Facultad 5: Informática Industrial.

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

Título: Herramienta de Simulación de Montecarlo para el “Sistema de Confiabilidad Integral de Activos para los pueblos del Alba”.

Autor: Riolvi Acosta Gonzalez.

Tutor: Ing. Jordenys Pérez Feria.

Ciudad de La Habana, Junio de 2010

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas el derecho patrimonial de la misma, con carácter exclusivo.

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

Riolvi Acosta Gonzalez

_____
Firma del Autor

Ing. Jordenys Pérez Feria

_____
Firma del Tutor

DATOS DE CONTACTO

Nombre y apellidos: Jordenys Pérez Feria.
Institución: Universidad de las Ciencias Informáticas (UCI).
Título: Ingeniero en Ciencias Informáticas.
e-mail: jferia@uci.cu.
Ingeniero en Ciencias Informáticas en el 2008, profesor en adiestramiento de la UCI, con 1 año de experiencia en su desempeño laboral.

AGRADECIMIENTOS

Agradezco:

A mi madre y abuela por el apoyo incondicional durante toda mi vida, por su amor, por ser mi fuente de inspiración diaria y por confiar en mí.

A mi novia por su amor, dedicación, paciencia y apoyo durante estos cuatro años.

A mi familia en general, por su apoyo e inspiración.

A mis segundos padres Lyla y Rafael por acogerme en su casa como un hijo.

A mi tutor Jordenys Pérez Feria por su apoyo, dedicación y confianza en estos años de trabajo.

A la Universidad y su colectivo de profesores que me han formado durante estos cinco años.

A todas las amistades que he conocido en la escuela.

A la Revolución y a Fidel por darme la oportunidad de realizar este sueño.

En fin, quiero agradecer a todas aquellas personas que de una forma u otra han contribuido al desarrollo de este trabajo de diploma y a mi formación tanto profesional como personal.

DEDICATORIA

A mi mamá Paula González Morales y mi abuela Juana Morales.

RESUMEN

La presente investigación se centra en el desarrollo de la herramienta de Simulación de Montecarlo para el proyecto SCIA, la cual permitirá la caracterización probabilística de variables aleatorias, la propagación de la incertidumbre asociada a cada variable de los modelos matemáticos manejados en cada una de sus metodologías, cuantificar el nivel de incertidumbre asociada a la salida de los mismos y realizar pruebas de bondad de ajuste. Para la implementación de la solución se hizo un análisis de las principales bibliografías asociadas a la investigación y de las diferentes herramientas existentes en el mundo relacionadas con el tema en cuestión. Como resultado se dotó a SCIA de una herramienta de código abierto, multiplataforma y bajo los principios de soberanía tecnológica.

PALABRAS CLAVE

Simulación de Montecarlo, caracterización probabilística, variables aleatorias, propagación de la incertidumbre, modelos matemáticos.

Índice de Contenido.

Índice de Figuras.

Índice de Tablas.

INTRODUCCIÓN

La dinámica de los negocios actuales exige a las industrias producir con menos costo, más calidad y con un mayor nivel de confiabilidad para cumplir con exigentes requerimientos de orden técnico, económico y legal. Adicionalmente para poder competir y permanecer vigentes en el mercado, las empresas se ven forzadas a invertir basadas en informaciones incompletas, inciertas o difusas. La empresa Petróleos de Venezuela S.A (PDVSA), producto a la utilización de un gran número de herramientas para el manejo de informaciones de confiabilidad que proporcionan resultados incompletos, que requieren del manejo manual de grandes volúmenes de información y con costos de licenciamiento muy elevados, no es exenta a estos problemas.

Producto a las relaciones Cuba-Venezuela, y a los proyectos existentes con la Universidad de las Ciencias Informáticas (UCI) y a los precedentes proyectos realizados bajo este convenio se asume el proyecto "Sistema de Confiabilidad Integral de Activos (SCIA) para los pueblos del Alba", conformando un equipo multidisciplinario de estudiantes y profesores de la UCI, los cuales trabajarán en conjunto con especialistas e Ingenieros de Confiabilidad del hermano país venezolano. El proyecto permite homologar metodologías de las diferentes etapas de la confiabilidad operacional, mejorar el proceso de gestión de mantenimiento, realizar análisis probabilístico de riesgo de exploración, perforación y yacimientos petrolíferos mejorando la eficiencia en las operaciones de los procesos realizados en PDVSA.

El proyecto maneja una gran cantidad de variables probabilísticas, de las cuales se necesita representar matemáticamente su comportamiento, generar la mayor cantidad de escenarios posibles bajo el supuesto de múltiples probables eventos con múltiples resultados independientes entre sí tomando como fuentes de información, data de campo, opinión de expertos o parámetros probabilísticos, con la finalidad de poder realizar predicciones con cierto grado de certeza.

El proyecto SCIA no posee una herramienta de código abierto, multiplataforma y bajo los principios de soberanía tecnológica, que permita realizar pruebas de bondad de ajuste para determinar la distribución de probabilidad que mejor representa a las múltiples variables que maneja, propagar la incertidumbre asociada a cada variable de los modelos matemáticos manejados en cada una de sus metodologías y cuantificar el nivel de incertidumbre asociada a la salida de los mismos.

Analizando la situación problémica expuesta con anterioridad se define como *problema investigativo*:

¿Cómo lograr una caracterización probabilística de variables aleatorias y propagación de la incertidumbre que permita cuantificar el nivel de incertidumbre asociada a las variables de salida de los modelos matemáticos manejados en el proyecto SCIA?

Teniendo en cuenta el problema investigativo se precisa como *objeto de estudio:* La Inferencia Estadística.

Por todo lo anterior, y para darle solución al problema de investigación se plantea el siguiente *objetivo general*:

Desarrollar una herramienta de Simulación de Montecarlo que permita cuantificar el nivel de incertidumbre asociada a las variables de salida de los modelos matemáticos manejados en el proyecto SCIA.

Por tanto el campo de acción es: La Simulación de Montecarlo aplicada a la Confiabilidad Integral.

Para dar cumplimiento al objetivo general se plantean las siguientes *tareas investigativas*:

I. Elaboración del marco teórico centrado en la propagación de la incertidumbre a través del método de simulación de Montecarlo y los artefactos del proyecto SCIA para la caracterización probabilística de las variables aleatorias y la cuantificación de la incertidumbre.
II. Diseño de la herramienta de Simulación de Montecarlo.
III. Diseño de la interfaz de usuario de la herramienta de Simulación de Montecarlo y su integración con la aplicación del proyecto SCIA.
IV. Implementación de la herramienta de Simulación de Montecarlo.
V. Integración de la herramienta de Simulación de Montecarlo a la aplicación del proyecto SCIA.
VI. Documentación y estandarización del código fuente de la herramienta de Simulación de Montecarlo y de la integración con la aplicación del proyecto SCIA.
VII. Diseño y desarrollo de pruebas unitarias a la herramienta de Simulación de Montecarlo e integración con la aplicación del proyecto SCIA.

De acuerdo con el problema científico planteado se define la siguiente *idea a defender*:

El desarrollo de la herramienta de Simulación de Montecarlo para el proyecto SCIA permitirá la caracterización probabilística de variables aleatorias, propagación de la incertidumbre y cuantificación del nivel de incertidumbre asociada a las variables de salida de los modelos matemáticos.

El presente documento se encuentra estructurado por tres capítulos, con toda la información referente al tema de investigación. Estos se encuentran distribuidos de la siguiente manera:

Capítulo I: “Fundamentación Teórica”, se realiza un estudio de los conceptos asociados al tema de la investigación, las herramientas existentes, la metodología y las tecnologías a utilizar.

Capítulo II: “Diseño e Implementación”, se analiza la arquitectura de la solución, el diseño y descripción de los subsistemas y clases más importantes para la comprensión del código fuente, el modelo de implementación y los requerimientos para el despliegue de la solución.

Capítulo III: “Validación de la Solución”, diseño e implementación de pruebas para validar la solución.

Durante toda la investigación se emplearon varios *métodos científicos* en la búsqueda y procesamiento de la información:

A nivel teórico:

- ✓ Análisis-síntesis e inducción-deducción: Para el estudio y recopilación de los conceptos referentes a la modelación matemática en la ingeniería de confiabilidad.
- ✓ Análisis histórico-lógico: Para conocer con mayor profundidad los antecedentes y tendencias actuales de las herramientas de confiabilidad existentes, que efectúan la caracterización probabilística de variables aleatorias y propagación de la incertidumbre asociada a cada variable de los modelos matemáticos mediante el método de Simulación de Montecarlo.

A nivel empírico:

- ✓ 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

Introducción.

El presente capítulo aborda el estado del arte referente a la caracterización probabilística de variables aleatorias y propagación de la incertidumbre asociada a cada variable de los modelos matemáticos en los procesos de Confiabilidad Integral mediante la Simulación de Montecarlo, los principales conceptos y herramientas existentes asociados al tema, así como las tecnologías y metodologías de desarrollo de software a utilizar en la implementación de la solución.

1.1. Variable aleatoria o distribuida.

Se denomina variable aleatoria o distribuida, a una variable X que por sus características pueda tomar un conjunto de valores (x1, x2, x3, x4,... xn-1,) cada uno de los cuales tiene una probabilidad de ocurrencia (p1, p2, p3, p4,... pn-1,), sin que se pueda asegurar específicamente cuál de todos estos probables valores tomará la variable. Las variables aleatorias pueden ser continuas o discretas (1).

Las variables aleatorias discretas sólo pueden tomar valores enteros, es decir, un número finito o infinito de valores numerables o contables. Se obtienen a través de un proceso de medición (*medición de peso, volumen, longitud, tiempo, etc.*). Mientras las variables aleatorias continuas pueden tomar todos los valores de un intervalo dado, obteniéndose a través de un proceso de conteo (*número de vehículos, número de pozos secos, eventos por unidad de tiempo, etc.*) (1).

1.2. Incertidumbre.

Al estado o calidad de no estar seguro, estar falto de conocimiento o con duda se le conoce como incertidumbre (2).

La incertidumbre es un atributo de todo cuanto rodea al hombre y del hombre mismo, se encuentra cuantificado por el nivel de incertidumbre como medida de la inseguridad o grado de desconocimiento acerca del valor que puede tomar una variable, proceso o fenómeno bajo estudio. Cuando se estudia un proceso específico, el nivel de conocimiento sobre el mismo puede variar desde el extremo de no saber absolutamente nada acerca de él (*ignorancia total*), hasta el extremo de llegar a entender y modelar completamente su comportamiento (*certidumbre total*). Ambos extremos, son poco probables en la realidad, ya que a pesar de no disponer de ningún modelo que caracterice la variable, fenómeno o

proceso bajo estudio, siempre se dispone de un mínimo de información que nos separa de la ignorancia total.

1.3. Probabilidad.

La probabilidad se define como una medida de la posibilidad de ocurrencia de un evento (1).

Para definir formalmente probabilidad existen dos escuelas de pensamiento que regulan su significado y aplicación, la frecuentista o clásica de probabilidad y la subjetivista o bayesiana de probabilidad. La clásica de probabilidad apoya la idea de que la frecuencia de ocurrencia de un evento es un indicador de probabilidad de dicho evento, es decir, si el evento es muy frecuente se asume que su probabilidad de ocurrencia es alta (*su probabilidad tiende a 1*); si por el contrario, el evento es poco frecuente se asume que su probabilidad de ocurrencia es baja (*su probabilidad tiende a 0*). La bayesiana define a la probabilidad como el grado de confianza en la veracidad de una proposición. Una proposición es una hipótesis que puede ser probada como verdadera o falsa.

1.3.1. Probabilidad de Falla.

La Probabilidad de Falla es la probabilidad de que un componente, sistema o proceso falle o deje de realizar lo que del mismo se requiere, en un intervalo de tiempo determinado (1).

1.3.2. Distribuciones de Probabilidad.

Las Distribuciones de Probabilidad son modelos que describen la forma en que se espera que varíen los resultados o probables valores de una variable aleatoria y se caracteriza por tres criterios fundamentales: el valor central o medida de posición (*la media, la mediana o la moda*), el grado de dispersión (*la desviación estándar*) y la forma de la curva (*forma general de la distribución probabilística*) (1).

Existen distribuciones de probabilidad para describir variables continuas tales como: distribución Normal, distribución Lognormal, distribución Exponencial, distribución Weibull, distribución Beta, distribución Gamma, distribución Triangular, distribución Uniforme, y a variables discretas como son: distribución Binomial, distribución de Poisson, distribución hipergonométrica, distribución geométrica, entre otras.

1.4. Pruebas de Bondad de Ajuste.

La Prueba de Bondad de Ajuste mide la discrepancia entre una distribución observada y otra teórica. En otras palabras se puede definir como la probabilidad de reproducir el conjunto de datos de la muestra, a

partir de una distribución teórica paramétrica seleccionada. Existen diferentes pruebas de bondad de ajuste como son: prueba de Chi-Square, Kolmogorov-Smirnov y Anderson-Darling (1).

Estas pruebas contemplan los siguientes pasos:

- ✓ Graficar cada una de las curvas de las distribuciones hipótesis teóricas obtenidas con sus parámetros estimados y el histograma de los datos de la muestra.
- ✓ Calcular para cada distribución hipótesis el valor de prueba y compararlo contra el valor crítico.

Si el valor de prueba es menor que el valor crítico entonces la distribución hipotética se considera un buen ajuste y la hipótesis no es rechazada. Si por el contrario, el valor de prueba es mayor que el valor crítico, la hipótesis se rechaza.

1.5. Confiabilidad.

La Confiabilidad es el complemento probabilístico de la probabilidad de fallas (1).

1.5.1. Confiabilidad Integral.

Se define Confiabilidad Integral como una sinergia de disciplinas y metodologías basadas en confiabilidad y riesgo, que incorpora herramientas para el manejo probabilístico de información y su incertidumbre y avanzadas técnicas de diagnóstico, modelaje y pronóstico, en la búsqueda de reducir sistemáticamente la ocurrencia de fallas, paros y eventos no deseados, para optimizar el costo del ciclo de vida de los activos. Dentro de los objetivos de la confiabilidad integral se encuentra predecir todos los escenarios de producción factibles, modelando las incertidumbres asociadas a las variables técnicas que rigen su proceso particular de producción, predecir probabilísticamente la ocurrencia de eventos no deseados, e identificar acciones concretas para minimizar su ocurrencia y explorar las implicaciones económicas de cada escenario posible y diseñar planes y estrategias óptimas para el manejo del negocio (1).

La Confiabilidad Integral se sustenta en la integración de diversas disciplinas y metodologías tales como:

Dentro de las disciplinas: *probabilidad y estadística descriptiva, gerencia de la incertidumbre, ingeniería de confiabilidad, análisis de riesgo, confiabilidad humana y gerencia de activos.*

Entre las metodologías: *análisis de criticidad , análisis de confiabilidad, disponibilidad y mantenibilidad (CDM), mantenimiento basado en confiabilidad o cuidados integral de activos (MCC), inspección basada en riesgos (IBR), optimización costo y riesgo (OCR), análisis causa raíz (ACR), análisis económico de ciclos de vida (AECV), entre otras*.

Las disciplinas son las llamadas ciencias base que proveen conceptos, leyes y herramientas para resolver problemas de diversa índole; son de naturaleza eminentemente genérica. Por su parte, las metodologías son recetas para resolver problemas específicos que se diseñan combinando herramientas provenientes de diversas disciplinas (1).

Para garantizar la adecuada utilización de las metodologías en la Confiabilidad Integral se recomienda poseer conocimientos sobre las disciplinas o ciencias base.

1.5.1.1. Gerencia de la Incertidumbre.

La Gerencia de la Incertidumbre es un proceso que se inicia con la cuantificación de los niveles de incertidumbre de cada una de las variables que intervienen en el modelo de decisión utilizado, así como el de la incertidumbre propia de los modelos a utilizar. Posteriormente se determina el impacto de la incertidumbre en el modelo de decisión considerado, básicamente se propaga la incertidumbre de cada una de las variables de entrada a través del modelo escogido o grupo de modelos seleccionados para representar el proceso o fenómeno bajo estudio. De la misma manera, en esta fase se determina el costo de la incertidumbre y se realiza un análisis de sensibilidad para identificar las variables de mayor impacto sobre el modelo. Por último se toman acciones para reducir la incertidumbre en caso de ser técnicamente factible y económicamente rentable. Es aquí donde debe compararse las consecuencias de una acción orientada a reducir la incertidumbre (*y por consiguiente incrementar nuestra probabilidad de acierto*), con las consecuencias de decidir sin reducir la incertidumbre. Por lo que se puede inferir, el objetivo en sí no es disminuir la incertidumbre simplemente por disminuirla, sino reducirla cuando ello se traduce en un beneficio neto dentro del proceso de toma de decisiones (2).

1.5.1.2. Cuantificación de la Incertidumbre.

La cuantificación de la incertidumbre comprende la cuantificación y caracterización probabilística de cada una de las variables de entrada, la propagación de la incertidumbre de las variables de entrada a través del modelo y la cuantificación y caracterización probabilística de la variable de salida (2).

1.5.1.3. Caracterización Probabilística de Variables Aleatorias.

La caracterización probabilística de variables aleatorias es el proceso para conocer el comportamiento de las variables, así como un conjunto de distribuciones para representar la heterogeneidad e incertidumbre de las mismas (1).

La caracterización se clasifica en caracterización por la naturaleza física, confinamiento o representación matemática de la variable a modelar.

En el desarrollo de la investigación se abordará la caracterización a través de la naturaleza física de la variable a modelar, donde las variables pueden ser continuas o discretas.

En función de la naturaleza de la variable de interés y de la disponibilidad de data, existen dos fuentes básicas de información para describir las variables:

- ✓ Datos provenientes de observaciones directas y/o medición de dispositivos de campo.
- ✓ Datos provenientes del conocimiento empírico del proceso.

En el primer caso, los datos son perfectamente caracterizados desde el punto de vista probabilístico. La cuantificación de la frecuencia de aparición de los datos en la muestra permite obtener una distribución de probabilidad y hacer una evaluación probabilística objetiva debido a que se parte de la abundancia de la evidencia de la muestra.

En el segundo caso, la obtención de los datos se hace en forma de datos. En estas situaciones se recurre a la utilización de la opinión de expertos, lo que permite combinar el conocimiento de las personas sobre el área o proceso en análisis con la poca evidencia o datos disponibles. Esta combinación agrega el componente subjetivo a los análisis. Afortunadamente existen un sin número de instrumentos que permiten minimizar la subjetividad de la información por parte del experto.

1.5.1.3.1. Caracterización Probabilística de Variables con Información de Campo.

La caracterización probabilística de variables con información de campo se rige por una serie de pasos que se describen a continuación:

1. Plantear las hipótesis de las distribuciones paramétricas que podrían hacer un buen ajuste con los datos.
2. Calcular los parámetros de cada una de las distribuciones hipótesis con los datos de la muestra.
3. Realizar alguna de las pruebas de bondad de ajuste.
4. Seleccionar entre las distribuciones hipotéticas no rechazadas, aquella que tenga el valor de prueba más bajo.

1.5.1.4. Propagación de la Incertidumbre.

La propagación de incertidumbre es el procedimiento mediante el cual se puede incluir y contabilizar la incertidumbre asociada a las variables de entrada, en un determinado modelo matemático (*ecuación,*

inecuación o correlación), para cuantificar la incertidumbre de la variable de salida; en otras palabras, es la metodología o proceso para resolver ecuaciones, cuando las variables de entrada son distribuciones de probabilidad. Si las variables de entrada al modelo tienen incertidumbre, entonces el resultado o salida del modelo debe tener incertidumbre (2).

Para la propagación de la incertidumbre existen dos métodos de solución: las soluciones analíticas por medio del método de los Momentos con series de Taylor y las soluciones numéricas a través del método de simulación de Montecarlo.

1.5.1.5. Método de Simulación de Montecarlo.

La simulación es el proceso de diseñar y desarrollar un modelo computarizado de un sistema o proceso y conducir experimentos con este modelo con el propósito de entender el comportamiento del sistema o evaluar varias estrategias con las cuales se puede operar el sistema (3). Está compuesta por un conjunto de hipótesis expresado como relaciones matemáticas y/o lógicas entre los elementos del sistema y la ejecución a través del tiempo en un ordenador para generar muestras representativas del comportamiento.

Actualmente existen diversos métodos de simulación entre los más importantes se encuentran: la simulación continua (*los estados del sistema cambian continuamente su valor y son modelados con ecuaciones diferenciales*), por eventos (*el comportamiento varía en instantes de tiempo dados*), por autómatas celulares (*divide el comportamiento del sistema en subsistemas más pequeños denominadas células*) y estadística o de Montecarlo (*muestreo sistemático de variables aleatorias*).

El método de Montecarlo es uno de los métodos matemáticos computacionales más usados y antiguos existentes, proveniente de la referencia al Casino de Montecarlo (Principado de Mónaco) por ser “la capital del juego de azar”, el cual data aproximadamente de 1944 con el desarrollo de la computadora, aunque existen instancias aisladas y no desarrolladas anteriores a este año. En sus primeras investigaciones John von Neumann y Stanislao Ulam refinaron la ruleta rusa y los métodos de división, sin embargo, Harris y Herman Kahn en 1948 fueron los que llevaron el desarrollo sistemático. El uso real del método de Montecarlo como una herramienta de investigación, proviene del trabajo de la bomba atómica durante la Segunda Guerra Mundial, este trabajo involucró la simulación directa de problemas probabilísticos de hidrodinámica concernientes a la difusión de neutrones aleatorios en material de fusión. Alrededor de 1970, los desarrollos teóricos en complejidad computacional comienzan a proveer mayor precisión y relación para el empleo del método Montecarlo (4) .

El método Montecarlo o Simulación de Montecarlo agrupa una serie de procedimientos que analizan distribuciones de variables aleatorias usando simulación de números aleatorios, dando solución a una gran variedad de problemas matemáticos estocásticos o determinísticos, a través de experimentos con muestreos estadísticos en una computadora. El método de Montecarlo se usa para analizar problemas que no tienen un componente aleatorio explícito, para resolver integrales que no se pueden resolver por métodos analíticos o para cualquier esquema que emplee números aleatorios, usando variables aleatorias con distribuciones de probabilidad conocidas, el cual es usado para resolver ciertos problemas estocásticos y determinísticos, donde el tiempo no desempeña un papel importante (1) (2) (4) (5).
La Simulación de Montecarlo constituye el método que brinda mayor cantidad de información de las variables de salida de los modelos matemáticos utilizado en la Confiabilidad Integral.

1.6. Herramientas Existentes.

A continuación se analizan brevemente las principales herramientas existentes que realizan la Simulación de Montecarlo.
SimulAr: software desarrollado en Argentina y diseñado como complemento de Microsoft Excel 2003 o 2007 (6).
W3MCSIM: software diseñado como complemento de Microsoft Internet Explorer 6 (7).
@Risk: software que realiza análisis de riesgo a través de la Simulación de Montecarlo integrado como un complemento (*add-in*) de Microsoft Excel. Posee una variedad de opciones de licenciamiento, incluyendo licenciamiento corporativo, por redes y servidores, y mediante licencias académicas sumamente costosas (8).
Invest Sign: herramienta de análisis de riesgo a través de Simulación de Montecarlo con un elevado costo de licenciamiento (9).
Cristal Ball: software de Simulación de Montecarlo, propiedad de Oracle, con altos costos de licenciamiento e integrado como complemento de Microsoft Excel (10).
YASAI: software de Simulación de Montecarlo suplementario para el Microsoft Excel creado por la Universidad de Rutgers (11).
GoldSim: herramienta de Simulación de Montecarlo, de alto costo de licenciamiento y requiere de la plataforma de Microsoft Window (12).

Herramientas	Código Abierto	Plataformas de Ejecución	Complementos		Costos de licenciamiento
			Microsoft Excel.	Internet Explorer	
SimulAr	No	Window	Sí	No	Sí
W3MCSIM	No	Window	No	Sí	Sí
@Risk	No	Window	Sí	No	Sí
Invest Sign	No	Window	No	No	Sí
YASAI	No	Window	Sí	No	Sí
GoldSim	No	Window	No	No	Sí
Cristal Ball	No	Window	Sí	No	Sí

Tabla 1: Herramientas de Simulación de Montecarlo.

1.7. Tecnología a Utilizar.

El lenguaje de programación C++ fue diseñado en los años 1980 por Bjarne Stroustrup, con la intención de extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. C++ provee de facilidades de programación genérica, estructurada y orientada a objetos, convirtiéndolo en un lenguaje *multiparadigma*. Dota a los desarrolladores de la posibilidad de redefinir los operadores (*sobrecarga de operadores*), crear tipos de datos genéricos y desarrollar tanto a alto como a bajo nivel (13).

El lenguaje de programación interpretado Python fue creado por Guido van Rossum en el año 1991. Su nombre proviene de la afición de su creador por los humoristas británicos Monty Python. El principal objetivo que persigue este lenguaje es la facilidad de lectura y de diseño. Es un lenguaje de programación multiparadigma, o sea provee a los programadores adoptar varios estilos: programación orientada a objetos, estructurada, funcional y otros soportados mediante extensiones. Además usa tipo de dato dinámico y reference counting para la utilización de la memoria. Una característica importante del lenguaje es la resolución dinámica de nombres (*ligadura dinámica de métodos*), o sea, enlaza un método con un nombre de variable durante la ejecución del programa. El diseño del lenguaje facilita la extensión, proveyendo la inclusión de nuevos módulos desarrollados en otros lenguajes fácilmente (14).

Boost es un conjunto de bibliotecas de código abierto que extiende las capacidades del lenguaje de programación C++. Su licencia permite el uso en cualquier tipo de proyecto, ya sea comercial o no.

Además se basa en el uso intensivo de plantillas con el objetivo de alcanzar el mayor rendimiento y flexibilidad posible, abarca desde bibliotecas de propósito general hasta abstracciones del sistema operativo. Dentro de las bibliotecas comprendidas se encuentra boost python, la cual permite la interoperabilidad sin fisuras entre C++ y el lenguaje de programación Python y boost math, que provee de un módulo estadístico con distribuciones de probabilidad y un conjunto de funciones y algoritmos matemáticos (15).

Eclipse es un Entorno de Desarrollo Integrado (IDE del inglés, Integrated Development Environment) de código abierto y multiplataforma, desarrollado originalmente por IBM y propiedad de la Fundación Eclipse actualmente. Emplea módulos (*plug-in*) para proporcionar las funcionalidades al usuario. Este mecanismo extiende las funcionalidades del IDE como son: soporte de lenguajes de programación (C/C++, Python, etc) o frameworks (Qt, entre otros) (16).

Qt (*del inglés Quasar Technologies*) es un framework multiplataforma para desarrollar interfaces gráficas de usuario, desarrollado por la compañía noruega Trolltech propiedad de Nokia desde junio de 2008 y bajo licencias GPL v1, GPL v2, GPL v3 y LGPL. Está desarrollado en el lenguaje de programación C++ con bindings para los lenguajes Python (*PyQt*), Java (*Qt Jambi*), Perl (*PerlQt*), Gambas (*gb.qt*), Ruby (*QtRuby*), PHP (*PHP-Qt*) y Mono (*Qyoto*), entre otros. El framework presenta una arquitectura modular estructurada por varios módulos de los cuales se encuentran: Core (núcleo no gráfico de la biblioteca Qt), Gui (colección básica de componentes gráficos), Database, Network, Scripting, OpenGL, XML, Multimedia, Font Engine, WebKit, entre otros. El framework provee de potentes herramientas de diseño de interfaces gráficas (QtDesigner), un entorno de desarrollo integrado (QtCreator) y un interactivo asistente de ayuda (QtAssistant) (17).

GSL (*del inglés GNU Scientific Library*) es la biblioteca científica de GNU para los lenguajes de programación C y C + +, creada en mayo de 1996, de código abierto, bajo una licencia GPL y compuesta por una amplia gama de rutinas matemáticas de más de 1000 funciones, con una extensa serie de pruebas, tales como: números complejos, raíces de polinomios, funciones especiales, vectores y matrices, permutaciones, algoritmos de ordenación, álgebra lineal, generación de números aleatorios para distribuciones, interpolaciones, cálculo de raíces, entre otros.

Para la realización de la pruebas de unidad a la herramienta de Simulación de Montecarlo se empleará el framework estándar PyUnit de python (18).

PyUnit es la herramienta o framework oficial para hacer pruebas unitarias en Python. Se incluye en la librería estándar (*módulo unittest*) y ha sido creado por los desarrolladores de JUnit para facilitar la creación y gestión de pruebas en módulos Python. Este framework es soportado perfectamente por los IDEs mediante extensiones para Python y permite separar el código de pruebas, del código del propio proyecto para mejorar la comprensión, refactorización y la facilidad para hacer cambios en el programa. PyUnit permite realizar pruebas de igualdad, excepciones, fallos, pruebas de cordura (*comprobar que no se producen situaciones imposibles*) y pruebas de mayúsculas (*relacionadas con las cadenas de caracteres*) (14).

Visual Paradigm para UML (VP-UML) es una herramienta case multiplataforma que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue, entre otros. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. El VP-UML soporta los últimos estándares de la notación UML 2.0 y múltiples lenguajes de programación (19).

1.8. Metodología a Utilizar.

Proceso Unificado de Rational (*RUP del inglés, Rational Unified Process*) es un proceso de desarrollo de software resultante de la experiencia de más de 30 años de trabajo de James Rumbaugh, Grady Booch e Ivar Jacobson que junto a UML, constituye la metodología estándar más utilizada para el análisis, diseño, implementación y documentación de sistemas orientados a objetos en la actualidad. RUP se divide en cuatro fases (*Inicio, Elaboración, Construcción, Transición*), presenta nueve flujos de trabajos, de ellos seis son de ingeniería (*Modelamiento del Negocio, Requerimientos, Análisis y Diseño, Implementación, Prueba, Instalación o Distribución*) y tres de apoyo (*Configuración y Administración de Cambio, Administración de Proyecto, Ambiente*) y posee como características fundamentales que es centrado en la arquitectura, guiado por casos de uso, iterativo e incremental. Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software (20).

Lenguaje Unificado de Modelado (*UML del inglés, Unified Modeling Language*) es un lenguaje de modelado empleado para definir, detallar y documentar los artefactos de un sistema de software. Entre los diagramas que propone UML para modelar un sistema encontramos (21):

- Diagramas de estructura (clases, componentes, objetos, despliegue. paquetes).
- Diagramas de comportamiento (actividades, casos de uso, estado).

- Diagramas de interacción (secuencia, comunicación).

Conclusiones.

Con el análisis del estado del arte asociado a los principales conceptos de caracterización probabilística de variables aleatorias, propagación de la incertidumbre asociada a cada variable de los modelos matemáticos, la Simulación de Montecarlo y el estudio de las herramientas existentes, las cuales en su mayoría son dependientes de una plataforma específica, de código cerrado, son complementos a hojas de cálculo de Microsoft Excel o Microsoft Internet Explorer, poseen altos costos de licenciamiento e incumplen con las políticas de soberanía tecnológicas exigidas por los convenios PDVSA-UCI, se propone el desarrollo de una herramienta de Simulación de Montecarlo de código abierto, multiplataforma e integrada al proyecto SCIA bajo los principios de soberanía tecnológica como solución al problema investigativo planteado. Para la realización de la misma se utilizará el lenguajes de programación C++ complementado por la biblioteca Boost para el desarrollo del módulo de distribuciones probabilísticas, con la finalidad de lograr la mayor eficiencia y rendimiento en los cálculos matemáticos, el lenguaje python integrado al framework Qt para el diseño de la interfaz de usuario y la visualización de los datos, con el objetivo de agilizar el desarrollo e integración de la misma.

CAPÍTULO 2: DISEÑO E IMPLEMENTACIÓN.

Introducción.

El presente capítulo describe la arquitectura, el modelo de diseño e implementación de la solución. Se realiza la representación, descripción, estandarización y documentación de las clases y funcionalidades diseñadas, así como, la representación de los componentes para su implementación y los requerimientos para el despliegue.

2.1. Insumos.

La metodología utilizada en la construcción de la solución propone que todo el ciclo de vida sea centrado en la arquitectura, guiado por casos de usos, iterativo e incremental. Por lo que es indispensable recibir como insumos los artefactos siguientes:

2.1.1. Artefactos.

- ✓ Documento de la Arquitectura.
- ✓ Casos de Usos.
- ✓ Descripción de los casos de usos.
- ✓ Documento de Especificaciones Técnicas de Simulación de Montecarlo.
- ✓ Documento de Especificación de Requisitos.

2.1.2. Casos de Usos.

- ✓ Caracterización Probabilística de Variables Aleatorias.
- ✓ Cálculo Numérico de la Disponibilidad y Número Esperado de Fallas para Equipos Reparables.
- ✓ Propagación de la Incertidumbre asociada a cada Variable del Modelo Matemático.

Cada uno de los casos de usos antes mencionados, representan un algoritmo del mismo nombre. Estos describen una secuencia de pasos para su correcta ejecución.

2.1.2.1. Caracterización Probabilística de Variables Aleatorias.

Pasos a seguir para la Caracterización Probabilística de Variables con Información de Campo:

1. Plantear las hipótesis de las distribuciones paramétricas que podrían hacer un buen ajuste.
2. Calcular los parámetros de cada una de las distribuciones hipótesis con los datos de la muestra.
3. Realizar alguna de las pruebas de bondad de ajuste.

Estas pruebas contemplan las siguientes etapas:

I. Elaborar el histograma de frecuencias relativas, en el cual se apreciará la función teórica de densidad que se ajusta mejor a los datos del histograma.
II. Graficar cada una de las curvas de las Distribuciones Hipótesis teóricas obtenidas con los parámetros estimados en el paso anterior, con el histograma de los datos de la muestra.
III. Calcular para cada Distribución Hipótesis el valor llamado “valor de prueba” y compararlo contra el valor llamado “valor crítico”.
IV. Comparar los Criterios de decisión. Si el “valor de prueba” es menor que el “valor crítico” entonces la Distribución Hipotética se considera un buen ajuste y la hipótesis no es rechazada. Si por el contrario, el “valor de prueba” es mayor que el “valor crítico”, la hipótesis se rechaza.

2.1.2.2. Cálculo Numérico de la Disponibilidad y el Número Esperado de Fallas para equipos reparables.

Pasos a seguir:

1. Caracterización Probabilística de los “*Tiempos de Operación entre Fallas*” y los “*Tiempos Fuera de Servicio*”, utilizando el historial registrado de los equipos reparables, se debe aplicar el algoritmo *Caracterización Probabilística de Variables Aleatorias*.
2. Generar aleatoriamente un valor de “*Tiempo de Operación entre Fallas*” y un valor de “*Tiempo Fuera de Servicio*”, a cada variable desde sus respectivas distribuciones de probabilidad, para esto es necesario generar números aleatorios de probabilidad (entre 0 y 1) que se introducirán en la ecuación de la distribución acumulada inversa, que caracterizan al “*Tiempo de Operación entre Fallas*” y al “*Tiempo Fuera de Servicio*”.
3. Sumar el “*Tiempo de Operación entre Fallas*” y registrar (Xj = Xj-1 + Xk).
4. Sumar el “*Tiempo Fuera de Servicio*” y registrar (Yj = Yj-1 + Yk).
5. Sumar los “*Tiempos de Operación entre Fallas*” (Xj) y los “*Tiempos Fuera de Servicio*” (Yj) y registrar Tj = Tj-1 + Xk + Yk (Tiempo Total).
6. Retornar al paso 2 y repetir nuevamente los pasos del dos al cinco hasta que el Tiempo Total (Tj) sea igual al Tiempo Misión (tm).
7. Determinar la Disponibilidad (Dj = Xj/Tj) y registrar el Número de Fallas.
8. Retornar al paso 2 y repetir nuevamente los pasos del 2 al 7 hasta completar un número “*m*” de

iteraciones.

9. Una vez completadas las "*m*" iteraciones, construir un histograma de frecuencias con los "*m*" probables valores de "*Disponibilidad*" y "*Número Esperado de Fallas*" almacenados.

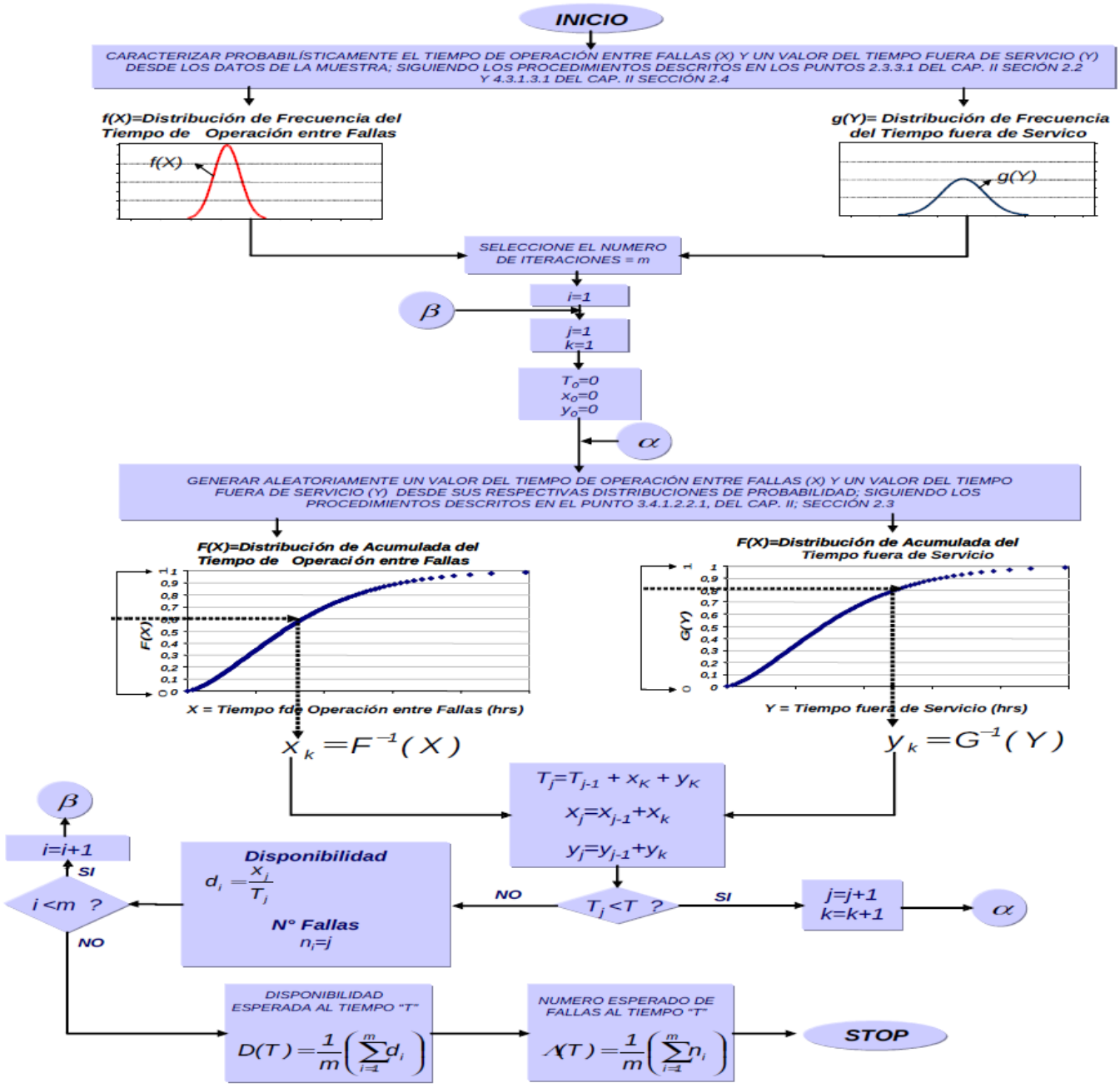

Ilustración 1: Algoritmo Cálculo Numérico de la Disponibilidad y el Número Esperado de Fallas en Equipos Reparables.

2.1.2.3. Propagación de la Incertidumbre asociada a cada Variable en el Modelo Matemático.

Pasos a seguir:

1. Generar aleatoriamente un valor de cada variable desde sus respectivas distribuciones de probabilidad, para esto es necesario generar números aleatorios de probabilidad (entre 0 y 1) que se introducirán en la ecuación de la distribución acumulada inversa, que caracteriza a la variable en estudio.
2. Sustituir en la función g (X1, X2, X3,.., Xn), el conjunto de valores generados aleatoriamente y obtener un probable valor de Y.
3. Registrar y almacenar el valor resultante de Y.
4. Retornar al paso 1 y repetir nuevamente los primeros cuatro pasos hasta completar un número "*m*" de iteraciones.
5. Una vez completadas las "*m*" iteraciones, construir un histograma de frecuencias con los "*m*" probables valores de "*Y*" almacenados.
6. Aplicar el algoritmo *de Caracterización Probabilística de Variables Aleatorias* a la variable de salida.

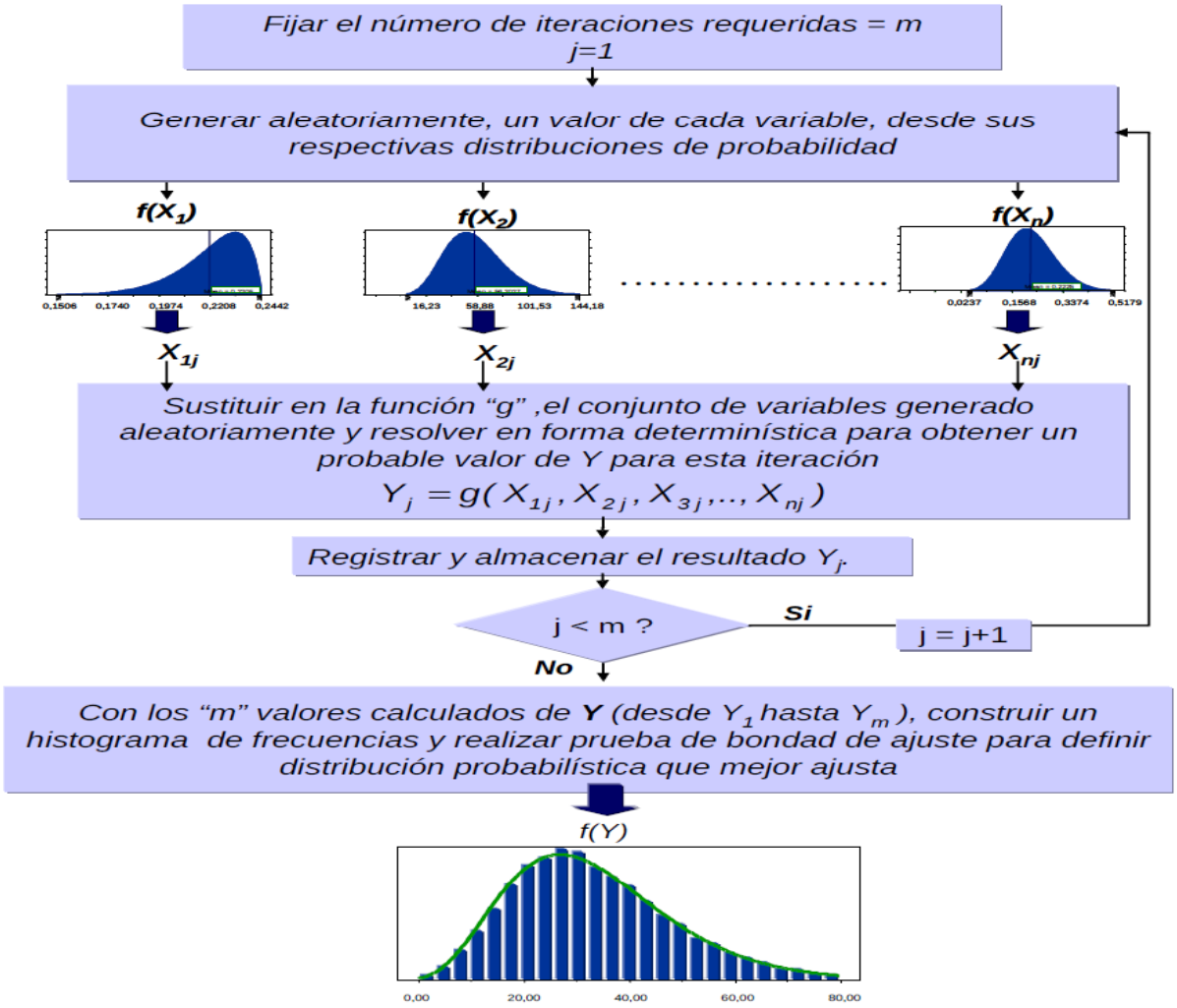

Ilustración 2: Algoritmo Propagación de la Incertidumbre asociada a cada Variable en el Modelo Matemático.

2.2. Consideraciones Generales de la Solución.

El diseño del sistema se basa en los estilos arquitectónicos siguientes:

I. Arquitectura Orientada en Componentes.
II. Arquitectura N-Capas.
III. Arquitectura Presentación Desacoplada.

2.2.1. Arquitectura Orientada en Componentes.

El estilo de arquitectura orientada en componentes describe un acercamiento al diseño de sistemas como un conjunto de componentes que exponen interfaces bien definidas y que colaboran entre sí para resolver el problema. (22)

Características.

- ✓ Es un estilo de diseño para aplicaciones compuestas de componentes individuales, los cuales pudieran ser las unidades de modelado, diseño e implementación.
- ✓ Pone énfasis en la descomposición del sistema en componentes lógicos o funcionales que tienen interfaces bien definidas, las cuales están separadas de las implementaciones, y las interfaces y sus interacciones son el centro de incumbencias en el diseño arquitectónico.
- ✓ Define una aproximación de diseño que usa componentes discretos, los que se comunican a través de las interfaces que contienen métodos, eventos y propiedades.

Beneficios.

- ✓ *Facilidad de Instalación:* Cuando una nueva versión esté disponible, usted podrá reemplazar la versión existente sin impacto en otros componentes o el sistema como un todo.
- ✓ *Costos reducidos:* El uso de componentes de terceros permite distribuir el costo del desarrollo y del mantenimiento.
- ✓ *Facilidad de desarrollo:* Los componentes implementan una interfaz bien definida para proveer la funcionalidad definida permitiendo el desarrollo sin impactar otras partes del sistema.
- ✓ *Reusable:* El uso de componentes reutilizables significa que ellos pueden ser usados para distribuir el desarrollo y el mantenimiento entre múltiples aplicaciones y sistemas.
- ✓ *Mitigación de complejidad técnica:* Los componentes mitigan la complejidad por medio del uso de contenedores de componentes y sus servicios. Ejemplos de servicios de componentes incluyen activación de componentes, gestión de la vida de los componentes, gestión de colas de mensajes para métodos del componente y transacciones.

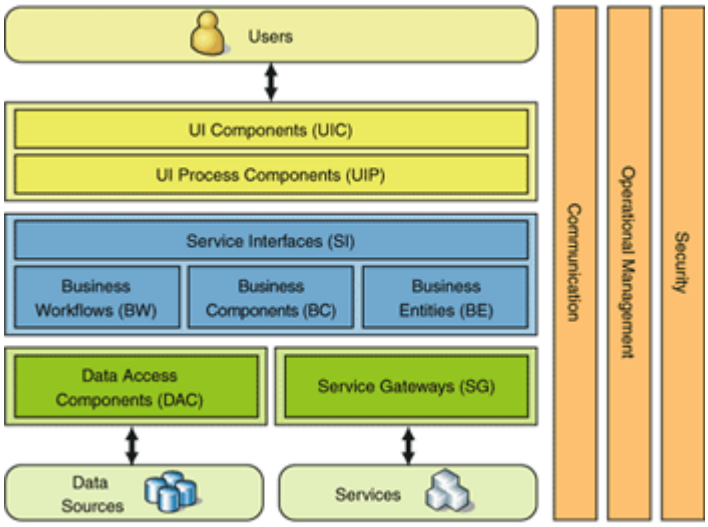

Ilustración 3: Arquitectura basada en componentes.

2.2.2. Arquitectura N-Capas.

El estilo arquitectónico en capas se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades la funcionalidad que implementan. (22)

Características.

- ✓ Descomposición de los servicios de forma que la mayoría de las interacciones ocurren solo entre capas vecinas.
- ✓ Las capas de una aplicación pueden residir en la misma máquina o pueden estar distribuidas entre varios equipos.
- ✓ Los componentes de cada capa se comunican con los componentes de otras capas a través de interfaces bien conocidas.
- ✓ Cada nivel agrega las responsabilidades y abstracciones del nivel inferior.
- ✓ Muestra una vista completa del modelo y a la vez proporciona suficientes detalles para entender las relaciones entre capas.
- ✓ Cada capa contiene la funcionalidad relacionada solo con las tareas de esa capa.
- ✓ Las capas inferiores no tienen dependencias de las capas superiores.

- ✓ La comunicación entre capas está basada en una abstracción que proporciona un bajo acoplamiento entre capas.

Beneficios.

- ✓ *Abstracción:* los cambios se realizan a alto nivel y se puede incrementar o reducir el nivel de abstracción que se usa en cada capa del modelo.
- ✓ *Aislamiento:* se pueden realizar actualizaciones en el interior de las capas sin que esto afecte al resto del sistema.
- ✓ *Rendimiento:* distribuyendo las capas en distintos niveles físicos se puede mejorar la escalabilidad, la tolerancia a fallos y el rendimiento.
- ✓ *Simplificación de las pruebas:* cada capa tiene una interfaz bien definida sobre la que realizar las pruebas y la habilidad de cambiar entre diferentes implementaciones de una capa.
- ✓ *Independencia:* elimina la necesidad de considerar el hardware y el despliegue así como las *dependencias con interfaces externas.*

Ilustración 4: Arquitectura N-Capas.

2.2.3. Arquitectura con la Presentación Desacoplada.

El estilo de presentación separada indica cómo debe realizarse el manejo de las acciones del usuario, la manipulación de la interfaz y los datos de la aplicación. Este estilo separa los componentes de la interfaz del flujo de datos y de la manipulación. (22)

Características.

- ✓ Es un estilo, para diseñar aplicaciones, basado en patrones de diseño conocidos.
- ✓ Separa la lógica para el manejo de la interacción de la representación de los datos con que trabaja el usuario.
- ✓ Permite a los diseñadores crear una interfaz gráfica mientras los desarrolladores escriben el código para su funcionamiento.
- ✓ Ofrece un mejor soporte para el testeo ya que se pueden testear los comportamientos individuales.

Beneficios.

- ✓ *Simplificación de las pruebas:* en las implementaciones comunes los roles son simplemente clases que pueden ser testeadas y reemplazadas por mocks que simulen su comportamiento.
- ✓ *Reusabilidad:* los controladores pueden ser aprovechados en otras vistas compatibles y las vistas pueden ser aprovechadas en otros controladores compatibles.

Ilustración 5: Arquitectura presentación desacoplada.

La herramienta se encuentra estructurada por dos módulos, la biblioteca de simulación para ingenieros en confiabilidad llamado Simulater y el módulo de integración Montecarlo con la aplicación del proyecto SCIA. Ambos módulos utilizan las principales ventajas de cada una de los estilos arquitectónicos antes mencionados.

La biblioteca Simulater se basa en una arquitectura orientada a componentes. Esta arquitectura aumenta la reutilización de las funcionalidades de sus componentes en distintos escenarios o aplicaciones independiente del entorno en el que se ejecute, facilita la extensión de los componentes a partir de software de terceros independiente del lenguaje de codificación, abstrae al desarrollador de la lógica interna de los componentes con el uso de interfaces para publicar las funcionalidades, facilita el despliegue y la actualización de los componentes sin afectar a otros o al sistema y reduce la complejidad del sistema.

El módulo Montecarlo se basa en una arquitectura de N-Capas y de presentación desacoplada. Este módulo está estructurado por dos capas, una capa de lógica de negocio y otra de presentación. La capa de presentación se encuentra desacoplada de la lógica de negocio aumentado la reusabilidad de ambas capas en otros módulos compatibles, facilitando el despliegue y la actualización de los capas sin afectar a otras, facilita la extensión de las capas a partir de software de terceros, reduce el nivel de abstracción y la complejidad de las capas con el aislamiento de las mismas.

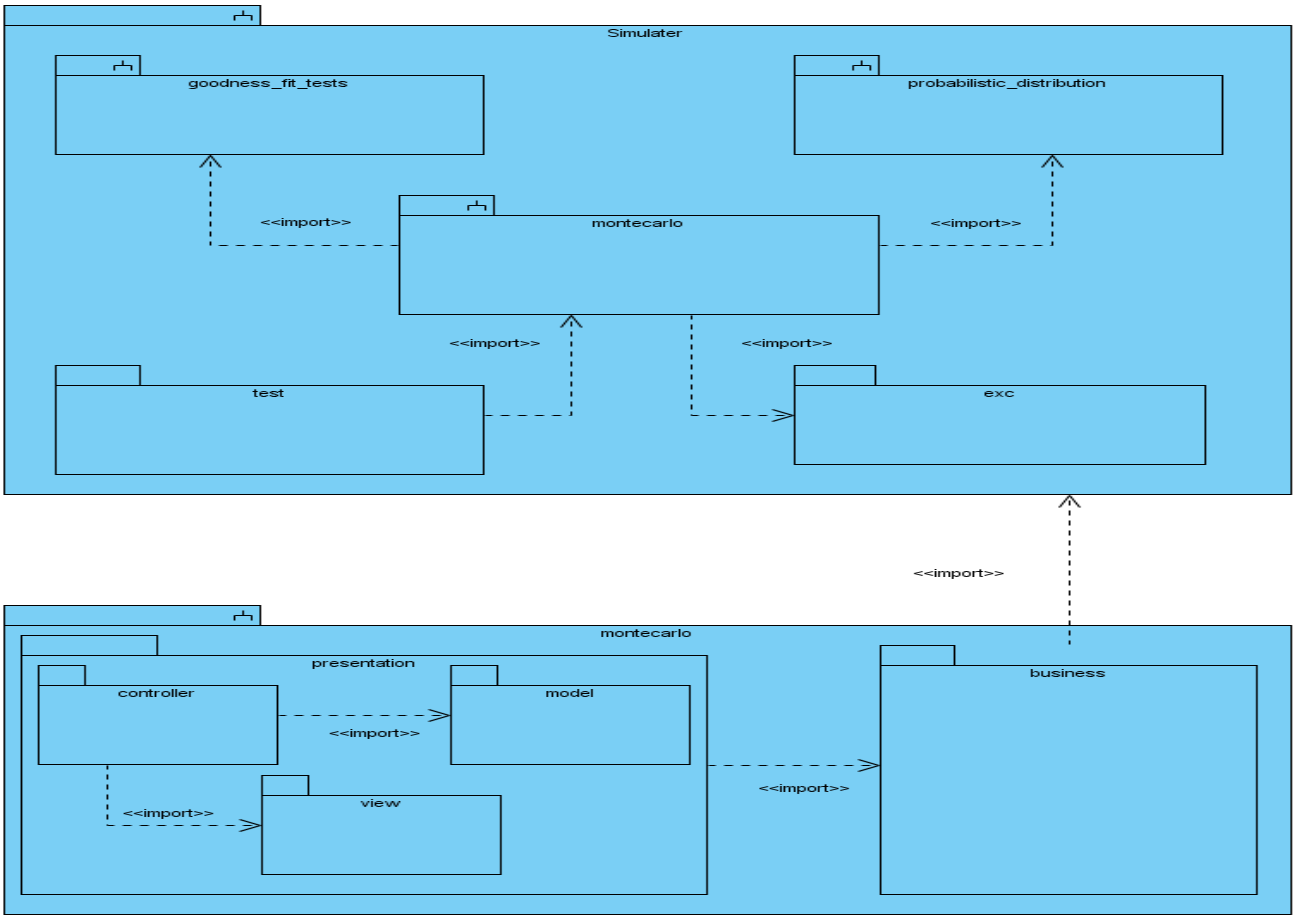

Ilustración 6: Arquitectura de la herramienta de Simulación de Montecarlo.

2.3. Descripción del Diseño.

En este epígrafe se describen los principales paquetes y subpaquetes utilizados en la implementación del sistema partiendo de un empaquetamiento de las clases del diseño para lograr un mejor entendimiento del código fuente de la solución, así como, las clases, atributos y funcionalidades más importantes.

Con el análisis de los requerimientos, casos de usos y el documento de la arquitectura del proyecto SCIA, se conforma el empaquetamiento de la solución. La misma se conforma por tres subsistemas principales y sus respectivos paquetes y subpaquetes que responden a las necesidades planteadas por el cliente, los mismos son:

- ✓ Subsistema pyprobdist: binding de la biblioteca de distribuciones de probabilidad del módulo matemático de la biblioteca estándar boost al lenguaje python.

- ✓ Subsistema simulater: biblioteca de Simulación de Montecarlo para ingenieros en Confiabilidad, el cual presenta funcionalidades como: algoritmo de caracterización probabilística de variables aleatorias, algoritmo de cálculo numérico de la disponibilidad y el número esperado de fallas para equipos reparables y el algoritmo de propagación de la incertidumbre asociada a las variables de entrada de los modelos matemáticos, entre otras. La biblioteca se encuentra estructurada por los siguientes subpaquetes:
    - o Paquete montecarlo: interfaz de la biblioteca Simulater.
    - o Paquete probabilistic_distribution: extensión de la biblioteca de distribuciones de probabilidad pyprobdist con funcionalidades específicas en el área de la confiabilidad integral.
    - o Paquete goodness_fit_tests: conjunto de pruebas de bondad de ajuste.
    - o Paquete utils: conjunto de funciones que complementan los paquetes de simulater.
    - o Paquete test: suite de pruebas de unidad a simulater.
- ✓ Subsistema Montecarlo: paquete de integración de la biblioteca de Simulación de Montecarlo a la aplicación de SCIA integrado por los subpaquetes de lógica de negocio e interfaz de usuario siguientes:
    - o Paquete presentation: paquete que maneja las acciones de los usuarios sobre la interfaz y visualiza los datos.
    - o Paquete business: paquete que maneja toda la lógica de negocio en el proceso de Simulación de Montecarlo.

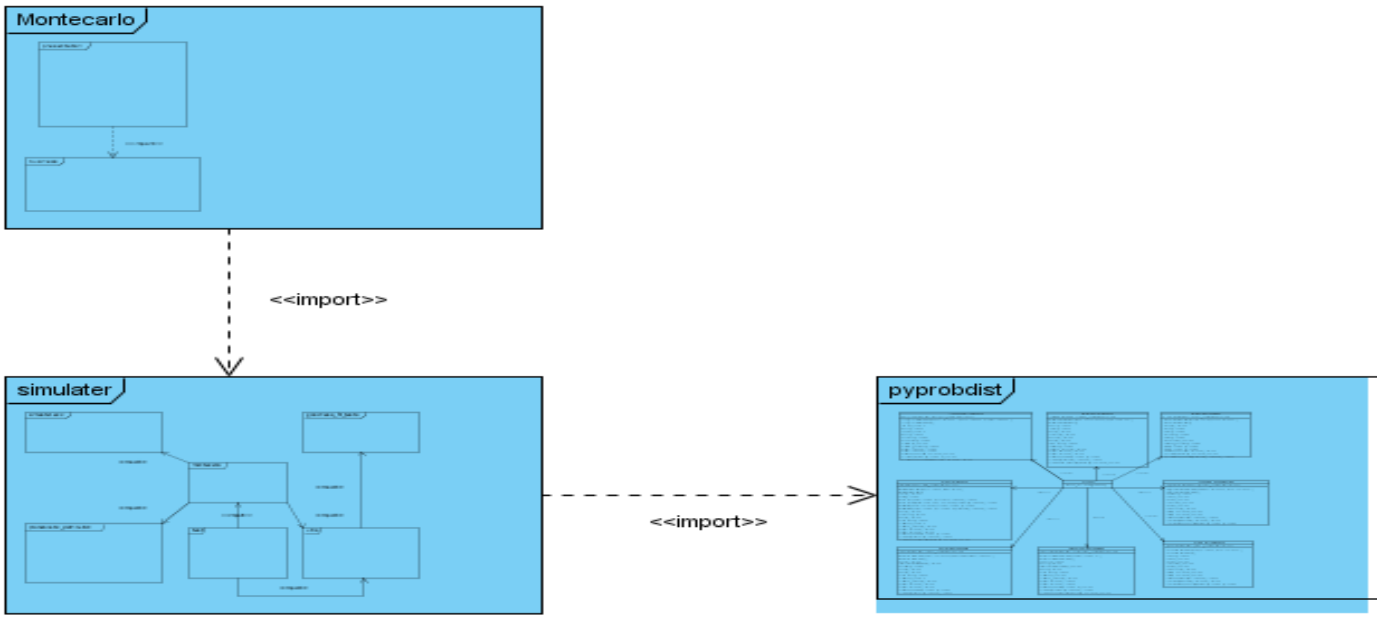

Ilustración 7: Diagrama de diseño de la herramienta de Simulación de Montecarlo.

2.3.1. Subsistema pyprobdist.

Este subsistema encapsula en clases un conjunto de funcionalidades brindadas por el módulo de distribuciones estadísticas de la biblioteca boost y a su vez las exporta del lenguaje de programación nativo c++ al lenguaje python en el cual se desarrollan todos los módulos de la herramienta.

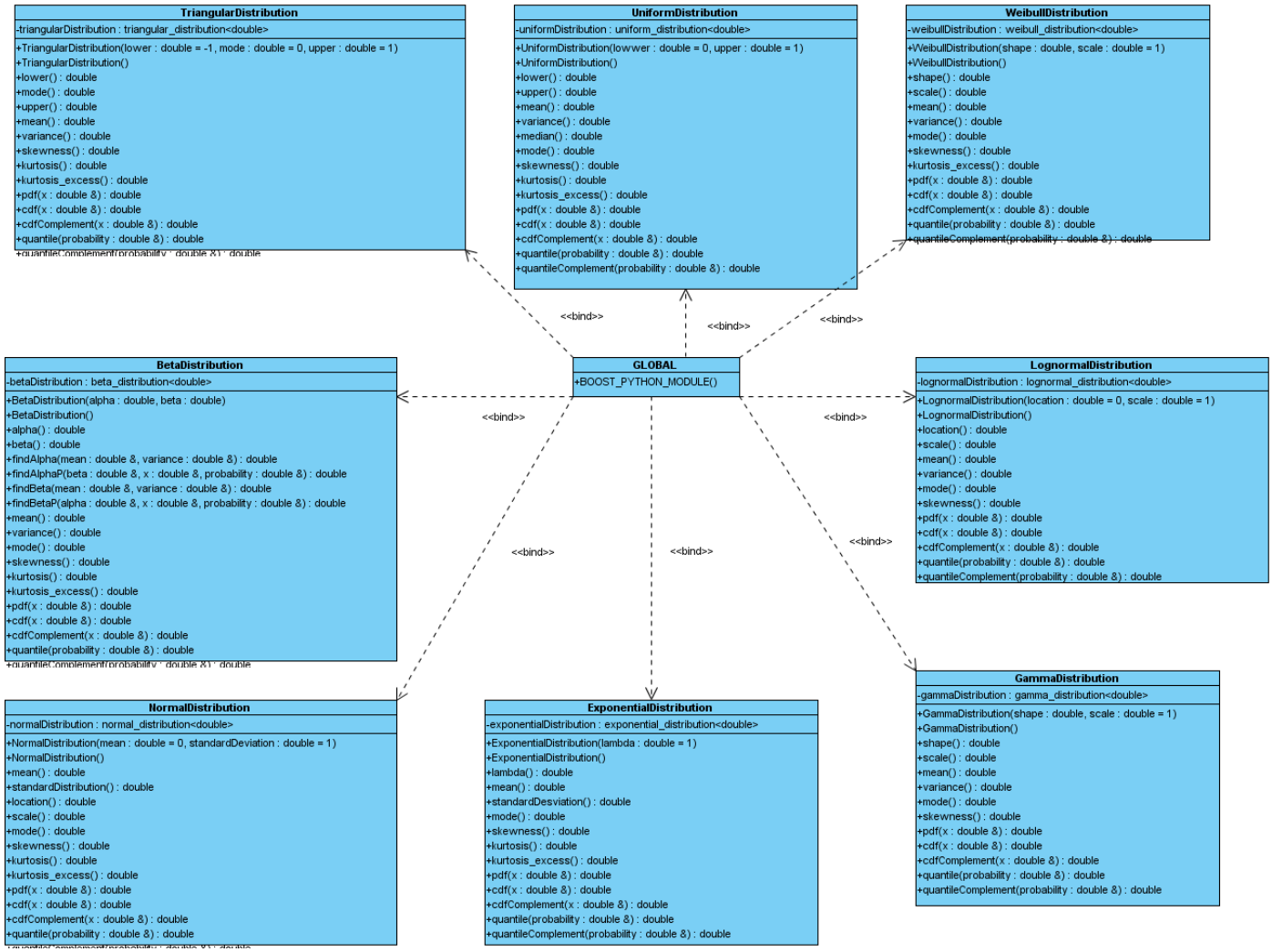

Ilustración 8: Diagrama de clases del diseño del subsistema pyprobdist.

2.3.2. Subsistema Simulater.

El subsistema herramienta de Simulación Matemática para Ingenieros en Confiabilidad (del inglés, Simulation Tool for Engineer Realisk) constituye el núcleo de la herramienta matemática, está compuesto por los paquetes: Montecarlo, probabilistic_distributions, goodness_fit_tests, utils, exc y test.

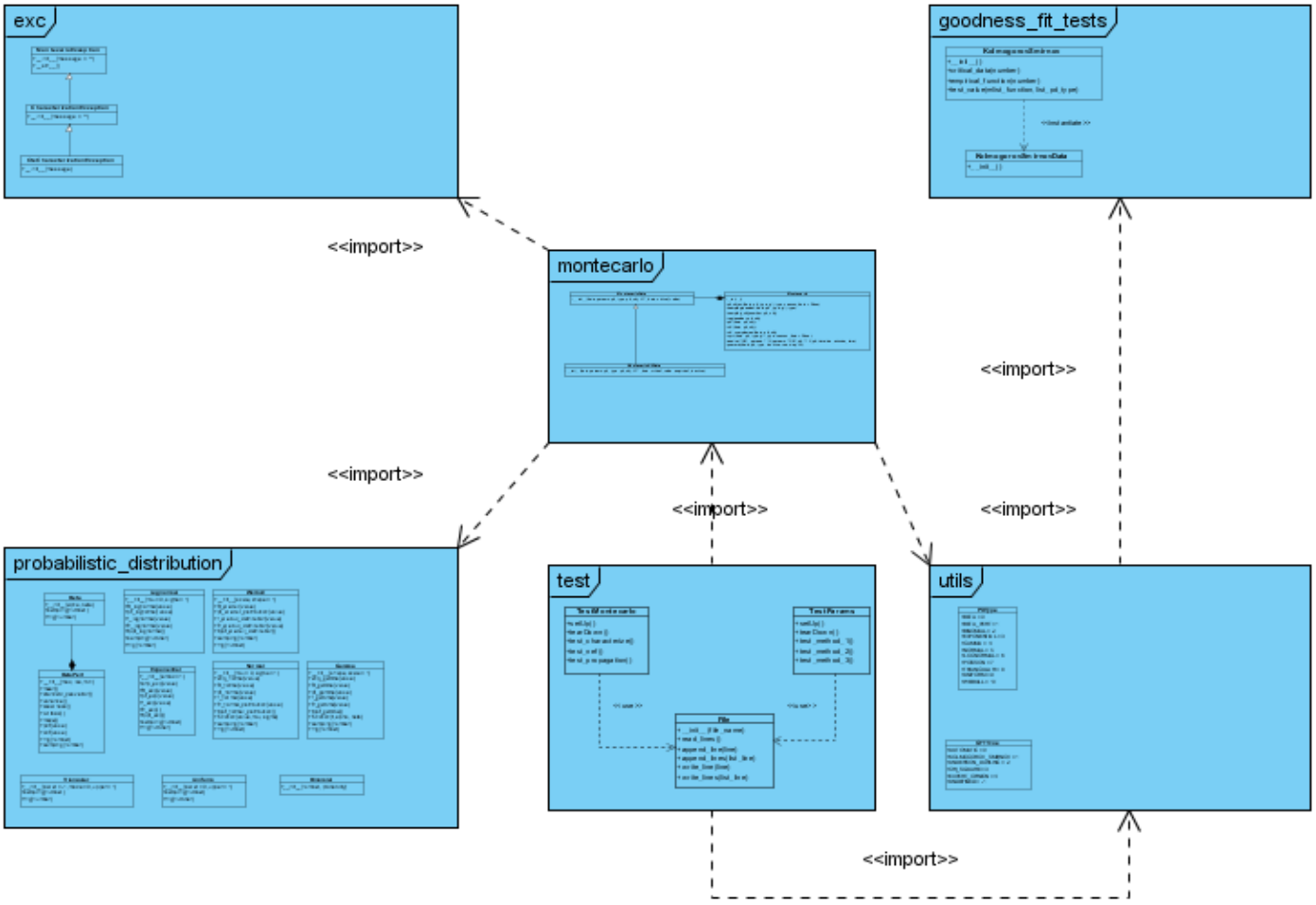

Ilustración 9: Diagrama de paquetes de simulater.

2.3.2.1. Paquete montecarlo.

Este paquete provee la interfaz de las funcionalidades brindadas por la biblioteca y la implementación de los algoritmos para la caracterización probabilística de variables aleatorias, propagación de la incertidumbre asociada a cada variable del modelo matemático y el cálculo de disponibilidad y número esperado de fallas para la ingeniería en confiabilidad, entre otros.

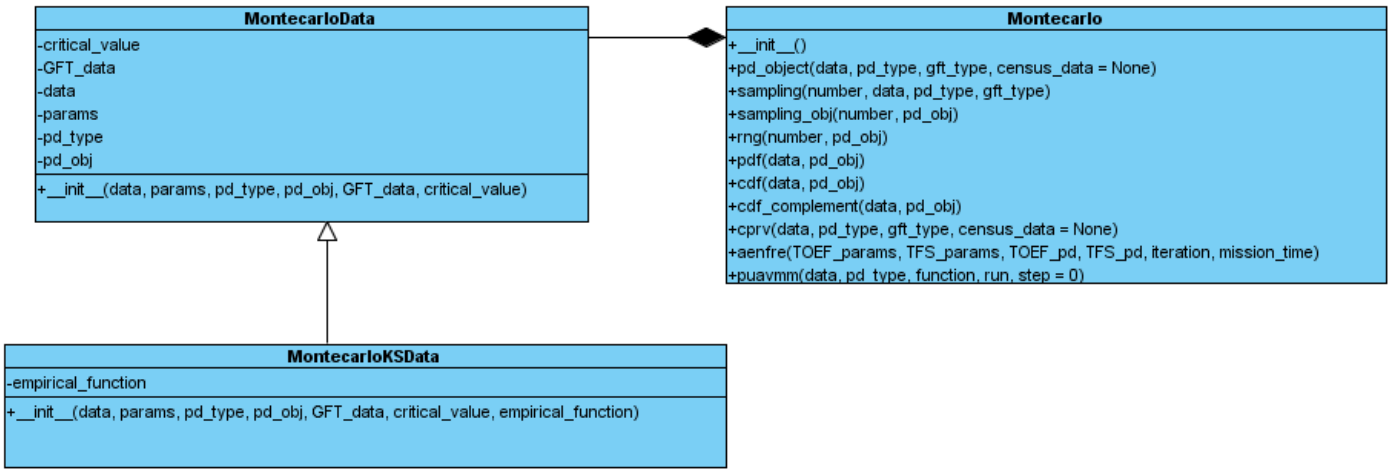

Ilustración 10: Diagrama de clases del diseño del paquete montecarlo.

Nombre: MontecarloData.	
Descripción: Clase genérica del modelo de datos para la Simulación de Montecarlo.	
Tipo de clase: Entidad.	
Atributo	Tipo
critical_value	float
GFT_data	list
data	list
params	list
pd_type	PDType
pd_obj	object
Para cada responsabilidad	
Nombre:	
Descripción:	

Tabla 2: Descripción de la clase MontecarloData.

Nombre: MonetcarloKSData.
Descripción: Clase del modelo de datos, para los datos resultantes de la prueba de bondad de ajuste Kolmogorov-Smirnov.

Tipo de clase: Entidad.	
Atributo	Tipo
empirical_function	list
Para cada responsabilidad	
Nombre:	
Descripción:	

Tabla 3: Descripción de la clase MontecarloKSData.

Nombre: Montecarlo.	
Descripción: Clase principal de la Simulación de Montecarlo.	
Tipo de clase: Controladora.	
Atributo	Tipo
Para cada responsabilidad	
Nombre:	pd_object(self, data, pd_type, gft_type, census_data=None)
Descripción:	Función que retorna una instancia de la distribución probabilística especificada y sus parámetros.
Nombre:	sampling(self, number, data, pd_type, gft_type)
Descripción:	Función que retorna una lista de números aleatorios siguiendo el comportamiento de la distribución que mejor se ajusta, de ser un valor constante se retorna el mismo número.
Nombre:	sampling_obj(self, number, pd_obj)
Descripción:	Función que retorna una lista de números aleatorios a través del método sampling dado una referencia de la distribución de probabilidad.
Nombre:	rng(self, number, pd_obj)
Descripción:	Función que retorna una lista de números

	aleatorios dado una referencia de la distribución de probabilidad.
Nombre:	pdf(self, data, pd_obj)
Descripción:	Función que retorna una lista de números calculados por la función de probabilidad.
Nombre:	cdf(self, data, pd_obj)
Descripción:	Función que retorna una lista de números calculados por la función de probabilidad acumulada.
Nombre:	cdf_complement(self, data, pd_obj)
Descripción:	Función que retorna una lista de números calculados por la función de probabilidad acumulada inversa.
Nombre:	cprv(self, data, pd_type, gft_type, census_data=None)
Descripción:	Función que retorna una tupla con los datos generados en el proceso de caracterización probabilística de la variable aleatoria.
Nombre:	aenfre(self, TOEF_params, TFS_params, TOEF_pd, TFS_pd, iteration, mission_time)
Descripción:	Función que retorna una tupla con la disponibilidad y el número esperado de fallas.
Nombre:	puavmm(self, data, pd_type, function, run, step=0)
Descripción:	Función que retorna una tupla con los datos generados en el proceso de propagación de la incertidumbre asociada a cada variable del modelo matemático.

Tabla 4: Descripción de la clase Montecarlo.

2.3.2.2. Paquete utils.

El paquete de útiles (del inglés, utils) proporciona a la biblioteca de simulaciones las funcionalidades necesarias para el cálculo de los parámetros de las distribuciones de probabilidad.

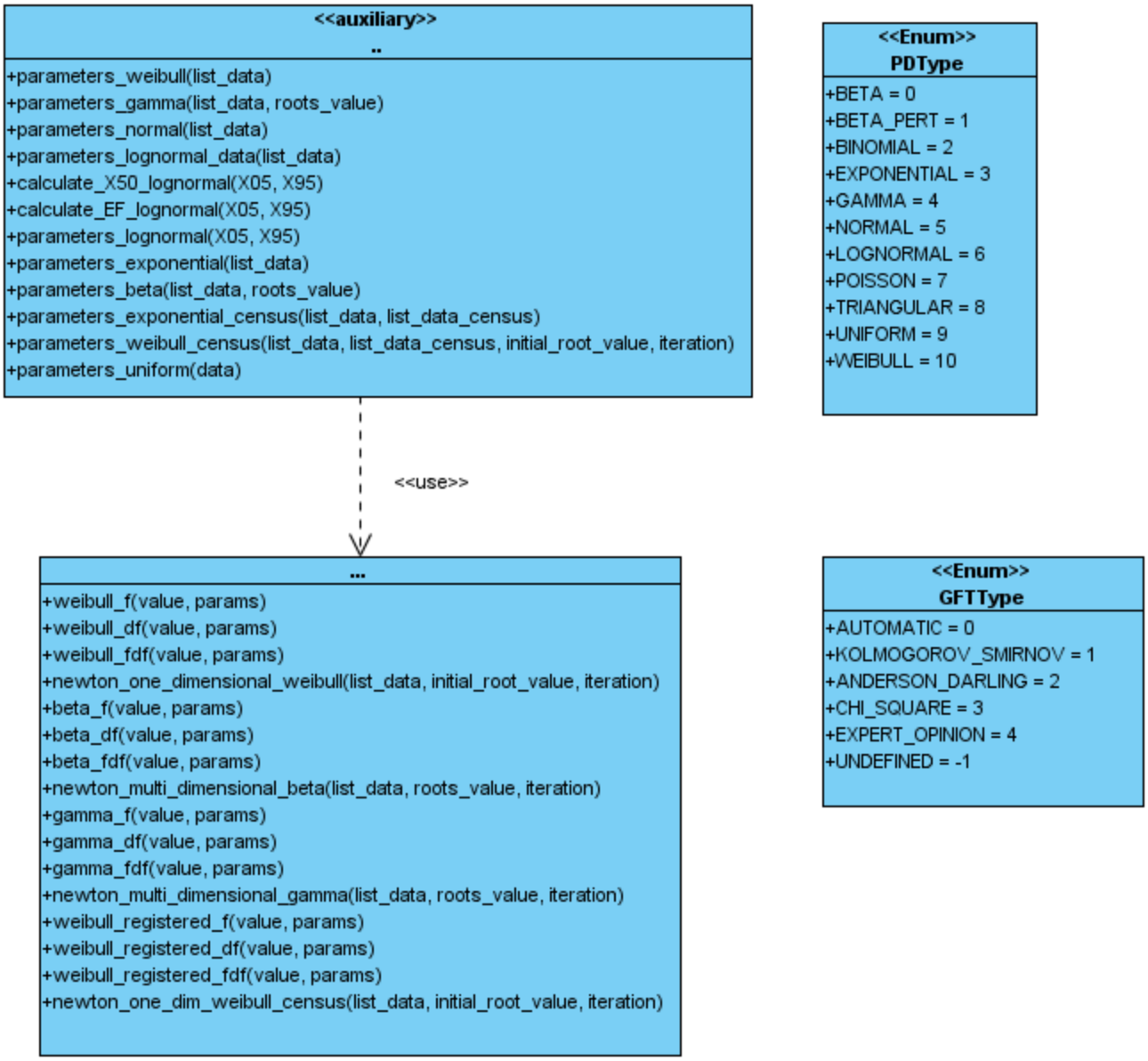

Ilustración 11: Diagrama de clases del diseño del paquete utils.

2.3.2.3. Paquete probabilistic_distribution.

El paquete de distribuciones de probabilidad (del inglés, probabilistic distributions) proporciona a la biblioteca de simulaciones para ingenieros en confiabilidad el conjunto de distribuciones de probabilidad necesarias para manejar el comportamiento de las variables aleatorias.

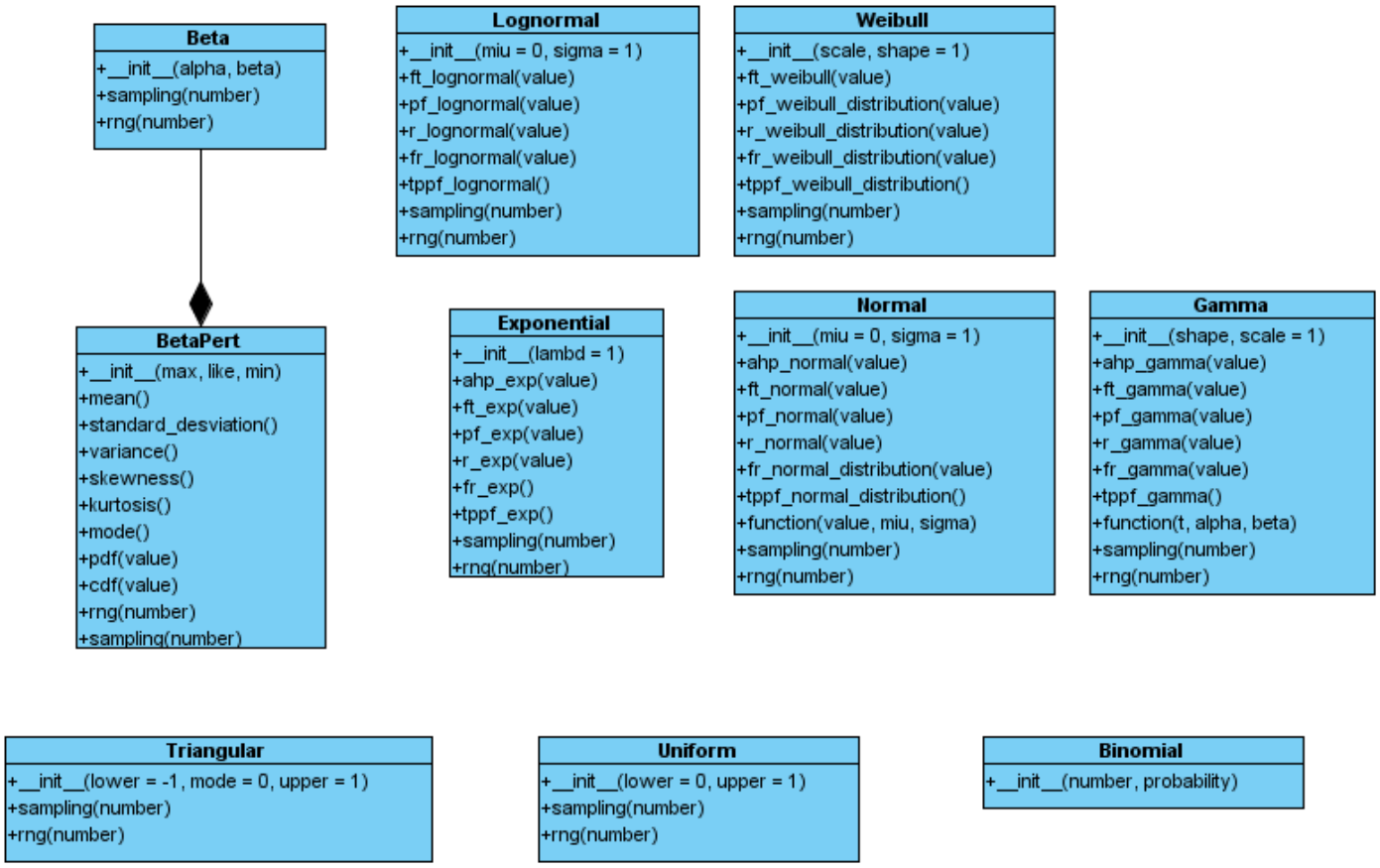

Ilustración 12: Diagrama de clases del diseño del paquete probabilistic_distribution.

2.3.2.4. Paquete goodness_fit_tests.

El paquete de pruebas de bondad de ajuste (del inglés, goodness fit test) provee a la caracterización probabilística de la biblioteca determinar la distribución que mejor se ajusta a los datos de una determinada variable aleatoria.

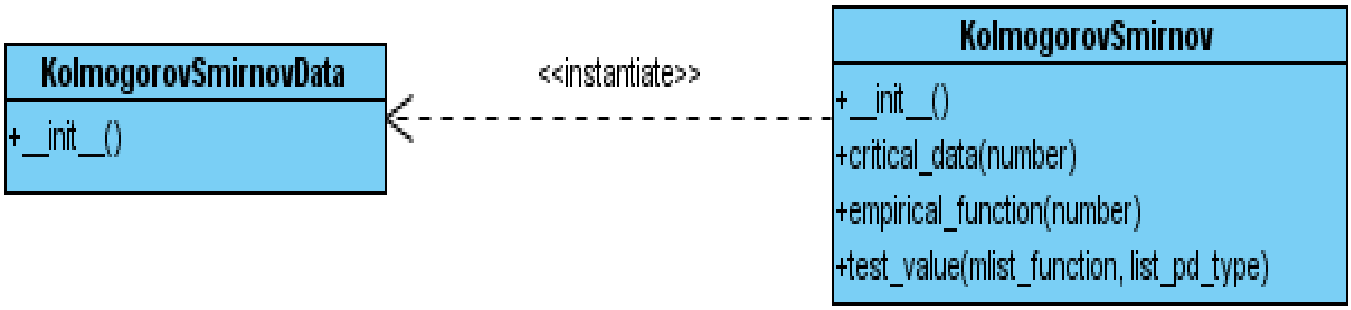

Ilustración 13: Diagrama de clases del diseño del paquete goodness_fit_tests.

Nombre: Kolmogorov_Smirnov.	
Descripción: Clase principal para la prueba de bondad de ajuste Kolmogorov-Smirnov.	
Tipo de clase: Controladora.	
Atributo	Tipo
Para cada responsabilidad	
Nombre:	critical_data(self, number)
Descripción:	Función que retorna una lista con los valores críticos de la prueba de bondad de ajuste de Kolmogorov-Smirnov.
Nombre:	empirical_function(self, number)
Descripción:	Función que retorna una lista con los valores calculados por la función empirical de la prueba de bondad de ajuste de Kolmogorov-Smirnov.
Nombre:	test_value(self, mlist_function, list_pd_type)
Descripción:	Función que retorna una tupla con los valores generados mediante la prueba de bondad de ajuste de Kolmogorov-Smirnov.

Tabla 5: Descripción de la clase KolmogorovSmirnov.

2.3.3. Subsistema Montecarlo.

El subsistema constituye el módulo principal de la integración de la biblioteca Simulater con la aplicación del proyecto SCIA. El mismo está estructurado por los subsistemas de presentación y lógica de negocio como propone la arquitectura trazada por el proyecto.

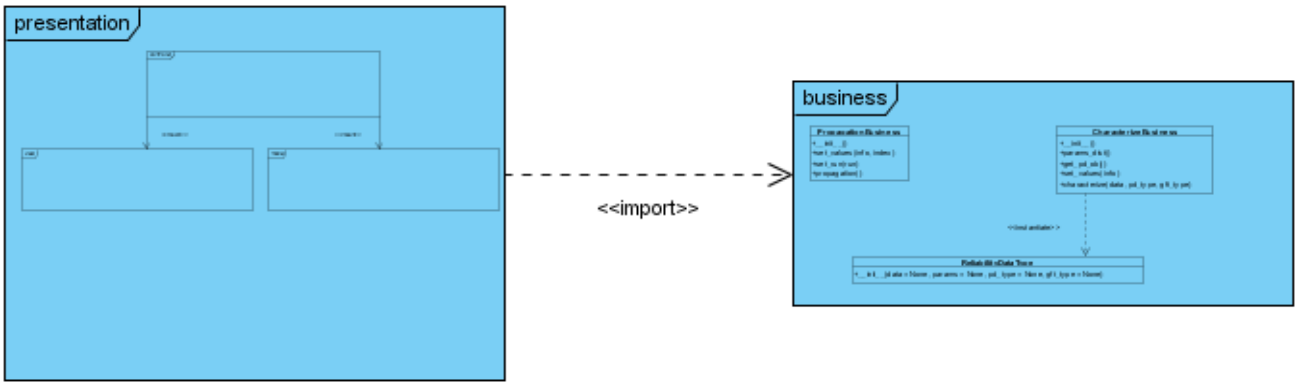

Ilustración 14: Diagrama de paquetes de la integración con la aplicación del SCIA.

2.3.3.1. Paquete presentation.

El paquete representa la capa de presentación, constituido por las interfaces de usuario, la lógica de las mismas y los modelos de datos.

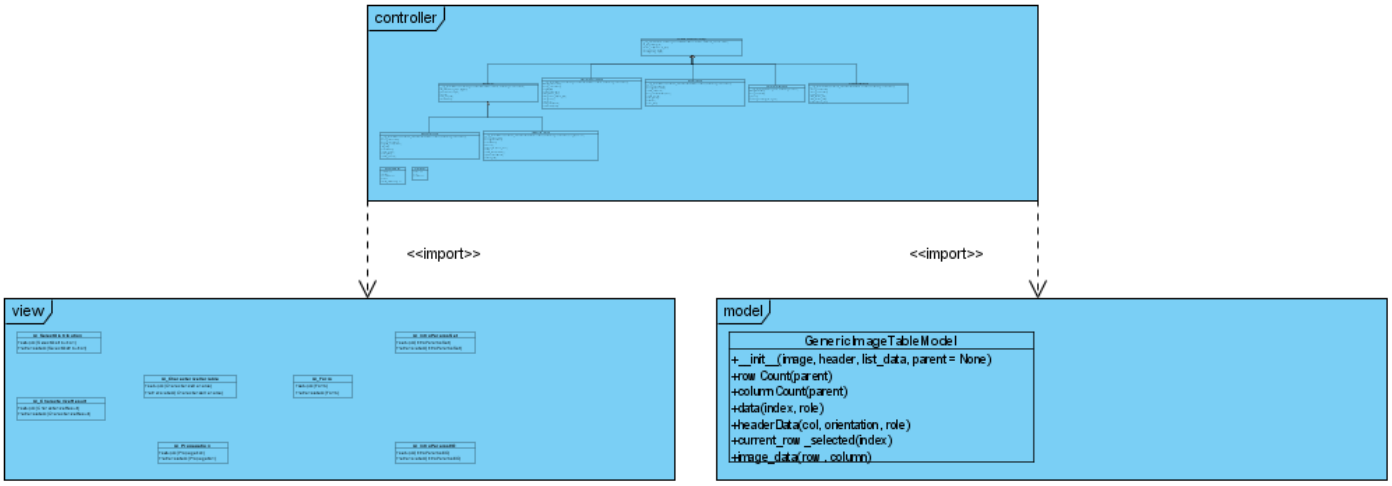

Ilustración 15: Diagrama de subpaquetes del paquete presentation.

2.3.3.1.1. Subpaquete controller.

El subpaquete de lógica de las interfaces de usuario contiene las clases encargadas de procesar los eventos provocados en la interfaz de usuario, actualizar y modificar el modelo que se esté visualizando siempre que sea necesaria.

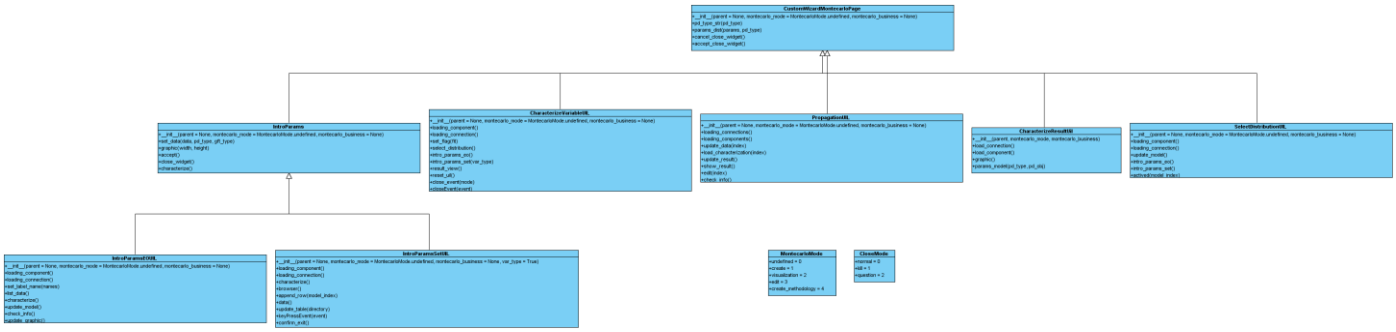

Ilustración 16: Diagrama de clases del diseño del subsistema controller.

2.3.3.2. Subpaquete business.

El subpaquete representa toda la lógica de negocio de la caracterización probabilística de variables aleatorias y la propagación de la incertidumbre asociada a cada variable del modelo matemático.

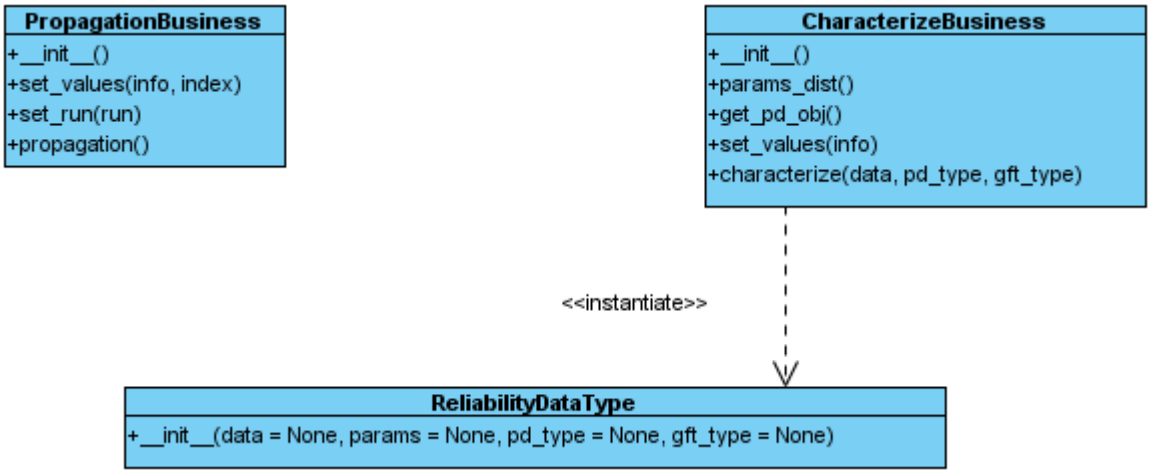

Ilustración 17: Diagrama de clases del diseño del subsistema business.

Nombre: CharacterizeBusiness.	
Descripción: Lógica de negocio de la caracterización probabilística a través de la simulación de Montecarlo.	
Tipo de clase: Controladora.	
Atributo	Tipo

Para cada responsabilidad	
Nombre:	params_dist(self)
Descripción:	Función que retorna los parámetros de las distribuciones.
Nombre:	get_pd_obj(self)
Descripción:	Función que retorna una instancia de la distribución probabilística que mejor ajusta.
Nombre:	set_values(self, info)
Descripción:	Función que permite cambiar los valores resultantes de la caracterización.
Nombre:	characterize(self, data, pd_type, gft_type):
Descripción:	Función que actualiza la instancia a la distribución probabilística que mejor se ajusta a través de la caracterización.

Tabla 6: Descripción de la clase CharacterizeBusiness.

Nombre: PropagationBusiness.	
Descripción: Lógica de negocio de la propagación de la incertidumbre de los modelos a través de la Simulación de Montecarlo.	
Tipo de clase: Controladora.	
Atributo	Tipo
Para cada responsabilidad	
Nombre:	set_values(self, info, index)
Descripción:	Función permite cambiar los valores resultantes de la propagación.
Nombre:	set_run(self, run)
Descripción:	Función permite cambiar la cantidad de corridas.
Nombre:	propagation(self)

Descripción:	Función que propaga la incertidumbre del modelo matemático.

Tabla 7: Descripción de la clase PropagationBusiness.

2.4. Modelo de Componentes General.

La herramienta de simulación de Montecarlo está estructurada por la biblioteca pyprobdist, Simulater y el módulo de integración Montecarlo.

La biblioteca pyprobdist encapsula y exporta al lenguaje de programación python las funcionalidades matemáticas de las distribuciones estadísticas a través de la biblioteca boost-python.

El módulo Simulater utiliza los algoritmos matemáticos de reducción de polinomios en varias dimensiones a través de los métodos de Newton que provee la biblioteca PyGsl para el cálculo de los parámetros de las distribuciones de probabilidad que le provee la biblioteca pyprobdist.

El módulo de Montecarlo depende de los recursos comunes de la aplicación SCIA, de las interfaces de usuario y los modelos de almacenamiento de datos en memoria que provee el framework Qt.

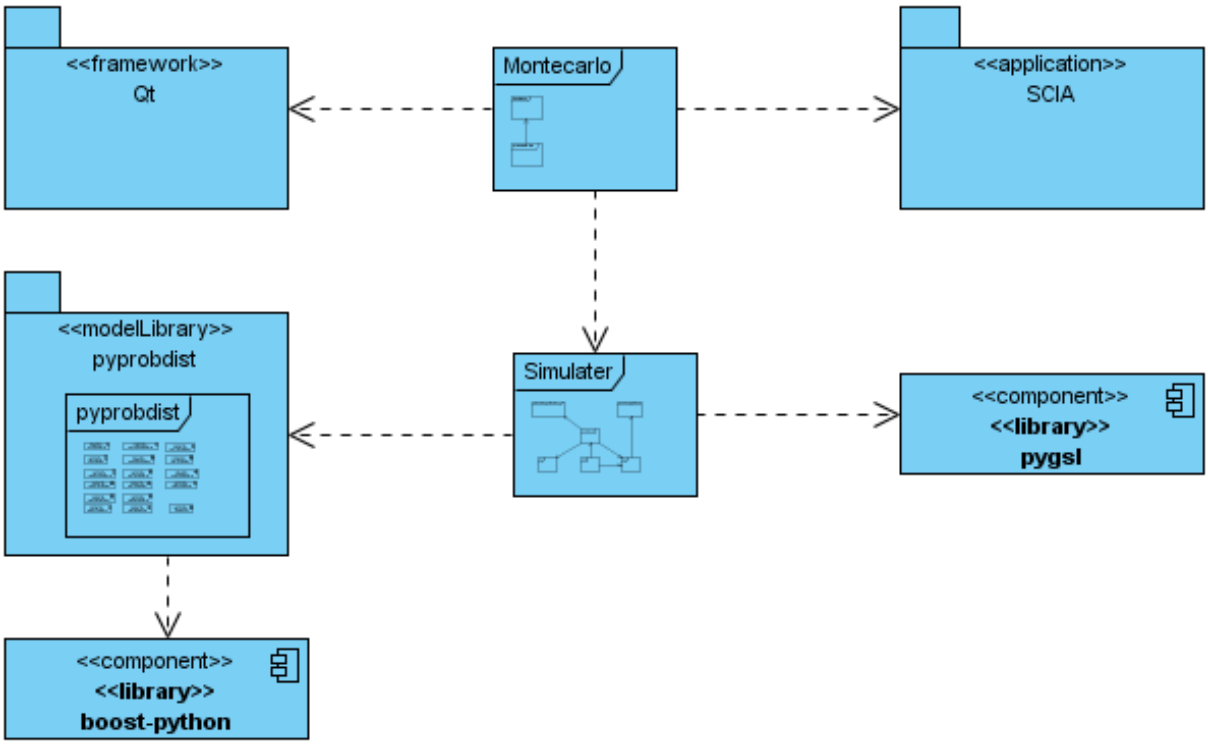

Ilustración 18: Diagrama de componentes de la herramienta de simulación de Montecarlo.

2.5. Modelo de Despliegue.

El modelo de despliegue o físico provee un modelo detallado de la forma en que los componentes de la herramienta se desplegarán a lo largo de la infraestructura del sistema. Para el despliegue de la biblioteca se necesita un computador para la ejecución de las funcionalidades.

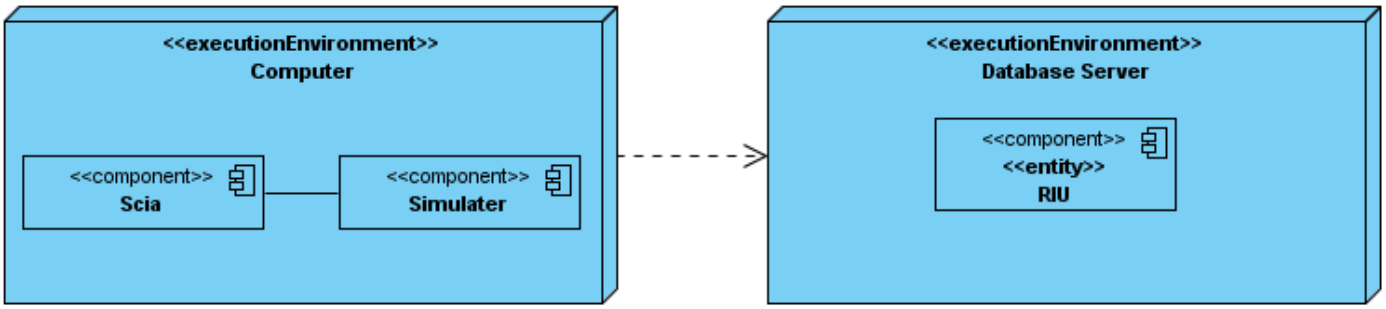

Ilustración 19: Diagrama de despliegue de la herramienta de Simulación de Montecarlo.

2.6. Estandarización y Documentación.

Un estándar de codificación es una tecnología, formato o método desarrollado, adoptado a través de proceso abierto de consenso, con la ventaja de facilitar la legibilidad, mantenibilidad, interoperabilidad y distribución del código fuente de cualquier aplicación. Los estándares abiertos son capaces de soportar y administrar en forma más sencilla y menos costosa la creciente complejidad de los sistemas, constituyendo una especie de "*garantía universal*" de compatibilidad, ajena a cualquier proveedor de la industria por sí mismo.

La estandarización del código fuente repercute directamente en lo bien que un programador comprende un sistema de software, la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento. Por lo que, si se aplica de forma continuada un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas, y posteriormente se efectúan revisiones del código de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

Como elemento de apoyo a las técnicas de codificación en la facilitación de la inteligibilidad del código se encuentra la documentación interna de un software. Documentar el código de un programa es añadir suficiente información como para explicar lo que hace, de forma que no sólo los ordenadores sepan qué hacer, sino que además los humanos entiendan qué están haciendo y por qué. En el mundo del software todo programa que tenga éxito será modificado en el futuro, bien por el programador original o por otro

programador que le sustituya, por lo que es importante que el programa se entienda para poder repararlo y modificarlo en el menor tiempo posible.

Con el objetivo de facilitar la comprensión, mantenimiento e inteligibilidad del código fuente de la herramienta de simulación de Montecarlo, se utiliza como estándar de codificación PEP-8, propuesta por Guido van Rossum creador del lenguaje de programación python y el script Doxypy de Doxygen para la generación automática de la documentación interna. Para la supervisión y evaluación del estándar de codificación durante el ciclo de vida del software se usa la herramienta Pylint.

Conclusiones.

En este capítulo se ha realizado un análisis de la arquitectura de los módulos de la herramienta mediante la revisión de los artefactos provenientes de fases anteriores del ciclo de vida del producto. Se enfatizó en la reusabilidad, extensión, reducción de costos de desarrollo y mantenimiento, facilidad de despliegue y actualización y reducción de la complejidad de los componentes de la solución desarrollada. Además de la descripción de la mayor parte de los componentes implementados, se seleccionaron los subsistemas más significativos en la solución y sus principales clases con el objetivo de una mayor comprensión del código fuente. También se abordó del estándar de codificación y la documentación del código fuente usada para facilitar la comprensión y distribución de la herramienta.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN.

Introducción.

Con el objetivo de validar la solución y obtener una aplicación con la calidad requerida, basado en los cuatro niveles de prueba propuestos por la metodología RUP (Unidad, Integración, Sistema y Aceptación), se implementan y ejecutan un conjunto de pruebas automatizadas que permiten detectar y corregir fallas desde las fases tempranas del desarrollo del software. Estas posibilitan evaluar el grado de cumplimiento respecto a las especificaciones iniciales, comprobando las funciones de cada uno de los módulos que componen el sistema, verifican la correcta unión de los componentes entre sí a través de sus interfaces, examinan que cada elemento encaje de forma adecuada, que se alcance la funcionalidad y el rendimiento del sistema, además de que el software esté listo y que pueda ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales fue construido.

3.1. Proceso de Pruebas.

3.1.1. Pruebas de Unidad.

En el proceso de construcción y validación del software, la implementación y realización de pruebas unitarias es, entre otros, uno de los pilares fundamentales. La creación de pruebas unitarias y su ejecución automatizada permiten dotar a dicho proceso, de confianza y calidad. Con estas pruebas no se encontrarán todos los errores que tenga el producto, pero ayudan a asegurar que cada una de las partes individuales que conforman el sistema funcionan correctamente.

3.1.1.1. Diseño de los Casos de Prueba.

Para la realización de las pruebas de unidad se utilizan como recursos físicos: un computador con microprocesador Intel Celeron de velocidad de la Unidad Central de Procesamiento (CPU del inglés, Central Processing Unit) de 2.0 GHz, Memoria de Acceso Aleatorio (RAM del inglés, Random Access Memory) de 1024 Mb y un disco duro con capacidad de 120 Gb y como recursos lógicos: sistema operativo GNU/Linux Debian Lenny con núcleo 2.6.26-2-686.

El documento de caso de prueba de unidad de Montecarlo se encuentra en la siguiente dirección:

http://trac.dst.pdvsa.com/scia/browser/docs/implementacion/pruebasUnitarias/casoPruebaMontecarlo.odt

Aspecto/Función a probar: Funcionalidades del módulo utils.				
Casos de uso asociados: Caracterización probabilística de variables aleatorias.				
Caso #	Pasos, Procedimiento o Precondiciones de ejecución de la prueba	Datos de Entrada	Resultados Esperados	Resultados Obtenidos
1	Funcionalidad: parameters_exponential(data)	data = valores del fichero Cipag103.txt	Lambda = 0.000682	Lambda = 0.00068229920909
2	Funcionalidad: parameters_gamma(data, roots)	data = valores del fichero Cipag103.txt roots = [3.0, 3.0]	Alpha = 1399.872280 Beta = 1.046976	Alpha = 1399.8722804400597 Beta = 1.0469759802662086
3	Funcionalidad: parameters_normal(data)	data = valores del fichero Cipag103.txt roots = [3.0, 3.0]	Alpha = 6.706095 Beta = 1.268026	Alpha = 6.706094828678709 Beta = 1.2680256953240452
4	Funcionalidad: parameters_weibull(data)	data = valores del fichero Cipag103.txt	Miu = 1465.632653 Sigma = 1432.375134	Miu = 1465.6326530612246 Sigma =

				1432.375133939511
5	Funcionalidad: parameters_exponential_census(data, census)	Data = valores del fichero Cipag103.txt census = [123, 124, 156]	Lambda = 0.000679	Lambda = 0.000678491809635
6	Funcionalidad: parameters_weibull_census(data, census, root, run)	Data = valores del fichero Cipag103.txt census = [123, 124, 156] root = 1.0 run = 100	Alpha = 1478.749528 Beta = 1.008365	Alpha = 1478.7495278709171 Beta = 1.0083649361784746
7	Funcionalidad: parameters_lognormal(x05, x95)	X05 = 1 x95 = 10	Alpha = 0.5 Beta = 0.294985	Alpha = 0.5 Beta = 0.29498525073746307

Tabla 8: Caso de Prueba del paquete utils.

Aspecto/Función a probar: Funcionalidades del módulo montecarlo.				
Casos de uso asociados: Caracterización probabilística de variables aleatorias. Cálculo de disponibilidad y número esperado de fallas. Propagación de la incertidumbre asociada a cada variable del modelo matemático.				
Caso #	Pasos, Procedimiento o Precondiciones de ejecución de la prueba	Datos de Entrada	Resultados Esperados	Resultados Obtenidos
1	Funcionalidad:	Params = valores del fichero	Weibull	Weibull

	cprv(params, pd_type, gft_type)	CIpag103.txt pd_type = None gft_type = None		
2	aenfre(TOEF_params, TFS_params, TOEF_pd, TFS_pd, run, mission_time)	TOEF_params = 2.0 TFS_params = 2.0, 10.0 TOEF_pd = PDType.EXPONENTIAL TOEF_pd = PDType.WEIBULL run = 15 mission_time = 8760	Disponibilidad = 0.002 <= x <= 0.002999 Número esperado de fallas = 37 <= x <= 41 Historial del número esperado de fallas = 35 - 48	Disponibilidad = 0.002645 Número esperado de fallas = 38 Historial del número esperado de fallas = 39
3	puavmm(params, pd_type, function, run)	Params = 2.0 pd_type = 2.0, 10.0 function = sum_function run = 100	Valor = 1 <= x <= 2	Valor = 1.6

Tabla 9: Caso de Prueba del paquete de montecarlo.

Las pruebas de unidad realizadas a la biblioteca de Simulación de Montecarlo fueron realizadas con éxito, comprobándose el correcto funcionamiento de cada una de las funcionalidades desarrolladas en el mismo.

3.1.2. Pruebas de Integración.

La necesidad de realizar las pruebas de integración viene dada por el hecho de que los módulos que forman un programa suelen fallar cuando trabajan de forma conjunta, aunque previamente se haya demostrado que funcionan correctamente de manera individual. Estas se realizan para comprobar si los módulos que están relacionados se ejecutan debidamente. Con su aplicación se va conformando el programa global a medida que se valida como los distintos componentes interaccionan y se comunican libres de errores. Para la validación de la integración se sometió a la solución a cuatro ciclos de pruebas, donde se comprobó el funcionamiento, interacción, comunicación y ejecución de los módulos relacionados, además permitió la corrección y refinamiento de la aplicación a distribuir.

3.1.2.1. Diseño de los Casos de Prueba.

A nivel de integración, las pruebas de aceptación de SCIA probaran la línea base de la arquitectura y el buen funcionamiento de la aplicación en sistemas operativos GNU/Linux y Windows, así como, la creación y visualización de datos para verificar el funcionamiento de la base de datos. Para la realización de las pruebas se utilizan como recursos físicos: un computador con microprocesador Intel Pentium 4 CPU 2.0 GHz, RAM de 1024 Mb y un disco duro con capacidad de 80 Gb y como recursos lógicos: sistema operativo GNU/Linux Debian Lenny con núcleo 2.6.26-2-686.

El documento de casos de prueba se encuentra en la siguiente dirección:

http://trac.dst.pdvsa.com/scia/browser/docs/integracion/planPruebasIntegracion.odt?rev=1262

#	Datos de entrada	Resultado Esperado	Resultado Obtenido	Validación (llenado por revisión)
1	Prueba de Instalación: Realizar una instalación del SCIA en una configuración de un único equipo con S.O Linux. Verificar que la línea base de arquitectura funcione correctamente (Conexión con base de datos)	Instalación realizada correctamente	Instalación realizada correctamente	Validado con éxito
2	Prueba de Instalación: Realizar una instalación del SCIA en una configuración de un único equipo con S.O. windows. Verificar que la línea base de arquitectura funcione correctamente (Conexión con base de datos)	Instalación realizada correctamente	Instalación realizada correctamente	N/A. No se posee aún un instalador para Windows.
3	Con la instalación de las máquinas en cada uno de los sistemas operativos se debe crear, modificar y eliminar una ubicación técnica del equipo	Creación, edición y eliminación correcta de la ubicación técnica, debe cumplir con las validaciones de completitud y exactitud de los datos.	Creación, edición y eliminación correcta de la ubicación técnica, debe cumplir con las validaciones de completitud y exactitud de los datos.	Verificado con éxito
4	Creación, modificación y eliminación	Creación, edición	Creación, edición	Verificado con

	de un equipo	y eliminación correcta de la ubicación técnica, debe cumplir con las validaciones de completitud y exactitud de los datos.	y eliminación correcta de la ubicación técnica, debe cumplir con las validaciones de completitud y exactitud de los datos.	éxito
5	Buscar Información de objetos técnicos	Buscar información de objetos técnicos creados en el RIU con éxito.	Buscar información de objetos técnicos creados en el RIU con éxito.	Verificado con éxito
6	Construir curva de impacto total	Curva de impacto total	Curva de impacto total	Verificado con éxito
7	Construir curva de riesgo	Curva de riesgo.	Curva de riesgo.	Verificado con éxito
8	Construir curva de costo	Curva de costo.	Curva de costo.	Verificado con éxito
9	Instalación y desinstalación del producto	El ambiente exactamente igual antes de la instalación.	El ambiente exactamente igual antes de la instalación.	Verificado con éxito

Tabla 10: Caso de Prueba de Integración de la aplicación.

De acuerdo a los parámetros de calidad de la disciplina de integración la construcción se toma como aprobado, pues no posee ningún error bloqueante hasta donde determinaron las pruebas de humo.

3.1.3. Pruebas de Sistema.

Con la finalidad de asegurar que el sistema está funcionando como fue establecido por los clientes y futuros usuarios del mismo, además de garantizar que aspectos específicos de su comportamiento tales como seguridad, rendimiento, fiabilidad y accesibilidad se cumplen satisfactoriamente, se implementaron y ejecutaron pruebas de sistema automatizadas, en un entorno lo más parecido posible al entorno final de despliegue.

Se aplicaron una serie de pruebas diferentes cuyo propósito primordial fue ejercitar profundamente el sistema. Dentro de ellas se encuentran las de funcionamiento para examinar si el software cumple con necesidades especificadas en el diseño, permitiendo verificar si el mismo lleva a cabo correctamente

todas las funciones requeridas. Otras pruebas realizadas fueron las de usabilidad para comprobar que tan fácil de usar para los usuarios es el producto desarrollado cuando entran en contacto con el mismo. Estas últimas posibilitan chequear aspectos como:

I. Aprendizaje (cuán rápido pueden realizar las tareas básicas la primera vez que interactúan con el sistema).
II. Eficiencia (una vez que han aprendido algo del sistema, que tan rápido pueden llevar a cabo las tareas).
III. Manejo de errores (cuántos errores comete el usuario, qué tan graves son éstos y qué tan fácil es para él recuperarse de ellos).
IV. Grado de satisfacción (qué tan satisfactorio es usar el sistema).

Para la validación del sistema se realizaron cuatro ciclos de pruebas, de ellos se automatizaron las pruebas de interfaz de usuario y de funcionalidad, tomando como valores de comprobación los arrojados por las herramientas de confiabilidad usadas en las industrias de PDVSA (Cristal Ball y APT-Montainer).

3.1.3.1. Diseño de los Casos de Prueba.

Para la realización de las pruebas de sistema se utilizan como recursos físicos: 5 computadores con microprocesador Intel Pentium 4 CPU 2.0 GHz, RAM de 1024 Mb y un disco duro con capacidad de 80 Gb y como recursos lógicos: sistema operativo GNU/Linux Debian Lenny con núcleo 2.6.26-2-686.

El documento de casos de prueba se encuentra en la siguiente dirección:

http://trac.dst.pdvsa.com/scia/browser/docs/prueba/construccionUno/documentacion/analisisDisenno/casoDePruebaAplicarPropagacionDeIncertidumbre.ods

http://trac.dst.pdvsa.com/scia/browser/docs/prueba/construccionUno/documentacion/analisisDisenno/casoDePruebaCaracterizarProbabilisticamenteVariablesAleatorias.ods

http://trac.dst.pdvsa.com/scia/browser/docs/prueba/construccionUno/documentacion/analisisDisenno/casodePruebaCalcularDisponibilidadYNumEspFallas.ods

Durante las ejecuciones de los primeros ciclos de prueba se detectaron algunas fallas que se corrigieron de inmediato por el equipo de desarrollo, actualmente el software no cuenta con fallos bloqueantes, ni críticos, por lo que cumple con los requisitos necesarios para su despliegue.

3.1.4. Pruebas de Aceptación.

Es la prueba final antes del despliegue del sistema. Tiene como propósito verificar que el software está listo y que puede ser usado por los usuarios finales. Se efectuaron estas pruebas con el fin de validar que el sistema cumple con el funcionamiento esperado y permitir al usuario que determine su aceptación, desde el punto de vista funcional, de rendimiento, seguridad de acceso a la aplicación, los datos y procesos, así como a los distintos recursos del sistema.

La solución se sometió a pruebas de aceptación alfa con el cliente en el área de desarrollo AIT-DST Mérida. Las mismas validaron el cumplimiento de los requerimientos y necesidades exigidas junto al equipo de desarrollo, permitiendo la corrección y refinación de errores e inconformidades en el ciclo de vida del software. Actualmente se comienza el despliegue de las pruebas de aceptación beta con los clientes de las áreas de PDVSA de Occidente y Oriente con expertos en el tema de Maracaibo y Maturín respectivamente.

Conclusiones.

En este capítulo se ha realizado la validación de la solución a través del desarrollo de diversas pruebas que permiten comprobar la completitud, correctitud, eficiencia, seguridad y estabilidad del sistema entre otros factores de gran importancia que contribuyen a la calidad del producto creado .Se aplicaron pruebas de unidad al código fuente durante la etapa de implementación, permitiendo identificar algunos errores cometidos en los algoritmos durante la codificación. De integración, que posibilitaron comprobar que la solución desarrollada funciona correctamente cuando es probada como un todo. Las de sistema permitieron verificar si este se comporta estable y ofrece una respuesta correcta a las necesidades funcionales. Por último, las de aceptación para que los usuarios dieran su aprobación sobre si el software es adecuado, fácil de manejar, soluciona sus necesidades y cumple sus expectativas.

CONCLUSIONES GENERALES

Para obtener el software resultante de esta investigación se transitó por varias etapas: primero la investigación del estado del arte, análisis de las herramientas existentes y las tecnologías a emplear, luego el análisis de la arquitectura, el diseño y la implementación de los subsistemas y por último las pruebas de unidad al código fuente.

En sentido general se obtuvieron resultados satisfactorios que dieron cumplimiento a los objetivos propuestos:

- ✓ Se diseñó e implementó la herramienta de Simulación de Montecarlo.
- ✓ Se diseñó e implementó la interfaz de usuario de la herramienta de Simulación de Montecarlo y su integración con la aplicación del proyecto SCIA.
- ✓ Se documentó y estandarizó el código fuente de la herramienta de Simulación de Montecarlo y su integración con la aplicación del proyecto SCIA.
- ✓ Se diseñó e implementó una suite de pruebas de unidad para la validación la solución.
- ✓ El proyecto SCIA cuenta con una herramienta de Simulación de Montecarlo para la caracterización probabilística de variables aleatorias, la propagación de la incertidumbre asociada a cada variable del modelo matemático de sus metodologías.
- ✓ Se desarrolló una herramienta de Simulación de Montecarlo multiplataforma, de código abierto y bajo los principios de la soberanía tecnológica.

RECOMENDACIONES

Cada día los ingenieros en confiabilidad requieren de nuevas funcionalidades. Por lo que se recomienda:

- ✓ Implementar las pruebas de bondad de ajuste por los algoritmos de Chi-Square y Anderson Darling.
- ✓ Ampliar la biblioteca de distribuciones de probabilidad de la herramienta.
- ✓ Implementar la automatización de la selección de la distribución de probabilidad que mejor se ajusta a un conjunto de datos sin la especificación de la prueba de bondad de ajuste.

REFERENCIA BIBLIOGRÁFICA

1. Yañes, Medardo, y otros, y otros. *Manual de Confiabilidad Integral.* 1ra. Maracaibo : Reliability and Risk Management, S. A., 2007. Vol. I. 1.

2. Gómez de la Vega, Hernado, Yañez, Medardo y Valbuena, Genebelin. *Gerencia de la Incertidumbre.* Maracaibo : RELIABILITY AND RISK MANAGEMENT S.A (R2M S.A), Julio 2003. 25.

3. Shannon, Robert y Johannes, James D. *Systems Simulation: The Art and Science.* Octubre, 1976. págs. 273-274. Vol. 6, Versión actual 12 Noviembre 2007.

4. Aspray, William. *John von Neumann y los orígenes de la computación moderna.* s.l. : Gedisa Editorial, 1992.

5. Valbuena, Genebelin, Yañez, Medardo y Gómez de la Vega, Hernando. *Ingeniería de Confiabilidad y Análisis Probabilístico de Riesgo.* Maracaibo : s.n., Junio 2003.

6. SimulAr. [En línea] http://www.simularsoft.com.ar/.

7. [En línea] http://www.biocyb.cs.ucla.edu/biocybmodeling.html.

8. [En línea] http://www.palisade.com/.

9. [En línea] http://www.investsign-home.com/.

10. Oracle. [En línea] http://www.oracle.com/appserver/business-intelligence/crystalball/index.html.

11. Yasai. [En línea] http://www.yasai.rutgers.edu/.

12. [En línea] http://www.goldsim.com/.

13. [En línea] http://www.cplusplus.com/.

14. [En línea] http://www.python.org/.

15. [En línea] http://www.boost.org/.

16. [En línea] http://www.eclipse.org/.

17. Nokia. [En línea] Nokia. http://qt.nokia.com/. 4.

18. [En línea] http://www.gnu.org/software/gsl/.

19. [En línea] http://www.visual-paradigm.com/.

20. IBM. [En línea] IBM. http://www-01.ibm.com/software/awdtools/rup/.

21. Craig, Larman. *UML y Patrones.* [trad.] Luz María Hernández Rodríguez. Montevideo : Universidad de la República de Uruguay, 2001. 3.

22. Microsoft Corporation. *Microsoft Application Architecture Guide.* 2da. s.l. : Microsoft Corporation, 2009. 2.

BIBLIOGRAFÍA

1. Yañez, Medardo, Semeco, Karina y Medina, Neyrih. *Enfoque Práctico para la Estimación de Confiabilidad y Disponibilidad de Equipos, con base en Datos Genéricos y Opinión de Expertos.* Maracaibo : Universidad Simón Bolivar, 2005.

2. R2M S.A. [En línea] http://www.reliarisk.com/.

3. [En línea] http://www.python.org/dev/peps/pep-0008/.

4. [En línea] http://interho.dynup.net/estandares/.

5. [En línea] http://www.brighton-webs.co.uk.

6. [En línea] http://pypi.python.org/pypi/doxypy/0.3.

7. [En línea] http:// www.doxygen.org.

8. [En línea] http://msdn.microsoft.com/es-es/library/aa291591(VS.71).aspx.

9. *AIT-DST Mérida.* [En línea] http://trac.dst.pdvsa.com/scia/browser/docs/requerimientos/especificacionRequerimientosSoftware.odt.

10. *AIT-DST Mérida.* [En línea] http://svn.dst.pdvsa.com/scia/trunk/Montecarlo.

11. *AIT-DST Mérida.* [En línea] http://trac.dst.pdvsa.com/productosExternos/Scia/.

12. *AIT-DST Mérida.* [En línea] http://trac.dst.pdvsa.com/scia/browser/trunk/arquitectura/pruebasDeConceptos/simulacionMonteCarlo.

13. *AIT-DST Mérida.* [En línea] http://trac.dst.pdvsa.com/scia/browser/docs/arquitectura/DASv1.2.odt.

14. *AIT-DST Mérida.* [En línea] http://trac.dst.pdvsa.com/scia/wiki/CasosDeUso.

GLOSARIO DE TÉRMINOS

Caracterización probabilística: determinar los atributos probabilísticos peculiares de alguien o de algo, de modo que claramente se distinga de los demás.

Código abierto: Es una tendencia internacional del desarrollo de software que profesa la distribución del código junto a las aplicaciones, se rigen por licencias tales como GNU/GPL.

Confiabilidad: es la probabilidad de que un componente, sistema o proceso realice en un intervalo de tiempo determinado lo que se requiere del mismo.

GNU: General Public License (Licencia Pública General).

GNU/GPL: GNU General Public License (Licencia Pública General de GNU).

Incertidumbre: grado de desconocimiento de un tema.

Modelos matemáticos: Representación en pequeño de algunos comportamientos similares en las matemáticas.

Multiplataforma: es un término utilizado frecuentemente en informática para indicar la capacidad o características de poder funcionar o mantener una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas.

PDVSA: Petróleo de Venezuela S.A.

Probabilidad: En un proceso aleatorio, razón entre el número de casos favorables y el número de casos posibles.

RUP: Rational Unified Project (Proceso Unificado de Desarrollo).

SCIA: Sistema de Confiabilidad Integral de Activos.

Simulación: Alteración aparente de la causa, la índole o el objeto verdadero de un acto o contrato.

Simulación de Montecarlo: La simulación a través del algoritmo matemático de Montecarlo.

Subsistema: Una agrupación de elementos, de los que algunos constituyen una especificación del comportamiento ofrecido por los otros elementos contenidos.

UCI: Universidad de las Ciencias Informáticas.

UML: Unified Modeling Language (Lenguaje Unificado de Modelado): Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

Variables aleatorias: Magnitud que puede tener un valor cualquiera de los comprendidos en un conjunto.