos de la programación orientada a objetos, así como el entendimiento claro de la ejecución de un programa creado bajo este paradigma.

Para lograr dicho objetivo se necesitará analizar la teoría existente sobre la visualización de algoritmos y la historia de algunos sistemas previos creados para facilitar el aprendizaje de otros conceptos diferentes a la programación pero con igual o mayor dificultad para asimilar todos los conceptos necesarios.

De igual forma, esta tesis intentará indagar cuáles son las partes más difíciles del aprendizaje en la programación orientada a objetos, para esto será preciso repasar los conceptos fundamentales de este paradigma de programación y parte de su historia.

Con esta información en esta tesis se evaluarán las ventajas e inconvenientes de la creación de un sistema con los ejemplos principales de la programación orientada a objetos y tratando de encontrar y emplear las maneras más simples de representar los conceptos involucrados en la creación y ejecución de un programa creado bajo el paradigma de orientación a objetos.

Estructura.

Para lograr los objetivos previamente descritos, esta tesis está conformada por los capítulos siguientes:

Capítulo 1. Conceptos de la programación orientada a objetos.

En este capítulo se revisa la historia de la programación orientada a objetos, conociendo un poco la motivación que llevo a sus creadores a pensar en un modelo orientado a objetos, posteriormente se estudian las características y la metodología de dicha programación repasando los conceptos básicos e indispensables para un buen conocimiento del paradigma.

Conjuntamente, se hace un listado de los lenguajes orientados a objetos y se realiza una comparación breve con la programación orientada a procedimientos.

Capítulo 2. Animación de algoritmos.

En esta sección se revisa la teoría referente a la visualización de animaciones graficas junto con los retos y dificultades que estos programas presentan. Además, se hace un breve recorrido por la historia de la animación de algoritmos, y después se analizan los intentos previos por crear sistemas o entornos que ayuden a tener una mejor compresión al usuario, que le aminoren la carga cognoscitiva o le planteen retos intelectuales que le ayuden a reforzar lo aprendido.

Capítulo 3. Sistema de animación de algoritmos en la programación orientada a objetos.

En esta parte se describen los componentes y las funcionalidades con las que cuenta el sistema creado en esta tesis.

También se agregan los diagramas y documentos relativos a la ingeniería de software involucrada en la creación del sistema.

Finalmente se presentaran las conclusiones de este trabajo, describiendo los beneficios que se obtienen con el uso de esta herramienta.

1. Conceptos de la programación orientada a objetos.

La programación orientada a objetos (POO u OOP según sus siglas en inglés) es una metodología de diseño de software y un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos) y comportamiento (esto es, procedimientos o métodos). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que se comunican entre ellos para realizar tareas. Esto difiere de los lenguajes orientados a procedimientos tradicionales, en los que los datos y los procedimientos están separados y sin relación. Estos métodos están pensados para hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

Otra manera en que esto es expresado a menudo, es que la programación orientada a objetos anima al programador a pensar en los programas principalmente en términos de tipos de datos, y en segundo lugar en las operaciones (métodos) específicas a esos tipos de datos. Los lenguajes orientados a procedimientos animan al programador a pensar sobre todo en términos de procedimientos, y en segundo lugar en los datos que esos procedimientos manejan.

Los programadores que emplean lenguajes orientados a procedimientos, escriben funciones y después les pasan datos. Los programadores que emplean lenguajes orientados a objetos definen objetos con datos y métodos y después envían mensajes a los objetos solicitando que realicen esos métodos en sí mismos.

Algunas personas también diferencian la POO sin clases, la cual es llamada en ocasiones programación basada en objetos.

1.1. Historia

Los conceptos de la programación orientada a objetos tienen origen en Simula 67, un lenguaje diseñado para hacer simulaciones, creado por Ole-Johan Dahl y Kristen Nygaard del Centro de Cómputo Noruego en Oslo. Según se informa, la historia es que trabajaban en simulaciones de naves, y fueron confundidos por la explosión combinatoria de cómo las diversas cualidades de diversas naves podían afectar unas a las otras [7]. La idea ocurrió para agrupar los diversos tipos de naves en diversas clases de objetos, siendo responsable cada clase de objetos de definir sus propios datos y comportamiento. La idea de los objetos fue refinada más tarde en Smalltalk, que fue desarrollado en Simula en Xerox PARC pero diseñado para ser un sistema completamente dinámico en el cual los objetos se podrían crear y modificar "en marcha" en lugar de tener un sistema basado en programas estáticos.

Uno de los problemas al inicio de los años setentas era que pocos sistemas lograban terminarse, pocos se terminaban con los requisitos iniciales y no todos los que se terminaban cumpliendo con los requerimientos se usaban según lo planificado.

El problema consistía en cómo adaptar el software a nuevos requerimientos imposibles de haber sido planificados inicialmente.

Este alto grado de planificación y previsión es contrario a la propia realidad. El hombre aprende y crea a través de la experimentación, no de la planeación. La orientación a objetos brinda estos métodos de experimentación, no exige la planificación de un proyecto por completo antes de escribir la primera línea de código.

Hasta este momento de los años setentas, uno de los defectos más graves de la programación orientada a objetos es que las variables eran visibles desde cualquier parte del código y podían ser modificadas, incluyendo la posibilidad de cambiar su contenido (no existen niveles de usuarios o de seguridad, o lo que se conoce como visibilidad).

Quien tuvo la idea de restringir el alcance de las variables, fue David L. Parnas [8] cuando propuso la disciplina de ocultar la información. Su idea era encapsular cada una de las variables globales de la aplicación en un solo módulo junto con sus operaciones asociadas, sólo mediante las cuales se podía tener acceso a esas variables.

El resto de los módulos (objetos) podían acceder a las variables sólo de forma indirecta mediante las operaciones diseñadas para tal efecto. Esto aún no era lo que hoy en día conocemos como programación orientada a objetos

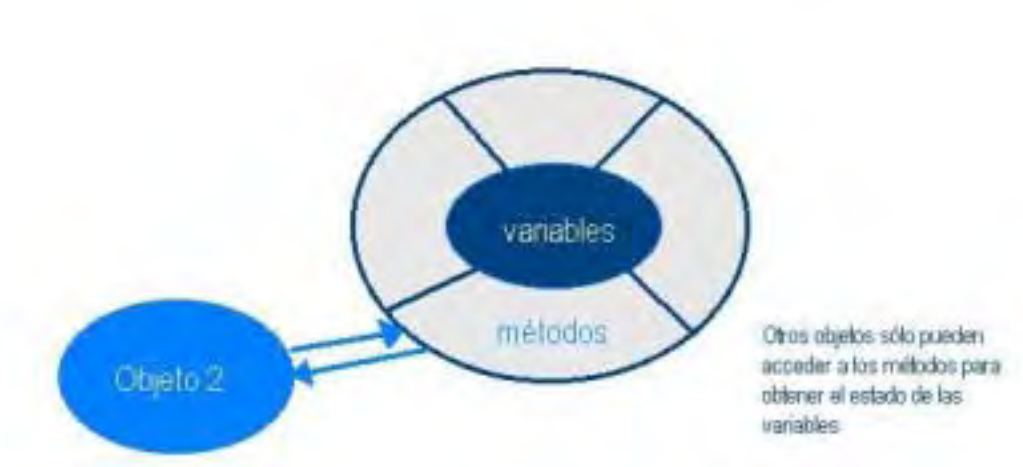

Figura 1.- Objetos accediendo a variables a través de métodos.

La programación orientada a objetos tomó posición como la metodología de programación dominante a fines de los años ochenta, en gran parte debido a la influencia de Smalltalk y más tarde a C++ , una extensión del lenguaje de programación C creado por Bjarne Stroustrup de AT&T Labs. Su dominación fue consolidada gracias al auge de las interfaces gráficas de usuario, para las cuales la programación orientada a objetos está particularmente bien adaptada.

En esa misma década se desarrollaron otros lenguajes Orientados a Objetos como Objective C, Common Lisp Object System (CIOS), Ada y otros. Otros lenguajes que originalmente no eran orientado a objetos, fueron agregando módulos que les permitieron adoptar el paradigma como el Object Pascal.

La adición de estas características a los lenguajes que no fueron diseñados inicialmente para ellas condujo a menudo a problemas de compatibilidad y a la capacidad de mantenimiento del código. Los lenguajes orientados a objetos "puros", por otra parte, carecían de las características de las cuales muchos programadores habían venido dependiendo. Para saltar este obstáculo, se hicieron muchas tentativas para crear nuevos lenguajes basados en métodos orientados a objetos, pero permitiendo algunas características de la orientación a procedimientos de maneras "seguras".

El Eiffel de Bertrand Meyer fue un temprano y moderadamente acertado lenguaje con esos objetivos.

En el inicio de los 90’s se consolida la orientación a objetos como una de las mejores maneras para resolver problemas. Aumenta la necesidad de generar prototipos más rápidamente (concepto RAD Rapid Aplication Developments), sin esperar a que los requerimientos iniciales estén totalmente precisos.

En 1990 surge un desarrollo llamado Java, creado por James Gosling, bajo el auspicio de Sun Microsistems. Su filosofía es aprovechar el software existente y facilitar la adaptación del mismo a otros usos diferentes a los originales sin necesidad de modificar el código ya existente.

En 1997-98 se desarrollan herramientas 'CASE' orientadas a objetos (como el diseño asistido por computadora).

De 1998 a la fecha se desarrolla la arquitectura de objetos distribuidos RMI (Remote Method Invocation), CORBA (Common Object Request Broker Architecture), COM (Component Object Model), DCOM (Distributed COM).

Entre los creadores de metodologías orientadas a objetos se encuentran: Grady Booch, James Rambaught, Ivar Jacobson y Peter Cheng.

Actualmente la orientación a objetos parece ser el mejor paradigma, no obstante, no es una solución a todos los problemas [9].

1.1.1 Complejidad.

La complejidad es otra característica propia del software. Existen dos tipos de complejidades:

1. Complejidad irregular. Es a la que se enfrenta un programador solitario, por ejemplo, simular una calculadora (esta complejidad es la menor en comparación al otro tipo).
2. Complejidad industrial. Cuando están involucrados más de un programador, implica muchos algoritmos. Una característica distintiva del software industrial es que resulta sumamente difícil para el desarrollador individual comprender todas las sutilezas de su diseño. Lamentablemente, parece ser una propiedad esencial de todos los sistemas de software de gran tamaño.

Todo software tiene una complejidad implícita derivada de los siguientes factores:

- La complejidad del dominio del problema. Surge habitualmente entre los usuarios del sistema y sus desarrolladores; los usuarios suelen encontrar grandes dificultades al intentar expresar sus necesidades de tal manera que los desarrolladores las entiendan.
- Complejidad surgida durante el desarrollo del software. Esta complejidad se refiere a que los requisitos de un software cambian frecuentemente durante su desarrollo, especialmente porque la mera existencia del proyecto altera las reglas del problema.

- Complejidad de evolución con respecto al tiempo. Por ejemplo, hace algunas décadas los datos que se manejaban era mucho menor en contenido y en variedad que los que en estos momentos se pueden manejar. (Ver figura 2)

Figura 2.- Diferencias de contenidos en los años 80´s a la fecha.

1.1.2 Atributos de un sistema complejo

- La complejidad toma la forma de una jerarquía. Un sistema complejo se compone de subsistemas relacionados que tienen a su vez sus propios subsistemas, y así sucesivamente hasta que alcanza un nivel de componentes elementales.
- La elección de qué componentes de un sistema son elementales es relativamente arbitraria, y queda en gran medida a decisión del observador.
- Los sistemas jerárquicos están compuestos usualmente de sólo unas pocas clases diferentes de subsistemas en varias combinaciones y disposiciones.

Debido a esta complejidad, y a la estructura que se puede generar con la programación orientada a objetos, es posible crear software con las características necesarias para solucionar de una forma más eficiente los problemas mencionados anteriormente. [5]

1.2. Resolución de problemas con software orientado a objetos

La orientación a objetos está basada en los siguientes métodos de organización:

a) Entre un objeto y sus atributos. Por ejemplo, un automóvil tiene marca, color, número de llantas, etc.
b) Entre un objeto y sus componentes donde incluso otros objetos pueden formar parte de objetos (agregación). Por ejemplo, un camión tiene motor, parabrisas, llantas, donde motor es a su vez otro objeto.
c) Entre un objeto y su relación con otros objetos. Por ejemplo, un camión pertenecería a los vehículos automotores; una bicicleta no entraría en esta relación.

El proceso del desarrollo de software orientado a objetos consiste en:

- Conocer el espacio del problema.
- Realizar una abstracción de problema.
- Crear los objetos (espacio de la solución).
- Dejar que los objetos cubran sus funciones (operen sus métodos).

1.3. La programación orientada a objetos como solución.

En términos de la orientación a objetos, la forma de solucionar un problema es iniciando una acción mediante la transmisión de un mensaje a un objeto responsable de realizarla. El mensaje codificará la petición, y en algunas ocasiones, se necesitará de información adicional, que podrá ser enviada como parámetro. El objeto receptor es el agente al que se le envía el mensaje, si éste lo acepta, también acepta la responsabilidad de realizar la acción solicitada y como respuesta este receptor empleará algún método que satisfaga la petición.

La programación orientada a objetos es una forma de programar que proliferó a partir de los años ochentas, la cual emplea los siguientes conceptos:

- Objetos: Entidades complejas provistas de datos (propiedades, atributos) y comportamiento (funcionalidad, métodos) que corresponden a los objetos reales del mundo que nos rodea. Los atributos o variables especifican la estructura del objeto y su valor, el cual está asociado al estado del objeto. Para alterar el estado de un objeto se necesita invocar un método. Este método es algo que el objeto sabe y por lo tanto define su comportamiento.

Los objetos comprenden 3 tipos de componentes: una estructura, algún comportamiento y una identidad. La estructura está formada por elementos o propiedades que describen al objeto, la definición de estas propiedades depende de la aplicación en particular del objeto, a esta tarea se le conoce como abstracción y más adelante se profundizará en este tema. La identidad es la propiedad que permite distinguir entre objetos que tienen la misma estructura y comportamiento. Es posible tener objetos con estado y comportamientos iguales pero con diferente identidad, esto se puede ejemplificar con automóviles de la misma marca y modelo pero con diferentes números de placas. Así pues, la parte estática del objeto está dada por su estructura, entonces su comportamiento formaría la parte dinámica. El comportamiento está definido por las tareas que puede realizar el objeto. Estas peticiones de servicios entre objetos se realizan a través de mensajes.

El objeto que solicita un servicio se denomina cliente o actor, mientras que el objeto que lo atiende se le conoce como agente o servidor.

- Clases: son conjuntos de objetos que comparten propiedades y comportamiento. Las clases representan la esencia de los objetos. Así pues, los objetos son ejemplares o mejor dicho, instancias de las clases. Instanciar es el proceso de creación de un objeto según la definición de la clase a la cual pertenece. Se puede considerar a una clase como un patrón para la creación de objetos de dicha clase.

  En la clase se define la estructura y comportamiento de los objetos y la identidad es proporcionada al momento de crear el objeto.

En las clases hay elementos que son visibles al exterior de la misma, como las interfaces, que son listas de métodos que la clase es capaz de atender. Las partes de la clase que permanecen ocultas al exterior nos llevan al concepto de encapsulación, el cual se ampliará posteriormente.

- Método: es un programa asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena mediante un "mensaje". En analogía con un lenguaje orientado a procedimiento se le llamaría "función".

  En la programación orientada a objetos existen cuatro tipos de métodos:

  1. Constructores: Permiten la creación de objetos asignándoles un estado inicial. La única forma de crear un objeto es a través de la llamada a un método constructor.

  2. Mutadores: Cambian el estado de un objeto. La única forma de cambiar el estado de un objeto es a través de una llamada a estos métodos.

  3. Inspectores o de acceso: Examinan el estado actual de un objeto. Estos métodos a diferencia de los anteriores, no cambian el estado de un objeto, solo obtienen la información referente al estado del objeto.

  4. Calculadores o manipuladores: Obtienen otro tipo de resultados a partir del estado del objeto y/o enviando mensajes a otro objeto. Estos datos

resultantes pueden ser consecuencia de cálculos o el cambio de caracteres o de palabras completas.

- Mensaje: una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros.

- Propiedad, atributo o variable: datos asociados a un objeto o a una clase de objetos.

- Herencia: las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen.

- Encapsulación: cada objeto está aislado del exterior, es un módulo natural, y la aplicación entera se reduce a un agregado o rompecabezas de objetos. El aislamiento protege a los datos asociados a un objeto contra su modificación por quien no tenga derecho a acceder a ellos, eliminando efectos secundarios e interacciones.

- Polimorfismo: programas diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, aunque el significado del programa varíe según el objeto al que se aplica.

1.4. Características de la programación orientada a objetos.

Algunas de las características más importantes son las siguientes:

- Abstracción: Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos y cuando los están, una variedad de técnicas son requeridas para ampliar una abstracción. Una clase abstracta es una clase que sirve para especificar el comportamiento deseado de las subclases pero sin entrar en detalles de cómo hacerlo, por lo mismo, no es posible crear ejemplares o instanciar una clase abstracta. Por ejemplo, suponiendo que tenemos una clase Animal, donde se especifican los métodos relativos a las acciones que puede realizar un animal, ya sea el caminar o el tipo de sonido que hace, junto con otros métodos más. Las subclases de Animal, implementarían los métodos de caminar, relativo a cada animal, ya sea en dos patas, cuatro o las necesarias, junto con el orden y cadencia, igualmente ocurriría con el sonido relativo a cada animal, ya sea mugir, bramar, piar, rugir, etc.

- Encapsulación: También llamada "ocultación de la información", esto asegura que los objetos no pueden cambiar el estado interno de otros objetos de maneras inesperadas; solamente los propios métodos internos del objeto pueden acceder a

su estado. Cada tipo de objeto expone una interfaz a otros objetos que especifica cómo otros objetos pueden interactuar con él. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. Como ejemplo podemos suponer la clase Coche, la cual puede tener diferentes instancias según su modelo, número de cilindros, volumen de los cilindros, número de ejes, etc. Sin embargo cualquiera de estas implementaciones tendría un método relacionado con el comportamiento del coche al girar, sin bien no es necesario conocer como realiza cada clase esta operación, simplemente que puede llevarse a acabo sin necesidad de conocer detalles particulares.

- Polimorfismo: Las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del referente. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama asignación tardía o asignación dinámica. Algunos lenguajes proporcionan medios más estáticos (en "tiempo de compilación") de polimorfismo, tales como las plantillas y la sobrecarga de operadores de C++. Entonces, el polimorfismo es la habilidad de que dos o más objetos de diferentes clases respondan al mismo mensaje en diferentes formas según la clase a la que pertenecen, esto es, el utilizar el mismo nombre para diferentes métodos o mejor dicho, que estos métodos con mismos nombres se implementen de diferente forma. Imaginemos la clase Figura, la cual tiene dos subclases llamadas círculo y triángulo, se podría crear una

instancia de cualquiera de estas dos clases como si fuera de la clase padre, es decir Figura y no presentaría ningún error, este es un ejemplo de asignación tardía.

- Herencia: Organiza y facilita el polimorfismo y la encapsulación permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que reimplementar su comportamiento. Esto suele hacerse habitualmente agrupando los objetos en clases y las clases en árboles o matrices que reflejan un comportamiento común. La herencia en si, define relaciones entre clases que comparten estructura o comportamiento. Las clases con estructura y comportamiento común se les llama superclase o clase base, y las clases que añaden o redefinen componentes de su superclase se les conocen como subclases, clases hijas o clases derivadas. Al tener herencia, las clases pueden ser organizadas en jerarquías similares a las de un árbol genealógico. Por consiguiente, las subclases heredan características (atributos y comportamientos) de las subclases en el árbol. Pensemos en una clase raíz llamada Transporte, de esta pueden derivarse clases mas especificas como podrían ser la clase Automóvil, la clase Avión o la clase Barco entre otras. Las cuales tendrían comportamientos similares, y desde luego también tendrían comportamientos distintos según las características propias de cada uno, como el tipo de medio en el que se desplazan, el tipo de combustible que consumen y el tipo de objetos que trasportan, entre otras cualidades.

1.5. Lenguajes orientados a objetos.

Entre los lenguajes orientados a objetos destacan los siguientes:

Smalltalk, Simula, Objective-C, C++, Ada 95, Java, Ocaml, Phyton, Delphi, Lexico (en castellano), C Sharp, Eiffel, Ruby, ActionScript, .NET, PHP entre otros. Estos lenguajes de programación son puros a la orientación a objetos.

Al igual que C++ otros lenguajes, como OOCOBOL, OOLISP, OOPROLOG y OOREXX, han sido creados añadiendo extensiones orientadas a objetos a un lenguaje de programación clásico.

Algunas características de algunos de estos lenguajes son:

Smalltalk:

-El estilo del lenguaje se enfatiza la ligadura dinámica y no realiza chequeo de tipos.

- El bloque fundamental es la clase, que contiene la descripción de variables y métodos.

- Los métodos contienen el código, y definen cómo responde un objeto a un mensaje.

- La ejecución de una rutina de un objeto se denomina en la terminología de Smalltalk "enviar un mensaje" al objeto, cuya clase encontrará la forma apropiada de manejar dicho mensaje.

- Todo en Smalltalk son objetos, incluyendo a las propias clases.

- El entorno de Smalltalk permite un rápido desarrollo de programas.

- La biblioteca de clases fue diseñada para ser extendida y adaptada por adición de subclases para satisfacer las necesidades de la aplicación.

- La arquitectura Modelo/Vista/Controlador (MVC) se comienza a gestar gracias a Smalltalk entre otros.

C++:

- Dispone de capacidades para la herencia simple y múltiple.
- Por defecto C++ tiene ligadura estática.
- Las funciones definidas como virtuales pueden beneficiarse del concepto de ligadura dinámica.
- Esquema de tipos estricto, aunque admite la posibilidad de casting.
- No cuenta con un recolector de basura.
- La ausencia de un gestor automático de memoria por defecto, obliga al concepto de destructor.
- Ocultamiento de la información, incluso a los descendientes.
- Cuenta con un mecanismo de acceso especial por parte de una clase o una función que se declare como friend.
- Soporte de manejo de excepciones.
- Permite sobrecarga de operadores.
- Lenguaje complejo, preocupado por la detección temprana de errores, y por la eficiencia de la ejecución, perdiendo algo de sencillez y de flexibilidad para el diseño.

Eiffel:

- Es un lenguaje orientado al objeto puro.
- Soporta ligadura dinámica.
- Tiene comprobación estricta de tipos.
- Admite herencia múltiple.
- Soporta clases parametrizadas.

- La gestión de memoria la lleva a cabo el entorno de programación.

- Eiffel proporciona una biblioteca de clases predefinidas muy completa.

- Eiffel cuenta con la clase como único criterio de estructuración.

- Una declaración de clase en Eiffel puede incluir una lista de características exportadas, una lista de clases predecesoras, y una lista de declaraciones de características.

- Cuenta con manejo de excepciones.

Java:

- Soporta threads o hilos de ejecución.

-Recogida de basura automática.

- Gracias al recolector de basura no son necesarios los métodos destructores.

- Java produce un bytecode que será interpretado por una máquina virtual.

- La máquina virtual se encuentra a menudo en navegadores web.

- Cuenta con manejo de excepciones.

- Esquema de tipos estricto, aunque admite la posibilidad de casting.

- Cuenta con herencia, encapsulación y polimorfismo.

- Administra la memoria automáticamente.

1.6. Diferencias con la programación orientada a procedimientos.

La programación orientada a procedimientos evolucionó hacia la programación estructurada y los métodos actuales de diseño de software orientado a objetos. Todo esto incluye refinamientos tales como el uso de los patrones de diseño, diseño por contrato, y lenguajes de modelado (tales como UML).

La programación orientada a procedimientos clásica presenta ciertos problemas, que fueron haciéndose más graves, a medida que se construían aplicaciones y sistemas informáticos más complejos, entre los que destacan los siguientes:

- Modelo mental anómalo. Nuestra imagen del mundo se apoya en los seres, a los que asignamos nombres sustantivos, mientras la programación clásica se basa en el comportamiento, representado usualmente por verbos.

- Dificultad para modificar y extender los programas, pues suele haber datos compartidos por varios subprogramas, que introducen interacciones ocultas entre ellos.

- Dificultad para mantener los programas. Casi todos los sistemas informáticos grandes tienen errores ocultos, que no surgen a la luz hasta después de muchas horas de funcionamiento.

- Dificultad para reutilizar los programas. Es prácticamente imposible aprovechar en una aplicación nueva las subrutinas que se diseñaron para otra.

En la programación orientada a objetos pura no deben utilizarse llamadas de subrutinas, únicamente mensajes. Por ello, a veces recibe el nombre de programación sin CALL, igual que la programación estructurada se llama también programación sin GOTO.

Sin embargo, no todos los lenguajes orientados a objetos prohíben la instrucción CALL (o su equivalente), permitiendo realizar programación híbrida, orientada a procedimientos y orientada a objetos a la vez.

2. Animación de Algoritmos.

La visualización de un algoritmo muestra una secuencia de gráficos que representan la ejecución del mismo, dicha representación puede ser controlada por el usuario de diferentes maneras, ya sea manipulando su velocidad o añadiendo o quitando elementos de la animación.

La idea de usar animaciones para ilustrar el comportamiento dinámico de algoritmos de computación aparece desde hace más de 15 años y más de 100 sistemas de animación de algoritmos han sido creados desde entonces, con la convicción de que las animaciones sirven a los estudiantes de manera efectiva como ayuda en el aprendizaje. Estas animaciones visualmente codifican y presentan cómo los datos son procesados en un programa en ejecución.

Para crear animaciones con un diseño más efectivo en la visualización de algoritmos es necesario crear un marco teórico basado en algunas consideraciones, las cuales exponen analogías interactivas y animaciones en hipermedios para mejorar el aprendizaje.

Hace más de 20 años, con la presentación de la película “Sorting Out Sorting” en la conferencia de Graficación por Computadora (ACM SIGGRAPH-81), nació la idea de usar gráficos y animaciones para ilustrar el comportamiento dinámico y la funcionalidad de algoritmos de computación. La animación de algoritmos prometía ser de gran ayuda en la instrucción de usuarios y muchos sistemas han sido construidos desde entonces, algunos de estos son BALSA (Brown, 1988) y TANGO (Stasko, 1990). Muchos de estos sistemas fueron desarrollados con la creencia de que la animación de algoritmos podría

servir como suplemento a exposiciones para que los estudiantes pudieran aprender más fácilmente sobre algoritmos, esta creencia tiene fuertes bases intuitivas, ya que los estudiantes tienen una dificultad relativa al momento de comprender nociones matemáticas abstractas, especialmente cuando estas incluyen dinámicas de cómo los algoritmos manipulan datos. Varios experimentos han sido conducidos para probar cómo dichas animaciones ayudan al mejoramiento del aprendizaje, como ejemplo Hundhausen (1997) presentó un compendio de resultados de 29 estudios empíricos relativos a la eficacia de la comprensión de la animación de los algoritmos y más generalmente del software de visualización. Gracias a la habilidad de combinar imágenes e hipertexto con que cuenta Internet. Se presentó como un medio ideal para la creación de estos sistemas de visualización de algoritmos, pero en un principio se presentaron problemas con animaciones creadas en lenguajes específicos y que dependerían de un *plug-in* especifico, este problema fue parcialmente resuelto con la aparición de la máquina virtual de Java.

La animación de algoritmos en la forma de *programa de visualización* se puede considerar como el uso de tecnología para el despliegue interactivo de gráficas y el arte del diseño digital, tipografía, animación, y cinematografía para realzar la presentación y la comprensión de programas de computación. Un programa de visualización se puede clasificar en dos formas, los que ilustran código o datos y los que muestran un despliegue dinámico o estático [1]. Los despliegues dinámicos, a su vez pueden ser pasivos, como en el caso de un video o interactivos como en el caso de los sistemas interactivos que corren en una estación de trabajo y que muestran el desarrollo de un algoritmo.

Los despliegues estáticos o código de programa, típicamente son diagramas de flujo, diagramas Nassi-Shneiderman, diagramas de alcance, y el mismo texto, cuando es

resaltado con tipografía o formato para realzar ideas. Este código de programa puede ser animado automáticamente remarcando las partes apropiadas del código que se ejecuta, uno de estos ejemplos es el sistema PECAN de Steven P. Reiss [3].

Los despliegues estáticos de datos de programa son más difíciles de automatizar que los despliegues estáticos de código. El problema principal está en que las estructuras de datos dadas pueden ser implementadas en muchas formas distintas. Muchas estructuras de datos y algoritmos son altamente especializados y requieren un tipo específico de despliegue para hacer comprensibles aspectos de sus propiedades, y desafortunadamente no hay aún principios para el diseño gráfico de estas estructuras de datos. Esto generó un modelo de proceso para la funcionalidad de los sistemas de visualización de algoritmos [3]. (figura 3).

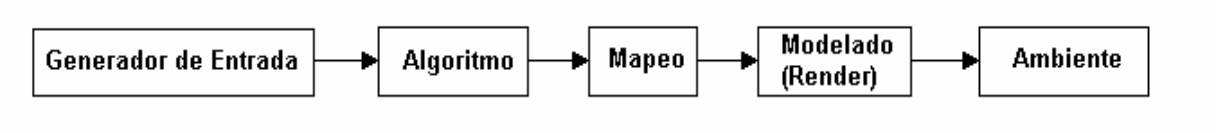

Figura 3.- Proceso de una visualización de algoritmo en Java.

Como se muestra en la figura 3, primeramente son generados los datos con los que operará el algoritmo, el algoritmo es ejecutado y pasa la información del estado actual a la etapa de mapeo. Esta información es llamada una pista del algoritmo y puede tomar la forma de descripciones textuales o el estado de estructuras de datos como arreglos, árboles, etc. El mapeo transforma la información del estado en gráficos primitivos, a su vez, estos gráficos son pasados a la etapa de render, la cual es el proceso de generación de un modelo a través de un programa que hace los cálculos relacionados con la

geometría, luz, sombra, textura de dicho modelo y donde éste es una descripción tridimensional de un objeto en un lenguaje definido o en una estructura de datos. Al final de esta secuencia la salida es direccionada al dispositivo de salida, el cual es manipulado por el usuario en el ambiente en el que se encuentra [4].

La creación de despliegues dinámicos de datos del programa tiene todos los problemas de la creación de los despliegues estáticos y más aún. En particular, los despliegues dinámicos deben decidir en qué momentos se deben actualizar los despliegues gráficos y de qué forma se deben hacer para que resulten efectivos. El despliegue de la animación de algoritmos es esencialmente dinámico mostrando operaciones fundamentales del programa, no solamente sus datos o el código. Las operaciones abarcan tanto el acceso a datos como el flujo de control de un programa.

2.1. Taxonomía de un despliegue de animación de algoritmos.

El despliegue de animación de algoritmos puede ser caracterizado a lo largo de tres dimensiones, contenido, transformación y persistencia como se muestra en la figura 4.

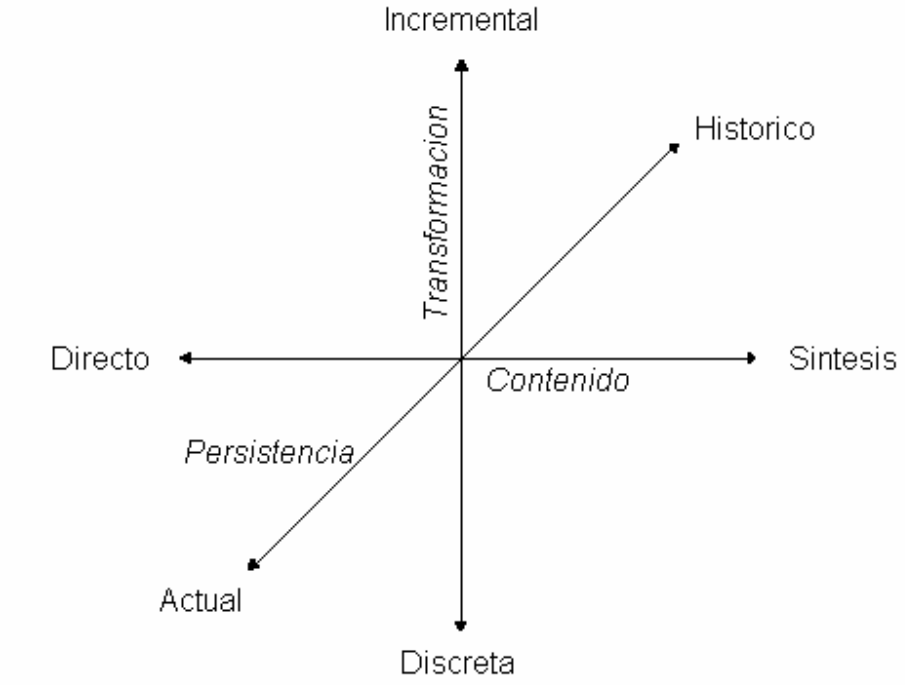

Figura 4.- Dimensiones en la animación de algoritmos

El *contenido* del despliegue va de la representación directa de los datos del programa o código a imágenes sintetizadas. La *persistencia* va del desplegado mostrando el estado actual de la información hasta un histórico de los cambios en la información. La *transformación* muestra los cambios en las imágenes de una forma discreta o bien, cambios incrementales y continuos.

Analizando más detalladamente cada uno de los ejes, es posible observar que en el caso del eje del contenido, los despliegues directos son imágenes que son isomórficos a una o más estructuras de datos en el programa. En algún momento, las estructuras de datos pueden ser construidas desde el despliegue gráfico y a su vez, el despliegue gráfico puede ser construido desde las estructuras de datos según sea necesario de acuerdo al programa. Para el despliegue sintetizado no es necesario hacer un mapeo de ninguna variable del programa, se pueden mostrar cambios en los datos del programa causados por operaciones del mismo, o simplemente pueden ser abstracciones de los datos. Varios despliegues están compuestos de componentes con un desplegado directo y sintético.

Un segundo criterio para clasificar los desplegados gráficos es la manera en la que se muestra la información *actual* o se despliega una breve historia de los cambios ocurridos en la ejecución. Inclusive las vistas de código forman una especie de histórico.

El tercer criterio para clasificar animaciones está basado en la naturaleza de las transiciones de los desplegados anteriores a los actualizados. Las transformaciones *incrementales* muestran transiciones del despliegue de una manera suave, mientras que las discretas simplemente borran los viejos valores y los remplazan con nuevos despliegues de los nuevos valores. Las transformaciones discretas son percibidas como incrementales cuando la diferencia entre las imágenes viejas y las nuevas son lo suficientemente pequeñas en relación a la complejidad de los datos. Las transformaciones discretas genuinas tienden a ser más útiles en conjuntos grandes de datos, mientras que las transformaciones incrementales son de mayor utilidad cuando los usuarios están examinando algoritmos en ejecución con un conjunto pequeño de datos, ya que disminuyen en gran medida la velocidad de la animación y contienen demasiado detalle a

un nivel muy bajo. Otro argumento para considerar las transformaciones incrementales es precisamente relativo al tiempo que se va a consumir en ellas. Por ejemplo, cambiar un solo apuntador en el nodo de un árbol puede causar que tenga que moverse una gran distancia.

2.2. Animaciones automáticas de algoritmos.

Los desplegados para la animación de algoritmos no pueden ser creados automáticamente porque son esencialmente monitores de las operaciones fundamentales de los algoritmos, y una operación de un algoritmo no puede ser deducida de un algoritmo arbitrario automáticamente [1], pero puede ser definida por una persona que conozca el funcionamiento de la operación realizada. Esto presenta sólo una parte del problema, y para comprender mejor esto es necesario considerar otros factores, como la naturaleza de la visualización del programa que puede ser creada automáticamente.

2.3. Desplegados de visualización automática de programas.

Los desplegados dinámicos requieren de dos tipos de información: La *entidad* a ser desplegada y una *delta* que describe los cambios en la entidad. Los desplegados estáticos solo necesitan información sobre la entidad.

Los desplegados dinámicos o estáticos, de código estático o en ejecución pueden ser creados automáticamente, porque el conjunto de entidades y deltas esté bien definido y pueden ser accesados directamente por una rutina encargada del despliegue. Las entidades son derivadas de código fuente y las deltas de los cambios en los contadores del programa. Ejemplos de entidades de código son los procedimientos, declaraciones, archivos, y bloques; ejemplos de las deltas pueden ser los avances en la marcación de las líneas de código, entrada y salida de los procedimientos, y entradas y salidas de bloques.

Las entidades en este caso serán las estructuras de datos que serán desplegadas, y las deltas pueden ser inferidas de los datos cada vez que estos son accesados o modificados. Automáticamente una rutina para el desplegado de una imagen estática de una estructura de datos podrá acceder al dato a través del entorno en el tiempo de ejecución. Es necesario decir que la representación de los datos en el programa y el tipo de despliegue deseado habrá que definirlo previamente.

2.4. Dificultades en el desplegado de animación de algoritmos.

Analizando más detalladamente algunos de los problemas de la creación automática de animaciones de algoritmos se obtienen algunos elementos a considerar.

- Captura de operaciones. Las operaciones algorítmicas no necesariamente corresponden a cada acceso o modificación a las estructuras de datos del algoritmo. En particular, acceder a una variable tiene diferentes significados en diferentes puntos del algoritmo, además, una cantidad arbitraria de accesos y modificaciones pueden resultar de una sola operación.
- Desempeño en tiempo real. En general, los datos accesados o modificados, difícilmente pueden ser identificados, y pueden resultar en un desempeño poco aceptable. Por ejemplo, saber qué nodo en un árbol será modificado y determinar sus padres e hijos, puede ser muy costoso en tiempo de cómputo y afectar mucho en el desempeño del programa.
- Despliegues informativos. Muchos despliegues necesitan un conocimiento detallado del comportamiento del algoritmo en tiempo de ejecución y el dato específico sobre el que el algoritmo está trabajando.

Otros puntos a considerar en el desplegado y la creación de programas de animación de algoritmos es la economía cognoscitiva, la cual es inducida por la configuración del sistema e implica dos factores; en el primero la economía cognoscitiva busca reducir la carga de información manejada por el usuario y las tareas que el usuario debe realizar y

que no pertenecen a la animación sino al funcionamiento y configuración del sistema. El segundo factor se refiere a la maximización de información correspondiente al problema o representación de éste que la visualización pretende comunicar al usuario. Esto puede oponerse a la disminución de tareas que se mencionó anteriormente, sin embargo, lo ideal es buscar un equilibrio entre la información estrictamente necesaria para que el usuario comprenda bien el funcionamiento del algoritmo sin tener que mostrarle una sobrecarga de controles o datos que sólo agrandarían la carga mental del usuario, haciéndole más difícil la comprensión del algoritmo.

Para mejorar esta economía cognoscitiva se siguen tres pasos, el primero de estos se concentra en la eliminación de representaciones indirectas, esto es, desplegados que a pesar de mostrar información necesaria en algún punto, no es indispensable que sea mostrada y sólo distraerá al usuario. El segundo paso se refiere a la automatización de la representación de los datos y desplegados de la animación, con esto el programa deberá encargarse de calcular y optimizar el despliegue de los datos y estructuras necesarias para la animación del algoritmo de la forma más eficiente y sin que el usuario tenga que llevar a cabo tareas extras. El tercer paso consiste en enriquecer las animaciones con representación explícita de la visualización de la sintaxis, así se puede hacer una relación entre las entidades computacionales y los elementos dinámicamente gráficos. Esta sintaxis puede ser desplegada a través de leyendas, que pueden ser interactivas y que permiten al usuario entender de forma continua las vistas de la animación, o bien, se puede hacer uso de mensajes informativos o de marcación de texto según sea conveniente.

Tratando más a fondo estos pasos se obtienen algunas características importantes a considerar.

Así, para lograr una buena presentación automática es necesario tomar en cuenta otros tres factores:

- *Expresividad:* Se deben mostrar todas las entidades y relaciones en la escena y nada más, evitando colocar controles o leyendas innecesarias para comprender el algoritmo.

- *Clasificación visual:* Las entidades deben estar bien distinguidas y ordenadas, ya sea por medio de áreas, formas o colores que simplifiquen la carga perceptiva.

- *Continuidad:* Asegurar que la representación de las escenas de la animación sea consecutivamente consistente con las anteriores. Para lograr esto es necesario que la representación de los datos, mensajes y avisos se mantenga de la misma forma en que fue representada por primera vez, a menos que se quiera resaltar alguna transformación o información especial relativa al algoritmo.

2.5. Sistemas existentes de animación de algoritmos.

Actualmente existen diversos sistemas encargados de la animación de algoritmos y desarrollados en distintos lenguajes, inclusive hay grupos de desarrollo para la creación de estos sistemas, tomando diferentes acercamientos para la solución de este problema. A continuación se presentan algunas características de los más destacados.

2.5.1. BALSA (Brown ALgorithm Simulator and Animador) y BALSA II.

Este es un entorno de los más conocidos, empleados y difundidos para la animación de algoritmos dentro del mundo de la enseñanza e investigación. Desde su primera versión lanzada en el 84 y la segunda del 88 este sistema se colocó entre los más usados debido a las facilidades de uso de scripts de animaciones, uso de colores en las animaciones así como sonido y entradas provistas por el usuario en tiempo de ejecución. Este sistema tiene un conjunto de ventanas encargadas de mostrar las animaciones según el algoritmo que se pretenda estudiar, además de permitir animaciones simultáneas. Este sistema resulta ser una piedra angular en el desarrollo de sistemas y aplicaciones enfocadas a la animación de algoritmos y el apoyo a la educación. En la figura 5 y 6 se puede observar BALSA y BALSA II, cada uno mostrando la ejecución de dos algoritmos en dos grafos.

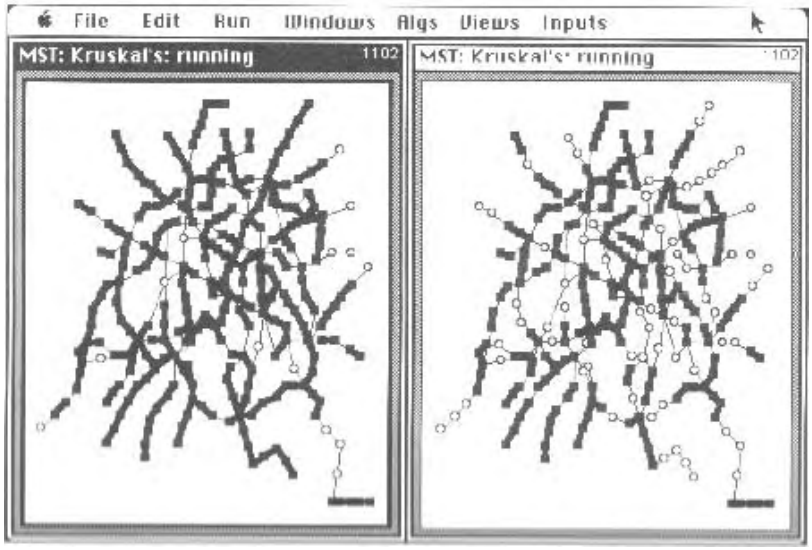

Figura 5. Representación del algoritmo de Kruskat para encontrar un árbol expandido mínimo en una gráfica conexa corriendo en dos ventanas simultáneamente.

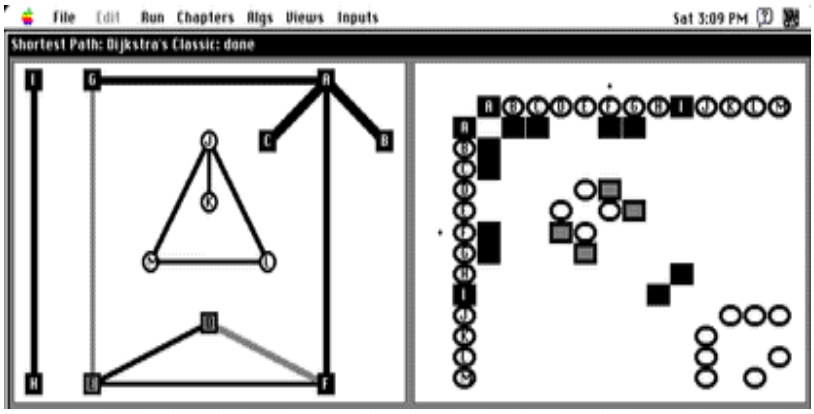

Figura 6.- Animación del algoritmo de Dijkstra para encontrar la ruta mas corta.

2.5.2. TANGO (Transition-based Animation GeneratiOn) y XTANGO.

Es un sistema de animación de algoritmos formado por un conjunto de bibliotecas en C, el cual con su versión para interfaz gráfica (XTANGO), se ejecuta sobre el sistema X Windows. Este sistema está enfocado a la comprensión de procedimientos más que en la parte declarativa. Debido a esto el sistema solo muestra la animación de los datos y sus cambios a través de la ejecución de algún algoritmo, este, al igual como su contemporáneo BALSA, puede mostrar varias ventanas con animaciones simultáneas y el uso de colores en los desplegados. Para la creación de animaciones se requieren conocimientos básicos de C. En la figura 7 y 8 se presentan las interfaces te TANGO Y XTANGO respectivamente.

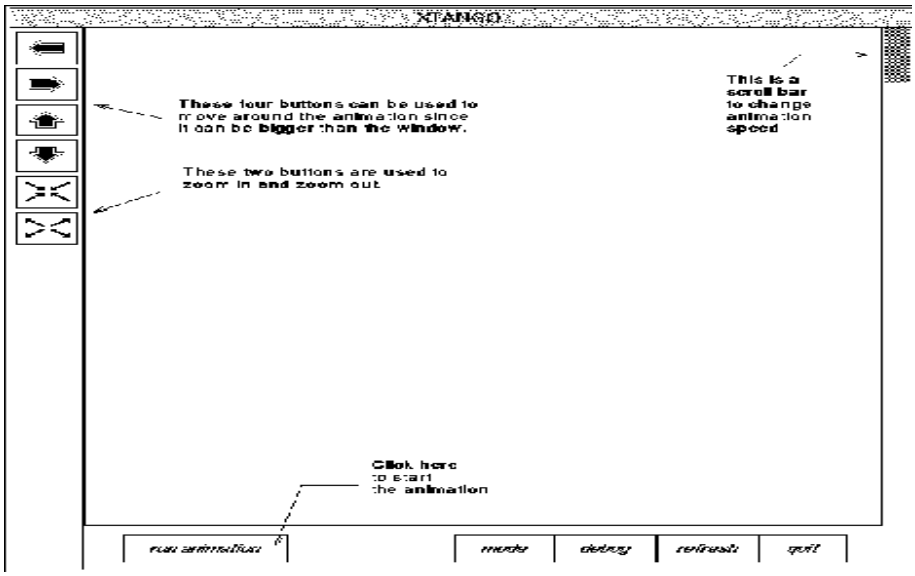

Figura 7.- Interfaz del programa de animación XTango.

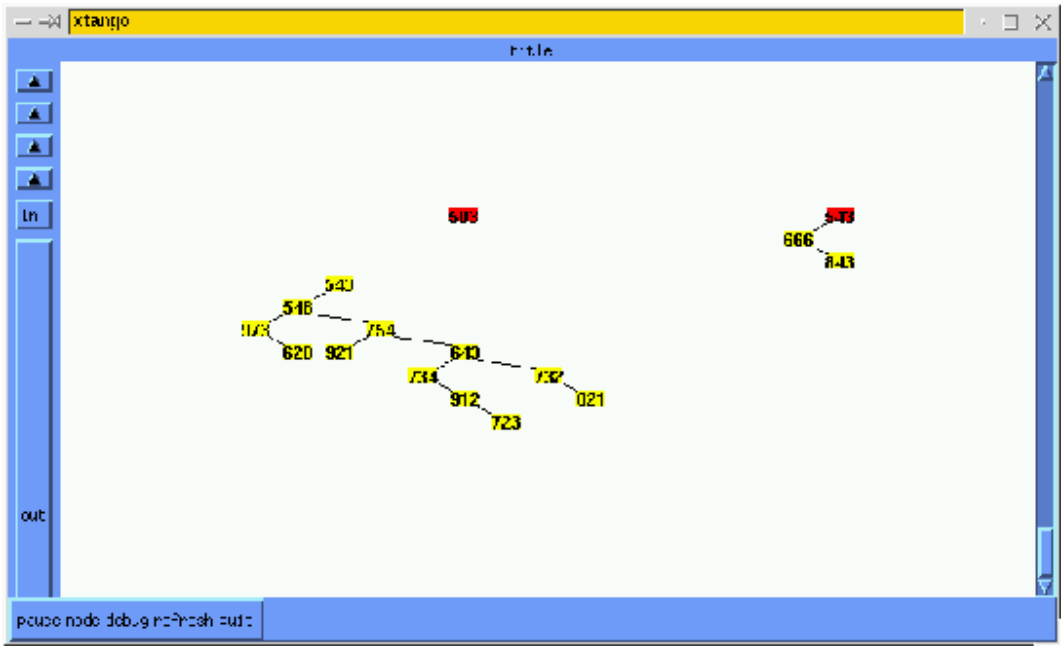

Figura 8.- XTango mostrando la animación de un ordenamiento en un árbol.

2.5.3. JHAVÉ (Java-Hosted Algorithm Visualization Environment).

Es una aplicación desarrollada en Java que ayuda a la representación visual de datos y soporta la visualización de algoritmos a través de scripts. La aplicación funciona vía web, y se conecta con un servidor que contiene animaciones de algoritmos previamente definidas y organizadas por categoría. Además cuenta con una sección de preguntas y respuestas o de “trivia” que logran una mayor interacción con el usuario. El sistema contiene varias ventanas donde se despliega la información de código del algoritmo a estudiar, así como la representación gráfica de éste al ser ejecutado más el apartado donde se realiza la pregunta y el espacio para colocar la respuesta. También contiene una sección de controles donde se puede manipular la velocidad de la animación o los pasos de la misma y la activación o desactivación de la trivia.

En la figura 9 se presenta la interfaz, con la bienvenida y las opciones para realizar la conexión con el servidor, mientras que en la figura 10 se muestra la interfaz ejecutando un algoritmo, dando un resumen de este y preguntando al usuario la respuesta para el paso siguiente.

Figura 9.- Interfaz de inicio y conexión con el servidor de JHAVÉ.

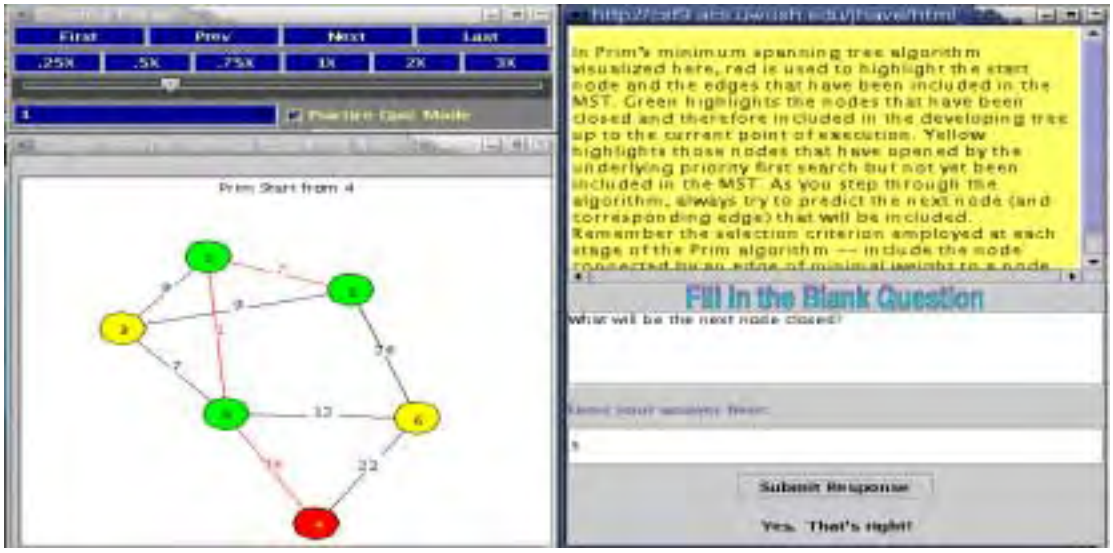

Figura 10.- JHAVÉ mostrando el algoritmo de Prim para encontrar un árbol mínimo en una gráfica conexa no dirigido, junto con la ventana de trivia al lado derecho.

2.5.4. SAMBA y JSAMBA.

Es un intérprete interactivo de animaciones además de un generador de animaciones. Este sistema proporciona la flexibilidad de crear una animación conociendo los comandos y la sintaxis del intérprete. Los desarrolladores crearon una interfaz gráfica programada en Java y accesible desde Internet. En sus ventanas se muestra la información relativa al pseudocódigo necesario para crear la animación y varias representaciones gráficas de los datos manipulados en la animación, ya sea en forma de árboles o en columnas. Este intérprete también cuenta con controles de velocidad de la animación. En la figura 11 se observa la interfaz ejecutando uno de sus scripts de ejemplo.

Figura 11.- Applet que interpreta un script de ejemplo mostrando varias ventanas con animaciones sencillas y una barra de controles.

2.5.5. Alice.

Es un entorno de programación diseñado específicamente para facilitar el aprendizaje de la programación orientada a objetos. Inicialmente creado como un proyecto de la Universidad de Carnegie Mellon y posteriormente apoyada por la compañía Stage3 Research Group subsidiaria de la misma universidad a través de Entertainment Technology Center and Human Computer Interaction Institute. La interfaz cuenta con muchas opciones y es más parecida a un entorno de desarrollo que a un ambiente de animación de algoritmos, esta complejidad puede afectar un poco a la curva de aprendizaje del estudiante, por lo cual decidieron añadir varios tutoriales que intentan explicar cómo crear una animación, junto con objetos y comportamientos (métodos) para dicha animación. La interfaz está dividida en unos controles básicos de animación, una ventana donde se muestran el estado de la animación, es decir, el escenario y los objetos que la componen, en el recuadro adyacente se despliegan los eventos relacionados a los objetos y en los momentos de la animación en los que deben ocurrir. Debajo de estos aparece un espacio que funciona como editor de métodos, brinda la facilidad de agregar parámetros y variables a los métodos, así como estructuras de control. Al lado izquierdo está la ventana referente a propiedades, las cuales son las características o rasgos de los componentes de la animación, por ejemplo, el color del escenario, la distancia, etc. Los métodos que existen, su edición y creación. Y por ultimo, las funciones, que son subrutinas que ayudan a los métodos a realizar los eventos. Arriba de esta ventana se encuentra los elementos escenograficos de la animación, es decir, las cámaras que intervendrán, las luces y el tipo de suelo. Todos estos elementos proveen un poderoso y

complejo ambiente de aprendizaje de la programación orientada a objetos, con un enriquecido contexto grafico con posibilidad de ampliación a través de la creación o descarga de nuevos gráficos.

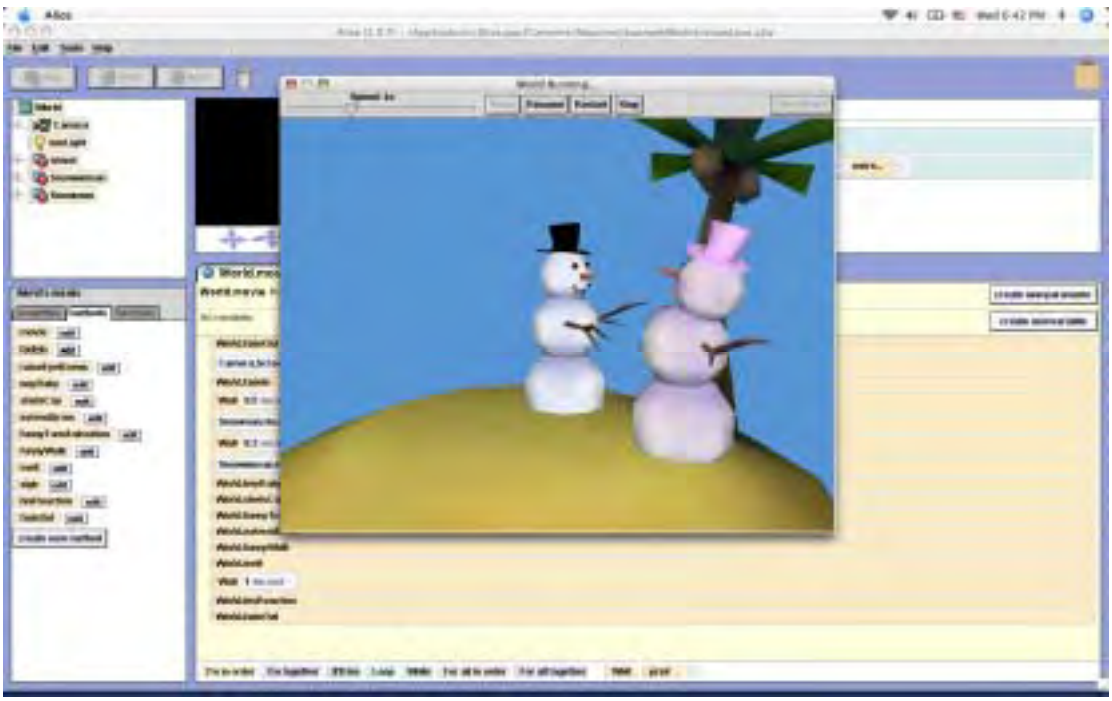

Figura 12.- Ejemplo de la interfaz del sistema Alice.

En la figura 12 se puede observar una de las ventanas del sistema mostrando una animación. Al fondo, en segundo plano, se observa la interfaz con los diferentes apartados de creación de eventos, métodos, edición e iluminación.

3. Sistema de animación de algoritmos en la programación orientada a objetos.

Uno de los objetivos de esta tesis es el desarrollar un sistema accesible a través de Internet y sin necesidad de complicadas instalaciones que permitiese comprender de manera más sencilla e interactiva los conceptos básicos o fundamentales de la programación orientada a objetos, mostrándole al usuario una representación grafica de lo que ocurre al ejecutar un código de un programa, escrito siguiendo el paradigma de la programación orientada a objetos. Esto se logró implementado con Java applets un sistema que cuenta con un conjunto de animaciones precargadas y controles de velocidad y paso a paso de la animación escogida, así como de reinicio. Como se puede ver en la figura 13, en una de sus ventanas, se despliega el código del programa del cual se verá su ejecución, mientras que en la ventana que se encuentra al lado izquierdo, se muestra la animación de la ejecución en sincronía con el código.

El sistema crea una representación de los objetos en una aproximación semejante a UML (Unified Modeling Language) presentando la información relativa a los métodos y al estado de los atributos de cada objeto creado. Esta representación es mostrada en el recuadro principal de la interfaz (mitad izquierda).

Esta presentación se puede observar durante la secuencia de ejecución del programa el estado de cada una de las representaciones de los objetos involucradas en cada uno de los diferentes ejemplos junto con sus posibles relaciones entre objetos.

Para mayor comodidad del usuario, la velocidad de las secuencias de ejecución puede ser modificada deslizando el indicador de la barra de velocidad de la etiqueta “lento” a la

etiqueta “rápido”, según sea necesario para incrementar o disminuir la velocidad de las animaciones. También puede revisarse la animación paso a paso para conseguir una mayor comprensión de lo que ocurre en ambas ventanas.

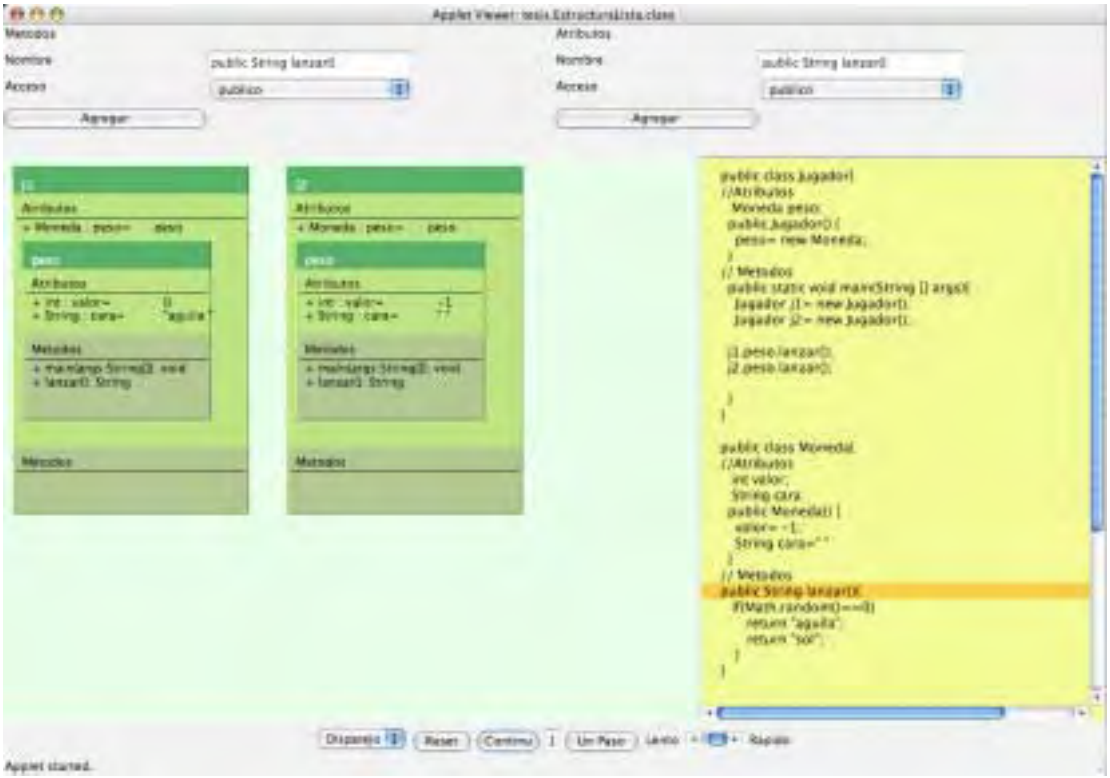

Figura 13.- Ejemplo de Interfaz del SAAPOO.

En la figura 13 se presenta una animación de un programa orientado a objetos relacionado a un juego de volados. En las secciones se muestra la animación referente a la instanciación y estado de los objetos, mientras que en el apartado derecho se sigue la secuencia de las líneas del código ejecutado y en la parte baja los controles de velocidad.

Para lograr ejemplificar mejor el funcionamiento del programa, seguiremos la ejecución de uno de las animaciones existentes en el sistema, para esto, primeramente se seleccionara un ejemplo de la lista disponible.

Figura 14.- Selección de un ejemplo disponible.

Una vez que el ejemplo ha sido escogido, el código correspondiente se presentará en el recuadro derecho (Figura 15).

```
public class Moneda{
//Atributos
  int valor;
  String cara
 public Moneda() {
   valor= -1;
   String cara=" "
 }
// Metodos
public String lanzar(){
  if(Math.random()==0)
     return "aguila";
     return "sol";}
 public static void main(String [] args){
    Moneda peso= new Moneda();
   peso.lanzar();
  }
 }
}
```

Figura 15.- Recuadro derecho de la interfaz conteniendo el código del ejemplo “Volados”. La cual contiene la clase Moneda y un método lanzar, el cual determina la cara de la moneda lanzada.

Para comenzar la animación se presiona el botón “iniciar”, con esto se presentará la creación de los objetos en el recuadro derecho según vayan siendo instanciados de acuerdo al código del recuadro izquierdo.

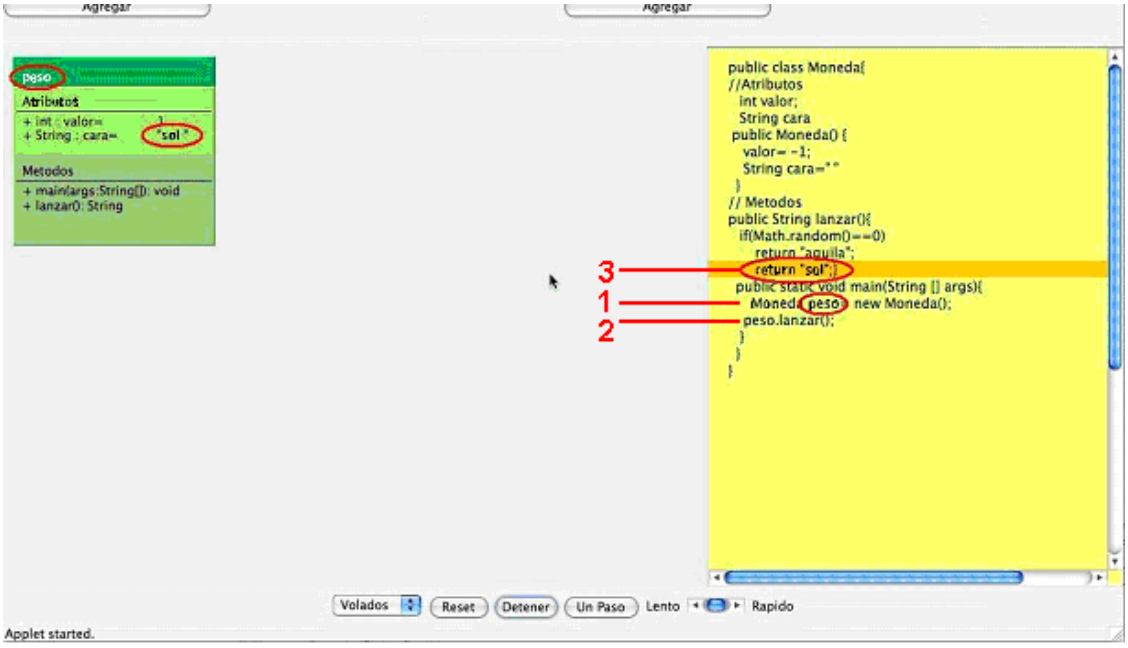

Figura 16.- Animación de la ejecución del programa “Volados”.

Como se observa en la figura 16, en el paso marcado con el número 1, primero se crea el objeto llamado peso de la clase Moneda, la cual en un inicio tiene todos sus atributos inicializados según su constructor, posteriormente, en el paso 2 se ejecuta el método lanzar, el cual, en el paso final 3, regresa la cadena con el tipo de cara que resultó del lanzamiento aleatorio de la moneda, en este caso resultando el valor “sol”. Se puede observar como la representación UML muestra los nombres de los objetos de la clase correspondiente, así como sus métodos y atributos, junto con los cambios que sufren éstos al ser afectados a través de métodos u otras clases.

Con estas animaciones es posible seguir el momento exacto de creación o “instanciación” de un objeto, la relación que puede compartir con otros objetos, y el estado de sus atributos durante la ejecución de un programa. Esto facilita la comprensión de la ejecución de un programa, permitiendo seguir línea a línea su código, entendiendo qué pasa en cada instrucción y cómo afectan estas a los objetos relacionados, y qué interacción existe entre ellas. En el caso de que los objetos tengan relaciones menos triviales, permite comprender cómo las clases herederas comparten comportamiento y propiedades con sus ancestros mediante líneas que representan su relación, como se muestra en la figura 17.

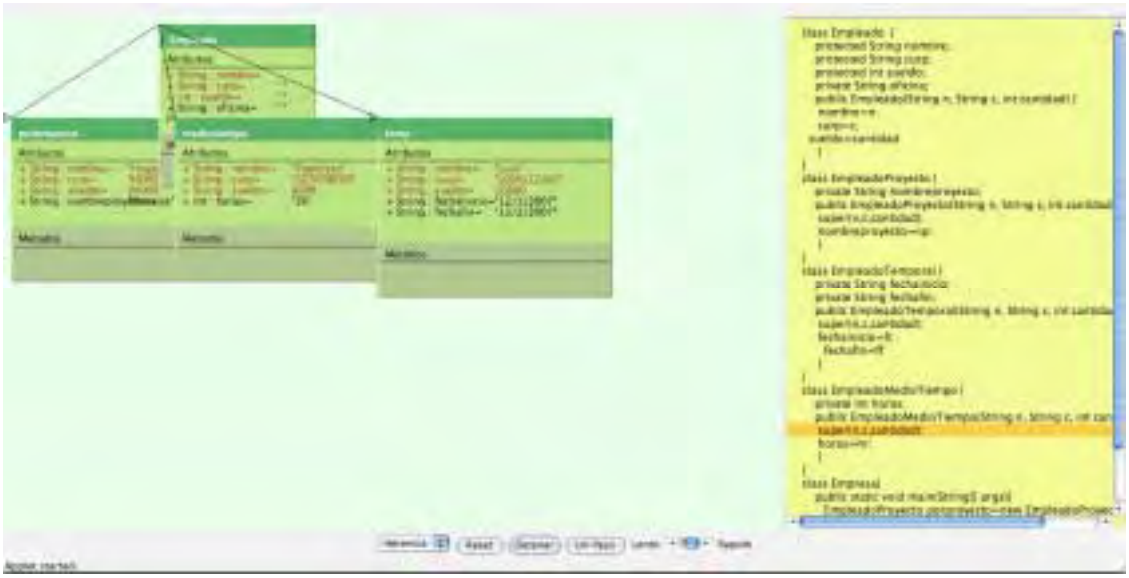

Figura 17.- Imagen donde se muestra la herencia entre tres clases (posproyecto, mediotiempo y temp) y la clase padre (Empleado).

3.1. Arquitectura y diseño.

La arquitectura del sistema de animación de algoritmos para programación orientada a objetos está definida en base al patrón por capas y de la cual se toman las siguientes capas:

- Capa de Interfaz de Usuario:
    - Esta capa comprende el paquete denominado Interfaz, el cual contempla la interfaz grafica en la que se mostraran todas las ventanas donde aparecerán los desplegados gráficos, tanto animaciones, como códigos relacionados a éstas, así como los controles para su funcionamiento.

- Capa de Control:
    - En esta capa es donde se comprenden tres paquetes: Sincronía, Graficación y Objetos. Entre estos tres paquetes se divide la responsabilidad de la creación, sincronía, control y dibujo de las representaciones de cada desplegado grafico.

En este modelo no se integró una capa de almacenamiento de datos, ya que no se necesita de una base de datos, y si bien a futuro la especificación extra de datos para el despliegue puede ser necesaria, su implementación puede ser más viable a través de archivos de propiedades o bien con archivos XML.

A continuación se muestran los diagramas de casos de uso, paquetes, clases y dependencias que conforman el sistema.

Diagrama de Paquetes.

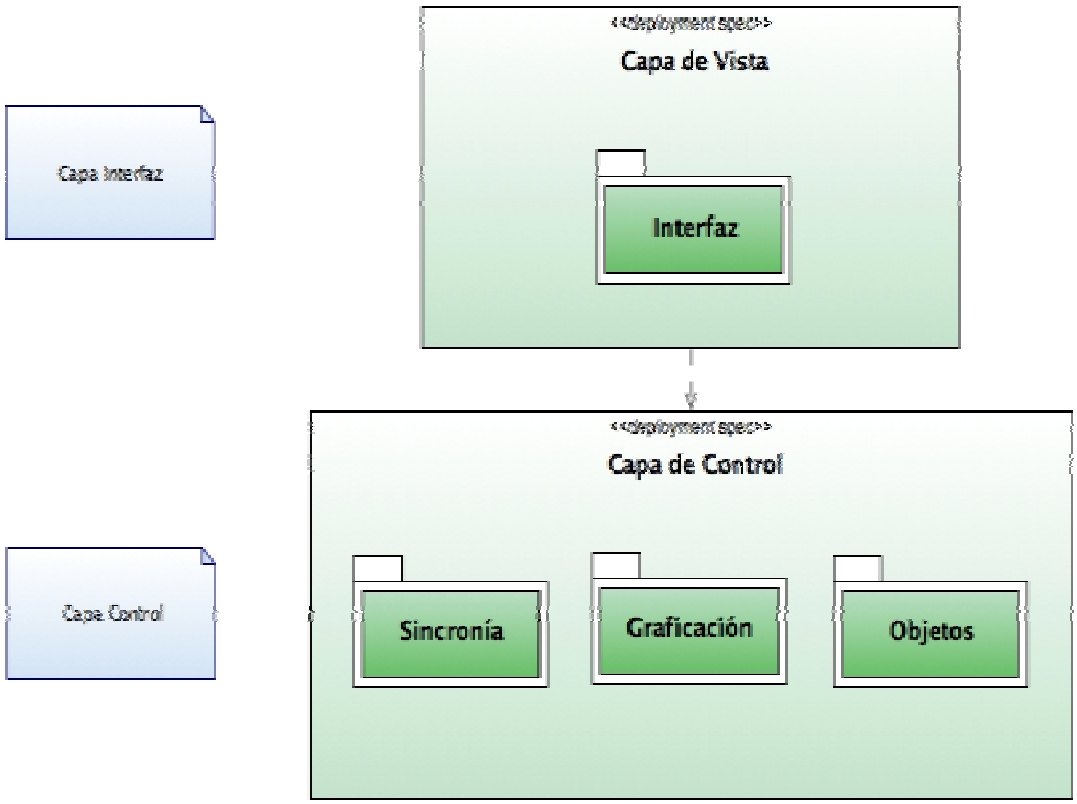

Figura 18.- Diagrama mostrando la arquitectura a seguir en el desarrollo del sistema.

Diagrama General de Casos de Uso.

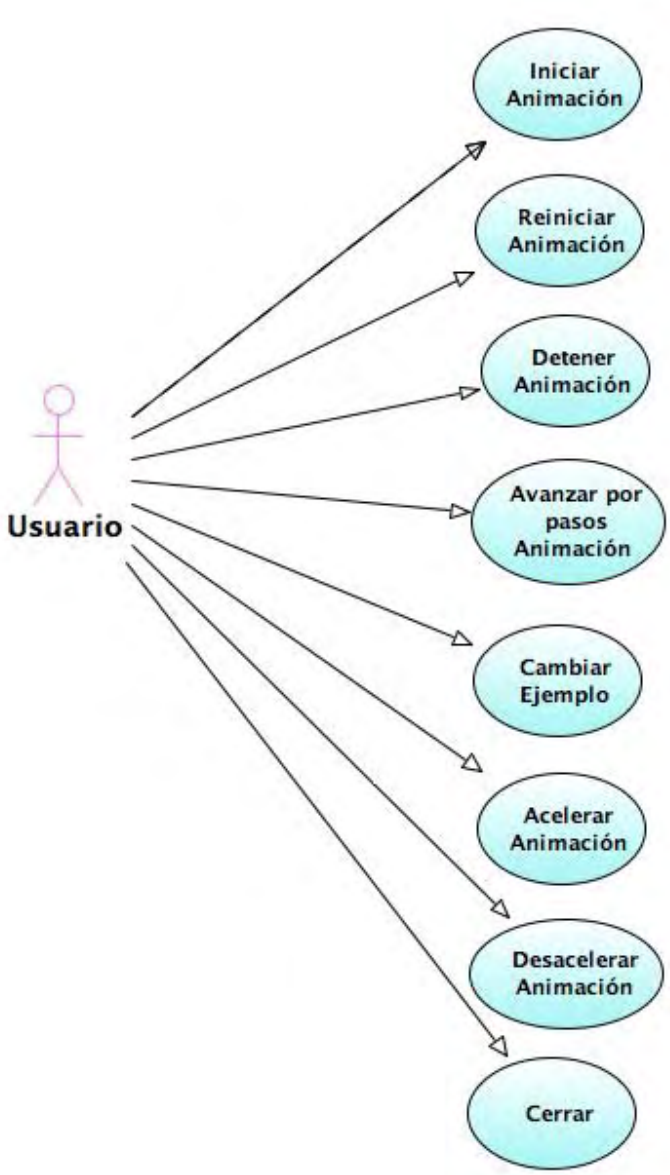

Figura 19.- Diagrama general de casos de uso, en el que se muestran las diversas acciones que puede realizar un usuario al interactuar con el sistema.

Diagramas de Clase.

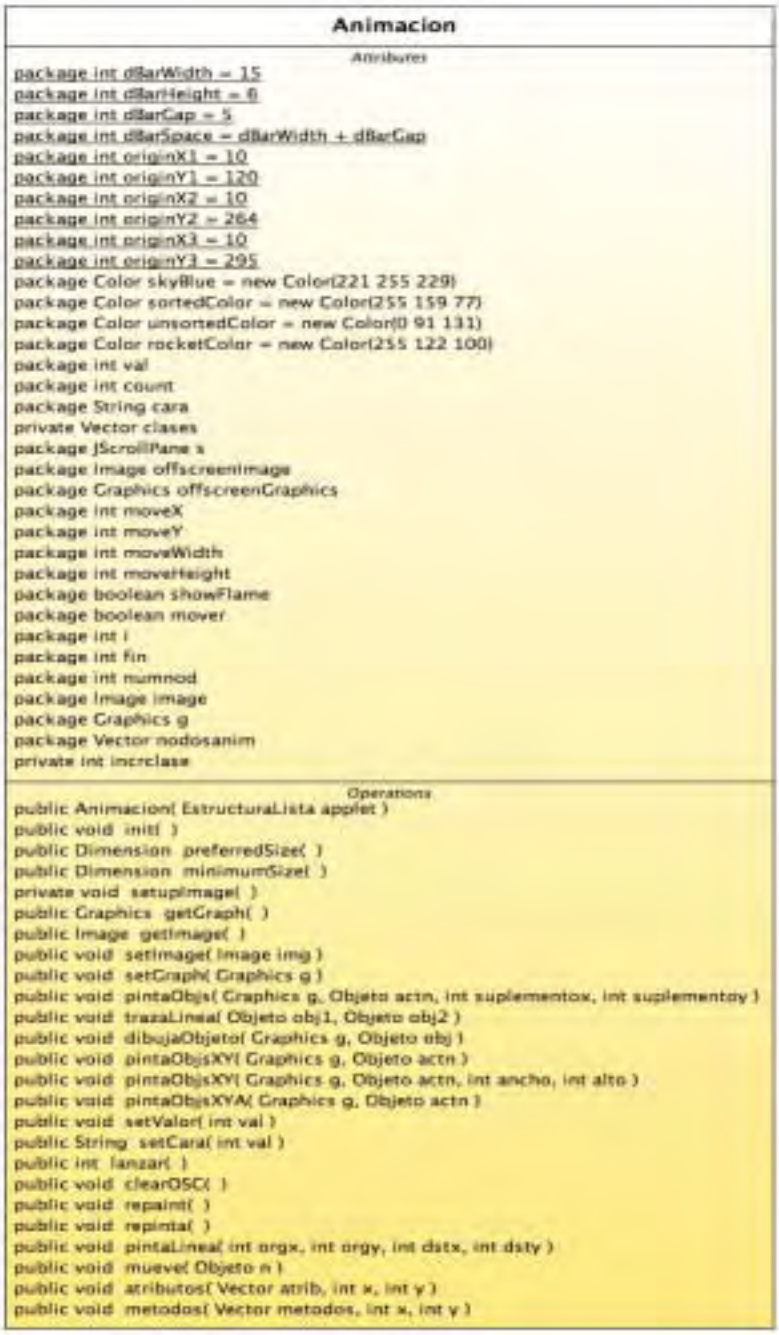

Figura 20.- Clase Animación.

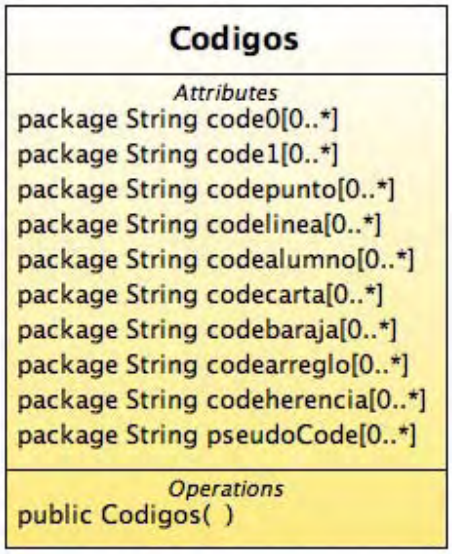

Figura 21.- Clase Codigos.

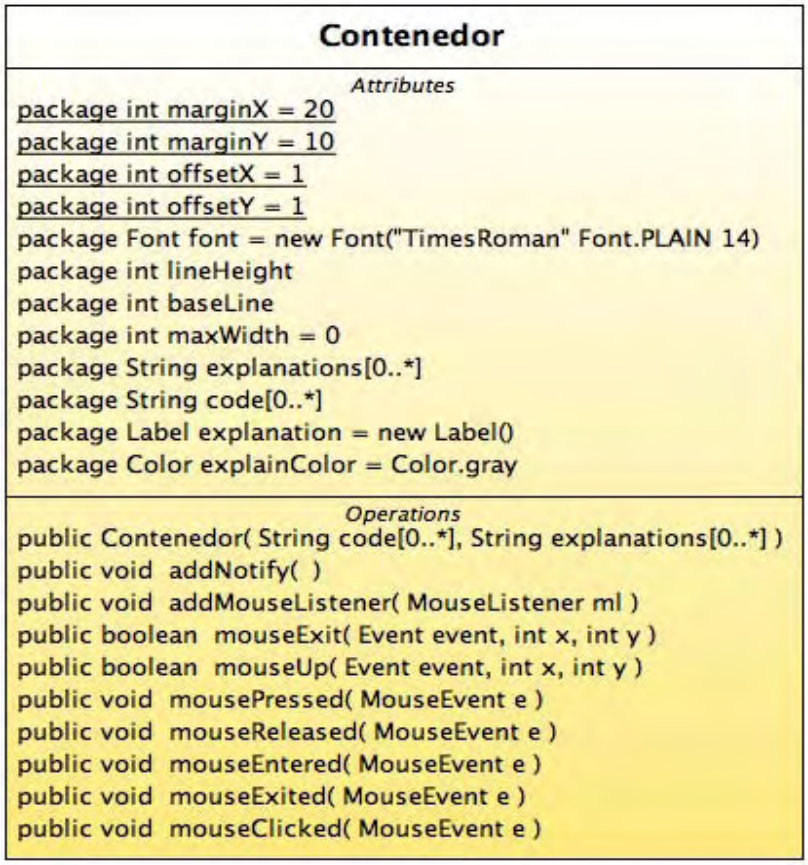

Figura 22.- Clase Contenedor.

Figura 23.- Clase EstructuraLista.

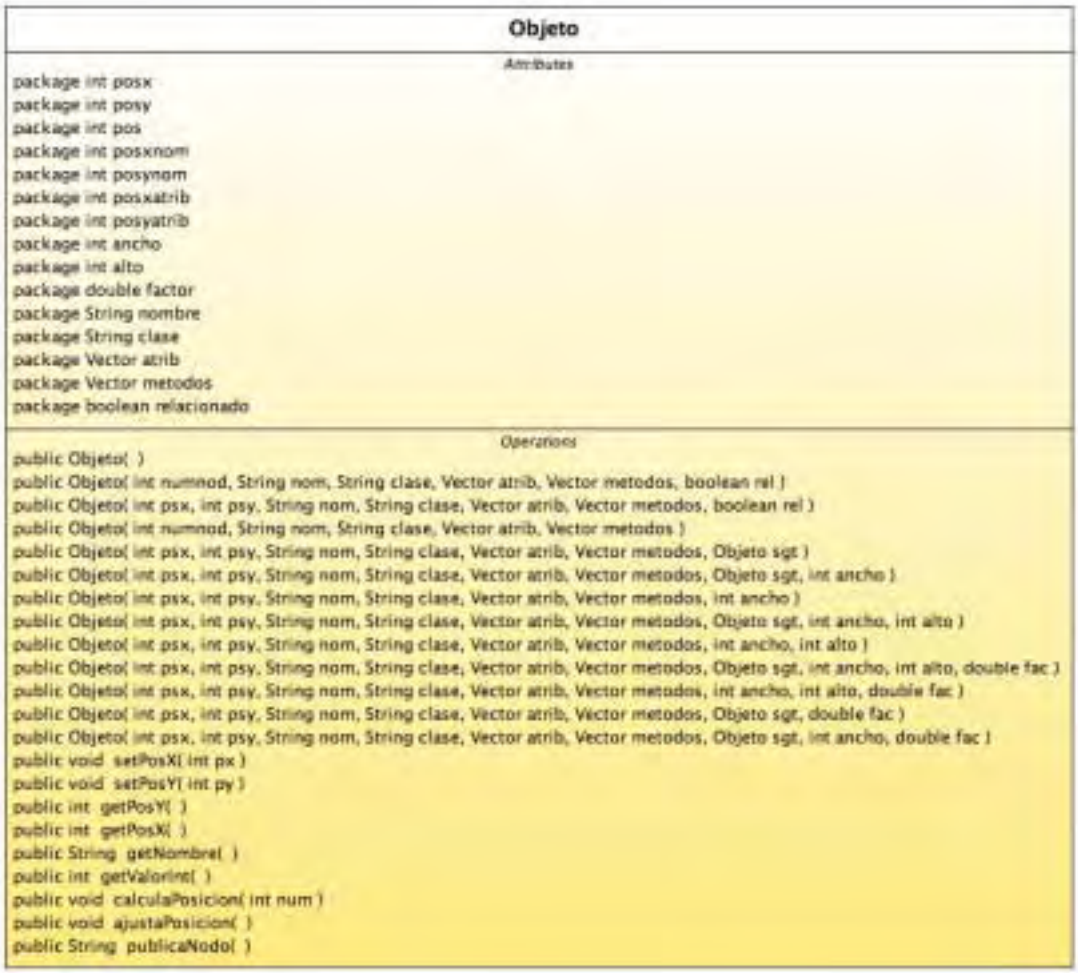

Figura 24.- Clase Objeto.

Figura 25.- Clase PanelMetodos.

Figura 26.- Clase PanelCodigo.

Diagrama de Dependencias.

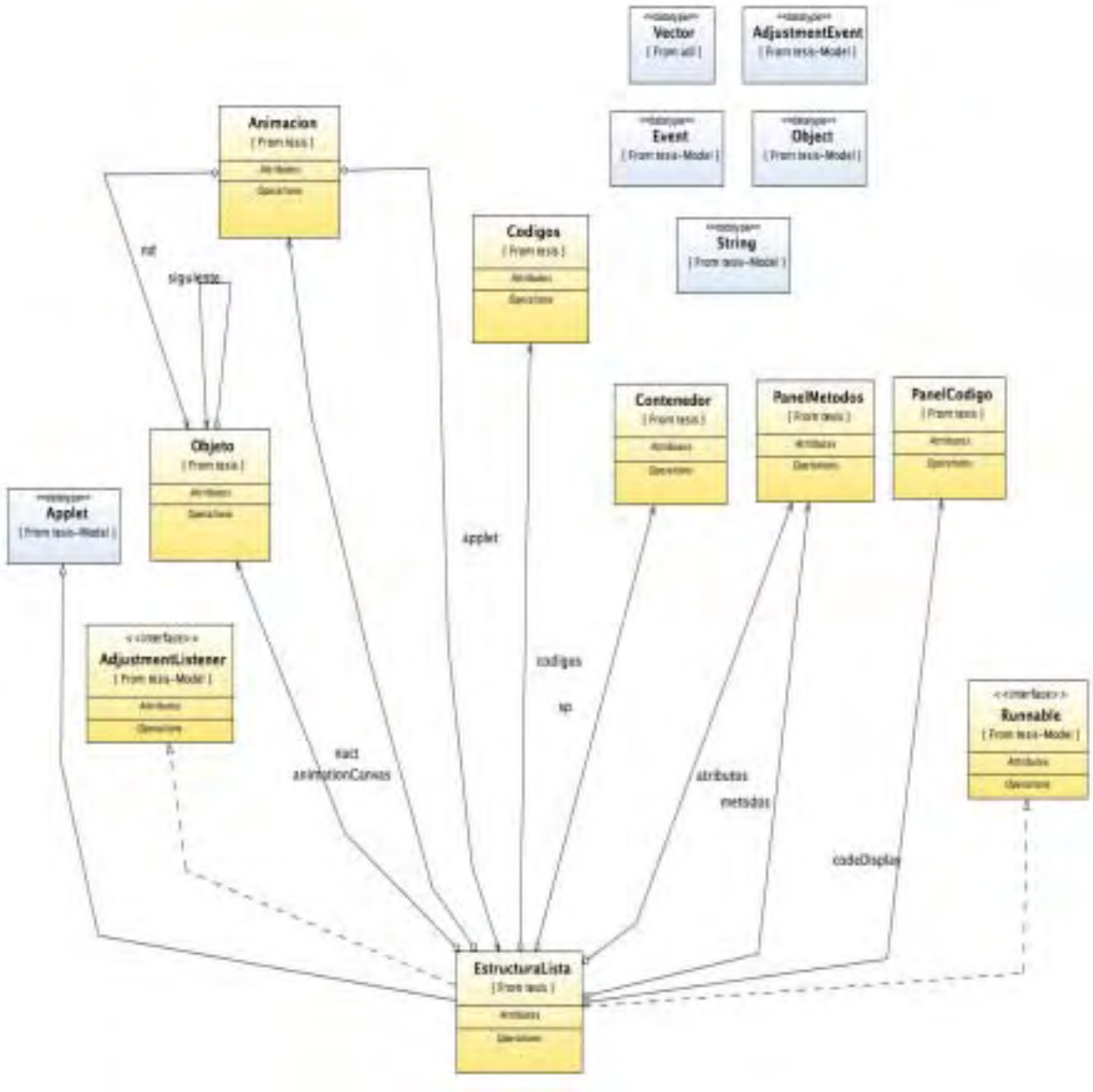

Figura 27.- Diagrama de dependencias entre clases.

3.2. Comparativas.

Como vimos anteriormente con la breve descripción de los sistemas, todos cuentan con una interfaz que despliega las animaciones, en el caso de BALSA y JHAVÉ pueden hacer la interpretación de scripts, lo cual brinda al usuario la posibilidad de crear nuevas animaciones, siempre y cuando se conozca la sintaxis para dichos scripts. BALSA y XTango tienen la capacidad de tener varias animaciones en diferentes ventanas. En el caso de JSamba, JHAVÉ y el sistema objeto de esta tesis, todos son creados con Java, los dos últimos son accesibles a través de Internet y el primero y el último utilizan el componente applet para su despliegue. JHAVÉ realiza preguntas acerca del proceso de ejecución para enfatizar los pasos de aprendizaje. Alice es el único sistema enfocado a la orientación de objetos y quizá uno de los más poderosos y desarrollados al paso del tiempo, sin embargo por su complejidad y énfasis en el despliegue gráfico puede distraer al usuario haciendo que se concentre más en los gráficos que en aprender el paradigma de programación orientada a objetos. El sistema de esta tesis es capaz de seguir el código seleccionado en la secuencia de ejecución en sincronía con la animación de los objetos que se van creando. Con JSamba se pueden crear nuevas animaciones siguiendo la sintaxis específica y agregándola a su intérprete en línea cargando el archivo creado. La mayoría de los sistemas tienen una representación libre de los componentes representados en sus despliegues gráficos, mientras que, como veremos a continuación, en nuestro sistema se tiene la representación semejante al UML, esta representación tiene la ventaja de tener la información en cada uno de los objetos de la animación y no diseminada alrededor de la interfaz.

Conclusiones.

El sistema de animación de algoritmos para programación orientada a objetos, como ya se ha mencionado, está desarrollado utilizado Java applets, lo cual le da la gran ventaja de poder ser utilizado desde cualquier computadora que cuente con Java instalado, sin importar el sistema operativo. Y más aun, el sistema puede ser fácilmente trasladado a una aplicación de escritorio, si en algún momento esto resultará necesario. La ventaja de que haya sido desarrollado como Java applets en vez de como una aplicación de escritorio, es que no es necesario distribuir el código a cada computadora donde se vaya a correr, solamente es necesario contar con un navegador, y que éste disponga de una maquina virtual de Java.

Además, la representación escogida brinda un doble beneficio al usuario, ya que le puede servir como una breve introducción al UML mientras aprende los conceptos de la programación orientada a objetos.

Quizás el sistema cuenta con menos poder de graficación que sus antecesores, ya que su principal función no es el despliegue grafico ni proporcionar varias animaciones simultaneas, aunque en algún momento pueda hacerse posible, ya que el desarrollo provee la flexibilidad para extender sus capacidades. Sin embargo, esta “limitante” es en realidad uno de sus puntos fuertes, ya que en vez de sobrecargar las habilidades perceptivas del usuario, el sistema le mostrará información relativa al código en secuencia con la única animación, lo cual muy pocos de los sistemas descritos anteriormente son capaces de hacer.

Una limitante real del programa es la definición estática del pseudocódigo de los ejemplos, esto en parte, es porque el sistema se enfoca en los ejemplos representativos de la programación orientada a objetos, la agregación de más pseudocódigo, podría desembocar en la sobrecarga cognoscitiva del usuario.

Los controles del sistema permiten tener un dominio más preciso de la secuencia de la animación, permitiendo al usuario parar o ir más lento en las partes del pseudocódigo que le parezcan más difíciles de comprender, o para recalcar alguna idea en el proceso de aprendizaje.

A la vez, las ventanas y los fondos de colores en ellas, le permiten al usuario filtrar y enfocar su atención en zonas precisas, dándole orden a las secciones de pseudocódigo, animación de los objetos y controles del sistema.

Por lo tanto, el sistema resulta ser una potente herramienta en el apoyo de la ejemplificación del paradigma de programación orientado a objetos, habiendo muy pocos sistemas que se enfoquen a este modelo y con los datos necesarios y más indispensables para su comprensión. Además de contar con una sincronía en tiempo real del pseudocódigo ejecutado y la animación de los objetos creados, la cual la mayoría de los sistemas carecen, sin contar que es el único enfocado exclusivamente a la programación orientada a objetos, sin embargo, hay que decir que el programa es susceptible de ser modificado para cambiar el paradigma de programación al que se quiera explicar.

Al momento del diseño del programa no se contemplaron las dificultades que implicaría la sincronía de los elementos de la animación, es decir, el marcado adecuado de la línea de código ejecutándose y la animación correspondiente. Esto tuvo que ser solucionado

empleando hilos de ejecución, los cuales tenían que seguir el estado de las representaciones de los objetos junto con la línea ejecutada, a lo cual el esquema de trazado o dibujo planteado por los applets complico el desarrollo. De ahí que se tuviera que construir una clase especifica “Objeto”, conteniendo la información de la representación gráfica, y una clase “Animacion” que controlaba el dibujo de dicho objeto, a través del hilo de ejecución que revisaba la línea de código a ejecutar, según la información enviada por la clase “PanelCodigo”. Todas estas clases están conjuntadas en la clase “EstructuraLista”, la cual proporciona el desplegado de paneles necesario para presentar cada una de las partes.

A futuro.

El sistema puede ser extendido y mejorado con la agregación de un intérprete que sea capaz de leer archivos con animaciones y códigos, definiendo un lenguaje lo más simple y comprensible para el usuario, para que su curva de aprendizaje sea la menor, y no tenga que aprender todo un nuevo y complejo lenguaje para lograr entender el paradigma de la orientación a objetos. También el sistema sería susceptible a la creación de un módulo capaz de agregar objetos y métodos al pseudocódigo y a su vez a las animaciones para lograr mayor interacción con el usuario. En este mismo tema, la adición de un módulo de preguntas o respuestas, o trivias relacionadas a lo que espera el usuario de la secuencia de ejecución. Esto ayudaría a agregarle un reto al usuario y permitirle reafirmar o poner a prueba los conceptos ya antes aprendidos.

El despliegue de gráficas también podría ser mejorado, quizá utilizando gráficas en 3D, tomando en cuenta que esto implicaría un replanteamiento en el diseño de la representación de objetos para sacarle provecho al espacio disponible, desde luego, tomando siempre en cuenta de no caer en excesos de despliegues que puedan distraer la atención del usuario y con la disposición mínima de controles que sirvan para crear y manipular dichos despliegues gráficos.

Referencias.

[1] Meyers, Brad. A. Visual Programming, Programming by Example, and Program visualization: A Taxonomy, In Proc. ACM SIGCHI Conf. on Human Factors in Computing Systems.

[2] Hamilton-Taylor, A. and Kraemer, E. Suporting algorithm and data structure discussion. In Proceedings of the Thirty-third SGCSE Technical Symposium on Computer Science Education (SIGCSE-02), 2002

[3] Steven P. Reiss. Displaying Program and Data Structures. Technical Report CS-86-19, Brown University, Providence, ir, Abril 1986

[4] Meyers, Brad. A. Taxonomy of Visual Programming and Program Visualization. Journal of Visual Languajes and Computing, 1999

[5] Balagurusamy, E. Programación Orientada a Objetos con C++.

2007. Páginas: 704.

[6] Enseñanzas de cursos de la programación orientada a objetos para los principiantes.

http://www.allbusiness.com/ingenier-a/19990101/3005085-1.html

Ultimo acceso: Enero 2008.

[7] Programación Orientada a Objetos.

http://www.esimez.ipn.mx/acadcompu/apuntes_notas%20breves/programacion_orientada_objetos.pdf.

Ultimo acceso: Marzo 2007.

[8] Lazzarini, Héctor César. Programación Orientada a Objetos.
Formosa. Argentina. 2005

[9] Gayo Avello Daniel, Cernuda del Río Agustín, Cueva Lovelle Juan Manuel. Reflexiones y experiencias sobre la enseñanza de POO como único paradigma Departamento de Informática. Universidad de Oviedo. 2003