UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

FACULTAD DE INGENIERÍA

"SIMULACIÓN Y OPTIMIZACIÓN, APLICACIÓN EN SISTEMAS DE LÍNEAS DE ESPERA"

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA

INGENIERÍA DE SISTEMAS – INVESTIGACIÓN DE OPERACIONES

P R E S E N T A :

ACT. JOSÉ VÍCTOR CABALLERO RUIZ

TUTOR

DRA. MAYRA ELIZONDO CORTÉS

2012

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

JURADO ASIGNADO:

Presidente:	DR. ACEVES GARCÍA RICARDO
Secretario:	DR. RAMÍREZ RODRÍGUEZ JAVIER
Vocal:	DRA. MAYRA ELIZONDO CORTÉS
1er. Suplente:	DR. ORDORICA MELLADO MANUEL
2do. Suplente:	DRA. SÁNCHEZ LARIOS HÉRICA

Lugar o lugares donde se realizó la tesis:
CIUDAD UNIVERSITARIA.

TUTOR DE TESIS

DRA. MAYRA ELIZONDO CORTÉS

_____

FIRMA

AGRADECIMEINTOS.

Primeramente quiero agradecer a la Dra. Mayra Elizondo Cortés por su constante apoyo, comprensión y guía durante la elaboración de este trabajo.

Agradezco también:

A mis sinodales: Dr. Aceves García Ricardo, Dr. Ramírez Rodríguez Javier, Dra. Mayra Elizondo Cortés, Dr. Ordorica Mellado Manuel y a la Dra. Sánchez Larios Hérica por sus valiosos comentarios y aportaciones.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) el cual, a través de su beca, logró también que estos estudios fueran posibles.

A la universidad Nacional Autónoma de México por haberme recibido por segunda vez en mi vida en sus instalaciones, así como también a todos mis maestros y compañeros.

A mi hija Claudia que me ha cambiado la forma de ver la vida, y en el día a día me inspira a seguir creciendo como persona.

A mi esposa por su compresión, apoyo y trabajo diario para con nuestra familia.

ÍNDICE

ÍNDICE DE TABLAS Y FIGURAS

TABLAS.

FIGURAS.

INTRODUCCIÓN

Hablar del desarrollo de la simulación a través de la historia requiere obligadamente hacer referencia al “boom” computacional que vivimos desde mediados del siglo pasado hasta nuestros días. Así, los avances en el diseño de hardware y software han permitido a la simulación llegar a lo que conocemos hoy en día, posicionándose como una de las herramientas más utilizadas de la investigación de operaciones (I. de O.) para la toma de decisiones, tal vez solo antes de la estadística.

El auge de la simulación se debe principalmente a dos factores; el primero, su flexibilidad para modelar muchas de las situaciones que se presentan en la vida cotidiana, lo que se logra mediante la utilización de modelos lógicos, matemáticos e indiscutiblemente computacionales; la simulación nos da la posibilidad de analizar el comportamiento que tendrá un sistema y por supuesto anticipar situaciones que se podrían presentar en el futuro, todo esto con solo entender el funcionamiento del sistema en cuestión y trasladarlo a lo que se conoce como un modelo de simulación. Así, la simulación hace posible que se puedan experimentar y comparar distintos sistemas únicamente mediante el uso de su

diseño, aun cuando estos no hayan sido construidos o puestos en funcionamiento, hecho que nos lleva directo al segundo factor por el cual la simulación ha tenido un gran crecimiento a través de su historia, su relativo bajo costo comparado con otras técnicas de experimentación.

Sin embargo, la simulación desde sus inicios se ha enfocado, en gran medida, a la realización de análisis del tipo *"¿Qué pasa si?"*, análisis que se lleva a cabo mediante la experimentación y contraste de distintos escenarios que se presentan o pueden presentarse en un sistema. La naturaleza propia de este enfoque, que vale la pena señalar, es bajo el cual surge la técnica, así como las necesidades cada vez más exigentes de los problemas que confronta, hicieron evidente la necesidad de extender sus alcances cuando uno de los retos que plantea el problema es responder a premisas del tipo: "*¿Qué valores deben tomar determinadas variables en el modelo para alcanzar un óptimo en una característica (medida de desempeño) del sistema?".* En estos casos, es relativamente fácil discernir que la simulación por sí sola y utilizada con el enfoque tradicional, se convertiría tan solo en una herramienta que permitiría implementar una estrategia de ensayo y error para encontrar solución al problema, sobre todo en problemas donde el número de posibles soluciones (configuraciones de sistema a analizar) es muy alto.

Es aquí donde surge la idea de combinar o tratar de unir la simulación y la optimización para atacar este tipo de problemas, y donde también, gracias al desarrollo computacional, se da el surgimiento de una nueva área de investigación dentro de la investigación de operaciones, la "Simulación y Optimización" (SO). Este nacimiento, al menos en el campo de la práctica, se remonta ya a varias décadas atrás, cuando algunos investigadores comenzaron a trabajar y ver los beneficios de la combinación de ambas técnicas, como muestra, en 1977 Pedgen y Gately, desarrollaron un módulo de optimización para el software de simulación GASP IV. El desarrollo del módulo se basó en una variante del método Direct Search desarrollado por Hooke y Jeeves en 1961, de ahí hasta nuestros días el desarrollo constante en el área de la computación ha

permitido combinar la simulación con distintos avances en el área de la optimización dando cabida al nacimiento de varios softwares comerciales de SO.

El presente trabajo aborda el concepto SO, sus posibilidades y retos en el ámbito de la optimización de sistemas, y elaborara una propuesta de diseño encaminada a la aplicación del mismo en sistemas que pueden ser modelados con el uso de líneas de espera, esto, a partir del caso de estudio expuesto en el capítulo II del documento; para finalizar, en el último capítulo, se exponen algunos resultados obtenidos al aplicar la propuesta a una variante particular del caso de estudio, y en este sentido, se puede adelantar que los resultados muestran buena calidad sin necesidad de la exploración de todas las posibles soluciones al problema, que hay que mencionar, en la variante particular tratada, suman un total de 81,000.

OBJETIVOS Y ALCANCES

Este trabajo en “Simulación y Optimización” (SO), aplicada a sistemas que pueden ser representados mediante el uso de líneas de espera, tiene dos **objetivos fundamentales**:

I. Explorar el concepto, que hay que decir, pocas veces es abordado en las aulas, analizando la manera en la que surge, los enfoques que se le están dando actualmente, y por supuesto las ventajas y retos que plantea su utilización, al menos desde un punto de vista teórico, con respecto a la simulación y otras herramientas tradicionales de optimización de sistemas de la investigación de operaciones.

II. Desarrollar, con el uso de técnicas de diseño de software orientado a objetos, simulación orientada a objetos [1] - [2] y la heurística de búsqueda adaptativa conocida como algoritmos genéticos, una propuesta de diseño de software tendiente a aplicar el concepto SO a sistemas que pueden ser modelados con el uso de líneas de espera; esto, con ayuda de un caso de estudio en el que se recurre a un sistema de manufactura en donde se plantea la problemática

relacionada con el reto de establecer la mejor configuración inicial para la puesta en marcha del sistema.

Alcances:

1. Identificar las principales características de un diseño de un software SO.

2. Desarrollo, y documentación con estándares UML[1], de un diseño de software orientado a objetos que permita generar de manera dinámica, o lo que es lo mismo en tiempo de ejecución, modelos de simulación de sistemas que pueden ser imitados/emulados con el uso de líneas de espera, entre los que se encuentran por supuesto, el sistema de manufactura del caso de estudio.

3. Con base al diseño propuesto en el punto 3 y a la utilización de la heurística mencionada, desarrollar un modelo computacional e basado en el concepto SO que permita atacar el caso de estudio presentado.

Mediante esta propuesta se podrán visualizar los retos que se encaran en la implementación del concepto en este tipo de sistemas, cosa que no sería posible si únicamente se toma un software comercial para dar solución a una problemática específica.

[1] Lenguaje Unificado de Modelado, UML [9], por sus siglas en inglés

I. SIMULACIÓN Y OPTIMIZACIÓN (SO)

1.1 ORÍGENES DE LA SIMULACIÓN

Los orígenes de la simulación se remontan a la década de 1940, con la introducción del término “análisis de Monte Carlo”, hecha por John Von Neumann y Stanislaw Ulam. Término que fue utilizado para hacer referencia a los métodos matemáticos empleados al resolver problemas en el área de la física nuclear, que si hubieran sido resueltos de una manera experimental tradicional, hubieran sido demasiado costosos o imposibles de resolver. De ahí en adelante, la simulación ha sido ampliamente utilizada para resolver distintos problemas que por razones de costo, tiempo o complejidad matemática, serían muy difíciles de resolver de una manera analítica tradicional [3].

1.2 CONCEPTO

Existen varias definiciones de simulación sin embargo la concebida por Robert E. Shannon en 1975 sigue ajustándose a lo que de manera general se entiende al referirse a este concepto hasta nuestros días:

“Simulación es el proceso de diseñar un modelo de un sistema real y conducir experimentos con este modelo con el propósito ya sea de comprender el comportamiento del sistema o evaluar ciertas estrategias para la operación del sistema [4].”

Dentro de esta definición, e inmersos en el contexto mismo, se tienen dos conceptos clave en los que vale la pena ahondar *Sistema* y *Modelo*, además de otro que no se encuentra de manera explícita pero sí de manera implícita, *Toma de Decisiones*.

- Sistema

  Un sistema es un conjunto de elementos interconectados con funciones específicas trabajando con un fin común. En la definición se hace mención a un sistema “real”, se podría pensar que el concepto “real” se limita al ámbito de sistemas existentes, tangibles; sin embargo, desde el punto de vista de la simulación este concepto no es nada limitativo a la existencia o no del sistema, va más allá de ésta, abarcando cualquier sistema, existente o no, que pueda presentarse o no en el mundo real, es decir sistemas existente o no existentes.

- Modelo

  Un modelo es una representación abstracta de un sistema. Esta representación se construye mediante el uso de herramientas matemáticas,

lógicas y computacionales, su objetivo es, imitar el funcionamiento del sistema de la manera más apegada posible a la realidad.

- Toma de decisiones

Proceso mediante el cual, se selecciona un curso de acción de entre varias alternativas con base en la valoración de metas, objetivos y costos, que se desean alcanzar.

1.3 Modelo de Simulación

Un modelo, tal como se ha asentado anteriormente, es la representación abstracta y simplificada de un sistema. Es además, la parte medular de la simulación, el medio por el cual se imita el sistema y por tanto, se puede analizar su comportamiento bajo determinadas circunstancias (escenarios).

La construcción de un modelo de simulación es realizada mediante software, de manera que un modelo puede ser concebido como un conjunto de programas computacionales (código) que se ajustan a las características que definen el sistema y permiten imitar su comportamiento, y que por lo general deberán ser plasmadas en una etapa previa de diseño.

Es aquí donde se puede entender otra de las razones por las que la simulación ha llegado hasta donde se encuentra en nuestros días, y es que su desarrollo ha ido íntimamente ligado al desarrollo computacional que en la última mitad del siglo pasado ha experimentado un crecimiento sin precedentes. Las mejoras continuas que han experimentado hardware y software en las últimas décadas, han permitido hacer frente a sistemas más grandes y complicados, así como al manejo y procesamiento de volúmenes mayores de datos.

En la actualidad, al menos en la mayoría de las ocasiones, hay dos caminos para llevar a cabo la construcción de un sistema:

1. Mediante el uso software comercial[2], como pueden ser ARENA®, PROMODEL®, etc.

2. Mediante un desarrollo en la mayoría de los casos "a la medida", esto es, programando en algún lenguaje como C++®, FROTRAN®, Visual C®, Visual Basic®, etc.

La decisión de su utilización o no, es valorada por la persona encargada de la realización del modelo de simulación, con base en su experiencia, sistema que se está tratando y en la problemática que se afronta. Ambas opciones tienen "pros" y "contras", la utilización de un software comercial en el modelado del sistema tiene grandes ventajas como son:

- Reducir programación significativamente, pero no totalmente y hay que mencionarlo, esto depende muchas veces del tipo de software comercial que utilice. Dentro del software comercial hay dos vertientes:

    1. Software multipropósito, es decir, aquel que está diseñado para modelar algún tipo de sistema (por ejemplo discreto, continuo) independientemente del ámbito real donde se encuentre, es decir, no importa si éste está inmerso en un ámbito económico, social, productivo, científico.

    2. Software de propósito específico, aquellos que pretenden satisfacer la demanda de un sector específico, como pueden ser PROMODEL®, que está diseñado para el modelado de líneas de producción.

[2] Algunos criterios de selección son presentados en la siguiente referencia [28]

- Ayudar al proceso de verificación del modelo debido a que en muchas ocasiones el software es capaz de identificar automáticamente algunos errores en este ámbito.

- Facilitar el proceso de análisis de resultados, la mayoría del software comercial cuentan con un módulo de análisis estadístico.

- Por último, y en general como consecuencia de todos los puntos anteriores, disminuir el tiempo de desarrollo del proyecto. Por consiguiente los resultados siempre, o al menos en la mayoría de las ocasiones, deberían ser obtenidos en menor tiempo.

Sin embargo, hay que mencionar que la tarea de seleccionar el software no es algo que resulte fácil, primero está el hecho de que hay que investigar que software permite modelar el tipo de sistema y problemática que se está enfrentando y segundo, de entre este software habrá que valorar cuál nos permitirá llevar a cabo el estudio de manera “sencilla” y que además sea accesible al presupuesto del proyecto. Esto es, habrá que considerar también el costo de las licencias y capacitación, en muchas ocasiones aun cuando se encuentre el software “ideal” para llevar acabo el estudio, los gastos involucrados no justificarán su uso, debido en la mayoría de las ocasiones a las siguientes razones:

1. Tamaño del sistema involucrado;

2. No se cuenta con el presupuesto para costearlo;

3. El costo general rebasa al de un desarrollo a la medida.

Por estas razones no se podrá afirmar que el uso de software comercial es siempre la mejor decisión, se debe también estar abierto a considerar la segunda

opción, el desarrollo a la medida. Algunas razones que hacen interesante esta opción son:

1. En muchas ocasiones puede ser más simple o únicamente factible un desarrollo.

2. Mejor relación costo-beneficio.

3. Aprovechar al máximo recursos y conocimiento propios.

1.4 Posicionamiento de la Simulación en Nuestros Días

Dentro del ámbito de la investigación de operaciones, la simulación se ha consolidado como una de las herramientas de análisis más utilizadas para el soporte de toma de decisiones en varias áreas, que van desde las científicas hasta las productivas, administrativas y organizacionales.

Es un hecho que aun cuando la simulación es un proceso que requiere de varias disciplinas como son las matemáticas, probabilidad y estadística, diseño de experimentos; y en muchas ocasiones su implementación en la práctica requiere además de la planeación y administración de recursos, es la computación la disciplina que más íntimamente ligada está a la simulación, sin ella es natural pensar que la simulación no habría llegado hasta donde se encuentra en nuestros días, y en este sentido no se puede dejar de señalar que esto se debe, en gran medida, al crecimiento exponencial que ha tenido la computación a partir de 1950, donde las mejoras continuas de hardware y software, además de la aparición de las computadoras personales, han permitido el procesamiento y análisis volúmenes de información cada vez más grandes. Así, es además la computación el detonante que le permitirá seguir creciendo a grandes pasos en el futuro.

Como se ha señalado en el párrafo anterior es por un lado la computación la que ha llevado al crecimiento de la simulación y su extensa divulgación, sin embargo esto no hubiera sido posible si ella por sí misma no fuera valiosa desde su concepción, la computación ha ayudado como “una herramienta para perseguir el fin”, y es que realmente lo que la hace tan poderosa es el hecho de que:

1. Permite la experimentación con un sistema sin tener injerencia en el mismo, es decir el modelo de simulación no tiene absolutamente ninguna relación con el sistema real, lo que abre una amplia gama de posibilidades:

    - Experimentar diversos escenarios que podrían presentar en un sistema.

    - Experimentar con sistemas inexistentes.

    - Comparar dos sistemas que tienen un mismo objetivo aun si uno o ambos no existen.

    Y por supuesto estas posibilidades redundan también en grandes beneficios:

    - En muchas ocasiones es imposible o no factible la experimentación en cierto tipo sistemas, un ejemplo podrían ser los que se presentan en las áreas médica, armamentista, etc., en estos casos la simulación puede ser muchas veces no solo una alternativa sino la única para el análisis de estos sistemas.

    - Reducción significativa de costos, en la gran mayoría de los casos la construcción de un modelo de simulación es más barato, por ejemplo parar y ajustar líneas de producción con el único fin de experimentar podría resultar muy caro para una organización.

2. No existen restricciones en el sistema a tratar, solo se necesita que sea factible poder generar un modelo de matemático/computacional del mismo, es decir permite la experimentación con casi cualquier tipo de sistema, los problemas prácticos en el mundo real a menudo contienen condiciones de no linearidad, relaciones combinatorias e incertidumbre que a través del tiempo se ha demostrado son mucho más fáciles de representar en un modelo de simulación que en un modelo matemático.

3. Permite a los analistas tomar control del tiempo, en otras palabras, permite observar cómo se comportará el sistema, bajo determinadas circunstancias (escenarios), en largos periodos de tiempo que pueden ir desde semanas hasta meses, cosa que gracias a los avances en el área de la computación puede tomar solo segundos o minutos.

1.5 Simulación y Optimalidad en la Investigación de Operaciones

La investigación de operaciones es un área de la investigación científica que hace acopio de una serie de técnicas y algoritmos matemáticos para facilitar la toma de decisiones, mediante la representación de un sistema a través de un modelo analítico "conocido", en consecuencia aplica distintas técnicas/métodos/algoritmos para su análisis, la mayoría de ellas encaminadas a descifrar una configuración que permita el "óptimo" funcionamiento del sistema basado en los recursos disponibles y parámetros de funcionamiento.

La I. de O. ha sido desarrollada por décadas fundamentalmente en torno a modelos matemático/analíticos del siguiente tipo:

Max (Min) $f(x_1, x_2, \ldots, x_n)$

Sujeto a:

Restricciones sobre $(x_1, x_2, \ldots, x_n)$

Donde la función $f(x_1, x_2, \ldots, x_n)$ y las restricciones sobre $(x_1, x_2, \ldots, x_n)$ son planteadas a través de diversas expresiones analíticas.

Como se observa el modelo está totalmente basado en la búsqueda de optimalidad. Cuando la I. de O. aborda una problemática de optimización en un sistema, ésta trata de llevar dicha problemática a un modelo analítico conocido del tipo anterior, para, una vez realizada esta tarea, y con base en las características particulares del sistema que se está tratando, se pueda aplicar alguna de las siguientes técnicas de programación matemática, tradicionales de la I. de O.:

- ✓ Programación lineal
- ✓ Programación entera
- ✓ Programación no-lineal
- ✓ Programación estocástica
- ✓ Programación geométrica
- ✓ Programación dinámica

O bien alguna otra, como por ejemplo:

- ✓ Redes
- ✓ Teoría de decisiones
- ✓ …

Estas técnicas en conjunto son conocidas dentro de la I. de O. como **herramientas de optimización;** han sido desarrolladas y estudiadas durante años y cuentan con algoritmos matemáticos ampliamente probados para

encontrar una o varias soluciones al problema e inclusive identificar, si es el caso, si no existe solución al mismo.

Esto suena bien cuando el sistema sujeto de estudio puede ser “fácilmente” llevado a un modelo analítico de este tipo, y ajustado a alguna de las técnicas antes mencionadas; si es así, bastará con aplicar un algoritmo, y dependiendo también de la complejidad computacional del mismo, se encontrará en la mayoría de las situaciones, una solución en un tiempo razonable; sin embargo, en la vida real muchos de los sistemas no son fáciles de modelar analíticamente ya que son demasiado complejos. En muchas ocasiones, si nos obstinamos en hacerlo, nos veríamos en la necesidad de dejar a un lado aspectos clave de su naturaleza solo por hacer del mismo, una versión tratable desde un punto de vista analítico. A menudo los problemas prácticos contienen condiciones de no linealidad, relaciones combinatorias e incertidumbre que no pueden ser modeladas fácil ni efectivamente, a través de una función objetivo y un conjunto de restricciones; es en estos casos donde la I. de O. se ha auxiliado durante años de otras herramientas, conocidas como **herramientas de análisis**; solo por mencionar algunas de las más importantes tenemos:

- ✓ Simulación
- ✓ Teoría de Colas
- ✓ Cadenas de Markov
- ✓ Estadística
- ✓ …

Se dice que estas herramientas son de análisis, porque su objetivo no es atacar el modelo matemático de optimalidad que se ha mencionado, sino más bien, ahondar y predecir el comportamiento de un sistema bajo ciertos supuestos. Entre estas técnicas, la simulación destaca como una de las más completas y utilizadas en el análisis de sistemas, esto debido a su flexibilidad y a que en ella también se incorporan otras herramientas de análisis como pueden ser la teoría de colas y la estadística, entre otras.

Sin embrago, hay que decir, que la simulación había sido de alguna manera limitada dentro de la I. de O. hasta un par de décadas, la razón principal es que aun cuando, por un lado se tenía el hecho de que era mucho más sencillo representar un sistema a través de un modelo de simulación que a través de un modelo matemático-analítico, mediante la simulación no era posible atacar directamente problemas de optimalidad que se presentaban cada vez más frecuentemente en el mundo real. En muchas ocasiones la simulación se vislumbraba como el mejor, o muchas veces el único camino para poder realizar algún análisis sobre un sistema real, pero, la simulación por sí sola, únicamente permite valorar el comportamiento del sistema bajo determinado escenarios, llevando a cabo análisis del tipo *¿qué pasa si?,* por lo que su utilización se veía limitada solo como una herramienta de análisis. En la actualidad, hay que decirlo, son muchas las ocasiones en las que, ya sea por razones de competitividad en el caso de las empresas o por diversos fines en las áreas de investigación, cada vez son más frecuentes los problemas en los que se requiere alguna optimización de recursos. De tal forma que, muchas veces el objetivo del análisis va mucho más allá de la evaluación y confrontación de algunos escenarios en el sistema, y requería de contestar preguntas del tipo:

"*¿Cuáles son los valores que determinadas variables deben tomar para encontrar un óptimo de alguna característica del sistema?"*

En estos casos, y sobre todo cuando son muchos los escenarios entre los que se tiene que buscar este óptimo en la característica del sistema, era claro que por sí misma, la simulación no era una herramienta suficientemente poderosa para atacar la problemática, fue entonces cuando bajo la premisa:

"*¿Es posible aprovechar las ventajas que tiene el analizar un sistema mediante un modelo de simulación vs. su análisis mediante un modelo matemático-analítico tradicional de I. de O., cuando el objetivo del análisis es la búsqueda de optimalidad?"*

Se hizo necesario pensar en extender sus posibilidades; esto dio origen al nacimiento del concepto Simulación y Optimación (SO), el cual busca poder optimizar alguna característica de un sistema a partir de un modelo de simulación. En este sentido la SO presenta uno de los principales retos que tendrá la simulación en el futuro, éste es la búsqueda de la **Optimalidad**.

1.6 ORÍGENES DE LA SIMULACIÓN Y OPTIMIZACIÓN (SO)

Los orígenes de la Simulación y Optimización (SO) se remontan a la década de 1970, cuando en 1973 Dennis E. Smith sentara las primeras bases conceptuales para un software que automáticamente optimizará sistemas simulados, sugiriendo que un optimizador automático de este tipo debía ser una aplicación computacional externa al modelo de simulación. El optimizador que Smith visualizaba debía usar entradas y salidas del modelo, así como información proveída por el usuario para determinar un óptimo en el sistema, además de poseer la "inteligencia" requerida para determinar el método de optimización apropiado.

En 1977, Dennis Pegden y Michael Gately, llevaron los conceptos de Smith al mundo real siendo los primeros en desarrollar y reportar formalmente la liga entre un software de simulación discreta y un algoritmo de optimización, desarrollando un módulo de optimización para el software de simulación GASPIV, y más adelante (1980) para el software SLAM. Ambos módulos de optimización fueron basados en una variante del método de búsqueda directa (direct search method) desarrollado por Hook and Jeeves en 1961.

Aun cuando el concepto SO no es algo nuevo, el primer intento de llevar este concepto a una herramienta de simulación comercial para su explotación se dio en 1995 con la llegada de SimmRunner® de PROMODEL® (software comercial de simulación discreta), tres décadas después de que se comenzara a trabajar con el

concepto, debido a varios factores, entre ellos el desinterés de las compañías de software de simulación comercial en el área, por creerla poco aplicativa y por tanto no muy comúnmente usada. A partir de ese año, más empresas de software de simulación empezaron a preocuparse por incluir este concepto a sus desarrollos. Actualmente, son varios los sistemas de simulación discreta que cuentan con una herramienta SO.

1.7 Simulación y Optimización (SO)

El término simulación y optimización, dentro del contexto de la I. de O., se refiere a un área de la investigación científica que se ha dado a la tarea de unir a la simulación y la optimización, técnicas que comúnmente se han utilizado de manera aislada una de la otra para la toma de decisiones, con el fin de generar una nueva herramienta mucho más poderosa y robusta para el ataque de problemas de optimización de sistemas.

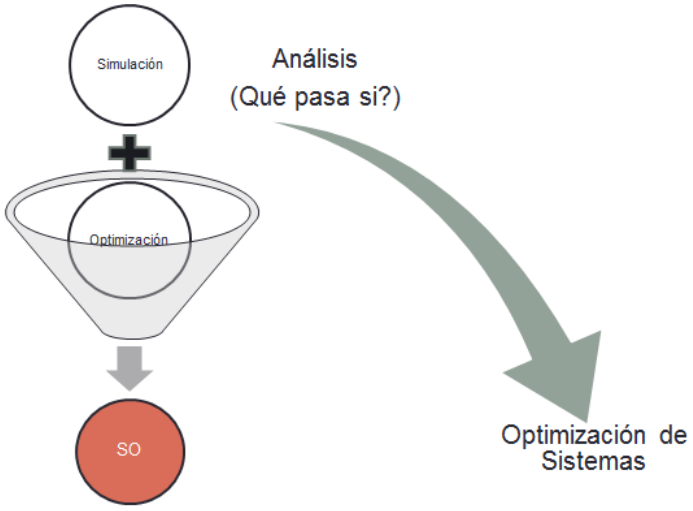


Figura 1. Simulación de Herramienta de Análisis a Optimización de Sistemas

La SO basa su funcionamiento en la explotación de las bondades y beneficios de la simulación para el modelado y evaluación de sistemas, mismos que ya han sido abordados en el presente documento, pero desde un enfoque de búsqueda optimalidad y no meramente de análisis. La SO pretende atacar y fortalecer las debilidades que presenta la técnica de simulación, cuando el objetivo de un estudio/proyecto involucra búsqueda de optimalidad. Esto se hace posible mediante el uso y aplicación de la optimización. El concepto SO pretende llevar a la simulación de una herramienta de análisis de escenarios a una herramienta de optimización de sistemas.

1.8 Problemática general abordada por la Simulación y Optimización (SO)

Como ya se ha mencionado el objetivo fundamental de la SO es atacar problemas de optimización de sistemas, sin embargo la problemática que aborda puede ser plantada de manera mucho más formal mediante el uso del siguiente modelo:

$$Max \text{ o } Min\ E[S(X_1,X_2,X_3,\ldots Xn)]$$

Sobre todas las posibles (factibles) combinaciones de $x_1, x_2, x_3, \ldots x_n$.

Antes de analizar los componentes del mismo, es pertinente hacer una pequeña pausa/reflexión, y observar que este modelo encaja perfectamente en el modelo básico de optimización, en torno al cual han sido desarrolladas muchas de las de las diversas técnicas/herramientas de optimización de la I. de O., y es así que la SO puede ser también catalogada como una herramienta de optimización dentro de la I. de O.

Regresando al modelo, tenemos que:

$X_1,X_2,X_3, \dots , X_n$ Es un vector compuesto por las **variables de entrada** del modelo de simulación. Los distintos vectores $(x_1,x_2,x_3,\dots, x_n)$ que se pueden obtener como resultado de la combinación de los valores que pueden tomar estas variables en el modelo representan los escenarios que se pueden presentar en el sistema de interés. En el contexto de la SO las variables que conforma este vector son cocidas como **variables de decisión.**

$S(X_1,X_2,X_3, \dots , X_n)$ Representa los posibles valores que toma(n) la(s) **medida(s) de efectividad o desempeño** que se busca optimizar en el sistema. En el contexto de la SO $S(X_1,X_2,X_3,\dots, X_n)$ es conocida como **función objetivo**, sin embargo vale la pena mencionar que no es una función con representada de analítica.

$S(x_1,x_2,x_3,\dots, x_n)$ representa el valor que toma(n) la(s) medida(s) de efectividad después de correr el modelo de simulación del sistema bajo el escenario $x_1,x_2,x_3,\dots, x_n$, por tal motivo es claro ver que en la mayoría de los casos $S(x_1,x_2,x_3,\dots, x_n)$ *es* una variable o vector aleatorio resultado de un experimento de simulación.

$E(S(X_1,X_2,X_3, \dots , X_n))$ Es la esperanza de $S(X_1,X_2,X_3,\dots, X_n)$, en este caso, la esperanza del valor que toma(n) la(s) **medida(s) de efectividad**, o en el contexto de la SO función objetivo.

El modelo representa la problemática de maximizar o minimizar $E(S(X_1,X_2,X_3, \dots , X_n))$ sobre todas las posibles (factibles) combinaciones de $x_1,x_2,x_3, \dots , x_n$. Vale la

pena señalar que la factibilidad de una solución, al igual que en otras técnicas de optimización, estará basada en distintas restricciones sobre las variables de decisión que pueden ser tan simples como cotas inferiores o superiores, hasta restricciones combinatorias, lineales y aún más complejas.

Es así como la SO puede ser vista como un proceso a través del cual se busca encontrar un escenario, o configuración de sistema, que optimice la esperanza de una o varias medidas de efectividad o desempeño en un sistema mediante el uso de un modelo de simulación.

Es también muy importante observar que el hecho de estar optimizando una esperanza probabilística, además de añadir complejidad al problema, deja ver que los esfuerzos de la SO estarán, en la mayoría de las ocasiones, más encaminados a dar una muy buena estimación de la solución del problema, que el óptimo con exactitud, aunque en algunas ocasiones el óptimo exacto podría ser encontrado.

1.9 Búsqueda de Optimalidad en ámbito de Sistemas, Simulación y Optimización (SO) vs. otras Técnicas de la I. de O.

Una vez que se ha situado a la SO en el área de optimización de sistemas dentro de la I. de O. es importante señalar las características que la distinguen de las herramientas que comparten con ella esta área, como son las distintas técnicas de programación matemática:

- La primera característica, y que se podría catalogar como principal, se hace evidente al analizar el camino que utilizan las herramientas de optimización antes mencionadas para atacar la problemática de

optimización de sistemas; nos referimos al *método utilizado para representar el sistema* y por consiguiente para "medir" su comportamiento.

- La segunda es la *relación que guarda el algoritmo de optimización con el modelo utilizado para la representación del sistema*.

A continuación se presenta un esquema que muestra con mayor detalle la manera en que es abordado un problema de optimización de sistemas con las herramientas de optimización tradicionales de la I. de O. vs. SO.

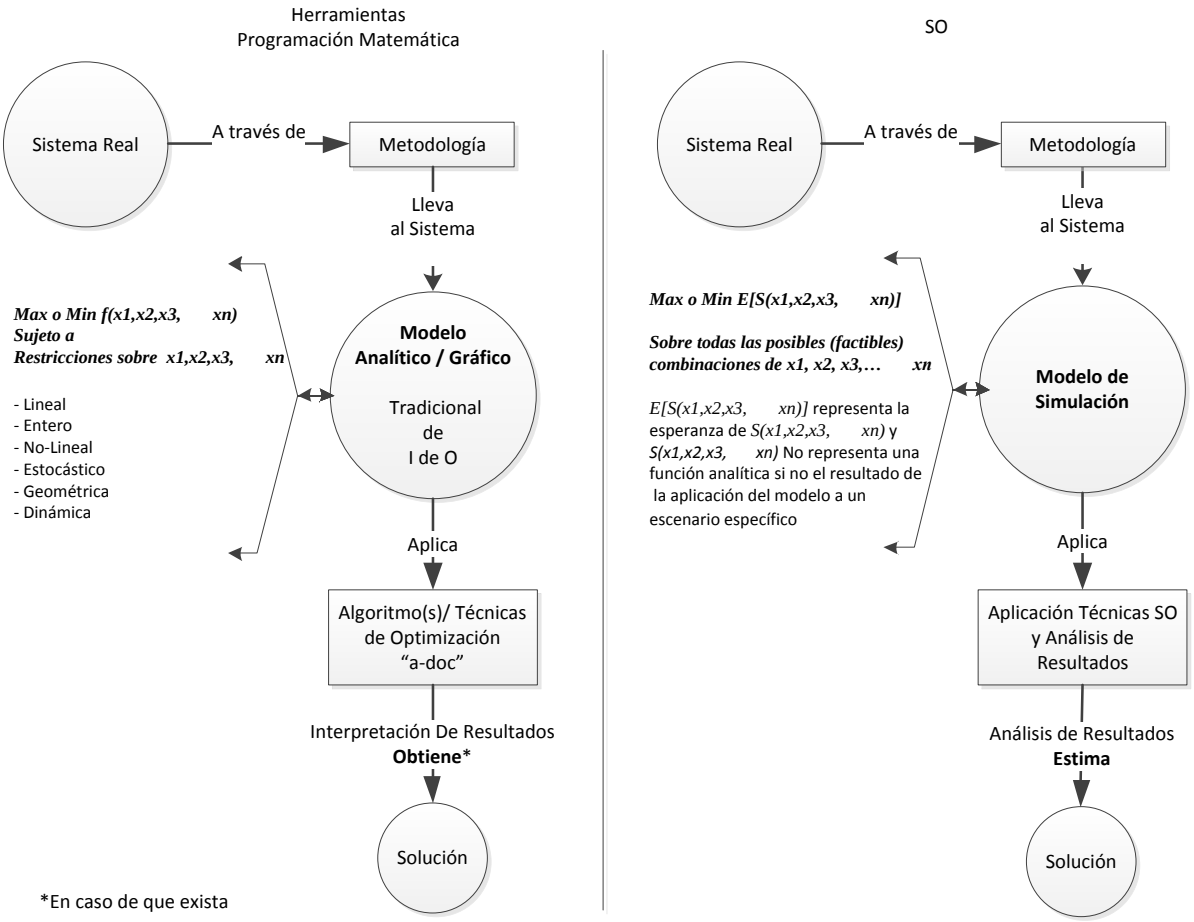


Figura 2. Programación Matemática vs. SO

Comparando la manera en que ambos procesos abordan la problemática de optimización de un sistema inmediatamente saltan a la vista las características que ya se han mencionado y que se discuten a continuación.

- El tipo de modelo utilizado para representar el sistema.

- La relación que guarda la técnica de optimización con el modelo utilizado para representar el sistema.

1.9.1 TIPO DE MODELO UTILIZADO PARA REPRESENTAR EL SISTEMA

La *SO* hace uso de un *modelo de simulación* para representar y medir el comportamiento del sistema; por otro lado las *herramientas de programación matemática tradicionales* utilizan un *modelo matemático-analítico* como medio de representación del sistema, este modelo analítico varía dependiendo de la técnica de programación matemática utilizada, y en la mayoría de las ocasiones, es con base en el mismo en el que se desarrollaron ya distintos algoritmos de optimización. Esta última observación nos remite de manera inmediata a la segunda característica distintiva de la SO:

1.9.2 RELACIÓN QUE GUARDA LA TÉCNICA DE OPTIMIZACIÓN CON EL MODELO UTILIZADO PARA REPRESENTAR EL SISTEMA

En la programación matemática, las técnicas de optimización surgen del análisis del modelo matemático utilizado para representar el sistema, es así como a partir del mismo se construyen algoritmos de optimización ad hoc para el ataque de la problemática en cuestión, esto es posible debido a que el sistema en todos los casos tiene una representación analítica que en ocasiones ha podido ser tratada. Por otro lado, en el caso de la SO, en la mayoría de las ocasiones, el modelo utilizado para representar al sistema no puede ser fácilmente llevado a una representación analítica, y aun si esto fuera posible, la expresión resultante

podría ser demasiado compleja como para construir un algoritmo de optimización alrededor de la misma.

Estas reflexiones nos dejan ver, que mientras en la programación matemática la técnica de optimización está íntimamente ligada al modelo utilizado para representar al sistema, en la SO la técnica de optimización no tiene una relación directa con el modelo en cuestión y el uso del mismo se limita únicamente a poder obtener una mediada de desempeño para un escenario específico.

A continuación se hace una breve reflexión de las ventajas que podría dar el uso de la SO vs. herramientas de programación matemática tradicionales al afrontar problemas de optimización de sistemas.

1.9.3 Posibles Ventajas de la SO en el ámbito de Optimización de Sistemas

Esta reflexión será realizada en torno a las dos características particulares de la técnica, *modelado utilizado para representar el sistema y relación del modelo con el algoritmo de optimización*.

1. Modelo utilizado para representar el sistema.

   La aplicación de cualquier técnica tradicionales de la I. de O. como lo es la Programación Lineal, No Lineal, Dinámica, etc., requieren poder representar, o lo que es lo mismo llevar el sistema a un modelo matemático-analítico específico o particular para la aplicación de la técnica. Esto en muchas ocasiones, sobre todo al tratar con sistemas que se abordan en la realidad, suele ser una tarea muy difícil y en muchas ocasiones puede tornarse en una tarea casi imposible. Los sistemas reales a menudo contienen condiciones de no linealidad, relaciones combinatorias e incertidumbre, que no suelen ser fácilmente modeladas de manera analítica a través de una función objetivo y un conjunto de restricciones matemático-analíticas; pero definitivamente en la

mayoría de los casos, son tratables mediante el uso de un modelo de simulación. Como ya se ha analizado, la simulación permite modelar y abordar problemas más complejos, además de que su aplicación en la actualidad se ha incrementado y dirigido hacia distintas áreas y sistemas. Estas características, las cuales hacen de la simulación una herramienta muy poderosa en el análisis de sistemas, son transmitidas íntegramente a la SO para brindarle grandes posibilidades como herramienta de optimización.

El hecho de que la SO utiliza un modelo de simulación para representar y medir el sistema, en lugar de un modelo matemático analítico como la mayoría de técnicas tradicionales, extiende los horizontes de los problemas de optimización de sistemas que pueden ser tratados con técnicas de optimización de la I. de O.

2. Relación del modelo con el Algoritmo de Optimización.

Utilizando el modelo del sistema únicamente como herramienta para evaluar las respuestas del mismo a configuraciones específicas, el diseño que soporta a la técnica de SO, brinda una mayor flexibilidad en la selección del algoritmo de optimización. Esto, en definitiva es una ventaja sobre las técnicas de programación matemática tradicionales, ya que en el caso de la SO, el algoritmo de optimización puede ser variado, con el objetivo de obtener mejores resultados o mejorar el tiempo de respuesta.

1.9.4 RETOS DE LA SO

Aunque el uso de la SO en problemas de optimización de sistemas puede plantear indiscutiblemente grandes ventajas, su utilización platea también varios retos, a continuación son mencionados los que a nuestro aparecer son los más importantes:

1. Al no existir funciones analíticas que representen tanto a la función objetivo como a las restricciones, la SO requiere ejecutar el modelo de simulación para obtener información acerca de las medidas de desempeño.

2. Si a lo anterior sumamos el hecho de que en la mayoría de las ocasiones los modelos de simulación utilizados para representar al sistema son de naturaleza estocástica, la SO requerirá “correr” varias veces el modelo de simulación para poder inferir sobre el comportamiento de un escenario específico.

3. En términos computacionales, los modelos de simulación son mucho más costosos de “correr” que la evaluación de una función analítica. Esto hace que la eficiencia del algoritmo de simulación sea crucial en la SO.

4. Por último, hay que mencionar que no es posible evaluar de manera precisa el desempeño de un diseño particular de un software SO hasta no ser puesto en operación. A pesar de que dicho desempeño podría ser estimado, esto muchas veces no ayudará mucho. Este punto es de gran relevancia cuando el diseño del software SO es propio, planteando grandes retos en las etapas de modelado del sistema y selección de la configuración del algoritmo de optimización.

Como observa el uso de SO no necesariamente implica la disminución del grado de dificultad en el ataque de la problemática, con respecto al que plantean otras técnicas de optimización, y cómo tal debe verse, más bien como una herramienta más dentro de la I. de O., de la que se puede echar mano cuando la problemática lo justifique.

1.10 Diseño base del Concepto Simulación y Optimización (SO)

Hasta este momento se ha analizado solo de manera conceptual a la SO, hablando de sus posibles ventajas desde el punto de vista teórico y analizado de manera formal la problemática que aborda esta área de la investigación científica, haciendo también referencia a que su funcionamiento se basa en la unión de la Optimización y Simulación. Sin embargo, no se ha analizado la manera en la que trabaja, es decir el diseño básico que soporta la técnica, cómo es que se unen la simulación y la optimización para atacar la problemática. Pues bien, el presente apartado pretende abordar este punto por demás importante, y es en este sentido que aun cuando la SO se ha dado a la tarea de unir ambas áreas de la investigación científica, la simulación y la optimización, bajo muy diversos enfoques en las últimas décadas, el diseño básico que soporta la técnica ha sido basado por mucho en los lineamientos que visualizara Dennis E. Smith en 1973 y que son presentados de manera gráfica en el siguiente esquema.

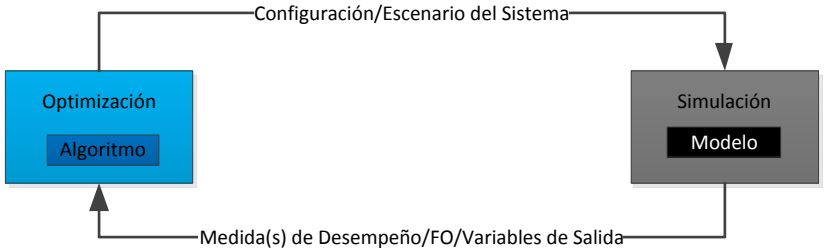


Figura 3. Concepto SO [5]

En la SO, *el modelo de simulación puede ser visto como una caja negra* utilizada para evaluar, mediante la experimentación, el comportamiento de las medidas de efectividad o desempeño del sistema en estudio bajo una configuración/escenario específico; mientras que *el algoritmo de optimización es el responsable de la*

búsqueda de optimalidad, en otras palabras, el algoritmo de optimización propone las configuraciones/escenarios por los que se conducirá la búsqueda del óptimo con base en la retroalimentación brindada por el modelo de simulación comúnmente mediante el análisis de el valor de la medida(s) de efectividad, o Función Objetivo (FO), del sistema que se pretenden optimizar, estableciendo para tal efecto los nuevos escenarios de sistema que serán corridos por el modelo de simulación en la búsqueda del óptimo.

1.11 Implementación en la Práctica del Concepto Simulación y Optimización (SO)

Después de analizar el diseño básico que soporta a la técnica, en el presente apartado se explica cómo dicho concepto ha sido llevado a la práctica y para hacerlo es muy importante primero entender que, sin lugar a dudas, al igual que la Simulación, la SO ha tenido un desarrollo paralelo al del área de computación. Por un lado se tiene el impulso que la computación ha brindado a la Simulación pos si sola, y por otro, el que ha brindado para el crecimiento de distintas áreas de la investigación que tienen que ver con la optimización, avances en software y hardware han permitido tratar problemas cada vez más grandes y complicados.

Es natural que debido al diseño base y naturaleza del concepto SO, su implementación en la práctica no podría ser de otra manera más que a través de software, mismo que de una manera general y a grandes rasgos trabaja bajo el diseño original del concepto.

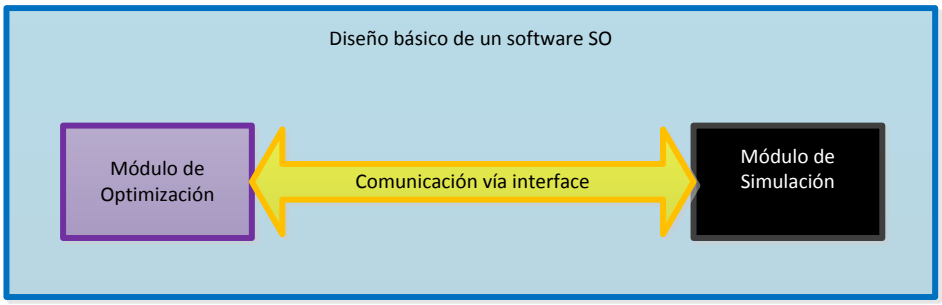


Figura 4. Diseño Básico de un Software SO

El diseño básico de un software SO está integrado por tres componentes clave: d*os módulos o subsistemas uno de simulación y otro de optimización,* además de *una interfaz de comunicación* que es la que hace posible el concepto SO como tal, encargándose de comunicar los dos módulos mencionados, haciendo que ambos trabajen juntos de manera automatizada en torno al objetivo de *búsqueda de optimalidad*. Este esquema es el que ha sido utilizado desde los primeros trabajos registrados en SO que realizaran Dennis Pegden y Michael Gately, hasta los más recientes realizados por distintos investigadores. Este esquema de implementación, y el diseño básico que soporta la técnica, permite que hoy en día existan varios enfoques en esta rama de la I. de O.

1.12 Enfoques Simulación y Optimización (SO)

Los esfuerzos de investigación en torno a la SO se han concentrado mayormente en el área de optimización, esto se debe principalmente a que, por un lado, la simulación es un área ya muy estudiada con un desarrollo propio muy importante, y a que, por otro lado, la intervención de esta área dentro de la SO se limita a medir el desempeño del sistema en estudio con respecto a una configuración específica. Es así como diversos investigadores se han dado a la tarea de incorporar diversos avances desarrollados en el campo de la optimización,

generándose así diversos enfoques de cómo el concepto ha sido implementado hoy por hoy; y entre los cuales se pueden destacar los siguientes:

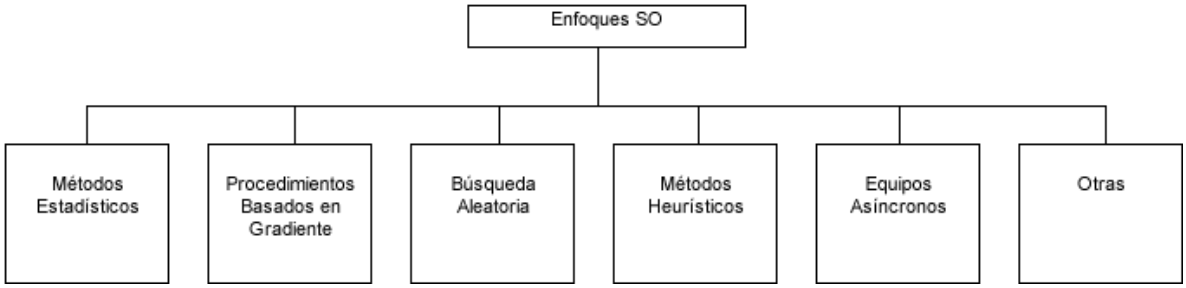


Figura 5. Enfoques SO (con información de [6] - [7])

- Métodos Estadísticos.

  Estos métodos basan su funcionamiento en herramientas estadísticas, fueron de los primeros utilizados, y entre los que más importantes se tienen:

  - ✓ Selección y clasificación (Ranking and Selection)
  - ✓ Metodología de superficie de respuesta (Response Surface Methodology)
  - ✓ Comparación múltiple (Multiple Comparison)
  - ✓ Muestreo por importancia (Importance Sampling)

- Procedimientos Basados en Gradiente.

  Los procedimientos basados en gradiente tratan de imitar a métodos de optimización determinística. Los métodos que se encuentran en esta categoría estiman la respuesta de la función gradiente (∇f), para entonces, determinar la dirección de los movimientos a realizar en la búsqueda del óptimo. Algunos métodos que se encuentran en esta categoría son:

- ✓ Aproximación estocástica (Stochastic Approximation)
- ✓ Diferencias finitas (Finite Differences)
- ✓ Perturbaciones simultaneas (Simultaneous Perturbations)
- ✓ Análisis de perturbación (Perturbation Analysis)
- ✓ Fuerza bruta (Brute-Force)
- ✓ Razón de verosimilitud (Likelihood Ratio)
- ✓ Experimentos de dominio de frecuencia (Frequency Domain Experiments)

- ➢ Búsqueda Aleatoria (Random Search).

Los algoritmos de búsqueda aleatoria, de forma similar a los procedimientos basados en gradiente, basan su funcionamiento en movimientos iterativos en la búsqueda del óptimo, sin embargo, a diferencia de los procedimientos basados en gradiente estos no requieren la estimación del gradiente para determinar la dirección en la que se hará el movimiento.

- ➢ Métodos Heurísticos.

Un métodos heurístico puede ser definido como: "*un procedimiento simple, a menudo basado en el sentido común, que se supone ofrecerá una buena solución (aunque no necesariamente la óptima) a problemas difíciles*" [8]. Un punto importante por el que estas técnicas son ampliamente usadas en la SO es porque la única condición para su implementación es que los valores de la función objetivo puedan ser obtenidos de alguna forma, algunos de los métodos más utilizados de este tipo son:

- ✓ Algoritmos genéticos (Genetic Algorithms).
- ✓ Estrategias evolutivas (Evolutionary Strategies).
- ✓ Recocido simulado (Simulated Annealing).
- ✓ Búsqueda tabú (Tabu Search).
- ✓ Búsqueda dispersa (Scatter Search)

- Equipos Asíncronos (A-Teams).

  Un Equipo Asíncrono es un proceso que involucra la combinación de varias estrategias de solución al enfrentar una problemática, el objetivo principal en el proceso es que estas estrategias actúen sinérgicamente. Por ejemplo bajo este enfoque se pueden tener la combinación de técnicas de heurísticas y hasta de alguna otra basada en gradiente.

- Otros.

  - ✓ Sample Path Optimization
  - ✓ Métodos basados en el modelo (model-based methods)
    - ✓ Método de colonia de hormigas(Ant Colony Optimization / Swarm Intelligence)
    - ✓ Algoritmos de estimación de distribución (Estimation of Distribution Algorithms)
    - ✓ Método de entropía cruzada (Cross-Entropy Method)
    - ✓ Modelo de búsqueda referenciada adaptativa (Model Reference Adaptive Search).

De los enfoques anteriores debemos mencionar que las heurísticas es una de las áreas en las que más se ha concentrado el trabajo, esto se observa incluso a escala comercial (Tabla 1. Software SO Comercial y Técnicas vs Algoritmos de Optimización), las posibles razones:

1. Los métodos heurísticos tienen como única condicionante en su implementación que el valor de la función objetivo pueda ser obtenido de alguna manera, no importa cuál. Esto hace posible su aplicación en casi cualquier problemática de optimización, y en particular en la que es enfrentada por la SO.

2. Estas técnicas han comprobado una buena efectividad al ser confrontados a distintos problemas de optimización de sistemas. Hay que resaltar que en los últimos años las innovaciones en el campo de las heurísticas y la computación han alcanzado un punto tal, que muchos de los problemas que no se habían atacado exitosamente en el pasado con el uso de estas técnicas, hoy pueden ya ser resueltos.

3. La mayoría de las heurísticas son relativamente “fáciles” de implementar con el uso de casi cualquier lenguaje de programación.

Optimizador	Software Simulación	Técnica de Optimización
Evolutionary Optimizer ®	Extend® Imagine That, Inc. *www.extendsim.com*	Estrategias Evolutivas, Algoritmos Genéticos
SimRunner® ProModel Corp. *www.promodel.com* OptQuest® OptTek Systems, Inc. *www.opttek.com*	Promodel® ProModel Corp. *www.promodel.com*	(SimRunner®) Estrategias Evolutivas, Algoritmos Genéticos (OptQuest®B) Búsqueda Dispersa, Búsqueda Tabú, Redes Neuronales
OptQuest® OptTek Systems, Inc. *www.opttek.com*	Arena® Rockwell Automation, Inc. *www.arenasimulation.com*	Búsqueda Dispersa, Búsqueda Tabú, Redes Neuronales
	Simul8® SIMUL8 Corp. *www.simul8.com*	
Optimizer®	WITNESS® Lanner Group, Inc. *www.lanner.com*	Recocido Simulado, Búsqueda Tabú
RISKOptimizer	@Risk® Palisade Corp. *www.palisade.com*	Algoritmos Genéticos

Tabla 1. Software SO Comercial y Técnicas vs Algoritmos de Optimización[3]

[3] Fuentes Varias [29], [30], [7]

II. Presentación Caso de Estudio / Problemática

El presente capítulo es dedicado a la presentación del caso de estudio, teórico - práctico, que apoyará la propuesta de diseño de software tendiente a aplicar el concepto SO a sistemas que pueden ser modelados con el uso de líneas de espera. El caso de estudio permitirá definir algunos aspectos necesarios para la aplicación inmediata de la propuesta de diseño desarrollada en el capítulo III.

2.1 Planteamiento del Problema

La SO desde sus inicios y hasta la fecha, al menos a nivel comercial, ha sido principalmente enfocada al ataque de problemas de optimización en sistemas que pueden ser representados con modelos de simulación discreta preponderantemente relacionados con líneas de espera. Así lo constatan los primeros esfuerzos de Pegden y Michael Gately, quienes unieron sus esfuerzos

para desarrollar un software de optimización en torno al sistema GASPIV. Más allá de que esto se deba a la casualidad, es más bien consecuencia del inmenso potencial de aplicación que este tipo de modelos brinda para abordar problemáticas de sistemas reales; los ámbitos de aplicación pueden ir desde el económico hasta el social, pasando por el productivo. Este hecho fue el que nos motivó a desarrollar el trabajo de la presente tesis alrededor de este tipo de sistemas, y de manera particular a establecer el caso de estudio en torno al desafío de la implantación de un sistema de manufactura que puede ser modelado/simulado con el uso de líneas de espera.

El sistema de manufactura en cuestión es una línea de producción con m estaciones de trabajo (WS), cada estación de trabajo podrá contar con un número fijo de servidores que llevan a cabo una función específica relativa a la estación en cuestión, en otras palabras, todos los servidores de una estación de trabajo llevan a cabo la misma función.

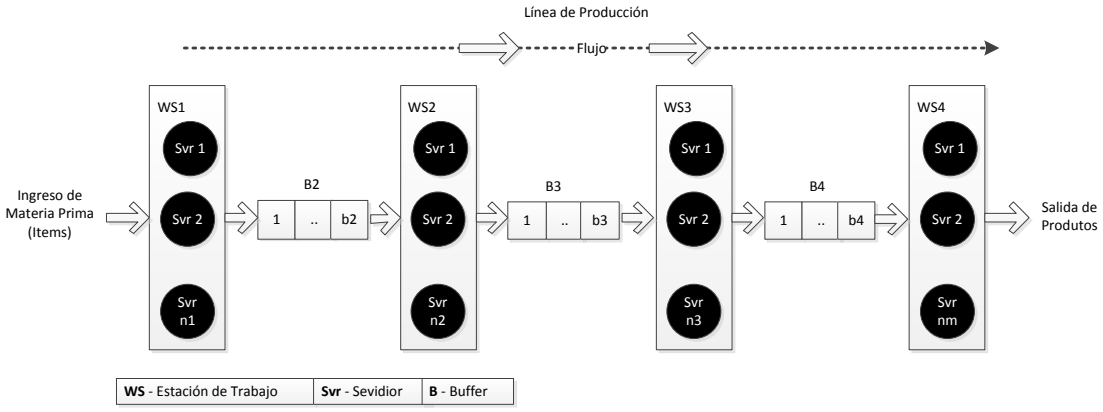


Figura 6. Caso de Estudio General

El funcionamiento de este sistema no es muy diferente a lo que se acostumbra: la línea transforma materia prima (items) en productos, uno a la vez a través de m procesos, mismos que se llevan a cabo en los servidores de las m estaciones de

trabajo (WS), sin embargo, nuestro proceso productivo tiene una peculiaridad, y es que entre las *m* estaciones de trabajo (WS) se encuentran colocados *m-1 buffers* (líneas de espera) con una capacidad finita específica $bi > 0$ $(i=2,3,.....m)$, donde cada ítem deberá esperar su turno antes de ser procesado por un servidor de la estación de trabajo en cuestión. Si un servidor dentro de una estación de trabajo termina el servicio (tarea) en un *ítem*, éste inmediatamente verifica si el *buffer* al que será enviado para esperar su próximo servicio dentro del proceso tiene todavía capacidad para almacenarlo, pudiendo en este punto ocurrir dos cosas: uno que el *buffer* específico si tenga espacio disponible para recibirlo, en cuyo caso el proceso sigue su paso natural y el *ítem* es colocado en el buffer donde esperará su próximo servicio como si estuviera formado en una cola; y dos que el *buffer* específico no tenga lugar disponible para recibir el *ítem,* en cuyo caso el servidor tendrá que retenerlo hasta que haya un espacio disponible en el buffer.

2.2 PROPÓSITO

Con solo comprender el funcionamiento del sistema, se hace evidente que uno de los principales retos que se tendrán que afrontar en el proyecto es hacer que el sistema como tal sea lo más eficiente posible, lo que únicamente se logrará si se establece una buena configuración inicial en función de los recursos con los que se podría contar, mismos que es importante mencionar siempre son limitados, como en todo proyecto, tanto por cuestiones tanto técnicas como económicas. Sin embargo, se asevera que este reto es solo uno de los principales por una razón, y es que éste contempla únicamente el aspecto funcional del sistema, mismo que, cuando se habla de negocios, no necesariamente es el más importante, ¿porqué?, pues bien, se puede tener una capacidad de producción muy buena, como resultado de cierta configuración del sistema, sin embargo la relación costo beneficio pudiera no serlo tanto, sobre todo después de considerar los gastos de operación asociados al funcionamiento del sistema bajo la esta configuración. Así es como en el presente trabajo nuestro principal objetivo será ***"maximizar la***

utilidad monetaria que brindará el sistema", misma que hay que mencionar depende de la selección de la configuración inicial para la puesta en marcha del sistema, y que de manera general podría ser calculada, para un periodo de tiempo determinado, mediante el uso de la siguiente expresión:

Utilidad Monetaria =
(Precio de venta del producto × Items convertidos en producto en el periodo de tiempo)
- Costo de operación de las estaciones de trabajo en el periodo de tiempo
- Costo de operación de los buffers en el periodo de tiempo
- Costo de la materia prima utilizada en el periodo de tiempo

2.3 PARÁMETROS INVOLUCRADOS

Para estar en posibilidad de dar una representación a cada posible solución al problema, es necesario primero identificar los parámetros que pueden afectar el valor de la medida de efectividad o desempeño que se pretende optimizar: *"utilidad monetaria que brindará el sistema"*, esto es, las variables de entrada del modelo de simulación. En el sistema que se está tratando, de forma general, se podrían identificar las siguientes variables candidatas:

Variable	Posibles Valores
Distribución de tiempo entre arribo de los ítems (materia prima) al sistema.	*Determinístico, Exponencial, ETC.*
Parámetro(s) de distribución de tiempo entre arribo de los items (materia prima) al sistema.	$\mu_1, \mu_2, \ldots, \mu_p$ donde μ_k con $k=1,\ldots,p$ son valores numéricos que dependen de la variable distribución de tiempo entre arribo.

Variable	Posibles Valores
Distribución de tiempo de servicio para los servidores en la estación de trabajo i (WS_i) con $i = 1, \ldots, m$.	*Determinístico, Exponencial, ETC.*
Parámetro(s) de distribución de tiempo de servicio para los servidores en la estación de trabajo i (WS_i) con $i = 1, \ldots, m$.	$\mu_{i1}, \mu_{i2}, \ldots, \mu_{ip}$ donde μ_{ik} con $k=1,\ldots,p$ son valores numéricos que dependen de la variable distribución de tiempo de servicio para los servidores en la estación de trabajo i (WS_i) con $i = 1, \ldots, m$.
Número de servidores en la estación de trabajo i (WS_i) con $i = 1, \ldots, m$.	$1,2,3, \ldots$ (Enteros Positivos)
Capacidad del buffer i (B_i) con $i = 2, \ldots, m$	$1,2,3, \ldots$ (Enteros Positivos)

Tabla 2. Variables candidatas a ser variables de entrada

Como se puede observar existen variables tanto numéricas como cualitativas de las que depende el comportamiento del sistema, sin embargo para efectos del presente trabajo se harán los siguientes supuestos:

1. La fuente de abastecimiento del sistema es infinita.

2. El comportamiento de los tiempos de servicio es idéntico para todos los servidores de una misma estación de trabajo, en otras palabras la variable aleatoria tiempo de servicio tendrá misma función de distribución y los mismos parámetros en todos los servidores pertenecientes a una estación de trabajo particular.

Estos supuestos no son “descabellados”, más todavía si se toma en cuenta que:

1. Lo que se pretende encontrar es la mejor configuración inicial para la implantación del sistema.

2. Todos los servidores en una misma estación de trabajo llevan a cabo la misma función.

Además la aplicación de los mismos elimina las variables relacionadas a la distribución de tiempo entre arribos y tiempos de servicio, por lo que la lista de variables de entrada de nuestro modelo se reducirá a:

1. Número de servidores en la estación de trabajo i (WS_i):

$$\boldsymbol{svrs_i} \qquad (i = 1,2,3, \ldots, m)$$

2. Capacidad del buffer i (B_i):

$$\boldsymbol{b_i} \qquad (i = 2,3,4, \ldots, m)$$

Analizando ahora los posibles valores que estas variables pueden tomar, se sabe por su definición que estos deben ser enteros positivos, pero no cualesquiera, ya que tanto por cuestiones operativas, por ejemplo las relacionadas con el espacio disponible, como por cuestiones de recursos, financieros por ejemplo, estas variables siempre estarán acotadas superiormente, de manera que:

1. El número de servidores en la estación de trabajo i (WS_i) / $\boldsymbol{svrs_i}$ deberá cumplir que:

$$\boldsymbol{0 < svrs_i \leq Usvrs_i}$$

con:

$\boldsymbol{svrs_i}$, $\boldsymbol{Usvrs_i}$ *enteros positivos* $(i = 1,2,3, \ldots, m)$

2. La capacidad del buffer i (B_i) / b_i deberá cumplir que:

$$0 < b_i \leq Ub_i$$

Con:

b_i, Ub_i *enteros positivos (con* $i = 2,3,4, \ldots, m$*)*

2.4 Espacio de Resultados

Una vez que se establecieron las variables de entrada del modelo, y los posibles valores que pueden tomar, se podrá representar cualquier solución factible (escenario o posible configuración del sistema), mediante el uso del vector que se muestra a continuación:

$$(svrs_1 , svrs_2 , svrs_3 , \ \ldots\ , svrs_m , b_2 , b_3 , b_4 , \ \ldots\ , b_m)$$

Donde:

$svrs_i$ representa el número de servidores colocados en la estación de trabajo i, y cumple:

$0 < svrs_i \leq Usvrs_i$

con $svrs_i$, $Usvrs_i$ *enteros positivos*

$i = 1,2,3, \ldots, m$

Donde:

b_i representa la capacidad del buffer i, y cumple:

$0 < b_i \leq Ub_i$

con b_i, Ub_i *enteros positivos*

$i = 2,3,4, \ldots, m$.

El espacio de resultados del problema estará conformado por todos los posibles vectores de esta forma.

2.5 Cardinalidad del Espacio de Resultados

Una vez definido el espacio de resultados es hasta cierto punto sencillo calcular el número de escenarios, o posibles configuraciones del sistema, que habrán de ser analizadas en la búsqueda de un óptimo para la medida de efectividad o desempeño del sistema.

Se busca encontrar el número de vectores diferentes de la forma:

$$(svrs_1 , svrs_2 , svrs_3 , \ \dots \ , svrs_m , b_2 , b_3 , b_4 , \ \dots , \ b_m)$$

Donde:
$0 < svrs_i \leq Usvrs_i$
con $svrs_i$, $Usvrs_i$ enteros positivos
$i = 1,2,3, \dots, m$

Donde:
$0 < b_i \leq Ub_i$
con b_i, Ub_i enteros positivos
$i = 2,3,4, \dots, m.$

Y este número puede ser obtenido de la siguiente manera:

Se tienen primero las m variables $\boldsymbol{svrs_i}$ que representan el número de servidores en cada una de las estaciones de trabajo, este número es un entero positivo que va desde *1* hasta $\boldsymbol{Usvrs_i}$, por lo que la primera entrada del vector puede tomar $\boldsymbol{Usvrs_1}$ valores diferentes; la segunda, $\boldsymbol{Usvrs_2}$ valores diferentes y así sucesivamente hasta la *m-ésima* entrada que puede tomar $\boldsymbol{Usvrs_m}$ valores diferentes.

Haciendo un análisis similar se tienen *m-1* variables $\boldsymbol{b_i}$ que representan la capacidad de cada uno de los buffers del sistema, este número es un entero positivo que va desde *1* hasta $\boldsymbol{Ub_i}$, por lo que la *m+1-ésima* entrada del vector puede tomar $\boldsymbol{Ub_2}$ valores diferentes, la *m+2-ésima* $\boldsymbol{Ub_3}$ valores diferentes y así sucesivamente hasta la *m+m-1-ésima* entrada que puede tomar $\boldsymbol{Ub_m}$ valores diferentes.

Por lo que el número total de permutaciones que se pueden tener con los valores que pueden tomar las $2m-1$ variables de nuestro vector es calculado de la siguiente manera:

$$Usvrs_1 \times Usvrs_2 \times Usvrs_3 \times \ldots \times Usvrs_m \times Ub_2 \times Ub_3 \times \ldots \times Ub_m$$

Esta es la cardinalidad de nuestro espacio de resultados, pero, ¿qué tan grande es este número?, para darnos una idea se puede hacer el siguiente análisis:

$$Usvrs_1, Usvrs_2, Usvrs_3, \ldots Usvrs_m, Ub_2, Ub_3, \ldots, Ub_m$$

son las cotas superiores de los valores que pueden tomar cada una de las variables de entrada del modelo de simulación, mismas que ya se ha mencionado solo pueden tomar valores enteros positivos, de ahí que para que estas puedan ser consideradas como variables, cada una de las cotas superiores debería ser mayor o igual 2, tomando en cuenta esto se puede generar la siguiente cota inferior para la cardinalidad del espacio de resultados:

$$Usvrs_1 \times Usvrs_2 \times Usvrs_3 \times \ldots \times Usvrs_m \times Ub_2 \times Ub_3 \times \ldots \times Ub_m$$

$$\geq$$

$$2 \times 2 \times 2 \times \ldots \times 2 \times 2 \times 2 \times \ldots \times 2$$

$$\Rightarrow$$

$$Usvrs_1 \times Usvrs_2 \times Usvrs_3 \times \ldots \times Usvrs_m \times Ub_2 \times Ub_3 \times \ldots \times Ub_m \geq 2^{2m-1}$$

De donde si m crece, el número total del escenarios/configuraciones que se tendrán que analizar para resolver la problemática, crece de forma exponencial en un orden superior a 2^{2m-1}, lo que puede llegar a ser un número muy grande, aun cuando las cotas superiores de los valores que pueden tomar cada una de las variables de entrada del modelo de simulación sean números aparentemente pequeños.

2.6 PROBLEMÁTICA

Es evidente que la problemática que se enfrenta se encuentra circunscrita en el ámbito de la SO, se desea encontrar una configuración/ escenario del sistema, o lo que es lo mismo, los valores que determinadas variables deberán tomar dentro del mismo, que nos permita optimizar un medida de interés, en este caso la *"utilidad monetaria que brindará el sistema"*, o de una manera más formal:

$$Max\ E\left[\left(\ PUV(ICP)\right) -\ CMP - \sum_{i=1}^{m}\sum_{k=1}^{svrs_i} t\ (Csvr_k s_i)\ - \sum_{i=2}^{m}\sum_{k=1}^{b_i} t\ (Ce_k b_i)\right]$$

Sobre todas las posibles combinaciones de $svrs_1$, $svrs_2$, $svrs_3$, … , $svrs_m$, b_2 , b_3 , b_4 , … , b_m *, o lo que es lo mismo sobre todos los vectores de la forma:*

$$(\ svrs_1\ ,\ svrs_2\ ,\ svrs_3\ ,\ \ldots\ ,\ svrs_m\ ,\ b_2\ ,\ b_3\ ,\ b_4\ ,\ \ldots\ ,\ b_m\ )$$

Donde:

$PUV = Precio\ unitario\ de\ Venta\ por\ Producto$
$Csvr_k s_i = Costo\ operación\ servidor\ k\ estación\ de\ servicio\ i\ por\ unidad\ de\ tiempo$
$Ce_k b_i = Costo\ operación\ espacio\ k\ buffer\ i\ por\ unidad\ de\ tiempo$
$t = periodo\ de\ tiempo\ en\ el\ que\ será\ calculda\ la\ utilidad\ monetaria\ del\ sistema$
$CMP = Costo\ Materia\ Prima\ utilzada\ en\ el\ periodo\ de\ tiempo\ t$
$ICP = Número\ de\ Items\ Convertidos\ en\ producto\ en\ el\ periode\ de\ tiempo\ t$
y

$svrs_i$ representa el número de servidores colocados en la estación de trabajo *i,* y cumple:

$0 < svrs_i \leq Usvrs_i$
con $svrs_i$, $Usvrs_i$ *enteros positivos*
$i = 1,2,3, \ldots, m$

b_i representa la capacidad del buffer *i,* y cumple:

$0 < b_i \leq Ub_i$
con b_i, Ub_i *enteros positivos*
$i = 2,3,4, \ldots, m.$

2.7 Alternativas de Solución

Aun cuando la problemática se ciñe perfectamente a lo que es la SO, es también teóricamente factible que ésta pueda ser atacada mediante el uso únicamente de simulación, y es de esta manera que a continuación, con el uso de la siguiente tabla, reflexionamos, a grandes rasgos, sobre las tareas que plantea abordar la problemática con el uso de cada una de éstas técnicas.

Simulación	SO
Generación del Modelo de Simulación. • Análisis y diseño. • Construcción, vía software comercial o desarrollo propio (módulo de simulación).	**Generación del Modelo de Simulación.** • Análisis y diseño. • Construcción, vía software comercial o desarrollo propio (módulo de simulación).
	Acoplamiento del Paquete Optimizador*. • Selección de la técnica de optimización. • Implementación del algoritmo vía software (módulo de optimización). • Desarrollo de la interfaz que comunicará al módulo de simulación con el módulo de optimización.
Diseño de Experimentos. • **Establecimiento de condiciones iniciales de experimentación.** • **Selección de escenarios que se ejecutarán en el modelo.**	**Diseño de Experimentos.** • Establecimiento de condiciones iniciales de experimentación.

Simulación	SO
Experimentación. • Evaluación los escenarios necesarios, mediante la ejecución del modelo de simulación, que permitan asegurar que el resultado obtenido tiene una "buena calidad".	**Experimentación.** • Ejecución del software SO bajo las condiciones iniciales de experimentación.
Análisis de Resultados. • Análisis de resultados y toma de decisiones.	**Análisis de Resultados.** • Análisis de resultados y toma de decisiones.

*Si el modelo de simulación es construido con un software comercial que tiene integrado el concepto SO esta tarea no será necesaria.

Tabla 3. Alternativas de Solución a la Problemática

Tanto la simulación como la SO plantean retos, algunos de ellos compartidos, sin embargo si nos centramos en el objetivo principal del problema, que es la búsqueda de optimalidad, y cómo ésta es conducida por ambas técnicas (Diseño de Experimentos y Experimentación), la utilización de simulación plantea un reto que puede llegar a ser muy grande debido a que *el encontrar, si no la mejor, una muy buena solución al problema*, requerirá del análisis de gran parte del espacio de resultados, si no es que de su totalidad, y es que mientras en la SO la búsqueda del óptimo es conducida por un algoritmo de optimización, en la simulación no existe una conducción hacia éste como tal.

Luego entonces, es válido preguntarse, ¿cuál sería la técnica más conveniente para atacar problemas de este tipo?, pues bien, al menos desde el punto de vista teórico, parecería ser que la respuesta a esta pregunta está siempre íntimamente ligada al número de escenarios del sistema que tendrían que ser analizados para encontrar la mejor configuración del mismo con respecto a cierta medida de interés, y aquí es importante subrayar que no decimos todas porque muchas pueden ser descartables con la visión experimentada de personas con conocimiento y experiencia en el sistema. Si este número no es muy grande,

puede ser preferible utilizar solo un modelo de simulación, es decir simulación por sí sola, explorando esas posibles configuraciones para encontrar la mejor. Si por el contrario, el número de configuraciones que se tendrían que explorar es muy grande, el uso de la SO, y los retos/esfuerzos adicionales que como consecuencia se tendrán, estarán de alguna manera justificados; el conducir la búsqueda del óptimo por medio de un algoritmo de optimización debería poder reducir el número de escenarios que tendrían que ser analizados para encontrar un buen resultado, esto claro al utilizar algún algoritmo de optimización cuyo diseño tiene ya, de alguna manera, eficacia probada.

En el caso de estudio presentado, enfocándonos únicamente en el número de escenarios sobre los que se tendría que buscar la solución al problema, y tomando en cuenta el análisis realizado sobre la cardinalidad del espacio de resultados, queda justificado de alguna manera el uso de SO.

III. Análisis y Propuesta de Diseño SO Orientado a Objetos

El presente capítulo aborda y documenta el proceso de análisis, y por supuesto, la propuesta de diseño de un software SO orientado a objetos (OO, por sus siglas en inglés) capaz de atacar la problemática general presentada en el capítulo anterior, y tendiente a tratar sistemas que pueden ser modelados con el uso de líneas de espera; todo esto con el uso de Lenguaje Unificado de Modelado (UML [9], por sus siglas en inglés).

En la actualidad el UML es una de las herramientas más utilizadas para el modelado de software y está respaldado por el consorcio internacional sin fines de lucro Object Management Group (OMG [10], por sus siglas en inglés). El objetivo del UML es el de proveer herramientas estandarizadas que permitan soportar las diferentes etapas en la concepción de un software (Análisis y Diseño); es así como el UML fija un estándar gráfico para visualizar, especificar y documentar un sistema.

Sin embrago, hay que observar que UML no es una metodología (método o proceso) para el desarrollo de sistemas, y en este sentido se hace necesario el uso de alguna que guíe las etapas de análisis y diseño, cruciales en el desarrollo de cualquier software.

Existen diferentes metodologías para llevar a cabo las etapas, o procesos, de análisis y de diseño de un software del tipo que se pretende [11], en nuestro caso se decidió incluir aspectos tanto del método desarrollado por Grady Booch en la década de 1990 [12], Booch OOAD[4], como del desarrollado por James Rumbaugh también en esa misma dedada [13], OMT[5], de manera a que, a grandes rasgos, estos dos procesos, análisis y diseño, serán soportados y documentados con la utilización de los modelos que se muestran a continuación [Figura 7].

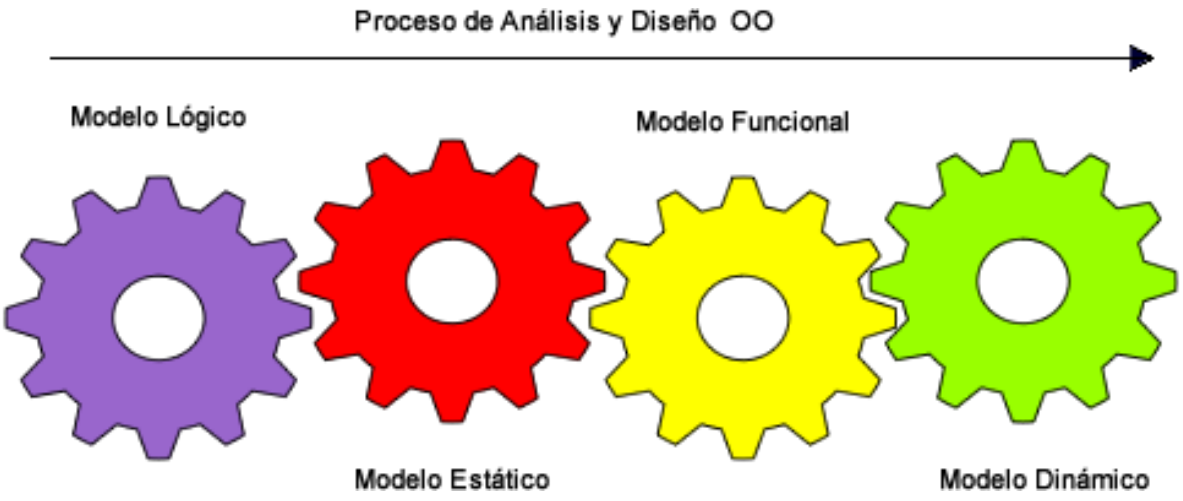


Figura 7 Metodología usada el diseño del software

[4] Object-oriented Analysis and Design, (OOAD, por sus siglas en inglés), también conocido como Object-Oriented Design (OOD, por sus siglas en inglés)
[5] Object Modeling Technique (OMT, por sus siglas en inglés)

A finales de los 90s Grady Booch, James Rumbaugh e Ivar Jacobson, con el objetivo de acelerar la adopción tecnología de objetos, trabajan juntos y comienzan sentar las bases de lo que ahora se conoce como UML

Modelo Lógico.

En el serán detalladas las necesidades funcionales del software. Diagramas, casos de uso y modelos de dominio permitirán identificar componentes tipo del sistema y sus relaciones.

Modelo Estático

Establece el detalle de diseño de las clases, o componentes tipo, de manera que éste permita cumplir con los requerimientos plasmados en el modelo lógico.

Modelo Funcional.

Hay que decir primero que no en el sentido de lo establecido por el método OMT, en el que este modelo sólo pretende describir las trasformaciones que sufre la información que el sistema maneja, si no más bien en el sentido de cómo trabajará el modelo estático, o el sistema, para lograr su objetivo.

Modelo Dinámico.

Muestra el orden y la manera en que los objetos, instanciados desde las clases definidas en el modelo estático, interactúan en tiempo de ejecución.

Hay que mencionar que ambas metodologías brindan un estándar gráfico para el desarrollo de los diagramas y gráficos que soportan estos modelos, sin embargo, y como ya se mencionó, estos serán desarrollados con estándares UML que son mucho más actuales.

3.1 ANÁLISIS - CASOS DE USO, SUJETO Y ACTORES

En términos de Ingeniería de Software[6] los casos de uso son los medios utilizados en la etapa de análisis, para especificar los usos requeridos de un sistema.

Un caso de uso es una descripción de pasos o acciones entre un usuario (actor) y un sistema (sujeto), la cual guía al usuario a algo útil. La idea básica detrás del modelado de casos de uso es llegar al corazón de lo que un sistema debe hacer [14], o lo que es lo mismo lo que se desea el sistema sea capaz de hacer.

En UML el modelado de casos de usos se realiza mediante una herramienta conocida como diagrama de caso de uso y los conceptos clave asociados a esta herramienta son:

Sujeto. El sujeto es el sistema al cual el caso de uso aplica.

Actores. Un *actor* específica el **rol** jugado por una entidad (Usuario/Sistema/Hardware) que interactúa con el sujeto.[7]

El concepto SO tiene implícitos a dos actores fundamentales: la simulación y la optimización, estos actores interactúan entre sí inmersos en el concepto [Figura 8], con el objetivo de brindar una propuesta de solución a la problemática de optimización enfrentada. La optimización propone soluciones y conduce la búsqueda hacia el óptimo con base al valor de la mediad de efectividad de cada una de las soluciones propuestas; la simulación es la encargada de obtener la media de efectividad de los escenarios propuestos a través de modelar y ejecutar los escenarios propuestos.

[6] Disciplina de la ingeniería que se preocupa por todos los aspectos que están involucrados en la producción de software [31]
[7] Un actor modela un tipo de rol jugado por una entidad que interactúa con el sujeto, pero el cual es externo al sujeto [9]

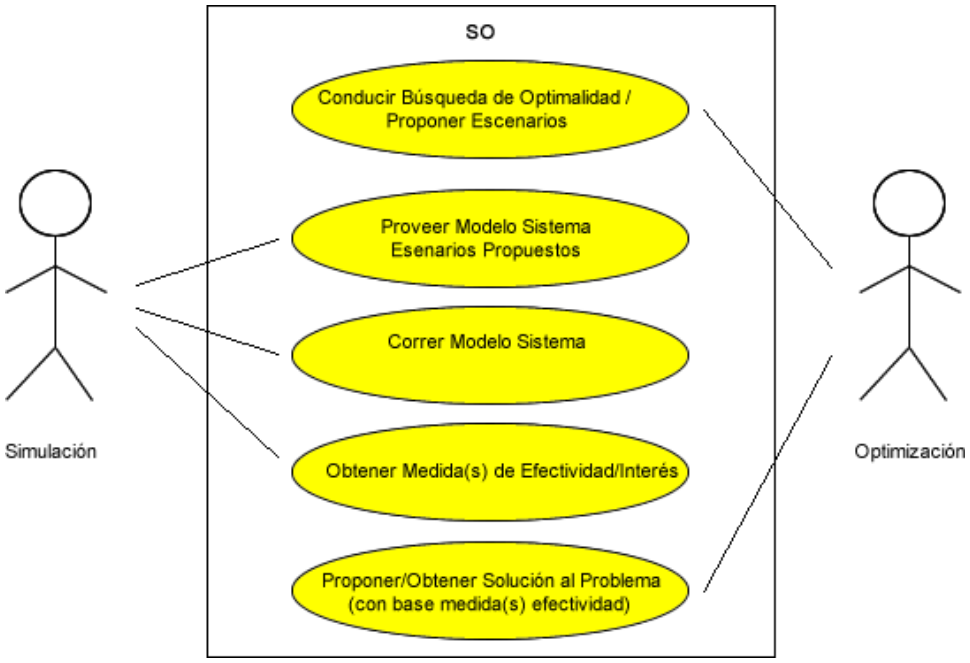


Figura 8. Diagrama de Caso de Uso Simulación y Optimización

3.2 Modelo Lógico – Modelo de Dominio SO

Como ya se ha mencionado en el presente trabajo un modelo es una representación abstracta de un sistema, en el contexto particular de la Ingeniería de Software un modelo de dominio es un artefacto de la disciplina de análisis que permite capturar y expresar el entendimiento ganado en un área bajo análisis como paso previo al diseño de un sistema [15]; un modelo de dominio modela el mundo real no objetos de software.

Como paso previo al diseño del software enseguida se presenta un modelo de dominio del concepto:

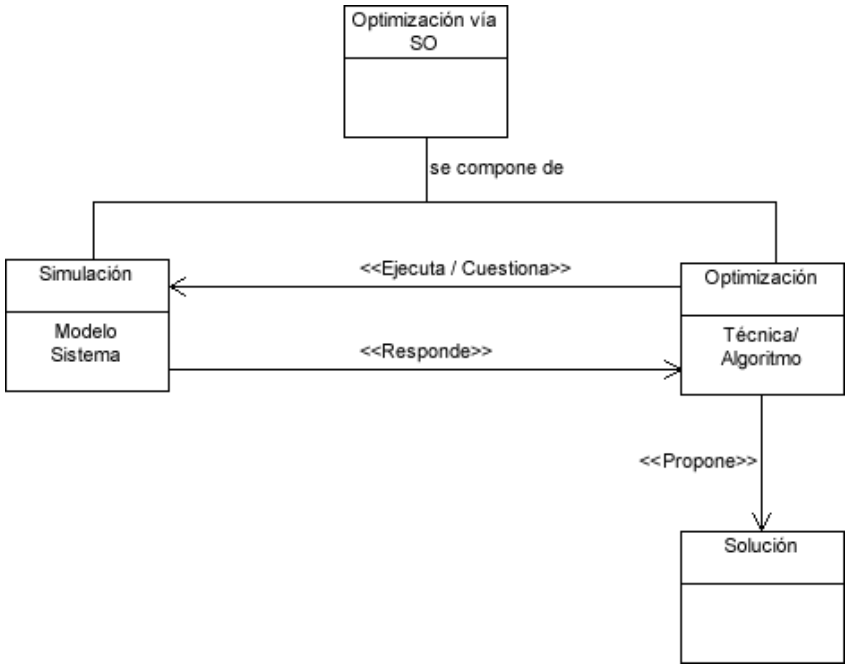


Figura 9. Modelo de Dominio Simulación y Optimización

En términos computacionales este modelo de dominio sugiere lo que ya se había visto previamente con respecto al diseño computacional básico de este tipo de software, y es que en términos computacionales un software basado en el concepto SO puede ser visualizado como un sistema con dos módulos o subsistemas principales, uno encargado de la simulación y el otro de la optimización trabajando juntos vía una interfaz de comunicación; y como consecuencia de ello el diseño de estos dos subsistemas puede ser llevado a cabo de manera independiente.

3.3 MÓDULO DE SIMULACIÓN

El módulo o subsistema de simulación no es otra cosa que el componente encargado de generar los modelos computacionales de simulación que representarán a un sistema/escenario en la problemática tratada, y para este

módulo en particular se fijaron varias metas en el diseño del mismo que, como se verá a continuación, están también encaminadas a la posibilidad de reutilizar este diseño en la elaboración de un software que permita realizar simulación discreta[8] de sistemas cuyo comportamiento puede ser imitado a través de la interacción líneas de espera.

1. Modelado de distintos tipos de sistemas de líneas de espera, desde el más simple (una fuente de arribo, una cola y un servidor) hasta algunos más complejos, como podrían ser aquellos que involucran más de una fuente de arribo y estaciones de servicio con múltiples servidores, etc.

2. Posibilidad de enlazar distintos sistemas de líneas de espera para el modelado de sistemas más complejos; en esta modalidad las salidas de servicio de un sistema podrían convertirse en los arribos de otro.

Esto es, de una manera más específica, y también como soporte del análisis de este módulo, buscamos que el diseño propuesto sea capaz de representar los escenarios mostrados a continuación [Tabla 4].

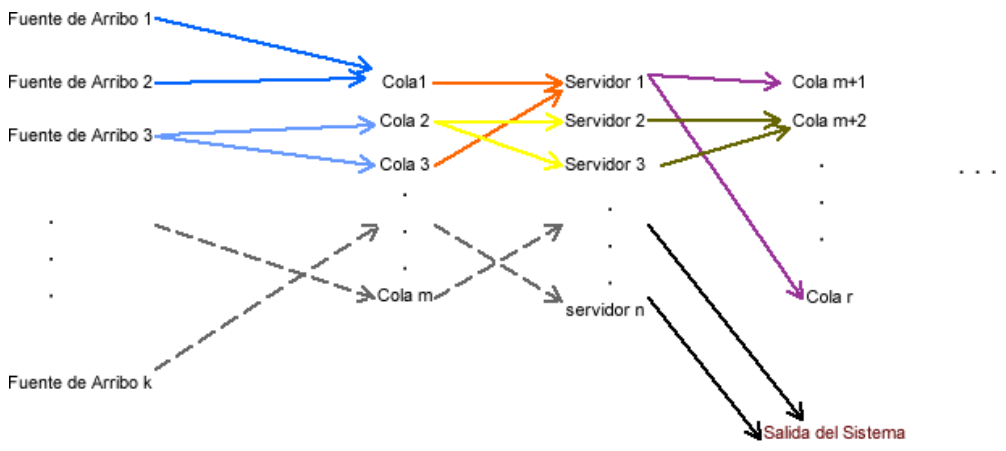


Figura 10. Escenarios Base Análisis

[8] En la simulación discreta la operación de un sistema es representada como una secuencia cronológica de eventos [25].

	Más de una fuente de arribo sirviendo a una sola cola
	Una sola fuente de arribos sirviendo a más de una cola
	Más de una cola enviando clientes a un solo servidor
	Una sola cola enviando clientes a más de un servidor
	Un solo servidor enviando clientes servidos a más de una cola
	Más de un servidor enviando clientes servidos a una sola cola
	Servidores expulsando clientes servidos fuera del sistema

Tabla 4. Descripción Escenarios Base Análisis

Es claro que estas metas no iban a ser fáciles de alcanzar si el desarrollo del software se basaba en un esquema tradicional de programación estructurada, es decir por medio de subrutinas y módulos, de ahí la necesidad de realizar un diseño de software como el que se presenta.

3.3.1 Análisis - Casos de Uso Simulación , Sujeto y Actores

La funcionalidad general requerida para este subsistema de simulación es presentada mediante un diagrama de caso de uso en la Figura 11. Como es posible observar, los casos de uso presentados en este diagrama para el *actor* "Simulación" ya manejan conceptos propios de un diseño de software orientado a objetos. El primer caso de uso contempla la generación de "entidades/componentes" necesarios para representar el sistema, esto no es otra cosa que crear y caracterizar los **objetos** necesarios que permitan, mediante su interacción, imitar el comportamiento de un sistema; otro aspecto importante de observar se deriva de los siguientes casos de uso:

- Inicializar Fuentes de Arribo al Sistema.

- Generación de eventos iniciales que detonarán la simulación

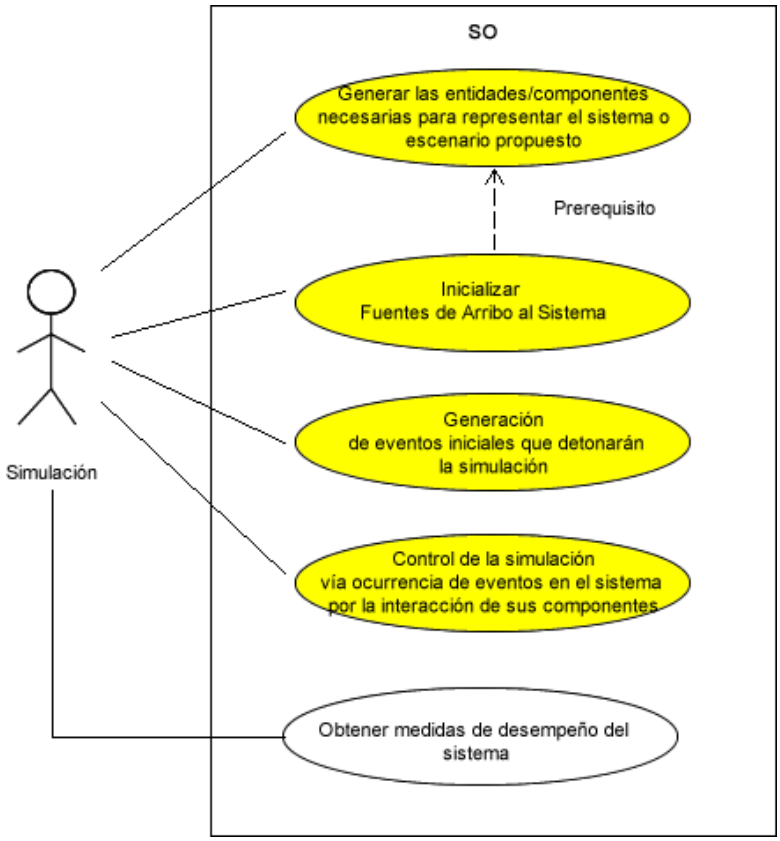


Figura 11. Diagrama de Caso de Uso Simulación

- Control de la simulación vía ocurrencia de eventos en el sistema por la interacción de sus componentes.

Y este es que, la manera en que se pretende imitar/simular el funcionamiento del sistema es a través de **eventos** o suceso detonados a lo largo del tiempo en la simulación.

3.3.2 Modelo Lógico – Modelo de Dominio Simulación

El siguiente modelo de dominio del módulo de simulación pretende plasmar las necesidades funcionales presentadas en el caso de uso anterior e integra ya algunos aspectos clave a considerar en diseño.

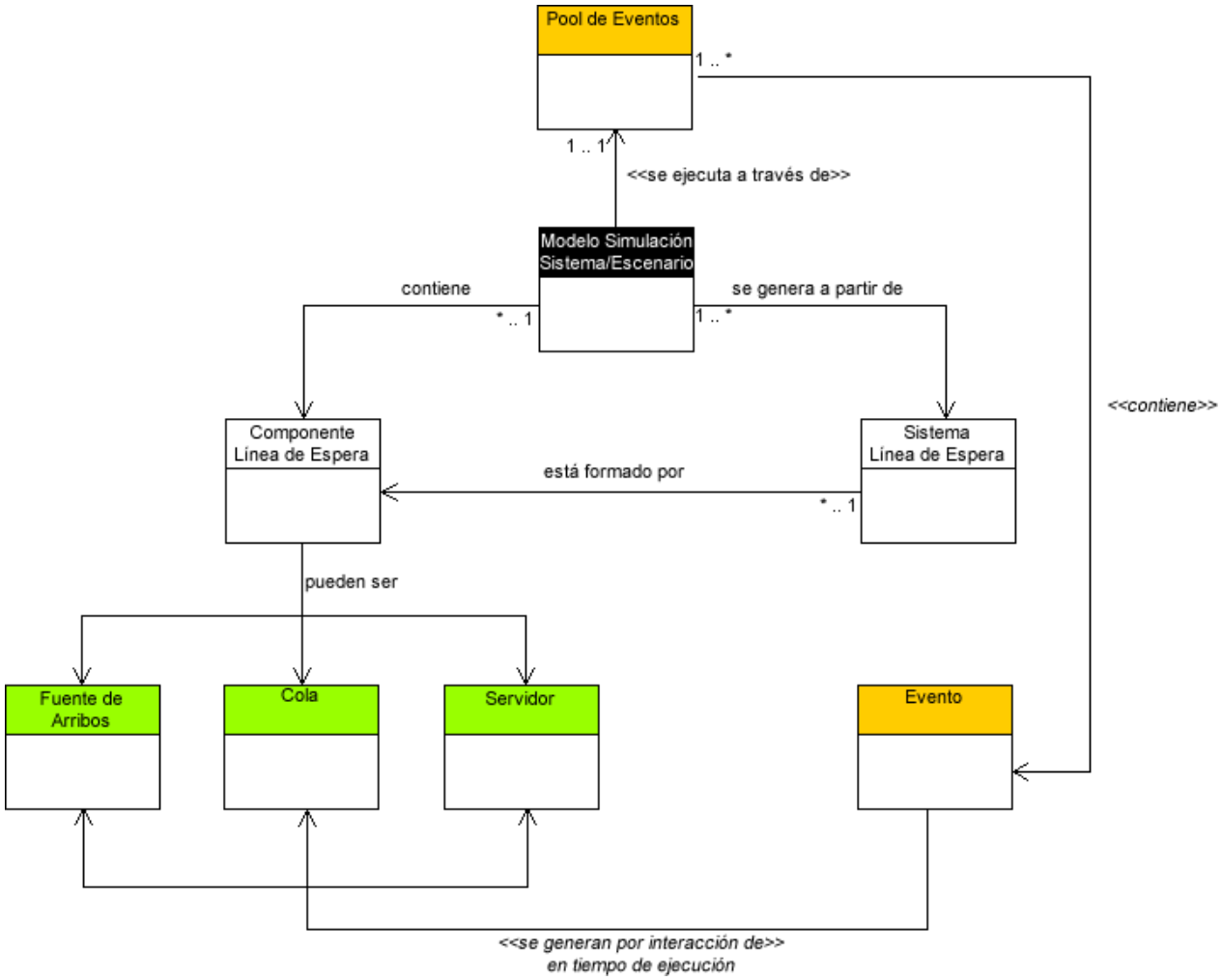


Figura 12. Modelo de Dominio Simulación

Es claro que este modelo de dominio gira en torno al modelo de simulación del sistema, mismo que:

1. Se generará a partir de sistemas de líneas de espera mismos que están compuestos por otros componentes más básicos que pueden ser: Fuentes de Arribos, Colas y Servidores.

2. Será ejecutado, “corrido” en términos computacionales, por medio de un pool de eventos. Este pool almacenará y despachará los eventos resultado de la

interacción, en tiempo de ejecución, de los componentes utilizados por el modelo: Fuentes de Arribos, Colas y Servidores.

3.3.3 Modelo Estático - Diagrama de Clase Simulación

Un paso por demás importante, si no es que el de mayor importancia, en un proceso de diseño de software orientado a objetos es la identificación de las *clases* o componentes "tipo" del sistema que, a través de su interacción y configuración, definen y hacen posible el funcionamiento del mismo.

Una clase es una caracterización precisa de propiedades estructurales o de comportamiento la cual es compartida por toda una colección de entidades [12].

En UML la herramienta que permite visualizar un sistema, y por consiguiente su estructura, con base a estos componentes tipo y sus relaciones, es conocida como diagrama de clases y una manera de definirlo es como una gráfica de elementos tipo conectados a través de las diferentes relaciones estáticas que se dan entre ellos [9].

El diagrama de clase utilizado en la etapa de análisis como base del diseño del módulo de simulación se muestra en la Figura 13.

Este diagrama permite visualizar a partir de que componentes tipo se realizará la construcción del módulo de simulación además de la forma en que estarán relacionados entre sí, al menos por ahora, desde un punto de vista estático.

Componentes Tipo

Las clases o componentes tipo para el módulo de simulación pueden ser categorizadas como sigue:

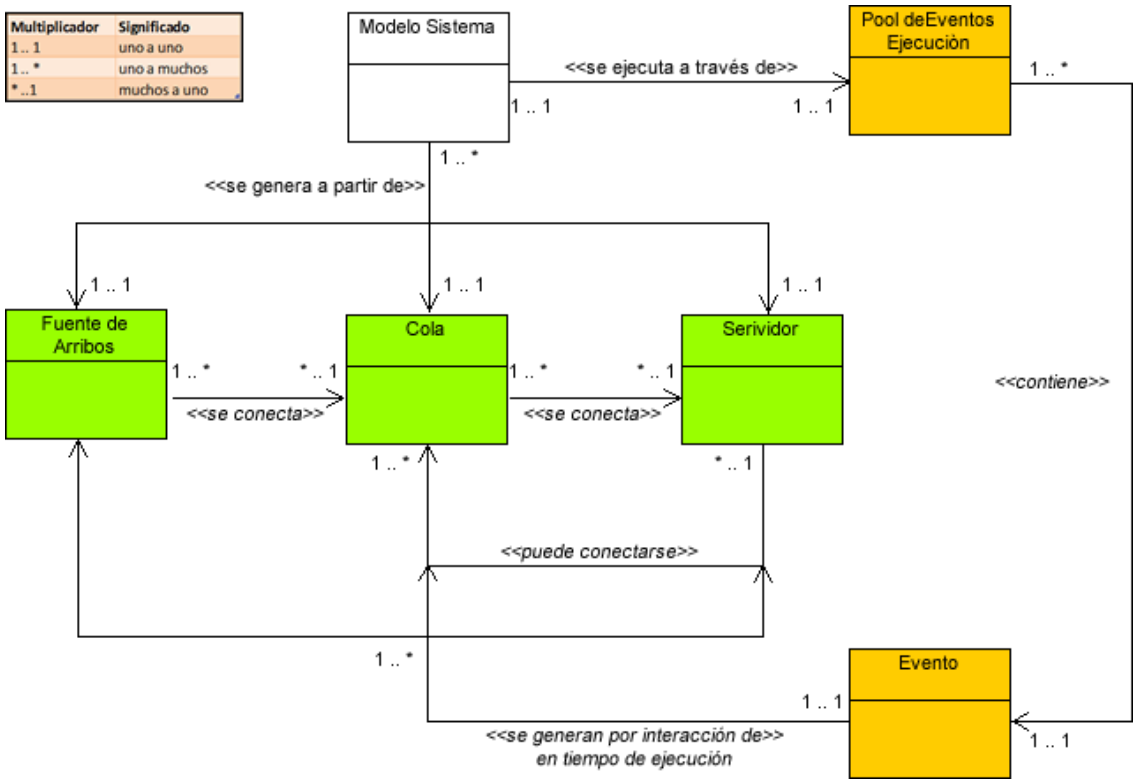


Figura 13. Diagrama de Clase Simulación - Etapa de Análisis

1. Clases relacionadas con la definición del modelo del sistema: **Modelo**, **Fuente de Arribos**, **Cola** y **Servidor**

2. Clases relacionadas con la ejecución de la simulación como tal: **Pool de Eventos**, **Evento**.

Vale la pena observar, que al menos la manera en que se da la identificación de las clases de la primera categoría, es muy sencilla y emerge de las etapas de análisis más tempranas con base al siguiente modelo:

Clientes provenientes de una (o varias) fuente(s) de arribo tienen como objetivo recibir un “servicio” en uno o varios servidores. Si hay un servidor disponible el cliente que arriba es recibido y espera ahí una determinada cantidad de tiempo mientras es atendido, después del cual el servidor vuelve a estar disponible para otro cliente que arribe o que esté esperando servicio. Si el cliente que arriba no encuentra un servidor disponible, éste puede esperar a que se le proporcione el servicio en un conjunto de espacios de espera conocidos como colas o bien marcharse [16].

Este concepto maneja los componentes *clave* presentes en cualquier sistema de este tipo y estos son **Fuentes de Arribo(s), Cola(s), y Servidor(es)**. Estos tres componentes, son definitorios en el comportamiento y funcionamiento del sistema, es decir, el número de cada uno de ellos presente en el mismo, así como su configuración individual, y la forma en que interactúan entre sí, define un modelo de línea de espera específico.

Para el caso de las clases mostradas en la segunda categoría, la idea de integrarlas al diseño del sistema está también ligada al concepto fundamental de línea de espera que se ha presentado, pero no desde el punto de vista de los elementos que componen y representan a un sistema de línea de espera, sino más bien de las herramientas que harán posible imitar el comportamiento del sistema de línea de espera por un periodo de tiempo una vez que se tienen los elementos antes mencionados.

RELACIONES ENTRE LOS COMPONENTES TIPO

En el diagrama de clase presentado para el módulo de simulación se encuentran también definidas las siguientes relaciones entre los componentes tipo del sistema:

Referencia	Entidad	Relación	Entidad
1	Modelo	*<<se genera a partir de>>*	**Fuente de arribos,** **Cola,** **Servidor**
2	Fuente de Arribo	*<<se conecta>>*	**Cola**
3	Cola	*<<se conecta>>*	**Servidor**
4*	Servidor	*<<puede conectarse>>*	**Cola**
5	Modelo	*<<se ejecuta a través de>>*	**Pool de Elementos**
6	Pool de Elementos	*<<contiene>>*	**Evento**
7	Eventos	*<<se generan por la interacción de>>* *en tiempo de ejecución*	**Fuente de Arribos,** **Cola,** **Servidor**

* Considera la posibilidad de enlazar la salida de servicio de un servidor hacia una cola, o lo que es lo mismo, las salidas de servicio de un servidor se convierten la fuente de arribos de una cola

Tabla 5. Relaciones Componentes Tipo - Clases Simulación

Estas relaciones tiene el objetivo de, por un lado permitir el modelado o representación de distintos sistemas basados en líneas de espera [Figura 10] así como, el de hacer posible la ejecución de los modelos representados por un periodo de tiempo específico [Tabla 6].

Referencia Relación	Entidad de Partida	Entidad Destino	Cardinalidad	Lectura	Observaciones
1	Modelo	Fuente de Arribos	uno a muchos	Un **Modelo** tiene asociado una o varias **Fuentes de Arribos**.	Da la posibilidad de modelar sistema con múltiples fuentes de arribo colas y servidores.
1	Fuente de Arribos	Modelo	uno a uno	Una **Fuente de Arribos** tiene asociado uno y solo un **Modelo.**	
1	Modelo	Cola	uno a muchos	Un **Modelo** tiene asociado una o varias **Colas**.	
1	Cola	Modelo	uno a uno	Una **Cola** tiene asociado uno y solo un **Modelo.**	
1	Modelo	Servidor	uno a muchos	Un **Modelo** tiene asociado uno o varios **Servidores**.	
1	Servidor	Modelo	uno a uno	Un **Servidor** tiene asociado uno y solo un **Modelo.**	
2	Fuente de Arribos	Cola	uno a muchos	Una **Fuente de Arribos** tiene asociado una o varias **Colas.**	Da la posibilidad de tener una sola fuente de arribos sirviendo a más de una cola.
2	Cola	Fuente de Arribos	uno a muchos	Una **Cola** tiene asociado una o varias **Fuente de Arribos.**	Da la posibilidad de tener más de una fuente de arribo sirviendo a una sola cola.
3	Cola	Servidor	uno a muchos	Una **Cola** tiene asociado uno o varios **Servidores.**	Da la posibilidad de tener una sola cola enviando clientes a más de un servidor.
3	Servidor	Cola	uno a muchos	Un **Servidor** tiene asociado una o varias **Colas.**	Da la posibilidad de tener más de una cola enviando clientes a un solo servidor.
4	Servidor	Cola	uno a muchos	Un **Servidor** tiene asociado una o varias **Colas.**	Da la posibilidad de tener un solo servidor enviando clientes servidos a más de una cola.

Referencia Relación	Entidad de Partida	Entidad Destino	Cardinalidad	Lectura	Observaciones
4	Cola	Servidor	uno a muchos	Una **Cola** tiene asociado uno o varios **Servidores.**	Da la posibilidad de tener más de un servidor enviando clientes servidos a una sola cola.
5	Modelo	Pool de Eventos	uno a uno	Un **Modelo** tiene asociado uno y solo un **Pool de Eventos.**	Hace posible y controla además la ejecución del modelo de simulación.
5	Pool de Eventos	Modelo	uno a uno	Un **Pool de Eventos** tiene asociado uno y solo un **Modelo.**	
6	Pool de Eventos	Evento	uno a muchos	Un **Pool de Eventos** tiene asociado uno o muchos **Eventos.**	
6	Evento	Pool de Eventos	uno a uno	Un **Evento** tiene asociado uno y solo un **Pool de Eventos.**	
7	Evento	Fuente de Arribos, Cola, Servidor	uno a uno	Un **Evento** esta relacionado con una y solo una de las siguientes entidades **Fuente de Arribos, Cola y Servidor.**	
7	Fuente de Arribos, Cola, Servidor	Evento	uno a muchos	Cada una de las siguientes entidades **Fuente de Arribos, Cola y Servidor** tiene asociado uno o varios **Eventos.**	

* Considera la posibilidad de enlazar la salida de servicio de un servidor hacia una cola, o lo que es lo mismo, las salidas de servicio de un servidor se convierten la fuente de arribos de una cola

Tabla 6. Cardinalidad Relaciones Componentes Tipo / Clases Simulación

3.3.4 MODELO ESTÁTICO - DISEÑO CLASES SIMULACIÓN

En el ámbito de diseño y programación orientada a objetos lo único que se codifica son las clases, mismas que guardan la definición y patrones de comportamiento intrínseco del componente “tipo” del sistema, a través de lo que se conoce como *propiedades/atributos* y *métodos* [17]. En tiempo de ejecución estas clases podrán ser instanciadas a manera de *objetos* las veces que sea necesario para modelar un sistema específico. Por poner un ejemplo, en el caso que nos ocupa, si el sistema de línea de espera que se desea modelar requiriera del uso de dos servidores, sería necesario entonces instanciar dos veces en memoria la clase **Servidor**; lo mismo aplicaría si se tuviera más de una fuente de arribos o cola, sin entrar en mayor detalle, mencionaremos además que las características propias del diseño y programación de esta técnica permiten que todas estas instancias (*objetos*) coexistan de manera independiente, aun siendo de la misma *clase*, esto es muy importante, ya que hace posible asignar distintos patrones de comportamiento a objetos de la misma clase, solo por poner un ejemplo, en el caso de los dos servidores, las dos instancias (objetos) de la clase **Servidor** podrían: manejar diferentes tiempos/patrones de servicio, atender distintas fuentes de arribo, tener distintas capacidades de servicio, etc.

UML provee una notación gráfica sujeta a ciertos estándares para la declaración y definición de clases, sin embargo, y con la finalidad de que la definición de estos componentes sea más sencilla de entender, el detalle de diseño de los mismos en el presente documento es expuesto utilizando únicamente tablas en las que clase por clase se hace la definición de los atributos y métodos respectivos.

Adicionalmente el soporte de diseño de las clases con estándares UML puede ser consultado en la sección de Anexos del presente documento [Figura 37].

Clase Fuente de Arribos

Atributos

Nombre	Tipo	Descripción	Valores Fijos
index	Numérico Entero	Identificador numérico para los objetos instanciados a partir de esta clase	
name	Alfanumérico	Identificador alfanumérico para los objetos instanciados a partir de esta clase	
sampling_size	Numérico Entero	Tamaño de la muestra de tiempos de arribo que contemplará el objeto (No necesariamente el número de arribos ya que estos podrían darse en grupo)	
distribution	Distribution	Distribución probabilística para los tiempos entre arribos	DET = Determinística, EXP = Exponencial
batch_distribution	Distribution	Distribución probabilística para el número de clientes que puede tener un solo arribo (arribos en grupo)	DET = Determinística
followers[]	Arreglo Objetos Instanciados desde la Clase Cola	Colas que usarán esta fuente de arribos	
output_rule	OutputRule	Regla de enrutamiento para cuando la fuente de arribos es usada por más de un objeto/cola.	SHQUE = Cola más corta, LOQUE = Cola más larga, RDN = Selección aleatoria de la Cola
output_rule_apply_to	Alfanumérico	Define si el atributo output rule aplica por cliente o por grupo	E = Elemento, B = Grupo
times_arrives[]	Arreglo Numérico	Tiempo de arribo por cliente, para cada cliente en el objeto	
status_elem[]	Arreglo Carácter	Status de cada cliente: 1. No procesado si todavía no ha sido asignado a una cola 2. Procesado si ya fue asignado a una cola	U=No Procesado, P = Procesado
batches[]	Arreglo Numérico Entero	Número de clientes que arribaron al mismo tiempo por cada cliente	
current_element	Numérico Entero	Primer cliente con status no procesado en el objeto	

Tabla 7. Diseño Clase Fuente de Arribos - Atributos

Clase Fuente de Arribos

Métodos

Name	Return	Parámetros	Funcionalidad
build_elements	NA	*sampling_size** Tamaño muestra tiempos de arribo, *distribution** Distribución de tiempos de arribo, batch_distribution* Distribución número de clientes por grupo	Genera todos los arribos/clientes que manejará el objeto, inicializa los atributos times_arrives[] y status_elem[]
arrive_process	NA	*idx_current_element** Indice arribo a partir del cual se procesa, *sys_events_obj* Pool de eventos	Envía los clientes de un arribo específico a los objetos especificados en el atributo followers según el output_rule seleccionado.

*Parámetros Implícitos en los atributos de la clase

Tabla 8. Diseño Clase Fuente de Arribos - Métodos

Clase Cola

Atributos

Nombre	Tipo	Descripción	Valores Fijos
index	Numérico Entero	Identificador numérico para los objetos instanciados a partir de esta clase	
name	Alfanumérico	Identificador alfanumérico para los objetos instanciados a partir de esta clase	
capacity	Numérico Entero	Capacidad de la cola	
discipline	Discipline	Regla en que los clientes son tomados de la cola para ser transferidos a un servidor.	FIFO = Primero en entrar primero en salir, LIFO = Último en entrar primero en salir, RDN = Selección aleatoria de la los clientes que pasarán al servidor primero
followers[]	Arreglo Objetos Instanciados desde la Clase Server	Servidores que usarán esta cola	
source_obj_elem[]	Arreglo Alfanumérico	Nombre del objeto arribo fuente donde fue generado el cliente	
idx_arrive_elem[]	Arreglo Numérico Entero	Índice del elemento en los arreglos times_arrives[], status_elem[] y batches[] del objeto arribo fuente donde fue generado el cliente	
input_clock_elem[]	Arreglo Numérico	Tiempo de entrada de cada cliente a la cola	
ouput_clock_elem[]	Arreglo Numérico	Tiempo de salida de cada cliente a la cola	
status_elem[]	Arreglo Carácter	Status de cada cliente: 1. No procesado si permanece en la cola y todavía no ha sido asignado a un server 2. Procesado si ya fue asignado a un server	U=No Procesado, P = Procesado
isempty	Boolean	Indica si la cola está vacía	
size	Numérico Entero	Longitud de la cola	
processed_elems	Numérico Entero	Número de clientes procesados	

Clase Cola

Atributos

Nombre	Tipo	Descripción	Valores Fijos
discarted_elems	Numérico Entero	Número de clientes que abandonaron el sistema	

Tabla 9. Diseño Clase Cola - Atributos

Métodos

Name	Return	Parámetros	Funcionalidad
input_element	NA	*source_obj_name* Nombre del objeto arribo fuente donde fue generado el cliente, *idx_arrive* Indice del elemento en los arreglos times_arrives[], status_elem[] y batches[] del objeto arribo fuente donde fue generado el cliente, *clock* Tiempo en que ocurre la entrada a la cola, *sys_events_obj* Pool de eventos	Registra el ingreso de clientes a la cola
output_element	Long (Idx cliente que sale de la cola)	*clock* Tiempo en que ocurre la salida de la cola, *sys_events_obj* Pool de eventos	Registra la salida de clientes de la cola

Tabla 10. Diseño Clase Cola - Métodos

Clase Servidor

Atributos

Nombre	Tipo	Descripción	Valores Fijos
index	Numérico Entero	Identificador numérico para los objetos instanciados a partir de esta clase	
name	Alfanumérico	Identificador alfanumérico para los objetos instanciados a partir de esta clase	
sampling_size	Numérico Entero	Tamaño de la muestra de tiempos de arribo que contemplará el objeto (No necesariamente el número de arribos ya que estos podrían darse en grupo)	
distribution	Distribution	Distribución probabilística para los tiempos de servicio	DET = Determinística, EXP = Exponencial
batch_distribution	Distribution	Distribución probabilística para el número de clientes que puede ser servido a la vez (servicio en grupo)	DET = Determinística
min_serv_percent	Numérico	En caso de que el servidor pueda dar servicio a más de un cliente a la vez indica cual es el mínimo de clientes requerido para iniciar el servicio	Entre 0 y 1
followers[]	Arreglo Objetos Instanciados desde la Clase Que	Colas que usarán como fuente de arribos los clientes servidos por este servidor	
output_rule	OutputRule	Regla de enrutamiento para cuando los clientes servidos por este servidor sean la fuente de arribos usada por más de un objeto/cola.	SHQUE = Cola más corta, LOQUE = Cola más larga, RDN = Selección aleatoria de la Cola
output_rule_apply_to	Alfanumérico	Define si el atributo output rule aplica por cliente o por grupo servido	E = Elemento, B = Grupo
input_rule	InputRule	Regla de enrutamiento para cuando el servidor sea abastecido por más de un objeto/cola.	SHQUE = Cola más corta, LOQUE = Cola más larga, RDN = Selección aleatoria de la Cola
input_rule_apply_to	Alfanumérico	Define si el atributo input rule aplica por cliente o por grupo	E = Elemento, B = Grupo
source_obj_elem[]	Arreglo Alfanumérico	Nombre del objeto arribo fuente donde fue generado el cliente	

Clase Servidor

Atributos

Nombre	Tipo	Descripción	Valores Fijos
idx_arrive_elem[]	Arreglo Numérico Entero	Índice del elemento en los arreglos times_arrives[], status_elem[] y batches[] del objeto arribo fuente donde fue generado el cliente	
input_clock_elem[]	Arreglo Numérico	Tiempo de entrada de cada cliente al servidor	
ouput_clock_elem[]	Arreglo Numérico	Tiempo de salida de cada cliente al servidor	
time_services[]	Arreglo Numérico	Tiempo de servicio de cada cliente en el servidor	
status_elem[]	Arreglo Carácter	Status de cada cliente: 1. No procesado si permanece en servicio 2. Procesado si ya fue servido	U=No Procesado, P = Procesado
batches[]	Arreglo Numérico Entero	Número de clientes que entraron al servicio al mismo tiempo por cada cliente	
isbusy	Boolean	Indica si el servidor está ocupado	
pending_input	Boolean	Indica si hay un evento de entrada pendiente desde una fila vacía	
pending_input_time	Numeric	Tiempo en que se dará el evento de entrada pendiente	
output_response	Numérico Entero	Número de clientes que han sido servidos por el servidor	
current_element	Numérico Entero	Primer cliente con status no procesado en el objeto	
infinite_arrive_source	Boolean	Indica si el está enlazado con una fuente infinita de arribos supone Batch = 1	

Tabla 11. Diseño Clase Servidor - Atributos

Clase Servidor

Métodos

Name	Return	Parámetros	Funcionalidad
build_first_ISRV_events	NA	NA	Generación de los eventos para los arribos iniciales al servidor: add_event(ISRV, 0)* *solo si infinite_arrive_source = true
input_element	NA	*source_obj_name* Nombre del objeto arribo fuente donde fue generado el cliente, *idx_arrive* Indice del elemento en los arreglos times_arrives[], status_elem[] y batches[] del objeto arribo fuente donde fue generado el cliente, *clock* Tiempo en que ocurre la entrada al servidor, *sys_events_obj* Pool de eventos	Registra el ingreso de clientes a la cola
output_element	NA	*clock* Tiempo en que ocurre la salida del servidor, *sys_events_obj* Pool de eventos	Registra la salida de clientes de la cola

Tabla 12. Diseño Clase Servidor - Métodos

Clase Pool de Eventos

Atributos

Nombre	Tipo	Descripción	Valores Fijos
IQUE_Events[]	Arreglo Eventos	Pool de eventos Input Que "IQUE" generados en tiempo de ejecución del modelo de simulación	
ISRV_Events[]	Arreglo Eventos	Pool de eventos Input Server "ISRV" generados en tiempo de ejecución del modelo de simulación	
OSRV_Events[]	Arreglo Eventos	Pool de eventos Output Server "OSRV" generados en tiempo de ejecución del modelo de simulación	
curr_event_IQUE	Numérico Entero	Índice del último evento "IQUE" procesado	
curr_event_ISRV	Numérico Entero	Índice del último evento "ISRV" procesado	
curr_event_OSRV	Numérico Entero	Índice del último evento "OSRV" procesado	

Tabla 13. Diseño Clase Pool de Eventos - Atributos

Clase Pool de Eventos

Métodos

Name	Return	Parámetros	Funcionalidad
event_pool_ini <<constructor>>	NA	NA	**I.** Ejecuta el método build_elements de todos los objetos instanciados desde la clase Fuente de Arribos **II.** Genera los eventos "IQUE" de todas las fuentes de arribo **III.** Genera los eventos "ISRV" de todos los objetos instanciados desde la clase servidor con fuente de arribos infinita
add_event	NA	*event* Evento	Añade un evento al pool de eventos correspondiente: • IQUE_Events[] • ISRV_Events[] • OSRV_Events[]

Tabla 14. Diseño Clase Pool de Eventos - Métodos

Clase Evento

Atributos

Nombre	Tipo	Descripción	Valores Fijos
event_type	Tipo Evento	Evento	IQUE= Input Que, ISRV = Input Server, OSRV = Output Server
clock	Numérico	Tiempo en que ocurre el evento durante la ejecución del modelo de simulación	
object/entity_type	Clase del objeto/entidad con la que se relaciona el evento	Clase del objeto/entidad	A=fuente de arribos, S=servidor
entity_index	Numérico Entero	Identificador numérico del objeto/entidad	
entity_subindex	Numérico Entero	Índice del Elemento/Cliente al que se hace referencia en la entidad/objeto	
event_status	Carácter	Status del evento	U=No Procesado, P = Procesado, W=Está en proceso

Tabla 15. Diseño Clase Evento - Atributos

Clase Modelo

Atributos

Nombre	Tipo	Descripción	Valores Fijos
name	Alfanumérico	Identificador alfanumérico para los objetos instanciados a partir de esta clase	
arrives[]	Arreglo de objetos instanciados desde la clase Fuentes de Arribo	Fuentes de arribo que manejará el sistema	
ques[]	Arreglo de objetos instanciados desde la clase Cola	Colas que manejará el sistema	
server[]	Arreglo de objetos instanciados desde la clase Servidor	Servidores que manejará el sistema	
pool_events	Pool Eventos	Pool de Eventos para la ejecución del modelo	
Warmup	Numérico	Tiempo "Precalentamiento" del sistema	
run_period	Numérico	Tiempo en que se ejecutará la simulación cuando se ejecute el modelo	

Tabla 16. Diseño Clase Modelo - Atributos

Métodos

Name	Return	Parámetros	Funcionalidad
model_ini()	NA	NA	Genera los objetos necesarios (arrives[], ques[], server[]) y asigna los atributos de los mismos, para representar el sistema
simulation()	NA	NA	Corre el modelo de simulación por el tiempo establecido en el atributo run_period

Tabla 17. Diseño Clase Modelo - Métodos

3.3.5 MODELO FUNCIONAL

El modelo funcional que se presenta a continuación define la manera en la que el modelo estático trabaja para cumplir los dos objetivos principales del módulo de simulación:

1. Generar un modelo de simulación que represente el sistema/escenario.

2. Ejecutar este modelo por un periodo de tiempo específico para obtener una media de efectividad del sistema

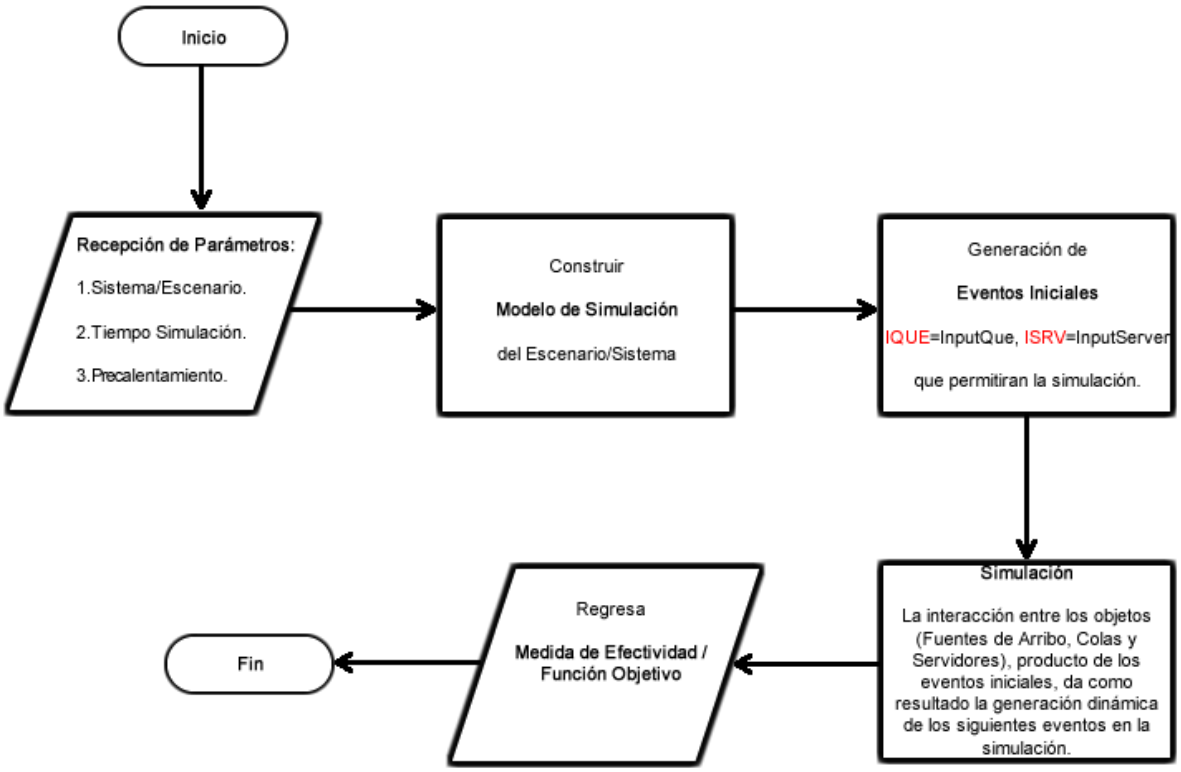


Tabla 18. Modelo Funcional Simulación

DETALLE PROCESO MODELO FUNCIONAL SIMULACIÓN

I. Recepción de parámetros (serán enviados por el módulo de optimización).

Estos parámetros definen el sistema /escenario a modelar, así como el tiempo que habrá que simular o "ejecutar" el modelo generado antes de obtener la medida de efectividad del sistema:

1. **Sistema/Escenario**. Definición del sistema/escenario.
Fuentes de Arribo, Colas y Servidores que en conjunto con sus parámetros y relaciones permiten emular el sistema real.

El medio por el cual serán representados los sistemas/escenarios, para la aplicación de este diseño al caso de estudio, es a través del componente tipo **scenario**, cuya definición se basa en la siguiente estructura de datos.

Nombre	Tipo	Descripción
workstations (WSs)	Numérico Entero	Número de estaciones de trabajo que tendrá el modelo
num_servers []	Arreglo Numérico Entero	Arreglo con la información del número de servidores que contendrá cada estación de trabajo
buffer_capacity[]	Arreglo Numérico Entero	Arreglo con la información del tamaño de los Buffers (capacidad de cada cola en el modelo)
service_time_dist[]	Arreglo Distribución	Arreglo con la información de la distribución de tiempos de servicio por estación de trabajo

Tabla 19. Diseño Componente Tipo Scenario

Es muy importante señalar que, aun cuando, este componente tipo fue creado para representar específicamente sistemas/escenarios del caso de estudio abordado en el capítulo II del presente trabajo, el diseño presentado tiene la capacidad de modelar una gran variedad de sistemas cuyo comportamiento puede ser representado con el uso de líneas de espera.

2. **Tiempo Simulación**. Tiempo por el que se simulará el sistema.

3. **Precalentamiento**. Periodo por el cual será ejecutado el sistema antes de comenzar a tomar el **Tiempo Simulación**.

II. Construir Modelo de Simulación.

Construcción del modelo de simulación a través de la generación e inicialización de los objetos (Fuentes de Arribo, Colas y Servidores) que serán utilizados para representar el sistema.

En la aplicación del diseño al caso de estudio esta tarea se llevará a cabo instanciando un objeto a partir de la clase **Modelo** y haciendo un llamado al método *model_ini* con los parámetros recibidos.

REF: Figura 15. SD (Diagrama de Secuencia) modelo.model_ini().*****

*El diseño de este método trabaja conjuntamente con el componente tipo (Clase) **Scenario** y está específicamente desarrollado para representar sistemas/escenarios del caso de estudio presentado en el capítulo II.

III. Generación de Eventos Iniciales.

Generación de Eventos Iniciales *Entradas a Cola* (InputQue=**IQUE**), y *Entradas a Servicio* (InputServer =**ISRV**) que permitirán detonar el inicio de la simulación. Esto se lleva a cabo instanciando un objeto a partir de la clase **Pool de Eventos**.

REF: Figura 17. SD (Diagrama de Secuencia) pool.event_pool_ini() <<constructor>>.

IV. Simulación.

El proceso de simulación del modelo se lleva a cabo mediante el método *simulation* del objeto **Modelo** previamente instanciado.

REF: Figura 18. SD (Diagrama de Secuencia) modelo.simulation().

Este método controla la simulación del sistema mediante:

1. La utilización del objeto pool de eventos, ejecutando cada uno de los siguientes eventos según corresponda en el tiempo.

	IQUE = Input Que	**ISRV** = Input Server	**OSRV** = Output Server
Descripción	*Entradas a Colas*	*Entradas a Servicio*	*Salidas de Servicio*
Referencia (Diagrama de Secuencia)	REF: Figura 20. SD (Diagrama de Secuencia) Evento IQUE / Método: arrive_process Clase: Fuente de Arribos	REF: Figura 21. SD (Diagrama de Secuencia) Evento ISRV / Método: input_element Clase: Servidor	REF: Figura 23. SD (Diagrama de Secuencia) Evento OSRV / Método: output_element Clase: Servidor

Tabla 20. Eventos del Sistema

2. La interacción entre los objetos (Fuentes de Arribo, Colas y Servidores), producto de la simulación de los eventos antes señalados, da como resultado la generación dinámica de los siguientes eventos de este tipo que se presentarán en el sistema [Figura 14].

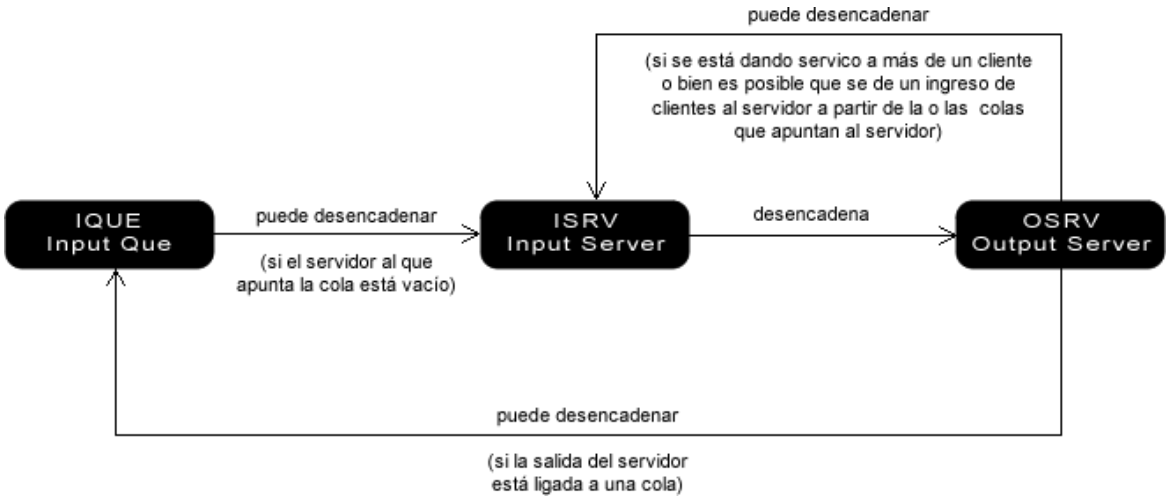


Figura 14. Modelo Dinámico de Eventos Simulación

Los nuevos eventos resultado de la misma simulación son añadidos al objeto pool de eventos mediante el método add_event de este mismo objeto.

REF:Figura 25. SD (Diagrama de Secuencia) pool.add_event (event Event).

V. Regresa Medida de Efectividad.

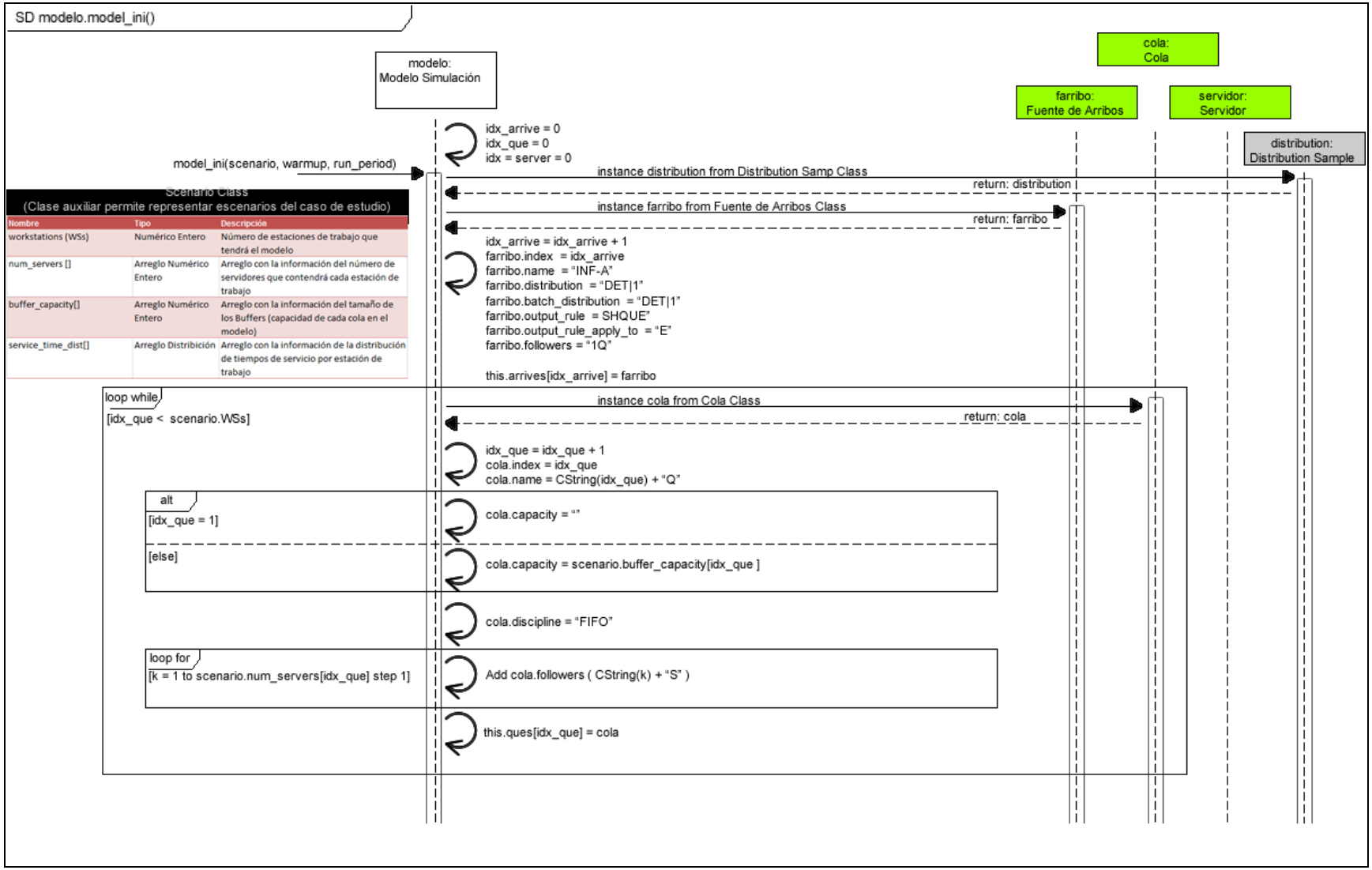
Después de ejecutar el modelo de simulación del sistema se regresa la medida de efectividad o bien los parámetros requeridos para el cálculo de la misma. Para la problemática general que se está abordando, en la que la medida de efectividad del sistema depende solo de los ítems convertidos en producto, una vez terminada la simulación del escenario propuesto, el número de ítems convertidos en producto podrá ser calculado mediante el atributo *output_response* de todos los objetos instanciados desde la clase servidor en los que el valor del atributo *followers[]* es un arreglo vacío[Tabla 11]. El atributo *output_response* de la clase servidor es actualizado por el evento **OSRV** REF: Figura 23. SD (Diagrama de Secuencia) Evento OSRV.

3.3.6 Modelo Dinámico – Diagramas de Secuencia Simulación

El modelo dinámico de un diseño de software orientado a objetos es soportado mayormente mediante el uso de diagramas de secuencia.

Los diagramas de secuencia son utilizados primordialmente para mostrar las interacciones entre objetos en el orden secuencial en que estas interacciones ocurren [18]. Con el uso de este tipo de diagramas además es posible modelar funciones y métodos.

Las páginas siguientes muestran los diagramas de secuencia correspondientes al diseño del módulo de simulación, a fin de facilitar su lectura se deja la siguiente referencia [19].



Scenario Class
(Clase auxiliar permite representar escenarios del caso de estudio)

Nombre	Tipo	Descripción
workstations (WSs)	Numérico Entero	Número de estaciones de trabajo que tendrá el modelo
num_servers []	Arreglo Numérico Entero	Arreglo con la información del número de servidores que contendrá cada estación de trabajo
buffer_capacity[]	Arreglo Numérico Entero	Arreglo con la información del tamaño de los Buffers (capacidad de cada cola en el modelo)
service_time_dist[]	Arreglo Distribición	Arreglo con la información de la distribución de tiempos de servicio por estación de trabajo

Figura 15. SD (Diagrama de Secuencia) modelo.model_ini()

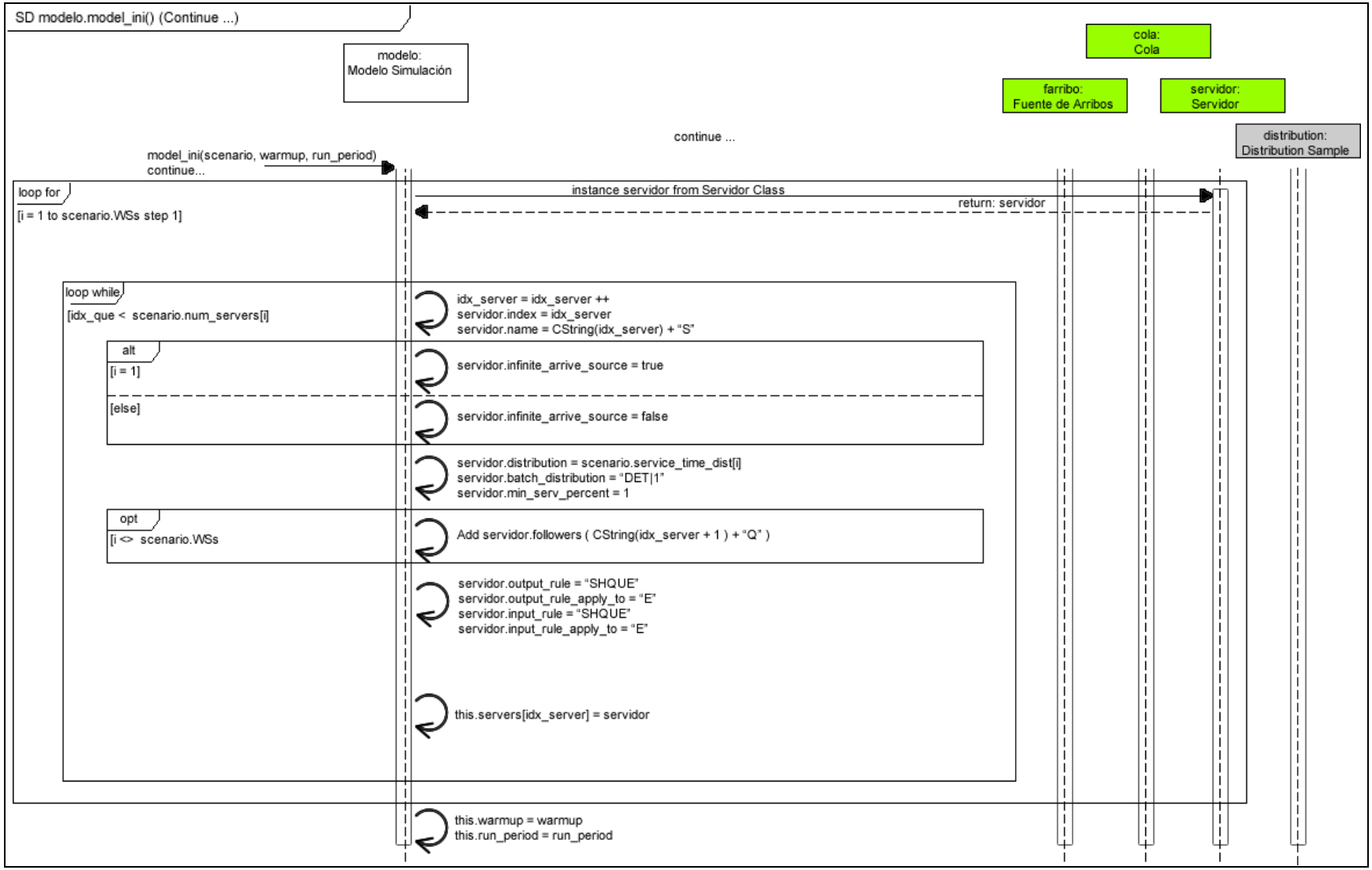


Figura 16. SD (Diagrama de Secuencia) modelo.model_ini() (Continue ...)

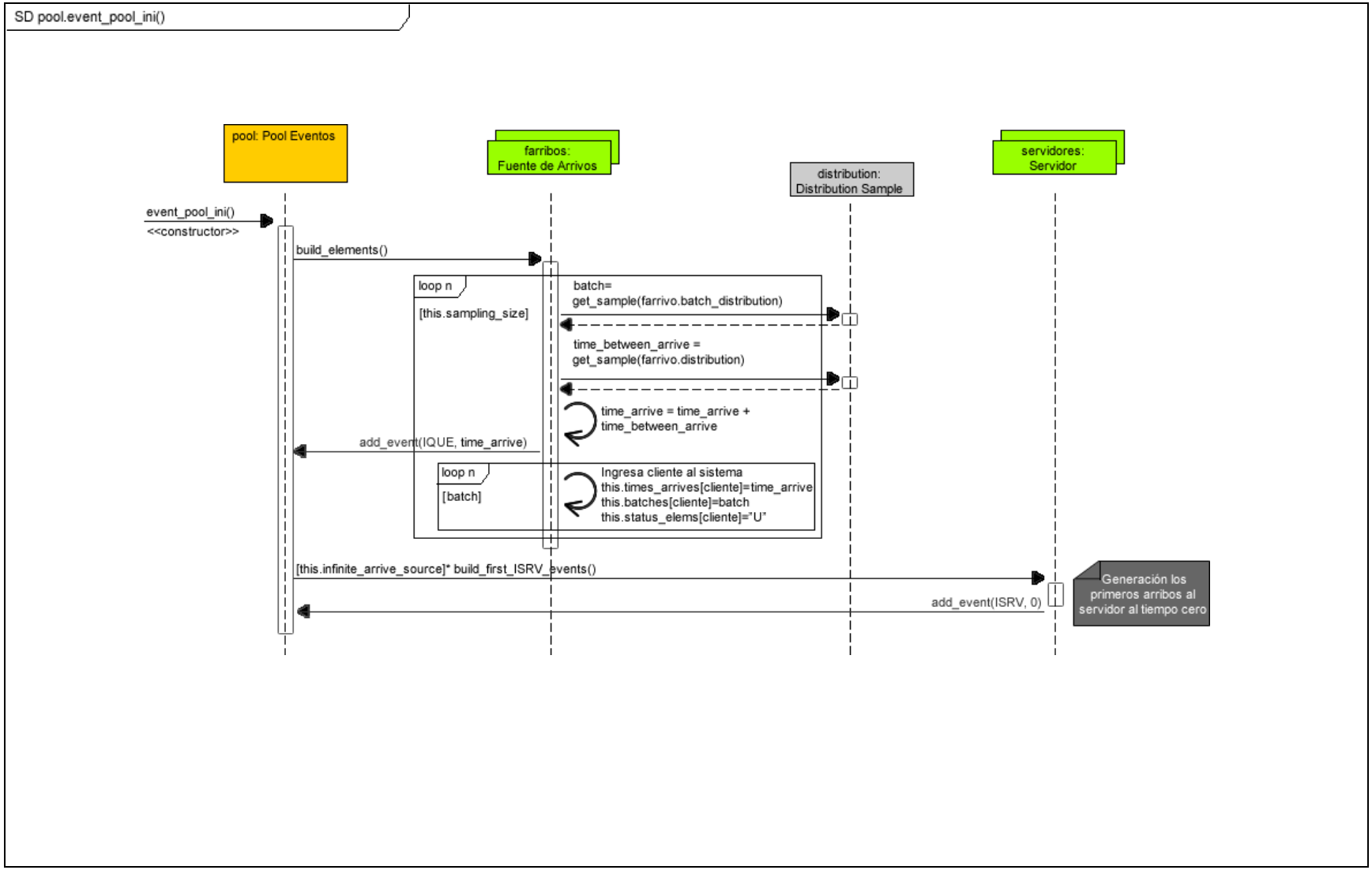


Figura 17. SD (Diagrama de Secuencia) pool.event_pool_ini() <<constructor>>

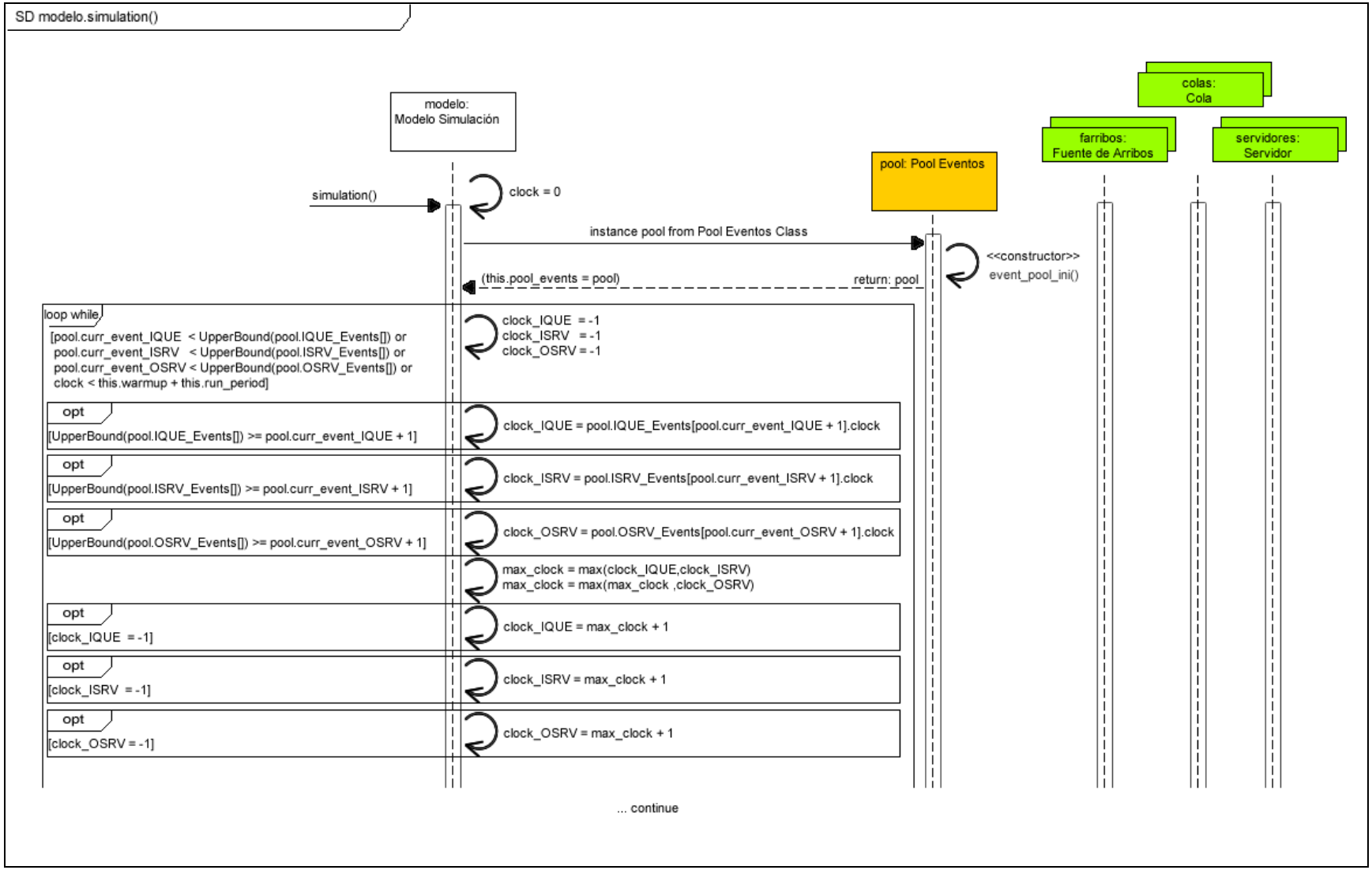


Figura 18. SD (Diagrama de Secuencia) modelo.simulation()

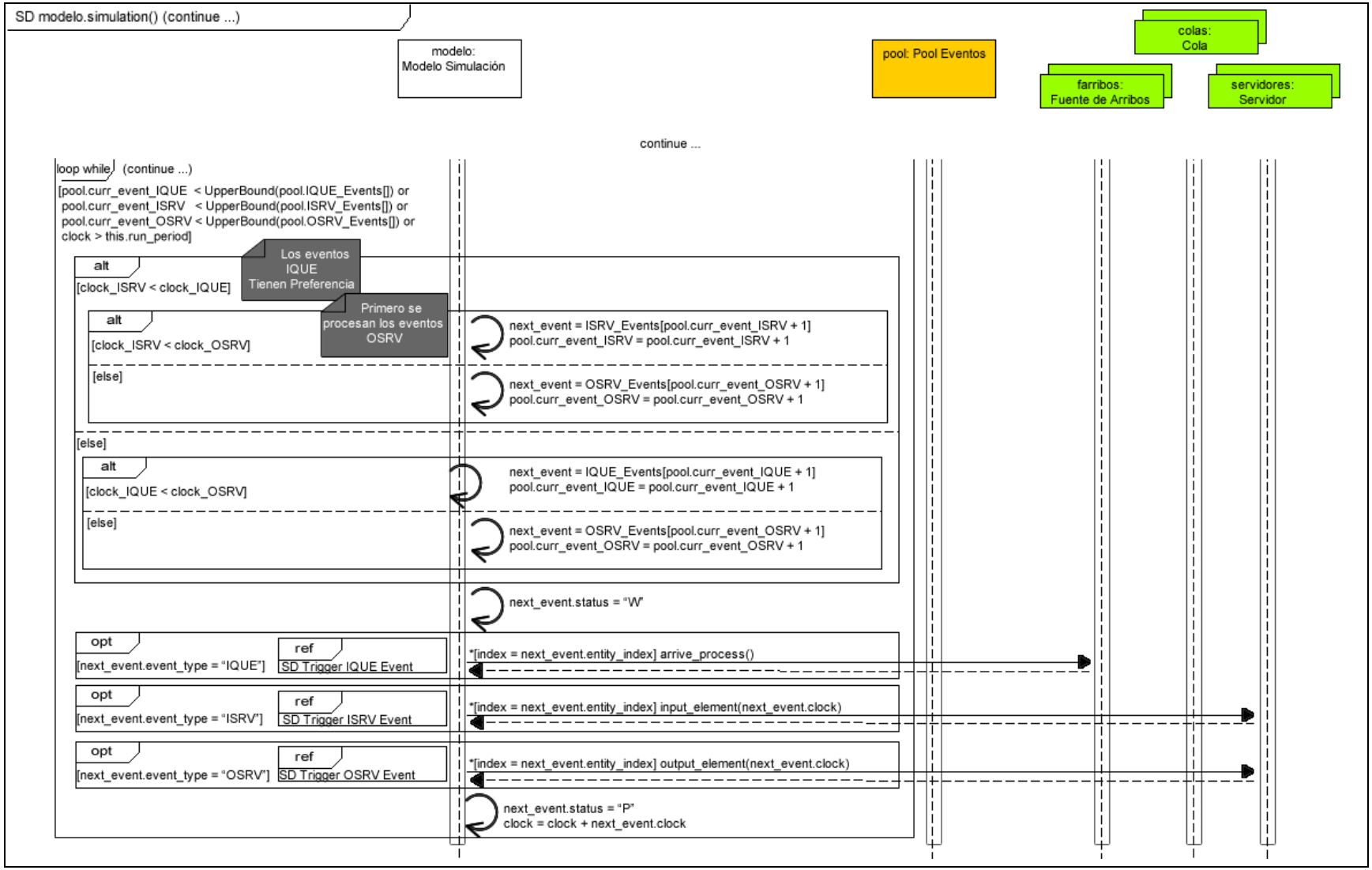


Figura 19 SD (Diagrama de Secuencia) modelo.simulation() (Continue ...)

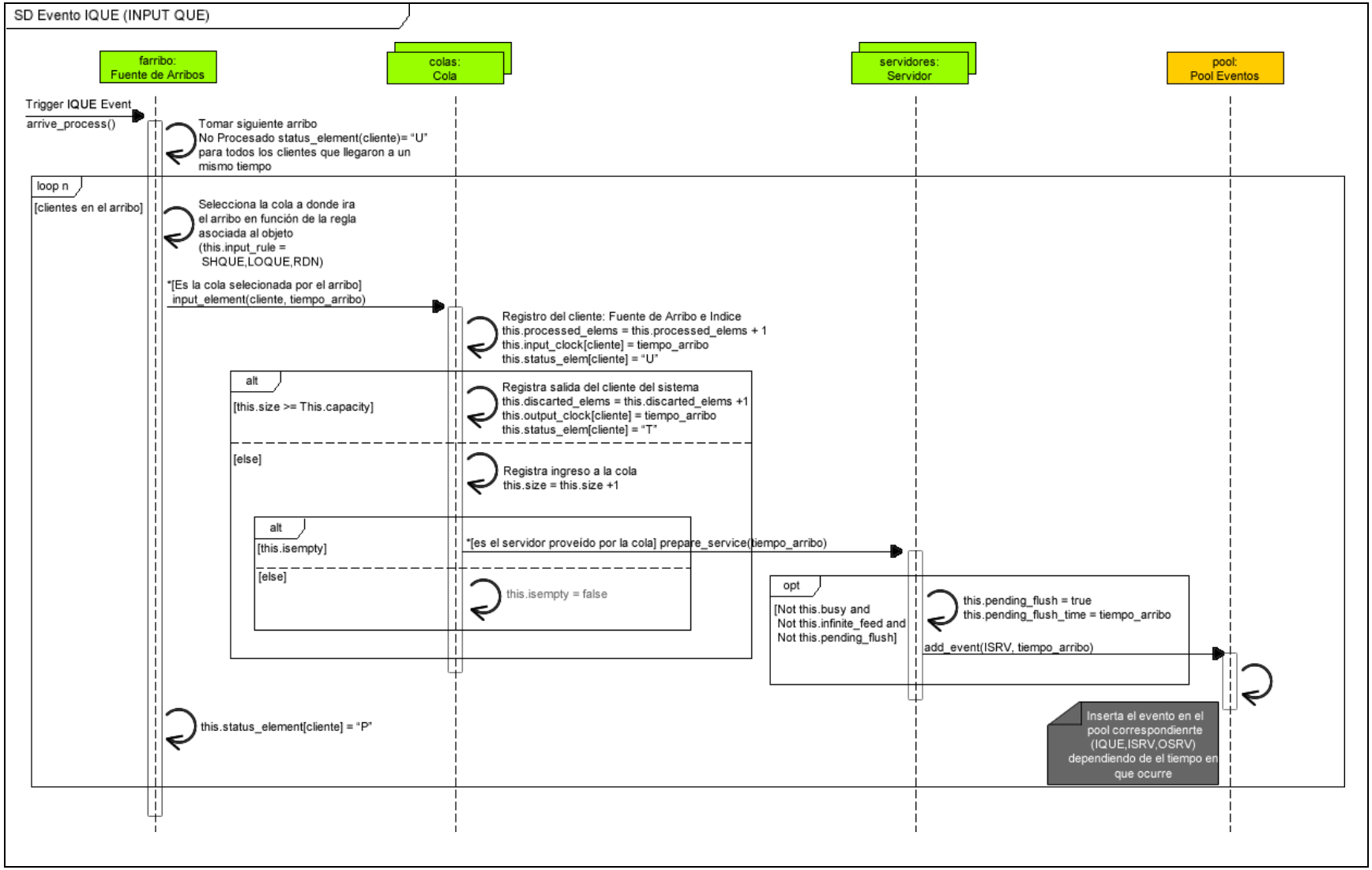


Figura 20. SD (Diagrama de Secuencia) Evento IQUE

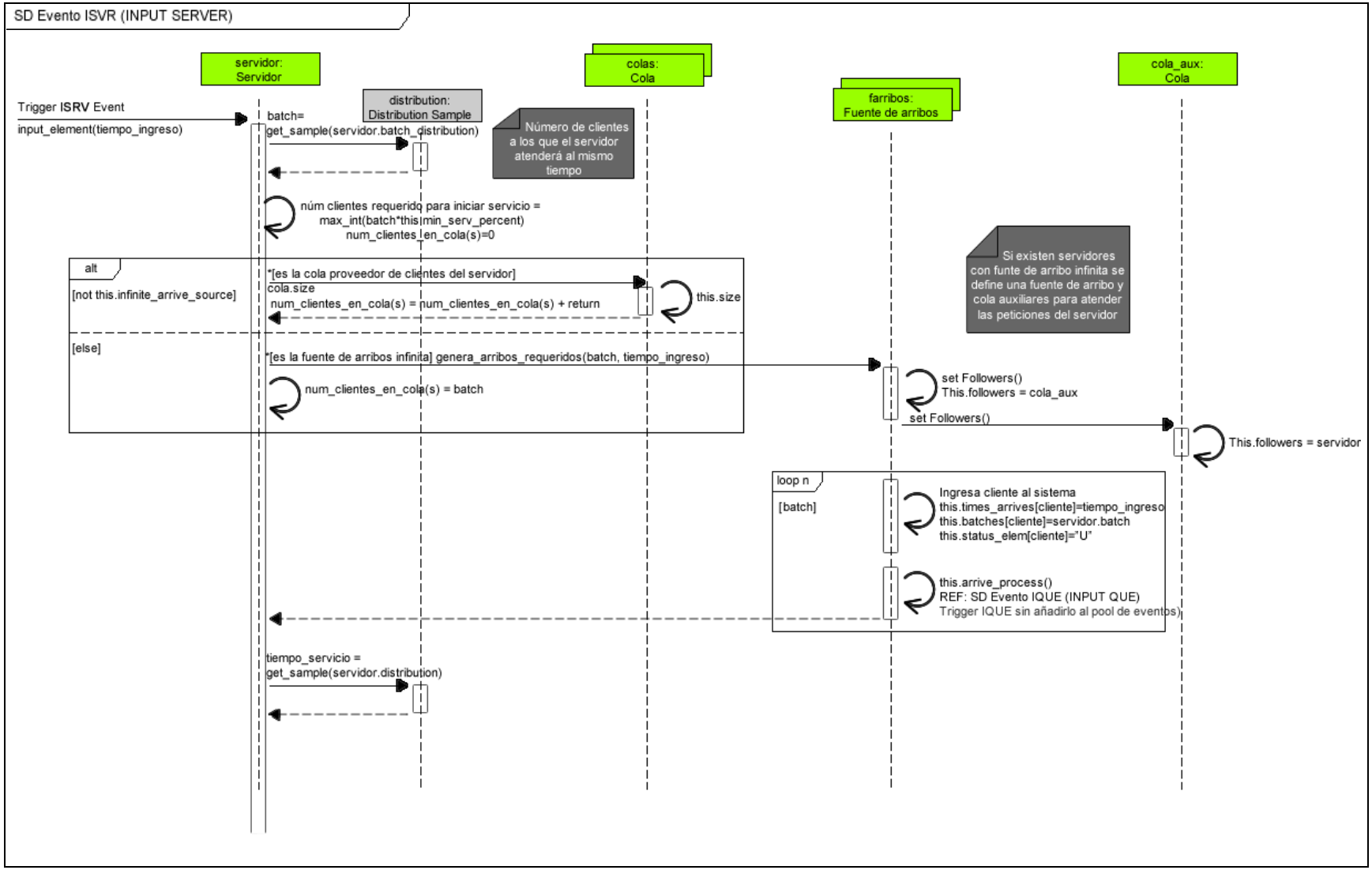


Figura 21. SD (Diagrama de Secuencia) Evento ISRV

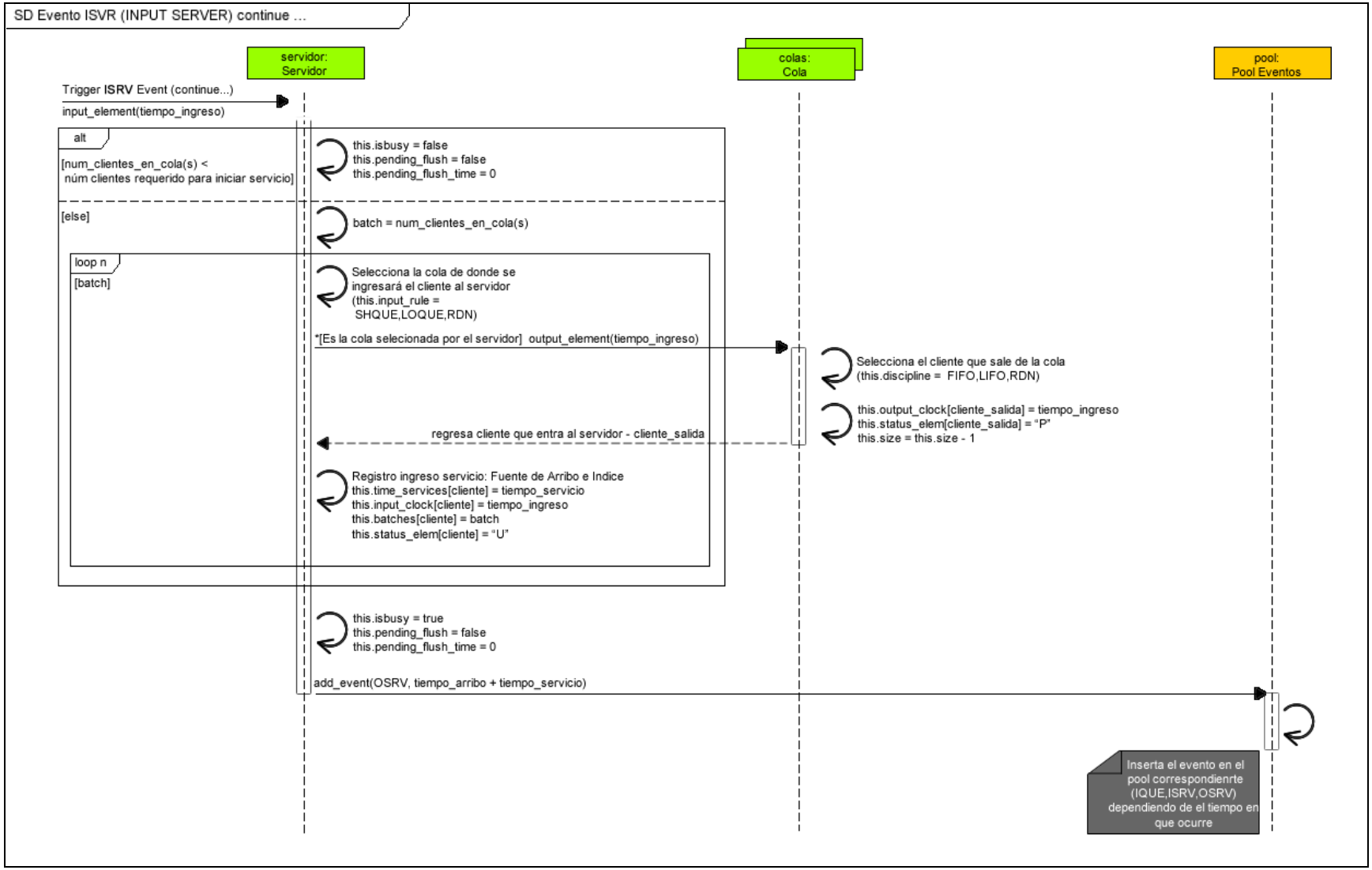


Figura 22. SD (Diagrama de Secuencia) Evento ISRV (Continue ...)

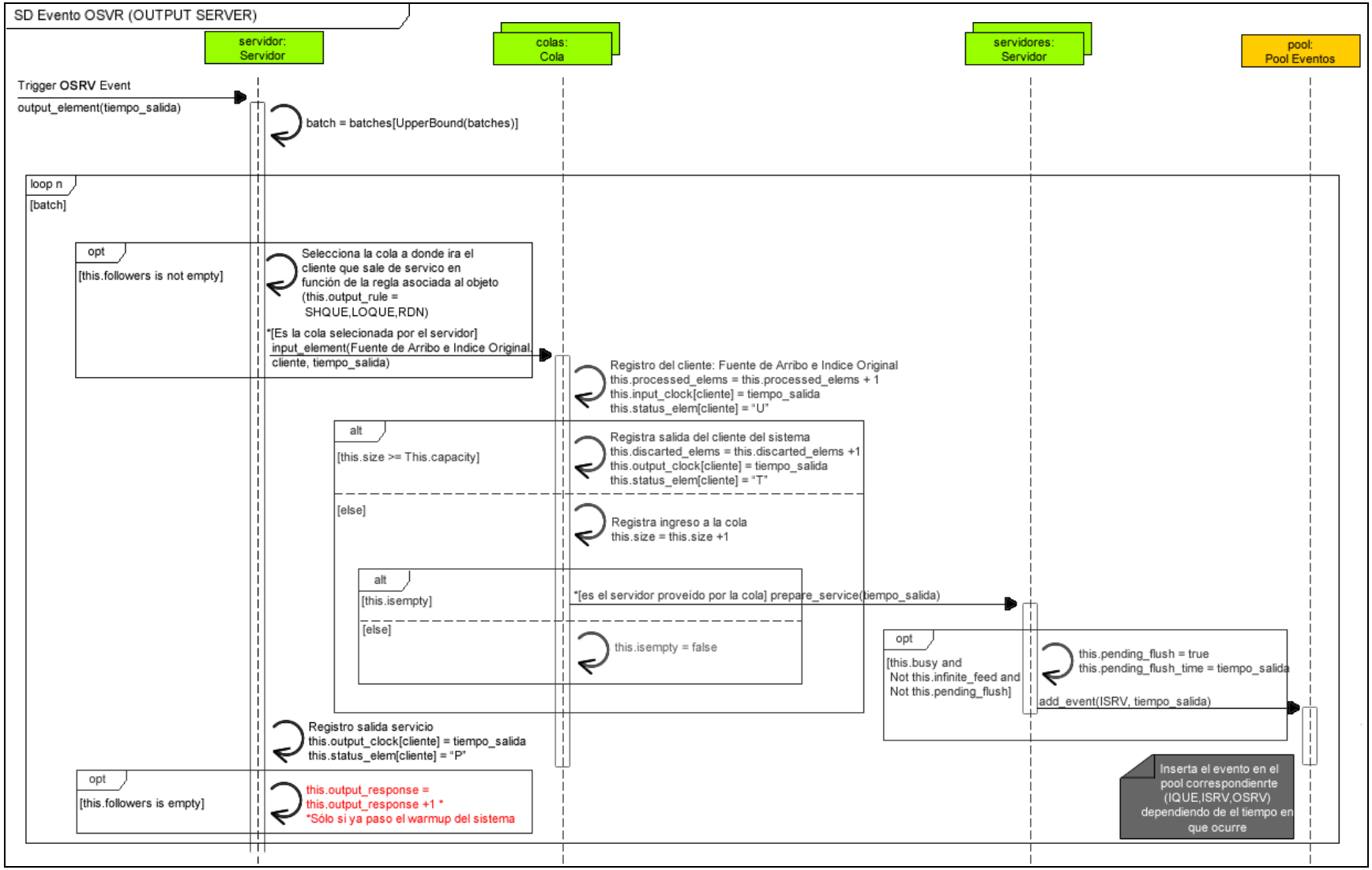


Figura 23. SD (Diagrama de Secuencia) Evento OSRV

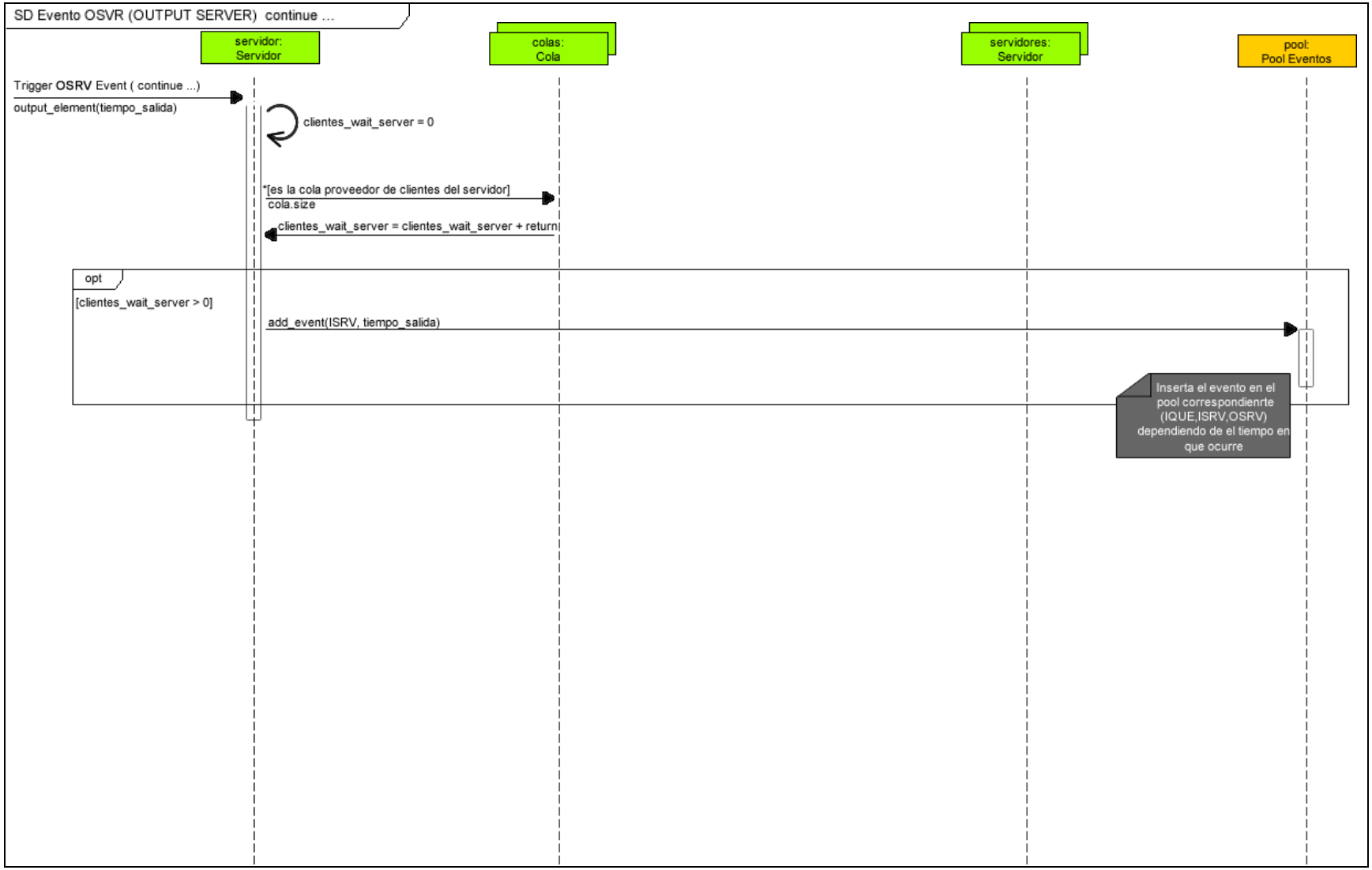


Figura 24. SD (Diagrama de Secuencia) Evento OSRV (Continue ...)

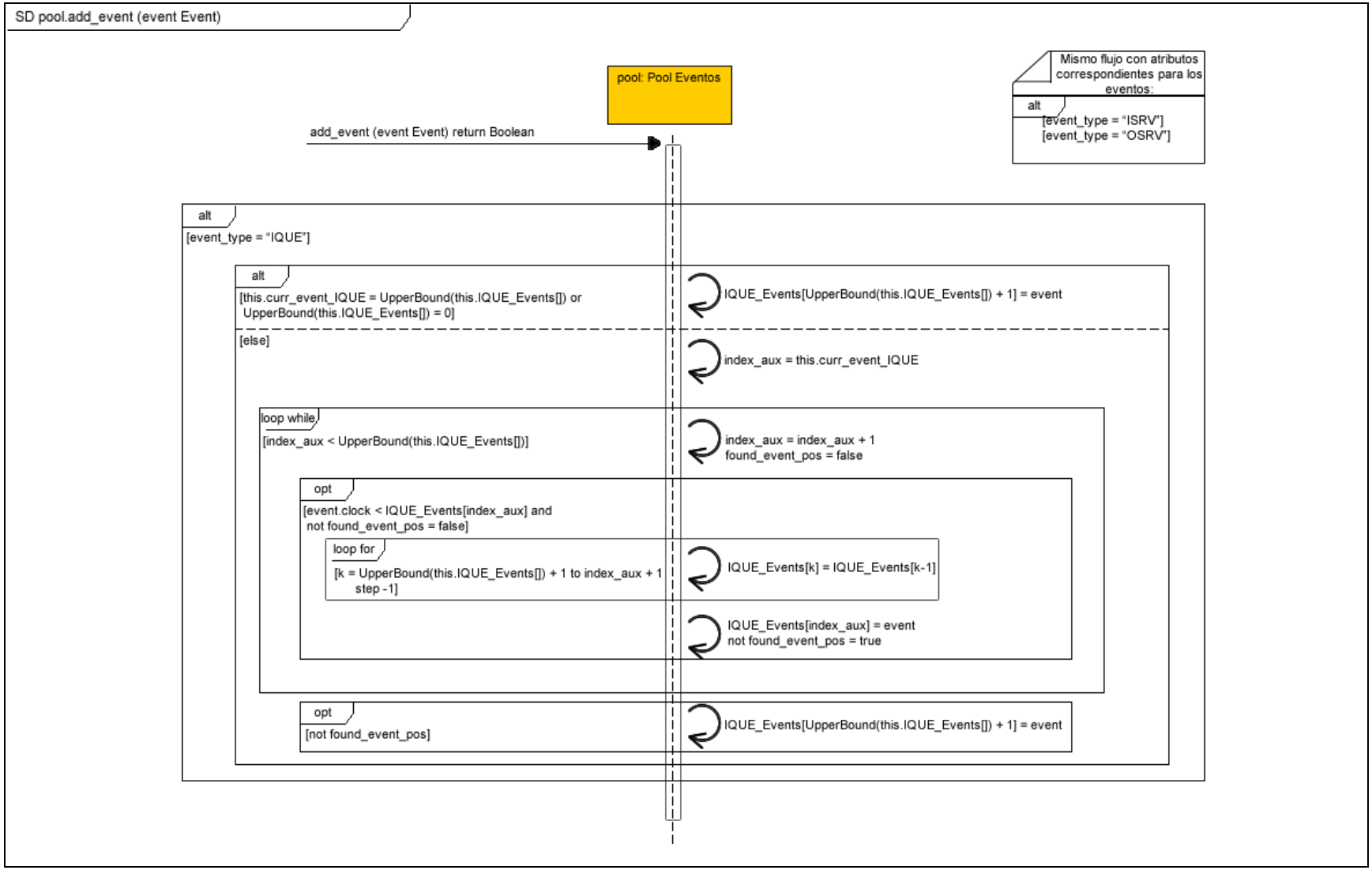


Figura 25. SD (Diagrama de Secuencia) pool.add_event (event Event)

3.3.7 Algunas Reflexiones acerca del Diseño Presentado para el Módulo de Simulación

Los beneficios que se obtiene de un diseño de software orientado a objetos como el que se ha presentado, son varios, y pueden ser referidos en la literatura al respecto [11] sin embargo, no se puede dejar de mencionar que el haber basado el diseño módulo de simulación en esta metodología derivo en:

1. Una significativa reducción del código requerido para el modelado de procesos/sistemas con todas sus consecuencias.

2. Posibilidad de tratar sistemas basados en líneas de espera y no casos particulares de los mismos. Los sistemas son modelados instanciado en memoria los objetos necesarios para modelar el sistema desde las clases: **Fuente de Arribos**, **Cola** y **Servidor**.

3. Posibilidades de reutilización futura del diseño aun si se cambiara por ejemplo el lenguaje de programación.

Para finalizar mencionaremos que el diseño de este módulo contempla también la posibilidad de variar distribuciones de tiempos entre arribo y servicio, con la utilización de la clase/componente tipo Distribución.

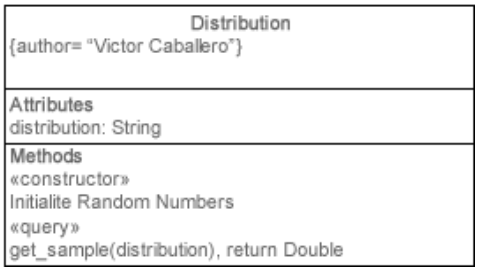


Figura 26. Diseño UML de la Clase Distribution

Esta clase pose solo un atributo *distribution*, que especifica el tipo de distribución que manejará cada objeto instanciado desde esta clase, y dos métodos:

«constructor»
Inicializa la generación de números aleatorios.

get_sample

El método get_sample obtiene muestras (independientes) de las distintas variables aleatorias, a través del método de transformación de probabilidad que puede ser consultado en la sección de anexos del este documento [pág. 151].

3.4 Módulo de Optimización

El diseño del módulo de optimización recaerá sobre la técnica/algoritmo de optimización en el que sea basado, la razón es simple, la técnica seleccionada es la que al final definirá el diseño del mismo. Como se mencionó en el capítulo I del presente trabajo, a la fecha se han implementado se ha experimentado con una gran variedad de técnicas de optimización, la flexibilidad del concepto así lo permite, sin embargo, unas de las técnicas más utilizadas han sido las heurísticas debido principalmente, al gran desarrollo que las mismas han alcanzado en las últimas décadas de la mano de las ciencias de la computación. Por último hay que señalar también, que la selección de la técnica/algoritmo de optimización, puede o no tener relación con la problemática o sistema que será tratado.

3.4.1 Técnica/Algoritmo de Optimización

En la presente propuesta se decidió utilizar como técnica de optimización una heurística basada en algoritmos evolutivos, más particularmente **algoritmos**

genéticos. A continuación se dan algunas razones del ¿por qué? de esta decisión en este trabajo:

1. Experiencias. No es difícil encontrar información acerca de buenas experiencias con el uso de la misma en el área de optimización. En muchos problemas en los que, como en el caso de estudio, el número de soluciones factibles es exponencial, el uso de técnicas heurísticas ha ayudado obtener buenas soluciones en un tiempo razonable de cómputo.

2. Resultados. Muchos de los módulos de optimización de software comercial están basados en este tipo de técnicas.

3. Didáctica. Esta técnica heurística está basada en un concepto biológico muy fácil de entender.

4. Implementación. Su implementación no es demasiado complicada en cualquier herramienta de programación.

Los algoritmos evolutivos son métodos de búsqueda adaptativa basados en el concepto biológico de evolución. La teoría de los algoritmos evolutivos encuentra sus bases en las investigaciones realizadas por John Holland en la década de 1960, quien motivado por los aspectos fundamentales de la teoría evolutiva, desarrolló un método de búsqueda adaptativa para encontrar solución a problemas de optimización “difíciles”, en los cuales los métodos numéricos tradicionales enfrentaban grandes complicaciones para su aplicación, que vale la pena mencionar, en muchas ocasiones estaban ligadas con la complejidad computacional del método en sí; este método de búsqueda adaptativa es conocido hoy en día como algoritmos genéticos y basa su funcionamiento en el siguiente esquema fundamental:

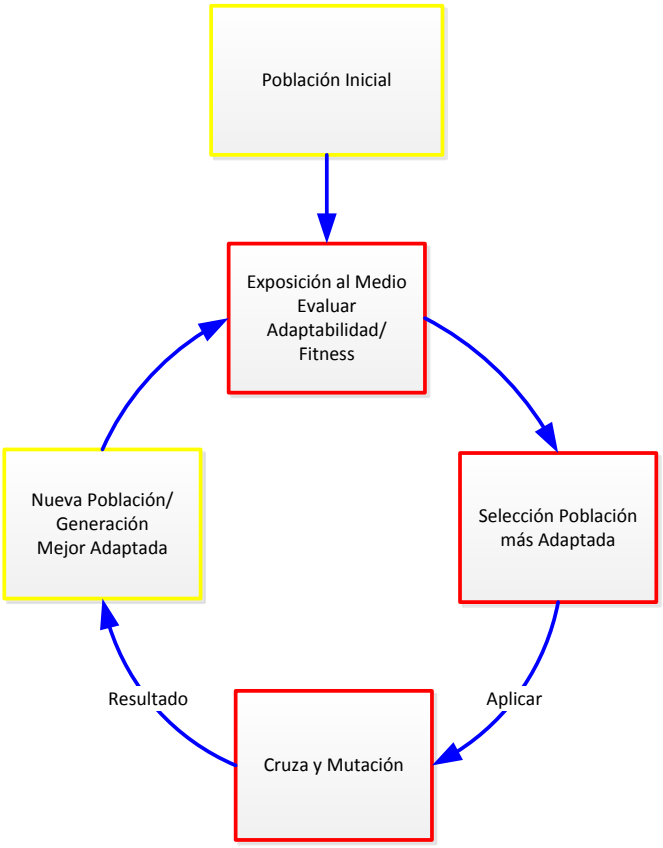


Figura 27. Esquema Fundamental de un Algoritmos Genético

Como ocurre en la evolución biológica, a la que el método trata de imitar, el proceso está orientado a lograr la trascendencia de una población en el tiempo, esto a través de la conservación de las características que mejor les permiten adaptarse a su medio ambiente. Así, el objetivo central del proceso es obtener una población/generación mejor adaptada desde otra población/generación base. La obtención de esta nueva población/generación se lleva a cabo mediante procesos de reproducción/cruza o mutación sobre la población/generación inicial, sin embargo esta reproducción o mutación no ocurre al azar, al igual que ocurre en la evolución de los seres vivos, ella está sujeta a un proceso de selección, donde la capacidad de adaptación al medio ambiente o aptitud de cada individuo, define las posibilidades de que éste pueda trasmitir sus características a nuevas generaciones, de forma que, a mayor grado de adaptación al medio, mayores

posibilidades de que las características de un individuo prosperen en las nuevas generaciones.

3.4.2 Implementación de la Técnica/Algoritmo

Respecto a la técnica de optimización seleccionada, algoritmos genéticos, existen cuatro aspectos fundamentales que deben ser tomados en cuenta en el diseño: representación de la población, función de aptitud, operadores genéticos y generación de la población inicial; a continuación se aborda la manera en que serán contemplados cada uno de ellos, estableciendo, para los dos primeros, representaciones en torno al caso de estudio.

I. Representación de la Población.

En los algoritmos genéticos una población, o generación como comúnmente es llamada en términos del algoritmo, está compuesta por un número x de individuos, y cada individuo representa una posible solución al problema que se está tratando.

En los trabajos iniciales de Holland, los individuos, o como él los llamó, cromosomas, eran representados únicamente por medio de cadenas/vectores binarios ($0,1$), donde cada entrada representaba el valor de determinado gen en el individuo en cuestión. Más adelante, esta representación fue abriéndose a consecuencia de los tipos de problemas que se planeaban enfrentar.

En el contexto del caso de estudio, el vector utilizado para representar los individuos guardará la misma configuración que el que se utilizó para representar las soluciones factibles del problema:

$$(svrs_1 , svrs_2 , svrs_3 , \ \dots , svrs_m , b_2 , b_3 , b_4 , \ \dots , \ b_m)$$

Donde:

$svrs_i$ representa el número de servidores colocados en la estación de trabajo *i,* y cumple:

$0 < svrs_i \leq Usvrs_i$

con $svrs_i$, $Usvrs_i$ *enteros positivos*

$i = 1,2,3, \ldots, m$

Donde:

b_i representa la capacidad del buffer *i,* y cumple:

$0 < b_i \leq Ub_i$

con b_i, Ub_i *enteros positivos*

$i = 2,3,4, \ldots, m.$

Cada posible valor de este vector representa un escenario o solución factible y única al problema. En otras palabras, si se quisiera ver como lo hizo John Holland, cada entrada de este vector representa el valor de una característica, gen, del individuo en cuestión.

II. Función de Aptitud/Función Objetivo (Fitness)

En los algoritmos genéticos existe un valor de aptitud (fitness) asociado a cada individuo, el cual determina su capacidad de sobrevivir, reproducirse y dejar su material genético a poblaciones subsecuentes, por lo que todo algoritmo de este tipo necesita una manera de poder calcularlo.

En torno al caso de estudio, este valor de aptitud no es otro que la *"utilidad monetaria que brindará el sistema"* y será calculada con base a la información brindada por el modelo de simulación respecto al número de ítems convertidos en producto, esto una vez que se haya ejecutado el modelo de simulación del escenario específico.

III. Operadores Genéticos

Estos operadores permiten obtener una nueva población/generación, teóricamente mejor adaptada, desde la población/generación actual, en otras palabras definen la manera en que se obtendrá una nueva población a partir de cualquier población base; entre los más utilizados en la teoría de algoritmos genéticos están: la cruza, la mutación y la selección.

- Cruza.

Este operador, análogo a la reproducción en términos biológicos, define la forma en que serán combinados las características o genes de por lo general dos individuos, para generar un nuevo individuo en la población. Existen varios tipos de cruza, uno de los primeros utilizados fue el cruce de un punto (*One-Point Crossover)*.

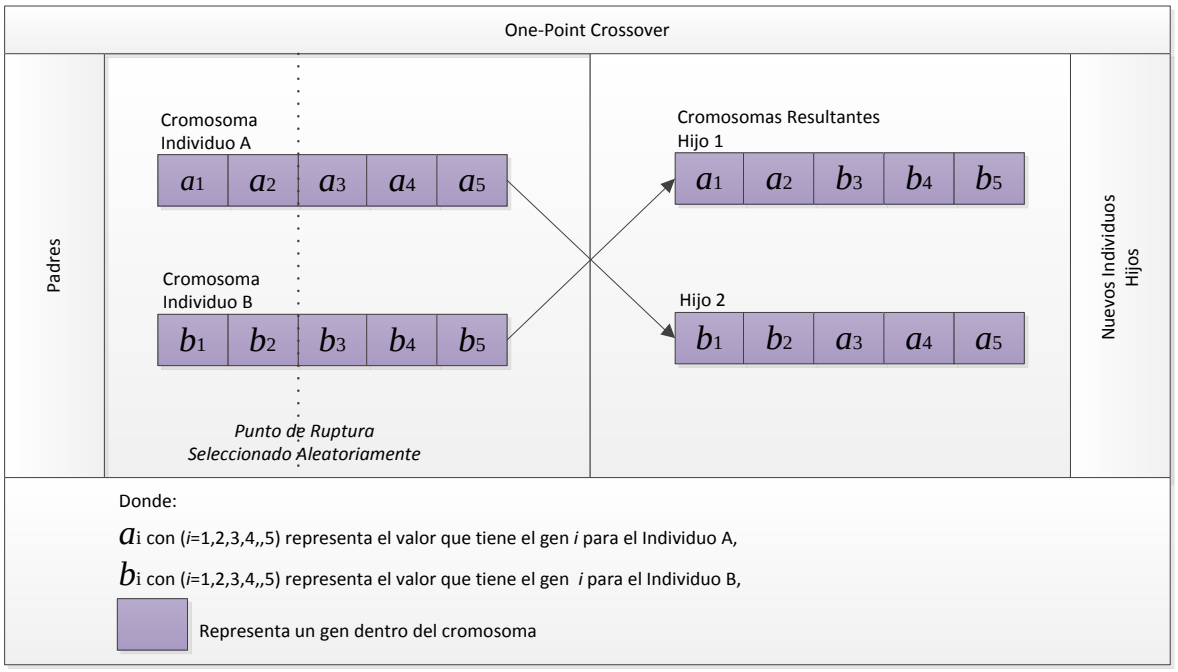


Figura 28. Cruce de un punto

En este tipo de cruza se parte de dos individuos, comúnmente llamados padres, en los que se elige aleatoriamente un "punto de ruptura" de tal manera que el material genético más allá de ese punto es intercambiado en ambos padres para crear así dos nuevos individuos (hijos), en el ejemplo mostrado el punto de ruptura seleccionado es el segundo gen.

Con el paso del tiempo y como resultado de la experimentación fueron generándose otros tipos de cruza como son el cruce de dos puntos (*Two-Point Crossover)*, que trabaja de manera similar al cruce de un punto, pero en el que en lugar de seleccionar un solo punto de ruptura se seleccionan dos; el cruce uniforme (*Uniform Crossover)*, y otros que fueron desarrollados para atacar problemáticas más específicas como: el cruce por emparejamiento parcial *(Partially-Matched Crossover (PMX))*, utilizado en tratamiento de problemas con individuos representados por cromosomas no binarios, el cruce estratégico al borde *(Strategic Edge Crossover (SEX))* propuesto inicialmente para el tratamiento del problema del agente viajero (TSP), etc.

En el diseño del módulo de optimización se utilizará únicamente cruce uniforme, tipo de cruza que, parte también de dos individuos (padres), de los que a partir de un proceso de mezcla de material genético, que se explica a continuación, se obtiene un nuevo individuo resultante (hijo).

En el cruce uniforme el valor de cada gen del nuevo individuo (hijo) es tomado del valor que tiene ese mismo gen en alguno de los cromosomas padres, la elección del padre que contribuirá con el valor del gen es llevada a cabo al azar, de esta manera el proceso es comenzado en el primer gen de donde se elige un padre al azar para que contribuya con el valor del mismo en el nuevo individuo (hijo), acto seguido es realizado el mismo procedimiento para el segundo gen, y así el proceso continuará sucesivamente hasta haber asignado un valor a cada uno de los gen del nuevo individuo como es mostrado en el siguiente ejemplo [Figura 29. Cruce Uniforme], en el que:

El individuos X y el individuo Y fungen como padres para la realización de la cruza, como primer paso se selecciona al azar el Individuo X para que contribuya con el valor de su primer gen x_1 al nuevo individuo, el segundo gen del nuevo individuo se selecciona al azar también, entre el valor del segundo gen en el individuo X y su valor en el Individuo Y, en el ejemplo el padre seleccionado es el

individuo X quien aporta el valor x_2, y así sucesivamente hasta llegar al *n-ésimo* gen cuyo valor y_n es aportado aleatoriamente por el individuo Y.

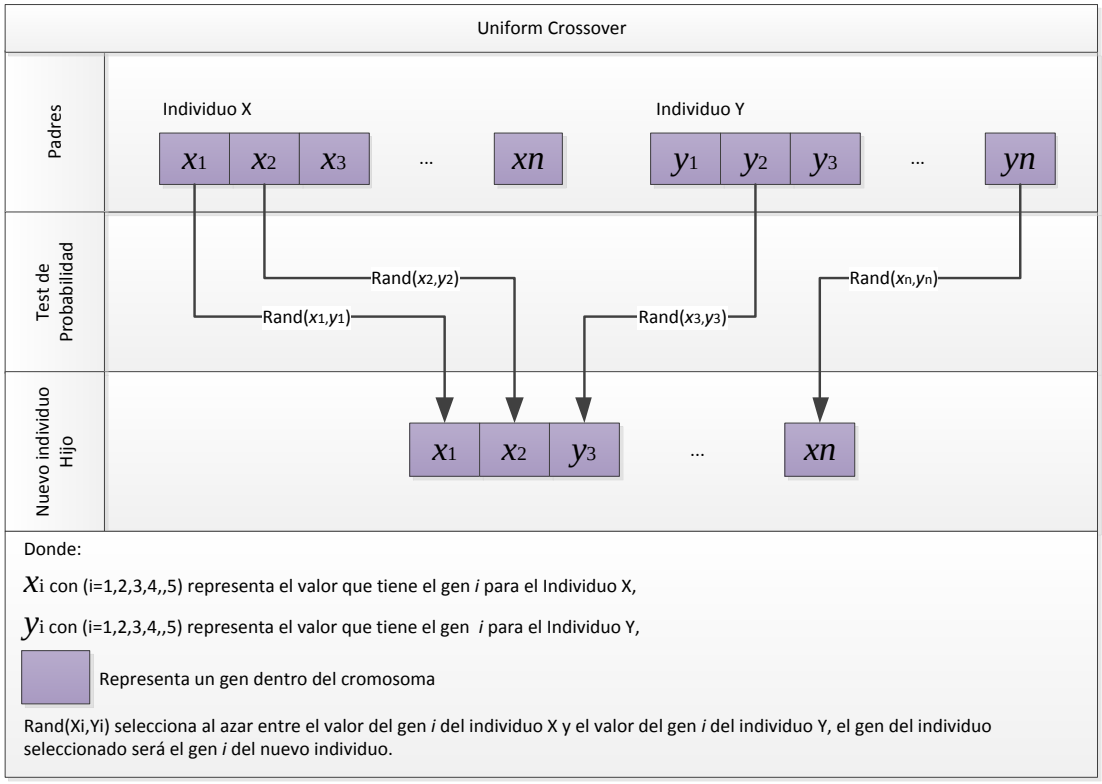


Figura 29. Cruce Uniforme

- Mutación.

El segundo operador fundamental de los algoritmos genéticos, usado para garantizar la diversidad genética de generación en generación, es la mutación, el cual consiste en perturbar, con cierta probabilidad, el valor de una característica o gen en un individuo.

El diseño del módulo de optimización contempla implementar este operador, previa selección del individuo, a través de dos procesos aleatorios: el primero

consiste en decidir, con respecto a una probabilidad específica (*Probabilidad de Mutación*), si el gen seleccionado será o no objeto de mutación, mientras que el segundo variará el valor del gen de una manera aleatoria en ±1 (más menos uno) solo cuando el primer proceso haya decidido realizar mutación en el mismo.

- Selección.

Se puede definir como el proceso mediante el cual son elegidos los individuos susceptibles de heredar su material genético a nuevas generaciones. En nuestro diseño se contempla llevar a cabo tres etapas de selección, y como resultado de las mismas, y de la aplicación de los operadores genéticos ya mencionados, se obtendrán las nuevas generaciones poblacionales en el algoritmo.

I. La primera etapa consiste en seleccionar un porcentaje específico de individuos de la población que pasará su información o material genético intacto a la siguiente generación poblacional. Hay que mencionar que estos individuos no son tomados al azar, ya que solo los individuos mejor adaptados tendrán esta oportunidad, es decir, los que tengan los mejores valores de la función objetivo o aptitud.

II. La segunda etapa es llevada a cabo para la aplicación del operador genético conocido como *cruza,* en ella la selección de los individuos será realizada mediante un procedimiento conocido como ruleta, en el que la probabilidad de que un individuo sea seleccionado es directamente proporcional a su aptitud o capacidad de adaptación (valor obtenido para la función objetivo o medida de efectividad).

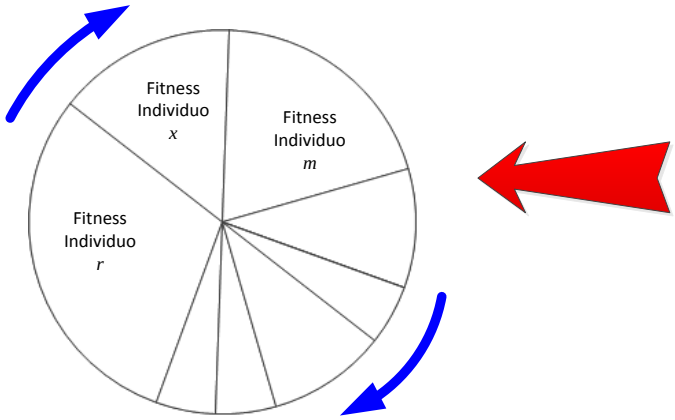


Figura 30. Ruleta *"Mayor aptitud, mayor probabilidad de ser elegido"*

Este importante señalar que este proceso de selección intenta "imitar" el proceso de selección natural, donde los individuos con mejor capacidad de adaptación al medio son los que tienen mayores probabilidades de reproducirse y por tanto heredar su material genético a posteriores generaciones.

III. La última etapa será realizada para aplicar el operador genético conocido como mutación, y en ella la selección de individuos será realizada totalmente al azar, la razón: no eliminar demasiado rápido la diversidad del material genético obtenido en la población inicial. Si elimina muy rápido la diversidad del material genético a través de las poblaciones aumenta, también rápidamente, la posibilidad de caer en óptimos locales.

IV. Generación de la Población Inicial

Todo algoritmo genético parte de una población/generación inicial, por lo que es un requisito generarla para la aplicación del algoritmo.

Existen diferentes maneras de llevar acabo la generación de la población inicial, desde utilizar alguna técnica especial, hasta hacerlo de manera aleatoria. En nuestro caso se determinó hacerlo de manera aleatoria, aunque hay que mencionar que la utilización de una técnica especial debe ser muy bien evaluada ya que, aunque por un lado podría allanar el camino al algoritmo genético haciendo que éste partiera de una población inicial "mejor adaptada", también se podría incrementar el riesgo de caer en un óptimo local.

3.4.3 Parámetros Asociados con la Aplicación de la Técnica/Algoritmo de Optimización

Del análisis de aspectos clave relacionados con la implementación de la técnica es posible establecer los parámetros de ejecución de la misma:

1. *Tamaño de la Población*. Número de individuos que conformarán la población.

2. *% Herencia Individuos Mejor Adaptados*. Porcentaje específico de individuos de la población que pasarán a la siguiente generación poblacional con su material genético intacto.

3. *% Individuos a Obtener por Cruza*. Porcentaje de individuos que serán obtenidos mediante cruza de una generación poblacional a otra.

4. *% Individuos a Obtener por Mutación*. Porcentaje de individuos que serán obtenidos mediante mutación de una generación poblacional a otra.

5. *Probabilidad de Mutación*. Probabilidad con la que un individuo puede o no mutar su información genética de generación en generación.

Estos parámetros son de gran importancia ya que están estrechamente ligados al comportamiento del algoritmo en la búsqueda de solución.

3.4.4 MODELO FUNCIONAL – DISEÑO MÓDULO OPTIMIZACIÓN.

A continuación se presenta el diseño básico sobre el que trabajará el módulo de optimización del software SO basado en algoritmos genéticos (AGs).

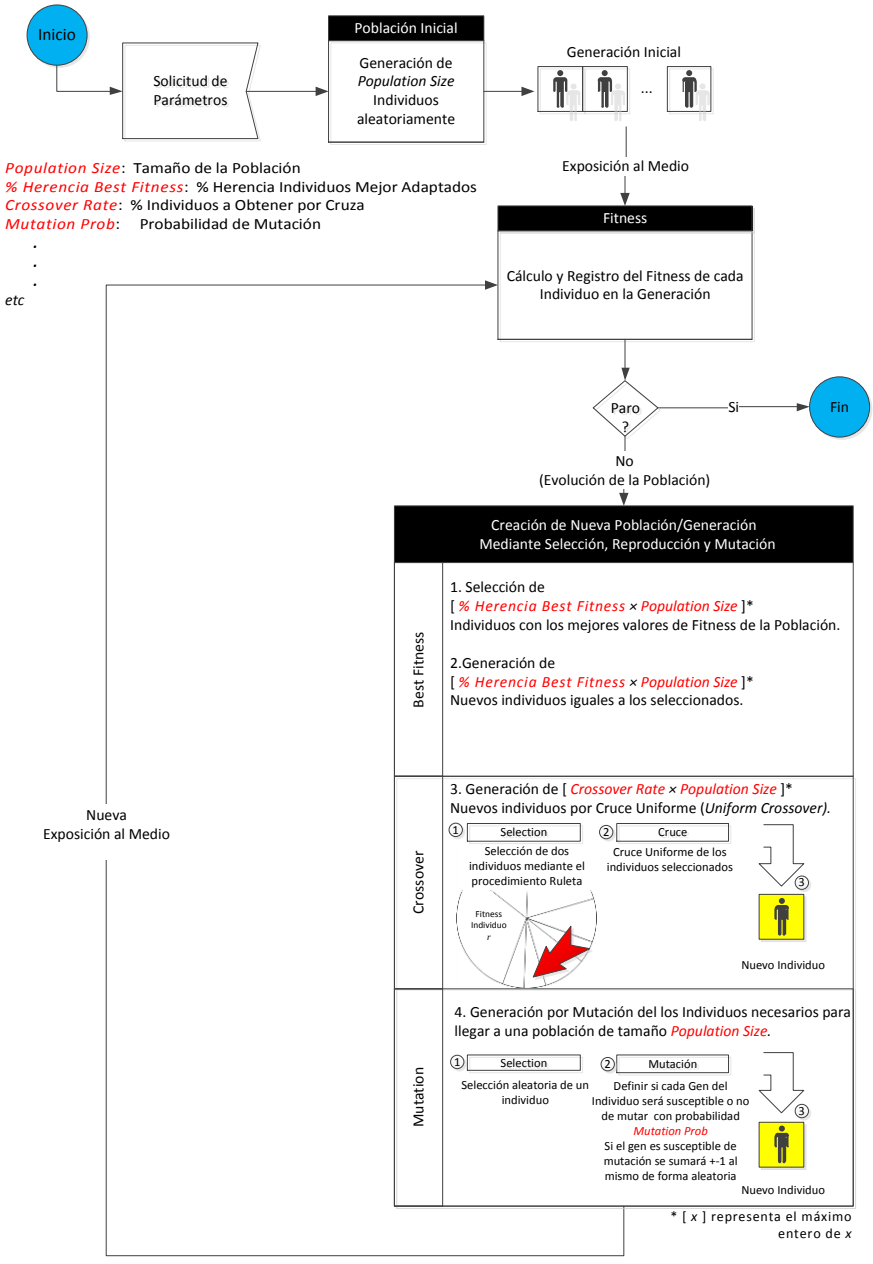


Figura 31. Diseño Módulo de Optimización (basado en AGs)

MODELO FUNCIONAL MÓDULO OPTIMIZACIÓN

El diseño que guarda el módulo de optimización no es otro que el de la técnica en que se basa. A continuación se muestra el detalle del diseño del modelo funcional por medio pseudocódigo.

PROCEDURE OPT

INPUT

Population Size - Tamaño de la Población,
% Herencia Best Fitness- % Herencia Individuos Mejor Adaptados
Cross.over Rate - % Individuos a Obtener por Cruza,
Mutation Prob - Probabilidad de Mutación,

BEGIN

- Generación aleatoria de [*Population Size*] individuos

- Formar un conjunto con los individuos generados

- Etiquetar el conjunto como *Generación 1*

- N= 1

WHILE (no se ordene parar)

- Obtener y registrar el fitness, función objetivo o medida de efectividad de cada uno de los individuos pertenecientes a la *Generación N*

- Seleccionar los [*% Herencia Best Fitness × Population Size*]* individuos con los mejores valores de Fitness en la *Generación N*

- Generar [*% Herencia Best Fitness × Population Size*]* nuevos individuos idénticos a la selección anterior

- Generar una estructura de selección aleatoria, con todos los individuos pertenecientes a la *Generación N*, en la que la probabilidad de selección de cada individuo sea proporcional a su fitness

- Etiquetar la estructura como *Ruleta*

FOR $i = 1$ **TO** [*Crossover Rate* × *Population Size*]*

- Seleccionar aleatoriamente dos individuos desde la estructura *Ruleta*

- Generar un Nuevo Individuo por el método de cruce uniforme con los dos individuos seleccionados

NEXT

FOR $i = 1$ **TO** *Population Size* - [(*% Herencia Best Fitness* × *Population Size* + *Crossover Rate*) × *Population Size*]*

- Seleccionar aleatoriamente un individuo en la *Generación N*

FOR $j = 1$ **TO** número de genes presentes en el individuo

IF (gen *j* es sujeto de mutación acorde a *Mutation Prob*) **THEN**

- Sumar aleatoriamente $\pm$ 1 al valor del gen, y asignar este valor al gen correspondiente del nuevo individuo

END IF

NEXT

NEXT

- $N = N + 1$

- Formar un nuevo conjunto con <u>todos</u> los individuos nuevos generados.

- ETIQUETAR el conjunto como *Generación N*.

LOOP WHILE

END

---

* [*x*] representa mayor entero de *x*

Figura 32. Pseudocódigo Diseño Módulo de Optimización (basado en AGs)

El modelo funcional presentado permite que en el diseño del software de optimización el parámetro "*% Individuos a Obtener por Mutación*" pueda ser eliminado, lo anterior debido a que el número de individuos que serán obtenidos vía mutación de una generación a otra puede ser calculado de la siguiente manera:

Número de individuos que serán obtenidos vía mutación de una generación a otra =

Tamaño de la Población

- [(*% Herencia Individuos Mejor Adaptados* + % Individuos a Obtener por Cruza) × *Tamaño de la población*]*

* [x] representa mayor entero de x

3.5 Modelo Funcional - Integración del Software SO

Una vez establecidos los diseños que soportan a los módulos de simulación y de optimización, habrá que ver la manera en que estos trabajan bajo el concepto SO mostrado en el capítulo I [Figura 4], y en este sentido, la siguiente figura[9] muestra a mayor detalle cómo es que se integran estos dos módulos vía interfaz de comunicación en un software SO. Es importante mencionar que bajo la propuesta de diseño que se está presentando, el elemento angular en esta interfaz entre el módulo de optimización y el módulo de simulación es el componente tipo **scenario** [Tabla 19], medio por el cual el software SO representará los sistemas o escenarios de sistema, y por el cual también, el módulo de optimización requerirá la simulación de los mismos utilizando el modelo funcional presentado en la pág.79.

[9] Elaboración propia con base a [27]

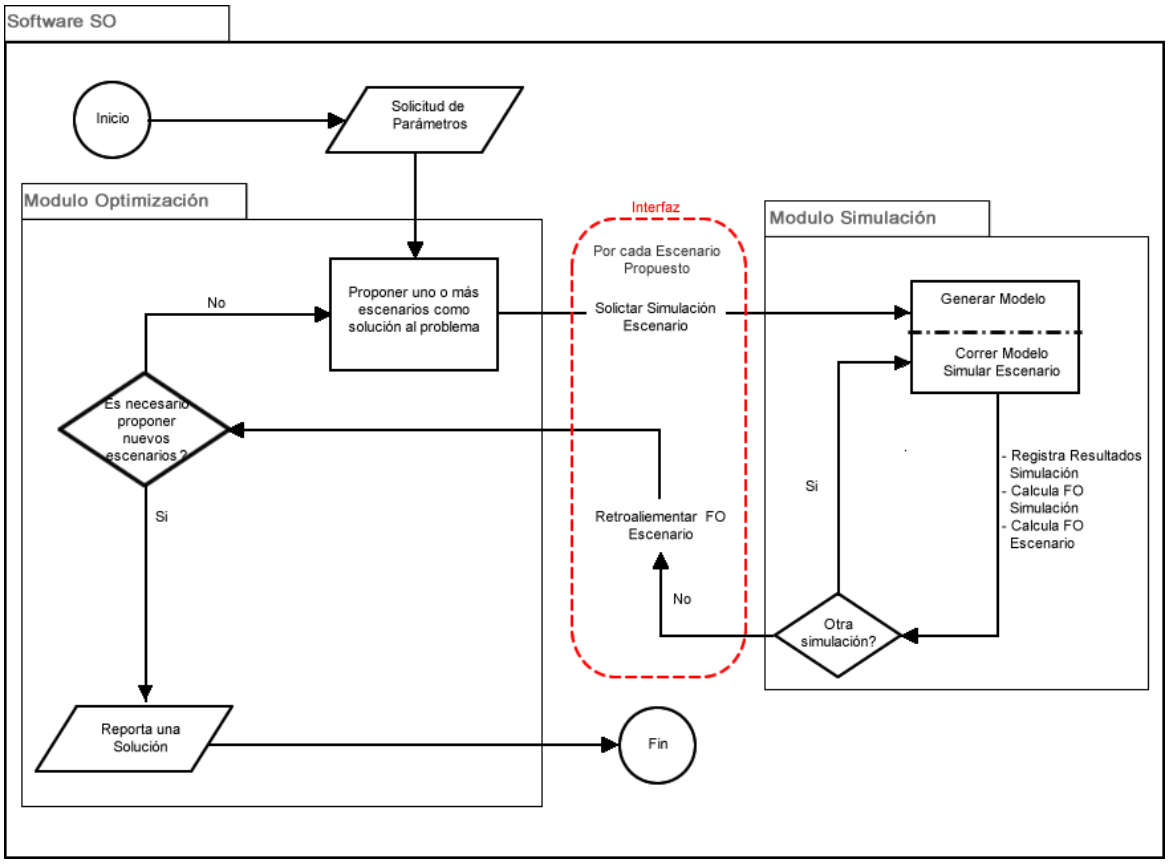


Figura 33. Integración Módulo Optimización y Módulo Simulación.

DETALLE PROCESO MODELO FUNCIONAL SOFTWARE SO

I. Solicitud de Parámetros Iniciales – Software SO.

Estos parámetros definirán la manera en que serán ejecutados tanto el módulo de optimización como el de simulación:

Parámetros Módulo Simulación.

1. Tiempo que durarán las simulaciones.

2. Número de repeticiones para obtener el valor de la F.O./Medida de Efectividad.

3. Semilla para la generación de las muestras *Uniforme (0,1)*.

Parámetros Módulo Optimización.
(Estos parámetros son específicos de la técnica de optimización utilizada)

1. Tamaño de la Población.

2. % Herencia Individuos Mejor Adaptados.

3. % Individuos a Obtener por Cruza.

4. Probabilidad de Mutación.

II. Proponer uno o más escenarios como solución al problema – Módulo Optimización.

III. Por cada escenario propuesto:

1. Solicitar la simulación – Interfaz de comunicación.

2. Generar Modelo de Simulación – Módulo Simulación.

3. Ejecutar/Correr el modelo de simulación tantas veces como defina el parámetro inicial recibido – Módulo Simulación.

4. Registrar los resultados arrojados por la simulación (base para el cálculo de la medida de efectividad), así como valores encontrados para la función objetivo.

5. Con base a los resultados anteriores, obtener el valor final de la función objetivo que será reportado para el escenario en cuestión (medida de efectividad).

6. Retroalimentar al algoritmo de optimización con la función objetivo / medida de efectividad del escenario – Interfaz de comunicación.

IV. Decidir si es necesario proponer nuevos escenarios en la búsqueda del óptimo (**Condición de Paro** - Módulo Optimización).

Esta decisión podrá ser tomada por el algoritmo de optimización con base a su propia definición, o bien conforme a los parámetros iniciales, para entonces:

1. Regresar al paso II, si se decide proponer nuevos escenarios.

2. Reportar el escenario estimado como solución al problema conjuntamente con el valor de la medida de efectividad asociada.

Los puntos III.4, III.5, III.6 y IV abordan dos aspectos clave dentro del diseño de un software SO, que además no se han tocado hasta este momento:

- obtención de la medida de efectividad de un escenario;
- condición de paro.

OBTENCIÓN DE LA MEDIDA DE EFECTIVIDAD DE UN ESCENARIO

En el capítulo I se citó que, cuando la SO enfrenta sistemas en los que una o muchas de sus variables se comportan de manera aleatoria, es necesario ejecutar varias veces la simulación de un escenario para poder obtener el valor “aproximado” de una medida de efectividad [Retos de la SO p.26], ¿cuántas veces?, pues bien, ésta es otra variable que sin duda debe también ser tomada

en cuenta en el diseño de un software SO, y en este sentido podrá ser implementada de dos formas diferentes:

1. La primera, que es como será considerada en el presente diseño, es a través de una variable fija de entrada, es decir todos los escenarios serán ejecutados un número fijo de veces antes de obtener el valor de la función objetivo o medida de efectividad, y este número fijo será pasado al software SO en forma de parámetro.

2. Y la segunda a través de un procedimiento dinámico dentro del software. Es decir que el mismo software decida cuantas veces será ejecutado el modelo de simulación de un escenario antes de tomar el valor de la función objetivo. Solo por poner un ejemplo, se podría pedir que el software no pare de realizar ejercicios de simulación de un escenario específico hasta que la varianza de los valores obtenidos para la función objetivo estén por abajo de un cierto límite.

Suponiendo que se han ejecutado ya todas las simulaciones requeridas para obtener la medida de efectividad, queda pendiente resolver la interrogante de ¿cómo será calculada?, pues bien el método más frecuentemente utilizado es mediante un promedio simple de los valores obtenidos para la media de efectividad en todas las simulaciones realizadas al escenario; aunque se debe también mencionar que esta no es la única forma, solo por mencionar algunas otras que se podrían considerar dependiendo de la problemática que se esté tratando, tenemos:

- Promediar las medias de efectividad obtenidas en todos los experimentos excluyendo el valor máximo y mínimo.

- Tomar únicamente el valor mínimo de la medida de efectividad en todas las simulaciones.

- Tomar únicamente el valor máximo de la medida de efectividad en todas las simulaciones.

En el nuestro caso se utilizó el promedio simple de los valores obtenidos para la media de efectividad en todas las simulaciones realizadas al escenario, hay que mencionar que este método es de los que requieren correr menos simulaciones del modelo.

CONDICIÓN DE PARO

El carácter estocástico que presentan la mayoría de las problemáticas tratadas mediante SO, hace que los esfuerzos de esta técnica estén más enfocados a encontrar buenas soluciones que a encontrar la configuración óptima para el sistema. Si a esto le sumamos el hecho de que para obtener el valor de una medida de desempeño/efectividad, es necesario correr múltiples veces un modelo de simulación, entendemos la importancia que tienen las condiciones de paro dentro del diseño del software.

En casi cualquier diseño de software SO se podrán implementar dos tipos de condiciones de paro:

1. **Condiciones que tienen que ver con el algoritmo de optimización en cuestión**. Están estrechamente ligadas al algoritmo de optimización utilizado en el diseño, por poner un ejemplo, si el software está basado en Algoritmos Evolutivos, algunas posibles condiciones de paro serían:

    - *"Número Máximo de Generaciones Poblaciones".*
    - *"Número Máximo de Individuos Analizados".*

2. **Condiciones que tienen ver con la calidad de los resultados obtenidos en la medida de desempeño**. Están más estrechamente ligados al valor de la

función objetivo en las distintas iteraciones del algoritmo, un ejemplo de éstas es:

- *"Número de iteraciones del algoritmo sin un aumento significativo en el valor de la función objetivo".*

Es importante señalar que, una vez que se han definido las condiciones de paro en un software SO, éstas deberán ser implementadas como parámetros de entrada dentro del diseño. En el presente diseño de software se decidió incorporar tres *condiciones de paro*; la primera está ligada al algoritmo de optimización que se utiliza, y las dos últimas están más bien enfocadas a la calidad del resultado en la función objetivo.

1. **Max Genearions.** Número Máximo de Generaciones.

2. **Max Gen Wout Improv**. Número Máximo de Iteraciones/Generaciones sin Ganancia en La función Objetivo.

3. **Min Impv Percent**. Mínimo Porcentaje de Mejora en el Valor de la Función Objetivo.

IV. ALGUNOS RESULTADOS

En el presente capítulo se muestran algunos resultados obtenidos al atacar una variante particular del caso de estudio presentado en el capítulo II con la utilización de un software implementado a partir del diseño documentado en el capítulo anterior.

Es importante recordar que el caso de estudio aborda la problemática general que supone la implantación de un sistema de manufactura con m estaciones de trabajo y $m-1$ buffers, y acorde con esto, este apartado muestra **solo algunos** de los **resultados** que pueden ser obtenidos con el uso del diseño, ya que éste tiene la posibilidad de trabajar con cualquier variante del sistema expuesto.

Hay que mencionar también que la variante particular del caso de estudio que se va a tratar en este capítulo no fue llevada a cabo al azar; debemos mencionar que se tenía la necesidad de mostrar los resultados que se podían obtener con la aplicación del diseño de software, pero no en una variante cualquiera del sistema, por poner un ejemplo, si se hubiese elegido una variante muy sencilla en la que la cardinalidad del espacio de resultados, o número de escenarios factibles, fuera

demasiado pequeño; tal vez la SO no se hubiera presentado como una buena alternativa para atacar el problema, es más tal vez no se justificaría el uso de la misma por encima de la simulación; era necesario entonces que en la selección de la variante particular se considerara por lo menos la cardinalidad del espacio de resultados, y en este sentido, la variante seleccionada debía tener un espacio de resultados "grande", ¿qué tan grande? , pues bien, tanto como para descartar a la simulación como herramienta inmediata de solución al problema por el número de escenarios que sería necesario analizar en la búsqueda del óptimo.

Adicionalmente se tenía el problema de cómo validar los resultados obtenidos por el software; en un principio se pensó en utilizar un software comercial, sin embargo no fue posible conseguir una versión *estudiantil*, o *demo*, que permitiera el modelado de una variante del caso de estudio en la que la cardinalidad del espacio de resultados justificara el uso de SO como medio de solución al problema.

Sin entrar en mayores detalles se decidió optar por una variante de la existen resultados documentados[10], además de que para la misma, se tiene también la posibilidad de, mediante un razonamiento no muy complicado, intuir por dónde debería estar ubicada la solución al problema.

4.1 Variante Particular del Caso de Estudio

La variante que se seleccionó para mostrar los resultados del software SO contempla los siguientes supuestos sobre la problemática planteada en el capítulo II del presente trabajo:

[10] Particularmente en el trabajo *"SIMULATION-BASED OPTIMIZATION"* de los autores Averill M. Law y Michael G. McComas (Averill M. Law & Associates Inc.), presentado en 2002 en la Conferencia de Simulación de Invierno (*Winter Simulation Conference*) [20]

1. **Estaciones de Servicio**.

Se considerarán solo 4 estaciones de servicio en el sistema, en otras palabras la variable $m=4$.

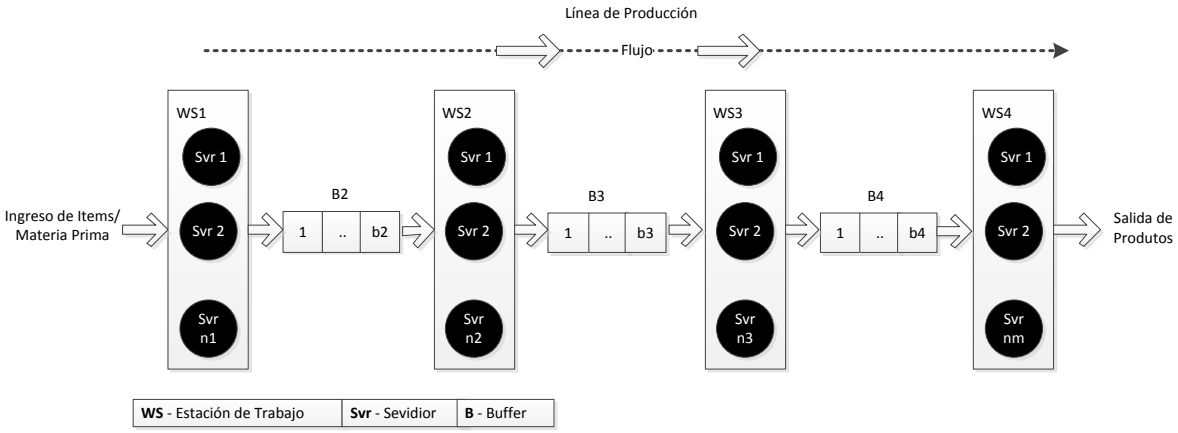


Figura 34. Variante Particular Caso de Estudio - 4 Estaciones de Trabajo

2. **Distribución de los tiempos de servicio por estación de trabajo.**

La siguiente tabla muestra las distribuciones de los tiempos de servicio que serán utilizadas:

Estación de Servicio	Distribución Tiempo de Servicio*
1	**Exponencial** / media de servicio **3** por unidad de tiempo.
2	**Exponencial** / media de servicio **2** por unidad de tiempo.
3	**Exponencial** / media de servicio **5** por unidad de tiempo.
4	**Exponencial** / media de servicio **4** por unidad de tiempo.

*Aplicable a todos los servidores de la Estación de Servicio.

Tabla 21. Distribución Tiempos de Servicio - Variante Particular

3. **Restricciones Operativas.**

Número máximo de servidores que se podrán instalar en cada una de las estaciones de trabajo $Usvrs_i$ $(i=1,2,3,4)$.

$0 < svrs_i \leq 3$

$(Usvrs_i = 3 \text{ para } i = 1,2,3,4)$

Capacidad máxima para cada uno de los Buffers Ub_i $(i=2,3,4)$.

$0 < b_i \leq 10$

$(Ub_I = 10 \text{ para } i = 2, \ldots, 10)$

Hay que mencionar que estas restricciones operativas, cuando se está abordando una problemática real, están íntimamente ligadas a las limitaciones técnicas y económicas que se presentan en cualquier proyecto, por ejemplo el espacio disponible para instalar el equipo y el presupuesto tope de inversión en maquinaria.

Para finalizar es importante señalar que el resto de los supuestos y condiciones que definen al sistema permanecerán sin cambios, es decir tal como se definieron originalmente en el caso de estudio.

4.2 Espacio de Resultados y Cardinalidad

Una vez definidos los supuestos específicos de la variante particular del caso de estudio que vamos a trabajar, es posible, retomando el análisis que hicimos del caso general, representar cualquier solución factible al problema con el uso del siguiente vector:

$$(svrs_1 , svrs_2 , svrs_3 , svrs_4 , b_2 , b_3 , b_4)$$

Donde:

$svrs_i$ representa el número de servidores colocados en la estación de trabajo i, y cumple:

$$0 < svrs_i \leq 3$$

$$(Usvrs_i = 3 \text{ para } i = 1,2,3,4)$$

Donde:

b_i representa la capacidad del buffer i, y cumple:

$$0 < b_i \leq 10$$

$$(Ub_i = 10 \text{ para } i = 2,3,4)$$

Por lo que el espacio de resultados tiene una cardinalidad de:

$$Usvrs_1 \times Usvrs_2 \times Usvrs_3 \times Usvrs_4 \times Ub_2 \times Ub_3 \times Ub_4 = 3 \times 3 \times 3 \times 3 \times 10 \times 10 \times 10$$

Es decir ***ochenta y un mil*** escenarios factibles, como se ve es un número bastante alto para un problema que a simple vista no parecería ser muy complejo, y un número también algo grande si pretendiéramos atacar el problema solo con simulación.

4.3 Función Objetivo / Medida de Efectividad

Como se vio anteriormente la medida de efectividad, o función objetivo en el contexto de la SO, que se pretende optimizar para el sistema es la ***"utilidad monetaria"***; para el caso particular que se está analizando, y como ya se mencionó también, con el fin de tener un parámetro de validación de los resultados obtenidos, el cálculo de esta medida de efectividad será realizado con base a los siguientes lineamientos:

1. Costo de operar un servidor de cualquier estación de trabajo es de *34.7222* unidades monetarias (UM) por hora de actividad del sistema.

2. Costo de operar un lugar de almacenaje en cualquiera de los *buffers* es de *1.3888* unidades monetarias (UM); por hora de actividad del sistema.

3. Precio de venta de un *ítem* convertido en producto es de *200* UM

4. Costo de insumos descartable.

Bajo estos supuestos el cálculo de la ***"utilidad monetaria"*** que brinda el sistema, bajo un escenario, o configura específica, podría ser definido como sigue:

Utilidad Monetaria =

(200 × Items Convertidos en Producto)

- (34.7222 ×

(Número total de servidores en la estación de trabajo 1 +

Número total de servidores en la estación de trabajo 2 +

Número total de servidores en la estación de trabajo 3 +

Número total de servidores en la estación de trabajo 4)

× (Horas Totales de Operación del Sistema)

)

- (1.3888 ×

(Número total de espacios disponibles en el buffer 2 +

Número total de espacios disponibles en el buffer 3 +

Número total de espacios disponibles en el buffer 4)

× (Horas Totales de Operación del Sistema)

)

4.4 PROBLEMÁTICA

Utilizando los supuestos planteados en las secciones anteriores, base de la definición de la variante particular del caso de estudio, la problemática que se abordara en el sistema puede ser representada más formalmente como:

$$Max\ E\left[\left(200(ICP)\right) - \sum_{i=1}^{m} 34.7222\ (svrs_i)(t)\ -\ \sum_{i=2}^{m} 1.3888\ (b_i)(t)\right]$$

Sobre todas las posibles combinaciones de $svrs_1$, $svrs_2$, $svrs_3$, $svrs_4$, b_2 , b_3 , b_4 , *o lo que es lo mismo sobre todos los vectores de la forma:*

$$(\ svrs_1\ ,\ svrs_2\ ,\ svrs_3\ ,\ svrs_4\ ,\ b_2\ ,\ b_3\ ,\ b_4\ )$$

Donde:

$t = periodo\ de\ tiempo\ en\ el\ que\ será\ calculda\ la\ utilidad\ monetaria\ del\ sistema$

$ICP = Número\ de\ Items\ Convertidos\ en\ producto\ en\ el\ periode\ de\ tiempo\ t$

y

$svrs_i$ representa el número de servidores colocados en la estación de trabajo *i,* y cumple:

$0 < svrs_i \leq 3$

con $svrs_i$, $Usvrs_i$ *enteros positivos*

i =1,2,3,4

b_i representa la capacidad del buffer *i,* y cumple:

$0 < b_i \leq 10$

con b_i, Ub_i *enteros positivos*

i =2,3,4

4.5 Estimación de la Solución a la Problemática

Antes de estimar una solución a la problemática presentada con el uso del diseño implementado, es necesario primero establecer las condiciones de experimentación, mismas que serán transmitidas al software como parámetros de entrada, y que fueron contemplados como tal en la etapa de diseño del mismo.

Parámetros de la Simulación. Están relacionados directamente con la simulación de los escenarios que se pueden presentar en el sistema, y en este tenor en la etapa de diseño del software se contemplaron los siguientes:

1. Tiempo que durarán las simulaciones.

2. Número de repeticiones para obtener el valor de la F.O./Medida de Efectividad.

3. Semilla para la generación de las muestras *Uniforme (0,1)*.

Parámetros del Algoritmo de Optimización. Están ligados con el algoritmo de optimización y de su configuración puede muchas veces depender el desempeño del algoritmo. En nuestro diseño de software, y hay que aclarar que estos parámetros son consecuencia del algoritmo de optimización seleccionado para la aplicación del concepto, se tienen primero los siguientes:

1. Tamaño de la Población.

2. % Herencia Individuos Mejor Adaptados.

3. % Individuos a Obtener por Cruza.

4. Probabilidad de Mutación.

Adicionalmente a estos, tenemos también los relacionados con la condición de paro del algoritmo de optimización:

5. Número Máximo de Generaciones.

6. % Mínimo de Mejora entre Generaciones.

7. Número Máximo de Generaciones sin Mejora.

De estos últimos hay que mencionar que para efectos de los resultados mostrados en este capítulo solo se utilizó el primero.

Estos parámetros definen por un lado las condiciones en torno a las cuales el modelo de simulación obtendrá el valor de la función objetivo, o medida de efectividad, y por otro, la configuración del algoritmo de optimización en la búsqueda del óptimo.

4.6 Búsqueda de la Solución / Experimentación y Resultados

1ª Estimación

La primera aproximación a la problemática presentada, mediante el uso del software desarrollado, fue realizada con el uso de las siguientes condiciones de experimentación.

Condiciones Experimentales

Parámetro	Valor
Tamaño de la Población	20
% Herencia Individuos Mejor Adaptados	10%
% Individuos a Obtener por Cruza	50%
Probabilidad de Mutación	0.70
Número Máximo de Generaciones	10
Tiempo que durarán las simulaciones	100
Número de Repeticiones	3
Semilla	26546

Tabla 22. Condiciones Experimentales Primera Estimación

Una vez que fijamos estos parámetros del software SO llevamos a cabo la ejecución del mismo; obteniendo los siguientes resultados:

El escenario analizado que obtuvo el máximo valor para nuestra función objetivo (F.O.).

Escenario	Máximo F.O.	Generación AG	Descripción Escenario
3,3,2,2,4,8,9	*79,296.53*	*5*	***3*** *servers* $WorkStation_1$, ***3*** *servers* $WorkStation_2$, ***2*** *servers* $WorkStation_3$, ***2*** *servers* $WorkStation_4$, $Buffer_2$ *con capacidad de* ***4*** *Unidades,* $Buffer_3$ *con capacidad de* ***8*** *Unidades y* $Buffer_4$ *con capacidad de* ***9*** *Unidades.*

Tabla 23. Mejor Escenario Primera Estimación

Como salida del software se genera un archivo de texto con el detalle del comportamiento de la función objetivo en cada uno de los escenarios analizados en la búsqueda del óptimo. La información dentro de este archivo, como es posible observar en la siguiente figura, permite dar un seguimiento puntual del trabajo realizado por el algoritmo de optimización. En el archivo son registrados los escenarios de cada una de las generaciones poblacionales construidas por el algoritmo genético, además del valor estimado para la función objetivo en cada escenario y las simulaciones requeridas para tal estimación.

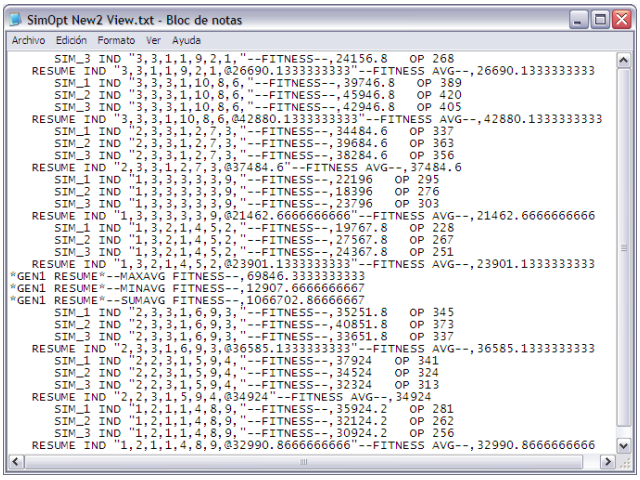


Figura 35. Archivo de detalle salida del software SO

En este punto ya contamos con un escenario que aproxima el máximo de nuestra función objetivo, a saber el escenario definido por el vector *(3,3,2,2,5,3,9)*, sin embargo ésta es solo una estimación, que tan buena, no sabemos, pero para darnos una idea podríamos utilizar el siguiente razonamiento:

La estación de trabajo 2 es potencialmente el mayor cuello de botella dentro del sistema, la razón es que sus servidores poseen la tasa de servicio más baja, 2 items por unidad de tiempo, esto nos lleva a pensar, al menos por puro sentido común, que esta estación de trabajo debería tener el mayor número de servidores permitidos, es decir 3, con lo que la estación obtendría una tasa potencial total de servicio de 6 items por unidad de tiempo. A continuación podríamos también argumentar que la estación de trabajo 3 debería de contar con al menos 2 servidores, si tuviera menos, es decir uno, automáticamente se convertiría en el siguiente cuello de botella, ya que solo podría servir a razón de 5 items por unidad de tiempo; con argumentos similares la estación de trabajo 4 debería también tener al menos 2 servidores. Finalmente ¿cuántos servidores debiera tener la estación de trabajo uno?, pues bien bajo el mismo razonamiento parecería que al menos 2 sería la cantidad ideal, debido a que con esta cantidad de servidores la tasa potencial total de servicio de esta estación de trabajo igualaría la de la estación de trabajo 2 [20].

Si además, tomamos en cuenta que a mayor número de servidores operando en el sistema, mayor será el costo de operación asociado a las estaciones de trabajo, y éste se verá reflejado de forma negativa en el valor nuestra función objetivo:

Utilidad Monetaria =

(Precio de venta del producto × Items convertidos en producto en el periodo de tiempo)

- Costo de operación de las estaciones de trabajo en el periodo de tiempo

- Costo de operación de los buffers en el periodo de tiempo

Vemos entonces que nuestra primera estimación, al escenario que optimiza la ***"utilidad monetaria"*** que brindará el sistema, hace sentido y parece ir por buen camino.

2ª ESTIMACIÓN

CONDICIONES EXPERIMENTALES

Parámetro	**Valor**
Tamaño de la Población	20
% Herencia Individuos Mejor Adaptados	10%
% Individuos a Obtener por Cruza	30%
Probabilidad de Mutación	0.40
Número Máximo de Generaciones	15
Tiempo que durarán las simulaciones	100
Número de Repeticiones	3
Semilla	7777

Tabla 24. Condiciones Experimentales Segunda Estimación

Cambios con respecto a la primera estimación:

1. Aumento del número de generaciones poblacionales que tendrá que obtener el algoritmo genético antes de parar.

2. Disminución de la probabilidad de mutación, para que los cambios en la población a causa de este operador se den más lentamente.
3. Finalmente se decidió también disminuir el % de individuos a obtener por cruza.

En esta ocasión, el escenario analizado por el algoritmo de optimización que obtuvo el máximo valor para nuestra función objetivo (F.O.) fue el siguiente:

Escenario	Máximo F.O.	Generación AG	Descripción Escenario
3,3,2,2,5,7,6	*80,246.26*	*11*	***3*** *servers* $WorkStation_1$, ***3*** *servers* $WorkStation_2$, ***2*** *servers* $WorkStation_3$, ***2*** *servers* $WorkStation_4$, $Buffer_2$ *con capacidad de* ***5*** *Unidades,* $Buffer_3$ *con capacidad de* ***7*** *Unidades y* $Buffer_4$ *con capacidad de* ***6*** *Unidades.*

Tabla 25. Mejor Escenario Segunda Estimación

Observemos que el número de servidores por estación de trabajo en este escenario es el mismo que el encontrado en la primera estimación:

- 3 Servidores Primera Estación de Trabajo
- 3 Servidores Segunda Estación de Trabajo
- 2 Servidores Tercera Estación de Trabajo
- 2 Servidores Cuarta Estación de Trabajo

3ª ESTIMACIÓN

CONDICIONES EXPERIMENTALES

Parámetro	Valor
Tamaño de la Población	20
% Herencia Individuos Mejor Adaptados	10%
% Individuos a Obtener por Cruza	20%
Probabilidad de Mutación	0.30
Número Máximo de Generaciones	15
Tiempo que durarán las simulaciones	100
Número de Repeticiones	3
Semilla	19999

Tabla 26. Condiciones Experimentales Tercera Estimación

Cambios con respecto a la anterior estimación:

1. Ajuste de Probabilidad de Mutación.

2. Ajuste de % Individuos a obtener por cruza.

Al ejecutar nuevamente el software bajo estas condiciones experimentales observamos que nuevamente el número de servidores por estación de trabajo en el escenario que esta vez obtuvo el máximo valor para nuestra función objetivo (F.O.) concordó con los resultados de las dos estimaciones anteriores:

Escenario	Máximo F.O.	Generación AG	Descripción Escenario
3,3,2,2,7,7,8	*83,891.06*	*14*	***3*** *servers* $WorkStation_1$, ***3*** *servers* $WorkStation_2$, ***2*** *servers* $WorkStation_3$, ***2*** *servers* $WorkStation_4$, $Buffer_2$ *con capacidad de* ***7*** *Unidades,* $Buffer_3$ *con capacidad de* ***7*** *Unidades y* $Buffer_4$ *con capacidad de* ***8*** *Unidades.*

Tabla 27. Mejor Escenario Tercera Estimación

En la siguiente gráfica se muestra el comportamiento del algoritmo de optimización para la presente estimación.

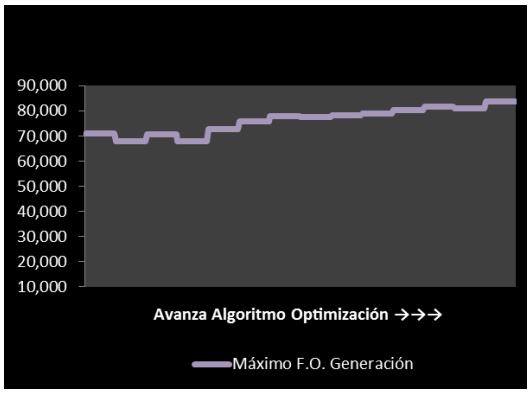


Figura 36. Gráfica de Resultados Tercera Estimación

Recapitulando un poco, sobre los resultados obtenidos en estas tres estimaciones, se podría observar primero que los cambios en los parámetros del algoritmo de optimización influyen directamente en el desempeño del mismo; adicionalmente, y como ya se mencionó, las tres estimaciones al escenario que

podría maximizar el valor de nuestra función objetivo concuerdan en cuanto al número de servidores por estación de trabajo:

- 3 Servidores Primera Estación de Trabajo
- 3 Servidores Segunda Estación de Trabajo
- 2 Servidores Tercera Estación de Trabajo
- 2 Servidores Cuarta Estación de Trabajo

Finalmente, y con el fin de poder comparar los resultados de obtenidos por el software con los presentados en la referencia ya citada [20], llevaremos a cabo una cuarta estimación, pero no sobre todo el espacio de resultados, como lo veremos a continuación.

4ª ESTIMACIÓN

CONDICIONES EXPERIMENTALES

Parámetro	Valor
Tamaño de la Población	20
% Herencia Individuos Mejor Adaptados	10%
% Individuos a Obtener por Cruza	20%
Probabilidad de Mutación	0.30
Número Máximo de Generaciones	8
Tiempo que durarán las simulaciones	720
Número de Repeticiones	3
Semilla	6111
Configuración Fija del Número de Servidores por Estación de Trabajo	**3** Servidores Primera Estación de Trabajo **3** Servidores Segunda Estación de Trabajo **2** Servidores Tercera Estación de Trabajo **2** Servidores Cuarta Estación de Trabajo
F.O. **=** *Utilidad Neta = 200 × (Items Convertidos en Producto) - (25,000 × (3 + 3 + 2 + 2)) - (1000 × (Número total de espacios disponibles en el buffer 2 + Número total de espacios disponibles en el buffer 3 + Número total de espacios disponibles en el buffer 4))*	

Tabla 28. Condiciones Experimentales Cuarta Estimación

Estas condiciones experimentales fueron motivadas por los siguientes hechos:

1. Todas las estimaciones anteriores concuerdan en cuanto al número de servidores por estación de trabajo que podría maximizar el valor de nuestra función objetivo.

2. Los archivos que contienen el detalle del comportamiento de la función objetivo, para cada una de estas estimaciones, muestran que es bajo esta configuración que se obtienen los mejores valores.

3. El análisis que hicimos en la primera estimación, con el objetivo de tratar de validar un poco el resultado obtenido, contempla también esta configuración.

Estos tres puntos nos llevan a pensar que es altamente probable que los escenarios del sistema con mejores valores de la función objetivo, o quizás el óptimo, ¿por qué no?, estén en torno a esa configuración específica, en lo referente al número de servidores por estación de trabajo.

Hay que observar que, con el uso de este supuesto, las únicas variables de entrada que moveremos en el modelo de simulación serán las relacionadas con los espacios disponibles en cada uno de los 3 *buffers* con los que cuenta el sistema, reduciendo la cardinalidad del espacio de resultados de ***ochenta y un mil*** a:

$$Ub_2 \times Ub_3 \times Ub_4 = 10 \times 10 \times 10$$

Es decir ***mil*** escenarios factibles, con lo que obtendremos una reducción significativa del espacio de resultados donde se llevará a cabo la búsqueda de optimalidad.

Por otro parte con fin de poder comparar los resultados del software con los ya citados, se establecerá también un tiempo de simulación mayor para obtener la medida de efectividad: 720 hrs.

Es así como, bajo las condiciones experimentales ya referidas, el escenario que maximiza el valor de nuestra función es el siguiente:

Escenario	Máximo F.O.	Generación	Descripción
3,3,2,2,7,6,8	*570,600*	*8*	***3*** *servers WorkStation*$_1$, ***3*** *servers WorkStation*$_2$, ***2*** *servers WorkStation*$_3$, ***2*** *servers WorkStation*$_4$, *Buffer*$_2$ *con capacidad de* ***7*** *Unidades, Buffer*$_3$ *con capacidad de* ***6*** *Unidades y Buffer*$_4$ *con capacidad de* ***8*** *Unidades.*

Tabla 29. Mejor Escenario Cuarta Estimación

Para finalizar dejamos una tabla que muestra los resultados obtenidos en la fuente ya mencionada[11].

		Valor Esperado de la Función Objetivo
Mejor Escenario	*3,3,2,2,7,8,4*	*591,512*
Siguiente Mejor Escenario	*3,3,2,2,7,7,4*	*548,488*

Tabla 30. Resultados Averill M. Law y Michael G. McComas [20]

[11] El detalle de los resultados puede ser consultado en [20]

CONCLUSIONES.

El presente trabajo explora el porqué de la importancia que está tomando el concepto “Simulación y Optimización” (SO), en particular en el área de simulación dentro de la investigación de operaciones, posicionándose como una de las ramas en las que se prevé un gran crecimiento futuro. Sin embargo, es importante señalar, que aun cuando se abordan las posibles ventajas que esta técnica podría tener sobre otras, también relacionadas con la optimización de sistemas en la I. de O., esto no ha sido con objetivo de poner a la SO por encimas de las mismas; lo que se pretende mostrar es como la SO enriquece a la I. de O. abriendo aún más las posibilidades en el área de optimización de sistemas, habrá muchas ocasiones en que la SO pueda ser una buena opción, pero también habrá muchas en las que no.

En el presente trabajo se muestra también la importancia que tiene la cardinalidad del espacio de resultados del problema en la justificación del uso de la técnica, pero, hay que decirlo, esto es solo un factor dentro de un abanico de posibilidades que puede ser muy amplio; no se debe olvidar que, entre otras

cosas, todo problema que puede ser atacado con SO también puede ser atacado con el uso de únicamente simulación, esto al menos teóricamente.

Ayudándose del caso de estudio al que es aplicado el concepto, el presente documento desarrolla y documenta un diseño completo de software basado en el concepto SO, esto con la utilización de dos metodologías relacionadas con el diseño de software orientado a objetos, la desarrollada por Grady Booch: Booch OOAD [12], y la desarrollada por James Rumbaugh: OMT [13], además de estándares de modelado UML [9]. El diseño abarca dos componentes de software principales el de simulación y el de optimización; en el caso del diseño del componente de simulación, el trabajo fue encaminado para que, además de poder representar sistemas del tipo del caso de estudio, éste fuera capaz de simular en general sistemas cuyo comportamiento puede ser visto como una interacción de varias líneas de espera; de manera que el diseño propuesto para el módulo de simulación puede bien ser reutilizado para la generación de un sistema de simulación discreta basado en líneas de espera.

Hay que mencionar también, que el proceso de diseño por si mismo deja evidencia de la importancia de la integración del concepto “orientado a objetos” (OO, por sus siglas en inglés) dentro del ámbito del diseño de software de simulación (Simulación Orientada a Objetos – OOS, por sus siglas en inglés). De hecho podemos asegurar que la presente propuesta de diseño, con los alcances fijados inicialmente, no hubiera podido ser llevada a cabo sin la utilización de este enfoque en el componente de simulación.

En cuanto a la validación de la propuesta de diseño, el capítulo IV muestra algunos de los resultados de la aplicación del mismo a una variante particular del caso de estudio, variante de la que se pudo obtener información para contrastar; sin embargo, hay que subrayar que el diseño propuesto cubre cualquier variante del caso abordado. En cuanto a los resultados obtenidos tras la aplicación del diseño, éste arrojó un escenario que aproxima de buena manera el máximo de la función objetivo sin necesidad de realizar un análisis directo de los 81,000

escenarios factibles que plantea la variante particular del caso, para realizar la estimación del escenario óptimo fueron analizados menos de 1000 escenarios factibles. En el capítulo IV también es posible observar que la SO también puede ser utilizada como una herramienta de análisis, y que el manejo de los parámetros iniciales en técnicas heurísticas de optimización que tratan de imitar los conceptos presentes en la evolución biológica, influye de manera directa en el comportamiento del algoritmo de optimización y por lo tanto en la búsqueda de optimalidad.

Hay que mencionar también, que aun cuando la propuesta de solución presentada hace uso de algoritmos genéticos, dichos algoritmos representan solo una alternativa, y en este sentido, una de las posibilidades que brinda el diseño de software presentado, es el poder incorporar con mayor facilidad otras técnicas de optimización.

En conclusión el presente trabajo aporta un diseño en SO funcional y con buenas posibilidades de reutilización futura, debido al enfoque con el que fue concebido, OO.

Por último se mencionan algunas mejoras que pueden ser llevadas a cabo, tanto en el diseño como en torno a la implementación del mismo, en trabajos futuros:

1) Utilización de otra herramienta de programación como podría ser C++®.

2) Utilización de estructuras de datos y apuntadores para el manejo de los eventos generados por el módulo de simulación para “imitar” el sistema.

3) Habilitar un proceso en automático, que en torno a la varianza encontrada en la medida de efectividad, recomiende el número de repeticiones en la simulación de un escenario específico.

4) Incorporación de alguna técnica estadística que ayude a valorar la calidad de los resultados obtenidos.

5) Incorporar alguna otra técnica de optimización heurística, e inclusive una mezcla de estas técnicas bajo el enfoque equipos asíncronos [Enfoques de la Simulación y Optimización p.30].

REFERENCIAS

1. *SIMULATION IN AN OBJECT-ORIENTED WORLD.* **A. Joines, Jeffrey / Roberts, Stephen D.** s.l. : P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds., Proceedings of the 1999 Winter Simulation Conference.

2. *AN INTRODUCTION TO OBJECT-ORIENTED SIMULATION IN C++.* **Jeffrey, A. Joines / Stephen, D. Roberts.** North Carolina State University : s.n., Proceedings of the 1997 Winter Simulation Conference.

3. **Hoover, Stewart V. / Perry, Ronald F.** *SIMULATION "A Problem-Solving Approach".* Northeastern University USA : Addison-Wesley Publishing Company, 1989.

4. **Shannon, Robert E.** *System Simulation: The Art and Science".* N.J. : Prentice Hall, Englewood Cliffs, 1975. pp. 723-724.

5. **Varios, PROMODEL Corporation /.** *Manuales PROMODEL.* UT U.S.A : PROMODEL, 2003. pp.285-309.

6. *SIMULATION OPTIMIZATION: METHODS AND APPLICATIONS.* **Carson, Yolanda / Anu, Maria.** State University of New York at Binghamton : ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, Proceedings of the 1997 Winter Simulation Conference.

7. *SIMULATION OPTIMIZATION: A REVIEW, NEW DEVELOPMENTS, AND APPLICATIONS.* **Fred W. Glover, Michael C. Fu.** Proceedings of the 2005 Winter Simulation Conference : M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds., 2005.

8. *Heuristic "Optimization": Why, When and How to Use It.* **Zanakis, S.H. / Evans J. R.** s.l. : Interfaces, Vol 11, no. 5, 1981.

9. **OMG.** OMG Unified Modeling Language (OMG UML). *Superstructure, V2.1.2.* [Online] November 2007. http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF.

10. About OMG®. [Online] http://www.omg.org/gettingstarted/gettingstartedindex.htm.

11. **Graham, Ian.** *Object Orineted Methods.* Swiis Bank Corporation, London : Addison-Wesley, 1993.

12. **Booch, Grady.** *Object Oriented Design with Applications.* Redwood City, California : The Benjamin/Cummings Publishing Company, Inc, 1991.

13. **Rumbaugh, James.** *Modelado y diseño orientados a objetos.* Madrid : Prentice Hall, 1996.

14. **Kurt Bittner, Ian Spence.** *Use Case Modeling.* Boston, MA : Addison-Wesley Booch Jacobson Rumbaugh, 2003.

15. **Synergix.** Tecnología y Synergix. [Online] http://synergix.wordpress.com/2008/07/10/modelo-de-dominio/.

16. **Cooper, Robert B.** *INTRODUCTION TO QUEUEING THEORY.* USA : Macmillan, 1972. pp. 1-9.

17. **Ruble, David A.** *Practical Analysis & Design For Client/Server & GUI Systems.* USA : Yourdon Press Computing Series, 1997. pp. 343-377.

18. **IBM / Donald Bell, IT Architect, IBM Corporation.** UML basics: The sequence diagram. [Online] 2010. http://www.ibm.com/developerworks/rational/library/3101.html.

19. **Larman, Craig.** *Applying UML and Patterns, An Introduction to Object-Oriented Analysis and the Unified Process.* United States of America : Prentice Hall, 2002.

20. **Law, Averill M. / McComas, Michael G.** SIMULATION-BASED OPTIMIZATION. Proceedings of the 2002 Winter Simulation Conference : Averill M. Law and Associates, Inc., 2002.

21. **Mood, Alexander M. / Graybill, Franklin A. / Boes, Duane C.** *Introduction to the theory of Statistics.* s.l. : McGraw-Hill INTERNATIONAL EDITIONS, 1974. pp. 202-203.

22. **Koza, John R.** *Genetic Programming.* Cambridge, MA : MIT Press, 1991.

23. **Davidor, Yuva.** *Genetic Algorithms and Robotics.* London : World Scientific, 1991.

24. **Fogel, David B.** *Evolutionary Computation.* s.l. : IEEE Press, John Wiley and Sons, 2006.

25. **Stewart, Robinson.** *Simulation - The practice of model development and use.* USA : Wiley, 2004.

26. **Davis, Lawrence David.** *Handbook of Genetic Algorithms.* NY USA : Van Nostrand Reinhold, 1991.

27. **Law, Averill M. / Kelton, W. David.** *Simulation Modeling and Analisys.* Third Edition : Mc Graw Hill. pp. 622-668.

28. *CRITERIA FOR SIMULATION SOFTWARE EVALUATION.* **Nikoukaran, Jalal / Hlupic, Vlatka / Ray, J. Paul.** Brunel University : s.n., Proceedings of the 1998 Winter Simulation Conference.

29. *PRACTICAL INTRODUCTION TO SIMULATION OPTIMIZATION.* **April, Jay / Glover, Fred / Kelly, James P. / Laguna Manuel.** s.l. : OptTek Systems, Proceedings of the 2003 Winter Simulation Conference.

30. *SIMULATION OPTIMIZATION.* **Fu, Michael C.** University of Maryland : B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds., Proceedings of the 2001 Winter Simulation Conference.

31. **Sommerville, Ian.** *Software Engineering.* University of St Andrews, Scotland : Addison-Wesley, Pearson Education, Inc., 2010.

ANEXOS

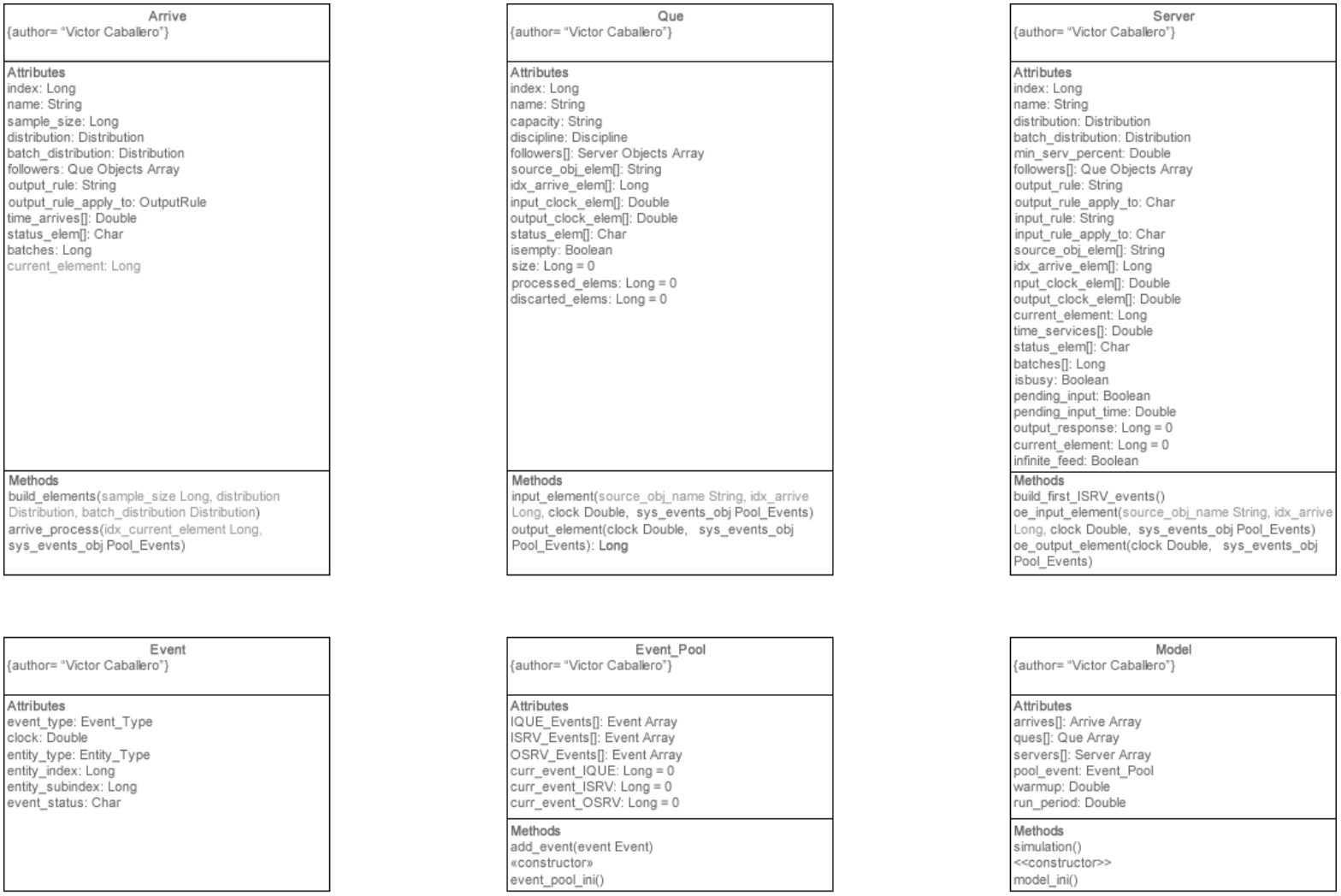


Figura 37. Diseño de Clases UML - Módulo de Simulación

MÉTODO DE TRANSFORMACIÓN DE PROBABILIDAD

Este método se basa en el siguiente teorema, también conocido como de inversión.

Sea X una variable aleatoria con función de distribución de probabilidad acumulada continua $F(x)$, Entonces $U = F(X)$ es uniformemente distribuida sobre el intervalo $(0,1)$. Como consecuencia, si U es una variable aleatoria uniformemente distribuida sobre el intervalo $(0,1)$, entonces la variable aleatoria $X = F^{-1}(U)$ tienen función de distribución de probabilidad acumulada $F(x)$. [21]

Para obtener una muestra de una variable aleatoria que tiene una función de distribución continua específica $F(x)$, el método de transformación de probabilidad toma primero una muestra aleatoria de una distribución $Uniforme\ (0,1)$, llamémosle $U_1, U_2, U_3, \ldots U_n$. Esta muestra sirve como base para obtener entonces la muestra de la distribución requerida $X_1, X_2, X_3, \ldots X_n$,donde X_j, la j-ésima $(j=1,2,3, \ldots n)$ muestra de la variable aleatoria con función de distribución $F(x)$, es obtenida únicamente proyectando U_j en la gráfica de $F(x)$ como se muestra en la siguiente figura .

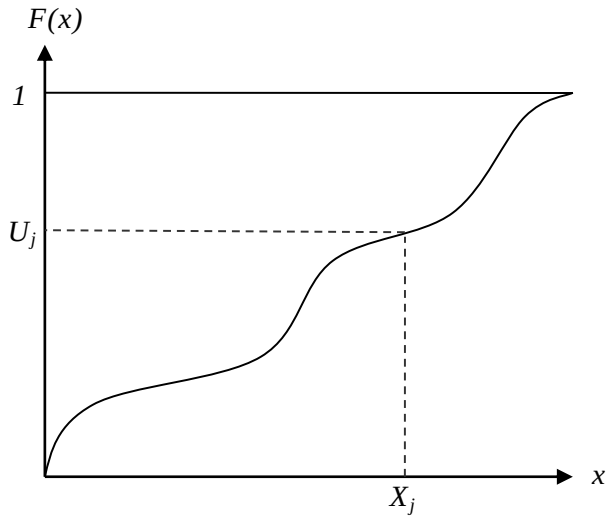


Figura 38. Método de Transformación de Probabilidad

Al obtener X_j de esta manera tendremos que $Pr\{ X_j \leq x \} = Pr\{ U_j \leq F(x) \}$, y debido a que U_j es $Uniforme\ (0,1)$; entonces $Pr\{ U_j \leq F(x) \} = F(x)$, de donde $Pr\{ Xj \leq x \} = F(x)$.

En la generación de las muestras aleatorias (en este caso pseudo-aleatorias) de la distribución uniforme en (0,1), necesarias para el método de transformación de probabilidad, se utilizó el proceso de Montecarlo, el cual se describe a continuación.

El proceso se inicia con la generación de una muestra de números pseudo-aleatorios dentro de un rango específico de valores. Esta muestra deberá cumplir con ciertas condiciones; entre las cuales destacan la igualdad de probabilidad de todos sus valores e independencia estadística. Vale la pena observar que el muestreo se dice pseudo-aleatorio porque puede ser totalmente reproducido mediante la utilización de un algoritmo matemático. En nuestro caso, para obtener esta muestra utilizaremos un algoritmo recursivo basado en métodos de congruencia, el cual ya ha sido probado estadísticamente, y donde el valor de la $n+1$ *–ésima* muestra estará determinado por:

$$l_{n+1} = mod\ (l_n \times k, m)$$

Donde $mod\ (l_n \times k, m) = Residuo\ del\ cociente\ (l_n \times k)/m$ y, k, m son enteros positivos, para efectos de nuestro sistema tendrán los siguientes valores: $k=65539$ y $m=231$. Estos valores son recomendados por algunos autores para efectuar una buena generación de números pseudo-aleatorios en computadoras a 32 bits. El valor de l_0, también llamado semilla, es incorporado como parámetro del software y es un número entero positivo que puede ser fijado a conveniencia para variar los resultados del muestreo. De esta manera al aplicar este algoritmo recursivo $n - 1$ veces, con $l_0 > 0$, obtendremos la muestra pseudo-aleatoria de tamaño n, de números enteros entre 1 y $m-1$, igualmente probables y estadísticamente independientes, requeridos para el proceso.

Como siguiente paso, a partir de la muestra anterior será obtenida una muestra de una distribución *Uniforme (0,1)* mediante el siguiente proceso de normalización:

$$U_n = mod\ (l_n \times k, m) / (m-1); \qquad (n \geq 1).$$