Universidad de las Ciencias Informáticas

Facultad 3

"Análisis y Diseño de un Simulador de Programación de Grafos de Precedencia para el Laboratorio Virtual de Sistemas Operativos"

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autora: Dayana Alfonso Hidalgo

Tutor: Ing. Carlos Y. Hidalgo García

Ciudad de la Habana, Cuba

Junio, 2011

Declaración de Autoría

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y se reconoce a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____

Firma del autor

(Dayana Alfonso Hidalgo)

_____

Firma del tutor

(Carlos Y. Hidalgo García)

"Y vegetar no es vivir. Vivir es tener algo que hacer. Vivir es tener una meta, un objetivo, una tarea, una obra a la cual dedicar el tiempo, la energía, y dedicar la vida. Y eso es lo que tienen ustedes, es lo que tienen los jóvenes universitarios en nuestro país."

Fidel Castro Ruz

Agradecimientos

Quiero agradecerle a mi familia por apoyarme y por estar siempre a mi lado en los momentos que más los he necesitado, en especial a mi tio Ángel pues sin él nunca hubiera podido alcanzar esta meta.

A mis padres por ser tan buenos conmigo, por ayudarme en todos estos años, por darme de ellos lo mejor. A los dos les digo que llegó mi momento de ayudarlos a ellos. A mi papá que lo amo con la vida y aunque no pudo estar en estos momentos conmigo decirle que siempre tendré presente lo que me dice porque lo nuestro es un contrato para toda la vida; a mi madre atenta , preocupada y amada agradecerle por estar aquí hoy a mi lado como siempre, por haber estado presente en todos los momentos más importantes de mi vida, por malcriarme y contribuir tan bien a lo hoy que soy.

A mi hermana Danay por ser más que mi hermana , por ser mi amiga y por darme la oportunidad de tener esa sobrina tan bella que tengo y adoro: Analia.

A mis suegros a los cuales admiro y quiero mucho.

A mis tios Ale y Eduardo y a mi abuela Olga por ser siempre muy preocupados: los quiero mucho.

A mis primas Amarelis, Taidy y Lianet por compartir conmigo desde pequeña.

A mi novio por ser el hombre más maravilloso que he conocido, por ser siempre mi apoyo todo este tiempo , aunque ahora no se encuentre presente le llegarán mis agradecimientos y mi eterna gratitud porque sin el hoy no me encontraría en esta sala.

Al tribunal por lo que pude aprender de ellos.

A mi tutor por ser ese amigo que te ayuda, que te da fuerzas, que te enseña y da confianza en ti misma.

A mis amigos: Alejandro, Laritza, Janny, Dainelis, Raydel, Yanelis y Aylin.

Y de todos mis amigos a uno en especial: Yaidel, que es mi hermano y esa palabra es grande.

A todos ustedes… Muchas Gracias por contribuir con su granito de arena a que un día como hoy pudiera ver mis sueños realizados.

Dedicatoria

A mi papá y a mi novio por ser los hombres a los que más amo en mi vida y de los que he aprendido mucho.

A mi tutor Carlos y a mi amigo Yaidel sin ellos no lo hubiera logrado.

A nuestro Comandante en Jefe Fidel por pensar y hacer realidad esta gran oportunidad para todos nosotros.

A la Revolución por darnos todo y hacer de nosotros mejores seres humanos.

Resumen

Teniendo en cuenta la necesidad del aumento de la calidad en la educación cubana, en la Universidad de las Ciencias Informáticas (UCI) se ha implementado un modelo de integración donde los procesos de Formación – Producción e Investigación se funden en uno solo para buscar aumentar la calidad del egresado en el que se destaca la utilización de recursos didácticos y el perfeccionamiento del plan de estudios.

Como objeto de nuestra investigación, se tomó la asignatura de Sistemas Operativos para la que se está desarrollando un Laboratorio Virtual con el objetivo de contribuir a la incorporación de la asignatura al modelo de formación y dentro del que se encuentra un subsistema de Simuladores, para el se realiza la modelación del simulador de programación de grafos de precedencia utilizando la estructura Parbegin/Parend mediante el uso de las primitivas P() y V() con semáforos.

En el desarrollo de la propuesta se utilizó la metodología de desarrollo RUP, como lenguajes de modelado UML y BPMN y Visual Paradigm 4.1 como herramienta CASE para el modelado, lo que permitió obtener los artefactos necesarios para su posterior implementación.

Para la validación de los requisitos se aplicó la técnica de prototipos, se aplicaron métricas de calidad de la especificación de requisitos y de casos de usos además de métricas para la validación del diseño, obteniéndose en todos los casos resultados satisfactorios.

Palabras Claves: aplicaciones educativas, simuladores, laboratorio virtual.

Índice de Contenido

Contenido

Índice de Contenido

Índice de Figuras

Introducción

Introducción

La informática es una de las ciencias que se encuentra en completa evolución y está presente en la mayoría de los sectores sociales del mundo. Cuba, desde hace algunos años, se ha unido al desarrollo informático mundial. Para alcanzarlo se han trazado en el país diversas estrategias que ayudan a elevar la cultura y conocimiento de esta ciencia. Algunas de estas estrategias son la creación de los Joven Club de Computación y el surgimiento de una universidad de nuevo tipo llamada Universidad de las Ciencias Informáticas (UCI).

La UCI se ha visto inmersa en un aumento de los compromisos y la asignación de tareas, ante la inmediata necesidad de convertirse en el motor impulsor del desarrollo de software en Cuba. A partir de una creciente vinculación de profesores y estudiantes a las labores productivas, se concibió un nuevo modelo de formación siguiendo los siguientes principios: centrado en el aprendizaje, establecimiento de un ciclo de formación básico y otro profesional en el cual tienen un uso protagónico los Entornos Virtuales de Aprendizaje (EVA) y la incorporación de los estudiantes, a partir del segundo semestre de 3er año, a proyectos de desarrollo donde tienen la oportunidad de desarrollar habilidades desde el punto de vista profesional en dependencia de los objetivos a alcanzar en cada uno de los años.

El uso del EVA en la UCI para el proceso de enseñanza - aprendizaje es creciente, en el que se destaca la utilización de recursos didácticos para llevar a cabo el proceso docente – educativo (PEA) en las distintas materias. En tal sentido es importante destacar el rol que desempeñan los laboratorios virtuales, siendo una significativa herramienta de apoyo al trabajo docente, tanto para el profesor como para el estudiante.

La asignatura Sistema Operativo es fundamental dentro del plan de estudio del 3er año de la carrera de Ingeniería en Ciencias Informáticas, la cual está dividida en tres temas fundamentales que comprenden los principales aspectos referentes al diseño y construcción de los mismos como son: proceso, memoria y entrada / salida de información.

Se ha podido observar que la asignatura en la actualidad carece de una fuerte integración con el desarrollo de las Tecnologías de la Informática y las Comunicaciones (TIC), lo que ha llevado a que las actividades sean en su mayoría presenciales, entrando en contradicción con el modelo de formación propuesto donde la semipresencialidad y la amplia y adecuada utilización de las TICs son los componentes principales para lograr la integración Formación - Producción - Investigación, y donde el estudiante debe ser capaz de realizar un autoestudio y ser el protagonista de su propio proceso de aprendizaje.

Específicamente, dentro del tema dedicado a procesos, se encuentra un contenido que por requerir de un análisis lógico y de un cierto grado de abstracción le resulta difícil al estudiante poder adquirir las habilidades necesarias para poder desarrollarlo, tal es el caso de la programación de grafos de precedencia utilizando la estructura Parbegin/Parend mediante el uso de las primitivas P() y V() con semáforos.

Dentro de la concepción del Laboratorio Virtual de Sistemas Operativos y como parte del subsistema de simuladores este contenido no se encuentra reflejado, por lo que el estudiante carece de un medio

didáctico que le permita poner en práctica los elementos impartidos en clases por el profesor y por ende les resulta engorroso el proceso de aprendizaje del tema

Los medios didácticos con que se cuenta como apoyo al proceso de enseñanza – aprendizaje del tema de procesos y específicamente lo referido a la programación de grafos de precedencia utilizando la estructura Parbegin/Parend mediante el uso de semáforos en la asignatura de Sistemas Operativos, no son suficientes para permitir que los estudiantes adquieran los conocimientos necesarios desde el punto de vista teórico, por lo que se requiere de la obtención de los artefactos necesarios que permitan la posterior implementación de un simulador, incrementando así los medios didácticos con que cuenta el estudiante para contribuir a la comprensión del tema correspondiente y que estará insertado dentro del subsistema de simuladores del Laboratorio Virtual de Sistemas Operativos.

Bajo estas condiciones es identificado el siguiente **problema de la investigación**: ¿Cómo lograr un entendimiento entre clientes y desarrolladores respecto a los temas referidos a la programación de grafos de precedencia utilizando la estructura Parbegin/Parend mediante el uso de semáforos para el Laboratorio Virtual de Sistemas Operativos, que permita la posterior implementación de un simulador?. Este problema se enmarca en el **objeto de estudio**: Desarrollo de software.

En este sentido la investigación que se presenta tiene como **objetivo general**: Realizar el Análisis y Diseño de un simulador de programación de grafos de precedencia utilizando la estructura Parbegin/Parend mediante el uso de semáforos para el Laboratorio Virtual de Sistemas Operativos y cuyo **campo de acción** es: Análisis y diseño de simuladores para la asignatura de Sistemas Operativos.

Se define como **idea a defender:** Realizando el Análisis y Diseño de los temas referidos a la programación de grafos de precedencia utilizando la estructura Parbegin/Parend mediante el uso de semáforos para el Laboratorio Virtual de Sistemas Operativos se logrará implementar el simulador correspondiente. Para el cumplimiento exitoso del objetivo general se han definido como **Objetivos específicos:**

- Elaborar el marco teórico de la investigación.
- Realizar los modelos de análisis y diseño.
- Validar los artefactos obtenidos.

Métodos de investigación:

Métodos teóricos:

- Análisis Histórico – Lógico: Para profundizar en los antecedentes de la utilización de las Tecnologías de la Información y las Comunicaciones en los procesos de enseñanza – aprendizaje y sus tendencias actuales.
- Analítico - Sintético: Se utiliza en la revisión bibliográfica, el estudio de reportes e informes sobre el estado de la infraestructura tecnológica de la UCI, la consulta de documentos rectores de la política de la UCI sobre la Informatización, la tendencia actual al aprendizaje autogestionado y la introducción de las TIC en el proceso de enseñanza-aprendizaje.

Métodos empíricos:

Introducción

- Entrevista: Para la recopilación de información especializada o dirigida a directivos, profesores y alumnos que interactúan con las TIC en el proceso de enseñanza – aprendizaje presencial o mixto de la asignatura de Sistemas Operativos I en la Facultad 3 de la Universidad de las Ciencias Informáticas.

- Análisis de las fuentes de información: Para consultar fuentes de información relacionadas con el tema de la investigación en la ingeniería de confiabilidad.

Capítulo 1: Fundamentación Teórica

1.1 Introducción

En el presente capítulo se abordarán, diferentes temas entre los que se encuentran, qué es un laboratorio virtual, así como los distintos tipos que existen y los que son afines a la investigación, también se exponen los beneficios e inconvenientes que estos presentan en conjunto con su función dentro del proceso de enseñanza, conjuntamente con ellos se estudia la simulación por computadora y los beneficios que aportan a los estudiantes. Se realiza además un estudio sobre metodologías de desarrollo, herramientas CASE (por sus siglas en inglés Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) y lenguajes de modelado que se van a utilizar en la realización de este trabajo.

1.2 Laboratorios virtuales

Muchos han sido los autores que han dado su definición de lo que es un Laboratorio Virtual, en lo adelante se citan algunos ejemplos.

Según James P. Vary, un laboratorio virtual es un “espacio electrónico de trabajo concebido para la colaboración y la experimentación a distancia con objeto de investigar o realizar otras actividades creativas, y elaborar y difundir resultados mediante tecnologías difundidas de información y comunicación”. (Vary, 2000)

Julián Monge Nájera define un laboratorio virtual como “simulaciones de prácticas manipulativas que pueden ser hechas por el estudiante lejos de la universidad y el docente”. (Nájera Estrada, et al., 2007)

Después de estudiar las definiciones dada por varios autores, se define que un Laboratorio Virtual no es más que un entorno de simulación y experimentación a distancia para realizar el conjunto de prácticas de laboratorio que son necesarias para poder superar las distintas asignaturas.

Existen diversos tipos de laboratorios virtuales, según la bibliografía consultada, se proponen tres clasificaciones que se mencionan a continuación.

1.2.1 Laboratorios virtuales software.

Los laboratorios virtuales de software, son laboratorios desarrollados como un programa de software independiente destinado a ejecutarse en la máquina del usuario, y cuyo servicio no requiere de un servidor Web. Es el caso de programas con instalación propia, que pueden estar destinados a plataformas Unix, Linux, M.S. Windows e incluso necesitar que otros componentes de software estén instalados previamente, pero que no necesitan los recursos de un servidor determinado para funcionar(como bases de datos o módulos de software de servidor). (Velázquez, 2008)

También determinados laboratorios virtuales pensados inicialmente como aplicaciones Java accesibles a través de un servidor Web se pueden considerar de este tipo si funcionan localmente y no necesitan recursos localizados en un servidor de aplicaciones o web.

1.2.2 Laboratorios virtuales Web.

En contraste con el anterior, este tipo de laboratorio se basa en un software que depende de los recursos de un servidor determinado. Estos recursos pueden ser bases de datos, software que requieren ejecutarse

en su servidor o la exigencia de determinado hardware para ejecutarse. No son programas que un usuario pueda descargar en su equipo para ejecutar localmente de forma independiente. (Nájera Estrada, et al., 2007)

1.2.3 Laboratorios remotos.

Los laboratorios remotos son aquellos que permiten operar remotamente cierto equipamiento, bien sea didáctico como maquetas específicas, o industrial, además de poder ofrecer capacidades de laboratorio virtual. En general, estos requieren de equipos servidores específicos que les den acceso a las máquinas a operar de forma remota, y no pueden ofrecer su funcionalidad ejecutándose de forma local. Otro motivo que los hace dependientes de sus servidores es la habitual gestión de usuarios en el servidor. (Herías, 2003)

En este sentido se va a trabajar con los laboratorios virtuales web a causa de que en la asignatura Sistema Operativo se están dando los primeros pasos en la realización de un laboratorio virtual, el mismo se va a incorporar en el EVA y va a tener integrado el trabajo de diploma realizado que forma parte de otro de sus módulos, con una aplicación de escritorio que es un simulador, facilitando que cualquier profesional o alumno interesado en el tema desarrollado pueda hacer uso del mismo y beneficiarse de sus prestaciones.

1.2.4 Ventajas de los laboratorios virtuales

El uso de laboratorios virtuales tiene sin duda muchos beneficios, permite al estudiante buscar información, en él se pueden relacionar fenómenos con sus consecuencias, se pueden repetir los eventos o fenómenos cuantas veces se requiera. Se incorporan las tecnologías de la información y comunicación en las prácticas educativas y sociales para beneficio de los estudiantes. El aprendizaje está basado en simulaciones. Pero además de esto tiene otras ventajas más significativas como son: (Herreros, et al., 2005)

- Simula situaciones que en la realidad tendrían escasas posibilidades de realizarlas.
- Se convierten en una ayuda interactiva para el aprendizaje de contenidos difíciles de demostrar en la realidad.
- La simulación en el laboratorio virtual, permite obtener una visión más intuitiva de aquellos fenómenos que en su realización manual no aportan suficiente claridad gráfica.
- Es una herramienta de auto aprendizaje.

Acompañado a estos beneficios también se presentan algunos inconvenientes con los cuales se corre el riesgo que el alumno se comporte como un simple espectador. Es preciso que el estudiante realice una actividad ordenada y progresiva, conveniente a alcanzar objetivos básicos concretos, además el alumno no utiliza elementos reales en el laboratorio virtual, lo que provoca una pérdida parcial hacia la visión de la realidad y los resultados son menos atractivos para los estudiantes. (Herreros, et al., 2005)

1.3 Simulación por Computadora

La 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. (Luis R. Izquierdo, 2008)

Capítulo 1: Fundamentación Teórica

La simulación por computadora es de mucha utilidad para el aprendizaje de los estudiantes en la asignatura Sistema Operativo, ya que la integración de un software educativo permitirá la incorporación de la tecnología informática, aportando herramientas que favorezcan el desempeño profesional de los alumnos, podrán experimentar en entornos que representan la realidad, a través de modelos de la misma, pero de una forma más interactiva y constructivista, además de fomentar la creatividad, el aprendizaje por descubrimiento y la enseñanza individualizada. Al mismo tiempo le es de mucha ayuda al estudiante en su proceso de auto aprendizaje porque puede observar paso a paso lo que quiere aprender de una forma más amena y apreciable. (Cárdenas, 2009)

1.3.1 Simuladores en la educación

Los simuladores constituyen un procedimiento, tanto para la formación de conceptos y construcción en general de conocimientos, como para la aplicación de éstos a nuevos contextos a los que, por diversas razones, el estudiante no puede acceder desde el contexto metodológico donde se desarrolla su aprendizaje. (José Manuel Ruiz Gutiérrez, 2000)

Mediante los simuladores se puede por ejemplo, desarrollar experimentos de química en el laboratorio de informática con mayor seguridad, es así como si a un estudiante se le ocurre agregar más de un determinado líquido la explosión que esto cause será una simple "simulación", cuando vaya a realizarlo en la práctica él estará informado de las consecuencias de este proceso.

Característica en la educación:

- Apoyan aprendizaje de tipo experimental y conjetural.
- Permite la ejercitación del aprendizaje.
- Suministran un entorno de aprendizaje abierto basado en modelos reales.
- Alto nivel de interactividad
- Tienen por objeto enseñar un determinado contenido.
- El usuario trata de entender las características de los fenómenos, cómo controlarlos o qué hacer ante diferentes circunstancias.
- Promueven situaciones excitantes o entretenidas que sirven de contexto al aprendizaje de un determinado tema.
- El usuario es un ser activo, convirtiéndose en el constructor de su aprendizaje a partir de su propia experiencia. (José Manuel Ruiz Gutiérrez, 2000)

1.4 Concurrencia de Procesos

1.4.1 Definición de concurrencia

Dos o más procesos decimos que son concurrentes, paralelos, o que se ejecutan concurrentemente, cuando son procesados al mismo tiempo, es decir, que para ejecutar uno de ellos, no hace falta que se haya ejecutado otro. (Tanenbaum, 1997)

Se debe tener en cuenta que mientras un proceso está escribiendo un valor en una variable determinada, puede darse el caso que otro proceso que es concurrente al primero vaya a leer o escribir en esa misma variable, entonces habrá que estudiar el caso en el que un proceso haga una operación sobre una variable (o recurso en general) y otro proceso concurrente a él realice otra operación de tal forma que no se realice correctamente.

1.4.2 Grafos de precedencia.

Es un grafo acíclico y orientado cuyos nodos corresponden a sentencias individuales. Un arco desde un nodo Si a un nodo Sj significa que la sentencia Sj puede ejecutarse sólo después de que Si haya terminado su ejecución. (Tanenbaum, 1997)

1.4.3 Instrucciones para crear procesos.

1.4.3.1 Parbegin / Parend

Se encerrarán entre Parbegin / Parend las instrucciones que se desean ejecutar concurrentemente. Ejemplo:

```
Parbegin
  A;
  B;
  C;
Parend
```

Ejecutándose A, B, C de forma simultánea.

1.5 Metodologías que se utilizan para el desarrollo de Software.

El desarrollo de software no es sin dudas una tarea fácil. Como resultado a este problema ha surgido una alternativa desde hace mucho tiempo: la Metodología. Las metodologías imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. Lo hacen desarrollando un proceso detallado con un fuerte énfasis en planificar inspirado por otras disciplinas de la ingeniería, también su propósito es establecer un contrato social entre todos los participantes en un proyecto para conseguir la solución más eficaz con los recursos disponibles. (Falgueras, 2003)

1.5.1 Proceso Unificado de Desarrollo de Software (RUP).

La metodología RUP (Rational Unified Process) es una metodología para la ingeniería de software que va más allá del simple análisis y diseño orientado a objetos para proporcionar una familia de técnicas que soportan el ciclo completo de desarrollo de software, se caracteriza por ser: Guiado por casos de uso, los mismos son el instrumento para validar la arquitectura del software y extraer los casos de prueba; centrado en la arquitectura, es donde los modelos son proyecciones del análisis y el diseño que constituyen la arquitectura del producto a desarrollar; iterativo e incremental, durante todo el proceso de desarrollo se producen versiones incrementales del producto en desarrollo. (Mendoza, 2004)

Esta metodología se divide en cuatro fases el proceso de desarrollo. Inicio, se hace mayor énfasis en actividades de modelado del negocio y de requerimientos, elaboración el objetivo es determinar la arquitectura óptima, le sigue construcción, en la cual se lleva a cabo la construcción del producto por medio de una serie de iteraciones y por último transición que su objetivo es obtener el producto preparado para su entrega a la comunidad de usuarios.

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes.

En RUP están presente elementos como actividades; que no son más que los procesos que se llegan a determinar en cada iteración, están los trabajadores; que vienen siendo las personas involucradas en cada proceso, los artefactos; que van a ser documentos, modelos, o un elemento de modelo y el flujo de actividades; que es la secuencia de actividades realizadas por trabajadores y que producen un resultado de valor observable. (Mendoza, 2004)

1.5.2 Microsoft Solution Framework (MSF)

Esta es una metodología manejable e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

Concretamente MSF se compone de principios, disciplinas y modelos, los principios no son más que promover la comunicación abierta, trabajar para una visión compartida, fortalecer los miembros del equipo, establecer responsabilidades claras y compartidas, focalizarse en agregar valor al negocio e invertir en la calidad. Las disciplinas van a ser la gestión de proyecto, el control de riesgo y el control de cambios. Además se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: modelo de arquitectura del proyecto, modelo de equipo, modelo de proceso, modelo de gestión del riesgo, modelo de diseño de proceso y finalmente el modelo de aplicación. (Mendoza, 2004)

1.5.3 Programación Extrema (XP)

La programación extrema se basa en la simplicidad, la comunicación y el reciclado continuo de código, para algunos no es más que aplicar una pura lógica. Es utilizada para proyectos de corto plazo.

Esta metodología está basada en pruebas unitarias, estas son las pruebas realizadas a los principales procesos, consisten en comprobaciones (manuales o automatizadas) que se realizan para verificar que el código correspondiente a un módulo concreto de un sistema software funciona de acuerdo con los requisitos del sistema; otra característica es la re fabricación, que se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio y la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. (Mendoza, 2004)

1.6 Lenguajes de Modelado

Un sistema, tanto del mundo real como en el mundo del software, es bastante complejo, por ello es necesario dividir el sistema en partes o fragmentos si se quiere entender y administrar su complejidad. Estas partes se pueden representar como modelos que describan sus aspectos esenciales. Por tanto, un paso útil en la construcción de un sistema de software es el de crear modelos que organicen y comuniquen los detalles más importante de la vida real con que se relacionan y del sistema a construir. (Falgueras, 2003)

1.6.1 Definición Integrada para el Modelado de Funciones (IDEF0)

IDEF0 es una técnica de modelación concebida para representar de manera estructurada y jerárquica las actividades que conforman un sistema o empresa, y los objetos o datos que soportan la interacción de esas actividades. Un modelo IDEF0 se compone de una serie jerárquica de diagramas que permiten mediante niveles de detalle, describir las funciones especificadas en el nivel superior. En las vistas

superiores del modelo la interacción entre las actividades representadas permite visualizar los procesos fundamentales que sustentan la organización. Los elementos gráficos utilizados para la construcción de los diagramas IDEF0 son cuadros y flechas. Permite representar el proceso cronológicamente. Se describe el flujo orientado al cliente final de ese negocio, cruzando todas las actividades de la organización que dan cumplimiento a la solicitud de producto o servicio que realiza el cliente, representando así la "cadena de valor" de la empresa. Permite incorporar en el flujo los datos que entran y salen de las actividades, así como las reglas del negocio y los actores, todo en la misma vista. (Hanrahan, 1999)

1.6.2 Notación para el Modelado de Procesos de Negocio (BPMN)

BPMN define un Diagrama de Procesos de Negocio (BPD, del inglés Business Process Diagram), que se basa en una técnica de grafos de flujo para crear modelos gráficos de operaciones de procesos de negocio. Un modelo de procesos de negocio, es una red de objetos gráficos, que son actividades (trabajo) y controles de flujo que definen su orden de rendimiento. El BPMN se compone de varios conjuntos de elementos que abarcan la representación, tanto de los objetos del flujo y sus conexiones como los instrumentos de ayuda que son las bandas (Swimlanes) y los artefactos. Además está diseñado para cubrir muchos tipos de modelados y para permitir la creación de segmentos de proceso así como procesos de negocio end-to-end, con diferentes niveles de fidelidad. (Barriento, 2008)

El objetivo principal de BPMN es proporcionarle a toda las personas involucradas en el negocios una notación factible para entender cómo funcionan los procesos que componen una empresa, desde el analista de negocio que es el encargado de hacer el borrador inicial, pasando por los desarrolladores que son los encargados de implementar todo el sistema que ejecutara dichos procesos, hasta las personas que son los encargados de ejecutar y gestionar los procesos. BPMN es considerado como un mecanismo simple para crear modelos de procesos de negocio, que lo maneja a través de las cuatro categorías básicas de elementos: objetos de flujo, objetos conectores, rol de proceso, artefactos.

1.6.3 Lenguaje de Modelado Unificado (UML)

El Lenguaje de Modelado Unificado UML es un lenguaje estándar para escribir planos de software. Puede utilizarse para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra gran cantidad de software. Es un lenguaje que ayuda a interpretar grandes sistemas mediante gráficos o texto, obteniendo modelos explícitos que contribuyen a la comunicación durante el desarrollo, ya que al ser estándar, pueden ser interpretados por personas que no participaron en su diseño. En este contexto, UML sirve para especificar modelos no ambiguos y completos. UML es un método formal de modelado. Esto aporta las siguientes ventajas: (Orallo, 2007)

- Mayor rigor en la especificación.
- Permite realizar una verificación y validación del modelo realizado.
- Se puede automatizar determinados procesos y permite generar código a partir de los modelos y a la inversa. Esto hace que el modelo y el código estén actualizados, con lo que siempre se puede mantener la visión en el diseño, de más alto nivel, de la estructura de un proyecto.

UML propone diagramas con la finalidad de presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. Un modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementarlo.

Capítulo 1: Fundamentación Teórica

1.7 Herramientas CASE

Las herramientas de desarrollo de software han desempeñado un importante papel en el desarrollo de aplicaciones. Como consecuencia del avance tecnológico éstas han experimentado también continuos cambios. Estas herramientas favorecen el apoyo al desarrollo de software, proporcionando un conjunto de programas de asistencia a los analistas para la Ingeniería de Software durante todo el ciclo de vida del desarrollo del sistema.

1.7.1 Rational Rose Enterprise Edition

Rational Rose es la herramienta CASE desarrollada por los creadores de UML, que cubre todo el ciclo de vida de un proyecto, desde la fase de inicio, formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases. Permite establecer una trazabilidad real entre el modelo (análisis y diseño) y el código ejecutable. Dentro de sus características principales se puede encontrar un avanzado modelado de UML para trabajar en diseños de bases de datos, con capacidad de representar la integración de los datos y los requisitos de aplicación a través de diseños lógicos y físicos, posee además una fuerte capacidad para integrarse con cualquier sistema de control de versiones. (G. Booch, 2000)

1.7.2 Visual Paradigm.

Visual Paradigm es una herramienta multiplataforma de modelado visual UML y una herramienta CASE muy fácil de utilizar. Tributa una excelente interoperabilidad con otras herramientas CASE ya que tiene conexión con Rational Rose en sus archivos de proyecto (.MDL / .CAT) los cuales pueden ser importados a Visual Paradigm UML a través de esta importante característica, es apoyado por un conjunto de lenguajes tanto en la generación del código como en la ingeniería inversa por mencionar algunos ejemplos, los cuales tiene la capacidad de soporte: Java, C + +, CORBA IDL, PHP, XML Schema, Ada y Python. Permite la captura de requisitos, análisis, diseño e implementación, también proporciona características tales como generación del código, ingeniería inversa y generación de informes. Permite dibujar todos los tipos de diagramas de clases. Apoya los estándares más recientes de las notaciones de UML. Incorpora el soporte para trabajo en equipo, permitiendo que varios desarrolladores trabajen a la vez en el mismo diagrama y vean en tiempo real los cambios hechos por sus compañeros. Dentro de sus características fundamentales se encuentran: (Sierra, 2011)

- Modelado colaborativo con CVS (Concurrent Versions System) y Subversión
- Ingeniería inversa, código a modelo, código a diagrama.
- Generación de código, modelo a código, diagrama a código.
- Editor de Detalles de Casos de Uso, entorno todo en uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Diagramas de flujo de datos.
- Generación de bases de datos, transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos, desde Sistemas Gestores de Bases de Datos existentes a diagramas de Entidad-Relación.
- Generador de informes para generación de documentación.
- Distribución automática de diagramas. Reorganización de las figuras y conectores de los diagramas UML.
- Modelo para realizar prototipos de interfaz.

Brinda la posibilidad de generar código a partir de los diagramas, para plataformas como .Net, Java y PHP, así como obtener diagramas a partir de código.

1.7.3 Enterprise Architect

Enterprise Architect es una herramienta comprensible de diseño y análisis UML, cubriendo el desarrollo de software desde el paso de los requerimientos a través de las etapas del análisis, modelos de diseño, pruebas y mantenimiento. Es una herramienta multi-usuario, basada en Windows, diseñada para ayudar a construir software robusto y fácil de mantener. Ofrece salida de documentación flexible y de alta calidad, provee trazabilidad completa desde el análisis de requerimientos hasta los artefactos de análisis y diseño, a través de la implementación y el despliegue. Combinados con la ubicación de recursos y tareas incorporados, los equipos de administradores de proyectos y calidad están equipados con la información que ellos necesitan para ayudarles a entregar proyectos en tiempo. Las bases de Enterprise Architect están construidas sobre la especificación de UML 1.0 y usa perfiles UML para extender el dominio de modelado, mientras que la validación del modelo asegura integridad. Combina procesos de negocio, información y flujos de trabajo en un modelo usando nuestras extensiones gratuitas para BPMN y el perfil Eriksson-Penker. (Sparxsystems, 2008)

1.8 Ingeniería de Requisitos

La actividad de análisis es una de las principales en el proceso de desarrollo de software y consiste en investigar el problema que da origen, interiorizarlo y describirlo, para luego obtener las necesidades del cliente o requisitos. En el momento de realizar el análisis es necesario una adecuada comunicación entre los clientes y los analistas, debido que partiendo del intercambio que se realiza se obtienen los requisitos a implementar en el sistema.

Para lograr comprender que es la Ingeniera de Requisitos, es necesario definir que es un requisito y cuál es su función. A continuación se mencionan algunos de los conceptos existentes actualmente: (Dávila, 2001)

- Es una condición o una capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar, especificación u otro documento formal.
- Es una descripción de los servicios que brindara el sistema y las restricciones que este incluye. Representa la necesidad que tienen los clientes de un sistema que les resuelve determinado problema.
- Los requisitos de software se clasifican en dos grupos: los funcionales y los no funcionales. Los requisitos funcionales determinan una capacidad o condición que debe tener el sistema y los no funcionales son una propiedad o cualidad que el producto debe tener, o sea, características que hacen el producto atractivo, rápido y confiable.
- La Ingeniería de Requisitos es la aplicación de los procedimientos, técnicas, lenguajes y herramientas para obtener como resultado el análisis, la documentación correspondiente y el seguimiento de las necesidades del usuario. Es un proceso que incluye el descubrimiento de lo que el usuario quiere, el refinamiento de estos, su modelado y su especificación. También se realiza un análisis detallado de las soluciones que se proponen.
- La Ingeniería de Requerimientos tiene un papel muy importante en la construcción de un software, pero a su vez es uno de los procesos más complejos, puesto que según Frederick P. Brooks “La parte más difícil de construir un sistema es precisamente saber qué construir. Ninguna otra parte del trabajo conceptual es tan difícil como establecer los requerimientos técnicos detallado.

Ninguna otra parte del trabajo afecta tanto el sistema si es hecha mal. Ninguna es tan difícil de corregir más adelante. Entonces, la tarea más importante que el ingeniero de software hace para el cliente es la extracción iterativa y el refinamiento de los requerimientos del producto (Brooks, 1995)

Su objetivo principal es lograr que se generen las especificaciones de las necesidades de los usuarios o clientes de forma correcta, con claridad, sin ambigüedades, en forma consistente y compacta.

1.8.1 Etapas de la Ingeniería de Requisitos

La ingeniería de software tiene en cuenta un grupo de etapas que se encuentran lógicamente definidas y estructuradas. Estas etapas cuentan con un grupo de técnicas que facilitan su aplicación en el momento de saber qué es lo que desea el cliente, o qué es exactamente lo que necesita. A continuación se describen cada una de ellas: (Pressman, 2005)

- La Elicitación o Identificación de requisitos es la primera de las etapas de la Ingeniería de requisitos. Consiste en extraer, de cualquier fuente de información disponible, los problemas que debe resolver el sistema. En esta etapa interactúan los clientes con los desarrolladores del sistema para obtener exactamente lo que debe cumplir este, identificándose los requisitos funcionales y los no funcionales.
- Análisis de Requisitos: Una vez que se tienen identificados los requisitos del sistema, se analizan detalladamente para ver si presentan alguna ambigüedad, si son consistentes y se verifica su completitud. En esta etapa es donde se agrupan y se clasifican de acuerdo a las funcionalidades que responde.
- Especificación de requisitos: Es la etapa de la Ingeniería de requisitos donde se describen las funciones y características del sistema. Puede ser mediante un documento o cualquier modelo gráfico que describa como es el comportamiento del software.
- La validación de requisitos es la etapa que se encarga de verificar la calidad de los requisitos identificados. Vela porque los requisitos identificados hayan sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones y que los errores detectados en las anteriores etapas hayan sido corregidos.
- La gestión de requisitos se encarga de darle seguimiento a los requisitos a lo largo de las etapas anteriores. El objetivo es realiza un conjunto de actividades que permiten identificar, seguir y controlar los cambios sufridos en cualquier momento del ciclo de vida del sistema.

1.8.2 Técnicas de Ingeniería de Requisitos

Dentro de las principales técnicas para la identificación de requisitos está la ***entrevista***, que es una técnica muy usada y consta de tres fases: preparación, realización y análisis.

La preparación como su nombre lo indica es la fase donde se hace un estudio previo del dominio del problema se seleccionan los entrevistados y se debe tener claro cuál es el objetivo de la entrevista.

En la realización se lleva a cabo lo antes preparado. Se debe realizar con un lenguaje natural y se debe ir dando conclusiones parciales cada vez que así lo requiera la situación, además de dar las conclusiones finales al terminar la entrevista. Por último, el análisis es donde se estudia y se detalla todo lo hablado durante la entrevista.

Otra de las técnicas utilizadas en la elicitación de requisitos es ***Desarrollo conjunto de aplicaciones (JAD- Joint Application Development)***. Esta técnica es considerada como una alternativa a las

entrevistas individuales. Son reuniones entre un grupo de personas pero de muy corta duración. Va acompañada de imágenes, graficas o cualquier elemento visual que facilite el entendimiento del mensaje que se quiere transmitir.

La **tormenta de ideas** es otra importante y muy utilizada técnica de ***elicitación de requisitos***. Consiste en hacer reuniones de un grupo de participante donde se generan muchas ideas sobre un tema específico. Debe estar guiado por un jefe que sea quien de inicio al debate. El objetivo es ver los diferentes puntos de vista de los participantes.

También la técnica de ***Casos de Uso*** es una técnica muy usada en la elicitación, donde se agrupan los requisitos por funcionalidades para luego describir las acciones del usuario y la respuesta del sistema, o sea, la secuencia de acciones que sigue para lograr un objetivo.

Los ***prototipos*** es una técnica usada en la validación de requisitos que se utilizada para mostrarle al cliente o usuario una propuesta de interfaz de las funcionalidades. Se emplea para lograr una mejor comprensión de los requisitos y consiste en diseñar una propuesta de interfaz de un requisito determinado que debe ir incluido en el producto final, aunque esta puede presentar modificaciones

1.9 Modelo de Sistema

El modelo de sistema, describe cada una de las funcionalidades que se proponen para el sistema. Incluye el diagrama de casos de uso con sus descripciones.
Un caso de uso representa la interacción entre un usuario, que puede ser una persona o una maquina, y el sistema que se desea obtener. Cada uno de estos casos de uso que van incluido en el modelo de casos de uso, tiene una descripción asociada, que describe la funcionalidades que engloba. (Falgueras, 2003)

1.9.1 Patrones de Casos de usos

Un patrón es un problema/solución que estandariza principios y sugerencias relacionadas frecuentemente con la asignación de responsabilidades. Es la representación de reiterados problemas en un tema determinado. (Larman, 1999)

Para la realización del diagrama de casos de uso se utilizan un conjunto de patrones que facilitan la representación de estos. Dentro de los principales patrones de casos de uso están:

- **Regla del negocio**: Se fundamenta en la extracción de información relacionada con el cliente (políticas, reglas y regulaciones del negocio) que luego son tratadas como reglas del negocio. (Larman, 1999)

Concordancia: (Larman, 1999)

- **Reusabilidad**: Es separar en un caso de uso la sub-secuencia de acciones que son obligatorias para más de un caso de uso.
- **Adición**: Es separar en un caso de uso la sub-secuencia de acciones que son alternativas a las funcionalidades de otros casos de uso.
- **Especialización**: Es la representación de una herencia pero enmarcada en casos de uso. Varios casos de uso (hijos) son del mismo tipo de un (padre) con funcionalidades genéricas.

CRUD (Create, Retreive, Update, Delete): (Larman, 1999)

- **Completo**: Representa a un conjunto de funcionalidades que engloban relación. Maneja operaciones como creación, lectura, actualización y eliminación. Por lo general son llamados CRUD o Gestionar.
- **Parcial**: Son similares a los CRUD Completos, pero separan una funcionalidad por determinadas características que la hacen compleja o tediosas.

Múltiples actores: (Larman, 1999)

- **Roles diferentes**: Consiste en un caso de uso y por lo menos dos actores. Es utilizado cuando dos actores juegan diferentes roles en un caso de uso, o sea, interactúan de forma diferente con el caso de uso.
- **Roles común**: Se aplica cuando dos actores juegan un mismo rol sobre un caso de uso. Representa la herencia entre actores.

1.10 Diseño de Sistemas

El Diseño de sistemas de software se realiza para transformar los requisitos identificados en un diseño detallado del sistema en cuestión. Tiene como finalidad evolucionar hacia una arquitectura sólida para el sistema y generar un diseño que sea entendible por los desarrolladores y que ayude y facilite la implementación. Cuando se realiza el diseño de un sistema, se tienen en cuenta un grupo de elementos que amplían y describen el comportamiento del sistema, facilitando el entendimiento de como se deben ejecutar las funcionalidades. A continuación se describen algunos de ellos, como los patrones de diseño, diagramas de secuencia y diagramas de clases del diseño. (Larman, 1999)

1.10.1 Patrones de Diseño

Los patrones de diseño son propuestas de soluciones a problemas frecuentes surgidos en el proceso de desarrollo de software, estos brindan una solución a problemas similares que está probada y bien descrita, o sea, que están sujetos a situaciones muy parecidas. Un patrón de diseño es un ente descriptible de acuerdo a su nombre, el problema (cuando aplicar un patrón), la solución (descripción abstracta del problema) así como sus consecuencias (costos y beneficios). De forma general, un patrón de diseño se puede definir como: (Larman, 1999)

- Una forma más práctica de describir determinados elementos de la organización de un programa.
- Una solución estándar para un problema común de programación.
- Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios.
- Un proyecto o estructura de implementación que logra una finalidad determinada.

Actualmente existen muchos patrones de diseño de software, estos a su vez se agrupan en grandes grupos, entre los populares tenemos:

Patrones GoF (Gang of Four):

Patrones Creacionales: Inicialización y configuración de objetos. (Larman, 1999)

- Fábrica Abstracta: Proporciona una interfaz para crear familias de objetos relacionados o dependientes sin especificar su clase concreta.

- Constructor: Permite a un objeto construir un objeto complejo especificando sólo su tipo y contenido.
- Prototipos: Permite a un objeto crear objetos personalizados sin conocer su clase exacta a los detalles de cómo crearlos.
- Solitario: Garantiza que solamente se crea una instancia de la clase y provee un punto de acceso global a él.

Patrones Estructurales: Separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes. (Larman, 1999)

- Adaptador: Convierte la interfaz que ofrece una clase en otra esperada por los clientes.
- Puente: Desacopla una abstracción de su implementación y les permite variar independientemente.
- Compuesto: Permite gestionar objetos complejos e individuales de forma uniforme.
- Decorador: Extiende la funcionalidad de un objeto dinámicamente de tal modo que es transparente a sus clientes.
- Fachada: Simplifica los accesos a un conjunto de objetos relacionados proporcionando un objeto de comunicación.

Patrones de Comportamiento: Más que describir objetos o clases, describen la comunicación entre ellos. (Larman, 1999)

- Cadena de Responsabilidad: Evita el acoplamiento entre quién envía una petición y el receptor de la misma.
- Comando: Encapsula una petición de un comando como un objeto.
- Intérprete: Dado un lenguaje define una representación para su gramática y permite interpretar sus sentencias.
- Iterador: Acceso secuencial a los elementos de una colección.
- Mediador: Define una comunicación simplificada entre clases.
- Observador: Una forma de notificar cambios a diferentes clases dependientes.
- Estado: Modifica el comportamiento de un objeto cuando su estado interno cambia.
- Estrategia: Define una familia de algoritmos, encapsula cada uno y los hace intercambiables.
- Visitante: Representa una operación que será realizada sobre los elementos de una estructura de objetos, permitiendo definir nuevas operaciones sin cambiar las clases de los elementos sobre los que opera.

Los patrones de diseño pueden aumentar o disminuir la capacidad de entendimiento de un diseño o de una implementación, pueden hacer lo mismo con la cantidad de código, todo depende del uso adecuado que se les dé. Una vez dominados los patrones de diseño, éstos serán de gran ayuda a la hora de incorporar calidad al producto que se desarrolla, así como valor agregado al mismo.

1.10.2 Modelo del Diseño

El modelo de diseño es un artefacto que se genera en el flujo de trabajo Análisis y Diseño, específicamente en la etapa de diseño. Este constituye una abstracción de la implementación del sistema y tiene como objetivo documentar el diseño sobre el que se apoya el flujo de trabajo de implementación. Está compuesto por clases de diseño, subsistemas, paquetes, colaboraciones y las relaciones entre ellos. Los paquetes de diseño son los encargados de agrupar los elementos de modelo de diseño relacionados

con el objetivo de darle organización. No ofrecen una interfaz formal, aunque puede darse el caso de que expongan algunos de sus contenidos que ofrecen comportamiento. Los paquetes de diseño deben utilizarse fundamentalmente como herramienta organizativa del modelo, para agrupar elementos relacionados. Los subsistemas, a diferencia de los paquetes de diseño, encapsulan comportamiento, proporcionando interfaces formales y no incluyen el contenido interno. Las clases del diseño son una representación de un grupo de objetos que tienen la misma responsabilidades, relaciones, operaciones y atributos. (Falgueras, 2003)

1.11 Métricas para el diseño

Las métricas en cualquier ciencia tienen una gran importancia, pues permiten medir la calidad con que se obtienen los resultados esperados. La ingeniería informática no se queda atrás en este sentido, se deben aplicar métricas que permitan medir con objetividad no con subjetividad la calidad del producto, las líneas de código producidas, velocidad de ejecución, tamaño de memoria, y los defectos informados durante un periodo de tiempo establecido. Se deben recopilar las métricas acorde a lo que se desea medir para que los indicadores del proceso y del producto puedan ser ciertos. (Autores, 2001)

1.12 Conclusiones

En este capítulo se hizo un estudio detallado de los principales aspectos relacionados con el Análisis y Diseño de un sistema. Se compararon acepciones de los diferentes autores sobre los conceptos más importantes para el desarrollo de la investigación. Se hizo una comparación de las principales herramientas de gestión de proyecto usadas en el Mundo, Cuba y específicamente en la UCI. Se realizó un estudio de las tendencias mundiales sobre metodologías de desarrollo, lenguajes de modelado y las herramientas CASE que permiten la modelación. Como resultado del estudio anteriormente mencionado se llegó a las siguientes conclusiones:

- Se seleccionó la metodología RUP por ser una herramienta rica en documentación y facilitarle la comprensión a las demás personas que van a dar continuación a las otras etapas de la investigación además, fue la seleccionada por la dirección del proyecto Laboratorio Virtual de Sistemas Operativos al cual pertenece la actual investigación.
- Se utilizará el lenguaje de modelado de negocio BPMN por ser un lenguaje muy descriptivo y fácil de modelar, que permite la clara comprensión del proceso en cuestión.
- Se utilizará el lenguaje de modelado de sistema UML por facilitar la comprensión de lo que se desea modelar, sirve para especificar modelos no ambiguos y completos además, por ser el lenguaje que propone la metodología RUP.
- Se empleará como herramienta CASE Visual Paradigm en su versión 4.1, por ser una herramienta muy buena para el modelado, que da soporte a varios lenguajes de modelado como BPMN y UML, permite realizar todo tipo de diagramas como prototipos de interfaz de usuario y modelado de bases de datos además de que es la herramienta CASE utilizada en la universidad para el modelado.

Capítulo 2: Descripción de la solución

2.1 Introducción

En el presente capítulo se muestra un mapa conceptual donde se relacionan los contenidos a tener en cuenta para la construcción del simulador, se realiza una modelación de los procesos a simular, esto se hace a través del diagrama de flujo de procesos. También se especifican los requisitos funcionales y los no funcionales por los cuales se regirá el sistema propuesto, se identificarán mediante el diagrama de casos de uso del sistema las relaciones entre sus actores y los casos de uso, así como la descripción de los mismos además en el análisis definido por la metodología de desarrollo escogida, se tuvieron en cuenta los requisitos funcionales durante la confección de los diagramas de clases del análisis y los diagramas de secuencia correspondientes a cada caso de uso. El diseño consolidará el análisis propuesto con los diagramas de clases y la aplicación de patrones de diseño como una manera más práctica de describir ciertos aspectos, teniendo en cuenta las cualidades o propiedades que el sistema debe tener.

2.2 Mapa Conceptual

Los mapas conceptuales son artefactos para la organización y representación del conocimiento. Su objetivo es representar relaciones entre conceptos en forma de proposiciones. Los conceptos están incluidos en cajas o círculos, mientras que las relaciones entre ellos se explicitan mediante líneas que unen sus cajas respectivas. Las líneas, a su vez, tienen palabras asociadas que describen cuál es la naturaleza de la relación que liga los conceptos. Se estructuran en forma jerárquica en la que los conceptos más generales están en la raíz del árbol y a medida que vamos descendiendo por el mismo nos vamos encontrando con conceptos más específicos. (Mapas Conceptuales, 2004)

El mapa conceptual mostrado a continuación representa la relación didáctica de todo el contenido referente al tema de sincronización y exclusión mutua de procesos, lo que constituye la base teórica en la cual se desarrolla nuestra investigación.

El sistema propuesto y cuyas fases de Análisis y Diseño se desarrollarán en este trabajo aborda los conceptos referentes a exclusión mutua con bloqueo, con lo cual se incluyen los conceptos de condiciones de competencia, variables de semáforos con la implementación de las primitivas P() y V() aplicadas a los grafos de precedencia y desarrollados utilizando la estructura Parbegin / Parend.

Capítulo 2: Descripción de la Solución

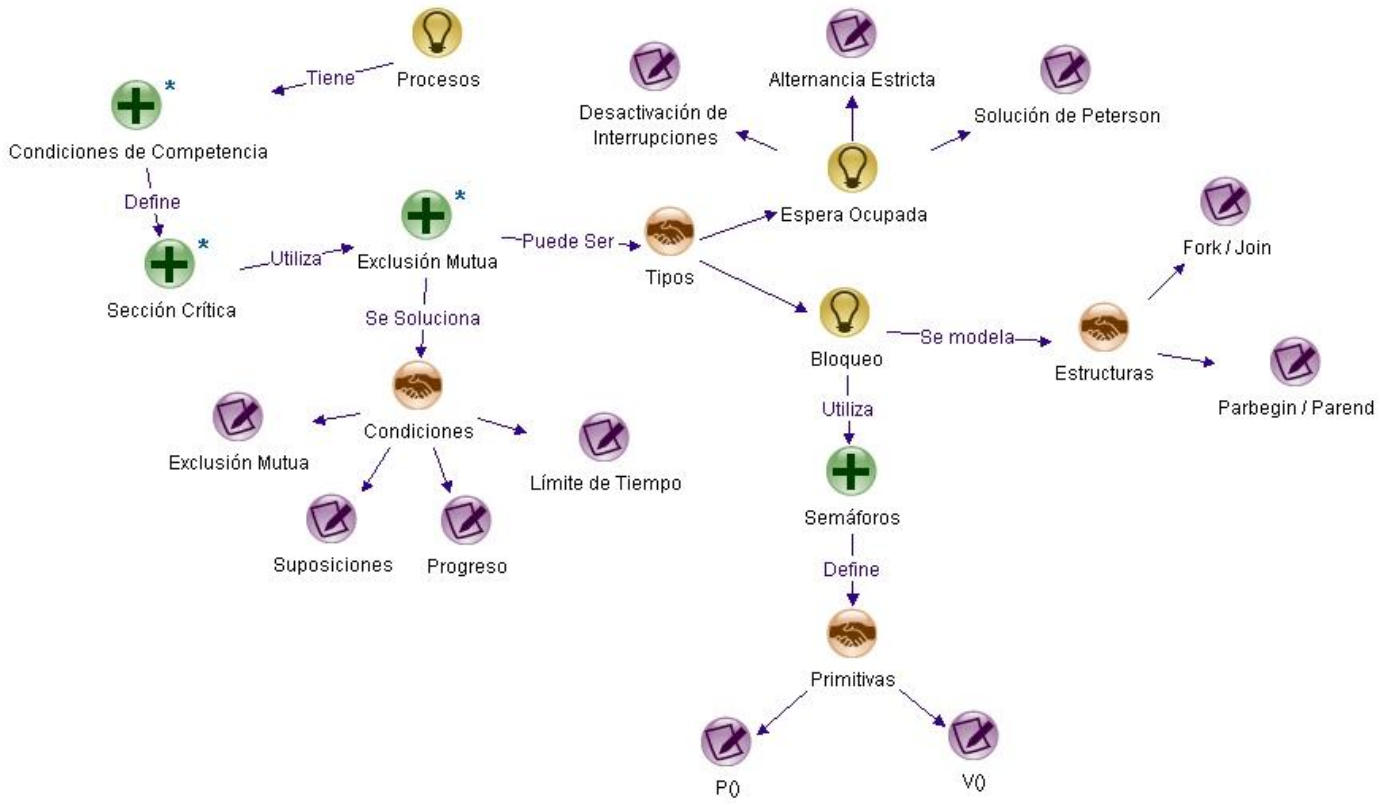


Figura 1: Mapa conceptual del tema "Sincronización y Exclusión mutua".

2.3 Diagramas de flujo de procesos

En el presente trabajo se realiza la simulación de los procesos del tema Administración de Procesos específicamente los referentes a la programación de grafos de precedencia utilizando la estructura Parbegin/Parend descritos, desde el punto de vista teórico, en el contenido # 3 de la asignatura Sistema Operativo. Como autor fundamental de referencia se utilizó Andrew S. Tanenbaum. Se identificaron 6 flujos críticos, los que muestran en los diagramas de procesos siguientes:

El proceso Inicializar Código describe las acciones que se desarrollan así como los datos que se van obteniendo en cada una de las etapas del proceso. Como entrada se tiene un código y como salidas se obtienen los nodos inicio y fin, la identificación de los bloques concurrentes, la ubicación de las primitivas y las variables de semáforos inicializadas.

Capítulo 2: Descripción de la Solución

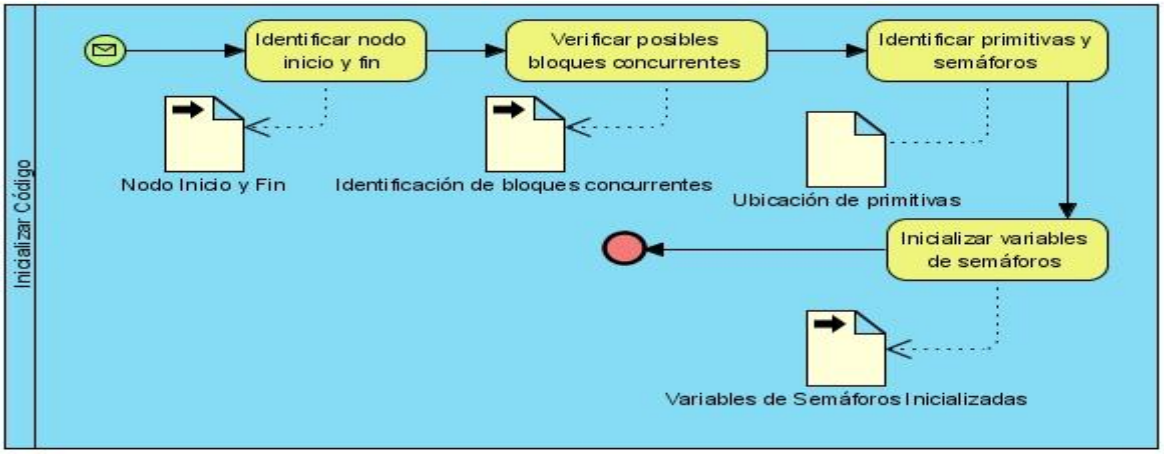


Figura 2: Diagrama de Proceso del CU "Inicializar Código"

Un flujo similar se describe en el diagrama de proceso correspondiente a "Inicializar Grafo" con la diferencia de que la entrada es un grafo realizado anteriormente por el usuario

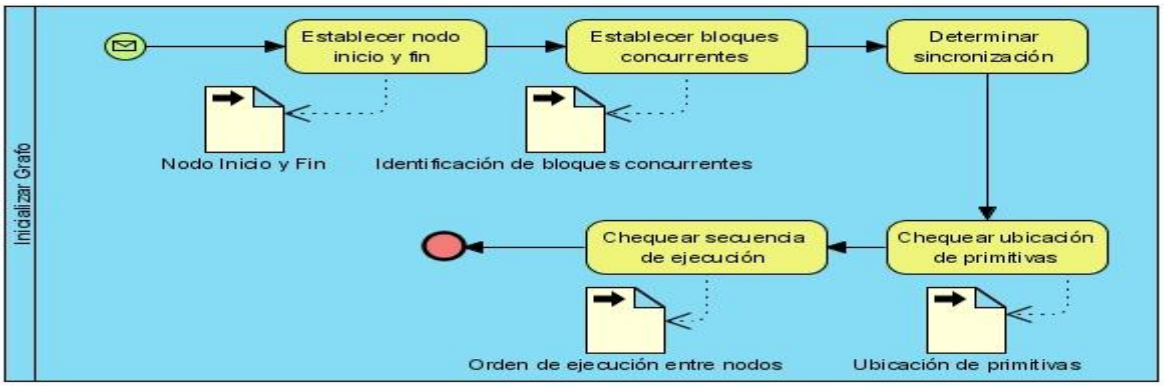


Figura 3: Diagrama de Proceso del CU "Inicializar Grafo"

En el caso del proceso "Chequear Código" la entrada del mismo es un código escrito por el usuario y que el sistema debe validar. Como salida se obtiene un mensaje del sistema informando de la correctitud o no del mismo.

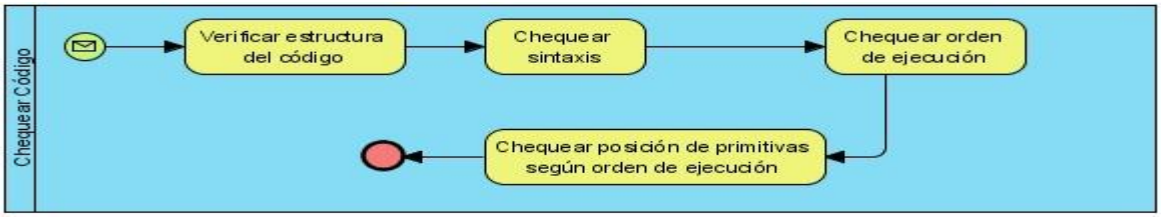


Figura 4: Diagrama de Proceso del CU "Chequear Código"

Capítulo 2: Descripción de la Solución

El proceso “Chequear Grafo” mantiene el mismo flujo que el proceso anterior excepto que la entrada es un grafo y que además de la validez del mismo se debe chequear secuencia de ejecución y estructura.

Figura 5: Diagrama de Proceso del CU “Chequear Grafo”

El proceso “Generar Código” describe el conjunto de acciones que realiza el sistema a partir de la petición del usuario de generar un código tomando desde punto de partida un grafo. Como salidas se obtiene la identificación del Nodo Inicio, la Estructura Lógica de Bloques Concurrentes, Posición de Primitivas, la Secuencia Lógica de Ejecución y el Nodo Fin.

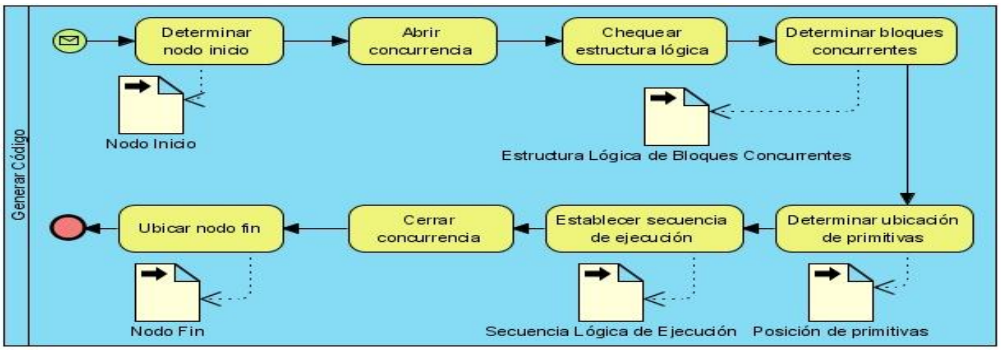


Figura 6: Diagrama de Proceso del CU “Generar Código”

Por su parte el proceso “Generar Grafo” describe el conjunto de acciones que realiza el sistema a partir de la petición del usuario de generar un grafo tomando desde punto de partida un código. Como salidas se obtiene la identificación del Nodo Inicio y Fin, Variables de Semáforos, Posición de Primitivas según concurrencia.

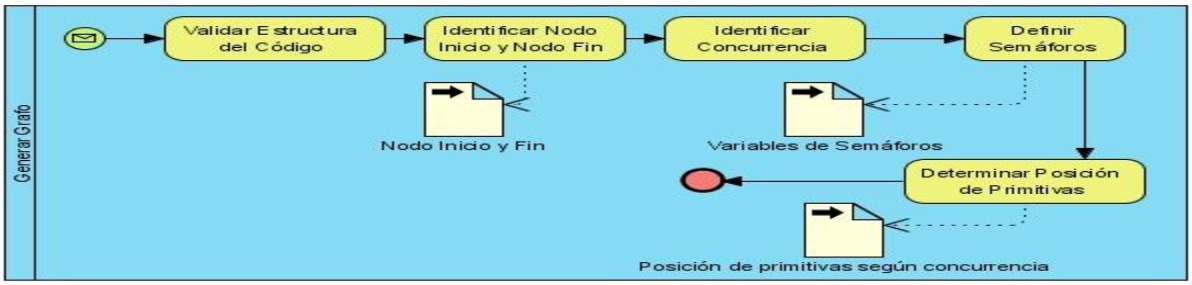


Figura 7: Diagrama de Proceso del CU “Generar Grafo”

Capítulo 2: Descripción de la Solución

2.4 Especificación de los requerimientos del software

2.4.1 Requisitos Funcionales

Referencias	Requisitos Funcionales
RF01	Definir nodo Inicio y Fin
RF02	Verificar bloques concurrentes
RF03	Identificar primitivas y semáforos
RF04	Verificar estructura del código
RF05	Chequear sintaxis
RF06	Chequear orden de ejecución
RF07	Chequear posición de primitivas según orden de ejecución
RF08	Verificar secuencia redundante
RF09	Chequear definición de bloques concurrentes
RF10	Validar Estructura del Código
RF11	Definir Semáforos
RF12	Determinar Posición de Primitivas
RF13	Establecer nodo inicio y fin
RF14	Establecer bloques concurrentes
RF15	Determinar sincronización
RF16	Abrir concurrencia
RF17	Chequear estructura lógica
RF18	Cerrar concurrencia
RF19	Ubicar nodo fin

2.4.2 Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener, forman una parte significativa de la especificación, las propiedades no funcionales; como cuán usable, seguro, conveniente y agradable, distinguen a un producto bien aceptado de uno con poca aceptación. (Brooks, 1995)

Capítulo 2: Descripción de la Solución

A continuación se muestran los requerimientos no funcionales:

Apariencia o interfaz externa:
RNF1: El diseño de la interfaz debe ser sencillo y fácil de usar con reconocimiento visual a través de elementos visibles que identifiquen cada una de sus acciones.
RNF2: La combinación de colores debe ser agradable a la vista del usuario.
RNF3: Debe contar con un vínculo a la ayuda en la ventana principal del trabajo.

Usabilidad:
RNF4: El sistema puede ser usado por cualquier estudiante que posea conocimientos básicos sobre el funcionamiento interno de un Sistema Operativo.

Rendimiento:
RNF5: La respuesta a solicitudes más complejas de los usuarios del sistema no debe exceder 9 segundos.
RNF6: La aplicación deberá estar disponible las 24 horas del día.

Portabilidad:
RNF7: Debe poder ejecutarse tanto en Sistemas Operativos desde Windows XP Profesional Service Pack 1 o Superior como en Sistemas Operativos Linux.

Hardware:
RNF8: Tarjeta de memoria RAM de 128 MB o superior.
RNF9: Procesador Pentium II o superior a 250 MHz como mínimo.
RNF10: Computadora cliente de 40 Gb de disco duro o superior.

Ayuda del sistema
RNF11 El sistema contará con una ayuda que constituirá una guía de apoyo en el momento de hacer uso de la aplicación. Esta describe todas las funcionalidades del sistema.

Interfaces de usuario
RNF12 El sistema tendrá las funcionalidades principales en una primera (principal) interfaz. El objetivo es no tener que acceder a varias secciones para ver los elementos que más interés le causen. Es un sistema con las funcionalidades disponible desde un bandeja de entrada con los elementos fundamentales para desempeñar su tarea.

2.5 Actores del sistema

Actor	Descripción
Usuario	Es el encargado de realizar cualquier operación dentro del sistema, como Chequear o Generar Grafo, Código o Solución así como Salvar o Cargar Grafo, Código o Solución.

2.5.1 Diagrama de casos de uso del sistema

Los diagramas de casos de uso sirven para especificar la funcionalidad y el comportamiento de un sistema mediante su interacción con los usuarios y otros sistemas.

Capítulo 2: Descripción de la Solución

A partir del análisis de los diagramas de procesos obtenidos y de la especificación de requisitos se llegó al siguiente diagrama de paquetes donde se relacionan los 3 diagramas de casos de usos que describen las funcionalidades en nuestro trabajo:

El diagrama de paquetes representa la relación entre Gestionar Grafo, Gestionar Código y Gestionar Solución los que a su vez incluyen los diagramas de casos de usos correspondientes a las funcionalidades del sistema.

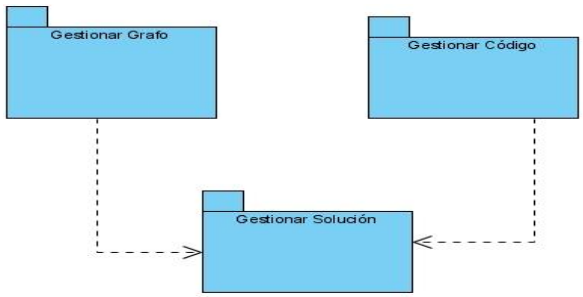


Figura 8: Diagrama de Paquetes.

El diagrama de casos de usos, incluido dentro del paquete Gestionar Código, muestra los casos de uso Chequear Código y Generar Código, ambos inicializado por el Usuario, los que a su vez tienen como incluido el caso de uso Inicializar Código. Todos son considerados casos de usos críticos.

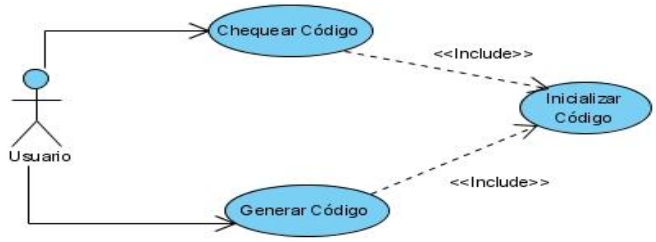


Figura 9: Diagrama de CU incluido dentro del paquete Gestionar Código

El diagrama de casos de usos, incluido dentro del paquete Gestionar Grafo, muestra los casos de uso Chequear Grafo y Generar Grafo, ambos inicializado por el Usuario, los que a su vez tienen como incluido el caso de uso Inicializar Grafo. Todos son considerados casos de usos críticos.

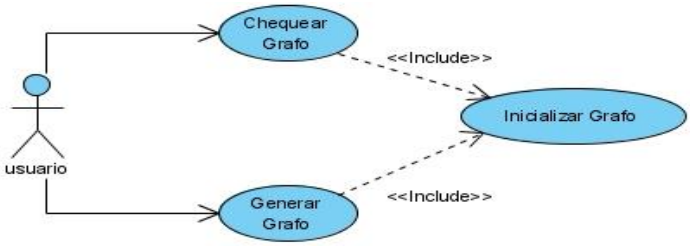


Figura 10: Diagrama de CU incluido dentro del paquete Gestionar Grafo.

El diagrama de casos de usos, incluido dentro del paquete Gestionar Solución, muestra dos conjuntos de acciones: Salvar y Cargar, las que son desglosadas en seis casos de usos inicializados todos por el Usuario, dentro de los que se encuentran Cargar Código, Grafo y Solución y Salvar Código, Grafo y Solución.

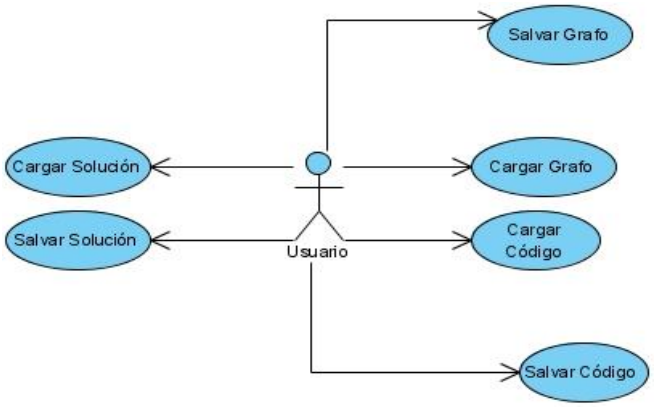


Figura 11: Diagrama de CU incluido dentro del paquete Gestionar Solución.

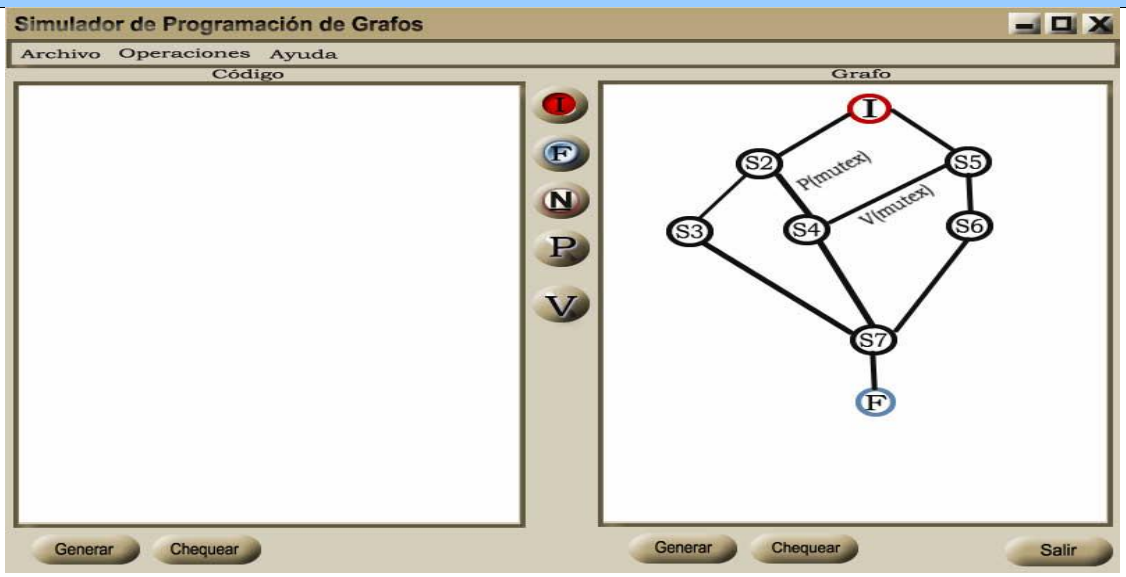
2.5.2 Descripción de los casos de uso del sistema

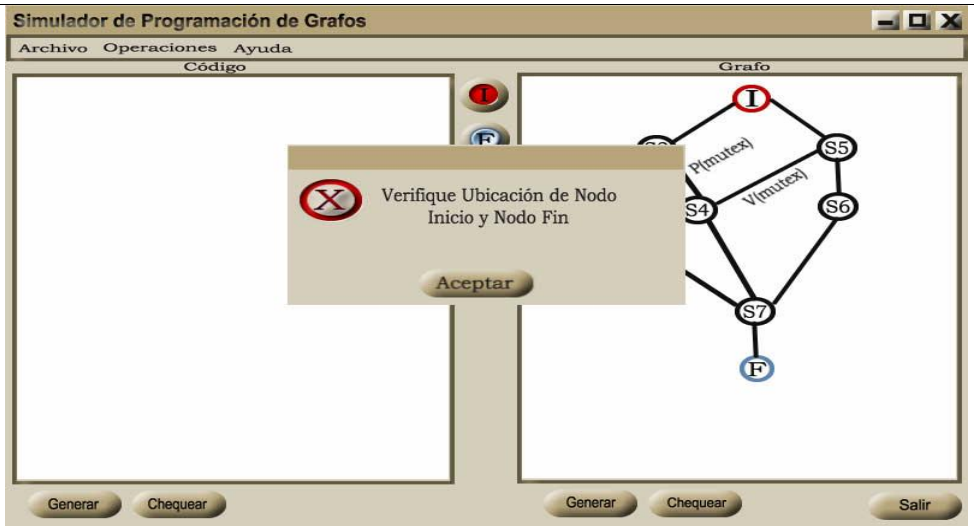
Los casos de uso son artefactos narrativos que describen, bajo la forma de acciones y reacciones, el comportamiento del sistema desde el punto de vista del usuario. Por lo tanto, establece un acuerdo entre clientes y desarrolladores sobre las condiciones y posibilidades (requisitos) que debe cumplir el sistema. (Larman, 1999)

A continuación se muestra la descripción textual del caso de uso "Generar Código".

Descripción del Caso de Uso: "Generar Código"

Caso de uso:	Generar Código
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario selecciona la opción "Generar Código" en el menú principal y termina cuando el código generado es visualizado en la aplicación
Precondiciones:	Se debe verificar que el grafo existe
Pos condición:	El sistema devuelve un mensaje de código generado y muestra el código en el área correspondiente
Casos de usos asociados:	Tiene incluido el caso de uso Inicializar Código
Referencia:	RF01, RF06, RF09, RF12, RF16, RF17
Prioridad:	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso comienza cuando el	2. Determinar nodo inicio y nodo fin

usuario selecciona la opción Generar Código	3. Abrir concurrencia 4. Chequear estructura lógica 5. Determinar bloques concurrentes 6. Determinar ubicación de primitivas 7. Establecer secuencia de ejecución 8. Cerrar concurrencia 9. Generar código 10. Visualizar código generado, regresar al menú inicial y culmina el caso de uso
Prototipo de Interfaz de Usuario	
	
Flujo Alterno 1: Determinar nodo inicio y nodo fin	
Acción del Actor	**Respuesta del Sistema**
	2. Si los datos no son válidos el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Prototipo de Interfaz de Usuario	

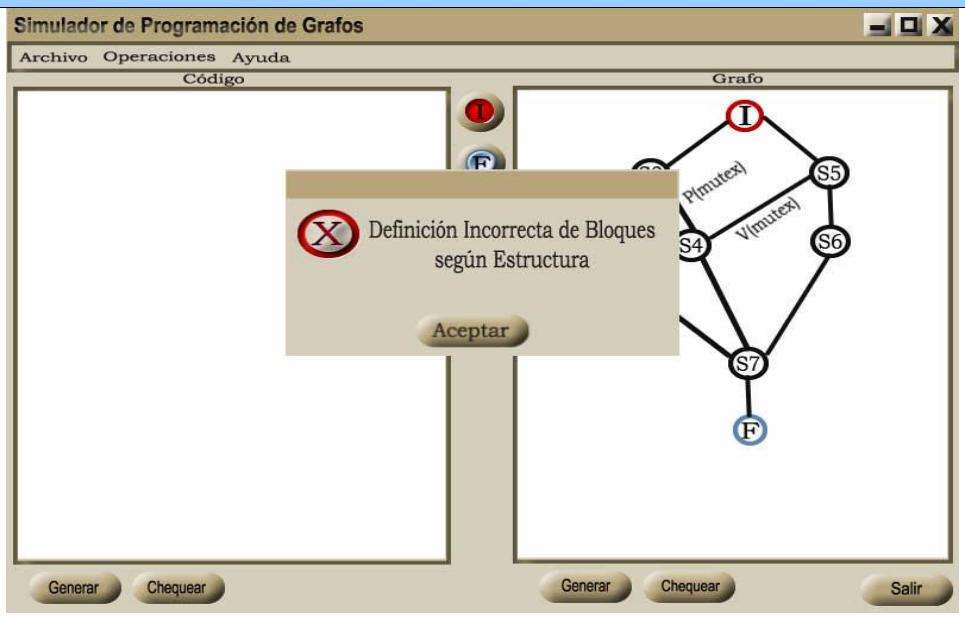


Flujo Alterno 2: Chequear estructura lógica	
Acción del Actor	**Respuesta del Sistema**
	4. Si la estructura no es válida el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.

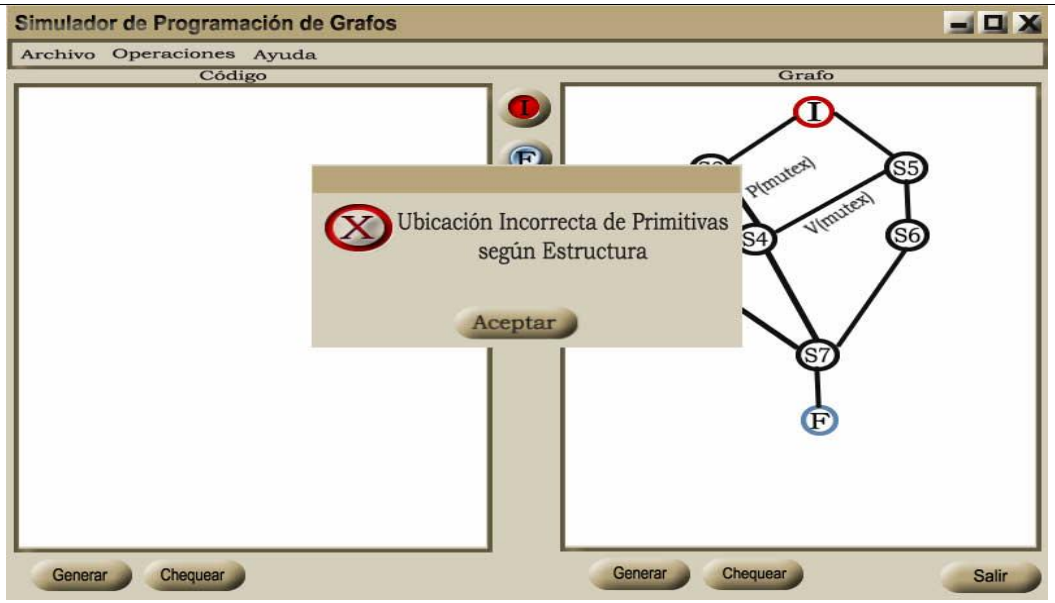
Prototipo de Interfaz de Usuario

Flujo Alterno 3: Determinar bloques concurrentes

Acción del Actor	Respuesta del Sistema
	5. Si los bloques definidos no coinciden con la estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Prototipo de Interfaz de Usuario	



Flujo Alterno 4: Determinar ubicación de primitivas	
Acción del Actor	**Respuesta del Sistema**
	6. Si las primitivas no se encuentran ubicadas según definición de bloques concurrentes y estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Prototipo de Interfaz de Usuario	



Flujo Alterno 5: Establecer secuencia de ejecución	
Acción del Actor	**Respuesta del Sistema**
	7. Si la secuencia de ejecución no está correctamente delimitada o no coincide con la estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.

Prototipo de Interfaz de Usuario

Capítulo 2: Descripción de la Solución

2.6 Análisis del Sistema

Las actividades del análisis son desarrolladas con el objetivo de facilitar la entrada al diseño, por lo que son un paso inicial y una primera aproximación conceptual para aumentar el nivel de especificidad en aras de garantizar el cubrimiento de los requisitos funcionales y obtener una visión de qué hace el sistema.

Las clases del análisis van a representar abstracciones de conceptos, en las cuales deben incluirse atributos y operaciones a un nivel alto. Existen tres estereotipos de clases del análisis estandarizado en UML y se utilizan para ayudar a los desarrolladores a distinguir el ámbito de las diferentes clases. Estas clases son: (Almaguer, 2009)

- Interfaz: Modelan la interacción entre el sistema y sus actores.
- Control: Coordinan la realización de un caso de uso, controlando las actividades de los objetos que implementan su funcionalidad.
- Entidad: Modelan información que posee larga vida y que por lo general es persistente. (Almaguer, 2009)

2.6.1 Diagramas de Clases de Análisis

El diagrama de clases del análisis (DCA) es un artefacto en el que se representan los conceptos en un dominio del problema. En este tipo de diagrama se representan las clases y sus relaciones. Ellos representan una vista estática del sistema. Se muestran los diagramas de clases del análisis de los casos de usos más significativos. (Almaguer, 2009)

A continuación se muestran los diagramas de clases del análisis correspondiente a los casos de uso.

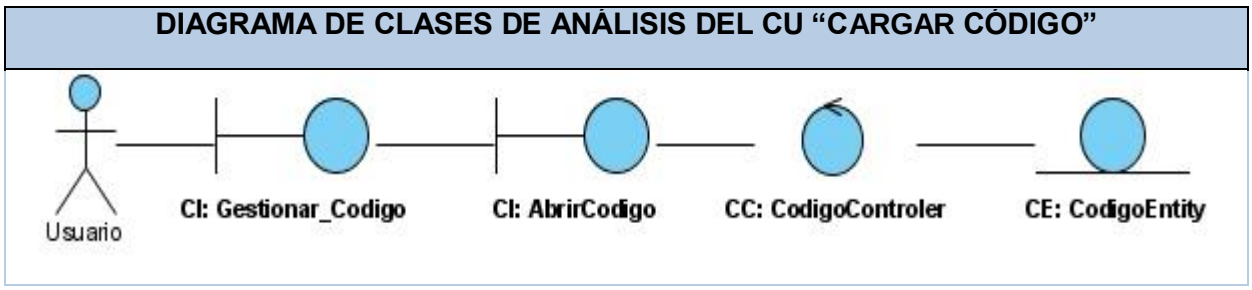


Figura 12: Diagrama de Clases de Análisis del CU "Cargar Código"

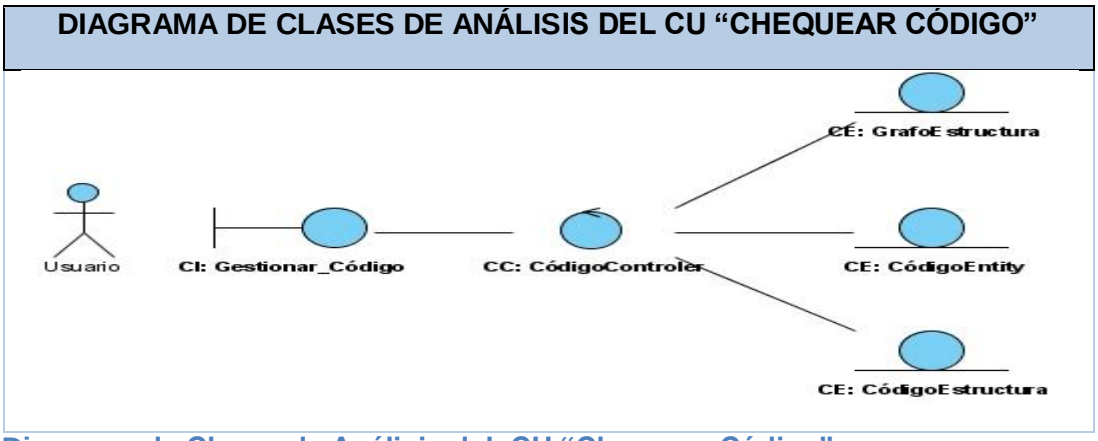


Figura 13: Diagrama de Clases de Análisis del CU "Chequear Código"

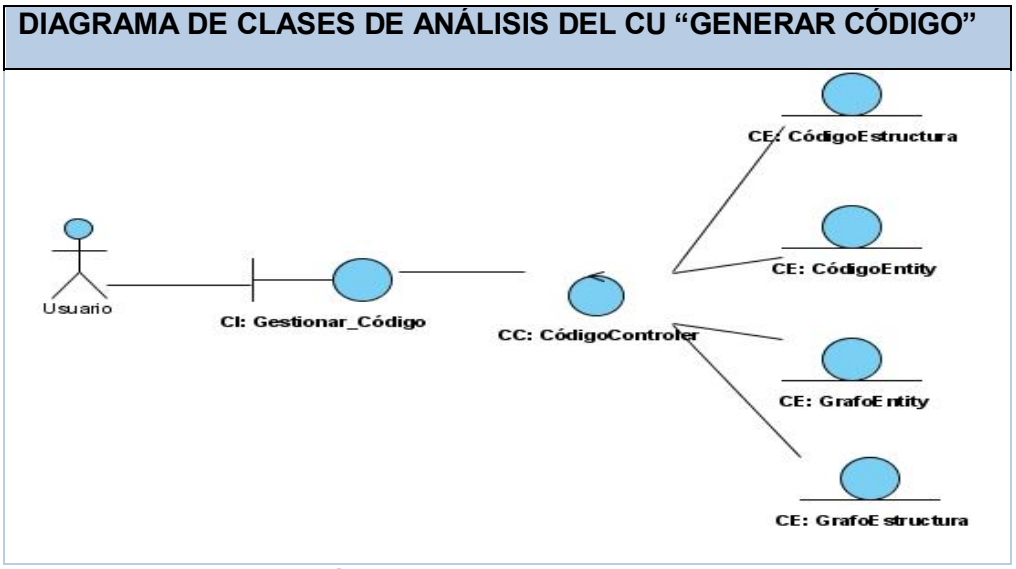


Figura 14: Diagrama de Clases de Análisis del CU "Generar Código"

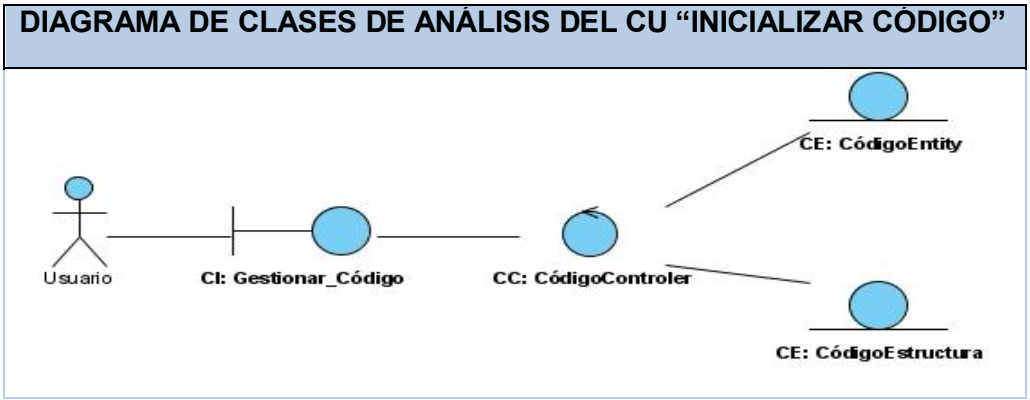


Figura 15: Diagrama de Clases de Análisis del CU "Inicializar Código"

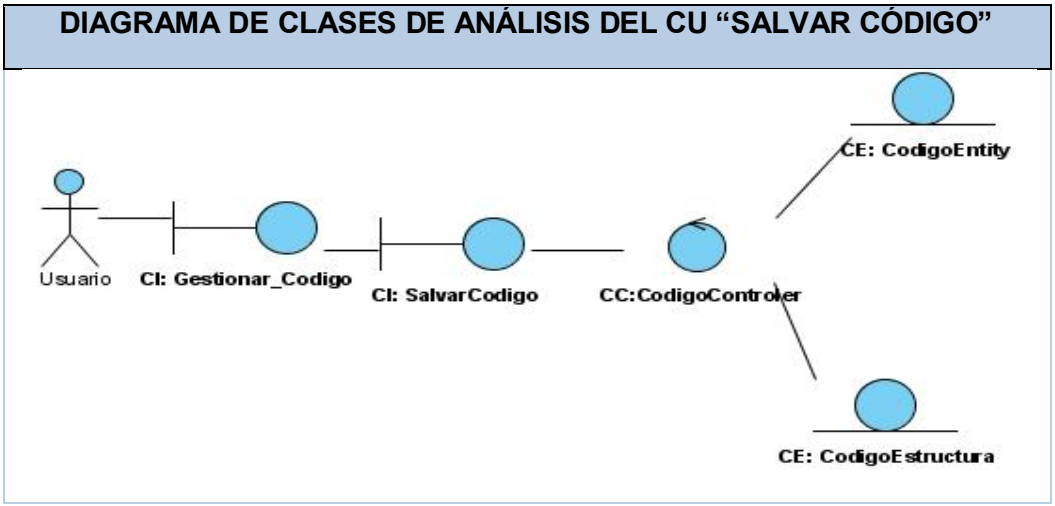


Figura 16: Diagrama de Clases de Análisis del CU "Salvar Código"

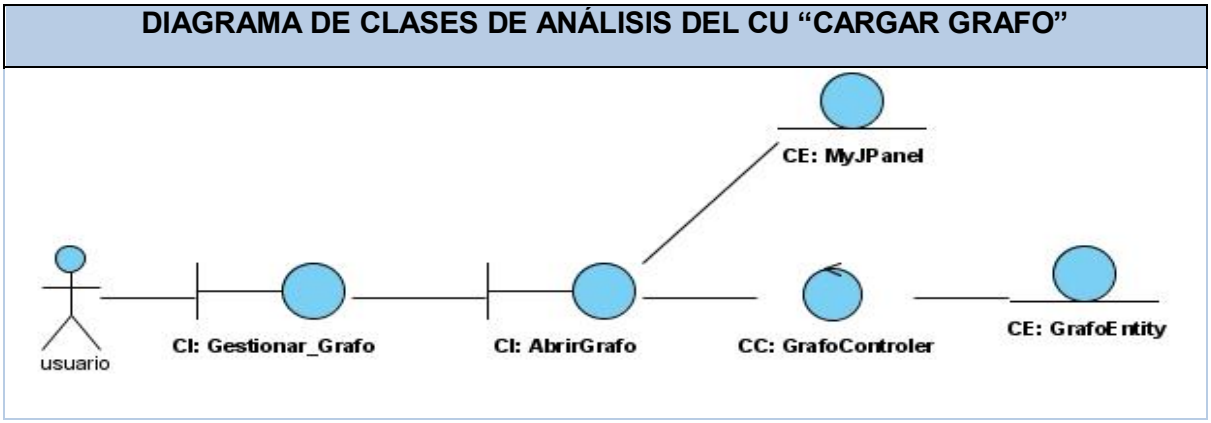


Figura 17: Diagrama de Clases de Análisis del CU "Cargar Grafo"

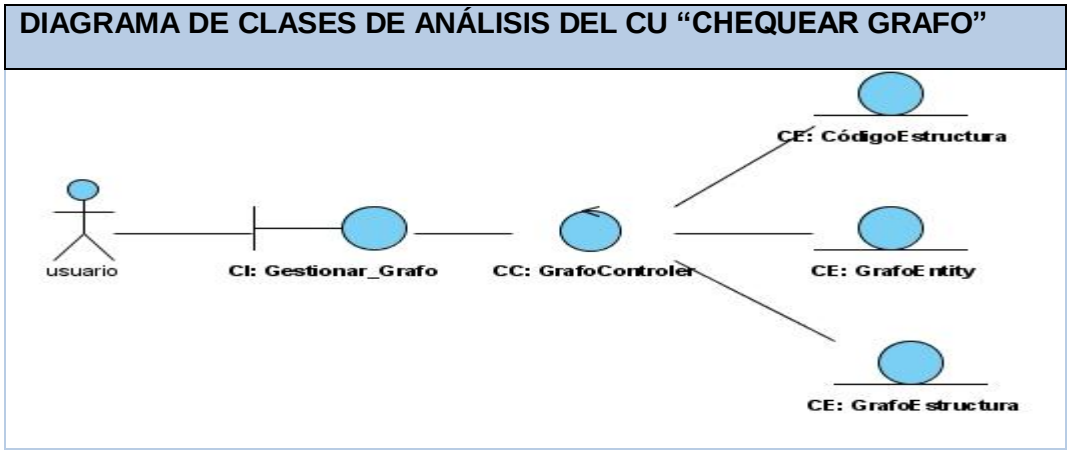


Figura 18: Diagrama de Clases de Análisis del CU "Chequear Grafo"

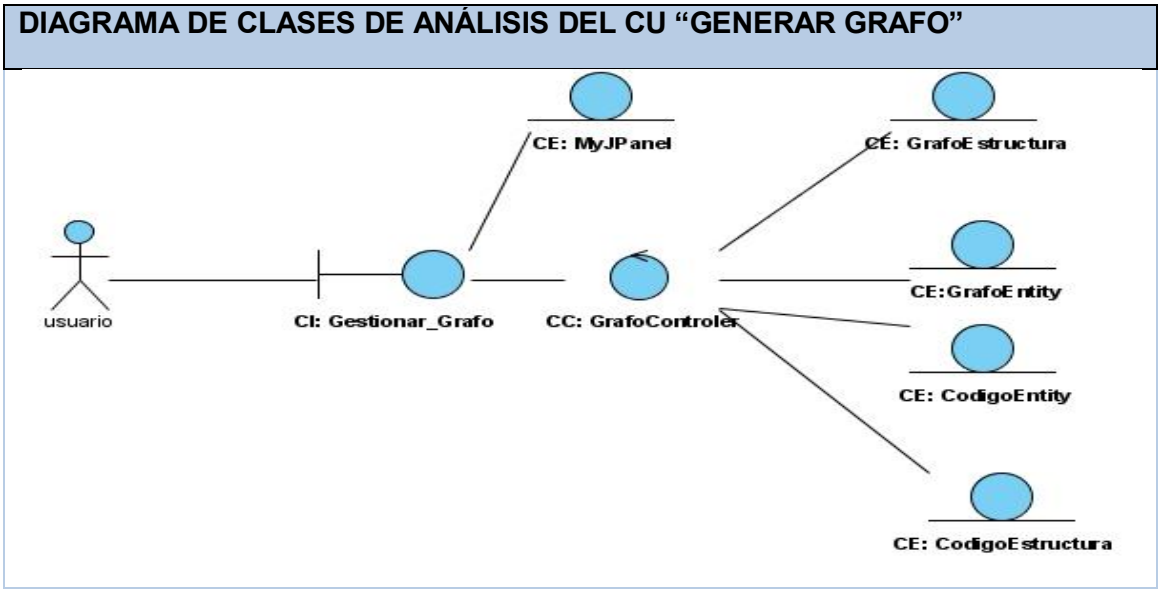


Figura 19: Diagrama de Clases de Análisis del CU "Generar Grafo"

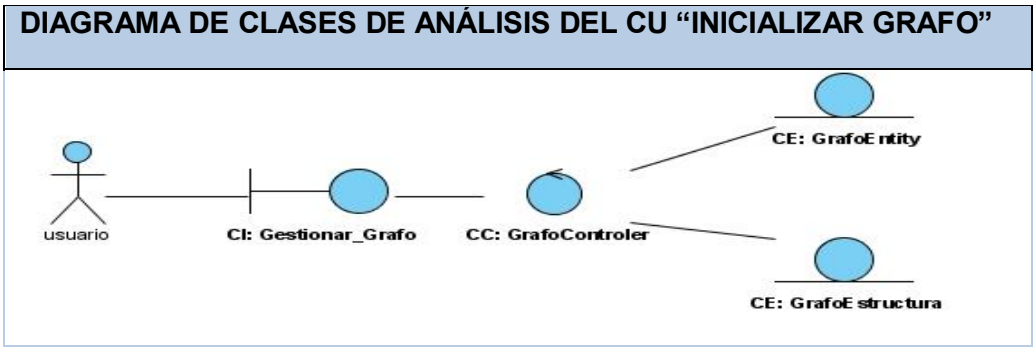


Figura 20: Diagrama de Clases de Análisis del CU "Inicializar Grafo"

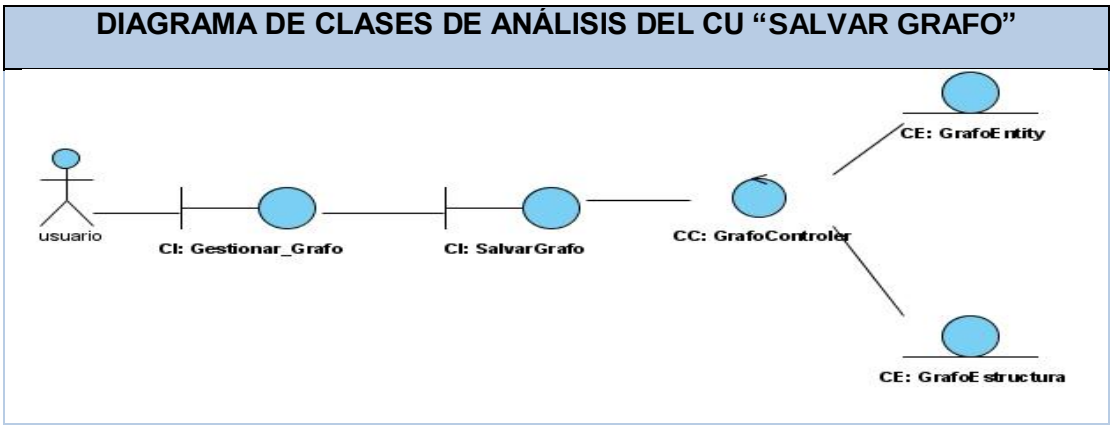


Figura 21: Diagrama de Clases de Análisis del CU "Salvar Grafo"

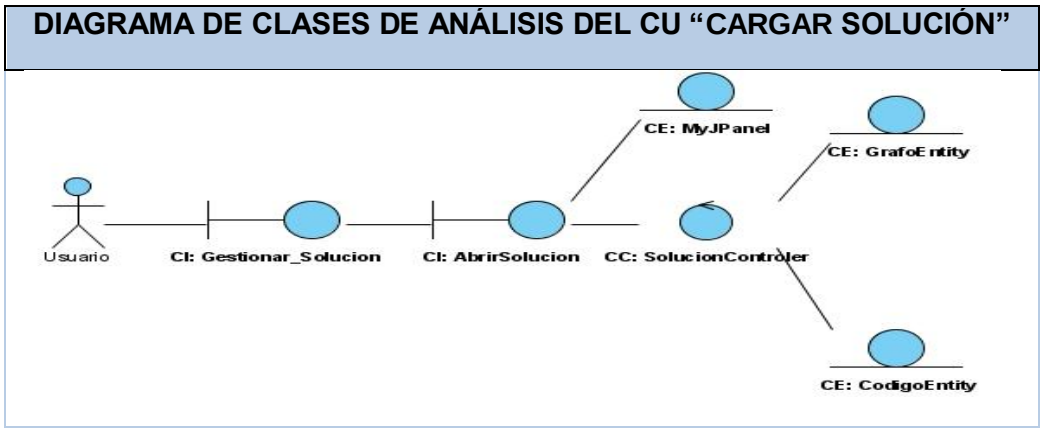


Figura 22: Diagrama de Clases de Análisis del CU "Cargar Solución"

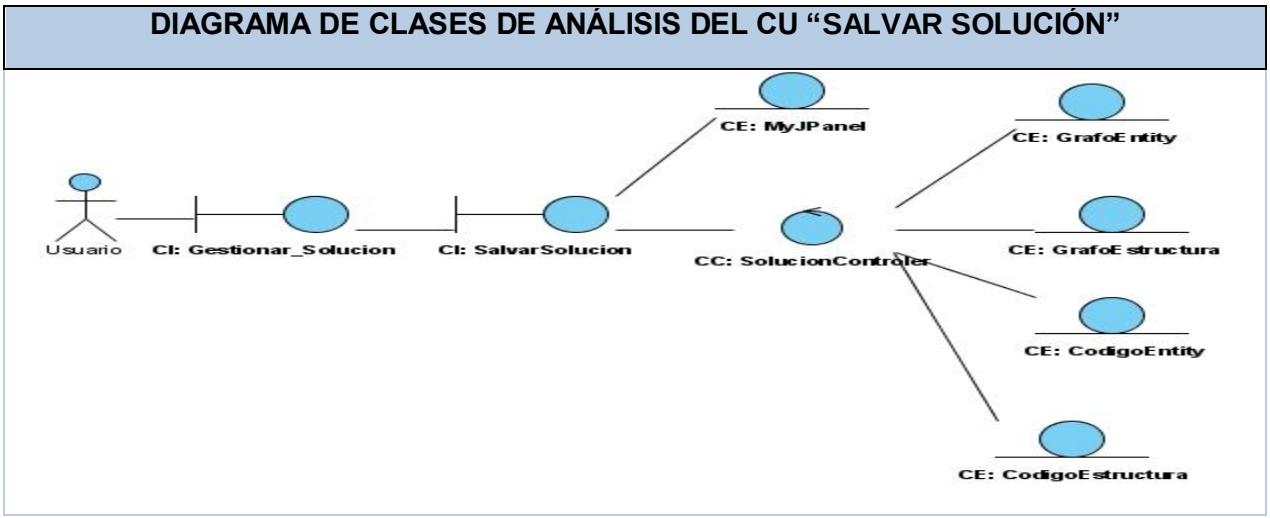


Figura 23: Diagrama de Clases de Análisis del CU "Salvar Solución"

2.6.2 Diagramas de Interacción

Un diagrama de interacción muestra gráficamente cómo los objetos interactúan a través de mensajes para realizar las tareas. No solo son importantes para modelar los aspectos dinámicos de un sistema, sino también para construir sistemas ejecutables por medio de ingeniería directa e inversa. Existen dos tipos de diagramas de interacción: secuencia y colaboración. Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación, contiene detalles de implementación de los escenarios, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos. (Oca, 2009)

Se realizaron los diagramas de secuencia correspondientes a los casos de uso críticos Inicializar memoria física, Simular técnicas, conformar memoria virtual y Simular algoritmos de Reemplazo. En estos diagramas se podrá apreciar y detallar mejor el análisis.

Capítulo 2: Descripción de la Solución

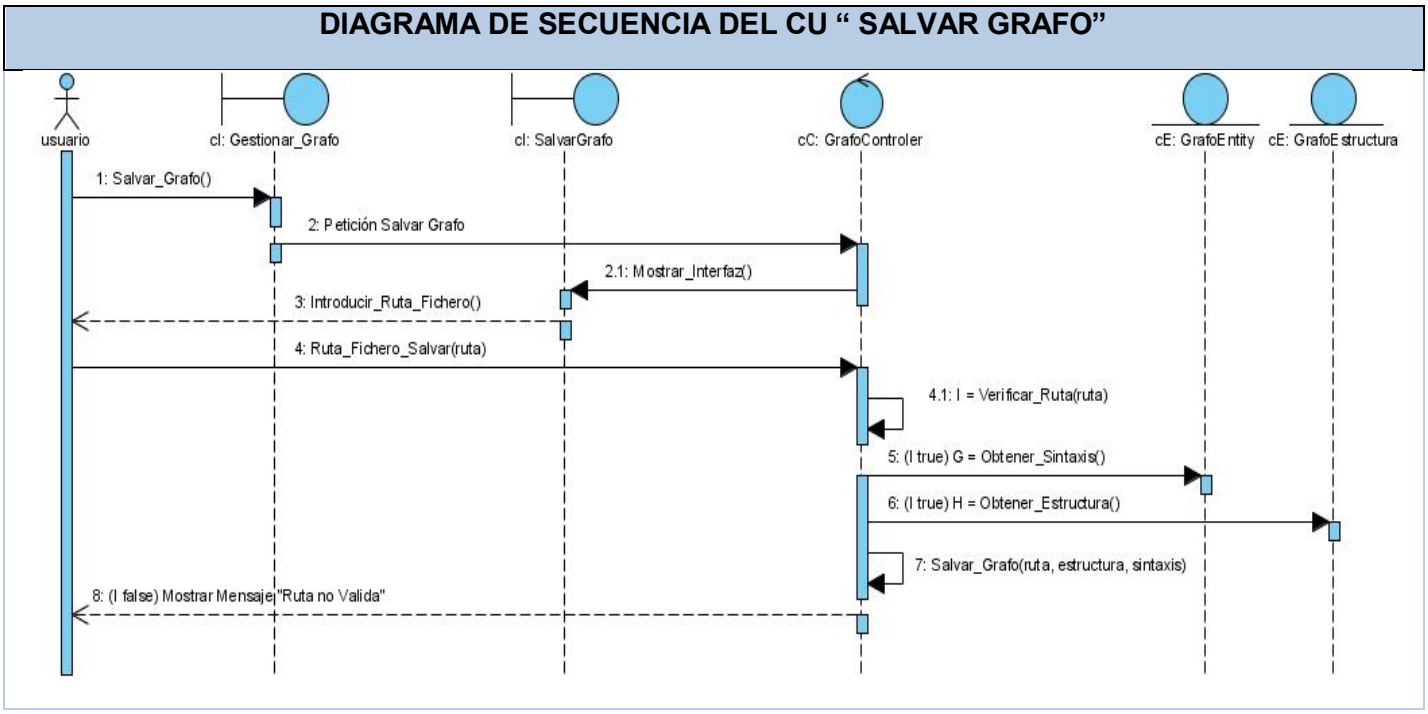


Figura 28: Diagrama de Secuencia del CU "Salvar Grafo"

2.7 Diseño

El diseño es un refinamiento del análisis que tiene en cuenta los requisitos no funcionales, o sea, cómo cumple el sistema sus objetivos. El diseño debe ser suficiente para que el sistema pueda ser implementado sin ambigüedades. En el diseño se modela el sistema y se define una arquitectura que soporte todos los requisitos. Específicamente se puede definir como propósitos del diseño: (Larman, 1999)

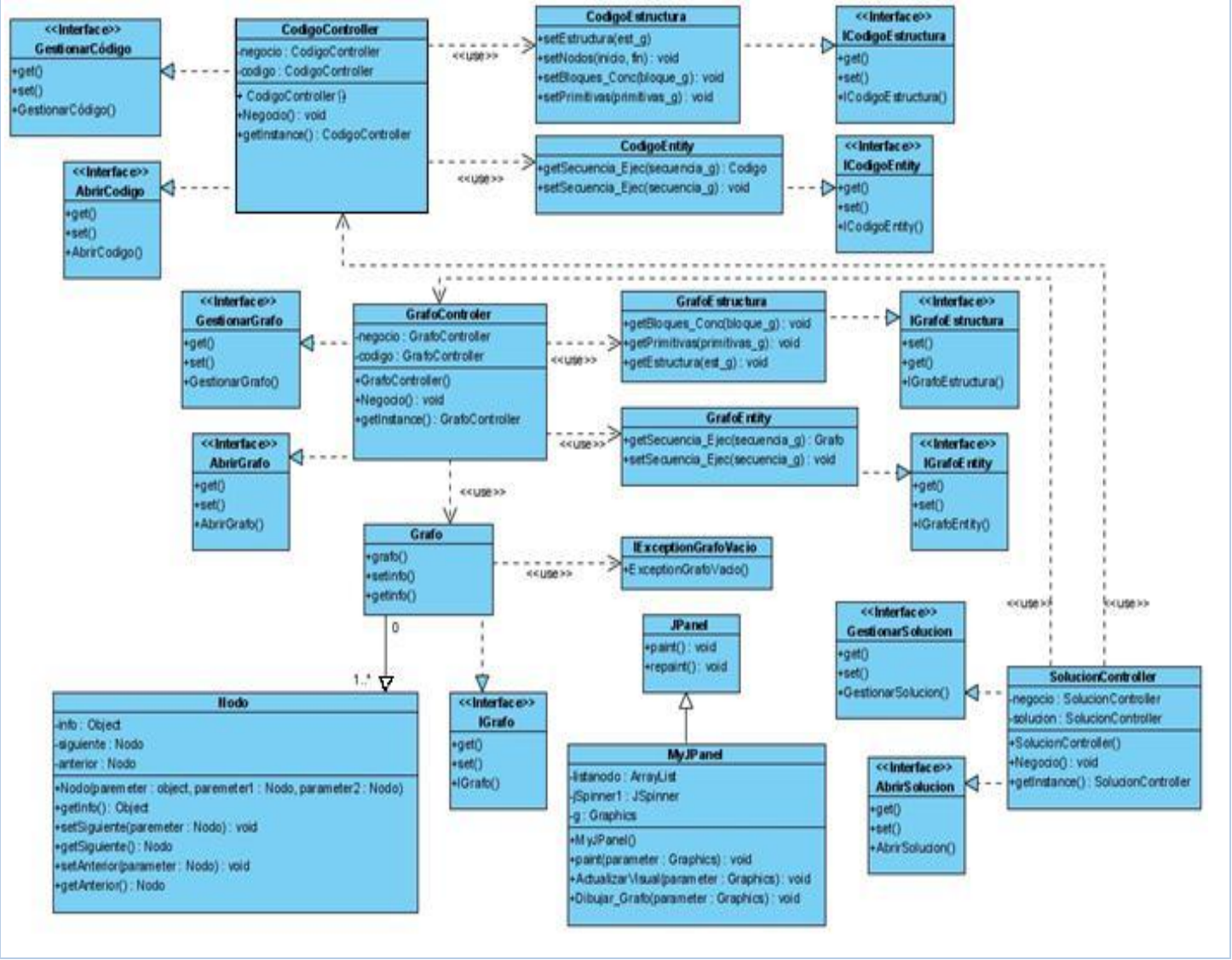
- Adquirir una comprensión de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia y tecnologías de interfaz de usuario.
- Crear una entrada apropiada y un punto de partida para actividades de implementación, capturando los requisitos o subsistemas individuales, interfaces y clases.
- Descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo.

2.7.1 Diagrama de clases del diseño

Un diagrama de clases de diseño muestra la especificación para las clases de software de una aplicación. El diagrama de clases referente al diseño del sistema se muestra a continuación.

Capítulo 2: Descripción de la Solución

DIAGRAMA DE CLASES DEL DISEÑO



2.8 Conclusiones

A través de la realización del modelo de diseño, se logró:

- Agilizar el proceso de desarrollo, generando solo la documentación necesaria que sirviera de guía para la construcción del software.
- Se desarrollaron varios artefactos como: diagramas de secuencia y diagrama de clases.

Capítulo 3: Validación y Prueba

Las métricas son el término que describen muchos y muy variados casos de medición. Siendo una medida estadística (no cuantitativa como en otras disciplinas) que se aplica a todos los aspectos de calidad de software, los cuales deben ser medidos desde diferentes puntos de vista como el análisis, construcción, funcional, documentación, métodos, proceso, usuario, entre otros. Actualmente en el proceso de desarrollo de software están presentes un conjunto de métricas, las cuales se utilizan para la validación de los requisitos identificados en la realización de un software, estas métricas permiten validar de una manera correcta que los requisitos identificados durante el proceso de desarrollo tienen la calidad requerida y cumplen con las normas internacionales. (Piña, y otros, 2008)

Existen además, métricas que permiten validar el modelo de casos de uso y el modelo de diseño.

Seguidamente se hará referencia a algunas de las métricas que fueron aplicadas a los requisitos de software, casos de uso y diagrama de clases de diseño.

3.1 Métricas de la calidad de la especificación

La validación de requisitos se realiza aplicando la métrica de la calidad de la especificación, examina las especificaciones para asegurar que todos los requisitos del sistema han sido establecidos sin ambigüedad. Para esto es necesario conocer el total de los requisitos **Rt** dado por:

Rt = Rf + Rnf

32 = 19 + 13

Dónde:

Rt: Total de requisitos.
Rf: Cantidad de requisitos funcionales.
Rnf: Cantidad de requisitos no funcionales.

Para determinar la especificidad (ausencia de ambigüedad) de los requisitos. Se explica una métrica basada en la consistencia de la interpretación de los revisores para cada requisito:
Se calcula **Q1** para determinar la especificidad de los requisitos.

Q1 = Rui / Rt

0.96 = 31/32

Dónde:

Rui: Número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas.
Q1: Ausencia de ambigüedad.

Cuanto más cerca de 1 esté el valor de **Q1**, menor será la ambigüedad de la especificación.

El valor de **Q1** = 0.96, esto demuestra que los requisitos se encuentran con un alto nivel de especificidad.

La ***estabilidad*** fue determinada con la fórmula $E = \frac{Rt - Rm}{Rt}$ donde R_m son los requisitos modificados, que es equivalente al número de requisitos. Se considera como valor óptimo para esta métrica el valor más próximo a 1.

Se sustituyen los valores:

E = (Rt – Rm) / Rt
E = (32 – 2) / 32
E = 0,94

El resultado obtenido indica que los requisitos son estables, ya que 0,94 se considera un valor de estabilidad alto, basado en el siguiente rango:

Alta (0.90 <= **E** <= 1).
Media (0.80 <= **E** < 0.90).
Baja (0.7 <= **E** < 0.80).

3.2 Métricas para validar los casos de usos del sistema

En este acápite se aplican un conjunto de métricas orientadas a objetos para evaluar las siguientes propiedades de calidad: consistencia, correctitud, completitud y complejidad. Cada uno de estos factores tendrá asociada una o más métricas, que establecen una medida cuantitativa del grado en que los factores presentan una pobre calidad. (Larman, 1999)

***Completitud*:** Grado en que se ha logrado detallar todos los casos de uso relevantes.
Consistencia: Grado en que los casos de uso del sistema describen las interacciones adecuadas entre el usuario y el sistema.
Correctitud: Grado en que las interacciones actor / sistema soportan adecuadamente el proceso del negocio.
Complejidad: Grado de claridad en la presentación de los elementos que describen el contexto y la claridad del sistema.
Para clasificar los resultados obtenidos se establecieron las siguientes reglas:

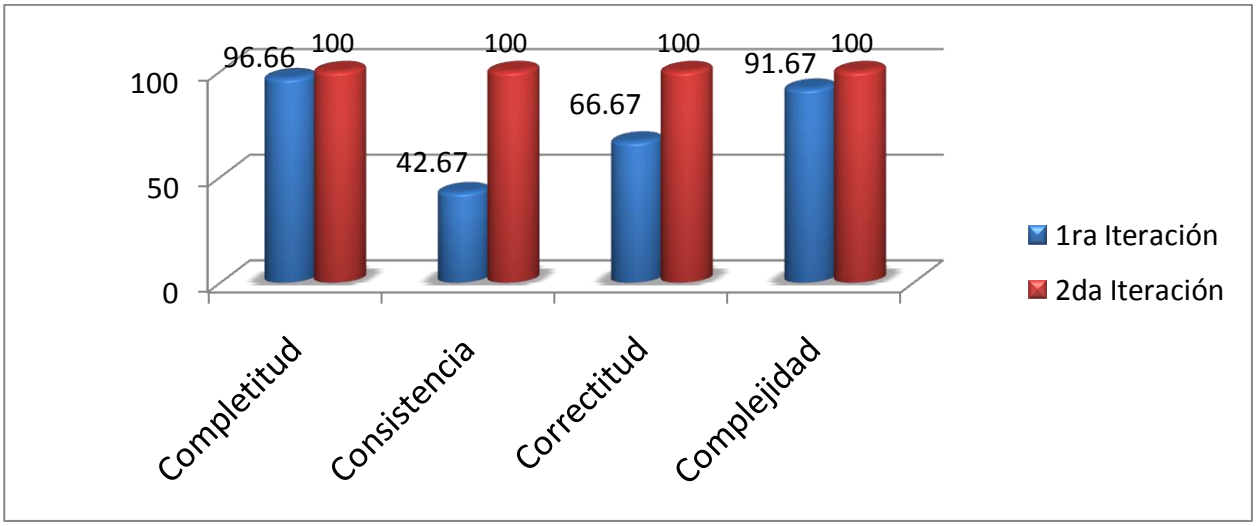
Alto (90% <= E <= 100%).
Medio (80%<= E < 90%).
Bajo (70%<= E < 80%).

Para conseguir una certera validación de los casos de uso del sistema se realizaron dos revisiones, a continuación se presentan los resultados obtenidos:

Se puede observar que en una primera iteración se obtuvo un 74.42% de funcionalidad de los casos de usos del diagrama, desglosados en un 96.66% para la completitud, un 42.67% de consistencia, 66.67% de correctitud y un 91.67% de complejidad, mostrando un bajo nivel en los resultados de la correctitud y

consistencia por lo que fue necesario realizar una segunda iteración buscando elevar el grado de funcionalidad del diagrama de casos de usos.

Gráfica de Factores de Métrica



Resultados: Como se puede apreciar en el gráfico anterior los resultados obtenidos en la segunda revisión fueron relevantes y con la aplicación de estas métricas se pudieron evaluar los factores completitud, consistencia, correctitud y complejidad del diagrama de casos de uso del sistema lo que permitió demostrar que los requisitos identificados están presentes en al menos un caso de uso, abarcando de esta forma todas las necesidades expresadas por el cliente, que los casos de uso fueron descritos detalladamente mostrando el flujo alterno a parte del flujo básico lo que da una mayor legibilidad de los mismos, evidenciándose además que estos son iniciados por la interacción del usuario con el sistema o por un evento interno dentro del mismo. A partir de los resultados se puede comprobar que el artefacto caso de uso del sistema se encuentra con la calidad requerida para entrar al flujo de trabajo de implementación donde se comenzaría la realización del software.

3.3 Métricas para validar el diseño

- **Tamaño operacional de clase (TOC)**

Las métricas empleadas están diseñadas para evaluar los siguientes atributos de calidad: (Autores, 2001)

- **Responsabilidad:** Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- **Complejidad de implementación:** Consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
- **Reutilización:** Consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.

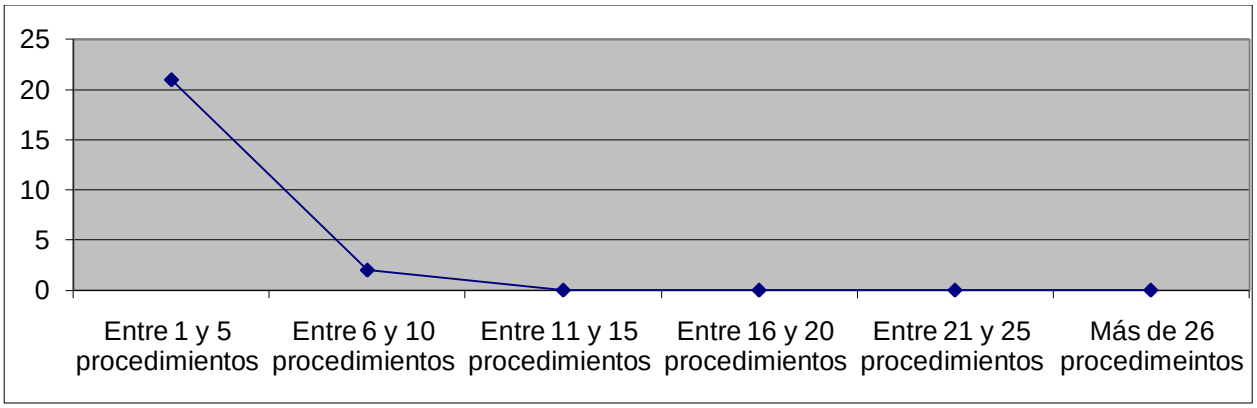
Atributos de calidad evaluados por la métrica TOC.

Atributo de Calidad.	Modo en que lo afecta.
Responsabilidad.	Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de Implementación.	Un aumento del TOC implica un aumento en la complejidad de implementación de la clase.
Reutilización.	Un aumento del TOC implica una disminución del grado de reutilización de la clase.

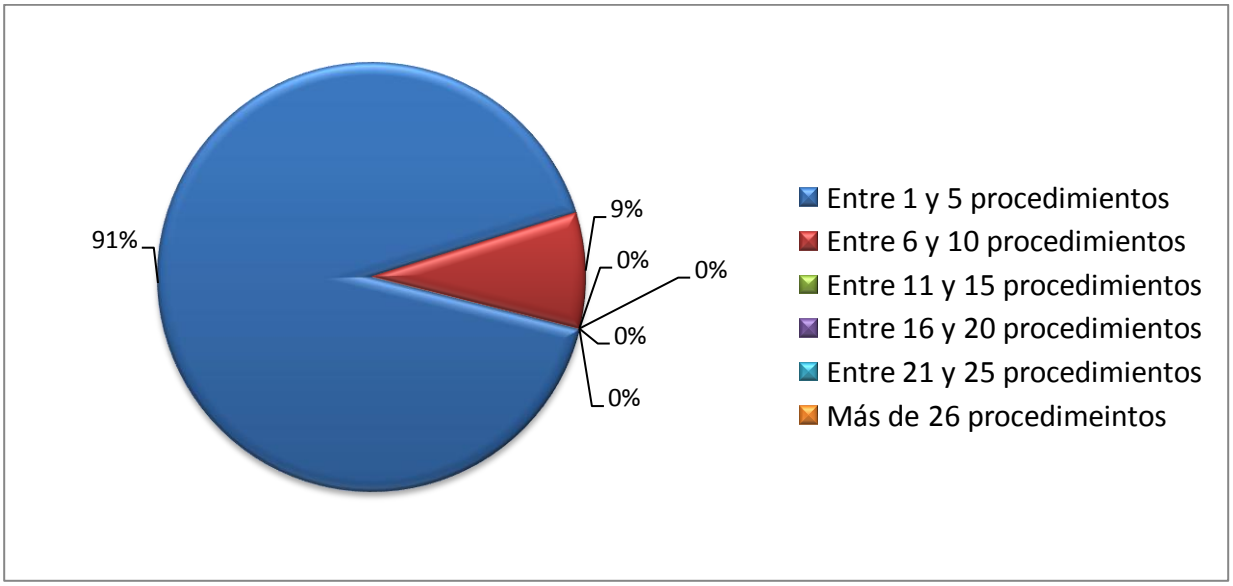
Para los cuales están definidos los siguientes criterios y categorías de evaluación:

Criterios de evaluación para la métrica TOC.

Atributo	Categoría	Criterio
Responsabilidad.	Baja.	<=Promedio
	Media.	Entre Promedio y 2*Promedio
	Alta.	>2*Promedio
Complejidad de Implementación.	Baja.	<=Promedio
	Media.	Entre Promedio y 2*Promedio
	Alta.	>2*Promedio
Reutilización.	Baja.	<=Promedio
	Media.	Entre Promedio y 2*Promedio
	Alta.	>2*Promedio

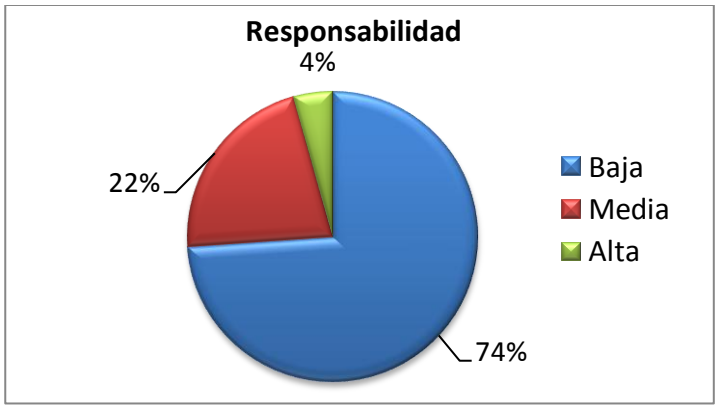


Representación de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

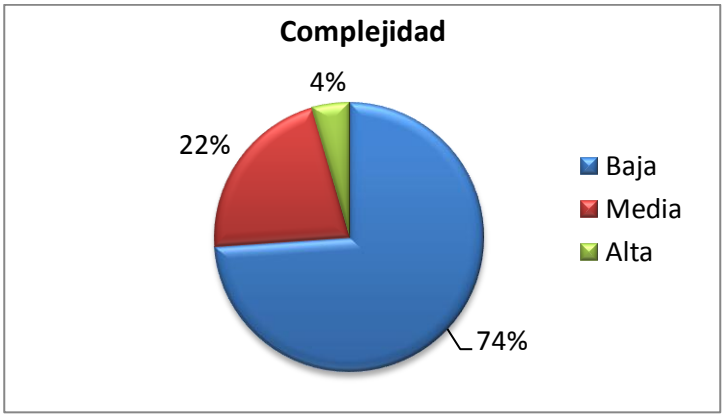


Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

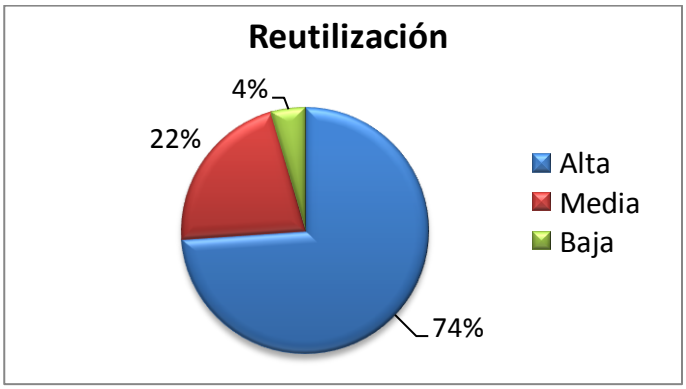
Representación de la incidencia de los resultados de la evaluación de la métrica TOC en cada uno de los atributos:



Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad



Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de Implementación.



Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización.

Al analizar los resultados obtenidos luego de aplicar el instrumento de medición de la métrica TOC, se puede concluir que el diseño propuesto está entre los límites aceptables de calidad, teniendo en cuenta que la mayoría de las clases (96%) posee menos cantidad de operaciones que la media registrada en las mediciones. Los atributos de calidad se encuentran en un nivel medio satisfactorio en el 96% de las clases; de manera que se puede observar cómo se fomenta la Reutilización (elemento clave en el proceso de desarrollo de software) y cómo están reducidas en menor grado la Responsabilidad y la Complejidad de implementación.

- **Relaciones entre clases (RC)**

Con la presente métrica se evalúan los siguientes atributos de calidad:

Responsabilidad: Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.

Complejidad del mantenimiento: Consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.

Reutilización: Consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.

Cantidad de pruebas: Consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (Unidad) del producto (componente, modulo, clase, conjunto de clases, etc.) diseñado.

Atributos de calidad evaluados por la métrica RC.

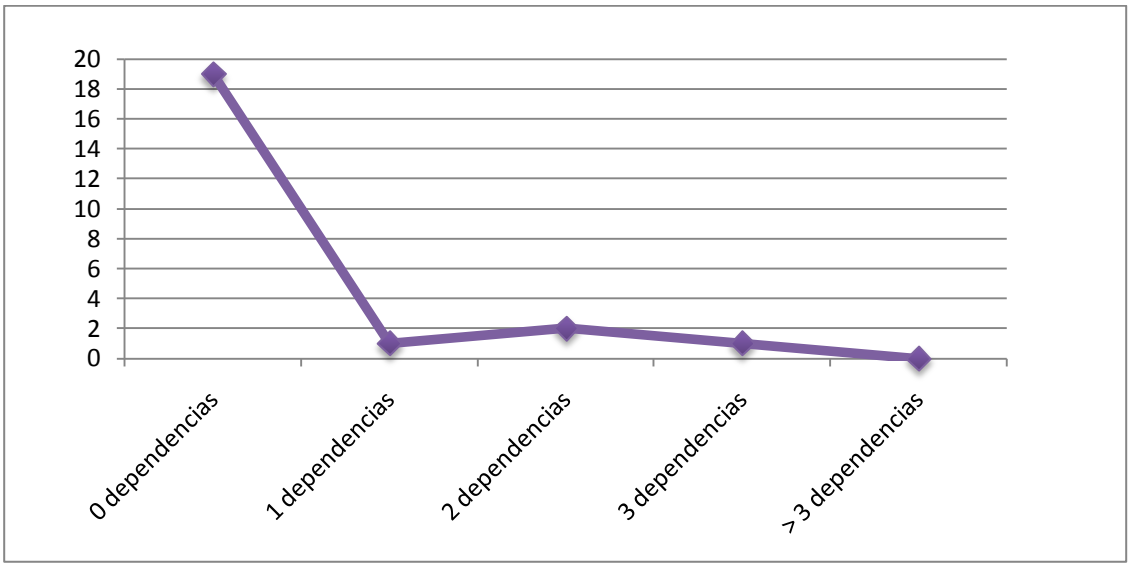
Atributo de Calidad.	Modo en que lo afecta.
Responsabilidad.	Un aumento del RC implica un aumento de la responsabilidad asignada a la clase.
Complejidad del mantenimiento.	Un aumento del RC implica un aumento en la complejidad del mantenimiento de la clase.
Reutilización.	Un aumento del RC implica una disminución del grado de reutilización de la clase.
Cantidad de pruebas.	Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Para los cuales están definidos los siguientes criterios y categorías de evaluación:

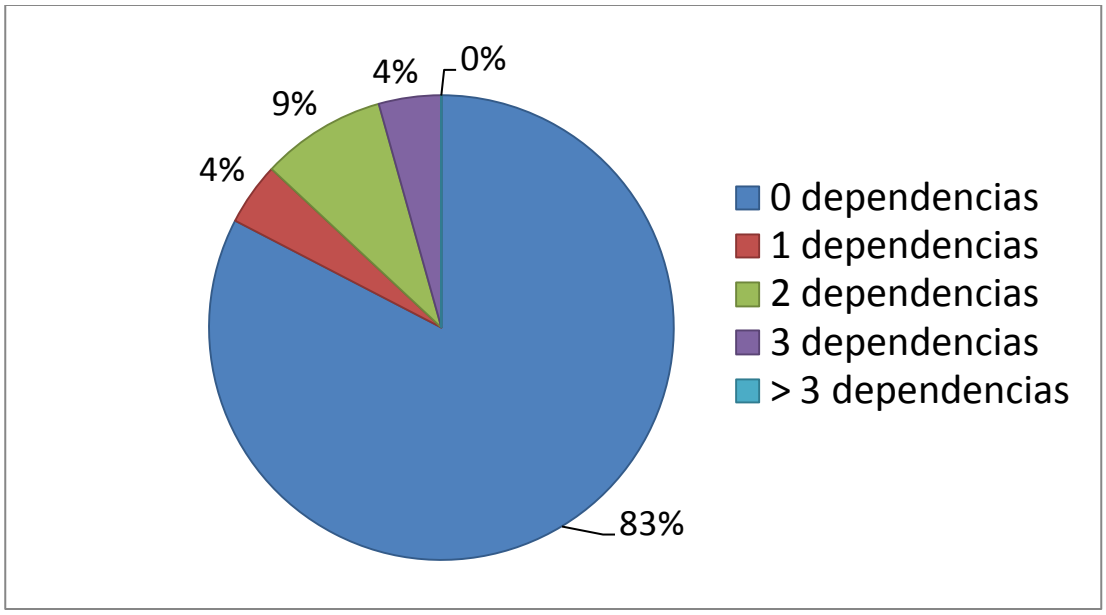
Criterios de evaluación de la métrica RC:

Atributo.	Categoría.	Criterio.
Acoplamiento.	Ninguno.	0

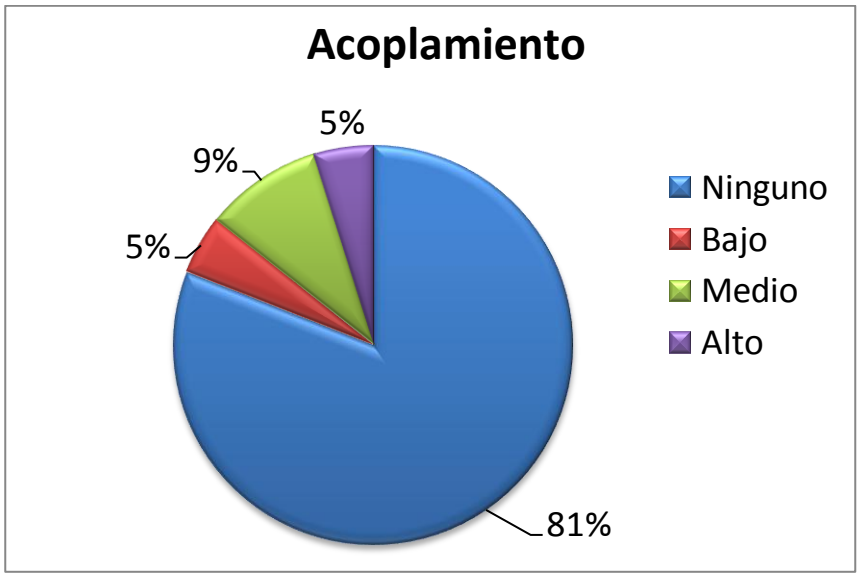
	Bajo.	1
	Medio.	2
	Alto.	>2
Complejidad de mantenimiento.	Baja.	<=Promedio
	Media.	Entre Promedio y 2*Promedio
	Alta.	>2*Promedio
Reutilización.	Baja.	>2*Promedio
	Media.	Entre Promedio y 2*Promedio
	Alta.	<=Promedio
Cantidad de pruebas.	Baja.	<=Promedio
	Media.	Entre Promedio y 2*Promedio
	Alta.	>2*Promedio



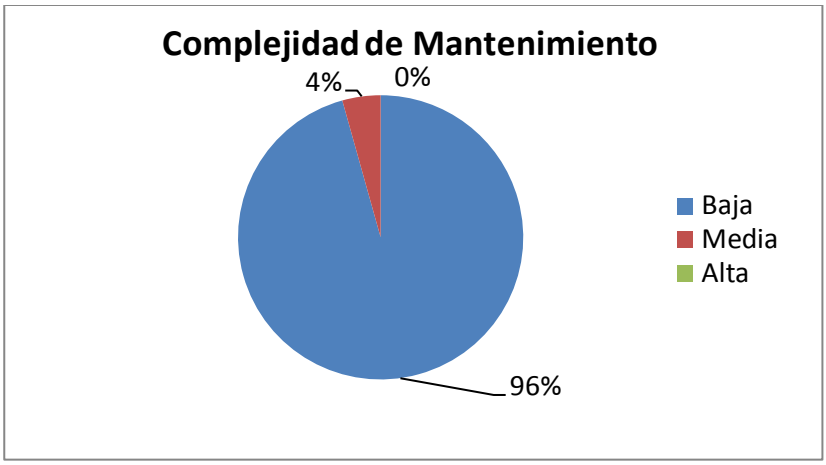
Gráfica de los resultados de la evaluación de la métrica RC agrupados por la tendencia de los Valores.



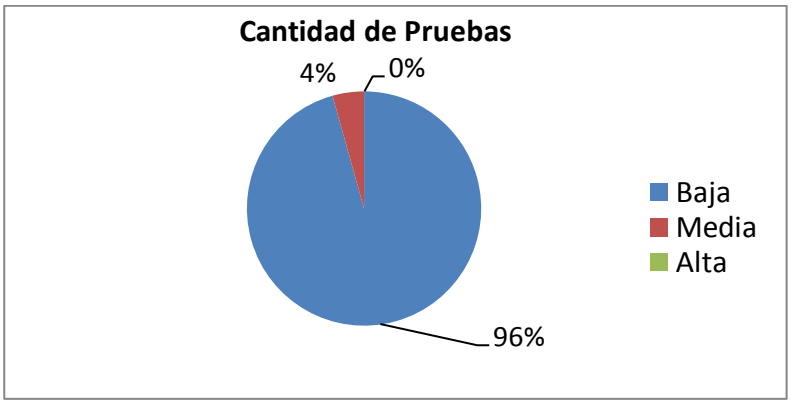
Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.



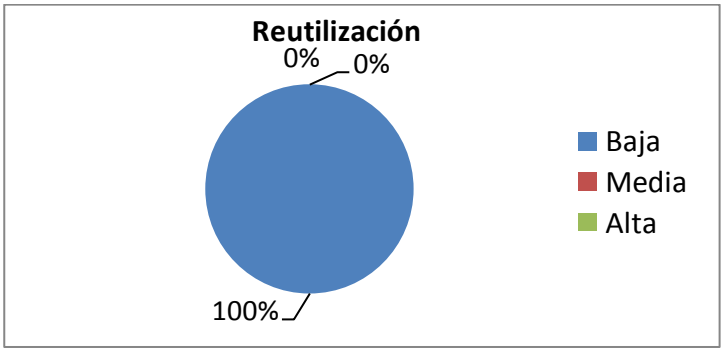
Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento.



Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento.



Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.



Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización

Al analizar los resultados obtenidos luego de aplicar el instrumento de medición de la métrica RC, se puede concluir que el diseño propuesto está entre los límites aceptables de calidad, teniendo en cuenta que la mayoría de las clases (83%) no poseen dependencias respecto a otras. Los atributos de calidad se encuentran en un nivel satisfactorio; en el 81% de las clases el grado de dependencia o acoplamiento es mínimo, la Complejidad de Mantenimiento, la Cantidad de Pruebas y la Reutilización se comportan favorablemente para un 96%, 96% y 100% de las clases respectivamente.

3.4 Conclusiones

En el presente capítulo se abordaron los temas que hacen alusión a la aplicación de métricas a los modelos obtenidos con el fin de validar la calidad de los mismos.

Se pudo constatar, teniendo en cuenta los resultados, que tanto los requisitos obtenidos como parte del proceso de captura de requisitos, como el diagrama de casos de uso del sistema que se encuentran dentro de los parámetros de calidad establecidos por las características de las métricas aplicadas.

A partir de la aplicación de las métricas para el diseño se puede afirmar que los artefactos obtenidos cuentan con un nivel óptimo de calidad lo que permitirá pasar a la fase de implementación.

Conclusiones

Conclusiones

- Con el estudio de los diferentes laboratorios virtuales, los conceptos fundamentales asociados al tema de investigación y la identificación de las necesidades en la asignatura se lograron identificar los principales requerimientos del sistema en cuestión.
- A partir del uso del Visual Paradigm para el modelado y siguiendo lo establecido por RUP como metodología y por UML y BPMN como lenguajes se lograron obtener los artefactos establecidos para las fases de Análisis y Diseño aplicando una estrategia para la construcción los modelos desde un punto de vista flexible y escalable a partir de la implantación de patrones, arquitectura y estándares.
- Con la aplicación de patrones para la obtención del diagrama de casos de uso y del diagrama de clases del diseño y de métricas a los requisitos funcionales, casos de uso del sistema y al modelo de diseño se pudo comprobar la calidad que presentaban los mismos, obteniéndose resultados satisfactorios y de este modo validar los resultados obtenidos.

Recomendaciones

Recomendaciones

Se recomienda para el seguimiento de la investigación:

- Realizar la implementación del Simulador para la programación de grafos de precedencia utilizando la estructura Parbegin/Parend mediante el uso de las primitivas P() y V() con semáforos para el Laboratorio Virtual de Sistemas Operativos, basándose en los artefactos obtenidos.

- Continuar perfeccionando los resultados obtenidos en la investigación mediante la actualización de los cambios que sean necesarios durante las etapas de implementación y pruebas.

Bibliografía

Bibliografía

1. **Almaguer, Anileidy Basso Mesa y Dayaima Gómez. 2009.** *Sistema automatizado para el proceso de caracterización de los estudiantes.* Cuidad de la Habana : s.n., 2009.
2. **Autores, Colectivo de. 2001.** Metricas para el diseño de software. [En línea] 2001. [Citado el: 21 de Enero de 2011.] www.infor.uva.es/~manso/calidad/metricasoo-2011.pdf.
3. **Barriento, Manuel Sánchez. 2008.** BPMN desventajas. [En línea] 2 de noviembre de 2008. [Citado el: 4 de noviembre de 2010.] http://www.aprendergratis.com/stag/bpmn-desventajas.html..
4. **Beck, Kent. 1999.** *Planning Extreme Programming.* s.l. : ISBN 0-201-71091-9, 1999.
5. **Borges, Rasiel Aponcio. 2009.** *Migración de la Capa de Acceso a Datos.* Cuidad de la Habana : s.n., 2009.
6. **Brooks, Frederick. 1995.** *The Mythical Man-Month: Essays on Software Engineering.* 1995.
7. **Cárdenas, Marina Elizabeth. 2009.** *Software de Simulación aplicado a entornos de e-learning.* 2009.
8. **Dávila, Nicolás. 2001.** *Ingeniería de Requerimientos una guía para extraer, analizar, especificar y validar los requerimientos de un proyecto.* 2001.
9. **Estrada, Yelena Hernadez. 2009.** Análisis del Módulo Proceso Confiscatorio de Bienes del proyecto Sistema Gestión Fiscal. Ciudad Habana : s.n., junio de 2009.
10. **Falgueras, Benet Campderrich. 2003.** *Ingenieria de Software.* Barcelona : Editoria UOC, edicion Aragon 182 08011, 2003.
11. **G. Booch, J. Rumbaugh, I. Jacobson. 2000.** *El proceso unificado de desarrollo de software.* s.l. : Addison Wesley, 2000.

Bibliografía

12. **Hanrahan, Robert P. 1999.** The IDEF0 Process Modeling Methodology. *Software Technology Support Center.* [En línea] 1999. [Citado el: 2010 de Noviembre de 5.]

13. **Herías, Francisco Andrés Candelas. 2003.** Propuesta de Portal de la Red de Laboratorios Virtuales y Remotos de CEA. [En línea] 27 de noviembre de 2003. [Citado el: 20 de octubre de 2010.] http://www.disc.ua.es/docenweb/Docs/PropuestaDePortal.pdf..

14. **Herreros, L. Rosado y R., J. 2005.** Nuevas aportaciones didácticas de los laboratorios virtuales y remotos en la enseñanza de la Física. [En línea] 2005. [Citado el: 19 de octubre de 2010.] http://www.formatex.org/micte2005/286.pdf..

15. **Javier Laborda, Josep Galimany, Rosa María Pena, Antoni Gual. 2000.** *Biblioteca práctica de la computación.* Barcelona : Ediciones Océano-Éxito, 2000.

16. **José Manuel Ruiz Gutiérrez. 2000.** La Simulación como Instrumento de Aprendizaje. [En línea] 2000. [Citado el: 5 de Febrero de 2010.] http://mami.uclm.es/jmruiz/materiales/Documentos/simulacion.PDF.

17. **Larman, Craig. 1999.** *UML y Patrones: Introducción al análisis y diseño orientado a objetos.* s.l. : Prentice Hall, 1999.

18. **Luis R. Izquierdo, José M. Galán, José I. Santos, Ricardo del Olmo. 2008.** Modelado de sistemas complejos mediante simulación basada en agentes y mediante dinámica de sistemas. [En línea] 2008. [Citado el: 21 de Enero de 2011.] http://www.luis.izqui.org/papers/Izquierdo_Galan_Santos_Olmo_2008.pdf.

19. *Mapas Conceptuales.* **Dürsteler, Juan C. 2004.** 114, s.l. : Inf@Vis, 2004.

20. **Mendoza, maria A. Sánchez. 2004.** Metodologías de Desarrollo de Software. [En línea] 7 de junio de 2004. [Citado el: 5 de noviembre de 2010.]

21. **Nájera Estrada, Julián Monge y Méndez, Víctor Hugo. 2007.** Ventajas y desventajas de usar laboratorios Vituales en educación a distancia: la opinión del estudiantado en un proyecto de seis

Bibliografía

años de duración. la opinión del estudiantado en un proyecto de seis años de duración. [En línea] 2007. [Citado el: 3 de noviembre de 2010.] http://www.latindex.ucr.ac.cr/edu31-1/edu-31-1-05.pdf..

22. **Oca, E. C. 2009.** Análisis y Diseño de una herramienta interactiva de simulación de procesos: Nodo Virtual de Procesos. Segunda Iteración. . Ciudad Habana, Cuba : s.n., junio de 2009.

23. **Orallo, Enrique Hernández. 2007.** El lenguaje Unificado de Modelado (UML). [En línea] 2007. [Citado el: 25 de septiembre de 2010.]

24. **Piña, Lianyi Ramos y Pulido, Yanelis. 2008.** *Análisis de los módulos Planificanción de Disco y Administración de Memoria de un Laboratorio Virtual de apoyo a la asignatura de SIstemas Operativos.* Habana, Cuba : Universisdad de Las Ciencias Informáticas, 2008.

25. **Pressman, Roger S. 2005.** *Ingeniería del software. Un enfoque práactico.* La Habana : Felix Varela, 2005.

26. **Sierra, Daniel. 2011.** www.slideshare.net. [En línea] www.slideshare.net, 2011. [Citado el: 14 de Junio de 2011.] http://www.slideshare.net/vanquishdarkenigma/visual-paradigm-for-uml.

27. **Sparxsystems. 2008.** www.sparxsystems.com. [En línea] 2008. [Citado el: 14 de Junio de 2011.] http://www.sparxsystems.com.ar.

28. **Tanenbaum, Andrews S. 1997.** *Sistemas Operativos. Diseño e Implementación.* s.l. : Prentice Hall, 1997.

29. **Vary, James P. 2000.** *Informe de la reunión de expertos.* 2000.

30. **Velázquez, Eugenio. 2008.** Laboratorios Virtuales de TechNet: una forma diferente de probar las soluciones Microsoft. [En línea] 2008. [Citado el: 5 de Noviembre de 2010.] http://www.tecnologiapyme.com/software/laboratorios-virtuales-de-technet-una-forma-diferente-de-probar-las-soluciones-microsoft.

Anexo 1: Descripción textual de los casos de usos

Descripción del Caso de Uso: "Cargar Código"

Caso de uso:	Cargar Código
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario selecciona la opción "Cargar Código" en el menú principal y termina cuando el código es visualizado en la aplicación
Precondiciones:	-
Pos condición:	El usuario podrá visualizar el código cargado en el sistema.
Casos de usos asociados:	-
Referencia:	RF04, RF05, RF06 y RF07
Prioridad:	Normal
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso comienza cuando el usuario selecciona la opción Cargar Código	2. Muestra la interfaz correspondiente: - *Cargar Código*
3. Selecciona el código a cargar y oprime el botón *"Aceptar*".	4. Leer contenido del fichero seleccionado 5. Verificar estructura del Código 6. Chequear sintaxis 7. Chequear orden de ejecución 8. Chequear orden de primitivas según orden de ejecución 9. Muestra el código cargado y culmina el caso de uso
Flujo Alterno 1: Verificar estructura del Código	
Acción del Actor	**Respuesta del Sistema**
	5. Si la estructura no es válida el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 2: Chequear Sintaxis	
Acción del Actor	**Respuesta del Sistema**
	6. Si la sintaxis no corresponde con las reglas definidas el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 3: Chequear Orden de Ejecución	
Acción del Actor	**Respuesta del Sistema**
	7. Si el orden de ejecución no corresponde con la estructura del código y con las reglas de la sintaxis el sistema emitirá un mensaje de error,

	regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 4: Chequear orden de primitivas según orden de ejecución	
Acción del Actor	**Respuesta del Sistema**
	8. Si las primitivas definidas en el código no están ubicadas en la posición correcta según la estructura del código el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.

Descripción del Caso de Uso: "Cargar Grafo"

Caso de uso:	Cargar Grafo
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario selecciona la opción "Cargar Grafo" en el menú principal y termina cuando el grafo es visualizado en la aplicación
Precondiciones:	-
Pos condición:	El usuario podrá visualizar el Grafo cargado en el sistema.
Casos de usos asociados:	-
Referencia:	RF01, RF06, RF08, RF09, RF12
Prioridad:	Normal
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso comienza cuando el usuario selecciona la opción Cargar Grafo	2. Muestra la interfaz correspondiente: - *Cargar Grafo*
3. Selecciona el fichero a cargar y oprime el botón *"Aceptar*".	4. Leer contenido del fichero seleccionado 5. Identificar nodo inicio y nodo fin 6. Chequear estructura lógica 7. Determinar bloques concurrentes 8. Determinar ubicación de primitivas 9. Establecer secuencia de ejecución 10. Muestra el grafo cargado y culmina el caso de uso
Flujo Alterno 1: Identificar nodo inicio y nodo fin	
Acción del Actor	**Respuesta del Sistema**
	5. Si los datos no son válidos el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 2: Chequear estructura lógica	
Acción del Actor	**Respuesta del Sistema**

	6. Si la estructura no coincide con las reglas definidas el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 3: Determinar bloques concurrentes	
Acción del Actor	**Respuesta del Sistema**
	7. Si los bloques definidos no coinciden con la estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 4: Determinar ubicación de primitivas	
Acción del Actor	**Respuesta del Sistema**
	8. Si las primitivas no se encuentran ubicadas según definición de bloques concurrentes y estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 5: Establecer secuencia de ejecución	
Acción del Actor	**Respuesta del Sistema**
	9. Si la secuencia de ejecución no está correctamente delimitada o no coincide con la estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.

Descripción del Caso de Uso: "Cargar Solución"

Caso de uso:	Cargar Solución
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario selecciona la opción "Cargar Solución" en el menú principal
Precondiciones:	-
Pos condición:	El usuario podrá visualizar la solución cargada en el sistema.
Casos de usos asociados:	-
Referencia:	RF01, RF04, RF07, RF08, RF09
Prioridad:	Normal
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Seleccionar opción Cargar Solución	2. Muestra la interfaz *"Abrir"* para que el usuario localice la Solución a cargar
3. Selecciona el fichero a cargar y	4. Leer contenido del fichero seleccionado 5. Identificar nodo inicio y fin

oprime el botón *"Aceptar"*.	6. Establecer secuencia de ejecución 7. Determinar bloques concurrentes 8. Determinar ubicación de primitivas 9. Chequear estructura lógica 10. Muestra la solución cargada y culmina el caso de uso
Flujo Alterno 1: Identificar nodo inicio y fin	
Acción del Actor	**Respuesta del Sistema**
	5. Si los datos no son válidos el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 2: Establecer secuencia de ejecución	
Acción del Actor	**Respuesta del Sistema**
	6. Si la secuencia de ejecución no está correctamente delimitada o no coincide con la estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 3: Determinar bloques concurrentes	
Acción del Actor	**Respuesta del Sistema**
	7. Si los bloques definidos no coinciden con la estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 4: Determinar ubicación de primitivas	
Acción del Actor	**Respuesta del Sistema**
	8. Si las primitivas no se encuentran ubicadas según definición de bloques concurrentes y estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 5: Chequear estructura lógica	
Acción del Actor	**Respuesta del Sistema**
	9. Si la estructura no coincide con las reglas definidas el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.

Descripción del Caso de Uso: "Chequear Código"

Caso de uso:	Chequear Código
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario selecciona la opción "Chequear Código" en el menú principal y termina cuando el sistema devuelve el mensaje "Código Correcto"
Precondiciones:	Se debe verificar que el grafo existe y que el usuario ha escrito el código

	a chequear
Pos condición:	El sistema devuelve un mensaje de código correcto en caso de que coincida con el grafo. En caso contrario se emitirá un mensaje de error
Casos de usos asociados:	Tiene incluido el caso de uso Inicializar Código
Referencia:	RF04, RF05, RF06, RF07
Prioridad:	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso comienza cuando el usuario selecciona la opción Chequear Código	2. Verificar estructura del código 3. Chequear sintaxis 4. Chequear orden de ejecución 5. Chequear posición de primitivas según orden de ejecución 6. Emitir mensaje “código correcto”, regresa al menú principal y culmina el caso de uso
Flujo Alterno 1: Verificar estructura del Código	
Acción del Actor	**Respuesta del Sistema**
	2. Si la estructura no es válida el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 2: Chequear Sintaxis	
Acción del Actor	**Respuesta del Sistema**
	3. Si la sintaxis no corresponde con las reglas definidas el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 3: Chequear Orden de Ejecución	
Acción del Actor	**Respuesta del Sistema**
	4. Si el orden de ejecución no corresponde con la estructura del código y con las reglas de la sintaxis el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 4: Chequear orden de primitivas según orden de ejecución	
Acción del Actor	**Respuesta del Sistema**
	5. Si las primitivas definidas en el código no están ubicadas en la posición correcta según la estructura del código el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.

Descripción del Caso de Uso: "Inicializar Código"

Caso de uso:	Inicializar Código
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario selecciona la opción "Generar Código" o "Chequear Código" en el menú principal y termina cuando los parámetros de inicialización del código a generar o chequear quedan establecidos.
Precondiciones:	El usuario debe haber seleccionado la opción "Generar Código" o "Chequear Código"
Pos condición:	Se reserva memoria y se establece los parámetros iniciales necesarios para la ejecución de la acción
Casos de usos asociados:	-
Referencia:	RF01, RF02, RF03
Prioridad:	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso comienza cuando el usuario selecciona la opción "Generar Código" o "Chequear Código"	2. Identificar nodo inicio y fin 3. Verificar posibles bloques concurrentes 4. Identificar primitivas y semáforos 5. Inicializa variables de semáforos y culmina el caso de uso
Flujo Alterno 1: Identificar nodo inicio y fin	
Acción del Actor	**Respuesta del Sistema**
	2. Si los datos no son válidos el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 2: Verificar posibles bloques concurrentes	
Acción del Actor	**Respuesta del Sistema**
	3. Si los bloques definidos no coinciden con la estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 3: Identificar primitivas y semáforos	
Acción del Actor	**Respuesta del Sistema**
	4. Si las primitivas no se encuentran ubicadas según definición de bloques concurrentes y estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.

Descripción del Caso de Uso: "Chequear Grafo"

Caso de uso:	Chequear Grafo
Actores:	Usuario

Resumen:	El caso de uso se inicia cuando el usuario selecciona la opción "Chequear Grafo" en el menú principal y termina cuando el sistema devuelve el mensaje "Grafo Correcto"
Precondiciones:	Se debe verificar que el grafo existe y que el usuario ha introducido el código a chequear
Pos condición:	El sistema devuelve un mensaje de "Grafo Correcto"
Casos de usos asociados:	Tiene incluido el caso de uso Inicializar Grafo
Referencia:	RF01, RF06, RF07, RF08, RF09
Prioridad:	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1, El caso de uso comienza cuando el usuario selecciona la opción Chequear Grafo	2. Verificar nodo inicial y final 3. Verificar orden de ejecución 4. Chequear definición de bloques concurrentes 5. Chequear ejecución de primitivas 6. Emite mensaje "Grafo Correcto" y termina el caso de uso
Flujo Alterno 1: Verificar nodo inicial y final	
Acción del Actor	**Respuesta del Sistema**
	2. Si los datos no son válidos el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 2: Verificar orden de ejecución	
Acción del Actor	**Respuesta del Sistema**
	3. Si el orden de ejecución no corresponde con la estructura del código y con las reglas de la sintaxis el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 3: Chequear definición de bloques concurrentes	
Acción del Actor	**Respuesta del Sistema**
	4. Si los bloques definidos no coinciden con la estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 4: Chequear ejecución de primitivas	
Acción del Actor	**Respuesta del Sistema**
	5. Si las primitivas no se encuentran ubicadas según definición de bloques concurrentes y estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.

Descripción del Caso de Uso: "Generar Grafo"

Caso de uso:	Generar Grafo
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario selecciona la opción "Generar Grafo" en el menú principal y termina cuando el grafo generado es visualizado en la aplicación
Precondiciones:	Se debe verificar que el código para la generación del grafo existe
Pos condición:	El sistema devuelve un mensaje de grafo generado y muestra el grafo obtenido en el área correspondiente
Casos de usos asociados:	Tiene incluido el caso de uso Inicializar Grafo
Referencia:	RF01 , RF02, RF04, RF11, RF12
Prioridad:	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso comienza cuando el usuario selecciona la opción Generar Grafo	2. Validar Estructura del Código 3. Identificar Nodo Inicio y Nodo Fin 4. Identificar Concurrencia 5. Definir Semáforos 6. Determinar Posición de Primitivas 7. Generar Grafo, 8. Visualizar grafo generado, emite un mensaje de "Grafo Generado Correctamente" y culmina el caso de uso.
Flujo Alterno 2: Validar Estructura del Código	
Acción del Actor	**Respuesta del Sistema**
	1. Si la estructura no coincide con las reglas definidas el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 3: Identificar Nodo Inicio y Nodo Fin	
Acción del Actor	**Respuesta del Sistema**
	2. Si los datos no son válidos el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 4: Identificar Concurrencia	
Acción del Actor	**Respuesta del Sistema**
	3. Si los bloques definidos no coinciden con la estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.

Flujo Alterno 6: Determinar Posición de Primitivas	
Acción del Actor	**Respuesta del Sistema**
	5. Si las primitivas no se encuentran ubicadas según definición de bloques concurrentes y estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.

Descripción del Caso de Uso: "Inicializar Grafo"

Caso de uso:	Inicializar Grafo
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario selecciona la opción "Generar Grafo" o "Chequear Grafo" en el menú principal y termina cuando los parámetros de inicialización del grafo a generar o chequear quedan establecidos.
Precondiciones:	El usuario debe haber seleccionado la opción "Generar Grafo" o "Chequear Grafo"
Pos condición:	Se reserva memoria y se establecen los parámetros iniciales necesarios para la ejecución del sistema
Casos de usos asociados:	-
Referencia:	RF06, RF12, RF13, RF14, RF15
Prioridad:	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso comienza cuando el usuario selecciona la opción "Generar Grafo" o "Chequear Grafo"	2. Establecer nodo inicio y fin 3. Establecer bloques concurrentes 4. Determinar sincronización 5. Chequear ubicación de primitivas 6. Chequear secuencia de ejecución 7. Inicializa cantidad de nodos a generar, posición según secuencia de ejecución, variables de precedencia y culmina el caso de uso
Flujo Alterno 1: Establecer nodo inicio y fin	
Acción del Actor	**Respuesta del Sistema**
	2. Si los datos no son válidos el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 2: Establecer bloques concurrentes	
Acción del Actor	**Respuesta del Sistema**
	3. Si los bloques definidos no coinciden con la estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el

	caso de uso.
Flujo Alterno 4: Chequear ubicación de primitivas	
Acción del Actor	**Respuesta del Sistema**
	5. Si las primitivas no se encuentran ubicadas según definición de bloques concurrentes y estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.
Flujo Alterno 5: Chequear secuencia de ejecución	
Acción del Actor	**Respuesta del Sistema**
	6. Si la secuencia de ejecución no está correctamente delimitada o no coincide con la estructura el sistema emitirá un mensaje de error, regresará al menú inicial y culminará el caso de uso.

Anexo 2: Diagramas de Secuencia

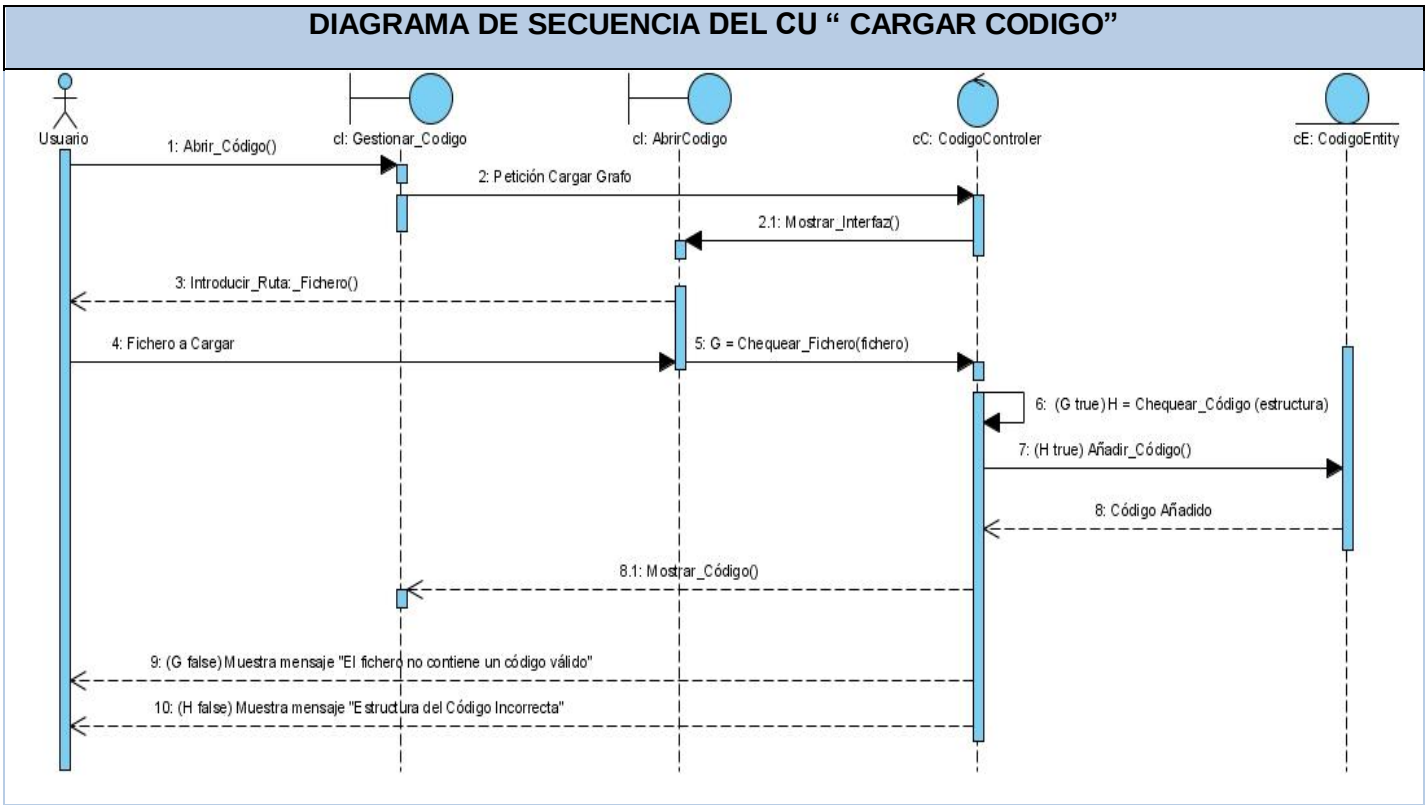


Figura 24: Diagrama de Secuencia del CU "Cargar Código"

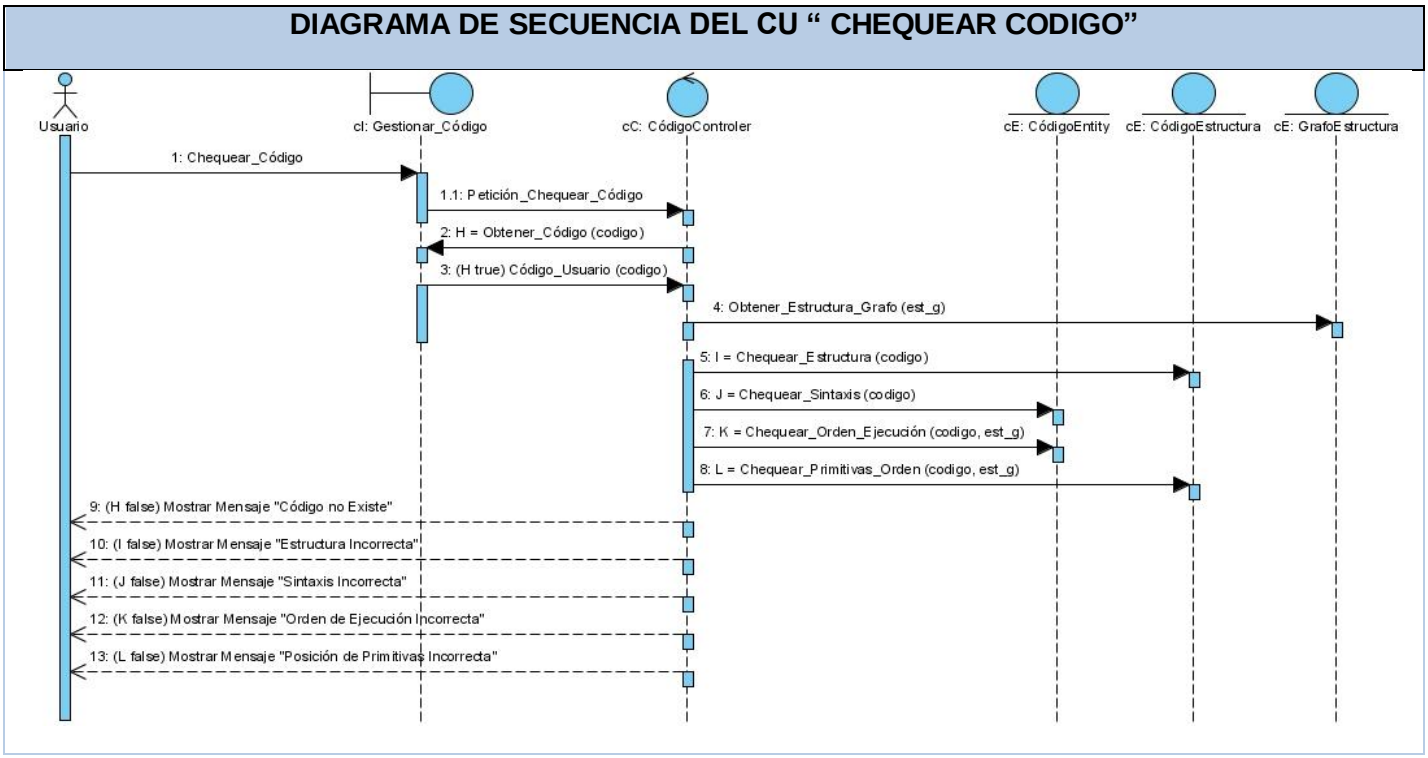


Figura 25: Diagrama de Secuencia del CU "Chequear Código"

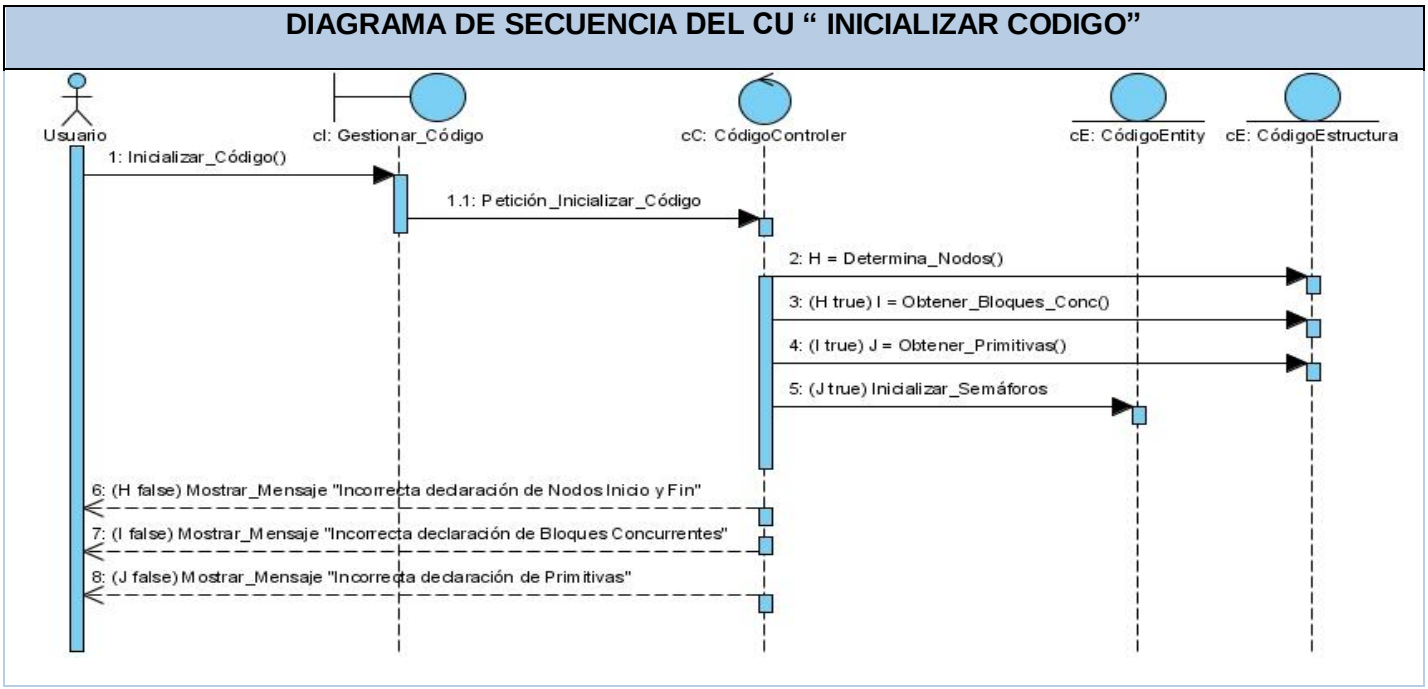


Figura 26: Diagrama de Secuencia del CU "Inicializar Código"

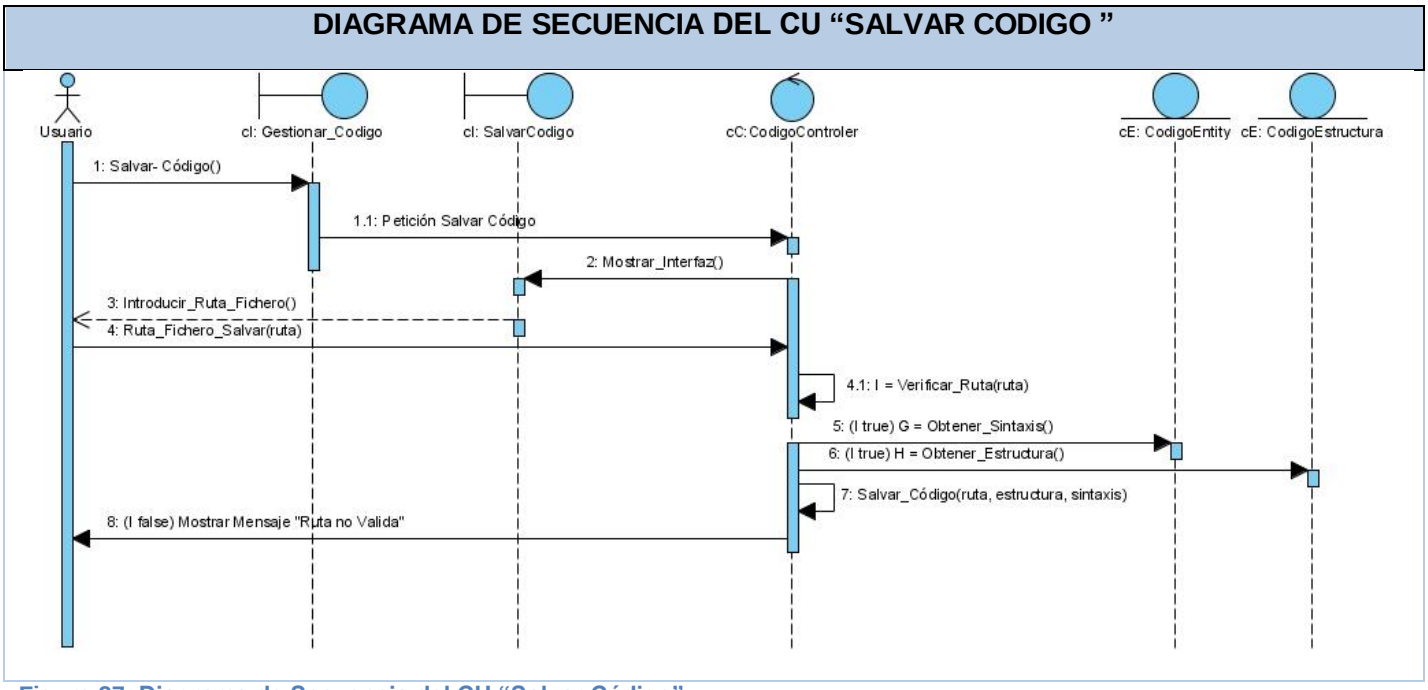


Figura 27: Diagrama de Secuencia del CU “Salvar Código”

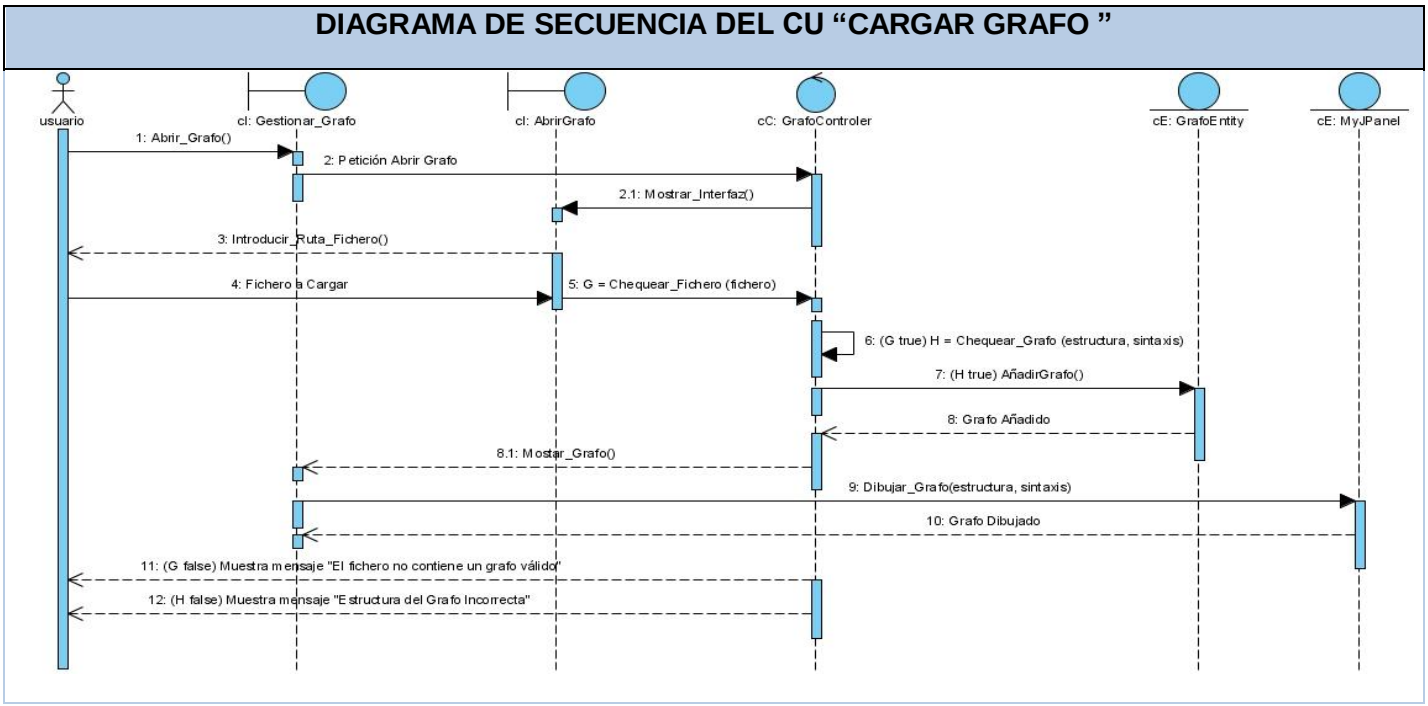


Figura 28: Diagrama de Secuencia del CU “Cargar Grafo”

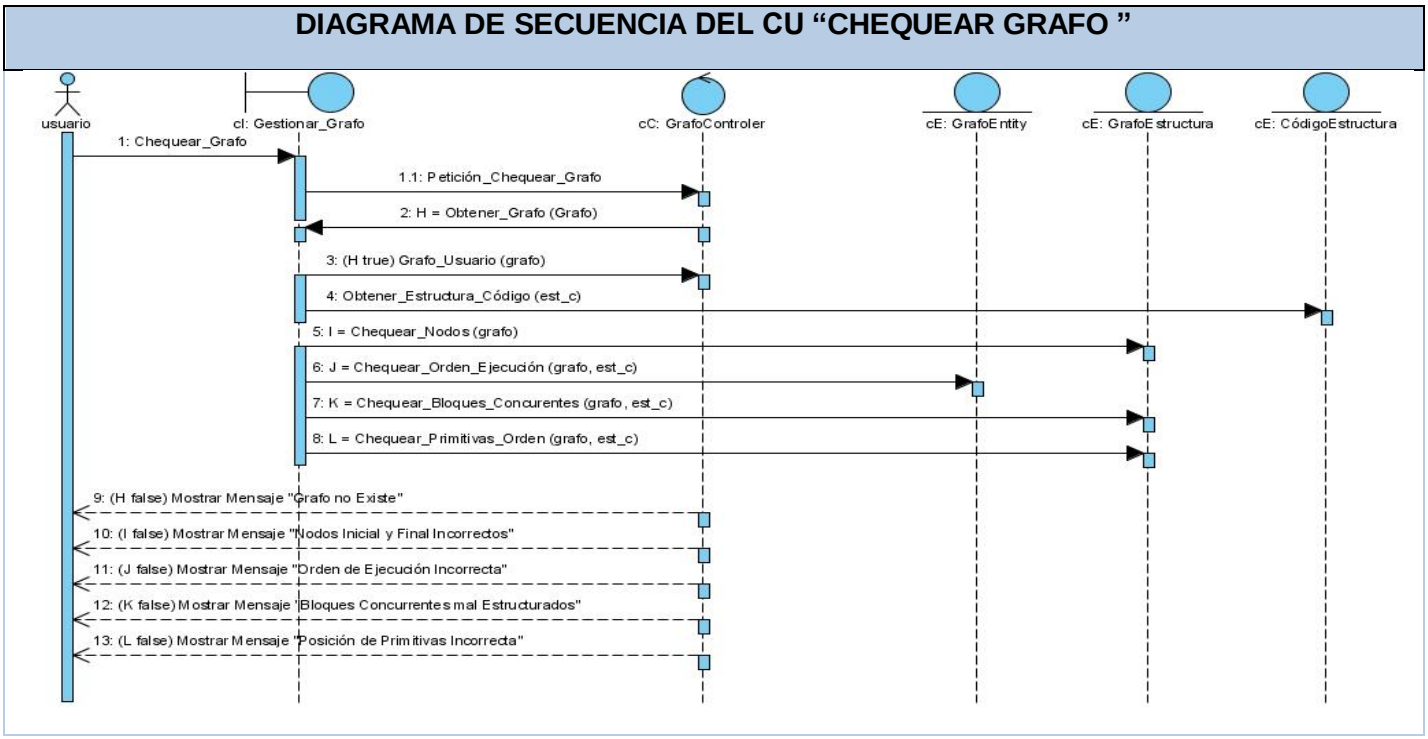


Figura 29: Diagrama de Secuencia del CU "Chequear Grafo"

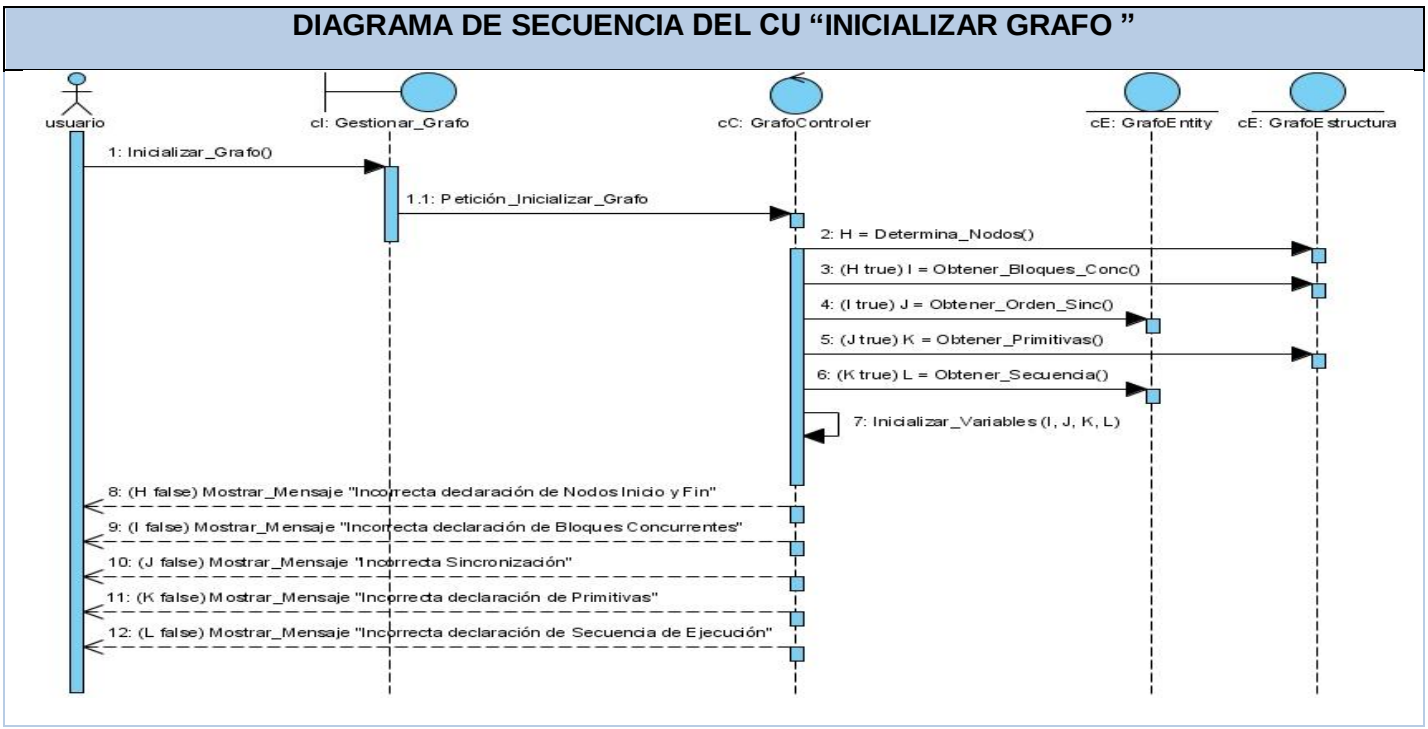


Figura 30: Diagrama de Secuencia del CU "Inicializar Grafo"

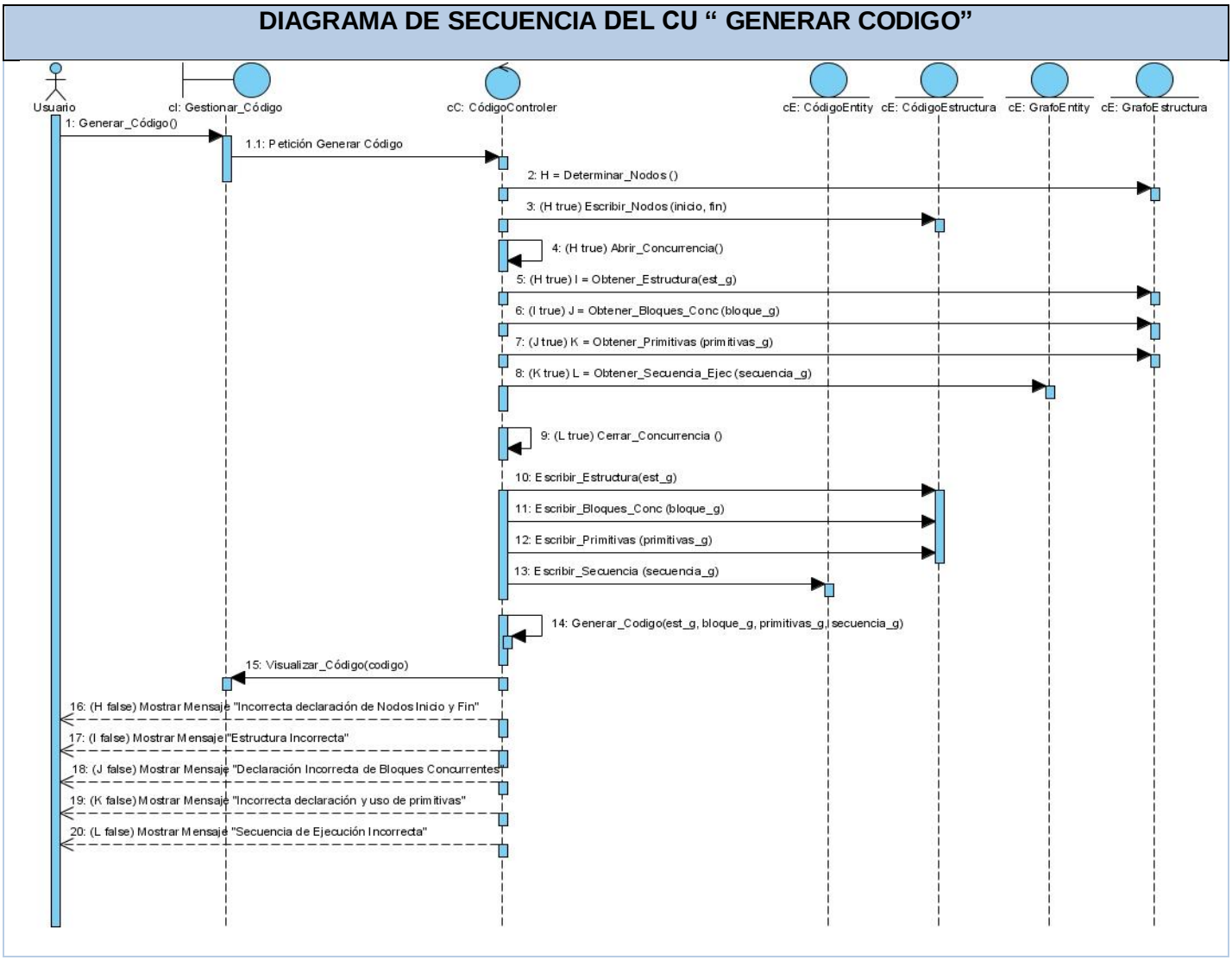


Figura 37: Diagrama de Secuencia del CU "Generar Código"

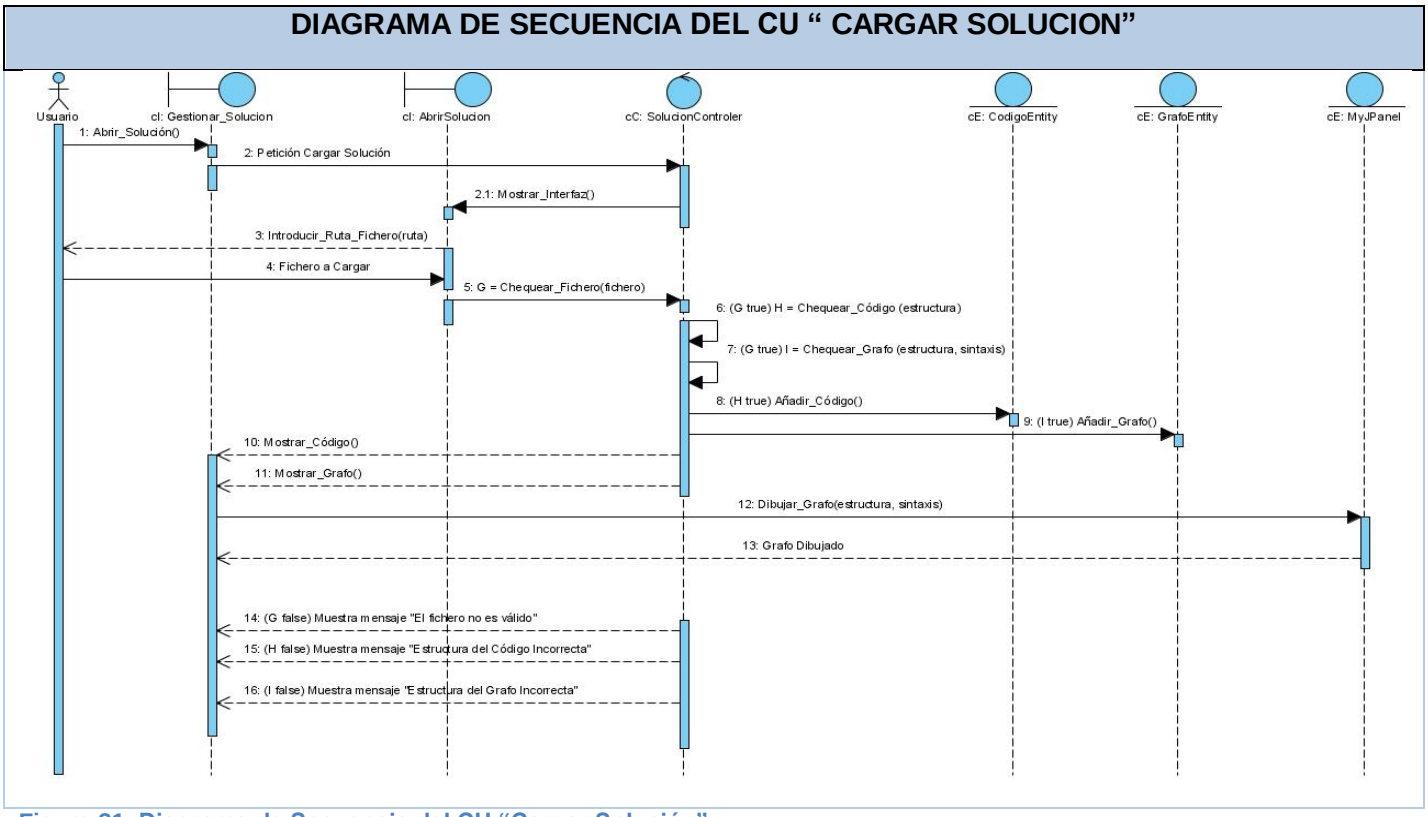


Figura 31: Diagrama de Secuencia del CU "Cargar Solución"

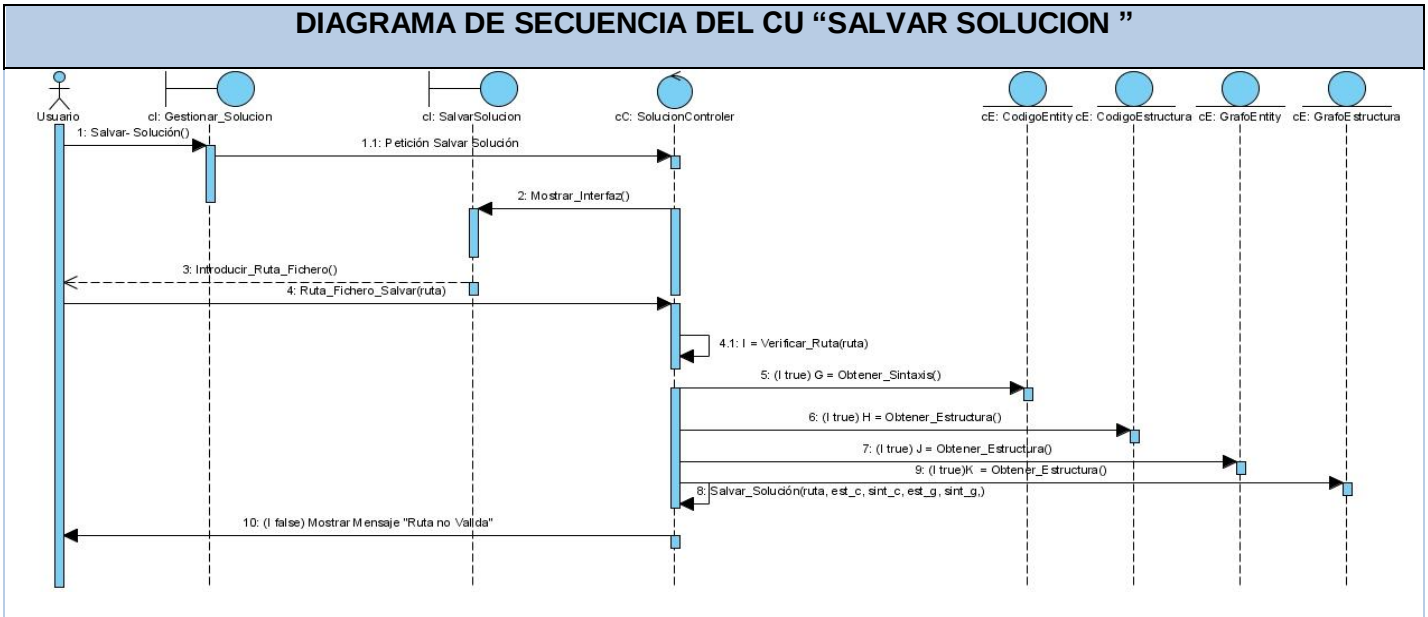


Figura 32: Diagrama de Secuencia del CU "Salvar Solución"

Anexo 3: Aplicación de las métricas para la calidad de los casos de uso

Primera Revisión:

Atributo	Factor	Métrica asociada	Valor
Completitud	Factor 1. ¿Han sido definidos todos los roles relevantes de usuario encargados de generar/ modificar o consultar información?	Métrica 1: Número de roles relevantes omitidos. Umbral < 90%	Número de roles relevantes omitidos: 0 Se presenta un 100%.
	Factor 2. ¿Se presenta una descripción resumida de todos los casos de uso?	Métrica 2. Número de casos de uso que no tiene descripción resumida. Umbral < 90%	Número de casos de uso que no tiene descripción resumida: 0 Se presenta un 100%.
	Factor 3. ¿Están definidos todos los requisitos que justifican la funcionalidad del caso de uso?	Métrica 3: Número de requisitos omitidos por caso de uso. Umbral: 90%	Número de requisitos omitidos por caso de uso: 1 Se presenta un 91.67%.
		Métrica 4: Número de casos de uso que tienen requisitos omitidos. Umbral: 80%	Número de casos de uso que tienen requisitos omitidos: 1 Se presenta un 91.67%.
	Factor 4. ¿Todos los casos de uso han sido clasificados de acuerdo a su relevancia en (crítico, secundario, auxiliar, opcional)?	Métrica 5. Número casos de que no han sido clasificados. Umbral: 80%	Número casos de que no han sido clasificados: 0 Se presenta un 100%.
		Umbral: 86%	Se presenta un: 96.66%
Consistencia	Factor 5. ¿El nombre dado a los casos de uso es una expresión verbal que describe alguna funcionalidad relevante en el contexto del usuario?	Métrica 6: Número de casos de uso que tienen un nombre incorrecto. Umbral: 80%	Número de casos de uso que tienen un nombre incorrecto: 0 Se presenta un 100%.

	Factor6.¿Está adecuadamente redactado (en el lenguaje del usuario) el flujo de eventos?	Métrica 7: Grado de adecuación de la descripción del flujo de eventos para un caso de uso Umbral: 95%	La descripción se define en el lenguaje del usuario. Se define el responsable de cada acción: 3. Se presenta un 75%.
	Factor 7. ¿La descripción del flujo de eventos se inicia con la descripción de una acción externa originada por un actor o por una condición interna del sistema claramente identificable?	Métrica 8: Número de casos de uso cuya descripción incluida no inicia con una acción externa o con una condición monitoreada por el sistema. Umbral < 90%	Número de casos de uso cuya descripción incluida no inicia con una acción externa o con una condición monitoreada por el sistema: 1 Se presenta un 91.67%.
	Factor 8. ¿Existe una adecuada separación entre el flujo básico de eventos y los flujos alternos y/o flujos subordinados?	Métrica 9: Número de casos de uso complejos que no tienen separación del flujo básico y de flujos alternos. Umbral < 80%	Número de casos de uso complejos que no tienen separación del flujo básico y de flujos alternos: 0 Se presenta un 100%.
		Umbral: 55%	Se presenta un: 42.67%
Correctitud	Factor 9. ¿Representa el caso de uso requisitos comprensibles por el usuario?	Métrica 10: Número de casos de uso en que los requisitos representados no son comprensibles por el usuario. Umbral < 95%	Número de casos de uso en que los requisitos representados no son comprensibles por el usuario: 2 Se presenta un 83.34%.
	Factor 10. ¿Las interacciones definidas describen la funcionalidad requerida del sistema?	Métrica 11: Número de casos de uso que deben ser modificados para adecuarlos a la funcionalidad del sistema. Umbral < 90%	Número de casos de uso que deben ser modificados para adecuarlos a la funcionalidad del sistema: 0 Se presenta un 100%.
	Factor 11.¿Se ajusta la representación del diagrama del caso de uso de acuerdo a lo normado en la metodología?	Métrica 12: Grado en que se ajusta el diagrama del caso de uso a la metodología Umbral < 90%	Grado en que se ajusta el diagrama del caso de uso a la metodología: 1 Se presenta un 91.67%.

	Factor 12. ¿Las interacciones definidas introducen mejoras al proceso actual?	Métrica 13: Número de casos de uso que deben ser modificados para mejorar el proceso actual. Umbral < 95%	Número de casos de uso que deben ser modificados para mejorar el proceso actual: 1 Se presenta un 91.67%.
		Umbral: 70%	Se presenta un: 66.67%
Complejidad	Factor 13. ¿Los elementos dentro del diagrama están adecuadamente ubicados de manera que facilitan su interpretación?	Métrica 14: Número de elementos del diagrama que requieren reubicación. Umbral < 70%	Número de elementos del diagrama que requieren reubicación: 1 Se presenta un 91.67%.
		Umbral: 70%	Se presenta un: 91.67%

Segunda Revisión:

Atributo	Factor	Métrica asociada	Valor
Completitud	Factor 1. ¿Han sido definidos todos los roles relevantes de usuario encargados de generar/ modificar o consultar información?	Métrica 1: Número de roles relevantes omitidos.	Número de roles relevantes omitidos: 0 Se presenta un 100%.
	Factor 2. ¿Se presenta una descripción resumida de todos los casos de uso?	Métrica 2. Número de casos de uso que no tiene descripción resumida.	Número de casos de uso que no tiene descripción resumida: 0 Se presenta un 100%.
	Factor 3. ¿Están definidos todos los requisitos que justifican la funcionalidad del caso de uso?	Métrica 3: Número de requisitos omitidos por caso de uso.	Número de requisitos omitidos por caso de uso: 0 Se presenta un 100%.
		Métrica 4: Número de casos de uso que tienen requisitos omitidos.	Número de casos de uso que tienen requisitos omitidos: 0 Se presenta un 100%.
	Factor 4. ¿Todos los casos de uso han sido clasificados de acuerdo a su relevancia en (crítico, secundario, auxiliar, opcional)?	Métrica 5. Número casos de que no han sido clasificados.	Número casos de que no han sido clasificados: 0 Se presenta un 100%.
			Se presenta un: 100%

Consistencia	Factor 5. ¿El nombre dado a los casos de uso es una expresión verbal que describe alguna funcionalidad relevante en el contexto del usuario?	Métrica 6: Número de casos de uso que tienen un nombre incorrecto.	Número de casos de uso que tienen un nombre incorrecto: 0 Se presenta un 100%.
	Factor 6. ¿Está adecuadamente redactado (en el lenguaje del usuario) el flujo de eventos?	Métrica 7: Grado de adecuación de la descripción del flujo de eventos para un caso de uso	La descripción se define en el lenguaje del usuario. Se define el responsable de cada acción: 0. Se presenta un 100%.
	Factor 7. ¿La descripción del flujo de eventos se inicia con la descripción de una acción externa originada por un actor o por una condición interna del sistema claramente identificable?	Métrica 8: Número de casos de uso cuya descripción incluida no inicia con una acción externa o con una condición monitoreada por el sistema.	Número de casos de uso cuya descripción incluida no inicia con una acción externa o con una condición monitoreada por el sistema: 01 Se presenta un 100%.
	Factor 8. ¿Existe una adecuada separación entre el flujo básico de eventos y los flujos alternos y/o flujos subordinados?	Métrica 9: Número de casos de uso complejos que no tienen separación del flujo básico y de flujos alternos.	Número de casos de uso complejos que no tienen separación del flujo básico y de flujos alternos: 0 Se presenta un 100%.
			Se presenta un: 100%
Correctitud	Factor 9. ¿Representa el caso de uso requisitos comprensibles por el usuario?	Métrica 10: Número de casos de uso en que los requisitos representados no son comprensibles por el usuario.	Número de casos de uso en que los requisitos representados no son comprensibles por el usuario: 0 Se presenta un 100%.
	Factor 10. ¿Las interacciones definidas describen la funcionalidad requerida del sistema?	Métrica 11: Número de casos de uso que deben ser modificados para adecuarlos a la funcionalidad del sistema.	Número de casos de uso que deben ser modificados para adecuarlos a la funcionalidad del sistema: 0 Se presenta un 100%.

	Factor 11.¿Se ajusta la representación del diagrama del caso de uso de acuerdo a lo normado en la metodología?	Métrica 12: Grado en que se ajusta el diagrama del caso de uso a la metodología	Grado en que se ajusta el diagrama del caso de uso a la metodología: 0 Se presenta un 100%.
	Factor 12. ¿Las interacciones definidas introducen mejoras al proceso actual?	Métrica 13: Número de casos de uso que deben ser modificados para mejorar el proceso actual.	Número de casos de uso que deben ser modificados para mejorar el proceso actual: 0 Se presenta un 100%.
			Se presenta un: 100%
Complejidad	Factor 13. ¿Los elementos dentro del diagrama están adecuadamente ubicados de manera que facilitan su interpretación?	Métrica 14: Número de elementos del diagrama que requieren reubicación.	Número de elementos del diagrama que requieren reubicación: 0 Se presenta un 100%.
			Se presenta un: 100%