Universidad de las Ciencias Informáticas

FISIM: SIMULADOR FÍSICO – MATEMÁTICO INTEGRADO A LA PLATAFORMA DE GESTIÓN DEL APRENDIZAJE ZERA

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autor

Carlos Miguel Pérez Reyes

Tutores

Ing. Marcel Puentes Rojas

Ing. Liana Toledo Bueno

La Habana, Cuba. Junio 2011

"Año 53 de la Revolución"

No pretendemos que las cosas cambien, si siempre hacemos lo mismo. La crisis es la mejor bendición que puede sucederle a las personas y países, porque la crisis trae progresos, la creatividad nace de la angustia como el día de la noche oscura. Es de la crisis que nacen la inventiva, los descubrimientos y las grandes estrategias. Quien supera la crisis se supera a sí mismo sin quedar superado. Quien atribuye la crisis a sus fracasos y penurias, violenta su propio talento y respeta más los problemas que las soluciones, la verdadera crisis es la crisis de la incompetencia. El inconveniente de las personas y los países es la pereza para encontrar las salidas y soluciones. Sin la crisis no hay desafíos, sin desafíos la vida es una rutina, una lenta agonía. Sin crisis no hay méritos. Es en la crisis donde aflora lo mejor de cada uno, porque sin crisis todo viento es caricia.

Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la facultad 4 de la Universidad de las Ciencias Informáticas, así como a dicho centro, a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de junio del año 2011.

Carlos Miguel Pérez Reyes	Ing. Marcel Puentes Rojas	Ing. Liana Toledo Bueno
Firma del autor	Firma del tutor	Firma del tutor

DATOS DE CONTACTO

Autor:

Carlos Miguel Pérez Reyes

Universidad de las Ciencias Informáticas, La Habana, Cuba.

Email: cpreyes@estudiantes.uci.cu / ucicarlos@gmail.com / cpreyes@ uci.cu

Tutores:

Ing. Marcel Puentes Rojas

Universidad de las Ciencias Informáticas, La Habana, Cuba.

Email: mpuentes@uci.cu

Ing. Liana Toledo Bueno

Universidad de las Ciencias Informáticas, La Habana, Cuba.

Email: ltoledo@uci.cu

AGRADECIMIENTOS

Al eterno profesor Tomás López Giménez, que aunque falleció a tan solo 2 años de conocerlo, me enseñó cosas para toda la Vida con esa mezcla de modestia y genialidad única.

Le agradezco cada día a mis padres por haberme permitido existir entre tantas dificultades, sin ellos no sería posible graduarme y contribuir con la Revolución, con la Sociedad Cubana.

Le agradezco a mis tutores, Liana y Marcel por haberme ayudado tanto a pesar de la cantidad de trabajo que tienen.

A mi novia Arlene que me apoyó y me impulsó siempre hacia el éxito.

A todos mis colegas y amigos de la FEU y a mis hermanos del Consejo Científico Estudiantil: Betty, Evelio, Hamler, Susana, Argilagos, Janet, Cire, Yidian.

Muchas gracias a todos mis profesores de la Universidad, que me han soportado durante 5 años: Isabel Lombillo, Maritza Calaña, Basulto, Francisca Mercedes, Alcides, Los Tony, Vislet, Dosagües, Enrique, Lara, Tito, Gulín, Rosa Alicia, Vega, Idelsis, Susana, todos, sin excepción.

Muchas gracias a todos mis amigos y a los que no lo fueron tanto, en esta Universidad, mi Universidad, no solo enseñan ingeniería informática, enseñan a vivir, a perdonar, a rectificar, a amar, a ser valientes, a ser revolucionarios.

Gracias UCI, quien te conoce, quien te vive, no puede dejar de enamorarse de ti. Sigue creciendo, no pares nunca de crecer. Naciste de un sueño y te convertiste en uno.

Del Autor.

RESUMEN

Los simuladores en la educación son una herramienta muy útil de aprendizaje. Facilitan al alumno y profesor el desarrollo del conocimiento con alto grado de autonomía, comprensión de situaciones reales, disminución de gastos, protección ante posibles efectos negativos, entre otros beneficios. El proyecto Alfaomega del Departamento de Producción de Herramientas Educativas de la Universidad de las Ciencias Informáticas tiene como principal objetivo desarrollar una plataforma de gestión del aprendizaje llamada ZERA que integra varios subsistemas y tipologías de software educativo con diferentes propósitos, y que tiene como precedente la Colección Futuro, la cual debe adecuarse a partir de los requerimientos identificados. Para dar solución a los requisitos funcionales y no funcionales del proyecto, se hace necesario desarrollar el simulador FISIM que se integre a dicha plataforma, debido a que ni el simulador FisMat de la Colección Futuro ni las restantes soluciones estudiadas satisfacen los requisitos. Luego de realizados los flujos de trabajo de análisis y diseño, se desarrolla en el marco de la presente investigación la implementación de FISIM empleando un conjunto de buenas prácticas que permiten obtener un producto de calidad con un elevado valor de uso. Lo novedoso de la investigación radica en las potencialidades de integración entre FISIM y ZERA para la autenticación, la generación de trazas, intercambio de información y la posibilidad de diseñar simulaciones definiendo el escenario y los modelos matemáticos con libertad de creación.

PALABRAS CLAVE: educación, simulador, física, matemática, Java

ABSTRACT

Simulators in education are a very useful learning tool. Facilitate to students and teachers knowledge development with a high degree of autonomy, understanding of real situations, reduced costs, protection against potential adverse effects and others benefits. The software project Alfaomega, of the Educational Tools Production Department of the University of Informatics Science, main objective is to develop a learning management platform that integrates ZERA to several subsystems with different purposes, and has a precedent calls "Colección Futuro" (Future Collection), which must be adapted from identified needs. To carry on the functional and nonfunctional requirements of the project, it is necessary to develop the simulator FISIM to be integrated to that platform, because neither, the simulator Fismat of "Colección Futuro" or the other solutions found during the investigation satisfies the requirements. After making the workflow analysis and design, is developed in the framework of this research the implementation of FISIM using a set of best practices to get a quality product with a high use value. The novelty of the research lies in the potential of FISIM and ZERA integration for authentication, trace generation, information sharing, among other benefits, and the possibility of setting the stage design simulations and mathematical models with creative freedom.

KEY WORDS: *education, simulator, physics, mathematics, Java*

ÍNDICE

Introducción

La sociedad, impulsada por el avance científico-tecnológico y apoyada en el uso generalizado de las potentes y diversas tecnologías de la información y las comunicaciones (TIC), ha experimentado cambios que alcanzan al mundo educativo con el desarrollo de nuevos instrumentos, herramientas y métodos en apoyo a los procesos de enseñanza y aprendizaje. Estas tecnologías se han convertido en medios indispensables de las instituciones educativas permitiendo informar, entrenar y guiar en el aprendizaje de una manera motivadora.

En Cuba, se aplican estrategias que tienen como objetivo desarrollar profundas transformaciones tecnológicas para lograr una sociedad informatizada estimulando el uso ordenado y masivo de las TIC con incidencia, tanto en el área económica, social como educativa. La Universidad de las Ciencias Informáticas (UCI) constituye, sin duda alguna, un paso de avance en este sentido y se evidencia en la creación del centro FORTES[1], que tiene entre sus objetivos desarrollar tecnologías que favorezcan la implementación de soluciones de formación aplicando las TIC, a todo tipo de instituciones con diferentes modelos de formación y condiciones tecnológicas.

El proyecto productivo *Alfaomega* del *Departamento de Producción de Herramientas Educativas* de dicho centro tiene como objetivo fundamental, desarrollar una plataforma para la gestión del aprendizaje nombrada *ZERA*[2], la cual tiene su origen en una concepción pedagógica propuesta y desarrollada por pedagogos del Ministerio de Educación de Cuba (MINED) denominada *hiperentorno de aprendizaje* con la característica de integrar en un mismo producto, todas o algunas de las tipologías de software educativo existentes como: multimedias, simuladores, laboratorios virtuales, libros electrónicos, juegos instructivos, entre otros, sustentados en una tecnología hipermedia.

En el marco del proyecto se previó la adecuación de los hiperentornos de aprendizaje de una colección de software desarrollada en Cuba. Esta colección se denomina *Colección Futuro* y es el resultado de un conjunto de software educativos realizados por el MINED donde uno de sus componentes es el software FisMat, un simulador físico-matemático cuya principal función es elaborar animaciones interactivas de objetos que se desplazan a partir de modelos matemáticos. Un análisis de FisMat, determina que el software presenta un grupo de deficiencias y que no cubre los requisitos necesarios.

[1] **FORTES:** Centro de Tecnologías para la Formación
[2] **ZERA:** Del idioma hebreo. simiente, semilla, carozo, semen, esperma descendencia, hijos.

A nivel internacional la mayoría de los simuladores gratuitos empleados en la educación son desarrollados en base a un fenómeno físico específico, permiten pocas configuraciones y provocan que los profesores y estudiantes se vean limitados en su uso. Asimismo algunas compañías de software privativo también desarrollan simuladores para la educación, pero éstas representan altos costos en licencias para las instituciones educativas y poca integración a otros sistemas de gestión de aprendizaje u otras tecnologías *e-learning*.

Derivado del análisis anterior se evidencia que**:** el simulador FISMAT de la Colección Futuro, así como el resto de las soluciones a nivel internacional encontradas durante la investigación, no satisfacen los requisitos funcionales y no funcionales establecidos en la adecuación de la Colección Futuro para el proyecto Alfaomega.

Como consecuencia, se plantea el siguiente **problema de investigación**: ¿Cómo realizar simulaciones físicas a partir de modelos matemáticos para el proceso de enseñanza – aprendizaje de la asignatura física y de forma integrada a la plataforma de gestión del aprendizaje ZERA?

Se presenta como **objeto de estudio:** los procesos para las simulaciones de física.

Para dar solución a la problemática descrita se plantea como **objetivo general de la investigación** implementar un simulador físico – matemático integrado a la plataforma de gestión del aprendizaje ZERA del proyecto Alfaomega.

El **campo de acción** lo constituyen las soluciones informáticas para la simulación física – matemática en la educación.

La **idea a defender** es la siguiente: Si se desarrolla un simulador físico – matemático integrado a la plataforma de gestión del aprendizaje ZERA entonces se incidirá de una forma dinámica e interactiva en el desarrollo del conocimiento en el proceso de enseñanza – aprendizaje de la asignatura física cumpliendo con los requisitos funcionales y no funcionales establecidos en la ingeniería de requisitos.

Los **objetivos específicos** que permitirán dar cumplimiento al objetivo general son:

1. Analizar el estado del arte teniendo en cuenta las principales definiciones, metodología, tecnologías para el desarrollo y soluciones similares.
2. Implementar los componentes necesarios que dan solución a los requisitos propuestos.
3. Integrar la solución a la plataforma de gestión del aprendizaje ZERA.
4. Validar la solución propuesta.

Con el fin de dar cumplimiento al objetivo general se plantean las siguientes **tareas investigativas**:

1. Definición de simulador en la educación.
2. Análisis de soluciones similares.
3. Análisis de las tendencias en el desarrollo de simuladores físicos.
4. Estudio de la metodología y tecnologías para el desarrollo.
5. Análisis de informaciones relevante en la literatura sobre integración a sistemas web.
6. Asimilación de los patrones de arquitectura definidos en el diseño.
7. Descripción de la arquitectura.
8. Refinamiento del diseño propuesto por el analista.
9. Elaboración de los artefactos pertenecientes al flujo de trabajo de implementación.
10. Implementación de los componentes necesarios que dan solución a los requisitos propuestos.
11. Elaborar la estrategia de integración entre el subsistema y la plataforma.
12. Implementar los componentes de la integración.
13. Integrar satisfactoriamente el subsistema a la plataforma ZERA.
14. Diseño de las pruebas del subsistema.
15. Realización las pruebas del subsistema.
16. Análisis de los resultados obtenidos en las pruebas.

El **aporte de la investigación** radica en que se contará con una solución informática que permitirá realizar simulaciones físicas a partir de modelos matemáticos de forma integrada a la plataforma de gestión del aprendizaje ZERA. Este simulador incidirá de forma dinámica e interactiva en el desarrollo del conocimiento de la asignatura de física y se podrá emplear como objeto de aprendizaje de ésta en los centros estudiantiles.

Lo **novedoso de la investigación** está reflejado en el empleo de las potencialidades de integración entre el simulador físico y la plataforma de gestión del aprendizaje ZERA, lo que logrará potenciar las relaciones entre el profesor y el estudiante en el proceso docente – educativo de la asignatura de física. Más allá de simular un fenómeno, este simulador posibilitará el diseño de las simulaciones definiendo el escenario y los modelos matemáticos con libertad de creación.

Métodos de investigación:

Con el fin de resolver y dar cumplimiento a los objetivos y las tareas propuestas se han utilizado los siguientes métodos científicos:

- *Dialéctico:* Apoyado en los procedimientos de inducción, deducción, análisis y síntesis, permite facilitar la comprensión del fenómeno que se investiga.
- *Analítico – Sintético:* permite estudiar los simuladores y buscar información referente a los simuladores físicos, así como las herramientas, lenguajes, metodologías que se utilizan en el desarrollo de estos.
- *Modelación:* específicamente el enfoque de sistema, que permite crear abstracciones con el objetivo de explicar la realidad. El lenguaje de modelado UML permitirá reflejar la estructura, relaciones internas y características de la solución a través de diagramas.
- *Análisis Histórico – Lógico:* permite estudiar cómo ha evolucionado el uso de simuladores y en especial de simuladores físicos – matemáticos en el proceso de enseñanza – aprendizaje.

El contenido estará estructurado en 4 capítulos que recogen el proceso de desarrollo de este trabajo.

Descripción por capítulo:

Capítulo 1: Fundamentación teórica. Se analizan las tendencias, tecnologías y metodologías relacionadas con la implementación, así como las plataformas de desarrollo que la soportan. Se realiza un estudio crítico y valorativo de la plataforma, bibliotecas y patrones de arquitectura usados para el desarrollo del subsistema. Se describen las herramientas utilizadas.

Capítulo 2: Descripción y análisis de la solución propuesta*.* Se realiza una valoración crítica del diseño, un análisis de los requisitos funcionales y no funcionales. Se describe además la arquitectura del subsistema y los patrones de diseño que se utilizan, así como un conjunto de buenas prácticas de programación, necesarias para lograr la robustez, legibilidad, continuidad, usabilidad y estandarización en la solución propuesta.

Capítulo 3: Implementación de la solución propuesta. Se implementa el subsistema en términos de componentes a partir de los artefactos generados en flujos de trabajo anteriores, principalmente en el de diseño.

Capítulo 4: Validación de la solución propuesta*.* Se realiza la descripción y análisis de diferentes tipos y métodos de prueba que permiten validar la solución, así como una evaluación de la ejecución de las pruebas y de los resultados obtenidos.

Capítulo 1: Fundamentación Teórica

Los simuladores en la educación son herramientas muy útiles que facilitan el proceso de enseñanza-aprendizaje. En las últimas décadas se ha diversificado la utilización de éstos en todos los niveles de enseñanza por las ventajas que ofrecen, mejoran la transferencia de conocimientos, incrementan la comprensión de conceptos abstractos y aumentan la motivación de estudiantes por el estudio. Para estos simuladores existen diversas clasificaciones en dependencia de sus objetivos y alcance, asimismo se han empleado diversas tecnologías y herramientas para su desarrollo dando lugar a variadas soluciones en Cuba y el mundo.

1.1 Simulación

Diferentes autores han definido simulación, así es el caso de *David M Himmelblau* y *Kenneth B Bischoff* en su libro *Análisis y Simulación de Procesos, donde* explican simulación como la *"representación de un fenómeno a través de modelos, lo que permite analizar sus características con mayor facilidad sin tener que desarrollar el fenómeno, con lo que se ahorra tiempo y recursos, uno de los objetivos primordiales de una simulación es analizar los resultados para así conocer con anterioridad su comportamiento y en caso posible mejorarlos en el momento que se lleve a cabo el fenómeno en la vida real.*"(1)

Esta definición está enfocada a la simulación de procesos, pero ayuda a comprender los elementos distintivos que debe poseer una simulación: analizar el fenómeno a través de modelos, las características de estos y sus posibles resultados, ahorrando tiempo y recursos.

Thomas H. Naylor plantea que "*simulación es una técnica numérica para conducir experimentos en una computadora digital. Estos experimentos comprenden ciertos tipos de relaciones matemáticas y lógicas, las cuales son necesarias para describir el comportamiento y la estructura de sistemas complejos del mundo real a través de largos períodos de tiempo.*"(2)

Esta definición caracteriza las simulaciones como una solución que permite describir el comportamiento y la estructura de fenómenos reales que generalmente requieren de un período largo de tiempo para llevarse a cabo. Aunque no siempre se requiera de estos períodos de tiempo, esta definición enfoca las simulaciones en el uso de técnicas matemáticas empleando la computación como herramienta para lograr los objetivos propuestos.

Por su parte *Robert E. Shannon* plantea que "*simulación es el proceso de diseñar un modelo de un sistema real y llevar a término experiencias con él, con la finalidad de comprender el comportamiento del*

sistema o evaluar nuevas estrategias -dentro de los límites impuestos por un cierto criterio o un conjunto de ellos - para el funcionamiento del sistema."(3)
En comparación con las anteriores definiciones, esta emplea el término experiencia, al cual se puede llegar a través de la comprensión del comportamiento del sistema y evaluación de nuevas estrategias.

Teniendo en cuenta lo anterior, se asume para la presente investigación, simulación como la representación de un fenómeno a través de un modelo, que posibilita analizar tanto las características del fenómeno como sus posibles resultados ahorrando tiempo y recursos. Se puede hacer uso de técnicas matemáticas y de la computación como herramientas para lograr los objetivos propuestos e incide en el proceso de acumulación y desarrollo del conocimiento por la experiencia que se logra a través de la comprensión del comportamiento del sistema y evaluación de nuevas estrategias.

De acuerdo a su función un simulador puede ubicarse en uno de los siguientes grupos:

1. Especializados en el entrenamiento, con un alto contenido físico – matemático como los simuladores de vuelo, de conducción y de tiro.
2. Los simuladores de procesos industriales que se especializan en optimizaciones mediante el estudio físico de elementos como: turbinas, túneles de viento, mecanismos de combustión, pero igualmente usados en la educación.
3. Especializados en predicciones de fenómenos físicos de la naturaleza como los simuladores meteorológicos y sísmicos.
4. Los de fenómenos puramente físicos de menos aplicación práctica por ser pensados en entornos ideales generalmente utilizados en la educación.

1.2 Simulador para la educación

Según el diccionario de la lengua española un simulador es un dispositivo o sistema diseñado para simular un determinado proceso como si fuera real. Existe una gama amplia de simuladores dirigidos a diferentes ramas de la ciencia y la técnica. La mayoría son sistemas informáticos capaces de manejar un gran número de variables y de gran importancia en las predicciones, entrenamiento, educación, etc. Su aplicación e importancia recae, entre otras razones, en la capacidad que tienen de representar fenómenos reales con diferentes niveles de complejidad, reduciendo los costos, optimizando los procesos y en ocasiones, salvando vidas humanas.

La presente investigación centra su atención en los simuladores para la educación. Los mismos han sido definidos y caracterizados por diferentes autores.

Según lo planteado por el *Ministerio de Educación Nacional de Colombia* en su Portal Educativo, los simuladores son: "*objetos de aprendizaje que mediante un programa de software, intentan modelar parte de una réplica de los fenómenos de la realidad y su propósito es que el usuario construya conocimiento a partir del trabajo exploratorio, la inferencia y el aprendizaje por descubrimiento. Los simuladores se desarrollan en un entorno interactivo, que permite al usuario modificar parámetros y ver cómo reacciona el sistema ante el cambio producido.*"(4)

Los expositores del proyecto *Agrega*, en el marco del *I Encuentro "Nuestra educación innova con Europa: competencia digital"*, abordaron los simuladores de la siguiente forma: "*Son Objetos Digitales Educativos reutilizables (ODEs) correspondientes al nivel de agregación 2 (Objetos de aprendizaje), es decir, a los ODEs más simples e indivisibles que conllevan una función didáctica explícita. Se obtienen al aplicar un diseño instructivo completo (contenidos, actividades, evaluación, etc.) a la combinación de uno o varios Medias o Medias Integrados."*(5)

Objeto de aprendizaje

Para la comprensión correcta de las ideas planteadas anteriormente es necesario definir el término, objeto de aprendizaje (OA): "*Mediador pedagógico que ha sido diseñado intencionalmente para un propósito de aprendizaje y que sirve a los actores de las diversas modalidades educativas. En tal sentido, dicho objeto debe diseñarse a partir de criterios como:*

- *Atemporalidad: Para que no pierda vigencia en el tiempo y en los contextos utilizados.*
- *Didáctica: El objeto tácitamente responde a qué, para qué, con qué y quién aprende.*
- *Usabilidad: Que facilite el uso intuitivo del usuario interesado.*
- *Interacción: Que motive al usuario a promulgar inquietudes y retornar respuestas o experiencias sustantivas de aprendizaje.*
- *Accesibilidad: Garantizada para el usuario según los intereses que le asisten.*"(6)

Conforme a los planteamientos anteriores se asume para la presente investigación, **simulador para la educación** como: Sistema informático que permite realizar simulaciones de fenómenos del mundo real a través de un modelo. Incide en el desarrollo del conocimiento como objeto de aprendizaje, en el cual de forma interactiva se permite a los usuarios estudiar los fenómenos y descubrir el comportamiento de ellos, modificando parámetros y observando los efectos producidos.

1.3 Simuladores estudiados

Como se explicó anteriormente existe un grupo de simuladores físicos de mayor uso en la educación. Éstos se caracterizan por tratar fenómenos físicos puros, de menor aplicación práctica por ser pensados en entornos ideales y basados en modelos matemáticos presentes en la bibliografía docente del nivel de educación correspondiente.

1.3.1 Simuladores físicos para la educación basados en fenómenos específicos

Dentro del grupo de simuladores estudiados existen aquellos que son desarrollados en base a un fenómeno físico específico, permiten pocas configuraciones y se caracterizan por tener el modelo físico – matemático del fenómeno previamente definido. A pesar de esto son muy utilizados en la actualidad. Ejemplos de ellos son:

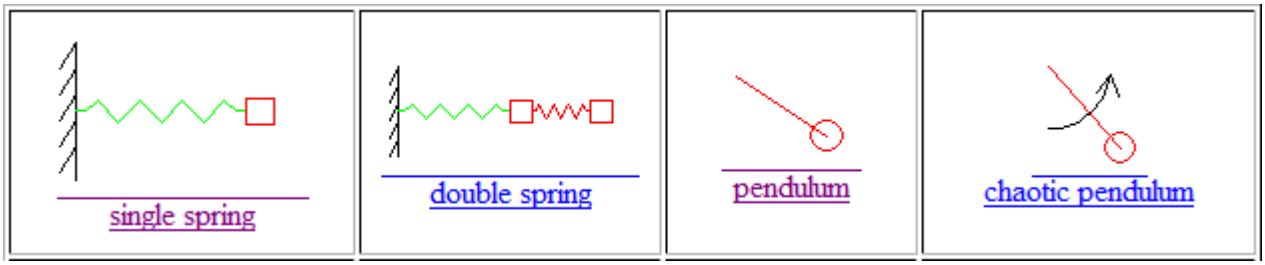


Fig.1 Colección MyPhysicsLab

MyPhysicsLab – PhysicsSimulatior: Fue desarrollado desde 2001. Está dirigido a todos los niveles de educación donde se imparte la asignatura de física. Se distribuye como un Applet de Java. Es código abierto y descargable. Se encuentra accesible desde la web: http://www.myphysicslab.com/. Se pueden cambiar parámetros como la constante elástica. Estas simulaciones físicas forman parte de una colección de más de 24 fenómenos físicos independientes.

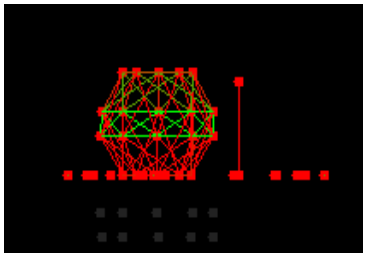


Fig. 2 Real time 3D

Real time 3D physics simulator: Fue desarrollado desde el 2004. Se distribuye como un Applet de Java. Está dirigido a la asignatura de mecánica clásica y física general. Simula el comportamiento de los arreglos de los átomos. Los arreglos se pueden mover, la gravedad puede activarse o desactivarse. Accesible desde la web: http://www.ambromley.co.uk/fizz.html.

Fig. 3 Javascript Ball Simulatior

Javascript Ball Simulatior: Fue desarrollado desde el 2006, e implementado en Javascript. Es accesible desde http://www.uselesspickles.com/jsballs/. Simula el comportamiento de bolas que colisionan entre sí variando la velocidad, la gravedad o la elasticidad.

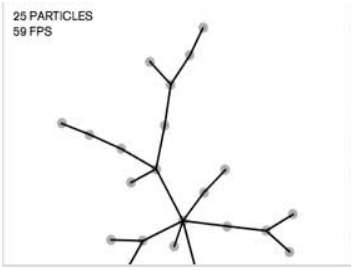


Fig. 4 RandomArboretum

Random Arboretum: Fue desarrollado en Javascript. Pertenece a una colección de 8 simulaciones. Se encuentra accesible desde la web:http://www.queness.com/post/3296/8-amazing-javascript-experiments-of-physic-and-gravity-simulation. Permite realizar simulaciones aleatorias de fenómenos de la naturaleza.

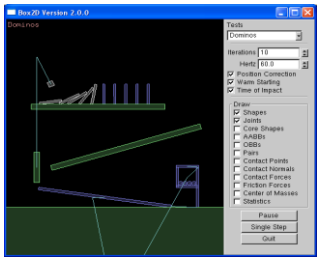


Fig. 5 Blox2D

Blox2D: Está implementado en C++, es una aplicación de escritorio que permite observar cómo interactúan elementos de un escenario previamente definidos en dependencia de los valores que tomen las variables del modelo. Es accesible desde: http://bradenburch.blogspot.com/2010/10/box2d.html.

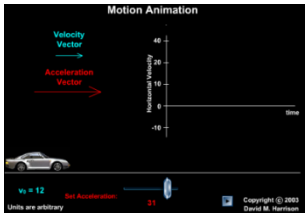


Fig. 6 Motion Animation

Motion Animation: Fue desarrollado en Flash. Pertenece a una colección de más de 100 simulaciones para 14 tipos diferentes de fenómenos físicos. Está dirigido a todos los niveles de educación donde se imparte la asignatura de física. Se encuentra accesible desde la web: http://www.upscale.utoronto.ca/PVB/Harrison/Flash/.

Desventajas técnicas de los simuladores para la educación basados en fenómenos físicos específicos

Se puede constatar que el desarrollo de simuladores basados en fenómenos físicos específicos tiende a aumentar considerablemente la cantidad de simuladores necesarios en una asignatura de física en cualquier nivel educacional. Esto se demuestra en la cantidad de fenómenos que se necesitan estudiar independientemente del nivel. Así lo demuestran también los más de 100 simuladores desarrollados en

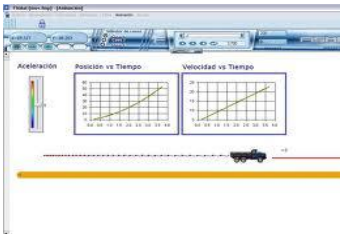
Flash de una misma colección y la colección de más de 24 simuladores de *MyPhysicsLab* que se ejemplifican en este epígrafe.

Ventajas técnicas de los simuladores para la educación basados en fenómenos físicos específicos

La ventaja principal de esta estrategia de desarrollo radica en el nivel de complejidad de los simuladores. Mientras más específica es una simulación más fácil será el desarrollo de la misma. De igual manera este tipo de simuladores resulta muy fácil comprenderlo y manejarlo, tienden a ser muy intuitivos.

1.3.2 Simuladores físicos interactivos para la educación

Los simuladores físicos interactivos se caracterizan por permitir el diseño de las simulaciones, es decir, no proporcionan modelos matemáticos preestablecidos, si no, un entorno de diseño y escenario donde se pueden insertar componentes que interactúan entre sí. Otra característica predominante es que permiten diseñar experimentos eléctricos, de ondas, de movimiento, fuerza y ópticos. Ejemplos de ellos son:



FisMat de la colección futuro: Es un componente de la *Colección Futuro* que permite realizar simulaciones físico-matemáticas. Su principal función es elaborar animaciones interactivas de objetos que se desplazan a partir de modelos matemáticos. Está dirigido al nivel: 10mo, 11no y 12mo. (7)

Fig. 7 Fismat. Colección futuro

Desventajas:

- No existe el código fuente de la aplicación, ni documentación del mismo.
- Presenta problemas de rendimiento.
- Inestabilidad.
- Simulaciones poco configurables que hacen rígido el diseño de estas.
- La información que brinda es insuficiente y de poco uso sobre el trabajo del usuario en el sistema.
- No se integra con ningún sistema del hiperentorno de aprendizaje de la Colección Futuro.
- Desarrollado en Basic año 2004 con herramientas propietarias.
- Módulo de reportes no funcional.

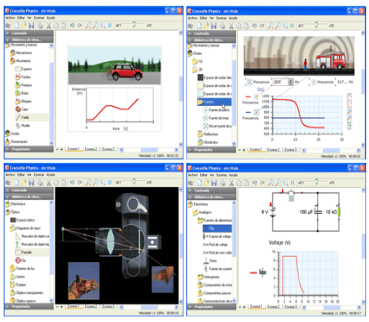


Fig. 8 Crocodile Physics

Crocodile Phisics: Fue desarrollado por Cocodrile en el año 1994. Es una aplicación de escritorio, multiplataforma (Windows/Linux), se encuentra disponible en inglés y español, y permite realizar experimentos físicos de: electricidad, ondas, movimiento, fuerza y óptica. Es accesible desde la web en: http://www.crocodile-clips.com/es/Crocodile Physics.htm

Desventajas:

- No es Open Source.
- No tiene integración con tecnologías **e-learning**.
- Incompatibilidad con versiones recientes de Windows.
- Desarrollado con tecnologías y versiones de java muy atrasadas.

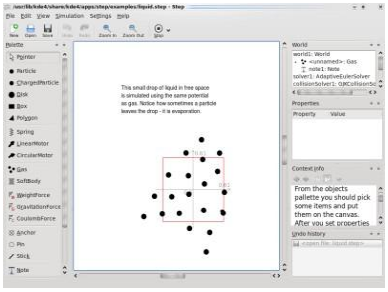


Fig. 9 Step. KDE Simulator

Step: Desarrollado por: *Google Summer of Code Project for KDE* by Vladimir Kuznetsov. Es una aplicación de código abierto realizada para plataforma Linux y en idioma inglés. Especialmente diseñado para la educación. Permite realizar experimentos de fuerza, movimiento, etc. (8)

Desventajas:

- No tiene integración a otros sistemas como **LMS** u otras **tecnologías e-learning**.
- Permite simular pocos fenómenos físicos como óptica, electricidad, oscilaciones y ondas, etc.

Fig. 10 Yenka. Plataforma de simulación.

Yenka: Fue desarrollada en el año 2007, y es conocida como plataforma de simulaciones. Agrupa fenómenos físicos, químicos, matemáticos, biológicos, entre otros. Es una aplicación de escritorio que emplea animaciones 3D. Es una aplicación multiplataforma: Windows, Mac OS X, versión beta inestable para Linux, y soporta varios idiomas. (9)

Desventajas:

- Altos costos de licencia para las instituciones educativas.
- No tiene integración a otros sistemas como **LMS** u otras tecnologías **e-learning**.
- No posee una versión estable para distribuciones libres de Linux.

Tabla 1 Comparación entre simuladores de física interactivos

Simulador	Lenguaje	Integración e-learning	Open Source	Sobre Linux	Sobre Windows	Varios Idiomas
Fismat	Visual Basic	No	No	No	Sí	No
Crocodile Physics	Java	No	No	No	Hasta XP	Sí
Step	C++	No	Sí	Sí	Sí	No
Yenka	Java	No	No	No	Sí	Sí

Otros simuladores para la educación, desarrollados por el proyecto *Agrega* (Plataforma Digital de Contenido Educativo, liderado por el *Ministerio de Educación e Industria de España),* hacen uso de las potencialidades de integración entre los simuladores y las plataformas e-learning a través de estándares de comunicación, si bien en este proyecto no se han desarrollado simuladores físicos, si no, de otros tipos, esta es una característica distintiva que poseen sus soluciones, brindando ventajas como generación de trazas, evidencias, creación de objetos de aprendizaje, entre otras ventajas. (10)

De lo analizado en este epígrafe se concluye que los simuladores basados en fenómenos físicos específicos tienden a aumentar considerablemente la cantidad de simuladores necesarios en la asignatura de física en cualquier nivel educacional. Por lo que los simuladores físicos interactivos, a pesar del alto nivel de complejidad para el desarrollo, son la solución aceptada de acuerdo al alcance y los objetivos de esta investigación.

No se encontraron soluciones de código abierto que cumplan con los requisitos, ni componentes reutilizables para el desarrollo de la solución, debido a que cada solución posee características muy específicas.

Una característica común de estas soluciones es la barra de componentes, que se emplea para agrupar los elementos que se pueden insertar en el escenario de una simulación. La misma es generada mayormente como árboles de componentes, divididos por temáticas que pueden arrastrarse hacia el escenario o insertarlos a través de clics.

Se pudo evidenciar la existencia de mecanismos diversos para interrelacionar objetos en el escenario. De acuerdo a la experiencia de usuarios, las variantes más aceptadas, son aquellas que se pueda obtener

como resultado de la menor cantidad de pasos posibles. Teniendo en cuenta lo anterior se recoge como experiencia la variante aplicada en la plataforma para simulaciones Yenka. Esta contiene una colección de objetos en dependencia del tipo de simulación, pero estos objetos no son independientes, si no, que son la unión de varios objetos pensados para un tipo de simulación específica y requieren pocas modificaciones una vez seleccionados.

1.3.3 Análisis entre tecnologías para el desarrollo

Existen tendencias al desarrollo de simuladores físicos para la educación utilizando unas tecnologías más que otras. El siguiente gráfico ilustra la cantidad de simuladores encontrados en el estudio del *epígrafe 1.3* para cada una de las tecnologías empleadas en su desarrollo.

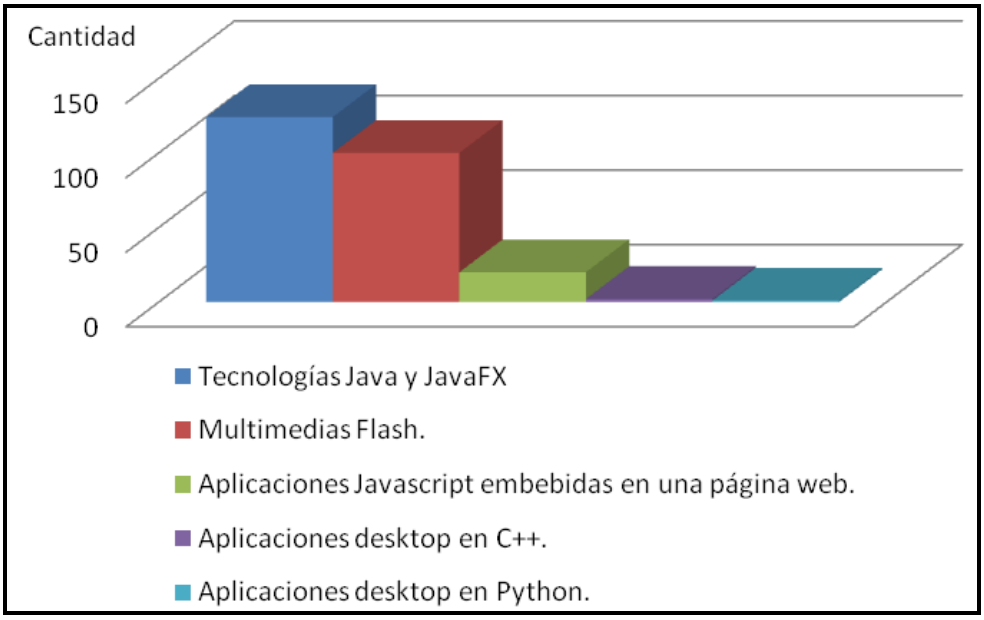


Fig. 11 Tecnologías empleadas en el desarrollo de simuladores.

Las tecnologías empleadas en el desarrollo de software en la actualidad, poseen particularidades que no las hacen mejores o peores entre sí, sino que cada una está más especializada para una tarea concreta con respecto a las otras.

Como ejemplo de lo anterior se encuentra el lenguaje de programación **Python**, el cual tiene como principales características ser: dinámico, multiparadigma, software libre, utilizado por grandes compañías como Google o YouTube pero poco utilizado en el desarrollo de simuladores físicos para la educación (11). Por su parte Blender, programa informático dedicado especialmente al modelado, animación y

creación de gráficos tridimensionales, posibilita programar *script* en Python para automatizar y controlar varias tareas, pero las API[3] de Python integradas a los *IDE*[4] de desarrollo como Komodo o NetBeans, aun usando bibliotecas para la generación de gráfico 2D y 3D, no constituyen una solución aceptada para desarrollar este tipo de soluciones. Otra desventaja es el mecanismo de distribución, ya que para poder ejecutar aplicaciones desarrolladas en Python requieren de instalación o ejecución local.

En otro análisis de lenguajes de programación se puede citar **C++** que se destaca por: su rapidez en operaciones de cálculo y su disponibilidad en múltiples IDEs tanto privativos como libres. Se ha utilizado en la implementación de algunos simuladores como Blox2D. Este lenguaje permite a través de sus bibliotecas el trabajo con gráfico 2D y 3D, pero de una forma muy trabajosa y ralentizando el proceso de desarrollo. En los simuladores estudiados no se logra visualizar animaciones tan visualmente atractivas como las logradas por otras tecnologías. Una de las principales desventajas de las aplicaciones desarrolladas en C++ son los mecanismos de distribución, y la poca vinculación con la web.

De más reciente utilización en la implementación de simuladores para la educación, resalta **Javascript** por ser el lenguaje de scripting más usado en el mundo (12). Es sencillo en comparación con C++, Java o C#, dada su simplicidad sintáctica y su manejabilidad. Sin embargo, esta es una de sus debilidades, ya que la simplicidad se basa en una disponibilidad de objetos limitada, por lo que algunos procedimientos, aparentemente muy sencillos, requieren script complejos. Por otro lado este lenguaje necesita ser interpretado por los navegadores para ejecutarse y aunque existen muchos ejemplos demostrativos de su potencialidad, son más lentos y por consiguiente las simulaciones son poco complejas. Otra desventaja es la incompatibilidad de aplicaciones Javascript con algunos navegadores, lo cual es importante valorar por la diversidad de navegadores que existen.

En la actualidad hay un auge en la utilización de las siguientes tecnologías: **ActionScript3.0** (Lenguaje de programación de Flash), **Flex4 y AJAX.** Flash es una tecnología de creación y manipulación vectorial y de manejo de código para realizar animaciones y contenidos interactivos sin importar la plataforma, ya que los archivos SWF pueden ser vistos tanto en la web desde un navegador o en el desktop con un reproductor Flash. ActionScript 3.0 es un lenguaje orientado a objetos utilizado por Flash y permite desarrollar animaciones con altísimo contenido interactivo. Una de sus ventajas fundamentales es que permite utilizar sonidos, imágenes, videos, componentes predefinidos como botones, etc. con mucha facilidad en comparación con las anteriores tecnologías mencionadas. Por otro lado las aplicaciones

[3] **API:** Interfaz de programación de aplicaciones (del inglés: Application Programming Interface).
[4] **IDE (Entorno Integrado de Desarrollo):** Es un programa informático compuesto por un conjunto de herramientas de programación.

desarrolladas en Flash suelen ser muy pesadas, aumentando el tráfico por la red haciendo más lenta la carga en los navegadores. Desde marzo del 2004 aparece la tecnología **Flex** basada en Flash que junto con **AJAX** dan soporte a las RIA[5], disminuyendo considerablemente el peso de las aplicaciones. Este tipo de aplicaciones permite fácil integración con otras tecnologías del lado del servidor como *Web Services* y *REST*. A pesar que el *SDK*[6] de Flex es libre, el principal problema que presenta es que el IDE de desarrollo *Flex Builder 3.0* no lo es, ni tampoco los *plugins* de Eclipse u otras variantes para el desarrollo, es decir, es posible escribir el código en un editor de texto y compilarlo con *mxmlc*, pero para poder utilizar verdaderamente las potencialidades, se debe pagar licencia. Los simuladores físicos para la educación desarrollados con Flex, generalmente combinan varias tecnologías de *Adobe* como *Photoshop*, imprimiéndoles una alta calidad gráfica y efectos visuales de una forma muy atractiva. Existen varias tecnologías sobre Flex que facilitan mucho el desarrollo de simuladores: *Action Script physics engines, 3d engines, Sparl Physics, etc.*

En diciembre del 2008 se lanza la tecnología **JavaFX** y el lenguaje **JavaFX Script,** para competir por el espacio que ocupaban Flash de Adobe y Silverlight de Microsoft en el desarrollo de RIA, aplicaciones de escritorio, aplicaciones móviles y televisión. Se crea en un momento donde la popularidad de las *UI*[7] de Java cayó a favor de Flash, Ajax y otras soluciones ya mencionadas. Parte de sus beneficios son: es de código abierto, compatible con los *IDE* NetBeans y Eclipse a través de *Plugins;* permite desarrollar aplicaciones multiplataforma con alto contenido multimedia; incorpora el tipo de dato "duración"; incluye el uso de la API SceneGraph; funciona integrado con *Swing;* incorpora *Swing2D;* facilita la creación de objetos 3D; etc. (13). Por las características anteriores, los simuladores de física desarrollados con ésta tecnología poseen un alto grado de realismo, considerado parte importante del proceso de simulación de fenómenos de la vida real. Además JavaFX posee la librería "*Physics Engine*" considerado una de las más importantes ventajas para el desarrollo. (14)

[5] **RIA:** Aplicaciones Enriquecidas de Internet (aplicaciones web) que tienen las características y capacidades de aplicaciones de escritorio.
[6] **SDK (Software Development Kit):** kit de desarrollo de software es generalmente un conjunto de herramientas de desarrollo que le permite a un programador crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software y frameworks.
[7] **User Interface:** Interfaz de usuario.

Razones de selección de Java

Del análisis anterior se arriba a la conclusión que las tecnologías de Java son las más utilizadas en el desarrollo de simuladores. Su aporte y ventajas para el desarrollo la ubican en las tecnologías más apropiadas para la implementación del simulador y así lo demuestra el estudio de las tendencias y las potencialidades que brinda la tecnología en sí. Como parte de los criterios de selección se tiene en cuenta además, el dominio de esta tecnología por pate del equipo de desarrollo. Por lo que se ratifica la selección del lenguaje propuesta en el flujo de trabajo *Análisis y Diseño* y se procede a realizar un estudio más profundo sobre *el mismo.*

1.4 Tecnología Java

Las tecnologías de Java y el lenguaje de programación, fueron diseñados con el propósito de crear un lenguaje que pudiera funcionar en redes computacionales heterogéneas (redes de computadoras formadas por más de un tipo de computadora), y que fuera independiente de la plataforma en la que se vaya a ejecutar.

Características:

- **Simple:** elimina la complejidad de los lenguajes como "C" y da paso al contexto de los lenguajes orientados a objetos. La filosofía de programación orientada a objetos es diferente a la programación convencional.
- **Robusto:** maneja la memoria de la computadora. No hay necesidad de preocuparse por apuntadores, memoria que no se esté utilizando, etc. Java realiza todo esto sin necesidad que el programador se lo indique.
- **Seguro:** tiene ciertas políticas que evitan se puedan codificar virus con este lenguaje. Existen muchas restricciones, especialmente para los Applet, que limitan lo que se puede o no hacer con los recursos críticos de una computadora.
- **Portable:** como el código compilado de Java (conocido como bytecode) es interpretado, un programa compilado de Java puede ser utilizado por cualquier computadora que tenga instalado el intérprete de Java.
- **Independiente a la arquitectura:** al compilar un programa en Java, el código resultante es un tipo de código binario conocido como bytecode. Este código es interpretado por diferentes computadoras de igual manera, solamente hay que instalar un intérprete para cada plataforma. De

esa manera Java logra ser un lenguaje que no depende de una arquitectura computacional disponible.

- **Dinámico:** no requiere que se compilen todas las clases de un programa para que este funcione. Al realizar una modificación a una clase, Java se encarga de realizar un Dynamic Bynding o un Dynamic Loading para encontrar las clases.
- **Multiplataforma:** debido a la Máquina Virtual de Java (JVM) las aplicaciones desarrolladas en Java funcionan en Linux, Windows, Mac OS, y en cualquier sistema operativo para el cual exista una JVM. (15)

1.4.1 Máquina Virtual de Java

La JVM es uno de los elementos fundamentales de la plataforma Java. Básicamente se sitúa en un nivel superior al hardware del sistema sobre el que se pretende ejecutar la aplicación, y este actúa como un puente que entiende tanto el bytecode, como el sistema en sí. Así, cuando se escribe una aplicación Java, se hace pensando que será ejecutada en una máquina virtual Java en concreto, siendo ésta la que en última instancia convierte de código bytecode a código nativo del dispositivo final. (16)

La gran ventaja de la JVM es aportar portabilidad al lenguaje de manera que un programa .class escrito en Windows puede ser interpretado en un entorno Linux. Tan solo es necesario disponer de dicha máquina virtual para dichos entornos. De ahí el famoso axioma que sigue a Java, "escríbelo una vez, ejecútalo en cualquier parte", o "Write once, run any where". (17)

1.4.2 Plataforma J2SE

La plataforma *J2SE*[8] proporciona un entorno de escritorio Core Java y desarrollo de aplicaciones Java. Tiene el compilador, herramientas, módulos de ejecución, y la API de Java que le permiten escribir, probar, implementar y ejecutar Applet y aplicaciones. (18)

J2SE tiene dos implementaciones de la máquina virtual:

- Java HotSpotClient VM: La máquina virtual por defecto, preparada para obtener el máximo rendimiento en la ejecución de aplicaciones en el entorno cliente, por ejemplo, reduciendo al máximo el tiempo de inicio de una aplicación Java.
- Java HotSpot Server VM: Preparada para obtener el máximo rendimiento en la ejecución de aplicaciones en el entorno de los servidores.

1.4.3 Java Web Start

El software de Java Web Start permite descargar y ejecutar aplicaciones Java desde la Web. Dicho software posee las siguientes características:

- Permite activar las aplicaciones con un simple clic.
- Garantiza que se está ejecutando la última versión de la aplicación.
- Elimina complejos procedimientos de instalación o actualización.
- Se incluye en el entorno de ejecución de Java (JRE) como parte de JRE 5.0, por lo que al instalar el JRE, se instala automáticamente.
- Se ejecuta automáticamente cuando se descarga por primera vez una aplicación que utiliza esta tecnología y además guarda dicha aplicación localmente en la memoria caché del equipo. De este modo, las subsiguientes ejecuciones son prácticamente instantáneas, ya que los recursos necesarios están disponibles de forma local. Cada vez que se inicia la aplicación, el software comprueba si en la sede Web de la aplicación hay una versión nueva disponible; si es así, la descarga y la ejecuta de forma automática. (19)

La ejecución de una aplicación con el software de Java Web Start se realiza de la siguiente manera:

- Desde un navegador: haga clic en un vínculo de una página Web.
- Desde un icono del escritorio: si utiliza una aplicación con frecuencia, puede crear un acceso directo en su escritorio o en el menú Inicio.

[8] **J2SE (Java 2 Standard Edition):** La versión estándar es la más común y cuenta con todo lo necesario para desarrollos de software y acceso a aplicaciones Java.

- Desde el Visualizador de la memoria caché de aplicaciones de Java.

1.4.4 Beneficios del protocolo JNLP

Para el despliegue de FISIM es necesario utilizar la generación automática en el servidor web, de un archivo formato JNLP, Java Networking Lauching Protocol, especificación usada por Java Web Start que permite tener centralizado en un servidor web un programa, evitando problemas de distribución e instalación. Al instalar cualquier aplicación, normalmente se siguen los siguientes pasos:

- Se descarga de internet.
- Se introduce un medio extraíble (CD/DVD/Disco USB...)
- Se instala en el ordenador
- Se ejecuta

El programa javaws de Java permite hacer esto de un modo más fácil y transparente al usuario, ya que se pueden realizar estos pasos simplemente accediendo sobre un enlace mientras se utiliza el navegador, de modo que la descarga, instalación y ejecución se realizan de modo transparente al usuario.

Java Web Start no usa Applet, descarga aplicaciones Java y necesita, por tanto, de una máquina virtual, además, viene incluido en el JRE de java desde la versión 1.4.

Cualquier enlace JNLP, al iniciar el proceso de ejecución, pide autorización al usuario. Las aplicaciones pueden estar firmadas (firma electrónica) como es el caso de esta solución, para asegurar el remitente de la aplicación de modo que pueden seguir el modelo de seguridad de la plataforma Java 2 para asegurar la integridad de los datos que se obtienen a través de la red, de forma que no se produzcan ataques de tipo *Man in the Middle*[9], *DNS poisoning*[10], o corrupción de datos. (20)

1.4.5 Beneficios del formato de ficheros JAR

Aunque se emplee para el despliegue el formato JNLP. El programa javaws de Java descarga aplicaciones Java en forma de ficheros JAR (Archivos Java).

Un fichero JAR proporciona los siguientes beneficios:

- **Seguridad:** firmar digitalmente el contenido de un fichero JAR. Los usuarios que reconozcan la firma pueden permitir al software privilegios de seguridad que de otro modo no tendría.

[9] **MitM o intermediario, en español:** Es un ataque en el que el enemigo adquiere la capacidad de leer, insertar y modificar a voluntad, los mensajes entre dos partes sin que ninguna de ellas conozca que el enlace entre ellos ha sido violado.

[10] **DNS cache poisoning / DNS Poisoning / Pharming**: Es una situación creada de manera maliciosa o no deseada que provee datos de un Servidor de Nombres de Dominio (DNS) que no se origina de fuentes autoritativas DNS.

- **Compresión:** el formato JAR permite comprimir los ficheros para ahorrar espacio.
- **Empaquetado versionado:** un fichero JAR puede contener datos sobre los ficheros que contiene, como por ejemplo información sobre la versión.
- **Portabilidad:** el mecanismo es una parte estándar del corazón de la API de la plataforma Java. Un JAR se puede ejecutar en Windows, Linux, Macintosh o en cualquier sistema operativo.

1.5 Integración entre sistemas

La integración con la Plataforma de Gestión del Aprendizaje ZERA es el requisito no funcional donde recae la principal novedad de la investigación con relación a similares en el mundo.

Existen diversas formas para lograr la comunicación entre diferentes sistemas, la presente investigación abarcó el estudio de las diferentes tecnologías que pueden ser empleadas en la integración de aplicaciones de escritorio desplegadas bajo la tecnología Java Web Start de Java con sistemas web.

Servicios web

Conjunto de protocolos y estándares (**SOAP, XML, WSDL, UDDI, WS-Security, etc.**) que sirven para intercambiar datos entre aplicaciones a través de la red desarrolladas en lenguajes de programación diferentes y ejecutadas sobre cualquier plataforma. (21)

Ventajas:

- Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.
- Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.
- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

Desventajas:

- Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI (Remote Method Invocation), CORBA o DCOM (Distributed Component Object Model).
- Bajos rendimientos al adoptar un formato basado en texto. Y es que entre los objetivos de XML no se encuentra la concisión ni la eficacia de procesamiento.
- Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera.

CORBA

Es un estándar que establece una plataforma de desarrollo de sistemas distribuidos facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos. Fue definido y está controlado por el Object Management Group (OMG) que define las APIs, el protocolo de comunicaciones y los mecanismos necesarios para permitir la interoperabilidad entre diferentes aplicaciones escritas en diferentes lenguajes y ejecutadas en diferentes plataformas, lo que es fundamental en computación distribuida. (22)

Ventajas:

- Estandarizado y con múltiples implementaciones (no se depende de un fabricante)
- Las especificaciones se adoptan por consenso.
- Permite integrar aplicaciones heterogéneas.

Desventajas:

- Es muy complejo de utilizar.
- Las especificaciones tardan en desarrollarse.

DCOM

Distributed Component Object Model (DCOM), en español Modelo de Objetos de Componentes Distribuidos, es una tecnología propietaria de Microsoft para desarrollar componentes software distribuidos sobre varios ordenadores y que se comunican entre sí. Extiende el modelo COM de Microsoft y proporciona el sustrato de comunicación entre la infraestructura del servidor de aplicaciones COM+ de Microsoft. Ha sido abandonada en favor del framework .NET (23)

Ventajas:

- Fácil uso.
- Rapidez en el desarrollo.
- Reducción de los Costos de Integración y mantenimiento.

Desventajas:

- Problemas de seguridad.
- IDL sin herencia de Objetos.
- Poco soporte para hilos.
- No tiene Servicio de Nombres.

CURL

El principal propósito y uso de cURL es automatizar transferencias de archivos o secuencias de operaciones no supervisadas. Es por ejemplo una buena herramienta para simular las acciones de un usuario en un navegador web.

Ventajas:

- Ha sido escrita en más de un lenguaje.
- Soporta múltiples protocolos de red (http, https, ftp, gopher, telnet, dict, file y ldap).
- Multiplataforma.

Desventajas:

- Poca documentación
- Problemas de inestabilidad.
- Significativamente más complejo de utilizar que otras variantes similares como wGet.

Bibliotecas y clases de Java

Java en su inicio nació como lenguaje para la red y sólo sucesivamente se convirtió en un verdadero lenguaje de programación. Como Java está muy ligado a las redes proporciona las herramientas necesarias recogidas fundamentalmente en el paquete **java.net**. Este consta de 6 interfaces, 27 clases y 11 Excepciones desde el JDK 1.0. (24)

Ventajas:

- Buena documentación, ejemplos y comunidad de desarrolladores activa.
- Permite realizar conexiones y transacciones a través de la red utilizando múltiples protocolos.
- Fácil de utilizar.
- Eficiente y rápido desarrollo.

Desventajas:

- De forma independiente no soluciona la interconexión entre aplicaciones, necesita consumir recursos de red.

Las tecnologías existentes desde la década de los 90 para la interconexión entre aplicaciones web y escritorio como las analizadas en este epígrafe (Servicios web, CURL, CORBA, DCOM, etc.), no permiten el desarrollo eficiente de algunas funcionalidades de FISIM. Esta situación está presente en otras empresas de software y corporaciones como Google, YouTube y Facebook, que implementan sus propios

mecanismos. Las razones principales de la no adopción de éstos, están dadas por los bajos rendimientos, como los provocados por el uso de servicios web, o una muy compleja implementación y poca compatibilidad en el caso de CORBA o DCOM respectivamente.

De acuerdo a lo anterior se evidencia la necesidad de implementar un nuevo mecanismo de interacción, ya que las alternativas encontradas en la investigación no satisfacen las necesidades del desarrollo y no permiten, de forma eficientes, la interacción entre el subsistema FISIM y la plataforma ZERA.

1.6 Entorno Integrado de Desarrollo

Un Entorno Integrado de Desarrollo (IDE) es un entorno de programación, que en su forma más básica, está compuesto por un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (conocido por sus siglas en inglés GUI). Estos comprenden un conjunto significativo de ventajas y herramientas que agilizan el proceso de desarrollo y permiten una mayor integración entre diferentes utilidades. La selección del IDE se realizó en los flujos de trabajo iniciales de la etapa de desarrollo. ***NetBeans 6.9*** es el IDE que utiliza el proyecto para la implementación de los componentes de Java en cada uno de los subsistemas que lo integran.

En la presente investigación se realiza un estudio de las ventajas y potencialidades de este IDE con el objetivo de hacer un uso más eficiente de todas las ventajas que brinda y lograr así, un desarrollo más rápido y eficiente.

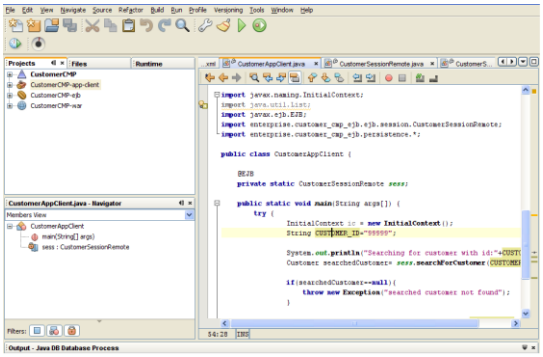


Fig. 12 IDE Netbeans

Ventajas:

- Garantiza una apariencia y funcionamiento común de las aplicaciones una vez desplegadas sobre diversos entornos como: Solaris, GNU/Linux, Microsoft Windows y MacOS.

- Con el modelado UML que ofrece NetBeans los analistas pueden diseñar la aplicación usando UML y luego generar el código desde el diagrama UML. Esto puede hacerse en ambos sentidos, pues se puede actualizar el modelo si se hacen cambios en el código fuente.
- Tiene la posibilidad de integrarse con herramientas de modelado como Visual Paradigm.
- Integración con Sistemas de Control de Versiones tales como SVN, CVS, Mercurial y Git. Desde el editor es posible realizar la administración de estos sistemas versionados, commit, branch, importar, exportar, revert, clonar, etc.
- Tiene un avanzado editor de código fuente que analiza el código mientras se va escribiendo. Puede ser totalmente personalizado.
- Posee un sistema para examinar todo los directorios de cada proyecto, haciendo reconocimiento y carga de clases, métodos y objetos, para acelerar la programación.

Valor añadido:

- Propone un "esqueleto" para organizar el código fuente, el editor conjuntamente integra los lenguajes como HTML, JavaScript y CSS.
- Integración con Symphony y ZenFramework.
- Contiene un editor visual para descriptores de despliegue y un seguimiento de los errores de HTTP Web.
- Servicios Orientados a arquitectura (Service-Oriented Architecture, SOA). El Enterprise Pack añade todas las funcionalidades necesaria al IDE para servicios orientados a arquitectura profesionales.

1.7 Metodología de desarrollo

El proyecto Alfaomega desde su inicio adoptó el uso de la metodología del Proceso Unificado de Desarrollo (Rational Unified Process) para guiar el ciclo de vida del software. En el actual epígrafe se analizan las características de esta metodología para alcanzar un dominio de las ventajas, buenas prácticas y elementos a tener en cuenta en la etapa de implementación.

1.7.1 Características de RUP

- Dirigido por Casos de Uso (CU): En RUP los CU constituyen la forma de especificar los requisitos del sistema, guiar su diseño, implementación y prueba. Son un elemento integrador y una guía del

trabajo, pues inician el proceso de desarrollo y proporcionan un hilo conductor, permitiendo establecer trazabilidad entre los artefactos que son generados en las diferentes actividades del proceso de desarrollo.

- Centrado en la arquitectura: la arquitectura de un sistema es la organización o estructura de sus partes más relevantes, lo que permite tener una visión común entre todos los involucrados (desarrolladores y usuarios) y una perspectiva clara del sistema completo, necesaria para controlar el desarrollo. En el caso de RUP además de utilizar los CU para guiar el proceso se presta especial atención al establecimiento temprano de una línea base de la arquitectura que no se vea fuertemente impactada ante cambios posteriores durante la construcción y el mantenimiento.
- Iterativo e incremental: el trabajo se divide en partes más pequeñas o mini proyectos, permitiendo que el equilibrio entre CU y arquitectura se vaya logrando durante cada mini proyecto y así durante todo el proceso de desarrollo. Cada mini proyecto se puede ver como una iteración (un recorrido más o menos completo a lo largo de todos los flujos de trabajo fundamentales) del cual se obtiene un incremento que produce un crecimiento en el producto.
- Presenta 4 fases: inicio, elaboración, construcción y transición, y un total de 9 flujos de trabajo que corresponden a 2 disciplinas distribuidos de la siguiente forma:

Disciplinas:

- *Modelación del negocio:* describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.
- *Requerimientos:* define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.
- *Análisis y diseño:* describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.
- *Implementación:* define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.
- *Prueba:* busca los defectos a los largo del ciclo de vida.
- *Despliegue:* produce un entregable del producto y realiza actividades como empaque, instalación, asistencia a usuarios, etc. para entregar el software a los usuarios finales.

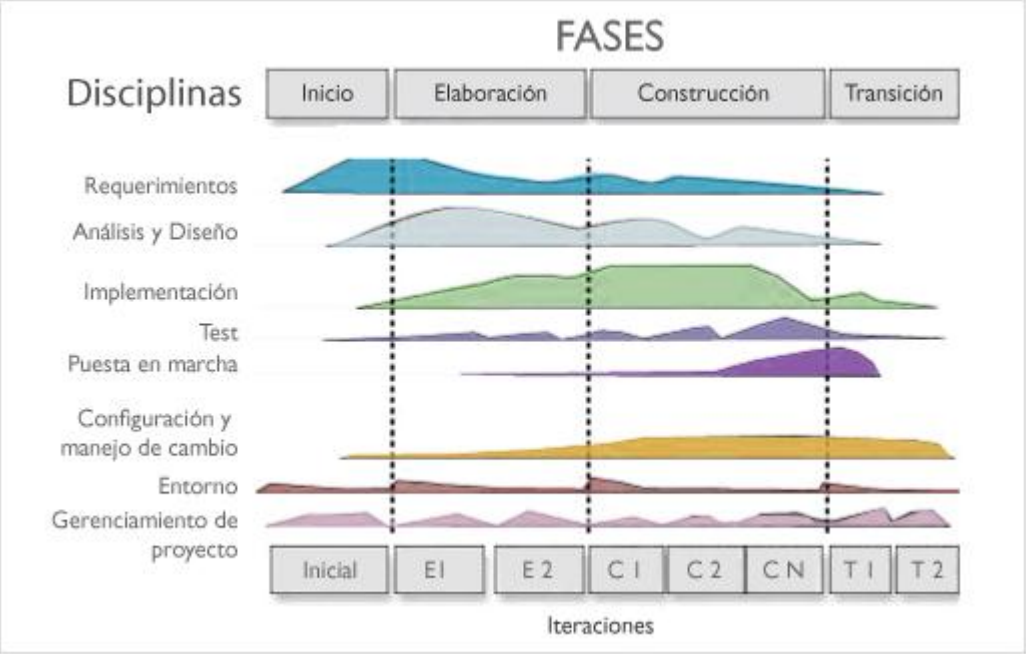


Fig. 13 Fases, iteraciones y disciplinas

1.7.2 Artefactos, actividades y flujos fundamentales de la implementación

Artefactos:

- **Componentes:** código fuente, clases, ejecutables, bibliotecas, documentos, etc.
- **Modelo de implementación:** describe cómo se organizan los componentes de acuerdo a las estructuras y los mecanismos de modularización que se disponga en el entorno y lenguaje de programación elegido, y cómo dependen los componentes unos de otros.

Actividades y flujos fundamentales:

1. **Implementar Arquitectura**: Identificar los componentes e incorporarlos al modelo. Genera el diagrama de componentes y asignar componentes a los nodos.
2. **Implementar las Clases de Diseño:** Generar el código fuente y clases partiendo de la descripción de las clases del diseño, implementar las operaciones, realizar pruebas de unidad e integrar el sistema.

1.7.3 Artefactos, actividades y flujos fundamentales de la fase de prueba

El objetivo de esta fase es realizar pruebas sobre la estructura del sistema que se va formando con los módulos implementados. Las pruebas que deben hacerse son de dos tipos: de integración y del sistema.

Artefactos:

- **Casos de prueba:** Un caso de prueba indica una manera de probar el sistema. Incluye qué probar junto con su entrada o salida y bajo qué condiciones probar.
- **Procedimiento de pruebas:** Un procedimiento de prueba especifica cómo realizar uno o varios casos de prueba.
- **Evaluación de pruebas:** Es una evaluación de los resultados de la prueba realzada.

Actividades y flujos fundamentales:

1. **Planificar y diseñar las pruebas:** Planificar y diseñar los casos de prueba y el orden en que se van a llevar a cabo.
2. **Pruebas de integración y sistema:** Ejecutar y comparar con los resultados esperados.

1.8 Lenguaje de modelado. UML

El Lenguaje Unificado de Modelado (UML) es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad, y además es el lenguaje que da soporte a la metodología RUP. Ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. Con este lenguaje, se pretende unificar las experiencias acumuladas sobre técnicas de modelado e incorporar las mejores prácticas actuales para lograr la abstracción del sistema y sus componentes.

*"Entre sus **objetivos fundamentales** se encuentran:*

- *Ser tan simple como sea posible, pero manteniendo la capacidad de modelar toda la gama de sistemas que se necesita construir.*
- *Ser lo suficientemente expresivo para manejar todos los conceptos que se originan en un sistema moderno, tales como la concurrencia y distribución, así como también los mecanismos de la ingeniería de software, como son el encapsulamiento y los componentes.*
- *Ser un lenguaje universal, como cualquier lenguaje de propósito general.*
- *Imponer un estándar mundial."* (25)

1.9 Herramientas CASE de modelado

"Las herramientas CASE, de sus siglas en inglés Computer-Aided Software Engineering (Ingeniería de Software asistida por computadora) son aplicaciones informáticas enfocadas a la automatización del proceso de desarrollo de software, documentación, generación de código, chequeo de errores y gestión del proyecto, permitiendo la reutilización, portabilidad y estandarización de la documentación." (26)

Las herramientas CASE de modelado de UML posibilitan realizar el análisis y diseño de sistemas orientados a objetos y muchas de ellas permiten incluso la generación de código fuente que resulta muy útil en la transformación del código en un modelo y en la sincronización de un modelo con el código al final de una iteración. Por lo anterior dichas herramientas juegan un papel fundamental en el aumento de la productividad durante el ciclo de vida del proyecto porque ayudan a reducir costos y tiempo.

1.9.1 Visual Paradigm

En el equipo de desarrollo se definió la herramienta Visual Paradigm por la dirección del proyecto para realizar el modelado UML del sistema a desarrollar debido a las diversas características que tiene, dentro de las cuales cabe destacar las siguientes:

- Utiliza UML como lenguaje de modelado.
- Soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue.
- Ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste.
- Se integra con herramientas Java tales como: Eclipse, JBuilder, NetBeans, Oracle JDeveloper, entre otras.
- Genera la documentación del proyecto automáticamente en varios formatos, por ejemplo: Web y PDF.
- Permite una mejor integración e ingeniería tanto directa e inversa entre modelado y código, lo cual facilita el mantenimiento de los artefactos de RUP y las trazas entre ellos.
- Permite control de versiones.
- Presenta una interfaz de uso intuitiva y con muchas facilidades a la hora de modelar los diagramas que soportan la Ingeniería de Requerimientos.
- Es una herramienta robusta, con gran usabilidad y portabilidad. (27)

1.10 Arquitectura del sistema

Según Philippe Kruchten[11]: La arquitectura de software, juega un papel fundamental en el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales, como la confiabilidad, escalabilidad, portabilidad y disponibilidad.

La arquitectura propuesta para el subsistema FISIM en el flujo de trabajo de análisis y diseño está basada en implementar la solución orientada a objetos mediante una variante del patrón arquitectónico Modelo – Vista – Controlador (MVC) el cual permite separar los datos de la aplicación, la interfaz de usuario y la lógica de control, en tres componentes distintos. Esta arquitectura fue propuesta en enero del 2008 por Sun Microsystem y su principal variación con respecto a la tradicional es que las notificaciones de cambios de estado en el modelo de objetos se comunican a la vista a través del controlador. Por lo tanto, el controlador media entre el flujo de datos entre objetos del modelo y la vista, en ambas direcciones.

1. La *vista* reconoce que una acción a través de eventos (eventos de escucha): botón o arrastrar una barra de desplazamiento, etc.
2. La *vista* llama al método correspondiente en el *controlador*.
3. El *controlador* accede al modelo y realiza la acción correspondiente a la acción del usuario.
4. Si el modelo ha sido alterado el cambio se envía al controlador. (28)

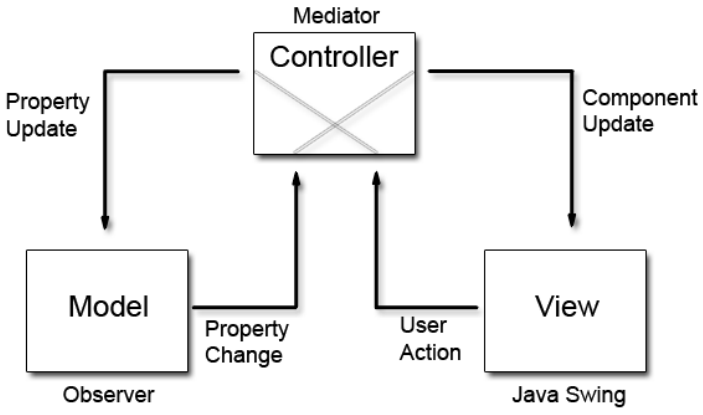


Fig. 14 Diseño MVC colocando controlador entre el Modelo y la Vista.

[11] **Philippe Kruchten:** Director de Proceso de Desarrollo (RUP) en Rational Software, desarrollador del modelo de vista 4 +1.

1.11 Patrones

En el flujo de trabajo de análisis y diseño se propone el uso de los patrones GRASP (Patrones generales de software para asignar responsabilidades) y el patrón Singleton, para el desarrollo del subsistema FISIM, los cuales se estudian a continuación.

Patrones GRASP:

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Dentro de esta clasificación de patrones se encuentran:

- Bajo Acoplamiento
- Alta Cohesión
- Experto
- Creador
- Controlador
- Polimorfismo
- Fabricación pura
- Indirección
- Variaciones protegidas

Estos son usados proporcionando a las clases la reusabilidad y flexibilidad necesarias. De manera general, a continuación se brinda una breve descripción de lo que realiza cada uno de estos patrones y para qué son utilizados fundamentalmente:

- **Experto:** *Asignar una responsabilidad al experto en información. La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada. Este patrón se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen* (29). Se pone de manifiesto en las clases entidades, ya que estas son las expertas en la información que poseen.
- **Creador:** Asignar a la clase B la responsabilidad de crear una instancia de clase A en alguno de los siguientes casos (B agrega los objetos de A; B contiene a los objetos de A; B registra las instancias de los objetos de A; B utiliza específicamente los objetos de A; B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado). El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento (29) Principalmente, este patrón es utilizado en las clases controladoras, ya que estas son las encargadas de crear las instancias de las clases que controlan.

- **Controlador:** Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad y otros). La mayor parte de los sistemas reciben eventos de entrada externa, por lo cual, si se recurre a un diseño orientado a objetos, hay que elegir los controladores que manejen esos eventos de entrada (29).
- **Alta Cohesión:** Asignar una responsabilidad de modo que la cohesión siga siendo alta. Una clase tiene responsabilidades moderadas en un área funcional y colabora con las otras para llevar a cabo las tareas. Una clase con mucha cohesión es útil porque es bastante fácil darle mantenimiento, entenderla y reutilizarla. Su alto grado de funcionalidad, combinada con una reducida cantidad de operaciones, también simplifica el mantenimiento y los mejoramientos. La ventaja que significa una gran funcionalidad también soporta un aumento de la capacidad de reutilización (29).
- **Bajo Acoplamiento:** Asignar una responsabilidad para mantener bajo acoplamiento. Debe haber pocas dependencias entre las clases. Si todas las clases dependen de todas ¿cuánto software podemos extraer de un modo independiente y reutilizable en otro proyecto? .Para determinar el nivel de acoplamiento de clases, son muy buenos los diagramas de colaboración de UML. Uno de los principales síntomas de un mal diseño y alto acoplamiento es una herencia muy profunda. Siempre hay que considerar las ventajas de la delegación respecto de la herencia. Además soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecienta la oportunidad de una mayor productividad (29).
- **Polimorfismo:** Siempre que se tenga que llevar a cabo una responsabilidad que dependa del tipo, se tiene que hacer uso del polimorfismo, cuando las alternativas o comportamientos relacionados varían según el tipo (clase), asigne la responsabilidad para el comportamiento a los tipos para los que varía el comportamiento. Asigna el mismo nombre a servicios en diferentes objetos.
- **Fabricación pura:** se da en las clases que no representan un ente u objeto real del dominio del problema, sino que se ha creado intencionadamente para disminuir el acoplamiento, aumentar la cohesión y/o potenciar la reutilización del código. Es la solución cuando el diseñador se encuentre con una clase poco cohesiva y no tenga otra clase en la que implementar algunos métodos.

Patrón Singleton:

También se utiliza el patrón de creación Singleton está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella. El patrón Singleton hace que la clase sea responsable de su única instancia, quitando así este problema a los clientes.

1.12 Bibliotecas para la generación de gráficas

Las gráficas, también llamadas Chart en inglés, son una vía visualmente atractiva y útil para los usuarios que requiere del análisis de datos, éstas favorecen la comprensión y análisis de resultados de una manera más sencilla. Es común encontrar gráficas en forma de barras, circulares, por puntos y otras de mayor o menor complejidad. Para la implementación del requisito funcional *Graficar* se hace necesario el estudio de diferentes bibliotecas que permitan de una manera ágil llevar a cabo este proceso. Las bibliotecas encontradas durante la investigación para este fin y que cumplen con la condición de ser Open Source fueron: Jfreechart, Chart2D, jGraph, rChart, JBChart y MonarchChart.

Criterios de selección a tener en cuenta:

- Desarrolladas en Java
- Open source
- Bien documentada
- Comunidad activa de desarrollo
- Rápida
- Fácil de usar

Luego de un análisis de estas bibliotecas se determina de acuerdo a los criterios anteriores y lo encontrado durante la investigación que las bibliotecas más utilizadas en el mundo y de mayor aceptación por la comunidad de desarrolladores eran Chart2D y jFreeChart. En ambas la generación de gráficas es visualmente atractiva y poseen características similares que permiten realizar en su totalidad todo lo requerido, pero luego de un estudio de rendimiento, se determina que Chart2D posee menor rapidez que jFreeChart, tardando 3 segundos más para generar las mismas gráficas. Por lo que se selecciona jFreeChart como librería para la creación de gráficas.

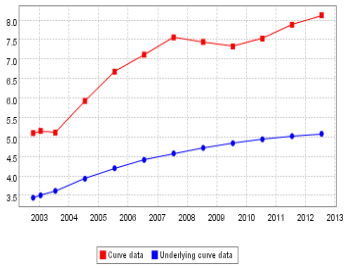


Fig. 15 Ejemplo de gráfica generada por Jfreechart.

Características:

- API bien documentada que admite una amplia gama de tipos de gráficos.
- Diseño flexible y fácil de extender.
- Soporte para los componentes Swing, archivos de imagen y gráficos vectoriales.
- Es Open Source. Se distribuye bajo los términos de la GNU Lesser General Public Licence (LGPL).

Conclusiones del capítulo

- No se encontraron en el estado del arte soluciones o componentes de código abierto reutilizables que cumplan con los requisitos, ratificando así la necesidad de desarrollar un simulador físico matemático propio.
- La solución debe estar enfocada a la creación de un simulador físico lo más interactivo posible a pesar del alto nivel de complejidad para su desarrollo.
- El principal valor añadido de la solución, con respecto a sus similares en el mundo, recae en la interacción con la plataforma ZERA, convirtiéndose en un nuevo paradigma entre diferentes tipologías de software educativo y las plataformas de gestión del aprendizaje.
- En relación con lo definido en los flujos de trabajo anteriores se reafirma el empleo de: Java y tecnología Java Web Start como lenguaje de programación y forma de distribución respectivamente, NetBeans como entorno de desarrollo integrado, RUP como metodología de desarrollo y Visual Paradigm como herramienta de modelado. Todas ellas aportan gran ventaja en el flujo de trabajo implementación.

Capítulo 2: Descripción y análisis de la solución propuesta

En la fase inicial del desarrollo de la aplicación (*Modelación del Negocio, Análisis y Diseño*) de software empleando RUP, se identificaron los requisitos funcionales y no funcionales del sistema y se generaron un conjunto de artefactos de entrada para el flujo de trabajo *Implementación*. De esta manera se describen las características y el comportamiento que el sistema debe tener una vez implementado. Para la construcción de un software de calidad es fundamental que se describan, a nivel de detalle las funcionalidades y se agrupen en módulos de acuerdo a los requisitos que abarca cada una, se defina de forma adecuada la arquitectura del sistema y se identifique la línea base.

2.1 Diagrama de Casos de Uso del Sistema

Durante el flujo de trabajo de requerimientos se realiza el diagrama de casos de uso del sistema. Este permite observar la relación existente entre los actores y los casos de uso del sistema, y posibilita identificar funcionalmente el alcance del sistema que se va a construir. A continuación se muestra el diagrama del subsistema FISIM:

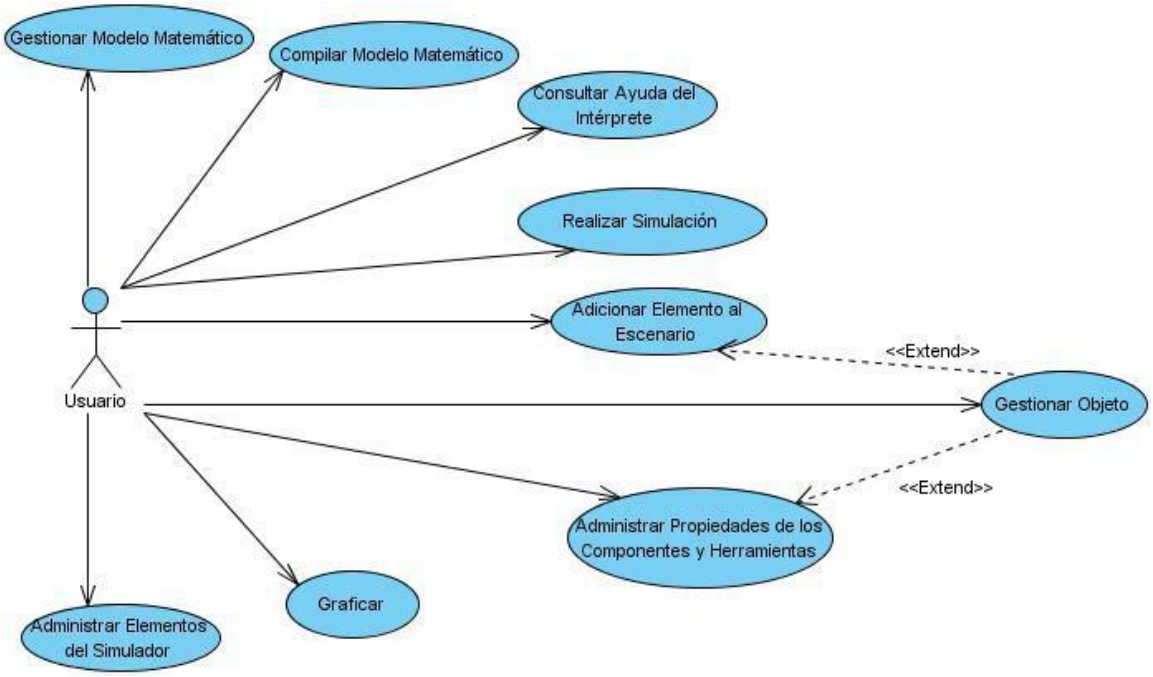


Fig. 16 Diagrama de Casos de Uso del Sistema

2.2 Requerimientos de configuración, conectividad y seguridad[12]

Luego de realizados los flujos de trabajo de modelación de negocio, requerimientos, y análisis y diseño, se tiene la información precisa y detallada de lo que el sistema debe hacer y las restricciones que va a tener.
A continuación se muestran requerimientos de configuración, conectividad y seguridad que debe tener el subsistema y de qué manera éstos se garantizan:

1. Contar con el motor Java 6, que también podrá ser descargado de la plataforma, en la misma ubicación que la aplicación del laboratorio de física.
2. La aplicación requiere validación de uso por el servidor local o central. Si no obtiene la validación, no puede iniciar.
3. La validación permite al servidor determinar, los datos que enviará a la aplicación, dependiendo del período escolar en que está inscrito el usuario. Es decir, el usuario recibe solo el contenido y material necesario para realizar las prácticas del nivel educativo al que pertenece.
4. La aplicación requiere conexión web todo el tiempo que esté activa, pues llama del servidor local o central, elementos dinámicos que la conforman.
5. El esqueleto que queda almacenado en cada PC no es funcional sin la conexión y validación previamente mencionadas.

2.3 Forma de operación

De igual forma se ha descrito el flujo de actividades que se harán de forma secuencial para que el usuario tenga acceso al subsistema FISIM y pueda trabajar en él.

1. Instalación del “esqueleto” y bibliotecas de la aplicación en la PC.
2. Ingreso de usuario y contraseña para validación.
3. Una vez autorizado el usuario por el servidor local o central, este envía los elementos que permiten la ejecución del laboratorio.
4. Una vez dentro, el usuario tiene la posibilidad de crear una simulación nueva o ver un catálogo de temas y subtemas de contenidos.
5. Una vez seleccionado el contenido sobre el cual se desea ver una simulación previamente elaborada, se carga un área de trabajo con las herramientas, objetos y elementos que correspondan al tema seleccionado y que permitan llevar a cabo la simulación.
6. Toda acción dentro del laboratorio genera traza y se reportan a la plataforma.

[12] Acta de Aceptación de Requerimientos y especificaciones. Documentación legal del proyecto.

2.4 Descripción de los módulos del subsistema

El subsistema FISIM está dividido en cuatro módulos que agrupan las funcionalidades que a continuación se describen:

Módulo Intérprete

En él se agrupan las funcionalidades que posibilitan realizar modelos matemáticos que describen el comportamiento de un fenómeno físico mediante la ayuda de un intérprete.

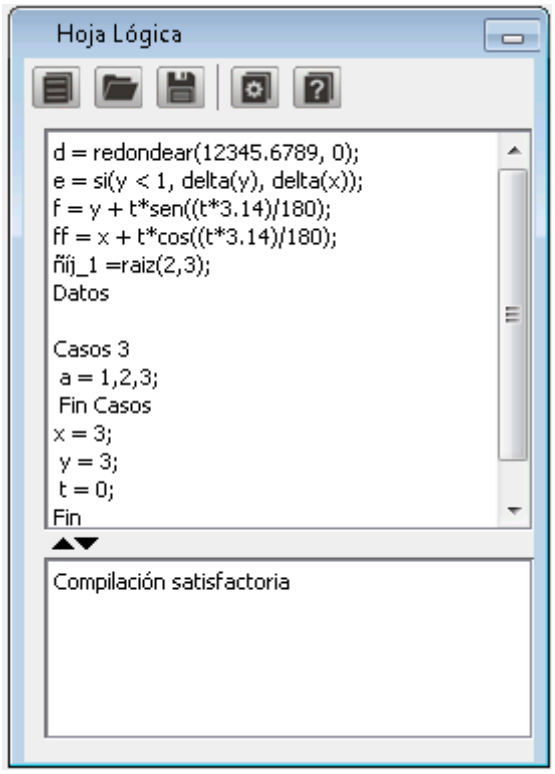


Fig. 17 Interfaz Hoja lógica

Gestionar Modelo Matemático:

- Permite definir y modificar un modelo matemático mediante el trabajo con variables, fórmulas, expresiones matemáticas y funciones (seno/coseno) de forma tal que pueda describirse el comportamiento de un fenómeno físico; además permite guardar y abrir un modelo previamente definido.

Compilar Modelo Matemático:

- Permite interpretar un modelo matemático y mostrar los errores que contenga.

Descripción:

Este módulo permite definir el o los modelos matemáticos base de la simulación. Para llevar a cabo este proceso se procedió a definir un lenguaje propio e interpretable que permitiera estandarizar la forma en que los usuarios describen el modelo matemático. De esta manera se puede salvar, abrir o modificar un modelo matemático y ser reutilizable por otras simulaciones.

Una vez escrito el código bajo ciertas restricciones del propio lenguaje se procede a interpretarlo, este proceso puede o no generar mensajes de errores o confirmaciones que son mostradas al usuario.

A través de la clase controladora AlfaParser se realizan las operaciones del intérprete. El diagrama de clases describe las clases principales que intervienen en este proceso:

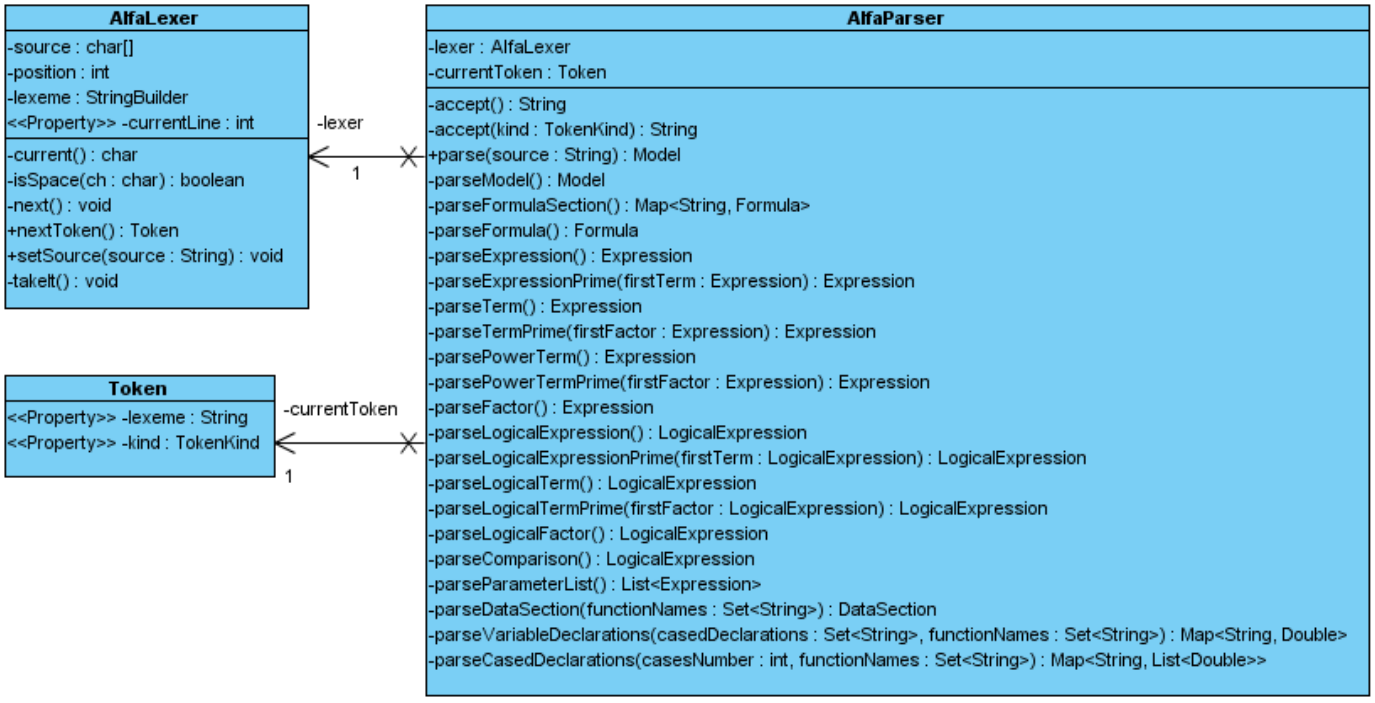


Fig. 18 Diagrama de Clases del intérprete

Módulo Escenario

Abarca las funcionalidades relacionadas con la gestión de objetos y el comportamiento de estos a la hora de realizar la simulación del modelo matemático definido.

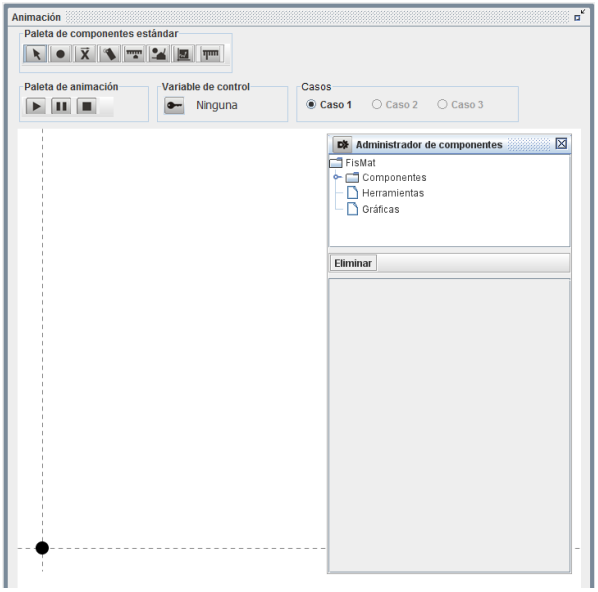


Fig. 19 Interfaz Escenario

Gestionar Objeto:

- Permite incluir y/o eliminar objetos en un escenario (zona de trabajo).
- Permite seleccionar un objeto y ver las propiedades del mismo. Las cuales pueden ser: posición del objeto en la pantalla, dimensión del objeto, etc.
- Define el comportamiento de cada objeto con relación a las variables del modelo matemático definido anteriormente. Por ejemplo, los objetos pueden ser: medidores (utilizados para medir o conocer el valor de una variable definida en el modelo matemático), puede ser una imagen (para representar y poder visualizar el comportamiento de una variable del modelo matemático, según como se comporte la imagen), y cualquier elemento que pueda adicionarse al escenario al cual se le asocia una variable o función del modelo matemático.

Definir Comportamiento de Objetos:

- Permite realizar las siguientes acciones sobre un objeto previamente incluido en cualquier posición dentro del escenario: modificar sus dimensiones, visibilidad, cambiar su posición en el eje “x” y/o en el eje “y” y moverlo hacia otra posición dentro del escenario.

- Permite realizar simulación, mostrando el comportamiento de los objetos según el modelo matemático descrito.
- Permite iniciar, pausar o detener una simulación.
- Permite administrar propiedades de los componentes y herramientas.
- Los objetos se ubican en el administrador de componentes desde el mismo momento en que son adicionados al escenario de trabajo. El actor puede ver las propiedades de un objeto seleccionado, como su posición (x; y) en el escenario, dimensión y otras características según el tipo de objeto.
- Permite eliminar un objeto seleccionado.

Descripción:

- Esta interfaz se encuentra dividida por componentes jPanel que permiten la agrupación y alineación de los componentes de una manera más sencilla y facilitando posteriores cambios de la interfaz.
- Estos jPanel conforman diferentes paletas como la de componentes que permite insertar: Puntos, vectores, etiquetas, cursores, imágenes, gráficas y medidores. La paleta de animación que permite inicial la animación, pausarla o detenerla. La paleta de que controla la variable de control en función del tiempo y la que permite seleccionar los casos que se quieres analizar.

Módulo Instrumentos

Gestiona los elementos involucrados en la simulación, y permite la creación y administración de trazas en el sistema.

Adicionar Elemento al Escenario:

- Brinda la posibilidad de incluir al escenario cualquiera de los siguientes elementos: cursores, medidores, etiquetas, puntos y vectores. Estos elementos permiten medir durante la simulación el valor de las variables definidas en el modelo matemático y mostrar de manera puntual o vectorial el comportamiento de estas.

Graficar:

- Permite graficar el comportamiento que describe un objeto según el valor o los valores que tome la variable o la función que fue asociada al objeto durante la simulación.

Medir Variables:

- Permite medir durante la simulación el valor que obtienen las variables o funciones que están definidas en el modelo matemático mediante objetos que identifiquen a estas.

Administrar Elementos del Simulador:

- Permite guardar una simulación y abrir una simulación que ha sido previamente guardada.
- Permite mostrar u ocultar el intérprete de modelos matemáticos, el escenario, el administrador de objetos y la ayuda general del software.

Módulo Integración

Contiene las funcionalidades que permiten la integración del subsistema FISIM a la plataforma de gestión de aprendizaje ZERA.

Autenticar:

- Permite autenticar al usuario a través del módulo de servicios de la plataforma.

Realizar verificación de usuario:

- Permite a través de servicios con la plataforma implementar la forma de autenticación *SSO*[13].

Generar de trazas:

- Permite crear un conjunto de trazas de su uso una vez autenticado en la plataforma, guardándolas en la base de datos para poder hacer uso de ellas.

Capturar información:

- Permite capturar información en forma de parámetros asociada al servidor web y formas de autenticación.

2.5 Descripción de la arquitectura

En consecuencia con la arquitectura MVC establecida se estructuró el sistema en forma tal que los datos de la aplicación, la interfaz de usuario y la lógica de control estuvieran separadas en tres componentes distintos. En esta variación, las clases de control *Escenario y Simulación* median entre el modelo (puntos, vectores, imágenes, etc.) y la vista (Interfaz de la aplicación) de la siguiente forma:

La *vista*, es decir la interfaz de la aplicación que interactúa con el usuario y la clase *Escenario*, reconoce que una acción fue realizada a través de eventos, ya sea: clic a un botón, arrastrar una barra de desplazamiento, etc.

[13] **Single sign-on (SSO):** es un procedimiento de autenticación que habilita al usuario para acceder a varios sistemas con una sola instancia de identificación.

Al dispararse un evento, la *vista* llama al método correspondiente en la clase *controladora (Simulacion)*. Esta como intermediaria*,* accede al modelo donde se encuentran las clases entidad y realiza la acción correspondiente a la acción del usuario en la vista. Si el modelo cambia de estado, entiéndase cambio de las propiedades de un punto, gráfica, etc., como suele suceder, este a través de la controladora, envía la respuesta o mensaje a la vista actualizando los datos.

Ejemplo de aplicación de la arquitectura en el sistema:

1. El usuario escribe un modelo matemático y da clic en el botón compilar.
2. El evento *jButtonCompilarActionPerformed* escucha la acción y llama al método *Interpretar(tp_codigo_fuente.getText)* alojado en la clase controladora *Simulacion*.
3. La clase controladora interactúa con la clase del modelo *Alfaparser,* que junto a otras clases, es la encargada de interpretar el modelo matemático.
4. La información es persistente en la controladora y a través de ésta, la vista conoce si fue o no posible compilar el modelo, que errores presenta e interactuar con el mismo.

2.6 Buenas prácticas empleadas

2.6.1 Ofuscación del código

Es relativamente sencillo obtener el código fuente (fichero .java) a partir del código compilado (fichero .class). Por lo que resulta imprescindible tomar precauciones para evitar que esto suceda ya que cualquier aplicación en Java puede ser víctima de ingeniería inversa y así obtener su código fuente trayendo como consecuencia: modificación al programa, eliminación de contadores de caducidad, controles de licencia, etc. Esto es un problema también desde el punto vista comercial porque nadie podría comercializar productos desarrollados en Java.

La ofuscación de código se refiere al acto deliberado de realizar un cambio no destructivo, ya sea en el código fuente de un programa informático o código máquina (cuando el programa está en forma compilada o binaria) con el fin de que no sea fácil de entender o leer. Un código ofuscado es aquél código que ha sido enrevesado específicamente para hacerlo ininteligible.

Esta ofuscación se consigue en la práctica por medio de la inclusión de bucles irrelevantes, cálculos innecesarios, comprobaciones absurdas, nombres de funciones y de variables que no tienen nada que ver con su cometido, funciones larguísimas que no sirven para nada, interacciones inverosímiles entre variables y funciones, etc.

Dado que ofuscar manualmente puede introducir gran cantidad de errores y reducir drásticamente su eficiencia, existen herramientas que permiten realizar esta función como es el caso de *GPL ProGuard* que permite, además de ofuscar las clases Java, optimizarlas, comprimirlas y añadirles información de pre verificación, de forma que la carga de las clases en Java SE 6 sea más rápida y más eficiente.

2.6.2 Elementos importantes a tener en cuenta durante la ofuscación

Como parte de los requisitos funcionales de la aplicación es necesario almacenar las simulaciones una vez creadas, de igual forma recuperarlas y modificarlas. Para lograr esto se utilizaron los objetos serializados de Java que brindan muchas facilidades para el almacenamiento.

El proceso de ofuscación provoca cambios a los nombres de las clases, paquetes, atributos, etc. Una vez que esto ocurre cambia la señalización de las clases y por consiguiente no es posible recuperar el fichero almacenado.

Previendo esto, la ofuscación del código tuvo en cuenta dicho problema y como solución se excluyeron de la ofuscación los elementos que provocan el cambio de Serie de las clases.

```
22  -keepclassmembers class * extends java.io.Serializable {
23      static final long serialVersionUID;
24      private static final java.io.ObjectStreamField[] serialPersistentFields;
25      !static !transient <fields>;
26      !private <fields>;
27      !private <methods>;
28      private void writeObject(java.io.ObjectOutputStream);
29      private void readObject(java.io.ObjectInputStream);
30      java.lang.Object writeReplace();
31      java.lang.Object readResolve();
32  }
```

Fig. 20 Fragmento de código del fichero de configuración de ProGuard

Como se puede observar en la Fig. 17 se preservan:

- Todos los nombres de miembros de clase que hereden de java.io.Serializable.
- Los atributos serialVersionUID;
- Todos los nombres de atributos y métodos de clases que hereden de java.io.Serializable.
- Métodos de almacenamiento y recuperación de objetos serializados.

2.6.3 Estándar de codificación

Al comenzar un proyecto de software, es necesario establecer un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada, ya sea de forma paralela o en nuevas versiones. El estándar de codificación establece cómo operar con la base de código existente.

A continuación se describe el estándar de codificación empleado en la implementación del simulador:

- **Constantes:** los nombres de las constantes se escriben completamente con mayúsculas, en caso de que el nombre esté compuesto por más de una palabra, estas son separadas por el carácter "_".
- **Variables:** las variables comenzarán con una letra minúscula, y a continuación la palabra o palabras que conformen el nombre de la variable se mantendrá en minúsculas.
- **Clases:** los nombres de las clases se escriben con letra inicial mayúscula, en caso de tener un nombre conformado por más de una palabra estas se escribirán separadas por el carácter "_" una a continuación de la otra comenzando siempre cada palabra con una letra mayúscula y el resto en minúscula.
- **Métodos miembros de clases:** los nombres de los métodos miembros de una clase comenzarán con letra inicial minúscula, en caso de que el nombre esté compuesto por más de una palabra la primera palabra será escrita en minúscula completamente y el resto comenzará con letra mayúscula.
- **Nombres de enumerativos:** los enumerados comenzarán con la letra "E" y a continuación la palabra o palabras que conformen el nombre, cada una comenzando con letra mayúscula y el resto en minúsculas. Los valores de los enumerados cumplen con las mismas reglas que las constantes.

2.6.4 Documentación del código fuente

Durante la implementación se utilizaron las siguientes buenas prácticas de documentación y comentarios:

- Documentación en forma explicativa los pasos que se van ejecutando con el objetivo de lograr la mayor claridad posible.
- Empleo de oraciones completas al documentar código para lograr que se entienda correctamente lo realizado.

- Documentar mientras se programa, es la mejor forma de documentar todos los aspectos sin omitir detalles.
- Documentar cualquiera aspecto que no sea obvio en el código de esta forma se evitan falsas suposiciones y futuros errores.
- Documentar cada rutina agregando nombre del desarrollador, fecha, parámetros de entrada, valores de retorno, precondiciones, pos condiciones, dependencia con otros métodos o funciones y descripción general del algoritmo.
- Evitar agregar comentarios al final de líneas de código, salvo en el caso de declaraciones.
- Antes de la entrega de la aplicación, eliminar todos los comentarios superfluos y/o temporales con la finalidad de evitar confusiones en su mantenimiento.

Para generar la documentación se empleó la utilidad javadoc que genera la documentación en formato HTML a partir del código fuente Java. Javadoc es el estándar de la industria para documentar clases de Java entre otras funciones de análisis de estructuras de una aplicación Java.

Para generar la documentación con Javadocse utilizaron etiquetas de HTML o ciertas palabras reservadas precedidas por el caracter "@".Estas etiquetas se escriben al principio de cada clase, miembro o método, dependiendo de qué objeto se desee describir, mediante un comentario iniciado con "/**" y acabado con "*/".

A continuación se explican las palabras reservadas más utilizadas en la documentación.

Tabla 2 Palabras reservadas utilizadas en la documentación del subsistema.

Tag	Descripción	Uso
@author	Nombre del desarrollador.	nombre_autor
@param	Definición de un parámetro de un método.	nombre_parametro descripción
@return	Informa lo que devuelve el método.	descripción
@throws	Excepción lanzada por el método	nombre_clase descripción
@version	Versión del método o clase.	versión

Conclusiones del capítulo

Las descripciones detalladas de los flujos de trabajo análisis y diseño propició el entendimiento y las bases de la implementación, arrojando con claridad las funcionalidades del sistema así como sus restricciones.

La arquitectura y el uso de patrones garantizarán la robustez, extensibilidad y escalabilidad del código, siendo éste, un paso fundamental en el proceso de desarrollo del software.

El uso de buenas prácticas garantizarán la documentación, usabilidad y el cumpliendo con un estándar de codificación único.

Capítulo 3: Implementación de la solución propuesta

El flujo de trabajo de implementación se inicia a partir de los resultados obtenidos durante el diseño del sistema. El subsistema se implementa en términos de componentes, los cuales se van integrando de forma incremental. Esto conforma el modelo de implementación que brinda una visión general de los componentes que van a ser implementados, su organización y dependencia en los nodos físicos en que será desplegada la aplicación.

3.1 Mecanismo de integración con la plataforma. JURL

JURL es el mecanismo diseñado e implementado en el marco del presente trabajo para interconectar el subsistema FISIM con la plataforma educativa ZERA. Además se ha convertido en el estándar de interconexión entre ZERA y el resto de los subsistemas basados en la tecnología Java Web Start del proyecto Alfaomega.

Tiene como objetivo fundamental sustituir mecanismos poco eficientes e implementar un conjunto de funcionalidades que permitan:

1. Autenticación.
2. Mecanismos seguro de transferencias de datos por canales inseguros de comunicación.
3. Sincronización.
4. Trasferencia de ficheros.
5. Generación de trazas.
6. Acceso a la base de datos.
7. Entre otros.

Características:

- Está conformado por una serie de componentes implementados en Java del lado de la aplicación de escritorio y un módulo de servicio en la aplicación web implementado en PHP.
- Se basa en los protocolos de Hipertexto HTTP y HTTPS.
- No requiere de implementaciones complejas, solo debe gestionar la forma en que recibe y transmite la información.

Ventajas:

- No es necesario implementar ninguna funcionalidad del negocio puesto que su principal ventaja reside en hacer uso de estas a conveniencia.
- Es totalmente independiente de sistema gestor de base de datos que se utilice.

- Es transparente a nuevas versiones o sistemas web, el único requisito es que las páginas sean generadas dinámicamente.
- No es necesario utilizar servidores adicionales ni aplicaciones de terceros.
- Es más eficiente en el desarrollo puesto que se elimina la necesidad de otros mecanismos más engorrosos de implementar y de servidores adicionales.
- Es más eficiente que otros similares como los servicios web y más compatible con aplicaciones web que otras soluciones como DCOM.

Requisitos:

- La aplicación web debe contar con un módulo de servicios, el cual no es más que una o varias páginas generadas dinámicamente. Estas permitirán recibir o enviar información y constituye el mecanismo mediante el cual se atenderán las solicitudes de diferentes aplicaciones al interactuar con la plataforma.

Formas de funcionamiento:

- En el módulo servicios se procesará la información solicitada por el grupo de aplicaciones de escritorio, sin tener que implementar elementos adicionales puesto que se utilizan las mismas funcionalidades de la aplicación web.
- Este módulo verifica además la identidad del cliente que accede al servicio.

Las bibliotecas de JURL permiten:

1. Enviar información encapsulada con POST en forma parametrizada usando protocolo HTTP o HTTPS.
2. Recibir información usando protocolo HTTP o HTTPS interpretado la respuesta del servidor web en forma cadenas.
3. Capacidad de utilizar protocolos seguros de comunicación SSL.
4. Formatear información con diferentes estándares en dependencia de las necesidades de la aplicación web.
5. Generación de KEY únicas como complemento de seguridad.
6. Búsqueda automática de proxy basado en mecanismos multiplataforma.
7. Búsqueda automática de dirección IP NonLoopBack Address basado en mecanismos multiplataforma.
8. Validación de paquetes y pérdida de conectividad.

jURL: Clase estática para consumir servicios de PHP permitiendo la autenticación o el intercambio de información entre el subsistema y la plataforma de gestión del aprendizaje.

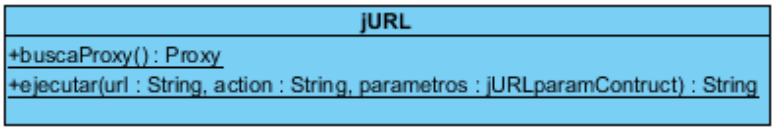


Fig. 21 Clase jURL

jURLparamContruct: Clase que permite construir la cadena de parámetros necesaria para utilizar por el método estático ejecutar de la clase jURL.

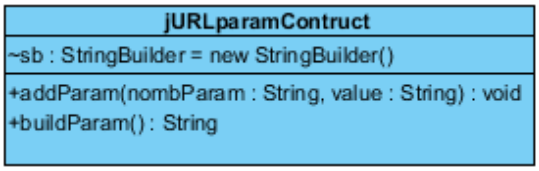


Fig. 22 Clase jURLparamContruct

3.3 Diagramas

3.3.1 Diagrama de paquetes

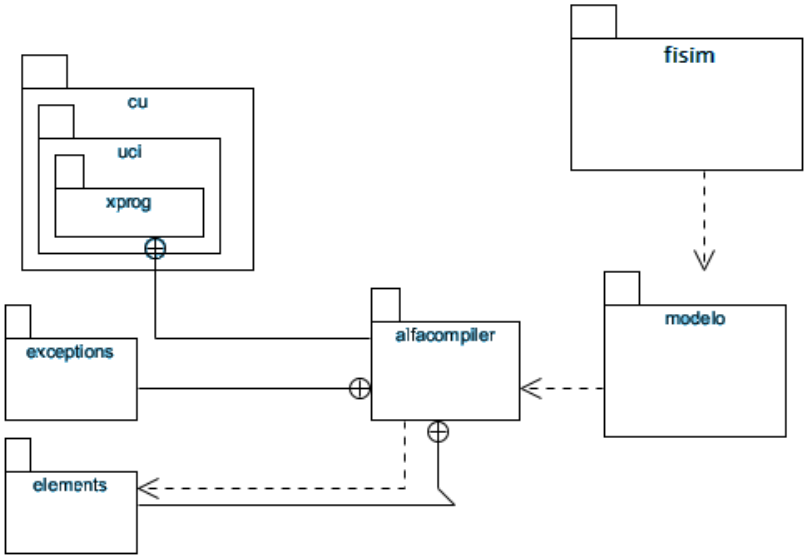


Fig. 23 Diagrama de Paquetes del sistema

3.3.2 Paquetes, descomposición en clases y sus relaciones

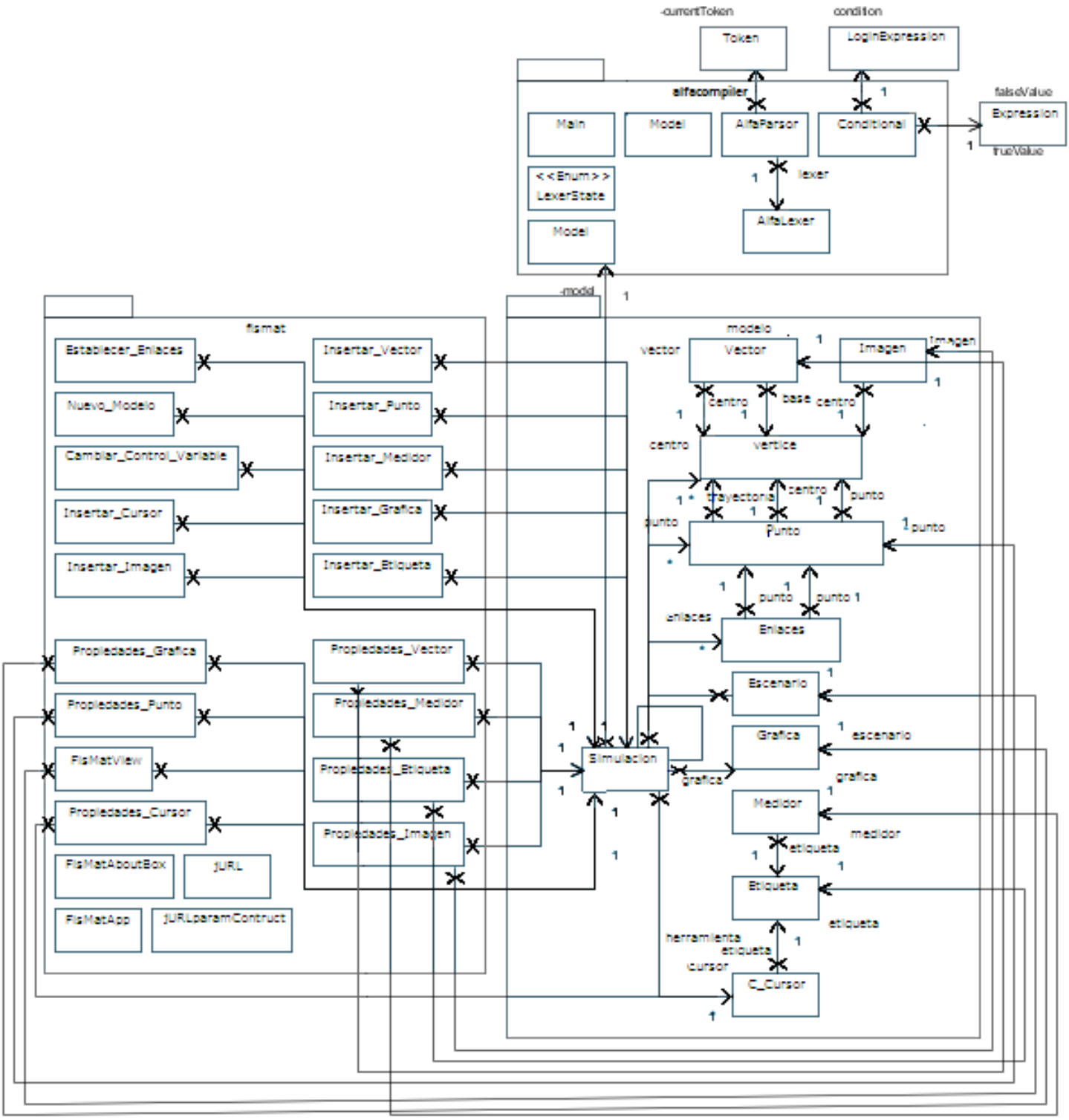


Fig. 24 Diagrama de clases del subsistema en paquetes.

3.3.3 Diagrama de los componentes principales

El diagrama de componentes ilustra los elementos que son utilizados en la construcción del sistema. UML define 5 estereotipos que se usan en dicho diagrama: ejecutable, biblioteca, tabla, archivo y documento.

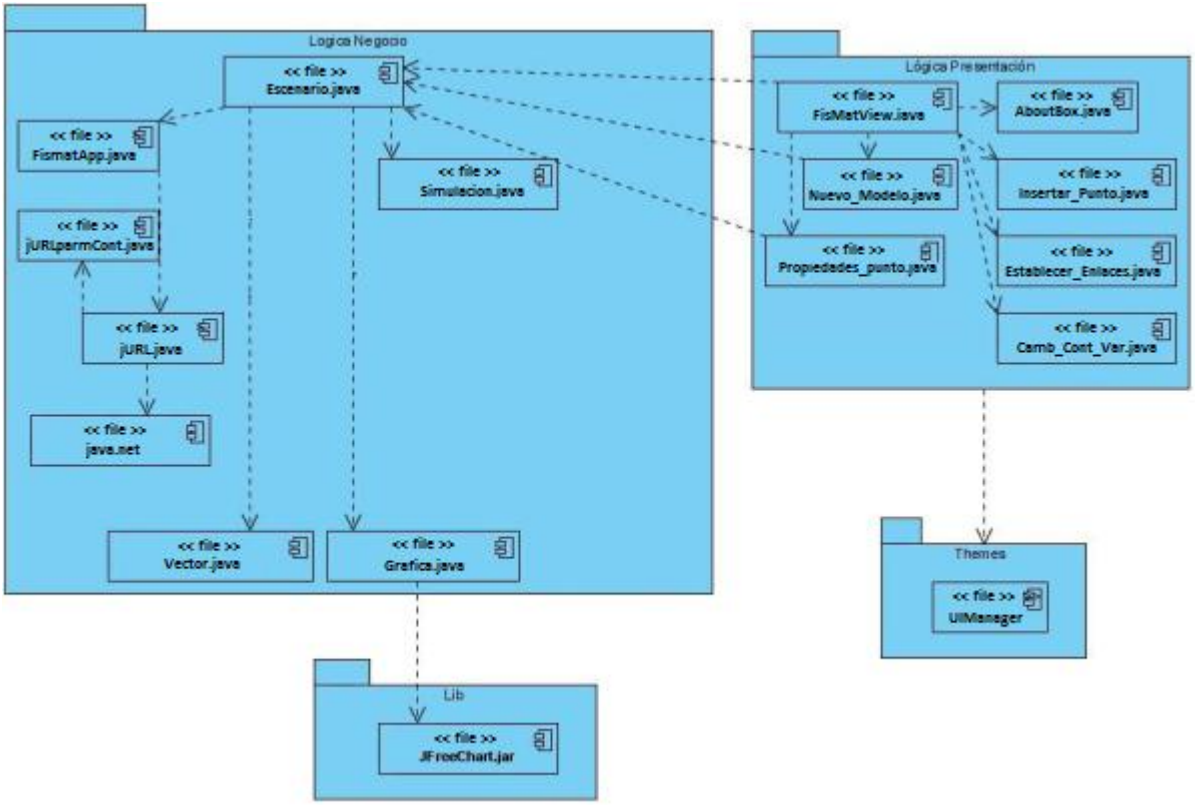


Fig. 25 Diagrama de componentes.

3.3.4 Diagrama de despliegue

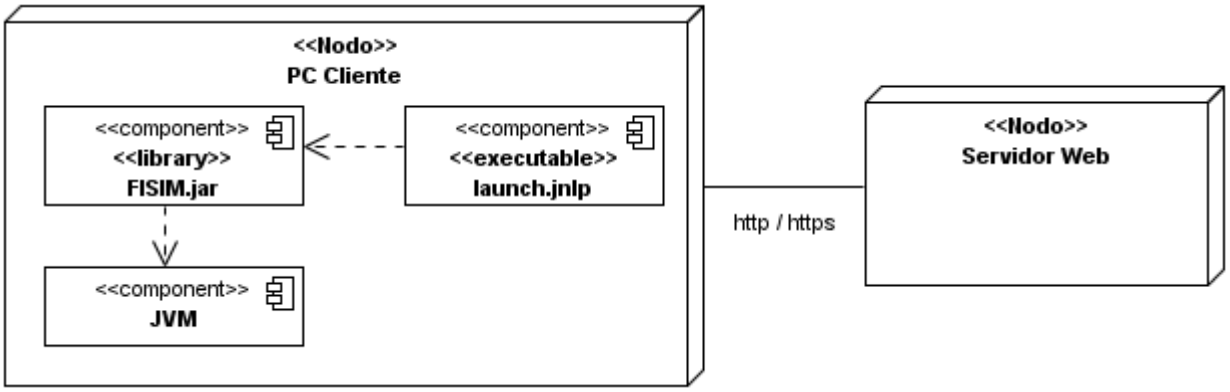


Fig. 26 Diagrama de despliegue.

Los Diagramas de Despliegue muestran las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación.

3.3.5 Diagrama de clases

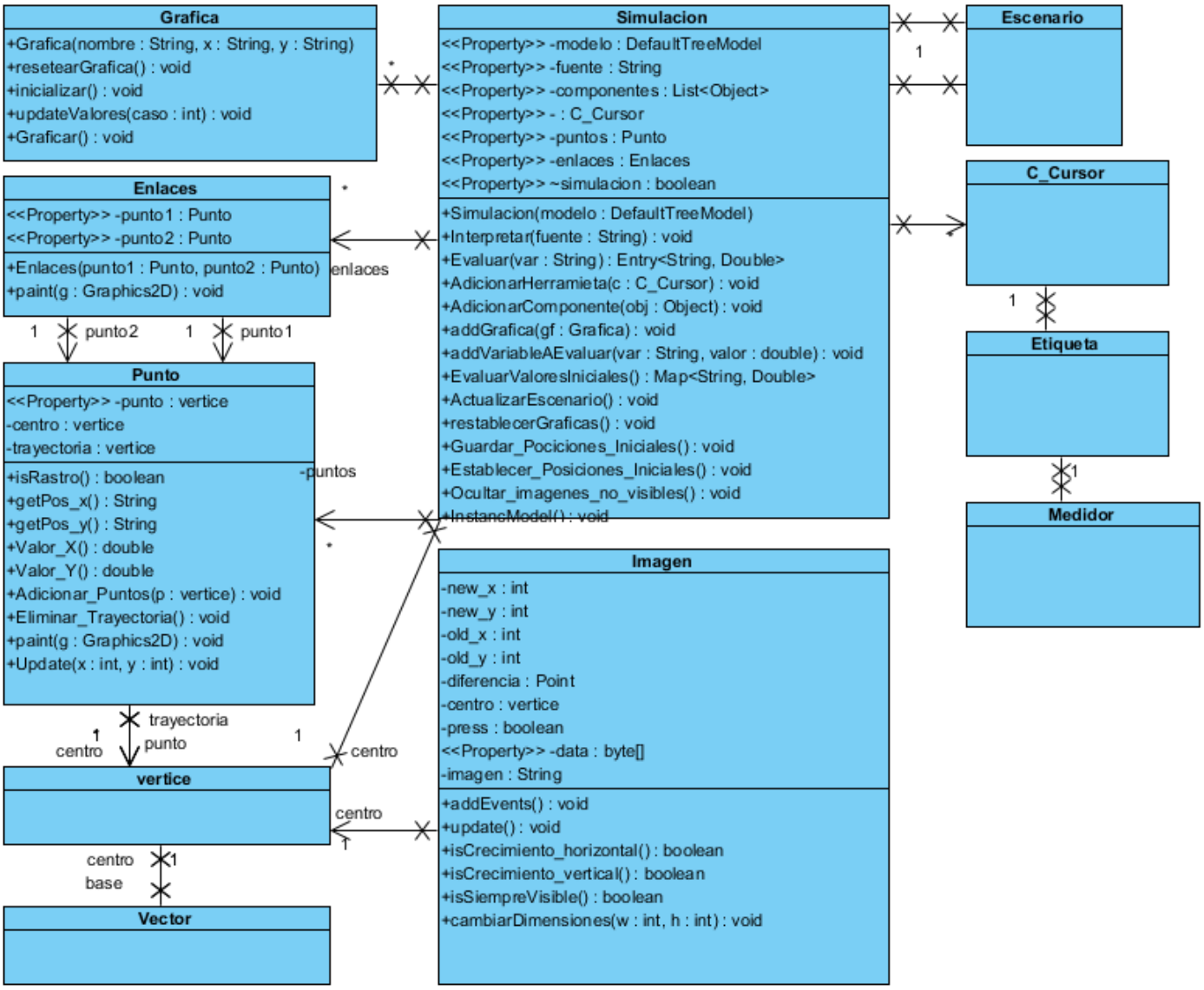


Fig. 27 Diagrama de clases del modelo

3.4 Clases principales. Descripción

Tabla. 3 Clase jURL

Nombre:	jURL	
Descripción:	Clase estática para consumir servicios de PHP	
Tipo de clase:		
Atributo		**Tipo**
-		-
Por cada responsabilidad:		**Tipo**
Nombre:	buscarProxy(String url)	Proxy
Descripción:	Retorna una dirección proxy válida para conectarse a una URL determinada.	
Nombre:	ejecutar(String url, String action, jURLparamContructparametros)	String
Descripción:	Dado la dirección URL, el servicio que se desea consumir y los parámetros requeridos por el servicio, retorna la respuesta del servicio.	

Tabla. 4 Clase jURLparamContruct.

Nombre:	jURLparamContruct	
Descripción:	Clase para generar la cadena de parámetros necesaria a utilizar por la clase jURL.	
Tipo de clase:		
Atributo		**Tipo**
-cadenaParametros		StringBuilder
Por cada responsabilidad:		**Tipo**
Nombre:	addParam(String nombParam, String value)	void
Descripción:	Adiciona a la cadena "*cadenaParametros*" un nuevo parámetro y su respectivo valor.	
Nombre:	buildParam()	String
Descripción:	Retorna de la cadena de parámetros.	

Tabla. 5 Clase FisMatApp

Nombre:	FisMatApp	
Descripción:	Clase principal del proyecto.	
Tipo de clase:	Controladora	
Atributo		**Tipo**
- StartType		int
- URL		String
- userName		String
- token		String
Por cada responsabilidad:		**Tipo**
Nombre:	startup()	void
Descripción:	Crea y muestra el Frame o ventana principal de la aplicación.	
Nombre:	main(String[] args)	void
Descripción:	Captura los parámetros de la aplicación y le da valores a algunos atributos de la clase.	
Nombre:	encryptMD5(String code)	String
Descripción:	Genera el MD5 correspondiente a una cadera específica.	
Nombre:	main(String[] args)	void
Descripción:	Captura los parámetros de la aplicación y le da valores a algunos atributos de la clase.	

Patrón Singleton en Java:

La utilización de Singleton en Java puede implementarse de varias maneras, pero no se implementa en el desarrollo, si no que se utiliza. En esencia consiste en restringir la creación de objetos pertenecientes a una clase y proporcionar un punto de acceso global a ella.

Para lograr ese objetivo:

- La propia clase es responsable de crear la única instancia.
- Permite el acceso global a dicha instancia mediante un método de clase.
- Declara el constructor de clase como privado para que no sea instanciable directamente.

En versiones de prueba se necesitó controlar la fecha en que determinado usuario accede al sistema para esto fue necesario utilizar la clase *Calendar* que obtiene los datos de fecha y hora del sistema. Para evitar más de una instancia de esta clase, ésta implementa Singleton y como resultado obtener una instancia de esta clase es a través de la instrucción *Calendar.getInstance();*

Tabla. 6 Clase FisMatView

Nombre:	FisMatView	
Descripción:	Clase de tipo FrameView. Actúa como Frame principal de la aplicación	
Tipo de clase:		
Atributo		**Tipo**
- simulación		simulacion
- modelo		DefaultTreeModel
- herramientas		DefaultMutableTreeNode
- componentes		DefaultMutableTreeNode
- graficas		DefaultMutableTreeNode
- componentSelected		Object
- selección		int[]
- fichero		File
- timer		Timer
- escenario		Escenario
- contPass		int
Por cada responsabilidad:		**Tipo**
Nombre:	actualizarVisual()	void
Descripción:	Permite actualizar los componentes visuales.	
Nombre:	startAnimation()	void
Descripción:	comienza la animación de los elementos del escenario	
Nombre:	stopAnimation()	void
Descripción:	Para la animación	
Nombre:	resetAnimation()	void
Descripción:	Se restablecen las posiciones iniciales de una animació	
Nombre:	interpretarActionPerformed(ActionEventev)	void
Descripción:	Llama al intérprete del modelo.	
Nombre:	cargarEscenario(Simulacionsimul, ActionEventevt)	Void
Descripción:	Dada una simulación permite cargar el escenario de esta en el frame.	

Nombre:	getNonLoopbackAddress()	String
Descripción:	Obtiene la dirección IP no Loopback de la PC que se conecta con el servidor web.	
Nombre:	autenticar()	boolean
Descripción:	Sistema de autenticación 1 que permite autenticar un usuario con la plataforma utilizando SSO.	
Nombre:	autenticar(String user, String pass)	Boolean
Descripción:	Sistema de autenticación 2 que permite autenticar un usuario con la plataforma utilizando nombre de usuario y contraseña.	
Nombre:	Guardar(String nombre)	void
Descripción:	Permite guardar una simulación	
Nombre:	GuardarEscenario(String nombre)	void
Descripción:	Permite guardar un escenario	

Tabla. 7 Clase Escenario

Nombre:	Escenario	
Descripción:	Representa el escenario donde se va a llevar a cabo la simulación	
Tipo de clase:	Controladora	
Atributo		**Tipo**
- ancho		int
- alto		int
- contain		boolean
- selección		int[]
- selected		object
- simulacion		Simulacion
- old_point		Point
Por cada responsabilidad:		**Tipo**
Nombre:	paintComponent(Graphics g)	void
Descripción:	Permite pintar en el escenario teniendo en cuenta sus dimensiones.	

El uso del patrón ***Alta cohesión*** proporcionó asignar las responsabilidades evitando que una clase haga cosas que no estén relacionadas entre sí, o muchas responsabilidades en un mismo lugar. El ejemplo más ilustrativo es la clase Escenario. Esta tiene responsabilidades como: *PaintComponent*, *SetEnlace*, *SetCentro, mouseMoved,* etc., pero podría tener otras responsabilidades como: *Establecer_Posiciones_Iniciales, getGraficas, etc.* En cambio haciendo uso de este patrón se procedió a ubicar estas responsabilidades en la clase *Simulacion.* En sentido general la clase *Escenario* se centra en

lo que debe hacer el sistema ante la interacción del usuario con la vista y la clase *Simulacion* en las relaciones entre los elementos del escenario.

Tabla. 8 Clase Simulación

Nombre:	Simulacion	
Descripción:	Clase que representa una simulación	
Tipo de clase:	Entidad	
Atributo		**Tipo**
- model		Model
- modelo		DefaultTreeModel
- fuente		String
- componentes		List<Object>
- herramientas		List<C_Cursor>
- puntos		List<Punto>
- enlaces		List<Enlaces>
- simulacion		boolean
- centro		vertice
- graficas		List<Grafica>
- puntos_iniciales		List<Point>
- variables		Set<String>
- delay		int
Por cada responsabilidad:		**Tipo**
Nombre:	getDelay()	Int
Descripción:	Obtiene el retraso en milisegundos de una simulación	
Nombre:	SetDelay(int delay)	void
Descripción:	Establece el retraso en milisegundos de una simulación	
Nombre:	Interpretar(String fuente)	void
Descripción:	Dado el código del modelo matemática lo interpreta a través de un parser	
Nombre:	VariablesDeclaradasDisponibles()	Set<String>
Descripción:	Retorna las variables que se encuentran disponibles	
Nombre:	ActualizarEscenario()	void
Descripción:	Actualiza todos los componentes del escenario	
Nombre:	restablecerGraficas()	void
Descripción:	Restablece el estatus por defecto de una gráfica	
Nombre:	Guardar_Pociciones_Iniciales()	void
Descripción:	Guarda las posiciones iniciales para poder llevar el escenario a su estado inicial luego de ejecutar una simulación.	
Nombre:	Establecer_Posiciones_Iniciales()	void
Descripción:	Establece las posiciones iniciales que con anterioridad se guardaron	

El patrón ***Controlador*** forma parte de la arquitectura del sistema al implementar MVC. Se utiliza al encapsular en una clase una interfaz de comunicación que implementa el algoritmo en su trasfondo. En esencia se separó la lógica de negocios de la capa de presentación aumentando la reutilización de código y a la vez tener un mayor control. La clase *Simulacion* es la encargada de esta función.

La utilización del patrón ***Experto*** **y Creador** son inherentes a las buenas prácticas de la programación orientada a objetos. Su utilización radicó en ubicar las responsabilidades en las clases que contienen la información así como crear instancias de clases exclusivamente cuando estamos en presencia de una agregación, dependencias, necesidad de almacenamiento, etc. Así es el caso, de la clase *Simulacion* que contiene los elementos de una simulación como: lista de puntos, gráficas, modelo matemático, etc. y tiene la responsabilidad por lo tanto de: interpretar un modelo matemático, Adicionar gráficas, puntos y componentes y las relaciones entre ellos. De igual manera esta clase es la encargada de instanciar las clases *componente, punto, modelo,* entre otras ya que estas tienen una relación de composición con la clase *Simulacion*.

Tabla. 9 Clase Imagen

Nombre:	Imagen	
Descripción:	Representa una imagen del escenario	
Tipo de clase:	Entidad	
Atributo		**Tipo**
- new_x		int
- new_y		int
- old_x		int
-old_y		int
- diferencia		Point
- centro		vertice
- data		Byte[]
- imagen		String
- valor		Double
- signo		String
- seleccion		int[ ]
- crecimiento_horizontal		boolean
- crecimiento_vertical		boolean
Por cada responsabilidad:		**Tipo**
Nombre:	cambiarDimensiones(int w, int h)	void
Descripción:	Dado el alto y el largo permite cambiar las dimensiones de la imagen.	
Nombre:	update()	void
Descripción:	Actualiza la imagen en su totalidad, incluyendo sus dimensiones.	

Conclusiones del capítulo

Las bibliotecas y clases utilizadas propiciaron un desarrollo rápido y de calidad profesional.

Se generaron los artefactos del flujo de trabajo implementación ilustrando la robustez de la arquitectura, la extensibilidad del código y constituye además una herramienta de comunicación imprescindible para el resto del equipo de desarrollo y futuras versiones de la aplicación.

Capítulo 4: Validación de la solución propuesta

La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión de las especificaciones del diseño y de la codificación. Desde hace algunos años, han surgido y evolucionado una variedad de métodos para realizar pruebas de software. La alternativa más significativa en este contexto son las pruebas de caja negra y caja blanca.

Es importante realizar diferentes tipos de prueba. Las pruebas que se basan en el análisis del cumplimiento de las descripciones de los Casos de Uso son bastante eficientes para validar la solución, pero no demuestran totalmente la ausencia de defectos, ni la correcta integración de la solución con el sistema y la arquitectura ya implementada. Las pruebas realizadas al subsistema FISIM fueron: funcionalidad, seguridad, disponibilidad, rendimiento, stress y usabilidad.

4.1 Pruebas unitarias y método de prueba de caja blanca

En la prueba de la caja blanca se comprueba los caminos lógicos del software proponiendo casos de prueba que se ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coinciden con el esperado o mencionado.
A nivel de Unidad (Clase o método) esta prueba la hace el mismo programador y consiste en ir ejecutando el código para convencerse de que "básicamente, funciona". Esta prueba suele consistir en pequeños ejemplos que se intentan ejecutar. La automatización de estas pruebas es fundamental para agilizar el proceso, por tal motivo es utilizado el framework JUnit. Este brinda al programador una serie de clases, interfaces y herramientas que este puede emplear para implementar su patrón de prueba.

Resumen de los resultados

- Se pudo constatar que el funcionamiento de los métodos para interconectar el subsistema con la plataforma ZERA retornaban el valor esperado. Esta prueba arrojó que el método *getNonLoopBackAddress()* retornaba en dependencia del sistema operativo una interfaz de red arbitraria. El problema fue resuelto satisfactoriamente.
- Se comprobó que algunos métodos no validaban apropiadamente un conjunto de excepciones de cálculos matemáticos por lo que se le realizó los arreglos pertinentes.
- Como resultado de estas pruebas se comprobó que los métodos y clases funcionan de forma independiente de una manera correcta y eficiente.

4.2 Pruebas de Integración

Las pruebas de integración se llevan a cabo durante la construcción del sistema, involucran a un número creciente de módulos y terminan probando el sistema como conjunto. Las pruebas funcionales de integración son similares a las pruebas de caja negra. Aquí se trata de encontrar fallos en la respuesta de un módulo cuando su operación depende de los servicios prestados por otro(s) módulo(s).

Resumen de los resultados

- La integración entre las clases del intérprete, las de fabricación pura como *Network*, la controladora, el modelo y la vista funcionan de forma monolítica y robusta.

4.3 Pruebas de Sistema

Estas pruebas verifican que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total. Para este nivel de pruebas se utilizan principalmente las técnicas de pruebas de caja negra. Son realizadas por el equipo de calidad interna del proyecto y el equipo del proyecto de Calidad UCI estas pruebas son realizadas basándose en los casos de pruebas diseñados debidamente para la solución propuesta de los cuales se muestra a continuación el principal.

Método de prueba por caja negra

Las pruebas de **caja negra**. Consiste en ir probando uno a uno los diferentes módulos que constituyen una aplicación. Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. O sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce el resultado esperado, así como que la integridad de la información externa se mantiene. Esta prueba examina algunos aspectos del modelo, fundamentalmente del sistema, sin tener mucho en cuenta la estructura interna del software. Se centran principalmente en los requisitos funcionales del software. Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos.

4.3.1 Resumen del diseño de casos de prueba

Tabla. 10 Resumen de caso de prueba del CU Gestionar modelo matemático

Nombre del caso de uso:	**Gestionar modelo matemático**	
Nombre del Caso de prueba:	Crear modelo matemático	
Entrada	**Descripción**	**Condiciones**
Selecciona la opción de crear un modelo matemático.	- Selecciona la cantidad de casos, permite introducir los datos para cada caso y los adiciona al modelo matemático. - El sistema emite un mensaje para que llene los campos obligatorios. - Permite introducir fórmulas al modelo. - Permite abrir, guardar o no un determinado modelo matemático existente.	- Los datos introducidos deben ser válidos. - No se pueden adicionar más de 3 casos. - No se pueden dejar campos vacíos.
Nombre del Caso de prueba:	Abrir modelo matemático	
Selecciona la opción abrir un modelo matemático.	- Permite especificar la dirección donde se encuentra el modelo, abrirlo o cancelar la operación - Muestra los datos del modelo y permite su modificación, interpretar el modelo o crear un nuevo modelo.	- No se puede abrir archivos que no sean válidos.
Nombre del Caso de prueba:	Modificar modelo matemático	
Selecciona la opción de Modificar modelo	- Muestra los datos del modelo y permite modificar los valores de las funciones, variables, casos, etc. - permite actualizar los datos del	- Los datos que se modifiquen deben ser válidos. - No se pueden dejar campos vacíos.

	modelo, interpretarlos y guardarlos.	
Nombre del Caso de prueba:	Interpretar modelo matemático	
Selecciona la opción de interpretar el modelo matemático.	- Muestra mensaje de confirmación de interpretación del modelo o de errores en el código del modelo matemático.	- Los datos que se modifiquen deben ser válidos. - No se pueden dejar campos vacíos.

Tabla. 11 Resumen de caso de prueba del CU Gestionar elemento.

Nombre del caso de uso:	**Gestionar elemento**	
Nombre del Caso de prueba:	Gestionar elemento	
Entrada	**Descripción**	**Condiciones**
Selecciona la opción de realizar una acción sobre un elemento.	- Brinda la posibilidad de incluir, eliminar y modificar un elemento. - Ver sección y datos del elemento, así como modificar estos datos. - Permite dar la posibilidad de enlazar dos puntos creando una recta entre ellos.	- Los datos que se modifiquen deben ser válidos. - No se pueden dejar campos vacíos.
Nombre del Caso de prueba:	Ver datos del elemento	
Selecciona la opción de ver los datos de un elemento.	- Muestra los datos del elemento seleccionado en dependencia del tipo de elemento que sea. - Permite modificar los datos de un elemento.	- Los datos que se modifiquen deben ser válidos. - No se pueden dejar campos vacíos.
Nombre del Caso de prueba:	Modificar datos de un elemento	
Selecciona la opción de modificar los datos de un elemento.	- Muestra en el administrador de componentes los datos del elemento seleccionado. - Permite modificar los datos en	- Los datos que se modifiquen deben ser válidos. - No se pueden dejar campos vacíos.

	dependencia del tipo de elemento que sea. - Actualiza los datos del elemento. - Permite redefinir el tipo de unidad de medida.	
Nombre del Caso de prueba:	Eliminar elemento	
Selecciona la opción de eliminar un elemento.	- Muestra en el administrador de componentes un resumen de los datos del elemento seleccionado. - Muestra un mensaje de información y permite: Aceptar o Cancelar. - Oculta el elemento. Actualizando los datos. Regresa a la vista anterior.	- Los datos que se modifiquen deben ser válidos. - No se pueden dejar campos vacíos.

Tabla. 12 Resumen de caso de prueba del CU Adicionar elemento.

Nombre del caso de uso:	**Adicionar elemento**	
Nombre del Caso de prueba:	Adicionar elemento	
Entrada	**Descripción**	**Condiciones**
Selecciona la opción de adicionar un elemento.	- Brinda la posibilidad de: Adicionar una imagen, cursos, etiqueta, gráfica, vector, etc. -Permite seleccionar el componente que se desea adicionar. - Permite colocar un componente en el escenario una vez seleccionado. -Permite establecer valores.	- Para insertar un componente debe estar en el área del escenario. - Los datos que se modifiquen deben ser válidos. - No se pueden dejar campos vacíos.

Nombre del Caso de prueba:	Adicionar cursor	
Selecciona la opción que le permite adicionar un cursor	Permite seleccionar: Orientación del cursor. Para la animación la variable que va a representar el cursor. Brinda la posibilidad de introducir: Las variables iniciales: Posición X, Y. Mínimo, Máximo	- Para insertar un componente debe estar en el área del escenario. - Los datos que se modifiquen deben ser válidos. - No se pueden dejar campos vacíos.
Nombre del Caso de prueba:	Adicionar etiqueta	
	Permite seleccionar: Si la etiqueta va a ser dinámica. Brinda la posibilidad de introducir: Las variables iniciales: Posición X, Y. Texto que identifique la etiqueta.	- Para insertar un componente debe estar en el área del escenario. - Los datos que se modifiquen deben ser válidos. - No se pueden dejar campos vacíos.
Nombre del Caso de prueba:	Adicionar punto	
	Permite seleccionar: Marcar trayectoria. Los datos de la animación: Posición X. Posición Y. Brinda la posibilidad de introducir: Las variables iniciales: Posición X, Y. Ancho. Permite además: Aceptar y Cancelar.	- Para insertar un componente debe estar en el área del escenario. - Los datos que se modifiquen deben ser válidos. - No se pueden dejar campos vacíos.
Nombre del Caso de prueba:	Adicionar gráfica	
	Permite seleccionar: Eje X, Y. Brinda la posibilidad de introducir: Nombre. Permite además:	- Para insertar un componente debe estar en el área del escenario.

	Aceptar. Cancelar.	- Los datos que se modifiquen deben ser válidos. - No se pueden dejar campos vacíos.
Nombre del Caso de prueba:	Adicionar medidor	
	Permite seleccionar: Para la animación la variable que va a representar el medidor. Brinda la posibilidad de introducir: Las variables iniciales: Posición X., Y. Cantidad de unidades. Unidad de medida. Además permite: Aceptar, Cancelar.	- Para insertar un componente debe estar en el área del escenario. - Los datos que se modifiquen deben ser válidos. - No se pueden dejar campos vacíos.
Nombre del Caso de prueba:	Adicionar vector	
	Permite seleccionar: Mostrar los componentes X y Y del vector. Los datos de la animación: Posición X, Y. Componente X, Y. Brinda la posibilidad de introducir: variables iniciales, Posición X.	- Para insertar un componente debe estar en el área del escenario. - Los datos que se modifiquen deben ser válidos. - No se pueden dejar campos vacíos.

Análisis de los resultados obtenidos en las pruebas de caja negra

- Como resultado de las pruebas realizadas en la etapa de implementación, la cantidad de No Conformidades (NC) detectadas durante el desarrollo de las pruebas de caja negra fue reducida.
- En la primera iteración de las pruebas se detectó que la interfaz del sistema sobrepasaba el marco de la pantalla. Este error fue corregido teniendo en cuenta la resolución de pantalla de las PC clientes.
- En esta misma iteración se constató que la interfaz de la aplicación se comportaba diferente en dependencia del sistema operativo en el que se ejecutara. Esta NC fue solucionada haciendo uso

de la IUManager [14]de Java que permite hacer configuraciones en tiempo de ejecución del lookAndField [15]de la aplicación.

- Se detectaron además 2 faltas ortográficas en los mensajes al usuario.
- Una vez corregidas estas NC se realizó una segunda iteración da las pruebas donde se corroboró la calidad de la aplicación y la respuesta a las NC detectadas.

Conclusiones del capítulo.

El empleo de diferentes tipos y métodos de prueba garantiza parámetros elevados de calidad y por lo tanto robustez y eficiencia de la solución. Las dos iteraciones de pruebas constataron que la aplicación cumple con los requisitos funcionales definidos en los flujos de trabajo de análisis y diseño y está lista para entrar a las pruebas de aceptación por parte del cliente.

[14] Administrador de interfaz de usuario de Java.
[15] Forma que tienen las interfaces de java de comportarse en forma de temas o en dependencia del sistema operativo.

Conclusiones

Al término de la presente investigación para la implementación del subsistema FISIM: simulador físico – matemático integrado a la plataforma de gestión del aprendizaje ZERA, se concluye que:

- Java es la tecnología más usada en el desarrollo de simuladores para la educación y sus facilidades permitieron implementar una aplicación robusta, eficiente y desplegada bajo la tecnología JNLP de probada eficacia en aplicaciones distribuidas.
- Se obtuvo el simulador físico – matemático FISIM integrado a la plataforma de gestión del aprendizaje ZERA, siendo esta integración, el valor añadido y aporte más significativo de la solución con respecto a sus similares en el mundo, contribuyendo a la interacción estudiante-profesor con la generación de trazas y el trabajo con evidencias.
- El empleo de diferentes tipos y métodos de prueba garantizan el buen funcionamiento y la calidad del producto final. De esta manera se minimiza la probabilidad detectar No Conformidades en las pruebas de aceptación por parte del cliente.

.

Recomendaciones

Para futuras versiones del simulador se recomienda:

- Mejorar el mecanismo para lograr la persistencia de las simulaciones teniendo en cuenta futuras versiones del sistema.
- Incrementar el nivel de integración del subsistema FISIM a la plataforma ZERA con el fin de mejorar la trazabilidad de los usuarios, agregar la asignación de tareas y la creación de objetos de aprendizaje.

A estudiantes y profesores (Recomendaciones de uso):

- Explotar al máximo el estudio de las variables del modelo con el uso de los componentes: Medidor, gráfica, vector y etiqueta.
- Hacer uso del intérprete de modelos matemáticos en todo su potencial, con el fin de diseñar de una manera generalizada las simulaciones.

Referencias bibliográficas

1. **Himmelblau, David M. y Bischoff, Kenneth B.** *Análisis y simulación de procesos.* Barcelona: Reverté, S.A., 1976. ISBN: 84-291-7235-1.
2. **Naylor, Thomas H., et al., et al.** *Computer simulation techniques.* New York : Wiley and Sons, 1966. QA 76 .5 N3.
3. **Shannon, Robert E. and Johannes, James D.** *Systems Simulation: The Art and Science.* s.l. : OAI Repositorio (Sistema LIBRUM), SERBIULA - Universidad de Los Andes, Venezuela, 1975. ISSN 0018-9472.
4. **Ministerio de Educación Nacional de Colombia.** Portal Colombia Aprendre. [En línea] Ministerio de Educación Nacional. República de Colombia. [Citado el: 24 de marzo del 2011.] http://www.colombiaaprende.edu.co/html/directivos/1598/article-75224.html
5. **Agrega.** ProyectoAgrega. [En línea] [Citado el: 24 de marzo de 2011.] http://www.proyectoagrega.es/blog/Agrega
6. **Ministerio de Educación Nacional de Colombia.** Portal Colombia Aprendre. [En línea] Ministerio de Educación Nacional de Colombia. [Citado el: 24 de marzo de 2011.] http://www.colombiaaprende.edu.co/html/directivos/1598/article-88892.html
7. **Colectivo de Autores.** *Pedagogía a tu alcance.* CD-ROM Colección Futuro.
8. **Orjuela, Harley J. y Hurtado, Alejandro.** *Perfeccionamiento de un nuevo simulador interactivo, bajo software libre gnu/Linux, xomo desarrollo de una nueva herramienta en la enseñanza y aprendizaje de la física.* Universidad Distrital Francisco José de Caldas, Colombia, 2009. ISBN: 1870-9095
9. **Ibáñez, Martín.** *La Educación a paso de Yenka.* 2006. ISBN: 1130-0426
10. **Sarasa, Antonio. y Canabal, Jose M.** *Agrega, un proyecto de software libre.* 2009.
11. **Knowlton, Jim.** *Python.*tr: Fernández Vélez, María Jesús (1 edición) 2009. ISBN: 978-84-415-2513-9
12. **Flanagan, David.** *JavaScript:The Definitive Guide (4 ª Edición edición).* 2002. ISBN 0-596-00048-0.
13. **García, Adolfo A.** *Inttroducción a JavaFX (I)*. ISBN: 1134-4792
14. **Jordan, Lucas L.** *JavaFX™ Special Effects.* s.l. : Apress, 2009. ISBN 978-1-4302-2623-9.
15. **Arnold, Ken.** *El lenguaje de programación Java.* Madrid, 1997**.** ISBN: 84-7829-010-9
16. **Brittain, Jason y Darwin, Ian F.** *ElTomcat 6.0. La guía definitiva***.** España. 2008. ISBN: 9788441524316.

17. **Lindholm, Tim y Yellin, Frank.** *Java Virtual Machine Specification.* s.l. : Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA ©1999, 1999. ISBN 0201432943.
18. **Moldes, Javier**. *Java 2 (J2SE 1.4).* Anaya-Multimedia Interactiva. España, 2003. ISBN: 84-415-1552-2
19. **www.java.com.** [En línea] miércoles de junio de 2011. [Citado el: 08 de 05 de 2011.] http://www.google.com.cu/url?sa=t&source=web&cd=1&ved=0CCQQFjAA&url=http%3A%2F%2Fwww.java.com%2Fes%2Fdownload%2Ffaq%2Fjava_webstart.xml&rct=j&q=java%20web%20start&ei=VgnwTYawM6Hg0QGBiun-DA&usg=AFQjCNFal5JcYuIn9-cEklFqzbBIn0bwLw
20. Documentación oficial de javaws. [En línea] 15 de 05 de 2011. http://java.sun.com/j2se/1.5.0/docs/guide/javaws/developersguide/javaws.html
21. **Cauldwell, Patrick.** *Servicios web XML.* Madrid, España : Anaya : s.n., 2002. 84-415-1363-5.
22. **Henning, Michi.** *Component Technologies.* New York, NY, USA : s.n., 2006, Vol. 4. 10.1145/1142031.1142044.
23. **Grimes, Richard y Grimes, Dr Richard.** *Professional Dcom Programming .* s.l. : Wrox Press Ltd. Birmingham, UK, UK ©1997 , 1997. 186100060X .
24. **www.java.com.** java.net (Java 2 Platform SE v1.4.2). [En línea] 18 de octubre de 2010. [Citado el: 17 de mayo de 2011.] http://download.oracle.com/javase/1.4.2/docs/api/java/net/package-summary.html.
25. **Booch, Jacobson, y Rumbaugh, J.** *Lenguaje Unificado de Modelado. Guía de Usuario.* ISBN: 0-201-57168-4.
26. **Davis, William y Mata, Antonio.** *Herramientas Case.* España. 1992. ISBN: 84-283-1927-8
27. Visual Paradigm for UML Product Home. [En línea] 14 de 04 de 2011. [Citado el: 14 de 04 de 2011.] http://www.visual-paradigm.com/product/vpuml/.
28. **Eckstein, Robert.** Java SE Application Design With MVC. [En línea] marzo de 2007. [Citado el: 10 de 04 de 2011.] http://www.oracle.com/technetwork/articles/javase/mvc-136693.html.
29. **Larman, Craig. 2004.** *UML y Patrones de Diseño. Introducción al análisis y diseño orientado a objetos.* La Habana: Félix Varela, 2004.