UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

AMPLIACIÓN Y MEJORAMIENTO DE LAS FUNCIONALIDADES DEL GRID COMPUTACIONAL SUMA PARA EL SOPORTE DE EJECUCIONES MÚLTIPLES SIMULTÁNEAS

TRABAJO ESPECIAL DE GRADO

presentado ante la

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

como parte de los requisitos para optar por el título de

INGENIERO EN INFORMÁTICA

REALIZADO POR: Br. Luzmary Di Felice Gatti

Br. Adrián A. García Blanco

PROFESOR GUÍA: Ing. Yudith Cardinale

FECHA: Caracas, 23 de Noviembre de 2006

ÍNDICE GENERAL

Pág.

SINOPSIS 1
CAPÍTULO I: DEFINICIÓN DEL PROBLEMA 2
I.1. Introducción 2
I.2. Planteamiento del Problema 3
I.3. Objetivos 5
I.3.1. Objetivo General 5
I.3.2. Objetivos Específicos 5
I.4. Justificación 6
I.5. Limitaciones y Alcances 7

CAPÍTULO II: MARCO REFERENCIAL 9
II.1. Descripción de Herramientas 9
II.1.1. XML (Extensile Markup Language) 10
II.1.1.1. Ventajas del XML 10
II.1.2. CORBA 11
II.2 Grid Computacional 12
II.2.1. Definición de Grid Computacional 14
II.2.2. Ventajas e Incovenientes del Grid 14
II.2.2.1 Beneficios del Grid 15
II.2.2.2 Desafíos 15
II.2.3. Requisitos Indispensables de un Grid / Aspectos de Diseño 16
II.2.4. Extensiones del Gris: Aplicaciones de Ejecuciones Múltiples Simultáneas 17

CAPÍTULO III: CASO DE ESTUDIO:SUMA/G 19
III.1. Definición de SUMA/G 19
III.1.1. Objetivos de SUMA/G 20
III.1.2. Descripción de SUMA/G 21
III.1.3. Ejecución Básica de SUMA/G 22
III.1.4. Componentes de SUMA/G 24
III.1.4.1. SUMA/G Internal Client (Client Stub) 24
III.1.4.2. SUMA/G Engine 25
III.1.4.3. SUMA/G Scheduler (Planificador) 26
III.1.4.4. SUMA/G Application Server 28
III.1.4.5. SUMA/G Resource Control 28
III.1.4.6. SUMA/G User Control 28
III.1.5. Funcionamiento de SUMA/G 28
III.1.5.1. Localización de Agentes 28
III.1.5.2. Ejecución multi-hilos y paralelas 29

CAPÍTULO IV: MARCO METODOLÓGICO 31
IV.1. Modelo Incremental 31

CAPÍTULO V: DESARROLLO DEL SERVICIO DE EJECUCIONES MÚLTIPLES SUMULTÁNEAS DE SUMA/G 35
V.1. Iteración Nº1: Modificación de la estructura de SUMA/G para el manejo de ejecuciones Parameter Sweep 35
V.1.1. Análisis 35
V.1.2. Diseño 39
V.1.2.1. Componentes de SUMA/G modificados 40
V.1.2.1.1. SUMA/G Internal Client 40
V.1.2.1.2. SUMA/G Engine 41
V.1.2.1.3. SUMA/G Scheduler 42
V.1.2.1.4. SUMA/G Application Server 43
V.1.2.1.5. SUMA/G Resource Control 44
V.1.2.2. Nuevo componente ha ser agregado a SUMA/G 44
V.1.2.2.1. Clase ServerPSWrapper 45
V.1.2.2.2. Clase ConcreteServerPS 46
V.1.2.2.3. Clase ParamSweep 47
V.1.2.3. Interacción entre los diferentes componentes de SUMA/G 50
V.1.3. Implementación 52
V.2. Iteración 2: Creación de módulos que provean el control de las instancias de ejecución. 52
V.2.1. Análisis 52
V.2.2. Diseño 53
V.2.2.1. Componente de SUMA/G a ser modificado..... 53
V.2.2.2. Incorporación de los servicios getStatusPS y getResultPS al ServerPSEngine 53
V.2.2.3. Creación de las clases JobStatusPS y JobResultPS 54
V.2.2.4. Diagrama de Secuencia 55
V.2.2. Implementación 56
V.3. Iteración 3: Creación del cliente gráfico para la realización de ejecuciones Pararmeter Sweep 57
V.3.1. Análisis 57
V.3.2. Diseño 59
V.3.3. Implementación 60

CAPÍTULO VI: RESULTADOS 61
CAPÍTULO VII: CONCLUSIONES Y RECOMENDACIONES 65
REFERENCIAS BIBLIOGRÁFICAS 68
ANEXOS 71
GLOSARIO 81

ÍNDICE DE FIGURAS

Pág

Figura Nº 1. Utilización de CORBA y XML dentro de un grid computacional 9
Figura Nº 2. Esquema de SUMA/G 21
Figura Nº 3. Cluster de Computadoras 22
Figura Nº 4. Diagrama de Clases de SUMAIntCliente 25
Figura Nº 5. Diagrama de Clases SUMAEngine 26
Figura Nº 6. Diagrama de clases del SUMAScheduler 28
Figura Nº 7. Metodología Incremental 33
Figura Nº 8. Objetivos a lograr en cada iteración 34
Figura Nº 9. Petición Submit 38
Figura Nº 10. Diagrama Petición Parameter Sweep 40
Figura Nº 11. Diagrama de Clases Ejecución Parameter Sweep 45
Figura Nº 12. Diagrama de la clase ConcreteServerPS 46
Figura Nº 13. Estructura del Archivo XML 49
Figura Nº 14. Diagrama de Secuencia de la Ejecución Parameter Sweep 51
Figura Nº 15. Diagrama de Secuencia de la petición JobStatusPS 56
Figura Nº 16. Diagrama de Secuencia de la petición JobResultPS 56
Figura Nº 17. Diagrama de casos de uso del Cliente ParamSweep 59
Figura Nº 18. Página principal del cliente gráfico 60
Figura Nº 19. Cuadro comparativo de las ejecuciones secuencias versus las ejecuciones múltiples simultáneas..... 63
Figura Nº 20. Tabla Comparativa del Tiempo de Ejecución entre ejecuciones secuenciales y una ParamSweep..... 64

SINOPSIS

Un Grid Computacional es una infraestructura de hardware y software que permite el uso coordinado y compartido de recursos que se encuentran geográficamente distribuidos.

SUMA/G es un grid computacional especializado en la ejecución de programas Java tanto secuenciales como paralelos, cuyo objetivo principal es el de aprovechar eficientemente las capacidades de los recursos con los que cuenta en su red. Este Trabajo de Grado tiene como objetivo, extender los servicios de ejecuciones de SUMA/G, mediante la implementación de una nueva funcionalidad que le permita procesar solicitudes de Ejecuciones Múltiples Simultáneas de un mismo programa con diferente conjunto de parámetros de entrada, es decir que le permita trabajar con Aplicaciones *Parameter Sweep*.

Con el fin de llevar a cabo este objetivo, se empleó una metodología incremental, debido a que ésta permite dividir el desarrollo del trabajo en diferentes iteraciones, cada una de las cuales requieren de las etapas de análisis, diseño, implementación y pruebas, que hacen posible, a través de cada iteración, la obtención de los resultados esperados, y al cumplimiento de los objetivos deseados. Todo esto para lograr la creación de un Servidor de Aplicaciones *Parameter Sweep* dentro del Grid Computacional de SUMA/G y la creación de un Cliente Gráfico para la ejecución de las mismas a través del portal web.

CAPITULO I

DEFINICIÓN DEL PROBLEMA

I.1 Introducción

El crecimiento de competencias en el mercado, el mayor flujo de información, la necesidad de respuestas más rápidas a consultas y de un ahorro general de tiempo y esfuerzo, el pensar que una computadora pasa 90% de su tiempo sin hacer nada, mientras otra es empleada continuamente, han hecho surgir una revolución que ha avanzado y ayudado a la consolidación y superación de numerosas infraestructuras de Tecnología de Información (TI) que aprovechan al máximo los recursos informáticos, esta revolución es conocida como *Grid Computing,* la cual obtiene el mismo rendimiento de una supercomputadora, pero con hardware mucho más económico.

El grupo de Sistemas Paralelos y Distribuidos de la Universidad Simón Bolívar con la colaboración de otras universidades de Venezuela, ha desarrollado un *Grid Computacional* bajo el nombre de *SUMA/G (Scientific Ubiquitous Metacomputing Architecture on Globus).* SUMA/G es una plataforma distribuida para la ejecución de aplicaciones en *bytecode* de Java, tanto aplicaciones multi-hilos (en un único espacio de memoria) como paralelas, comunicándose a través de paso de mensajes.

La ejecución de Aplicaciones *Parameter Sweep*, son parte importante de la estructura de un grid computacional, ya que le aporta

grandes beneficios a sus usuarios, permitiéndoles la ejecución de una aplicación con distintos conjuntos de parámetros de entrada. Por lo que, con el objeto de ofrecer siempre un mejor servicio y obtener todos los beneficios que ofrece esta estructura, los desarrolladores del proyecto de SUMA/G, han trazado la creación de esta funcionalidad como uno de sus objetivos a cumplir.

El presente Trabajo Especial de Grado busca colaborar con el proyecto de SUMA/G, agregándole el servicio de Ejecuciones Múltiples Simultáneas *(Parameter Sweep)* a su estructura, con el objetivo de aprovechar eficientemente los recursos disponibles en la red.

I.2 Planteamiento del Problema

SUMA/G (Scientific Ubiquitous Metacomputing Architecture on Globus) es un GRID de cómputo cuyo principal servicio es la ejecución remota de aplicaciones Java, tanto secuenciales como paralelas. Los servicios principales que SUMA/G permite son:

- Ejecución en línea en la cual las aplicaciones que se van a ejecutar son locales al cliente
- Ejecución fuera de línea (en batch) en la cual las aplicaciones que se van a ejecutar son locales al cliente pero requiere de la creación de clientes virtuales temporales durante la ejecución de la aplicación
- Ejecución vía web para acceder a aplicaciones que no son locales al cliente (pueden estar en el servidor web de SUMA/G o en cuentas

remotas del cliente)

Actualmente SUMA/G no permite la ejecución de aplicaciones que requieran varias instancias de ejecución con diferentes conjuntos de argumentos o parámetros o con diferentes datos de entrada, por lo que si un cliente requiere ejecutar varias instancias de ejecuciones de la misma aplicación, debe realizar tantos requerimientos de ejecución como instancias quiera ejecutar con un conjunto de argumentos o datos diferentes; además el usuario es quien debe controlar la ejecución de todas las instancias.

Existe un planteamiento muy básico desarrollado en una Tesis Especial de Grado en la UCAB [1], donde se presenta un diseño de Servicios de Ejecuciones Múltiples para SUMA/G y se establecen Clientes Gráficos que soportan tal servicio. Este trabajo ayudo a demostrar la factibilidad de la incorporación de este servicio en SUMA/G, a pesar de que no fue implantada en el Grid Computacional SUMA/G por no adaptarse a los requerimientos del cliente. El presente trabajo plantea la revisión, replanteamiento y puesta en marcha de ese diseño y su real incorporación e implementación en SUMA/G, así como completar la funcionalidad de los Clientes Gráficos.

I.3 Objetivos

Los objetivos que pretende cumplir el siguiente proyecto se presentan a continuación:

I.3.1. Objetivo General

Incorporar Servicios y Mecanismos al Grid Computacional SUMA/G que permitan Ejecuciones Múltiples Simultáneas de una misma aplicación con diferentes datos y parámetros, tanto para ejecuciones paralelas como secuenciales.

I.3.2. Objetivos Específicos

1.- Análisis, replanteamiento y puesta en marcha de la plataforma aplicada con anterioridad en el Grid Computacional SUMA/G, para la ejecución de múltiples instancias paralelas de una misma aplicación.

2.- Análisis y replanteamiento de los módulos que proveen el control de las diferentes instancias de ejecución y dan información acerca de las mismas, para dar al usuario del Grid Computacional SUMA/G mayor facilidad al momento de realizar las ejecuciones múltiples simultáneas.

3.- Adaptación de la interfaz gráfica planteada anteriormente para la ejecución de múltiples instancias de una aplicación según las necesidades actuales del cliente.

4.- Creación de un directorio en común para los clientes virtuales que contenga las clases de la aplicación a ejecutar, para evitar la existencia de datos duplicados y así disminuir el uso de espacio innecesario dándole al Grid Computacional de SUMA/G mayor eficiencia.

5.- Creación de un directorio particular para cada cliente virtual que contenga los datos que necesita para realizar su ejecución.

6.- Incorporar la funcionalidad del Sistema de Ejecuciones Múltiples al portal web de SUMA/G el cual representa una mejora en la metodología actual de trabajo de la organización, según una de sus necesidades planteadas.

7.- Demostrar la mejora en la eficiencia de la utilización de ejecuciones múltiples simultáneas versus la metodología actual de ejecuciones secuenciales del Grid Computacional de SUMA/G, mediante un estudio comparativo de resultados bajo las mismas condiciones de trabajo.

I.4 Justificación

El GRID computacional SUMA/G es una plataforma de cómputo distribuida que está formada por computadoras de distintos niveles de procesamiento, desde PCs personales hasta supercomputadores.

Dentro de los principales objetivos de SUMA/G se encuentra el

aprovechar eficientemente las capacidades de los recursos con que se cuenta en una red local o global, poniendo al alcance de todos los usuarios la velocidad de plataformas de cómputo intensivo.

Una manera de ayudar a cumplir con este objetivo es implantar un mecanismo que permita o garantice un mejor aprovechamiento de las capacidades que ofrece esta plataforma. Una solución al problema antes mencionado es el uso de aplicaciones del tipo *Parameter Sweep* (servicio de ejecuciones múltiples simultáneas), con la que es posible la ejecución en paralelo de distintas instancias de una aplicación por parte de un mismo usuario.

Con esto se conseguiría mayor facilidad para el usuario al momento de ejecutar una aplicación con diferentes parámetros y le ahorraría el tiempo que le tomaría el procesamiento independiente de cada una de ellas, que es como en la actualidad el usuario de SUMA/G debe llevar a cabo este tipo de peticiones, ya que al emplear esta nueva funcionalidad bastaría con una única petición de ejecución para obtener los resultados deseados de los diferentes conjuntos de datos especificados.

I.5 Limitaciones y alcances

- Modificación del Engine de SUMA/G para la generación de un directorio que contenga todas las clases comunes necesarias para la ejecución de múltiples instancias simultáneas de una aplicación,

y con esto lograr mayor eficiencia y ahorro de espacio dentro de la plataforma del Grid Computacional SUMA/G

- Generación de un directorio particular para cada cliente virtual que contenga los datos necesarios para la ejecución de la aplicación

- Modificación del portal web de SUMA/G, para soportar la realización de ejecuciones múltiples simultáneas

- Estudio comparativo entre ejecuciones secuenciales y múltiples en el Grid Computacional SUMA/G, basado en eficiencia y ahorro de espacio

- La solución planteada se ajustará a la capacidad actual del Grid Computacional de SUMA/G debido a que no se plantea la realización de modificaciones en la estructura actual

- La ejecución de múltiples instancias simultánea se deberá adaptar a futuros cambios en el Grid Computacional de SUMA/G

CAPÍTULO II

MARCO REFERENCIAL

II.1 Descripción de Herramientas

Para la implementación y puesta en marcha de un grid computacional, es necesario el uso de un conjunto de herramientas (Ver figura Nº 1). A continuación describiremos algunas de ellas, de las cuales es importante conocer su significado y funcionamiento, ya que son ampliamente utilizadas en este trabajo, por encargarse de establecer la comunicación entre diferentes servidores o procesos dentro del grid computacional SUMA/G.

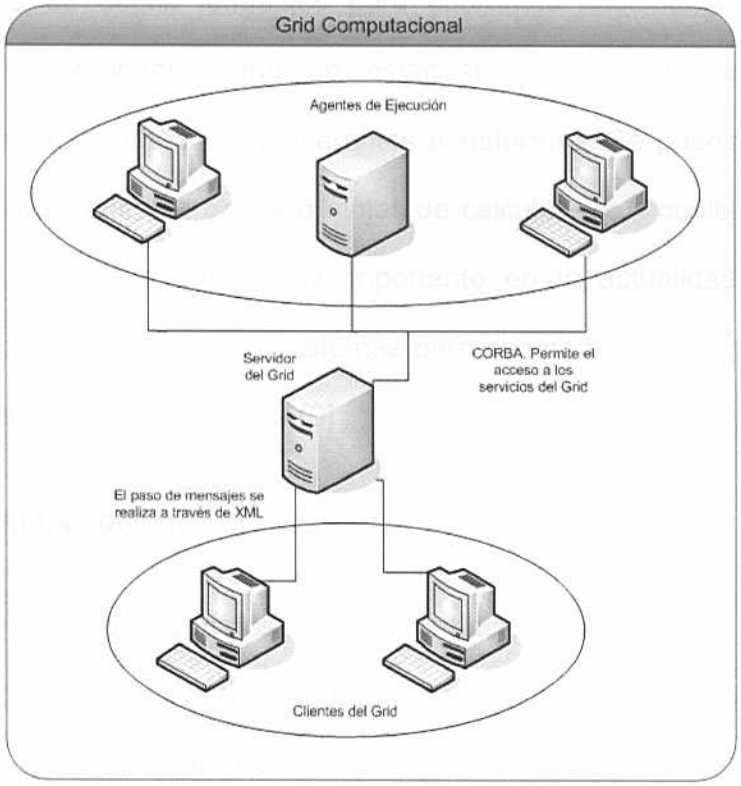


Figura Nº 1. Utilización de CORBA y XML dentro de un grid computacional. Fuente: Elaboración propia.

II.1.1 XML (Extensile Markup Language)

XML es un Lenguaje de Etiquetado Extensible muy simple, pero estricto que juega un papel fundamental en el intercambio de una gran variedad de datos. Desarrollado por el Consorcio de la World Wide Web (W3C). Es una simplificación y adaptación del SGML (Standard General Markup Language) (ISO 8879), y permite definir la gramática de lenguajes específicos. Es muy similar a HTML pero su función principal es describir datos y no mostrarlos como es el caso de HTML. XML es un formato que permite la lectura de datos a través de diferentes aplicaciones [8].

Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades (meta-lenguaje). Se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.[9]

II.1.1.1 Ventajas del XML

- Comunicación de datos. Si la información se transfiere en XML cualquier aplicación podría escribir un documento de texto plano con los datos que estaba manejando en formato XML y otra aplicación recibir esta información y trabajar con ella.

- Migración de datos. Si trabajamos en formato XML sería muy sencillo mover datos de una base de datos a otra.

- Aplicaciones Web. Con XML hay una sola aplicación que maneja los datos y para cada navegador podemos tener una hoja de estilo o similar para aplicarle el estilo adecuado [9]

II.1.2 CORBA

Common Object Request Broker Architecture (CORBA) es un estándar que establece una plataforma de desarrollo de sistemas distribuidos facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos [13].

CORBA fue definido y está controlado por el *Object Management Group* (OMG) que define las APIs, el protocolo de comunicaciones y los mecanismos necesarios para permitir la interoperabilidad entre diferentes aplicaciones escritas en diferentes lenguajes y ejecutadas en diferentes plataformas, lo que es fundamental en computación distribuida [13].

En un sentido general CORBA "envuelve" el código escrito en otro lenguaje en un paquete que contiene información adicional sobre las capacidades del código que contiene, y sobre cómo llamar a sus métodos. Los objetos que resultan pueden entonces ser invocados desde otro programa (u objeto CORBA) desde la red [13].

Los elementos más importantes de la arquitectura de CORBA son el lenguaje de definición de interfaces IDL (Interface Definition Language) para especificar los interfaces con los servicios que los objetos ofrecerán, traducciones de IDL a lenguajes de programación como Java, C++ y otros, y una infraestructura de distribución de objetos ORB (ObjectRequest Broker). CORBA enfatiza en la transparencia y flexibilidad, el cliente no se entera la ubicación del objeto servidor, y mediante IDL logra que las interfaces puedan ser implementadas en más de 30 lenguajes de programación [15].

CORBA es más que una especificación multiplataforma, también define servicios habitualmente necesarios como seguridad y transacciones [13]. Además de un marco estándar para interoperabilidad entre objetos con independencia del lenguaje y de forma transparente al programador [14].

II.2 Grid Computacional

La posibilidad de compartir recursos computacionales costosos a lo largo de un dominio extendido de usuarios y oferentes de recursos se ha implementado a través de sistemas basados en sistemas ya existentes, lo que se ha dado en llamar "metasistemas" o "metacomputadores", y más recientemente "grids".

La tecnología Grid surge del nuevo paradigma de computación distribuida propuesto por Ian Foster y Carl Kesselman a mediados de los

90[21]. Es una tecnología innovadora que permite utilizar de forma coordinada todo tipo de recursos (entre ellos cómputo, almacenamiento y aplicaciones específicas) que no están sujetos a un control centralizado. En este sentido es una nueva forma de computación distribuida, en la cual los recursos pueden ser heterogéneos (diferentes arquitecturas, supercomputadores, entre otros) y se encuentran conectados mediante redes de área extensa como Internet.

Se basa fundamentalmente en el acceso remoto a recursos computacionales, y pretende ser un paradigma de desarrollo sin centrarse en una tecnología concreta. Su objetivo es permitir gestionar y distribuir la potencia de cálculo disponible, de tal forma que los usuarios se beneficien de la potencia de ordenadores infrautilizados que se encuentran dispersos geográficamente. Éstos ceden parte de sus recursos para procesar aplicaciones con funciones de cálculo avanzadas.

La evolución de "*Grid Computing*" se refleja en el avance de la estandarización de esta tecnología (el estándar de Globus Project es el estándar de facto) donde se encuentra definida la arquitectura del grid, los niveles de acceso, los requisitos, los servicios, etc.

Gracias a la tecnología Grid, el usuario puede utilizar toda la red o redes de computadores para solicitar la potencia que necesita de otros equipos que no trabajan al máximo de sus posibilidades para que les preste esos recursos sobrantes. De esta manera el usuario podrá acceder

a un supercomputador virtual con la potencia suficiente para realizar, de forma sencilla, los cálculos avanzados. Es decir, si el usuario necesita más memoria RAM y más disco duro para almacenar un trabajo no hace falta que utilice los recursos de su propio equipo, sino que puede utilizar el de los otros ordenadores del Grid.

Grid se diferencia de los sistemas cliente-servidor y otras tecnologías actuales (CORBA, EJB o .NET), en que está orientada a los recursos computacionales y no a la información, la seguridad no está en un segundo plano y la comunicación es asíncrona.

II.2.1 Definición de Grid Computacional

Existen muchas definiciones de Grid Computacional, dentro de las cuales citamos la dada por Ian Foster y Carl Kesselam: Un grid computacional es una infraestructura hardware y software que suministra al que la utiliza acceso seguro (dependable), consistente (consistent), penetrante (pervasive) y barato (inexpensive), a unas elevadas capacidades computacionales. [1]

El concepto infraestructura se utiliza porque un grid es un conjunto de recursos (ciclos de CPU, datos, sensores, etc.), y todos esos recursos necesitan una interconexión hardware y un control software para que estén ensambladas en un grid. El que debe presentar las siguientes características:

1. La coordinación de recursos no está sujeta a un control centralizado
1. Utiliza estándares abiertos
2. Ofrece una calidad de servicio no trivial [2]

II.2.2 Ventajas e Inconvenientes del GRID

El concepto de Grid Computacional da idea de una gran potencia de cálculo y almacenamiento, y es un gran avance en las ciencias de la computación. Esta infraestructura debe proporcionar a los usuarios un servicio seguro a todos los niveles: capacidad de cómputo, de integridad de datos, de seguridad de acceso, etc. El servicio debe ser consistente, basado en estándares, y de esta manera el acceso y las operaciones sobre el grid estarán definidas por dichos estándares, permitiendo la heterogeneidad y haciéndola transparente al usuario. Por último, el acceso y uso del grid debe tener un costo económico, que lo haga atractivo para que su utilización se universalice.

II.2.2.1 Beneficios del Grid

- Permite alquilar recursos según necesidad
- Amortización de recursos propios
- Gran potencia de cálculo a precio bajo sin necesidad de adquirir equipamiento
- Mayor colaboración y compartición de recursos entre varios centros
- Creación de organizaciones virtuales
- Modelo de Negocios basados en proveer recursos.

II.2.2.2 Desafíos

- Recursos heterogéneos. Deben integrar y saber administrar equipos con distintos CPUs, memorias, discos, etc
- Descubrimiento, selección, reserva, asignación, gestión y monitorización de recursos. Conocer los recursos que posee e identificar cuales son los mejores para cada requerimiento.
- Desarrollo de aplicaciones. Crear aplicaciones que sean capaces de aprovechar todas las funcionalidades y bondades que ofrece un grid, es una tarea compleja.
- Comunicación lenta y no uniforme: Los equipos están conectados a través de redes distintas velocidades, protocolos y niveles.
- Organizativos: dominios de administración, modelo de explotación y costes, política de seguridad. Los equipos pertenecen y son manejados por organizaciones y/o individuos diferentes (probablemente desconfiados entre sí) con diferentes políticas de seguridad y distintas practicas.
- Económicos: Precio de los recursos, oferta/demanda, desarrollo e implementación.

II.2.3 Requisitos indispensables de un Grid / Aspectos de Diseño

La idea de Grid es que sean arquitecturas fiables, consistentes y accesibles. Por ello, cinco características indispensables están presentes en ellas:

- Uniformidad: el usuario del Grid ha de ver los diferentes

recursos y datos como un único recurso

- Transparencia: los datos en los diferentes formatos y tipos de archivos han de ser integrados en una base de datos virtual, de manera que el usuario pueda manejarlos con independencia de la fuente de que provengan
- Fiabilidad: disponibilidad permanente, que exige especial cuidado en la tolerancia a fallos y la redundancia, con un almacenamiento y capacidad de red robustos
- Ubicuidad: recursos disponibles para la mayor cantidad posible de usuarios, lo que implica un *middleware* preparado para soportar una amplia diversidad de plataformas y sistemas operativos
- Seguridad: es fundamental tanto en los datos como en el acceso a los recursos compartidos.

II.2.4 Extensiones del Grid: Aplicaciones de Ejecuciónes Múltiples Simultáneas

Una clase ideal de aplicaciones para un grid computacional son las aplicaciones *Parameter Sweep(es decir, de Ejecuciones Múltiples Simultáneas)*, que se estructuran como un conjunto de múltiples experimentos secuenciales e independientes (que no se comunican ni intercambian datos), cada uno de los cuales es ejecutado con un grupo diferente de parámetros, es decir, con datos de entrada diferentes para cada una de ellas.

Por ser el grid computacional una plataforma prometedora para la ejecución eficiente de aplicaciones *Parameter Sweep* con grandes parámetros, debemos lograr el desempeño deseado en el Grid, para esto tales aplicaciones deben ser colocadas de manera que los archivos compartidos se encuentren estratégicamente ubicados y maximizar su uso, y que la ejecución de la aplicación pueda adaptarse a la potencial entrega de recursos heterogéneos, distribuidos y compartidos [3].

CAPÍTULO III

CASO DE ESTUDIO: SUMA/G

III.1 Definición de SUMA/G

SUMA/G (Scientific Ubiquitous Metacomputing Architecture on Globus) es una plataforma distribuida de ejecución de aplicaciones en bytecode de Java, tanto aplicaciones multi-hilos (en un único espacio de memoria) como paralelas, comunicándose a través de paso de mensajes.

El Globus Toolkit (GT) es una tecnología fundamental para el grid, la cual permite compartir capacidad de cómputo, bases de datos, y otras herramientas con seguridad en línea a través de límites corporativos, institucionales, y geográficos sin sacrificar la autonomía local. Ofrece servicios que pueden ser combinados para desarrollar infraestructuras de grid. Estos cubren servicios de administración y supervisión de recursos, seguridad, administración de archivos, comunicación y ejecución de trabajos [24].

SUMA/G se implementa como un sistema de objetos distribuidos y tiene entre sus objetivos principales la transparencia de localidad a las aplicaciones que sean ejecutadas en el sistema. Adicionalmente, SUMA/G provee acceso a bibliotecas nativas de alto desempeño para permitir el desarrollo de aplicaciones más eficientes.

SUMA/G posee las siguientes características:

- Capacidad de ejecución de aplicaciones compiladas a bytecode de Java, sobre un conjunto de recursos distribuidos
- Una interfaz de ejecución de programas sencilla y transparente. La visión de máquina virtual presentada al usuario es la de una máquina poderosa en su escritorio
- Servicios implementados en modo nativo, por ejemplo librerías de álgebra lineal como PLapack
- Adicionalmente, SUMA/G provee herramientas para el perfilado de desempeño (performance profiling) de aplicaciones, así como de monitoreo y administración de la máquina virtual

III.1.1 Objetivos de SUMA/G

El objetivo principal de SUMA/G es ofrecer acceso a recursos distribuidos de alto desempeño para ejecución de bytecode de Java, incluyendo código paralelo con comunicación a través de mpiJava. Las plataformas de ejecución de bytecode pueden contener envoltorios (wrappers) de funciones científicas nativas, optimizadas para la plataforma de hardware sobre la que corre, con el propósito de compensar la baja velocidad de ejecución que ofrecen muchos interpretadores de *bytecode*. Además se busca cumplir con otros objetivos:

- Aprovechar eficientemente las capacidades de los recursos con que se cuenta en una red local o global, poniendo al alcance de todos los

usuarios la velocidad de plataformas de cómputo intensivo

- Compartir recursos computacionales bajo convenios de cooperación entre universidades y centros de investigación
- Aprovechar el tiempo ocioso de la capacidad computacional instalada.

III.1.2 Descripción de SUMA/G

La arquitectura de SUMA/G está diseñada en tres capas o niveles (Ver figura Nº 2).

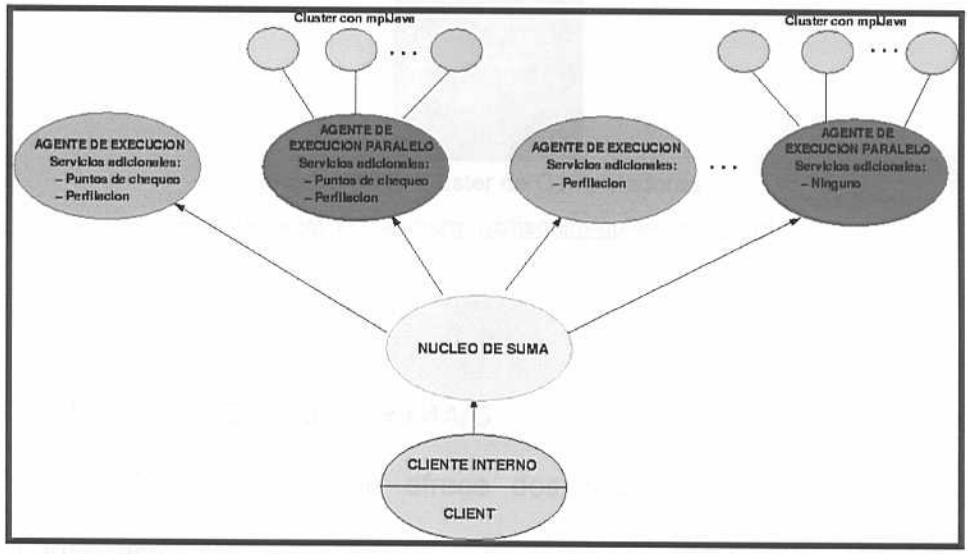


Figura Nº 2. Esquema de SUMA/G. Fuente: www.suma.ldc.usb.ve

- Los clientes, son aplicaciones de diversos tipos y funcionalidades que se comunican con el sistema de transporte, entregando la información necesaria para solicitar la ejecución de una aplicación.
- El Núcleo, se encarga de recibir las peticiones y encontrar un servidor de aplicaciones adecuado para la ejecución de cada petición, monitoreando el estado de los servidores y garantizando la ejecución de las aplicaciones. Basado principalmente en CORBA.

- Los agentes de ejecución, residen en servidores de ejecución, típicamente computadoras de alto rendimiento, donde finalmente se ejecutarán las aplicaciones de los usuarios. Comprenden máquinas de distintos tipos, tanto de memoria compartida (que incluyen las computadoras de un único procesador) como de memoria distribuida, por ejemplo clusters de computadoras (Ver figura Nº 3).

Figura Nº 3. Cluster de Computadoras.
Fuente: www.fisica.unam.mx/fisquim/guillermoweb.htm

III.1.3 Ejecución Básica de SUMA/G

SUMA/G actualmente ofrece dos modos de ejecución a sus usuarios, estos son:

- Ejecución en línea (*execute*), ofrece la ejecución remota de aplicaciones Java interactivas, secuenciales o paralelas. Este modo de ejecución requiere que el usuario a través del proceso cliente permanezca conectado al sistema de SUMA/G, ya que la carga de clases (a excepción de la clase *main*, que se especifica al inicio) y otros archivos de datos se leen desde el cliente en tiempo de ejecución. Las entradas y salidas son redireccionadas a la máquina del cliente desde el nodo remoto.

- Ejecución fuera de línea (*submit,* modalidad por lotes o *batch*), para este modo de ejecución no se requiere que el cliente se mantenga conectado a SUMA/G, pero requiere que todas las clases y archivos de entrada que necesite la aplicación sean empaquetados y enviados a priori para la ejecución remota. Los resultados de la ejecución pueden ser obtenidos luego por el cliente, a través del identificador de la ejecución que se le da al cliente al momento de realizar el *submit*.

El proceso de ejecución básica dentro del sistema de SUMA/G una vez que se le ha enviado el *main Class* del programa a ejecutar desde la máquina del cliente, sin importar el modo utilizado, funciona de la siguiente manera:

- Transparentemente SUMA/G encuentra un servidor (cluster o máquina) para la ejecución y envía un mensaje de petición al servidor
- Un Agente de Ejecución en el servidor es designado para que inicie la ejecución del programa, cargando dinámicamente las clases y archivos de entrada requeridos desde el cliente, como también enviando de regreso la salidas de la ejecución
- En caso de ejecuciones fuera de línea, la salida se queda en SUMA/G hasta que ésta sea requerida por el usuario

III.1.4 Componentes de SUMA/G

Para comprender mejor el funcionamiento de SUMA/G, a continuación se da una breve definición de cada uno de los componentes que forman parte de la arquitectura de SUMA/G.

III.1.4.1 SUMA/G Internal Client (Client Stub)

Es una librería para la implementación del Cliente de SUMA/G, que esconde a los clientes externos los detalles de comunicación con el resto de SUMA/G[4]. Provee los servicios para las ejecuciones tanto en línea como fuera de línea, de recuperación de resultados y de perfiles de rendimiento. De igual manera se encarga de crear y entregar las unidades de configuración (CU) que contienen información sobre el tipo de plataforma y los servicios solicitados por el usuario, necesaria para las ejecuciones; y atender las llamadas de los Agentes de Ejecución, para la carga de las clases y archivos dinámicamente. A continuación se muestra en la figura Nº 4, el diagrama de clases con el fin de especificar el comportamiento del componente:

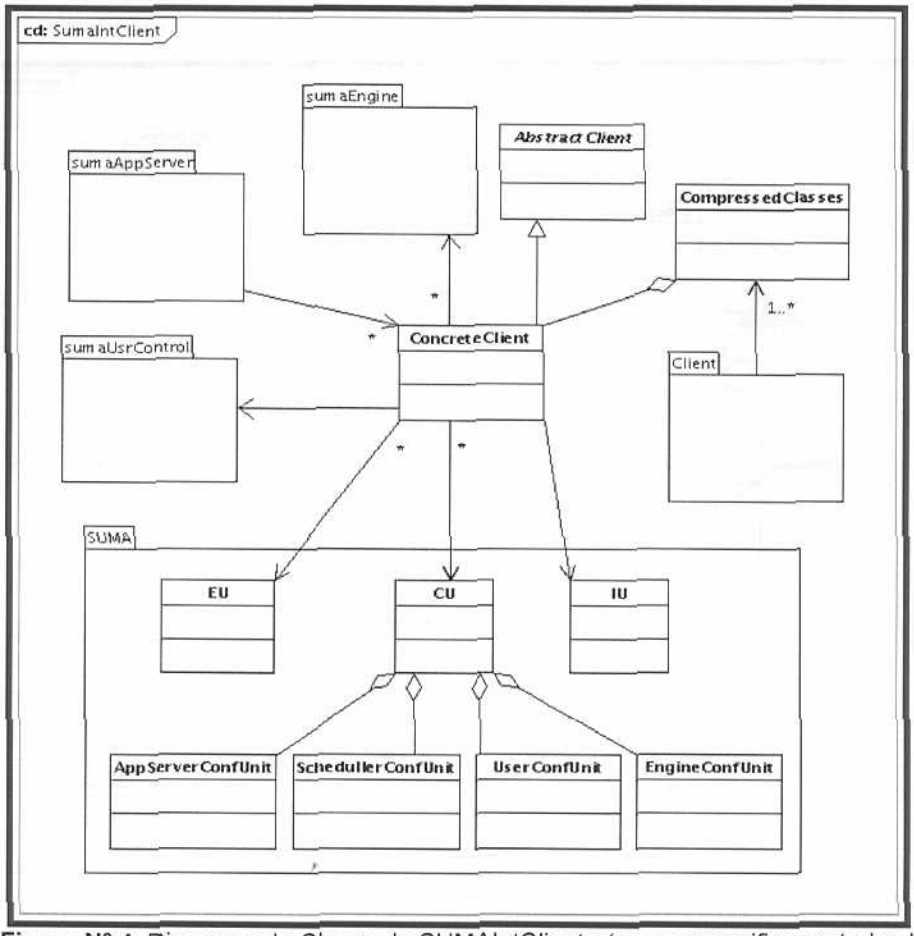


Figura Nº 4. Diagrama de Clases de SUMAIntCliente (no se especificaron todos los elementos de los paquetes con que interactúa).Fuente: Elaboración propia.

III.1.4.2 SUMA/G Engine

Es el encargado de coordinar las Ejecuciones enviadas por el cliente, preguntar al Scheduler (Planificador) por el servidor adecuado y entregarle a éste las unidades de configuración (CU), interactuar con el Application Server, manejar los resultados en caso de trabajos fuera de línea, conectar un cliente con el *Agente de Ejecución* de tal modo que estos componentes se comuniquen directamente entre sí (Ver figura Nº 5). El Engine es el despachador de todas las peticiones [4].

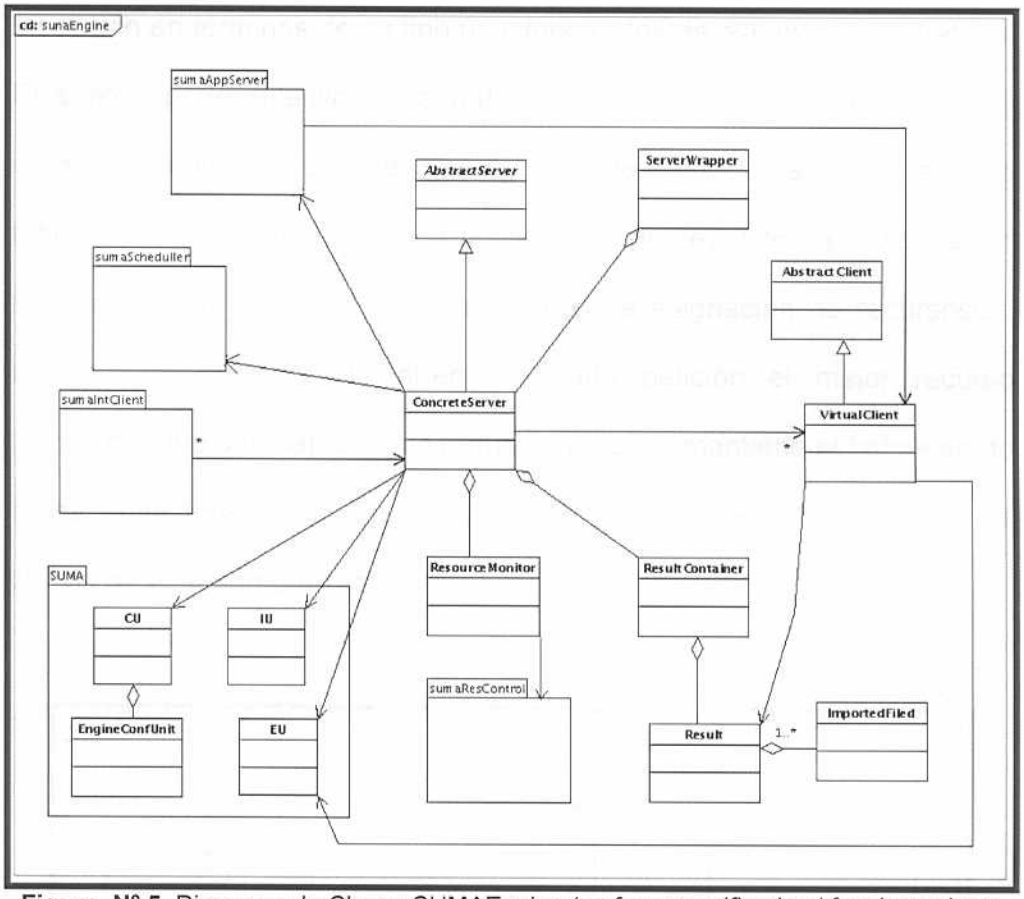


Figura Nº 5. Diagrama de Clases SUMAEngine (no fue especificado el funcionamiento interno de los paquetes que intervienen). Fuente: Elaboración propia.

III.1.4.3 SUMA/G Scheduler (Planificador)

El planificador tiene como principal objetivo asignar agentes de ejecución a los clientes que los solicitan, distribuyendo la carga equitativamente entre los agentes de ejecución. Adicionalmente implementa métodos para la selección del *Engine* y el *Agente de Ejecución*. Esta selección se basa en los requerimientos de la aplicación y en información de estado obtenida de las plataformas de ejecución. El *Planificador* mantiene información acerca de los recursos disponibles en

SUMA/G, como por ejemplo la descripción de las plataformas de ejecución en términos de su tipo (*clusters*, redes de estaciones de trabajo, PCs, etc.), poder relativo, tamaño de la memoria, bibliotecas disponibles y carga promedio. Los *Engine* y los *Agentes de Ejecución* se registran y se retiran ellos mismos en el *Planificador*, proveyendo la información necesaria para que éste tome decisiones de asignación de recursos[5]. De manera que permite ofrecer a cada petición el mejor recurso disponible que se adapte a sus requerimientos, y mantiene el balanceo de carga entre servidores. En la figura Nº 6 se muestra el diagrama de clases de este componente.

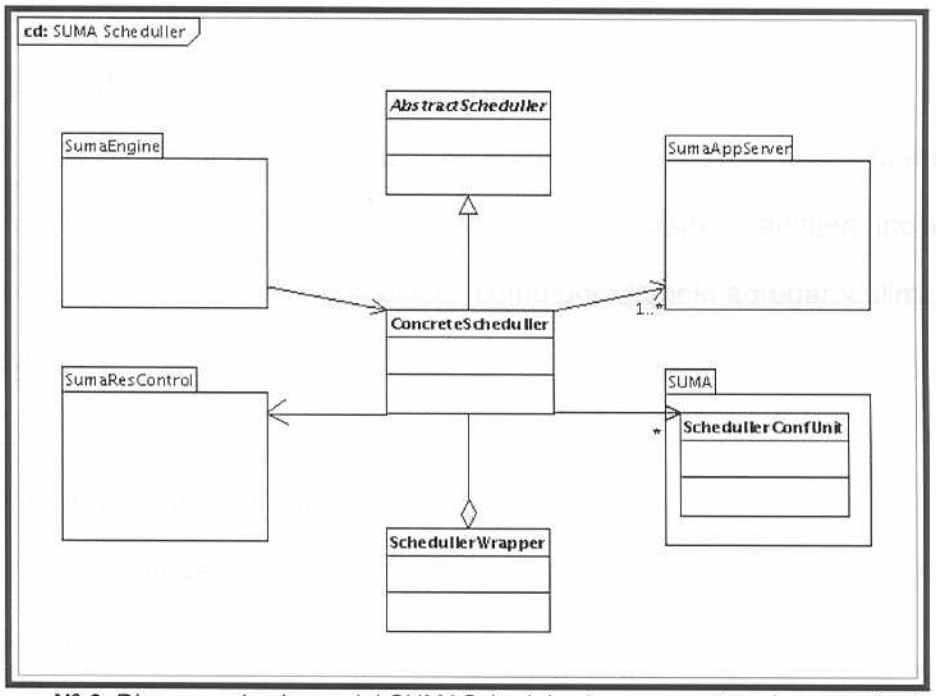


Figura Nº 6. Diagrama de clases del SUMAScheduler (no se muestra el comportamiento interno de los paquetes que son utilizados). Fuente: Elaboración propia.

III.1.4.4 SUMA/G Application Server

Su función es ejecutar concurrentemente, transparentemente y eficientemente las aplicaciones suministradas por los usuarios de SUMA/G[6]. Consiste en un coordinador y en varios Application Monitor Slaves. Recibe el estatus de los Agentes de Ejecución.

III.1.4.5 SUMA/G Resource Control

Es utilizado para el registro de los recursos de SUMA/G (servidores), manteniendo información estática y dinámica sobre los servidores, como tamaño de memoria, librerías disponibles, carga, etc[7].

III.1.4.6 SUMA/G User Control

Se encarga del registro y la autenticación de usuarios. Maneja *capabilities* para controlar el acceso a los recursos. También incluye métodos para tareas administrativas, como por ejemplo agregar y eliminar usuarios[5].

III.1.5 Funcionamiento de SUMA/G

A continuación se presenta una breve descripción, de algunas actividades importantes que permiten el funcionamiento de SUMA/G.

III.1.5.1 Localización de Agentes

Debido a que los componentes de SUMA/G son objetos CORBA, la localización está anclada en el servidor de nombres de la implantación de

CORBA utilizada. Inicialmente, algunos componentes del núcleo de SUMA/G, como el Scheduler y el agente de Control de Usuarios deben registrarse con el servidor de nombres, cuya combinación (Número-IP,Puerto) debe ser bien conocida por los componentes de SUMA/G. Los Agentes de Ejecución deben solicitar al servidor de nombres la referencia al *Scheduler*, para luego registrarse en éste. Los clientes de SUMA/G deben acudir al servidor de nombres para pedir la referencia al *Scheduler*, que elegirá un Agente de Ejecución y le pasará la referencia de éste al cliente. Los clientes realizan esta operación a través de un componente intermediario denominado *Engine*, pero una vez que el cliente tiene la referencia del Agente de Ejecución, la comunicación entre el cliente y el Agente de Ejecución será directa, sin intermediarios.

III.1.5.2 Ejecución multi-hilos y paralela

SUMA/G provee acceso a agentes de ejecución multi-hilos y paralelos. Un agente de ejecución multi-hilos, o simplemente agente de ejecución, ejecuta programas en un único espacio de memoria, y podría tener varios hilos de control. Un agente de ejecución paralelo es realmente una fachada o *front-end* de una máquina paralela o cluster de computadoras que funcione como una máquina paralela de memoria distribuida. En consecuencia, un agente de ejecución paralelo puede ejecutar programas paralelos cuyos procesos se comunican entre sí a través de paso de mensajes. De momento se dispone de un envoltorio o "*wrapper*" de MPI (*Message Passing Interface*) para Java, el cual se podría usar para ejecutar programas paralelos en SUMA/G. La ejecución

de programas paralelos está limitada a la disponibilidad de plataformas de ejecución paralelas fuertemente acopladas, como un cluster de computadoras o una computadora paralela. En el futuro se brindará soporte de comunicaciones para programas paralelos cuyos procesos se ejecutan en servidores distantes entre sí.

CAPÍTULO IV

MARCO METODOLÓGICO

En el presente capítulo se describe la metodología empleada para el desarrollo del Trabajo Especial de Grado. Esta metodología fue utilizada por ofrecer la facilidad de presentar prototipos al cliente, los cuales representan avances dentro del desarrollo que permiten obtener una retroalimentación de parte del mismo para asegurar el cumplimento de los objetivos planteados.

IV.1 Modelo Incremental:

Es un modelo evolutivo del proceso de desarrollo del Software, el cual fusiona el modelo lineal secuencial junto con la filosofía iterativa de construcción de prototipos. El modelo lineal secuencial, diseñado para el desarrollo en línea recta, de una secuencia de etapas de análisis, diseño, desarrollo y pruebas. Por su parte, el modelo de construcción de prototipos, diseñado para la comprensión de requisitos, se basa en, primero escuchar el cliente, y luego construir / revisar un prototipo y, posteriormente, la evaluación del mismo por parte del cliente. Este proceso se itera hasta alcanzar el objetivo esperado.

El Modelo Incremental aplica secuencias lineales de forma escalonada mientras progresa el tiempo. Con lo que se dice que el modelo Incremental entrega el software en partes pequeñas, pero funcionales, llamadas incrementos, y cada incremento se construye sobre

aquél que ya ha sido entregado [20].

Por lo tanto, para esta metodología, se tiene que:

- Cada secuencia lineal produce un incremento
- El primer incremento es un producto esencial (núcleo), se afrontan requisitos básicos y muchas funciones extras (conocidas o no) quedan para los siguientes incrementos.
- Es interactivo, al igual que el de construcción de prototipos y otros enfoques evolutivos. Pero a diferencia del modelo de construcción de prototipos, el modelo incremental entrega un producto operacional en cada incremento.
- El cliente usa el producto central y en base a la utilización y/o evaluación se desarrolla un plan para el incremento siguiente.
- Este proceso se repite hasta que se elabora el producto completo.

Cada incremento significa una funcionalidad nueva sobre el sistema. La fase de análisis de cada nuevo incremento coincide con la fase de diseño de la fase anterior, que es cuando se detectan los requisitos necesarios para comenzar la fase siguiente. Normalmente se aplica para sistemas cuyas funciones están fuertemente acopladas (Ver figura Nº 7).

Las etapas de cada secuencia Lineal son:

1.- Análisis de los requisitos: reunión y comprensión de requisitos y especificaciones del sistema.

2.- Diseño: estructura de datos, arquitectura del sistema, interfaz y detalle procedimental.

3.- Generación de Código: Programación del sistema.

4.- Pruebas: Evaluar el funcionamiento del sistema.

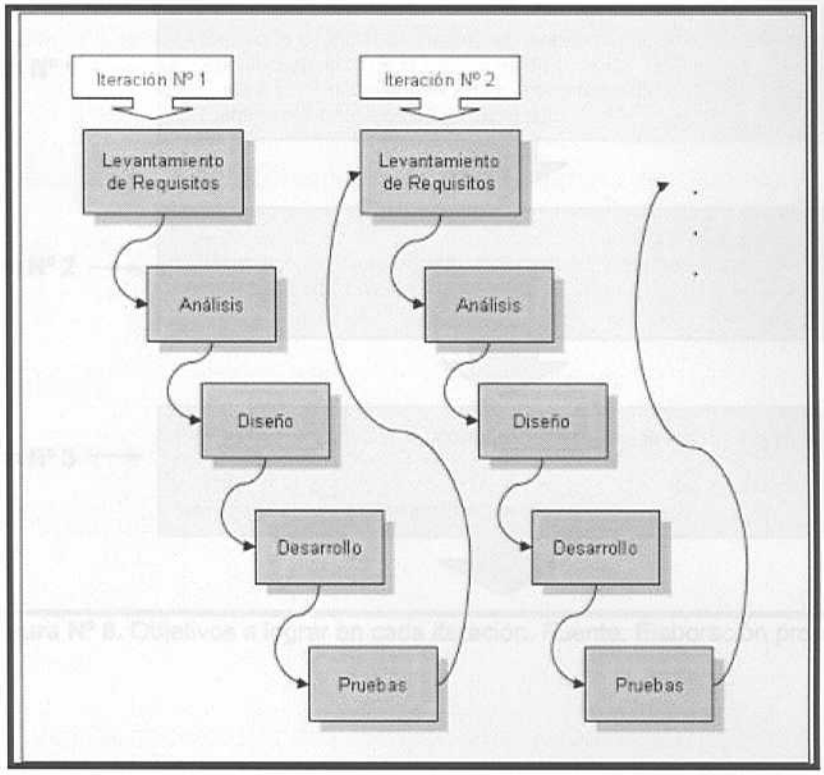


Figura Nº 7. Metodología Incremental. Fuente: Elaboración propia.

El presente Trabajo Especial de Grado constara de tres iteraciones, las cuales tendrán una serie de objetivos a lograr (Ver figura Nº 8). Estas iteraciones son las siguientes:

- Iteración Nº 1: Modificación de la estructura de SUMA/G para el manejo de Ejecuciones Parameter Sweep
- Iteración Nº 2: Creación de módulos que provean el control de las

instancias de ejecución

- Iteración Nº 3: Creación del Cliente Gráfico para la realización de ejecuciones Parameter Sweep.

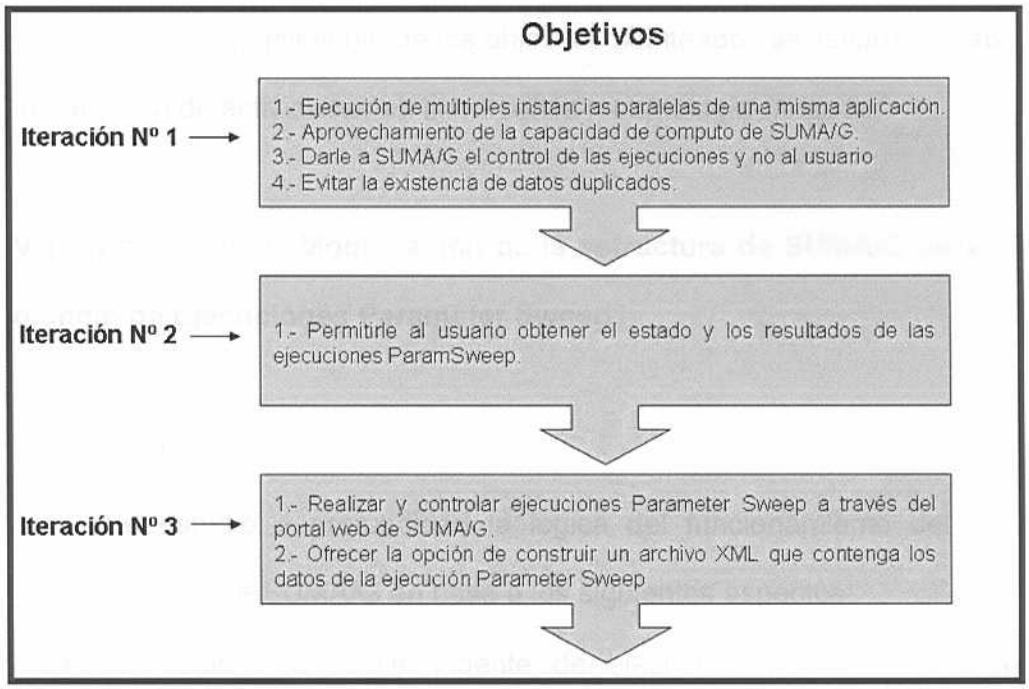


Figura Nº 8. Objetivos a lograr en cada iteración. Fuente: Elaboración propia

CAPÍTULO V

DESARROLLO DEL SERVICIO DE EJECUCIONES MÚLTIPLES SIMULTÁNEAS DE SUMA/G

Para el cumplimiento de los objetivos planteados se llevaron a cabo un conjunto de actividades en línea con la metodología utilizada.

V.1. Iteración Nº 1: Modificación de la estructura de SUMA/G para el manejo de Ejecuciones Parameter Sweep

V.1.1 Análisis

Se estudió a profundidad la lógica del funcionamiento del grid computacional de SUMA/G en base a los siguientes aspectos:

- Estructura: Schedulle, Agente de ejecución, Engine, Resource control
- Funcionamiento de cada componente: Información disponible en el Marco Referencial
- Comunicación, (ver Figura Nº 2)
- Ejecución del conjunto
- Resultados

Al realizar múltiples pruebas con el sistema de Ejecuciones Múltiples Simultáneas disponible en SUMA/G (inactivo), se observó que nunca se alcanzaba el resultado requerido por el cliente el cual es la ejecución de

requerimientos de manera distribuida, es decir, el procesamiento del requerimiento solo era realizado de manera local. Además, se replicaban los datos comunes de una ejecución tantas veces como instancias existieran.

Considerando los resultados antes explicados y las sugerencias del cliente, se descartó el sistema disponible en SUMA/G, siendo así necesario establecer una nueva solución que se adecuará a los requerimientos del cliente.

Actualmente SUMA/G maneja dos tipos de peticiones, que son las *Execute* y las *Submit*, estas se manejan utilizando el mismo *SUMAEngine*, nuestro objetivo es hacer que SUMA/G maneje las peticiones del tipo *parameter sweep*, pero tanto las peticiones en línea como las fuera de línea manejan una única instancia de la ejecución, y ambas son atendidas por el mismo *engine*, el cual está especializado para este tipo de ejecuciones, el objetivo entonces es crear un *engine* especializado en ejecuciones *parameter sweep* de manera que esté en la capacidad de coordinar y controlar varias instancias de una misma ejecución.

Se realizó el levantamiento de la información pertinente para conocer el funcionamiento y estructura de SUMA/G (Ver anexo Nº 1), de manera de establecer las principales características que debe poseer el servidor *Parameter Sweep* (el cual llamaremos *SUMAPSEngine)* dentro

del Grid de SUMA/G, para esto se debió profundizar en el manejo de las ejecuciones tanto en línea como fuera de línea que realiza cada uno de los componentes. Estableciendo la secuencia de invocaciones a métodos que se ejecutan dentro de SUMA/G para realizar un trabajo dado por un cliente.

Adicionalmente se observó el proceso de comunicación y registros de componentes en el servidor de nombres de CORBA donde se identificó que para cada uno de los servidores de SUMA/G se establece una clase encargada de crear la instancia del servidor correspondiente y registrar estos en el servidor de nombre de CORBA.

Partiendo del hecho de que las ejecuciones *Parameter Sweep*, son ejecuciones fuera de línea, al igual que las ejecuciones *Submit*, y por ende no necesitan que el cliente esté conectado para realizarse, se tomó el comportamiento de las ejecuciones *Submit* como referencia para establecer el funcionamiento del nuevo modo de ejecución, este comportamiento se puede observar en la figura Nº 9.

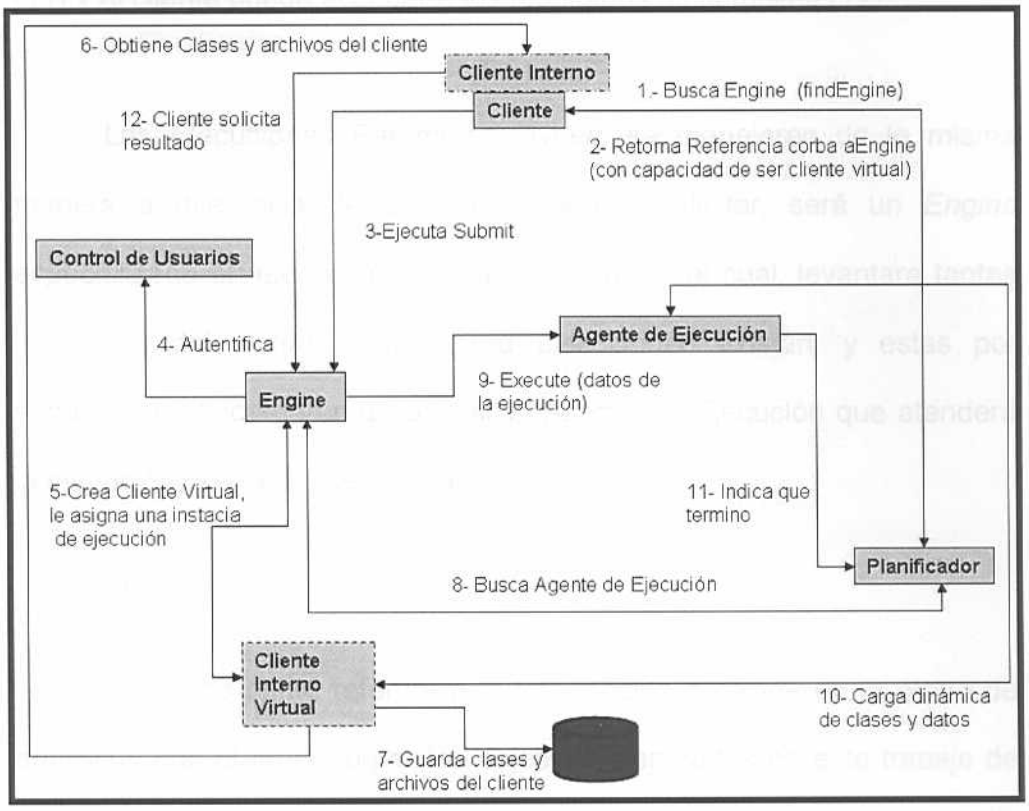


Figura Nº 9. Petición Submit. Fuente: Elaboración Propia.

Las Ejecuciones *Submit*, levantan un *Internal Client* el cual solicita al Planificador (Scheduler) el *Engine* que atenderá la petición(1), al recibir el id del *Engine*(2), le envía a este los datos de la ejecución(3), y luego él se encarga de autentificar al cliente(4), y de levantar un *Virtual Client*(5), quien solicita las clases y archivos necesarios al cliente interno (6) y los almacena(7), una vez realizado esto el *Engine* solicita al Planificador el Agente de Ejecución que llevará a cabo la ejecución(8), y le envía los datos de la misma, para que este los lleve a cabo(9). El Agente de Ejecución carga las clases en memoria dinámica (10) para procesar el requerimiento. Una vez terminado le indica al Planificador que culminó

(11) y el cliente puede acceder a los resultados de la misma (12).

Las Ejecuciones *Parameter Sweep*, se manejaran de la misma manera a diferencia de que, el *Engine* a solicitar, será un *Engine* especializado en ejecuciones *Parameter Sweep* el cual, levantara tantas instancias del *Virtual Client* como ejecuciones existan, y estas por separado se encargaran de buscar el Agente de Ejecución que atenderá la ejecución que se le fue asignada.

V.1.2 Diseño

Tomando como referencia la información obtenida en la etapa de análisis y con el fin de lograr los objetivos planteados en este trabajo de grado, se diseño la siguiente solución (Ver figura Nº 10), utilizando las herramientas proporcionadas por UML (Unified Modeling Language), que se describen en el anexo Nº 2. En la solución se establecieron modificaciones a los componentes de SUMA/G, como también la creación de un nuevo componente denominado *serverPSEngine*. A continuación, se explicará detalladamente el nuevo diseño establecido.

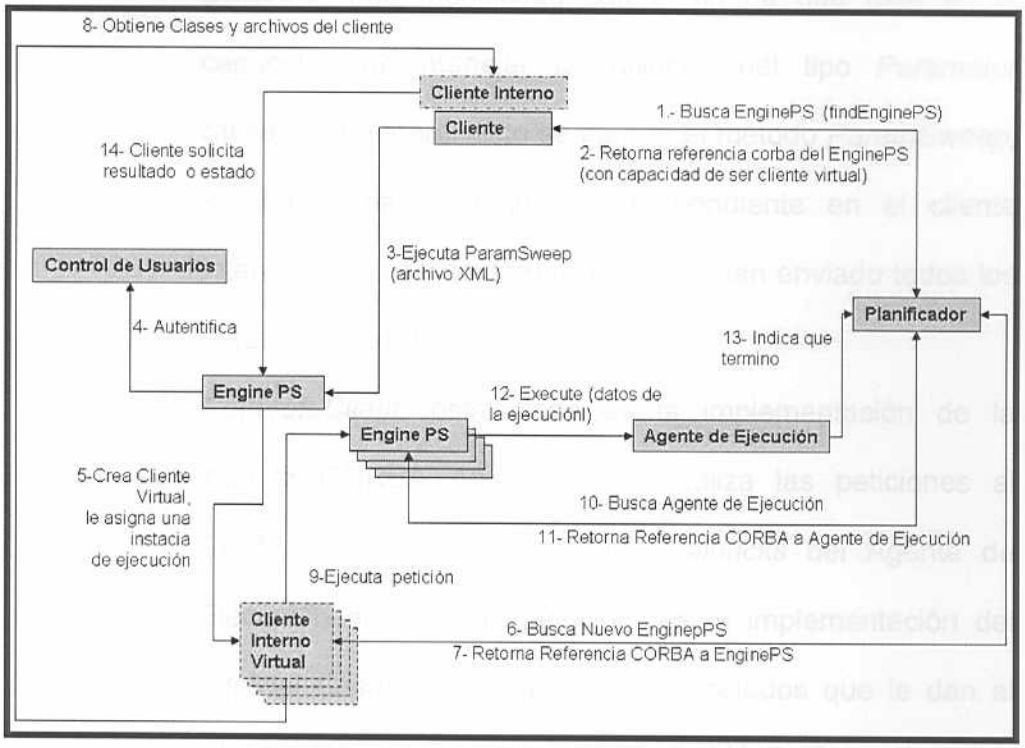


Figura Nº 10. Diagrama Petición Parameter Sweep. Fuente: Elaboración propia.

V.1.2.1 Componentes de SUMA/G modificados

Para lograr el cumplimiento de los objetivos planteados, es necesaria la modificación de los siguientes componentes.

V.1.2.1.1 SUMA/G Internal Client

Con el objetivo de que este componente reconozca las peticiones *Parameter Sweep* se deben modificar las siguientes clases:

1. *ClientWrapper*: funciona como la clase intermedia entre el cliente interno y externo, se encarga de establecer la comunicación entre ellos y de las validaciones de datos que son enviados al cliente interno. Esta clase es instanciada por el cliente externo, con el fin de invocar los servicios de

SUMA/G. Fue modificada con el fin de que esté en la capacidad de manejar ejecuciones del tipo *Parameter Sweep*. Para lograr esto se agregó el método *ParamSweep,* el cual llama al método correspondiente en el cliente Interno, luego de confirmar que se le han enviado todos los datos requeridos.

2. *ConcreteClient*: esta clase es la implementación de la interfaz CORBA *AbstractClient*, realiza las peticiones al SUMA/G *Engine* y atiende los *callbacks* del Agente de Ejecución durante la ejecución, es la implementación del *Internal Client*, por lo tanto ofrece métodos que le dan al cliente acceso a los servicios de SUMA/G. Para ofrecer el servicio de *Parameter Sweep*, se adicionaron los siguientes métodos a esta clase: *findEnginePS,* solicita al Planificador un *EnginePS* que esté disponible para manejar la petición; y el método *paramSweep*, carga las clases a utilizar y los respectivos archivos para ser enviados al *EnginePS* que se le fue asignado para atender la petición.

V.1.2.1.2 SUMA/G Engine

A pesar de que este componente en si no forma parte de la secuencia de llamadas para la ejecución de una petición *Parameter Sweep*, posee dentro de su conjunto de clases, una en particular con la cual se debe contar para la realización del trabajo de grado, la cual se describe a continuación:

1. *VirtualClient*: esta clase simula un cliente para que interactué con el *Application Server*, en el caso de las ejecuciones *Submit* donde el cliente real no está conectado al sistema. Fue modificada para que permita la ejecución *Parameter Sweep*, la cual también requiere la simulación de un cliente que interactúe con el *Application Server*. Las modificaciones se basaron en:

   a.- La creación de un constructor el cual esté en la capacidad de manejar las peticiones Parameter Sweep.

   b.- La modificación del método *run* de la clase (la cual extiende de *thread*), de manera que al momento de iniciar el hilo de la ejecución exista diferencia entre el manejo de las peticiones *Submit* y el de las peticiones *Parameter Sweep*.

   c.- La creación del método *findEnginePS*, que será instanciado por el método *run* si la petición es del tipo *Parameter Sweep*, con el fin de localizar a través del *Scheduler* el *EnginePS* que atenderá la ejecución.

V.1.2.1.3 SUMA/G Scheduler

Es importante que este componente controle el registro y el estado de cada servidor *Parameter Sweep (serverPSEngine)*, para que así pueda ofrecerle al cliente externo la posibilidad de solicitar este servicio (de Ejecuciones Múltiples Simultáneas) a SUMA/G. Para lograr esto se debe modificar la siguiente clase:

- *ConcreteScheduler*: esta clase es la encargada de

encontrar el mejor *Engine* y el mejor *Application Server* para atender los requerimiento del cliente, por lo tanto se le agregaron métodos, de manera que esté en la capacidad de atender peticiones *Parameter Sweep*, estos métodos son: *findEnginePS,* que retorna la referencia al *EnginePS* que esté disponible y que cumpla con los requerimientos del cliente externo; *findEnginebyPSId*, que retorna la referencia al EnginePS que está manejando la ejecución *Parameter Sweep* bajo el ID dado.

V.1.2.1.4 SUMA/G Application Server

Por ser este componente el encargado de llevar a cabo las ejecuciones enviadas por el cliente, debe estar en capacidad de manejar ejecuciones *Parameter Sweep,* por lo que en necesaria su modificación, las cuales son:

- *ConcreteNode*: es la implementación general del *Application Server*, se encarga de recibir y ejecutar las aplicaciones enviadas por el *Engine*, e inicializar un proceso por cada requerimiento. Por lo que es importante que en ella se encuentren los métodos necesarios para llevar a cabo las ejecuciones del tipo *Parameter Sweep*, estos son: *ExecutePS y intExecutePS*.

V.1.2.1.5 SUMA/G Resource Control

Las clases a modificar son:

- *ConcreteResource*: es la implementación de la interfaz CORBA *AbstractResource,* mantiene las instancias de los Agentes de Ejecución registrados en el sistema, por lo que se debió modificar, de manera que permita el registro de un servidor *Parameter Sweep* (*serverPSEngine*), para esto se agregó el método *registerEnginePS*.
- *EngineStructure*: esta clase mantiene la referencia del *Engine* y su información básica, se le agregó un nuevo constructor, de manera que también maneje *EnginePS*, y adicionalmente el método *getReferencePS* que permite obtener la referencia del *EnginePS.*
- *EngineMonitorListener*: se encarga de revisar el estado de los *Engine* registrados, y remover los que presente fallas. Con el objetivo de que también mantenga este control sobre los *EnginePS,* fue modificado el método *action* para que a su vez valide que los *EnginePS* registrados estén funcionando correctamente.

V.1.2.2 Nuevo componente ha ser agregado a SUMA/G

Dentro del diseño de la solución, se estableció la creación de un nuevo componente encargado del manejo de las Ejecuciones Múltiples Simultáneas (Parameter Sweep), en la figura Nº 11 se muestra el

diagrama de clases de este componente, el cual se llamará *ServerPSEngine*.

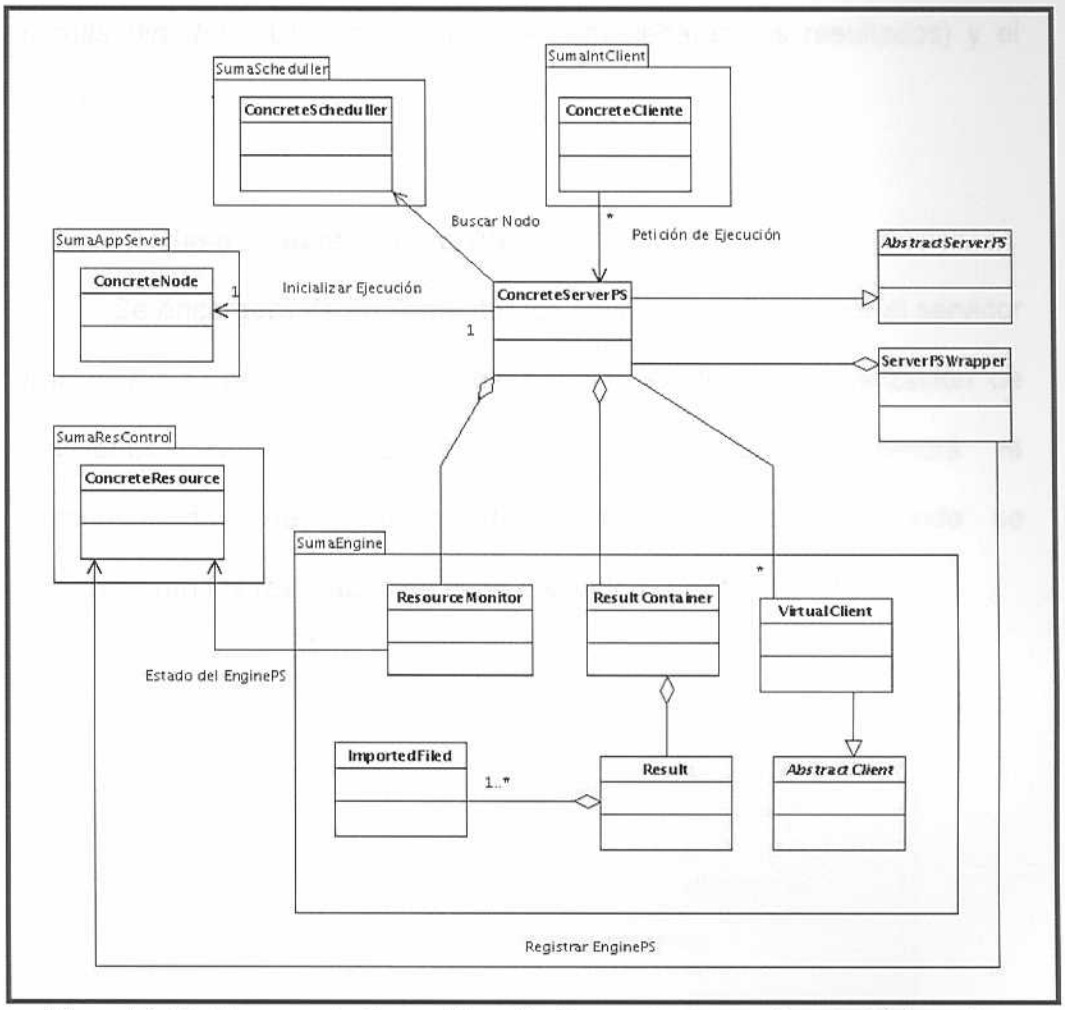


Figura Nº 11. Diagrama de Clases Ejecución Parameter Sweep. Fuente: Elaboración propia.

El *ServerPSEngine* estará estructurado de la siguiente manera:

V.1.2.2.1 Clase ServerPSWrapper

Esta clase será la responsable de levantar el *ServerPSEngine*, registrarla en el servidor de nombres e inicializar el ORB en el localhost,

para realizar esto es necesario un archivo de configuración (denominado PSengine.cfg) que contiene los parámetros necesarios, como el *EnginePort* (Puerto por el cual estará escuchando el servidor), el *results_directory* (Directorio donde se almacenarán los resultados) y el *log_file*.

V.1.2.2.2 Clase ConcreteServerPS

Se encargará de implementar los servicios a ofrecer por el servidor *Parameter Sweep*, de manera de permitir al cliente la realización de peticiones *Parameter Sweep*. Esta clase también tendrá la responsabilidad de crear el directorio de resultados, donde se almacenarán los resultados de cada ejecución (Ver figura Nº 12).

Figura Nº 12. Diagrama de la clase ConcreteServerPS. Fuente: Elaboración Propia.

V.1.2.2.3 Clase ParamSweep

Con el objeto de permitirle al cliente el acceso a los servicios de Ejecuciones Múltiples Simultáneas, se creará un cliente externo denominado *ParamSweep* encargado de recibir los parámetros necesarios para llevar a cabo la ejecución, estos parámetros deberán

estar contenidos en un archivo XML, cuya estructura se explica en la figura Nº 13. Adicionalmente, se encargará de validar la certificación del usuario, y de enviar al cliente interno la unidad de configuración de la ejecución.

Esta clase es la encargada de comenzar la ejecución de la petición. Para ejecutar una aplicación usando *Parameter Sweep* debe introducir, por línea de comandos, la siguiente orden:

> suma ParamSweep <Ruta Absoluta del Archivo .xml>

Ejemplo: suma ParamSweep DatosEjecucion.xml

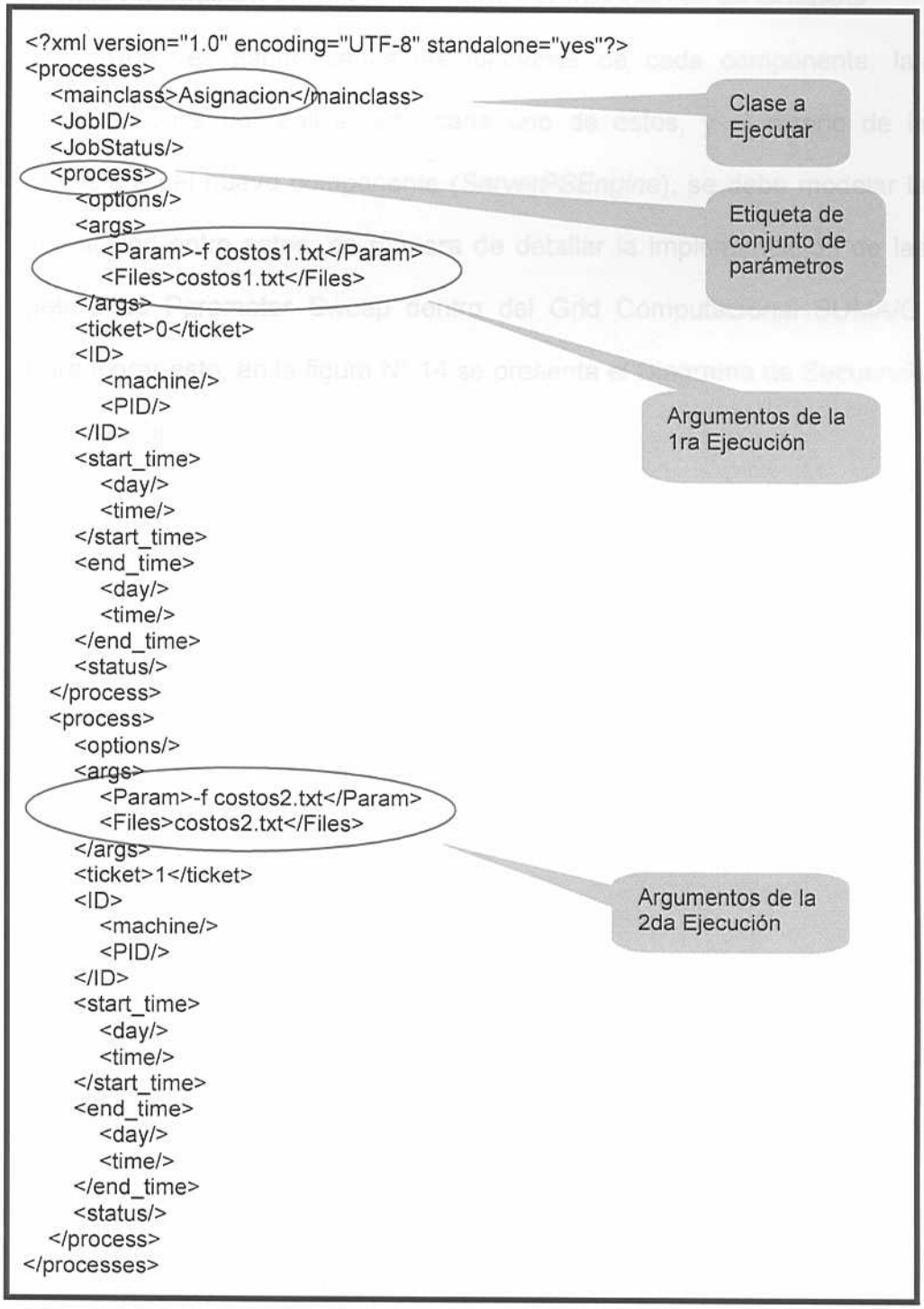


Figura Nº 13. Estructura del Archivo XML. Fuente: Elaboración propia.

V.1.2.3 Interacción entre los diferentes componentes de SUMA/G

Una vez especificadas las funciones de cada componente, las modificaciones ha realizar en cada uno de estos, y el diseño de la estructura del nuevo componente (*ServerPSEngine*), se debe modelar la interacción entre estos, de manera de detallar la implementación de las peticiones Parameter Sweep dentro del Grid Computacional SUMA/G. Para lograr esto, en la figura Nº 14 se presenta el Diagrama de Secuencia de la ejecución.

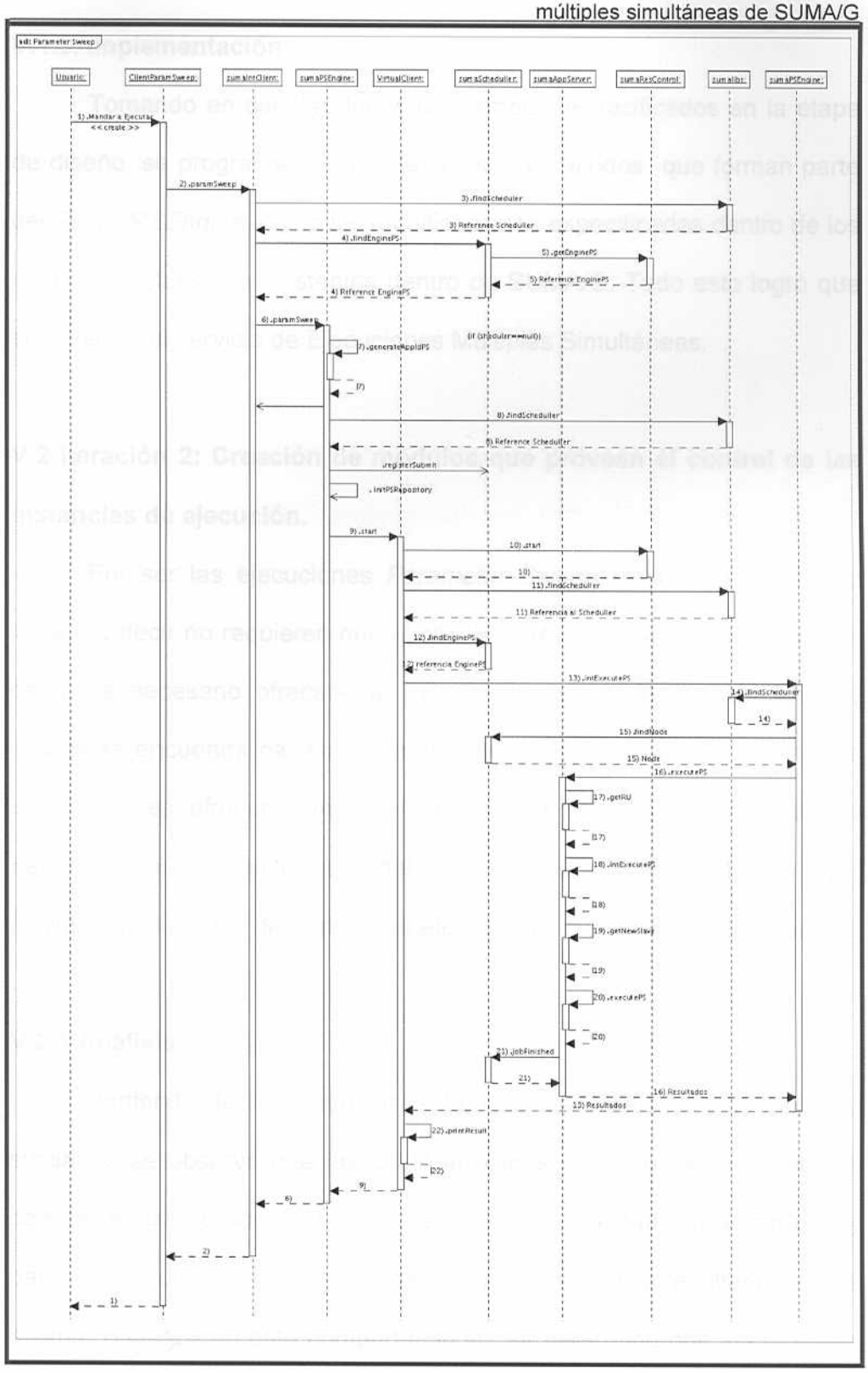


Figura Nº 14. Diagrama de Secuencia de la Ejecución Parameter Sweep. Fuente: Elaboración propia.

V.1.3. Implementación

Tomando en cuentas todos los cambios especificados en la etapa de diseño, se programaron tanto las clases y métodos que forman parte del *ServerPSEngine*, como las modificaciones especificadas dentro de los métodos y clases ya existentes dentro de SUMA/G. Todo esto logró que sea posible el servicio de Ejecuciones Múltiples Simultáneas.

V.2 Iteración 2: Creación de módulos que provean el control de las instancias de ejecución.

Por ser las ejecuciones *Parameter Sweep* ejecuciones fuera de línea, es decir no requieren que el cliente esté conectado para llevarse a cabo, es necesario ofrecerle al usuario la opción de conocer en que estado se encuentra cada una de las ejecuciones que envió, y aún más importante es ofrecerle una manera de obtener los resultados de su petición, y así dar al usuario del Grid Computacional SUMA/G mayor facilidad al momento de realizar las ejecuciones múltiples simultáneas.

V.2.1 Análisis

Partiendo de las ejecuciones *Submit*, las cuales utilizan métodos similares, se observó que los datos arrojados por estos al momento de solicitar el estado son: el ID de la ejecución y el estado de la misma; y para los resultados son: el ID de la ejecución y los resultados de la misma. Basados en este comportamiento, se determinó que los módulos de control para las ejecuciones paramSweep devolvieran los mismos datos para cada una de las sub-ejecuciones con su correspondiente

identificador.

V.2.2 Diseño

Se deben realizar modificaciones en los componentes de SUMA/G para permitir a los usuarios mayor control sobre sus ejecuciones. Las modificaciones serán explicadas a continuación.

V.2.2.1 Componente de SUMA/G ha ser modificado

La obtención del estado y del resultado de cada una de las instancias de la ejecución, requiere únicamente de la modificación del *SUMA/G Internal Client*, específicamente de dos de sus clases, estas son:

- ClientWrapper: En esta se agregarán los métodos *getStatusPS* y *getResultPS*, los cuales llaman a los métodos correspondientes del Cliente interno. Esto con el objetivo de que el usuario tenga acceso a estos servicios.
- ConcreteClient: Al igual que la clase anterior se le agregarán los métodos *getStatusPS* y *getResultPS*, encargados de obtener los resultados y el estado del *ServerPSEngine*.

V.2.2.2 Incorporación de los servicios *getStatusPS* y *getResultPS* al *ServerPSEngine*

Es preciso diseñar dentro del *ServerPSEngine* métodos, que permitan al usuario obtener el estado en que se encuentran cada una de las instancias, y también que le devuelvan los resultados de sus ejecuciones. Estos métodos deberán estar ubicados en este componente

debido a que éste se encarga del control y manejo de las ejecuciones del tipo *Parameter Sweep*.

Los métodos serán ubicados en la clase *ConcreteServerPS*, por ser esta la clase que implementa los servicios a ser ofrecidos al cliente, y serán los siguientes:

1. getStatusPS: obtendrá la información referente al estado actual en que se encuentra la instancia de ejecución (Esperando, Ejecutando o Finalizado).
2. getResultPS: será diseñado para obtener los resultados de cada una de las instancias de ejecución y enviárselos a la máquina del usuario que la solicite.

V.2.2.3 Creación de las clases JobStatusPS y JobResultPS

Se diseñarán dos clases, para que el cliente pueda tener acceso a estos servicios ofrecidos para las ejecuciones *Parameter Sweep*, estas clases son: *JobStatusPS* y *JobResultPS,* las cuales además de dar acceso al usuario a la información de las instancias de su ejecución, se encargaran de autentificar el cliente, con el fin de no permitir el acceso a usuarios no certificados y adicionalmente recibirá y enviará al *ServerPSEngine*, los parámetros necesarios para la realización de la tarea.

Estas clases serán las encargadas de comenzar la ejecución de la petición para obtener el estado y los resultados de una ejecución

Parameter Sweep. Para obtener esta información deberá introducir, por línea de comando, las siguientes órdenes:

- Para obtener el estado de la ejecución:

>suma JobStatusPS <ID de la ejecución>

Ejemplo: suma JobStatusPS 20060914-115512-Asignacion

- Para obtener los resultados de una ejecución:

> suma JobResultPS <ID de la ejecución> -d <Directorio donde se almacenaran los datos>

Ejemplo: suma JobResultPS 20060914-115512-Asignacion -d /home/juan

V.2.2.4 Diagrama de Secuencia

A continuación se presenta la secuencia de llamadas a métodos que se llevarán a cabo para obtener el estado y el resultado de las instancias de ejecución. Ver figura Nº 15 y Nº 16.

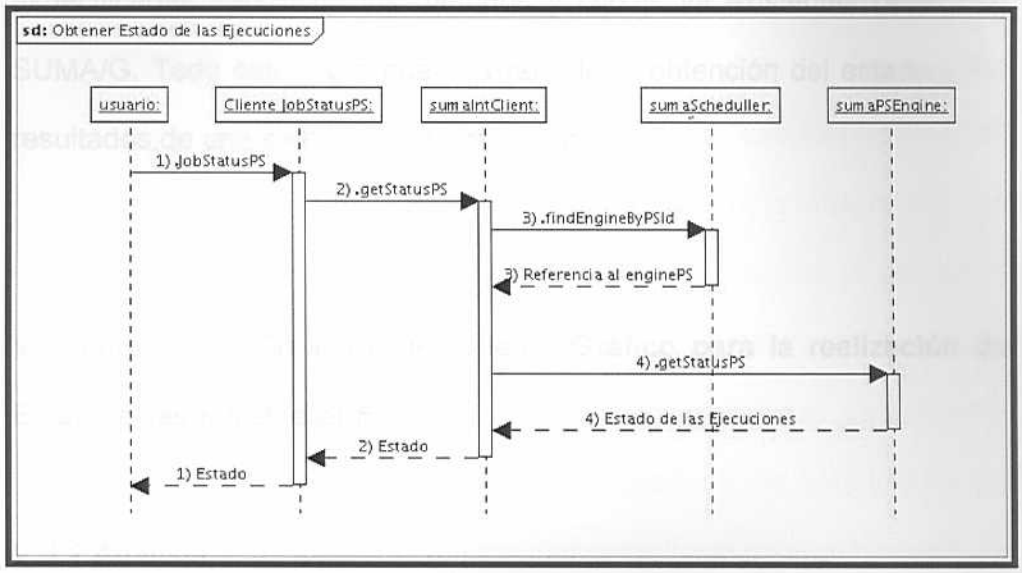


Figura Nº 15. Diagrama de Secuencia de la petición JobStatusPS. Fuente: Elaboración propia

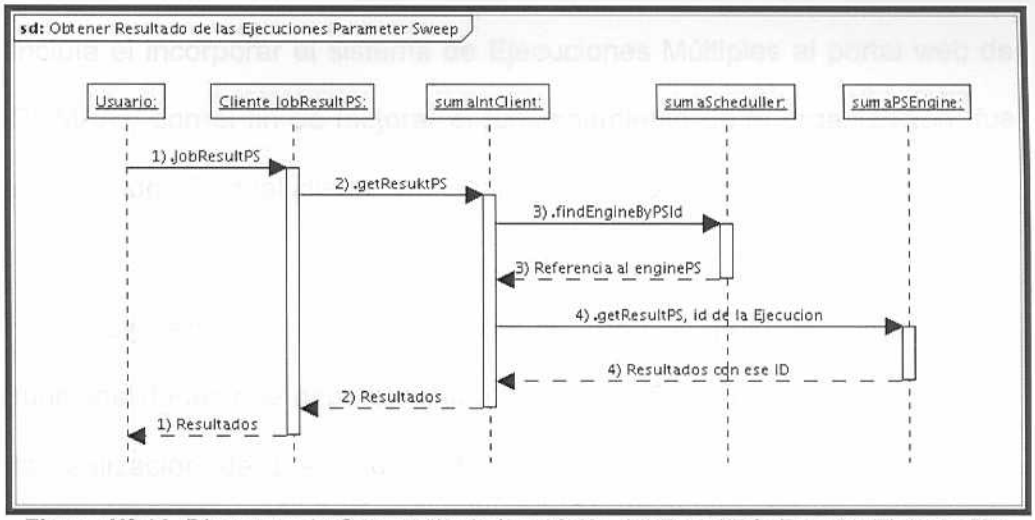


Figura Nº 16. Diagrama de Secuencia de la petición JobResultPS. Fuente: Elaboración propia.

V.1.3. Implementación

Tomando en cuentas todos los cambios especificados en la etapa de diseño, se programaron tanto las clases y métodos que forman parte del *JobStatusPS* y del *JobResultPS*, como las modificaciones

especificadas dentro de los métodos y clases ya existentes dentro de SUMA/G. Todo esto logró que sea posible la obtención del estado y los resultados de una ejecución ParamSweep.

V.3 Iteración 3: Creación del Cliente Gráfico para la realización de Ejecuciones Parameter Sweep

V.3.1 Análisis

Se observó el cliente gráfico planteado en una Tesis Especial de Grado de la UCAB [1] (Ver anexo Nº 3), pero tomando en cuenta que dentro de los objetivos planteados en presente Trabajo de Grado, se incluía el incorporar el sistema de Ejecuciones Múltiples al portal web de SUMA/G, con el fin de mejorar el funcionamiento de la organización, fue descartado el actual cliente gráfico, por no ofrecer esta versatilidad.

Siguiendo las especificaciones dadas por el cliente, las funcionalidades que debe cumplir el cliente gráfico para permitir al usuario la realización de Ejecuciones Múltiples Simultáneas, son las mismas funciones que se realizan a través de la ejecución por línea de comandos mas un agregado que es la generación del archivo XML que se utiliza para especificar los datos necesarios para ejecución, estas son:

1. Generar el archivo XML: para la realización de ejecuciones *Parameter Sweep* es necesario la construcción de un archivo XML, el cual contenga los parámetros de cada instancia de la

ejecución, con el objetivo de facilitarle el trabajo al usuario, la interfaz gráfica ofrece la opción de generarlo. El formato del archivo XML y la generación del mismo, ya estaban creados en SUMA/G, por lo que fueron reutilizados para la implementación de esta funcionalidad. A continuación se muestra el Diagrama de Casos de Uso de esta funcionalidad. Ver anexo Nº 4.

2. Realizar la Ejecución Parameter Sweep: una vez especificado el archivo xml que contiene los datos de las ejecuciones (la clase a ejecutar, la cantidad de ejecuciones y los parámetros de los mismos) y la clave correspondiente al certificado del cliente, éste se manda a ejecutar. Ver anexo Nº 5.
3. Obtener el estado de las ejecuciones: dado el id de la ejecución y la clave del certificado del cliente, esta funcionalidad muestra al usuario el estado en que se encuentra cada instancia de la ejecución. Ver anexo Nº 6.
4. Obtener el resultado de las ejecuciones: dado el id de la ejecución Parameter Sweep, la clave del certificado del cliente y el directorio donde desea almacenar los resultados, estos serán almacenados en la computadora del cliente. Ver anexo Nº 7.

A continuación, con el objeto de conocer cómo será la iteración del usuario con el Cliente Gráfico se muestra, el diagrama de casos de uso (Ver figura Nº 17):

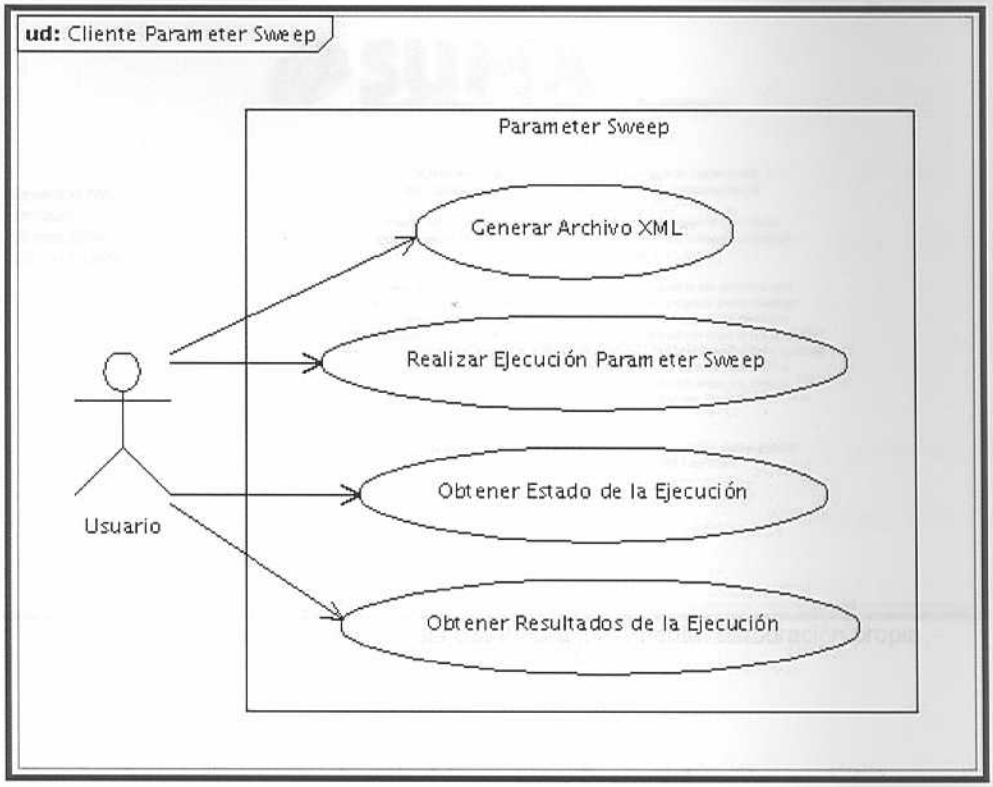


Figura Nº 17. Diagrama de casos de uso del Cliente ParamSweep (Para observar la descripción de cada caso vea anexos Nº 4 al 7). Fuente: Elaboración propia.

V.3.2 Diseño

De acuerdo a los requisitos dados por el cliente se diseño la interfaz gráfica. Este diseño preliminar, incluyo las funcionalidades antes descritas y la iteración con los servicio de SUMA/G. A continuación en la figura Nº 18, se muestra la página principal. En el anexo Nº 8 podrá observar el resto de las páginas que forman parte del cliente gráfico.

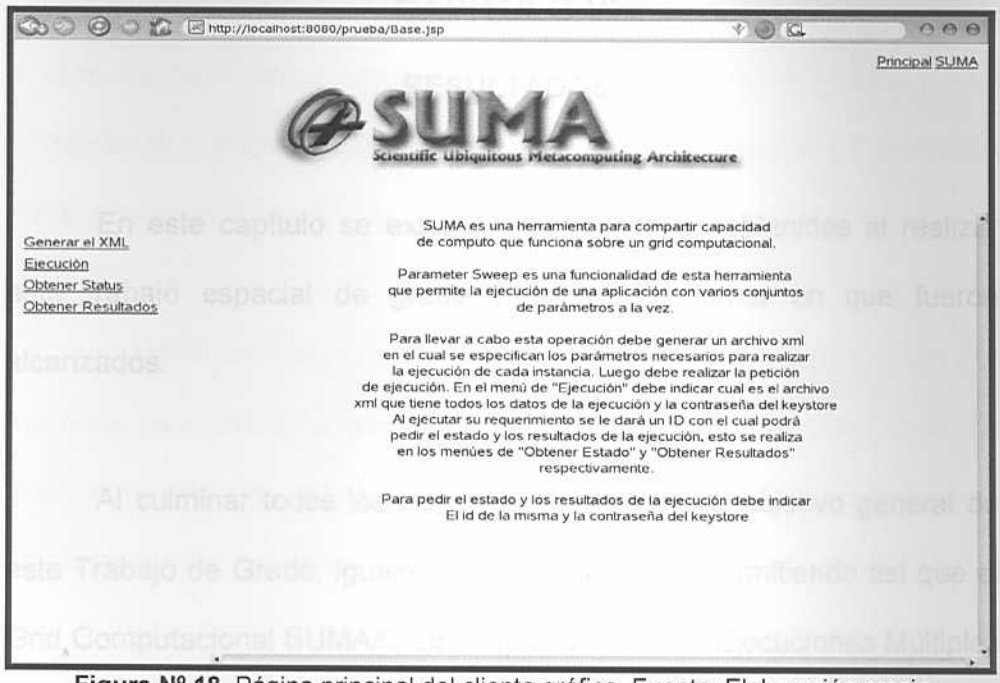


Figura Nº 18. Página principal del cliente gráfico. Fuente: Elaboración propia

V.3.3 Implementación

Se programo la interfaz establecida previamente utilizando la tecnología Java para la creación de páginas web (JSP, Java Server Pages), junto con la tecnología Servlets (objetos de Java que implementan la interfaz javax.servlet), ambas utilizadas para el desarrollo de páginas web.

CAPÍTULO VI

RESULTADOS

En este capítulo se exponen los resultados obtenidos al realizar este trabajo espacial de grado y explica la forma en que fueron alcanzados.

Al culminar todos los objetivos específicos, el objetivo general de este Trabajo de Grado, igualmente, es alcanzado, permitiendo así que el Grid Computacional SUMA/G sea capaz de realizar Ejecuciones Múltiples Simultáneas.

En primer orden, se instaló y probó El Grid Computacional SUMA/G para después realizar los distintos análisis con el fin de hacer un nuevo planteamiento para la realización del las ejecuciones múltiples simultáneas. Se le hizo un seguimiento al modo en que SUMA/G realiza las ejecuciones para así entender su funcionamiento y poder dar la mejor solución. Esta solución es explicada en el Capítulo V.

Después de entender el funcionamiento de SUMA/G se procedió a realizar el diseño de las funcionalidades a agregar, este diseño utilizó algunas funcionalidades ya existentes y algunas otras tuvieron que ser adaptadas a esta nueva implementación, además se agregaron nuevas para poder soportar este modo de ejecución.

En este nuevo diseño se usa un archivo xml en el cual se pasan las clases, los parámetros, archivos necesarios, por el cliente o usuario, para la realización de la ejecución deseada. Este archivo xml se encuentra explicado en la figura Nº 13.

Cuando se realiza una ejecución de múltiples instancias, cada una de ellas tiene archivos que solo le competen a una de esas instancias pero también hay clases que son comunes para todas ellas, en esta implementación las clases comunes son almacenadas en un directorio al que pueden acceder todas las instancias y así compartirlas. Esto nos permite ahorrar espacio en el disco puesto que no se copian para cada instancia todas las clases necesarias sino que se hace una sola copia que todas pueden utilizar.

A cada instancia se le crea un directorio particular en donde se guardan los datos, parámetros y archivos que necesita y solo le importa a ella. En este directorio también se almacenan los resultados de la ejecución de la instancia.

Para mejorar la apariencia y la usabilidad de esta funcionalidad, además de poder ser ejecutada por línea de comandos, fue creada una interfaz web, por medio de la cual se puede realizar no solo la ejecución antes mencionada sino también se puede generar a través de ella el archivo xml que se desee utilizar. Esta interfaz también es capaz de realizar otras funcionalidades como lo son el obtener el estado y los

resultados de una ejecución dada. Inicialmente se planteó adaptar la interfaz anterior a las nuevas necesidades pero al necesitarse que esta funcionalidad pudiera ser utilizada a través del portal web de SUMA/G se decidió implementar una nueva, puesto que la existente no estaba hecha para ser utilizada en este ambiente. Logrando así darle mayores facilidades al usuario, al momento de realizar este tipo de ejecuciones.

El cliente gráfico creado no fue incorporado al portal web de SUMA/G, debido a que este no se encuentra accesible a los usuarios, pero de igual manera su implementación es funcional, y puede ser incorporado una vez que el portal entre en funcionamiento.

Al comparar las ejecuciones secuenciales con las ejecuciones múltiples simultáneas, en base a tiempo de respuesta, usabilidad, control de la ejecución y eficiencia, se obtuvo que las aplicaciones ParamSweep arrojan mejores resultados, como se muestra en la figura Nº 19.

	Tipo de Ejecución	
Variable	**Secuenciales**	**ParamSweep**
Tiempo de Respuesta	Se procesan más lento que una ejecución ParamSweep. (Ver figura Nº 20).	Se procesan más rápido que una ejecución secuencial. (Ver figura Nº 20).
Usabilidad	El cliente debe hacer una petición de ejecución por cada conjunto de parámetros que desea proesar.	El cliente solo hace un petición de ejecución en la que, a través del arcivo xml, los conjuntos de parámetros que desea procesar.
Control de la Ejecución	Esta en manos del cliente porque es el el que debe ordenar a SUMA/G realizar las ejecuciones.	Esta en manos de SUMA/G, porque es SUMA/G el que ordena la realización de las ejecuciones.
Espacio en el Disco Duro Utilizado	Si se realizan varias ejecuciones de una misma aplicación, se copia en el servidor de suma las clases, tantas como número de peticiones haya realizado, que necesita para cumplir con el requerimiento	No importa cuantas peticiones tenga, implicitamente, una ejecución, SUMA/G solo copiará, en el servidor, las clases comunes una vez y cada peticion las tomará de alli cuabdo la necesite.

Figura Nº 19. Cuadro comparativo de las ejecuciones secuencias versus las ejecuciones múltiples simultáneas. Fuente: Elaboración propia

Se realizó un conjunto de pruebas que consistía en hacer la petición de ejecución de varios programas de forma secuencial y luego una sola petición paramSweep con la misma cantidad de instancias como ejecuciones de forma secuencial se realizaron para medir el tiempo utilizado por cada tipo de ejecución.

	Mediciones (HH:MM:SS,ns)						
	1	2	3	4	5	6	7
Comienza la ejecución del Script y las ejecuciones Submit	17:34:15,63	17:39:59,08	17:40:42,28	17:42:43,27	17:43:07,34	17:49:09,18	17:49:31,06
Finalizan las ejecuciones secuenciales del Submit y empieza la ejecución ParamSweep	17:34:20,12	17:40:03,99	17:40:46,87	17:42:47,40	17:43:11,62	17:49:14,62	17:49:35,39
Finaliza la ejecución ParamSweep y el Script	17:34:22,44	17:40:06,43	17:40:49,03	17:42:50,39	17:43:13,82	17:49:17,61	17:49:37,47
Duración de las Ejecuciones							
Secuenciales	00:04,50	00:04,91	00:04,58	00:04,13	00:04,28	00:05,45	00:04,33
ParamSweep	00:02,32	00:02,44	00:02,16	00:02,99	00:02,20	00:02,98	00:02,08

Promedios (HH:MM:SS,ns)	
Tiempo transcurrido para las ejecuciones Secuenciales	00:00:04,60
Tiempo transcurrido para la ejecución ParamSweep	00:00:02,45

Figura 20. Tabla Comparativa del Tiempo de Ejecución entre ejecuciones secuenciales y una ParamSweep.

CAPITULO VII

CONCLUSIONES Y RECOMENDACIONES

Gracias al presente Trabajo Especial de Grado, SUMA/G ofrece la ejecución de aplicaciones Parameter Sweep, además de los servicios de ejecuciones que ya tenía (ejecuciones en línea y fuera de línea), esto mediante la inclusión de una nueva petición de ejecución llamada ParamSweep, a SUMA/G, logrando así la ejecución simultánea de un programa o aplicación con diferentes conjuntos de parámetros de entrada, creando varias sub-ejecuciones de la original para realizar la tarea asignada. Este servicio brinda mayor facilidad a los usuarios a la hora de necesitar ejecutar una aplicación numerosas veces con diferentes parámetros de entrada, además de permitirle monitorear el estado de cada una de las ejecuciones.

La realización de este tipo de ejecuciones en el grid computacional de SUMA/G puede realizarse tanto por línea de comandos como a través del portal web de SUMA/G, esta ultima, gracias a la implementación de un cliente gráfico, el cual además de facilitar al usuario la ejecución de aplicaciones ParamSweep, y la obtención del estado y resultados de cada una de las sub-ejecuciones, le facilita la generación de un archivo XML necesario para establecer los parámetros de cada sub-ejecución.

La ejecución Parameter Sweep, ofrece al usuario mayor eficiencia en el momento de almacenar los datos necesarios para la realización de

cada una de las ejecuciones, puesto que, se evita las duplicaciones de clases, inevitables si se realizara cada una de las ejecuciones por separado. Adicionalmente, permite mantener una mejor organización de los datos y resultados que son almacenados por cada Cliente Virtual, gracias a la inclusión de directorios particulares para cada sub-ejecución.

La metodología de trabajo actual de la organización, permite a sus usuarios tener acceso a los servicios de ejecución Parameter Sweep de SUMA/G, a través de su portal web, representando esto una mejora en comparación con el sistema anterior, el cual no le daba a los usuarios esta opción, obligándolos a construir manualmente el archivo XML y a realizar la petición de ejecución ParamSweep por líneas de comando, al igual que las peticiones de estado y resultado de cada ejecución.

La implementación de este nuevo servicio de ejecución, evita que el usuario deba generar tantas ejecuciones submit como sub-ejecuciones de la aplicación necesite, permitiéndole así, con una única ejecución ParamSweep, obtener los resultados de cada sub-ejecución. Con esto el usuario realiza todas las ejecuciones más rápido, puesto que solo debe realizar una ejecución sin importar cuantas sub-ejecuciones necesite y a demás le permite monitorear el estado de cada ejecución con el uso de un único identificador, el cual es almacenado automáticamente en un archivo, para evitar que este se pierda. De esta manera el usuario obtiene un servicio más eficiente que cumple con sus necesidades.

Basados en los resultados obtenidos al momento de comparar las ejecuciones secuenciales versus las ejecuciones múltiples simultáneas, podemos decir que el uso de este tipo de aplicaciones (paramSweep), al momento de ejecutar varias instancias de una misma aplicación con diferentes parámetros de entrada, ofrece mejores resultados al usuario además de darle mayores facilidades, delegándole el control de cada ejecución a SUMA/G.

Recomendaciones

Cuando se realizan las ejecuciones Parameter Sweep, los datos de las ejecuciones son enviados en un archivo xml, el cual dentro de su esquema permite especificar para cada ejecución la máquina que la ejecuta, el tiempo de inicio y fin, estos datos deben ser completados al momento de realizar la ejecución. Por lo tanto, se recomienda que estos valores sean introducidos en el archivo xml, por el *enginePS* encargado de cada ejecución de manera, de poder ofrecer al usuario estos datos.

Dentro de SUMA/G cada una de las sub-ejecuciones tienen asignadas un identificador unico, aunque al usuario solo se le indica el identificador de la ejecución ParamSweep general. Se recomienda, con el fin de darle al usuario mayor control sobre cada una de las ejecuciones, que se le sean dados los identificadores de cada sub-ejecucion, y la posibilidad de acceder a los resultados de cada una de ellas por separado.

Referencias Bibliográficas

[1] Ian Foster y Carl Kesselam. *The Grid Blueprint for a Future Computing Infraestructura* (capítulo Computational Grids)

[2] Ian Foster. *What is the Grid? A Three Point Checklist*

[3] Henri Casanova, Graziano Obertelli, Francine Berman, Richard Wolski. "*The AppLeS Parameter Sweep Template:User-Level Middleware for the Grid*". Computer Science and Engineering Department University of California, San Diego USA
Disponible en: http://sc2000.org/techpapr/papers/pap.pap169.pdf

[4] Baquero, Eduardo; Berbin, Luis; Bouza, Roberto; García, Pedro. *SUMA-Scientific Ubiquitous Metacomputer*. Proyecto de Grado, ingeniería de la Computación. Universidad Simón Bolívar. 2000.

[5] Universidad Simon Bolivar. *SUMA*. Disponible en: www.suma.ldc.usb.ve/descripcion/index.html

[6] Kryzzambar Mota Peñaloza (2004). *Incorporación de Servicios de Ejecuciones Múltiples Simultáneas a un Grid Computacional*. Trabajo de Grado, Ingeniería Informática, Universidad Católica Andrés Bello, Caracas.

[7] Cardinale, Yudith; Figueira, Carlos; Hernández, Emilio. *SUMA: A Scientific Metacomputer*. Universidad Simon Bolívar, Caracas.

[8] W3C - World Wide Web Consorsium. *"Guía Breve de Tecnologías XML"*. España. Extraído el 16 de septiembre del 2005 en el sitio web: http://www.w3c.es/Divulgacion/Guiasbreves/TecnologiasXML

[9] Wikipedia, La Enciclopédia Libre. "*XML*". Extraído el 15 de septiembre del 2005 en el sitio web http://es.wikipedia.org/wiki/XML

[10] Alfredo Reino Moreno(2000). "*Introduccion a XML en castellano*"

[11] W3C - World Wide Web Consorsium. "*Extensible Markup Language (XML)*. Extraído el de septiembre del 2005 en el sitio web: http://www.w3.org/XML/

[12] SUN. "*El API JAXB*". Extraído el de septiembre del 2005 en el sitio web: http://www.programacion.com/html/tutorial/jaxb/2/

[13] Wikipedia, la Enciclopédia Libre. "*CORBA*". Extraído el 6 de noviembre del 2005 en el sitio web: http://es.wikipedia.org/wiki/CORBA

[14] Geocites." *Glosario Básico de Términos CORBA"* . Extraído el 6 de noviembre del 2005 en el sitio web: http://www.geocities.com/atomeua/GlosarioCorba.html

[15] Hillary Caituiro Monge. "*CORBA*". Universidad de Puerto Rico, Recinto Universitario de Mayagüez Mayagüez, Puerto Rico 00681-9042. Extraído el 7 de noviembre del 2005 en el sitio web: http://www.freewebz.com/amanecer/personal/papers/corba.htm.

[16] Xavier Ferré Grau, María Isabel Sánchez Segura "*Desarrollo Orientado a Objetos con UML*". Madrid – España. Extraído el 10 de noviembre del 2005 en el sitio web: http://www.clikear.com/manuales/uml/introduccion.asp.

[17] Patricio Salinas Caro, Nancy Histchfeld K. "*Tutorial de UML*". Extraído el 12 de noviembre del 2005 en el sitio web: "http://www.dcc.uchile.cl/~psalinas/uml/introduccion.html

[18] Antonio Sanz Pulido. "*Introducción al UML*". Extraído el 12 de noviembre del 2005 en el sitio web: http://www.yoprogramo.com/articulo4.php

[19] Osmosis Latina. "*Importancia de UML*". Extraído el 12 de noviembre del 2005 en el sitio web: http://www.osmosislatina.com/lenguajes/uml/basico.htm

[20] Pressman, *Ingeniería del Software*. (pag. 23)

[21] Millán, Ramón J. & Sánchez, Francisco. Comunicaciones World. *"Grid irrumpe en la empresa"*. Extraído el 16 de septiembre del 2005 en el sitio web: http://www.idg.es/comunicaciones/articulo.asp?id=174943

[22] SUMA: A Scientific Metacomputing. E. Hernández, Y. Cardinale, C. Figueira y A. Teruel. *Proceedings de PARCo'99*. Amsterdan, The Netherlands. 1999

[23] Fernando Martín Sánchez y Alberto Villafranca Ramos (Marzo 2005-09-12). *Introducción a la Tecnología GRID*.

[24] The Globus Alliance. The Globus Toolkit, 2006. Extraído el 19 de noviembre del 2006 en el sitio web: http://www.globus.org/

Anexos

Anexo Nº 1: Características de los elementos que constituyen el servidor de aplicaciones de SUMA/G

Clases	Métodos	Funcionalidad
ConcreteWrapper	main	Crea la instancia del engine y luego la registra en el servidor de nombres e inicializa el Orb en el localhost
ConcreteServer	setId	Especifica el Id del engine
	setResultDirectory	Seteá la variable que contiene el directorio donde se almacenaran los resultados
	setCorbaReference	Establece la referencia CORBA del engine
	configure	Establece los parámetros de configuración en el objeto
	generateAppId	Genera un id para la identificación de la aplicación
	validateCU	Valida la unidad de configuración en el Resourse Control
	initSumitRepository	Crea el directorio de resultados del Submit
	unRegisterSubmitId	Quita del Scheduller el identificador del Submit
	intExecute	Busca el nodo que realizará la ejecución, y le indica que la ejecute
	getID	Retorna el Identificador del engine
	execute	Chequea los permisos del cliente y llama a intExecute
	reconnect	Reconecta al servicio de ejecución
	checkAppId	Verifica que el identificador de la aplicación tenga un formato válido
	submit	Ejecuta remotamente un programa java, donde el cliente no debe permanecer conectado durante la ejecución
	cancel	Cancela el trabajo que aún no se ha ejecutado
	getResult	Retorna el resultado de una ejecución submit
	getStatus	Retorna el estado de una ejecución submit

Anexo Nº 2: UML

UML (Unified Modeling Language) es un lenguaje gráfico que permite visualizar, especificar, modelar, construir y documentar cada una de las partes que comprende el desarrollo de software.

UML entrega una forma de modelar cosas conceptuales como lo son procesos de negocio y funciones de sistema, además de cosas concretas como lo son escribir clases en un lenguaje determinado, esquemas de base de datos y componentes de software reusables.

El Lenguaje Unificado de Modelado, UML es el resultado de una propuesta de estandarización promovida por el consorcio OMG (Object Management Group), del cual forman parte las empresas más importantes que se dedican al desarrollo de software, en 1996.

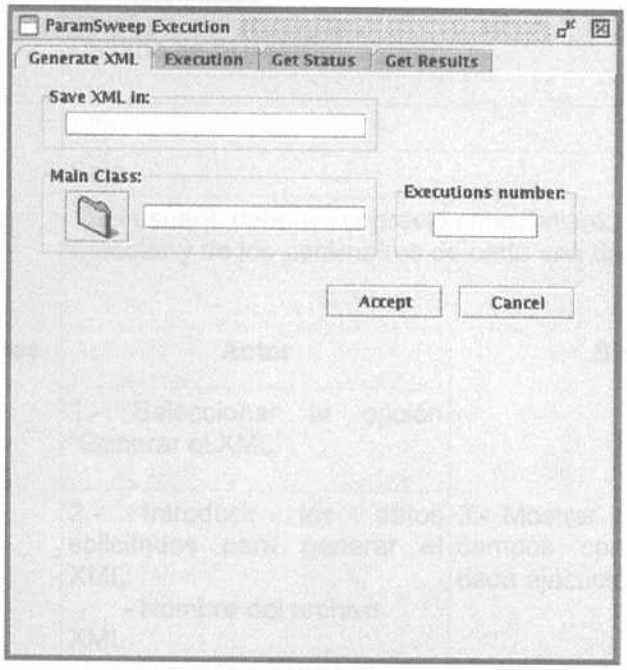
Es importante recalcar que, UML sólo se trata de una notación, es decir, de una serie de reglas y recomendaciones para representar modelos. UML no es un proceso de desarrollo, es decir, no describe los pasos sistemáticos a seguir para desarrollar software. UML sólo permite documentar y especificar los elementos creados mediante un lenguaje común describiendo modelos.

Mediante UML es posible establecer la serie de requerimientos y estructuras necesarias para plasmar un sistema de software previo al proceso intensivo de escribir código. Para esto se establecen una serie de diagramas, entre los cuales mencionaremos los siguientes:

- Diagrama de Casos de Uso: modela la funcionalidad del sistema agrupándola en descripciones de acciones ejecutadas por un sistema para obtener un resultado.
- Diagrama de Clases: muestra las clases (descripciones de objetos que comparten características comunes) que componen el sistema y cómo se relacionan entre sí.
- Diagrama de Secuencia: enfatiza la interacción entre los objetos y los mensajes que intercambian entre sí junto con el orden temporal de los mismos.

Puesto que UML es empleado en el análisis para sistemas de mediana-alta complejidad, era de esperarse que su base radica en otro paradigma empleado en diseños de sistemas de alto nivel que es la *orientación a objetos*, por lo que para trabajar en UML puede ser considerado un pre-requisito tener experiencia en un lenguaje orientado a objetos.

Anexo Nº 3: Cliente gráfico establecido en la Tesis Especial de Grado de la UCAB[6]



Anexo Nº 4: Caso de Uso para la generación de archivo XML

<table>
<tr><td>Nombre</td><td colspan="2">Generar Archivo XML</td></tr>
<tr><td>Descripción

Actor</td><td colspan="2">El cliente gráfico construye el archivo XML con los datos de las Ejecuciones

Usuario de SUMA/G</td></tr>
<tr><td>Prioridad</td><td colspan="2">Alta</td></tr>
<tr><td>Riesgo</td><td colspan="2">Bajo</td></tr>
<tr><td>Pre-condición</td><td colspan="2">1. El usuario debe tener previo conocimiento de la Clase a Ejecutar y de los parámetros de cada una de las instancias.</td></tr>
<tr><td rowspan="4">Flujo de Eventos</td><td>Actor</td><td>Sistema</td></tr>
<tr><td>1.- Seleccionar la opción “Generar el XML”</td><td></td></tr>
<tr><td>2.- Introducir los datos solicitados para generar el XML:
- Nombre del archivo XML
- Directorio donde se creará el XML
- Número de ejecuciones a realizar</td><td>3.- Mostrar la página con los campos correspondientes a cada ejecución.</td></tr>
<tr><td>4.- Introducir los datos que se solicitan:
- Nombre de la clase a Ejecutar
- Parámetros de cada ejecución
5.- Presionar “Construir XML”</td><td>6.- Crear el archivo XML.</td></tr>
<tr><td>Flujo Alternativo</td><td colspan="2">2.1 Si alguno de los campos están vacíos se muestra un mensaje de error
4.1 Si los campos están vacíos o no son validos se muestra un mensaje de error</td></tr>
<tr><td>Post-condición</td><td colspan="2">Archivo XML generado en el directorio especificado</td></tr>
</table>

Anexo Nº 5: Caso de Uso de la Ejecución Parameter Sweep

<table>
<tr><td>Nombre</td><td colspan="2">Realizar Ejecución Parameter Sweep</td></tr>
<tr><td>Descripción</td><td colspan="2">El cliente establece los datos de la ejecución, y realiza la ejecución Parameter Sweep</td></tr>
<tr><td>Actor</td><td colspan="2">Usuario de SUMA/G</td></tr>
<tr><td>Prioridad</td><td colspan="2">Alta</td></tr>
<tr><td>Riesgo</td><td colspan="2">Medio</td></tr>
<tr><td>Pre-condición</td><td colspan="2">1. El usuario debe haber generado el archivo XML con los datos de la Ejecución, y debe tener un certificado.</td></tr>
<tr><td rowspan="4">Flujo de Eventos</td><td>Actor</td><td>Sistema</td></tr>
<tr><td>1.- Seleccionar la opción “Ejecución”</td><td></td></tr>
<tr><td>2.- Introducir los datos solicitados para la ejecución:
- Archivo XML
- Contraseña para el certificado</td><td>3.- Realizar la ejecución Parameter Sweep con la datos dados</td></tr>
<tr><td></td><td>5.- Mostrar el Identificar de la Ejecución Parameter Sweep y almacenar este en el archivo “ParamSweepIDs.txt”</td></tr>
<tr><td>Flujo Alternativo</td><td colspan="2">2.1 Si alguno de los campos están vacíos se muestra un mensaje de error
2.2 Si el archivo XML no existe se muestra un mensaje de error
2.3 Contraseña incorrecta se muestra un mensaje de error</td></tr>
<tr><td>Post-condición</td><td colspan="2">Ejecución Parameter Sweep realizada.</td></tr>
</table>

Anexo Nº 6: Caso de Uso para la obtención del Estado de la Ejecución Parameter Sweep

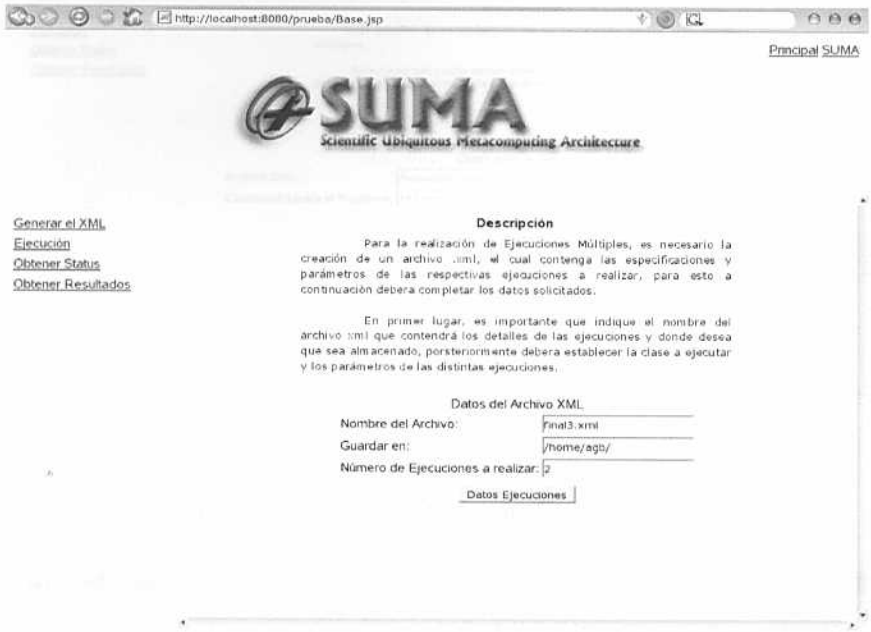
<table>
<tr><td>Nombre</td><td colspan="2">Obtener estado de la Ejecución Parameter Sweep</td></tr>
<tr><td>Descripción</td><td colspan="2">El cliente obtiene el estado en que se encuentran las ejecuciones.</td></tr>
<tr><td>Actor</td><td colspan="2">Usuario de SUMA/G</td></tr>
<tr><td>Prioridad</td><td colspan="2">Alta</td></tr>
<tr><td>Riesgo</td><td colspan="2">Bajo</td></tr>
<tr><td>Pre-condición</td><td colspan="2">1. El usuario debe haber mandado a ejecutar la clase, tener el ID de la Ejecución y el keystore</td></tr>
<tr><td rowspan="3">Flujo de Eventos</td><td>Actor</td><td>Sistema</td></tr>
<tr><td>1.- Seleccionar la opción “Obtener Estado”</td><td></td></tr>
<tr><td>2.- Introducir los datos solicitados:
- Identificador de la Ejecución
- Contraseña para el certificado</td><td>3.- Mostrar para cada instancia de la ejecución su identificador y el estado en el que se encuentra.</td></tr>
<tr><td>Flujo Alternativo</td><td colspan="2">2.1 Si alguno de los campos están vacíos se muestra un mensaje de error
2.2 Contraseña incorrecta se muestra un mensaje de error</td></tr>
<tr><td>Post-condición</td><td colspan="2">Mostrar el estado de la ejecución Parameter Sweep solicitada.</td></tr>
</table>

Anexo Nº 7: Caso de Uso para la obtención de los resultados de las Ejecuciones Parameter Sweep

Nombre	***Obtener resultados de la Ejecución Parameter Sweep***	
Descripción	El cliente obtiene el resultado de cada instancia de la ejecución.	
Actor	Usuario de SUMA/G	
Prioridad	Alta	
Riesgo	Bajo	
Pre-condición	1. El usuario debe haber mandado a ejecutar la clase, tener el ID de la Ejecución y el certificado	
Flujo de Eventos	**Actor**	**Sistema**
	1.- Seleccionar la opción "Obtener Resultados"	
	2.- Introducir los datos solicitados: - Identificador de la Ejecución - Directorio donde serán almacenados los resultados. - Contraseña para el certificado	3.- Descargar resultados de la ejecución en el directorio especificado. 4.- Mostrar página indicando que los resultados fueron descargados.
Flujo Alternativo	2.1 Si alguno de los campos están vacíos se muestra un mensaje de error 2.2 Contraseña incorrecta se muestra un mensaje de error 2.3 Directorio no existe: mensaje de error	
Post-condición	Resultados de la ejecución descargados.	

Anexo Nº 8: Diseño de las páginas que forman parte del cliente gráfico implementado en el presente Trabajo Especial de Grado.

- Página para Generar el Archivo XML (1/2)

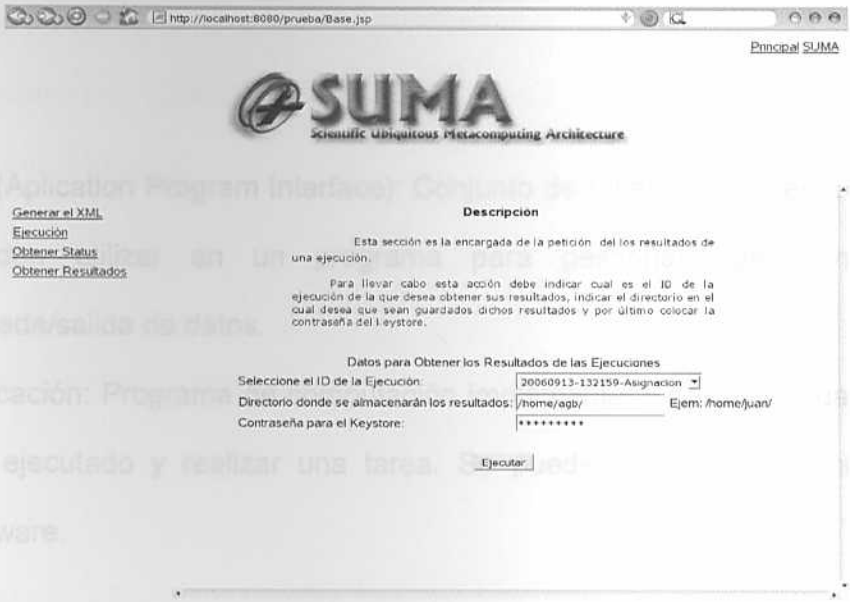


- Página para Generar el Archivo XML (2/2)

- Página de Ejecución ParamSweep

- Página para Obtener el Estado de Ejecución

- Página para Obtener los Resultados de una Ejecución



GLOSARIO

A

API(Aplication Program Interface): Conjunto de rutinas del sistema que se pueden utilizar en un programa para gestionar operaciones de entrada/salida de datos.

Aplicación: Programa de computación implementado en un lenguaje para ser ejecutado y realizar una tarea. Se puede considerar sinónimo de software.

B

Batch: Entorno de ejecución en lotes.

Bytecode: Código de objeto que es procesado por un programa conocido como máquina virtual.

C

Callbacks: Cuando un ente A realiza una llamada a un ente B, un *callback* sucede si durante esa llamada del ente B realiza una llamada al ente A.

Cluster: Es un grupo de equipos independientes que ejecutan una serie de aplicaciones de forma conjunta y aparecen ante clientes y aplicaciones como un solo sistema.

Computación distribuida: Es un sistema donde un conjunto de componentes y objetos que conforman una aplicación, están distribuidos

en diferentes computadoras y se comunican entre sí para el funcionamiento de la aplicación.

Concurrencia: petición simultanea de un recurso o archivo en el mismo instante de tiempo.

H

Heterogeneidad: Diversidad

I

Instancia: Un objeto de una determinada clase de componente. Cada instancia tiene sus propios elementos de datos privados o variables miembro. Una instancia de componente es sinónimo de objeto

J

Java: Es un lenguaje de programación de alto nivel orientado a objetos. Los programas creados con *Java* son independientes de la plataforma por estar basados en *bytecode*.

JAXB: Es el API de Java destinado a *XML Data Binding*, es decir, pasar de objeto *Java* a *XML* y viceversa.

M

Midleware: componente software ser usado como de puente de comunicación entre una aplicación y distintas bases de datos o otras aplicaciones.

MPI (Message Passing Interface): Es un estándar para la creación de librerías que permitan el pase de mensajes entre programas

MPIJava: Es una interfaz *Java*, basada en el paradigma de orientación a objetos del estándar *MPI*.

Multiplataforma: Es la capacidad o característica de poder funcionar o mantener una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas

N

Nodo: Equipo que está conectado a una red. En una red de área local, un nodo es un dispositivo que está conectado a la red y es capaz de comunicarse con otros dispositivos de la misma.

P

PLapack: Es una infraestructura para la implementación paralela de algoritmos de álgebra lineal y aplicaciones de supercomputadoras de memoria distribuida.

Protocolo de comunicación: Conjunto de reglas formuladas para controlar el intercambio de datos entre dos entidades comunicadas.

S

Supercomputadora: es una computadora con capacidades de calculo muy superiores a las comúnmente disponibles de las maquinas de escritorio de la misma época en que fue construida.