Universidad de las Ciencias Informáticas

FACULTAD 6


# Título: Aplicación genérica para la ejecución por lotes de problemas sobre la Plataforma de Tareas Distribuidas

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

**Autor: Yaneli Dayana Pérez Santana**

**Tutores: MsC. Longendri Aguilera Mendoza**
**Ing. César Raúl García Jacas**

**La Habana. Junio de 2013**
**"Año 55 de la Revolución"**

*"El camino del progreso no es ni rápido ni fácil."*

*Marie Curie*

**DECLARACIÓN DE AUTORÍA**

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____

MsC. Longendri Aguilera Mendoza

Firma del Tutor

_____

Ing. César Raúl García Jacas

Firma del Tutor

_____

Yaneli Dayana Pérez Santana

Firma del Autor

## DATOS DE CONTACTO

**Autores**:
Yaneli Dayana Pérez Santana.
Universidad de las Ciencias Informáticas.
e-mail: ydsantana@estudiantes.uci.cu

**Tutores**:
MsC. Longendri Aguilera Mendoza
Universidad de las Ciencias Informáticas (UCI)
e-mail: loge@uci.cu

Ing. César Raúl García Jacas
Universidad de las Ciencias Informáticas (UCI)
e-mail: crjacas@uci.cu

*En primer lugar a Dios y a la Virgen por permitirme estar aquí hoy, por darme fuerzas siempre. A ti mamiska, por la educación tan inmensa que me has dado, por llevarme a ser alguien en la vida, por inspirar, con solamente una mirada, el respeto más grande que pueda sentir por una persona. A ti papía, por dejarme saber siempre lo orgulloso que vides de mi. A ti Robertico, porque has sido un ejemplo para mi, por tu amor de padre. A mis abuelas, mami: por ser mi globo terráqueo, por tolerar mis malcriadeces; a mima, por los consejos de abuelita. A mi primo preferido, mi hermanito pequeño, a ti, niño por dejarme ser tu nana. A mi prima Sahily, gracias por la sonrisa que me das incluso cuando no la merezco. A mi primis Elsa María, Marthica y Pepitín, por lo buenos que han sido siempre conmigo. A mi hermano lindo, por los consejos y el apoyo, por todo lo que te he extrañado. A mi hermanita, por sus ojos tiernos, porque espero ser un día un ejemplo para ella. A mi padrino, por ser el primero en tenderle la mano a mi papá, por no alejarse de mí nunca.*

*A mis amigos de siempre: a ti Ely por estar siempre en mi vida, por seguir tu sueño; a ustedes Willy y Rey, por dejarme ganar siempre; por hacerme sentir como la hermana mayor, por dejarme ser el pegamento de este trío perfecto.*

*A mis amigos de mi mejor escuela: a ti Diani, mi alemanita, por darme la razón sólo cuando la tengo; a ti Daye, mi psicóloga, por hacerme sentir grande; a ti Mendi, mi bellota, por dejarme ser bombón, por la oportunidad de tener una amiga; a ti Celia, mi ingeniera habanerita, por estar siempre cerca. A todas gracias por los maravillosos consejos. A mi pequeño continente, mi América, por aguantar mis locuras. A mi azabache, mi Nancy, por recordarme el inmenso avance. A mi profe preferido, mi amigo, mi Chiqui, por decirme siempre la verdad.*

*A mis amigos de la universidad: a ti Inés, por ser la persona más buena que he conocido; a ti Are, por el apoyo tan inmenso que me has dado; a ti Mirix, mi negrita preferida, por enseñarme tantas cosas de la vida, por tus sabios consejos, por tu compañía; a ti Adrian, mi Adrian Monk, por hacerme reír a toda hora;*

*a mi Yordy, por ser mi hermana mayor, por ayudarme a seguir adelante; a mi psicólogo preferido, Dieter, por la lección de vida; a ti, Reicel, por enseñarme las dos caras de la moneda; a ti Robert, poeta, por tu amistad; a ti Pedri, por no dejar que la vida nos alejara; a ti Anita, loquilla, por el fundamento con el que me escuchas; a ti Migue, por tener fe en mi, por escucharme siempre, por la paciencia, por el titi, por todo.*

*A las personas que me han ayudado en esta etapa de tesis, gracias Eudel, Osniel, Osvel, Ale, Alina por todo el tiempo que me han dedicado, por la paciencia para aclarar mis dudas.*

*A mis tutores: Longendri, por la serenidad para organizar las cosas; César, por enseñarme a defender mis ideas, por hacerme sentir que lo más importante es estar seguro de lo que haces.*

*Al tribunal por la ayuda que me han dado, por los consejos para llegar a obtener un mejor resultado en mi trabajo de diploma. A mi oponente, por hacerme querer estudiarlo todo.*

*A todos mis profes, porque me han hecho crecer cada año.*

*A una persona que fue muy importante en esta etapa de mi vida que culmina, a Erick, por su apoyo durante tanto tiempo, por ayudarme a llegar hasta aquí, por el amor tan grande que me dio siempre. Gracias por haber sido mi familia cuando estaba tan lejos de la verdadera.*

*A todos aquellos que se han detenido un momento a preguntarme cómo iba la tesis, y a desearme éxitos.*

*A todos los que hoy me dan el placer de verlos aquí.*

*A ti mamá, por todas las veces que me dijiste por dónde debía ir, y por todas las que estuviste al final del camino mal elegido, para abrazarme y volverme a enseñar el correcto. A ti, porque esta tesis es tuya, como son tuyos todos mis logros, como es tuya mi vida.*

*A ti papá, por no alejarte de mí nunca, ni siquiera estando lejos; por dejarme ser tu aire; por estar hoy más presente que nunca, aunque no te pueda ver. Por luchar tanto.*

*A ti Robertico, por dejarme ser tu niña, porque eres el hombre más honrado que conozco; por quererme tanto y ser un ejemplo para mí.*

*De manera muy especial a dos estrellas que me iluminan desde el cielo:*

*A ti abuelito, que la vida nos dio poco tiempo para conocernos, pero suficiente para saber que heredé de ti el ser universitaria. A ti, porque dice el amor de mi vida, que si me vieras estuvieras orgulloso, yo estoy orgullosa de ti.*

*A ti tía, donde quiera que estés además de en mi corazón, porque de tanto decir: "Dayanita no seas así" me hiciste ser mejor persona. Porque eres mi ángel de la guarda. Por todo el amor que sé, me tienes.*

# ÍNDICE

# ÍNDICE DE TABLAS Y FIGURAS

**TABLAS**



**FIGURAS**

## RESUMEN

La revolución tecnológica en la cual se encuentra inmersa la sociedad actual, va dirigida al cambio de paradigmas. El surgimiento de una nueva rama de la ciencia: la Bioinformática, y el desarrollo de nuevas técnicas de programación como la distribuida, dejan el camino abierto a nuevos retos en cuanto al desarrollo de software que combinen estas innovaciones. La Plataforma de Tareas Distribuidas (T-arenal) es un sistema capaz de solucionar problemas que permitan descomponer sus datos de salida y que demandan de cómputo. Actualmente la Plataforma necesita el desarrollo de una aplicación independiente para cada uno de los problemas que serán solucionados; este proceso es considerablemente frecuente, por lo que constantemente se están desarrollando aplicaciones similares debido a que los problemas para los que son creadas, comparten un conjunto de características. Por lo anteriormente planteado se determina que puede optimizarse el tiempo de desarrollo de este tipo de aplicaciones. El presente trabajo de diploma propone el desarrollo de una aplicación genérica capaz de ejecutar cualquier problema que cumpla con las características anteriormente mencionadas. Debido a la similitud de estos problemas se determina que pueden ser definidos mediante un lenguaje de descripción de trabajo, que será utilizado por la aplicación propuesta para generar las tareas sobre la Plataforma; por lo que dicho lenguaje será otro resultado del presente trabajo; así como lo será también, el desarrollo de una herramienta de apoyo capaz de generar este lenguaje a partir de la especificación de las características y los datos de los problemas.

**PALABRAS CLAVES:** lenguaje de descripción de trabajo, JDL, tareas, Plataforma de Tareas Distribuidas, T-arenal, XML.

# INTRODUCCIÓN

Tradicionalmente el software ha estado orientado hacia la computación en serie, consistente en resolver problemas mediante la construcción de un algoritmo implementado en un flujo de instrucciones secuenciales, teniendo que esperar que finalice una para comenzar a ejecutarse la siguiente. Para los años 70, los avances en esta ciencia llegaban al desarrollo de una nueva técnica de programación: la computación paralela; consistente en resolver problemas, empleando elementos de procesamiento múltiple, simultáneamente. Con la computación paralela solamente no se introducía una técnica que ahorra tiempo y dinero, sino que permite solucionar problemas mayores, imposibles de resolver en un equipo; además de proporcionar el uso de recursos no locales, y concurrencia. (1)

Para la última década del siglo XX los avances en esta ciencia, la informática, condicionaron el surgimiento de una nueva disciplina, aparejado al desarrollo de la ingeniería genética, la biotecnología y la presencia de grandes volúmenes de datos biológicos: la Bioinformática, encargada del estudio, comprensión y análisis de estos datos utilizando herramientas informáticas. Esta nueva disciplina se divide en las siguientes áreas: *Genómica Computacional*; *Bioinformática Estructural*; *Biología de Sistemas*; *Algoritmos y Base de Datos.* (2) Los principales esfuerzos de investigación en estos campos incluyen el alineamiento de secuencias, la predicción de genes, predicción de la expresión génica, ensamblaje del genoma, alineamiento y predicción de estructuras y proteínas, interacciones proteína-proteína, y modelado de la evolución. (3) Por consiguiente constituyó un paso de avance en la búsqueda de nuevos programas, capaces de satisfacer la gran demanda de cómputos que exige, como toda disciplina científica, debido a las grandes cantidades de volúmenes de datos con los que trabaja.

En Cuba, con la creación de la Universidad de las Ciencias Informáticas surge el departamento de Bioinformática como línea de producción, asociado a la vez con centros pertenecientes al resto de país que desarrollan esta ciencia. Debido a la gran capacidad de cómputo que demandan las soluciones de determinados problemas dentro de la Bioinformática y que son objeto de estudio dentro del departamento, se desarrolla en el mismo, la Plataforma de Tareas Distribuidas (T-arenal). Esta plataforma permite aglutinar en un único conjunto varias estaciones con el fin utilizar la máxima cantidad de recursos computacionales bajo sus controles en una red local, creando las condiciones necesaria para satisfacer la

demanda de cómputo anteriormente mencionada para la solución de determinados problemas, que cumplen además con la premisa de permitir descomponer sus datos de salida. (4)

T-arenal necesita para la ejecución de cada uno de estos problemas, la implementación de una aplicación de computación distribuida, que sea capaz de generar los lotes de datos que permitirán dividir las salidas de las ejecuciones de cada una de las tareas correspondientes a cada problema. Cada aplicación debe ser desarrollada por un especialista informático que realice un análisis del negocio de cada problema de manera particular, para determinar las clases a implementar, codificarlas, y proceder entonces a su ejecución; esto trae consigo que los especialistas de las ciencias de la vida tengan una dependencia de un informático que desarrolle este tipo de aplicación independiente. Los problemas resueltos por T-arenal tienen la peculiaridad de presentar similitudes en cuanto a sus descripciones, por lo que las aplicaciones necesarias para cada uno presentan también similitudes en cuanto al proceso de desarrollo.

Por lo anteriormente expuesto se estipula que puede tanto optimizarse el tiempo de desarrollo de aplicaciones para problemas que necesiten ser distribuidos en la plataforma; como también lograr que los especialistas en las ciencias de la vida puedan solamente con el acceso a la plataforma y los datos del problema llegar a la solución del mismo sin la necesidad de un informático que implemente una aplicación para cada problema.

De allí que se plantee como **problema a resolver**: ¿Cómo reducir el tiempo de desarrollo de aplicaciones distribuidas sobre la plataforma T-arenal para obtener los resultados de determinados problemas frecuentes en la Bioinformática?

- **Objeto de estudio:** Lenguaje de definición de trabajo.
- **Campo de acción:** Lenguaje de definición de trabajo en la Plataforma de Tareas Distribuidas.
- **Objetivo General:** Desarrollar una aplicación genérica para la ejecución por lotes de problemas sobre la plataforma T-arenal, mediante un lenguaje de descripción de trabajo, para contribuir a disminuir el tiempo de desarrollo de aplicaciones independientes para cada problema.

### ➢ Objetivos específicos

1. Diseñar un lenguaje de definición de trabajo, que permita describir determinados problemas cuya solución se puede obtenerse a través de T-arenal.
2. Realizar el análisis de la aplicación que interprete el lenguaje de descripción de trabajo diseñado, para ejecutar los cálculos distribuidos sobre T-arenal y el de la herramienta de apoyo que facilite la descripción de un problema en dicho lenguaje.
3. Diseñar la aplicación y la herramienta de apoyo.
4. Implementar la aplicación y la herramienta de apoyo.
5. Validar la aplicación y la herramienta de apoyo.

### ➢ Tareas de la investigación

1. Descripción de los sistemas de cómputo distribuido, especialmente T-arenal.
2. Descripción de los problemas a resolver, frecuentes en la Bioinformática, en sistemas de cómputo distribuido.
3. Selección de las herramientas y tecnologías a utilizar para la implementación de la solución propuesta.
4. Identificar los requisitos correspondientes a la aplicación genérica y la herramienta de apoyo.
5. Descripción del modelo de Casos de Uso del Sistema.
6. Caracterización de los lenguajes de descripción de trabajo definidos por distintos middleware.
7. Definición de los elementos del lenguaje de descripción de trabajo a diseñar.
8. Definición del lenguaje de descripción de trabajo para la plataforma T-arenal con el uso del lenguaje formal XML.
9. Descripción del modelo de diseño.
10. Descripción del modelo de implementación.
11. Implementación de la aplicación genérica y la herramienta de apoyo.
12. Validación de la solución propuesta utilizando pruebas de caja blanca para la aplicación genérica, de caja negra para la herramienta de apoyo y el estudio de caso para demostrar que se reduce el tiempo de desarrollo.

**Estructura del documento:**

CAPÍTULO 1. **Fundamentación Teórica**, en este capítulo se hace la presentación del marco teórico, a partir del estudio de la bibliografía relacionada con el tema tratado y la realización de una recopilación de información referente al mismo; en este capítulo también se establecerán las herramientas y tecnologías a emplear para dar solución al problema en cuestión.

CAPÍTULO 2. **Características del Sistema**, en este capítulo se realiza una breve descripción del sistema a desarrollar, de la representación del modelo de dominio y de los requisitos funcionales y no funcionales. Se definen los actores, se presenta el diagrama de Casos de Usos del Sistema, detallando cada uno de ellos textualmente.

CAPÍTULO 3. **Diseño del Sistema**, en este capítulo se presenta el estilo arquitectónico definido así como los patrones de diseño de software utilizados y los diagramas de clases del diseño.

CAPÍTULO 4. **Implementación y Prueba**, en este capítulo se realizará la descripción de la implementación de la solución propuesta, la exposición de los diagramas de componentes, la representación del despliegue y la presentación de los resultados arrojados por las pruebas realizadas tanto a la aplicación genérica como a la herramienta de apoyo.

# CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se abordarán los principales temas referentes al funcionamiento de T-arenal así como la taxonomía de cómputo que realiza para dividir los problemas que resuelve. Se presentará además la estructura de los lenguajes de descripción de trabajo que utilizan algunos middleware para el procesamiento distribuido. Se explicarán determinados problemas frecuentes en la Bioinformática que pueden solucionarse con la utilización de la plataforma. Finalmente se presentarán las herramientas y tecnologías a utilizar en el desarrollo de la aplicación genérica y la herramienta de apoyo.

## 1.1 Cómputo distribuido sobre T-arenal

La Plataforma de Tareas Distribuidas (T-arenal) es un producto informático que ofrece una alternativa de cómputo y que aglutina en un único conjunto varias estaciones de trabajo, sin intentar eliminar o pretender aminorar las amplias posibilidades de aplicación de los modelos paralelos, sino de complementar todos los medios disponibles en una gran "supercomputadora virtual". Por tales motivos tiene como propósito utilizar la máxima cantidad de recursos computacionales bajo sus controles en una red local, sin limitaciones en el tamaño del sistema. Se divide en dos partes principales: back-end y front-end. El front-end es el acceso a las funcionalidades del sistema por los usuarios a través de una interfaz gráfica, mientras que el back-end es el responsable de realizar todas las solicitudes realizadas a través del front-end. El back-end se divide en tres componentes de software: el servidor central, el servidor de peticiones y el cliente. Al servidor central estarán asociados uno o varios servidores de peticiones los cuales son los encargados de solicitar una tarea. A cada servidor de peticiones le pertenecerán uno o varios clientes para realizar el procesamiento de una tarea determinada. (5)

### 1.1.1 Taxonomía de Cómputo

Una de las clasificaciones más utilizada es la taxonomía de Flynn[1]. Esta distingue arquitecturas de computadora con múltiples procesadores de acuerdo a la forma en que se pueden clasificar por las dos

[1] Michael J. Flynn 1934. En 1972, propuso esta taxonomía. Premio *Harry H. Goode Memorial Award* por sus contribuciones al área del procesamiento de la información.

dimensiones independientes de instrucción y de datos. Cada una de estas dimensiones sólo puede tener uno de dos estados posibles: simple o múltiple. Las cuatro clasificaciones definidas por Flynn son: simple instrucción, simple dato; múltiples instrucciones, simple dato; simple instrucción, múltiples datos y múltiples instrucciones, múltiples datos; o SISD, MISD, SIMD, MIMD respectivamente por sus siglas en inglés. (6) Por las características estudiadas de cada una se define que T-arenal utiliza como taxonomía de cómputo SIMD; consistente en un computador que explota varios flujos de datos dentro de un único flujo de instrucciones; de allí que los problemas a resolver estén determinados por un conjunto de datos y una instrucción.

Para la realización de los cálculos que demanda la solución de cada problema se divide el mismo en tareas esto consiste en dividir los datos en lotes que serán enviados a cada cliente, conjuntamente con la instrucción, conformándose así cada una de las tareas. Esta división en lotes se realiza mediante una de las técnicas de descomposición utilizadas para el procesamiento distribuido, estás técnicas se clasifican en: descomposición recursiva, descomposición de datos, descomposición exploratoria, y descomposición especulativa. (6) T-arenal, debido a la taxonomía que utiliza y a las características de los problemas que resuelve, emplea la descomposición de datos.

**Descomposición de datos**

La descomposición de datos es de gran alcance y de uso común para dividir algoritmos que operan sobre estructuras de datos de gran tamaño. En esta, la descomposición de los cálculos se realiza en dos pasos. En el primer paso: son divididos los datos que van a ser calculados; y en el segundo paso: esta partición de datos se utiliza para inducir una fragmentación de los cálculos dentro de las tareas.

Esta técnica a su vez está formada por cinco estrategias para descomponer los datos: *Output, Input, Output and Input, Intermediate y Owner-Computes-Rules.* (6) Cada lote de datos enviado a los clientes cumple con la característica de arrojar resultados independientes a los restantes, lo que define que la estrategia de descomposición de datos utilizada es Output. Esta estrategia está basada en la partición de los datos de salida. Este tipo de partición de datos se emplea cuando cada elemento de la salida puede ser calculado de forma independiente de los demás. En tales cálculos, una partición de los datos de salida automáticamente induce una descomposición de los problemas en tareas, donde se le es asignado a cada tarea el cálculo de una porción de la salida.

### 1.1.2 Desarrollo de aplicaciones

T-arenal, para generar tareas necesita de una aplicación distribuida para cada problema que partiendo de cada uno de ellos sea capaz de dividir los datos del mismo en lotes, identificar la instrucción y conformar cada una de estas tareas; la aplicación es, por ende, la encargada de gestionar el problema a solucionarse. La implementación de la misma sólo requiere heredar dos clases de Java: DataManager y Task.

La clase DataManager se ejecuta en el servidor de peticiones. Tiene como propósito generar todas las unidades o tareas que se enviarán a los clientes y procesar todos los resultados enviados por estos.

La clase Task se ejecuta en cada uno de los clientes pertenecientes al servidor de peticiones que atiende o colabora en la realización de dicha tarea. La instancia de la misma correspondiente a una tarea en particular. Su propósito es procesar las unidades de trabajo generadas por la clase DataManager. (7)

## 1.2 Problemas de la Bioinformática con características de procesamiento por lotes

Los problemas existentes dentro de la Bioinformática generalmente demandan de cálculos intensos, sin embargo esta no es la única característica común que tienen. Existen determinados problemas que su estructura puede definirse esencialmente en dos partes: una instrucción y un conjunto de datos. Por lo que la forma de ejecutar estos problemas es dividir los datos en lotes y ejecutarlos con la misma instrucción. Esta división de los datos, se puede realizar mediante la técnica de descomposición Output (mencionada anteriormente) debido a que la solución que puede arrojar cada una de las ejecuciones, compuestas por la instrucción y cada uno de los lotes de datos, son independientes.

Por lo anteriormente expuesto se determina que este tipo de problemas, que cumple con todas las características anteriores, pueden solucionarse con la utilización de T-arenal, por tanto pueden solucionarse también con la aplicación genérica propuesta como solución al problema a resolver. A continuación se describen algunos de estos problemas.

### 1.2.1 Acoplamiento molecular

Actualmente, debido a los avances científicos y tecnológicos en el campo de la biología molecular, se ha identificado que numerosas macromoléculas desempeñan funciones vitales en el funcionamiento de la

célula y cuya alteración está relacionada con ciertas enfermedades. Partiendo de que las proteínas son los componentes macromoleculares más abundantes, no resulta sorprendente que éstas sean las dianas más frecuentes de los fármacos. Razón por lo cual, cuando se conoce la base biológica de una enfermedad o de un desarreglo metabólico y se pueden definir las proteínas que causan tal desajuste, se diseña un fármaco que es una pequeña molécula (llamada ligando) con las propiedades químicas y biológicas necesarias para unirse al sitio activo de esa proteína y moldear su funcionamiento a condiciones favorables.

La importancia del diseño de fármaco basado en la estructura del receptor es que pretende mejorar las interacciones que ocurren entre las moléculas. Lo que se traduce en un fármaco más efectivo. Con este propósito, se recurre a uno de los métodos computacionales más usados en este campo: el Acoplamiento Molecular (Molecular Docking en inglés). Este método consiste en calcular computacionalmente cuál es la posición más favorable que tendría una molécula en el banco molecular, trabajando con sus representaciones tridimensionales. Es importante destacar el gran número de grados de libertad que pueden tener dos moléculas para unirse, por lo que este método demanda de gran potencia de cómputo. (8)

### 1.2.2 Filogenia

La filogenia no es más que la historia evolutiva del flujo hereditario a distintos niveles evolutivos/temporales, desde la genealogía de genes en poblaciones (micro -escala; dominio de la genética de poblaciones) hasta el árbol universal (macro –escala). Las reconstrucciones filogenéticas que reflejen la historia del “flujo de la herencia” son representadas a través de árboles denominados: árbol filogenético, este simboliza una estructura matemática usada para representar la historia evolutiva (relaciones de ancestro-descendiente) entre un grupo de secuencias u organismos. Dicho patrón de relaciones históricas es la estima hecha de la filogenia o árbol evolutivo. Actualmente, uno de los inconvenientes que presentan estos problemas de filogenia es la gran capacidad de cómputo que demanda el análisis de grandes cantidades de individuos por lo que se hace necesaria la utilización de conglomerado de cómputo para minimizar el tiempo de ejecución y obtención de la solución final. (9)

### 1.2.3 Cálculos de química computacional utilizando métodos semi-empíricos

La química computacional es una rama de la química que utiliza computadores para ayudar a resolver problemas químicos. Utiliza los resultados de la química teórica, incorporados en algún software para calcular las estructuras y las propiedades de moléculas y cuerpos sólidos. Mientras sus resultados normalmente complementan la información obtenida en experimentos químicos, pueden, en algunos casos, predecir fenómenos químicos no observados a la fecha. La química computacional es ampliamente utilizada en el diseño de nuevos medicamentos y materiales. (6)

Los métodos empleados cubren situaciones estáticas y dinámicas. En todos los casos, el tiempo de cálculo aumenta rápidamente a medida que el tamaño del sistema estudiado crece. Estos métodos, por lo tanto, se basan en teorías que van desde la alta precisión, pero apropiados para pequeños sistemas, a las buenas aproximaciones, pero apropiadas para grandes sistemas. Dentro de este último caso se encuentran los empíricos o semi-empíricos, obtenidos de resultados experimentales, a menudo de átomos o moléculas relacionadas.

**Métodos semi-empíricos**

Los métodos semi-empíricos parten de la data necesaria para describir adecuadamente el sistema, los parámetros utilizados de obtienen ajustando para reproducir valores experimentales o *ab-initio* de un conjunto de moléculas. Estos métodos son mucho más rápidos que *ab-initio*. Algunos de los métodos semi-empíricos son (6):

- Hückel: es el más simple de todos, refleja la simetría orbital y predice coeficientes orbitales, sólo modela los e-π de valencia de sistemas conjugados planos.
- Hückel extendido: modela todos los orbitales de valencia basados en el solapamiento y en potenciales de ionización y afinidades electrónicas experimentales.
- PPP (Pariser-Parr-People): mejora el método Hückel, permite obtener espectros electrónicos de buena calidad para hidrocarburos aromáticos. Sus aproximaciones son usadas para evaluar integrales en los métodos semi-empíricos actuales.

## 1.3 Lenguajes de descripción de trabajo

Un lenguaje de descripción de trabajo (JDL por sus siglas en inglés), no es más que un lenguaje de alto nivel, utilizado para describir las características y requerimientos así como recuperar las salidas de la ejecución, de un trabajo o tarea a realizar en un computador. Permite estandarizar la descripción de problemas que puedan, según sus características, descomponerse en los parámetros que componen un JDL; principalmente son: los básicos, los requerimientos de hardware y software y las características de los datos.

Actualmente existen distintos middleware para el procesamiento distribuido como Condor (10), gLite (11) , Globus Toolkit (12) entre otros. Estos middleware defines su JDL para describir los problemas que serán ejecutados, esto les permite trabajar genéricamente con todos aquellos problemas que puedan describirse por el lenguaje definido. Seguidamente se abordarán aspectos específicos de la utilización y definición de estos JDL en los middleware seleccionados, para tomar de cada unos los parámetros que puedan incluirse dentro del JDL que será definido posteriormente, como parte de la solución propuesta para el problema a resolver.

### 1.3.1 Condor

Antes de analizar cómo Condor representa sus job, es importante entender cómo se asignan los recursos; debido a que es este uno de los factores fundamentales en la representación que utiliza este middleware para su JDL; este proceso de asignación de recursos se denomina Matchmaking, y garantiza que cada job se distribuya en los ordenadores que cumplen con las características necesaria para su ejecución. Este proceso, que requiere para su realización el conocimiento de los requisitos de los job y los requisitos de las máquinas, hizo que Condor decidiera implementar su JDL mediante ClassAds (Classified Advertisement), que no es más que un conjunto de expresiones con un nombre único definidas como atributo. ClassAds solamente no describe atributos esenciales para la ejecución propia del job, como el nombre del programa a ejecutar, el directorio de trabajo inicial, el directorio final, el directorio de envío de errores así como los argumentos de línea de comandos, por citar los más generales; sino que además permite describir los recursos y características de las estaciones de trabajos, mediante las conocidas métricas tanto dinámicas y estáticas, dando a conocer así sus limitaciones y permitiendo utilizarlas para

fines estadísticos y de depuración, así como para la solicitud del estado actual del sistema; por lo que se define como un mecanismo flexible.

Los atributos definidos por ClassAds, sin importar qué describan, tienen la siguiente sintaxis: **atributo = valor**; algunos de estos atributos son obligatorios y otros son opcionales. Referente a los que describen las características de los job, es primordial que se describan: el nombre del ejecutable, el nombre del fichero en el que se guarda el resultado y un fichero de error (que puede ser el mismo que el de resultado).

- Executable = "test.sh";
- StdOutput = "std.out";
- StdError = "std.err";

Partiendo de estos, si es necesario la especificación de argumentos puede incluirse mediante el atributo:

- Arguments = "hello 10";

También existen otros atributos de carácter opcional para especificar requisitos sobre los datos de entrada que empleará el job y también expresar como se generará la información de salida. Estos atributos son: *InputData, DataAccessProtocol, OutputSE, OutputData, OutputFile, LogicalFileName, StorageElement y ReplicaCatalog.*

Referente a los atributos que describen las características a tomar en cuenta para el proceso de Matchmaking, resaltan dos fundamentales y de gran uso, definidas por los atributos: *Requirements y Rank*, que permiten al usuario especificar, cuáles son las necesidades y preferencias, en términos de recursos, de sus aplicaciones.

El atributo Requirements se puede usar para expresar cualquier tipo de restricción de los recursos en los que se ejecutará el job. El atributo Rank define las condiciones del ranking. Ambos son utilizados para la selección del destino del job para su ejecución, puesto que además de basarse en los requisitos indicados (Requirements), se base en un ranking (Rank) que es inversamente proporcional a la cantidad de job en espera, o equivalente al número de CPUs libre en ausencia de job en espera. (13) (14)

### 1.3.2 gLite

El JDL utilizado para gLite, se basa en el lenguaje Condor ClassAd, por lo que cumple con todas las características explicadas anteriormente sobre el ClassAds de Condor. Su construcción central es una estructura de registros, compuesto de un número finito de nombres de atributos asignados a distintas expresiones; el modelo de datos de este ClassAd es el siguiente:

- Flexible.
- Extensible.
- Puede representar servicios y restricciones de esos servicios.

Está estructurado por un conjunto de atributos predefinidos que tienen un significado especial para los componentes fundamentales del sistema de gestión de carga de trabajo. Algunos de ellos son obligatorios, mientras que otros son opcionales. El conjunto de atributos predefinidos se puede descomponer en los siguientes grupos (15):

- *Job attributes*: representan información específica del job y especifica las acciones que deben ser realizadas por el sistema de gestión de carga de datos (WMS por sus siglas en inglés, Workload Management System) para programar este job;

- *Data attributes*: representación de los datos de entrada del job y de los elementos de almacenamiento relacionados con la información. Se utilizan para la selección de los recursos desde los que la aplicación tiene el mejor acceso a los datos;

- *Requirements y Rank*: permite al usuario especificar, respectivamente, cuáles son las necesidades y preferencias, en términos de recursos, de sus aplicaciones.

Como es lógico, se debe indicar, al menos, el nombre del ejecutable, el nombre del fichero en el que se guarda el resultado y un fichero de error (que puede ser el mismo que el de resultado).

- Executable = "test.sh";
- StdOutput = "std.out";
- StdError = "std.err";

Si se necesitan, se pueden pasar argumentos al ejecutable:

- Arguments = "hello 10";

También existen otros atributos de carácter opcional para especificar requisitos sobre los datos de entrada que empleará el job y también expresar como se generará la información de salida. Estos atributos son: *InputData, DataAccessProtocol, OutputSE, OutputData, OutputFile, LogicalFileName, StorageElement and ReplicaCatalog.* (15)

El proceso de selección del destino del job se realiza de la misma forma que fue descrito con anterioridad para Condor.

### 1.3.3 Globus Toolkit

Globus Toolkit a diferencia de los middleware descritos anteriormente, define opciones de comandos que permiten establecer opciones globales usadas para esencialmente controlar el envío de jobs y proporcionar valores predeterminados para las subtareas cuando se utiliza la sintaxis Multijob. Algunas de estas opciones se describen a continuación:

- *Verbose**:*** imprime mensajes de depuración de secuencias de comandos adicionales de ejecución.
- *JobID*: especifica el ID del job.
- *Max-tiempo*: especifica la cantidad máxima de tiempo que el planificador debe permitir ejecutar el job.
- *Max-memoria*: especifica la cantidad máxima de memoria que el programador debe permitir ejecutar el job.
- *Min-memoria*: especifica la cantidad mínima de memoria que el job necesita para funcionar.
- *Project*: especifica el ID de proyecto relacionado con el job.
- *Dir*: especifica el directorio de trabajo del job.

Como se puede observar estas opciones están en función de las necesidades de los job para ser ejecutados, y se encargan del proceso de depuración para el envío de los job. Partiendo de la existencia de estos comandos, Globus enmarca la implementación del JDL en el establecimiento únicamente de las características del job y sus atributos para ser ejecutado; por lo que define su lenguaje a través de un

XML, que cuenta con atributos de carácter opcional y atributos de carácter obligatorio; definidos como etiquetas y detallados a continuación (16):

- *<jobType>:* define el tipo de job a ejecutar.
- *<argument>:* establece los argumentos del job.
- *<stdout>:* archivo que almacena la salida.
- *< stderr >:* archivo que almacena los errores encontrados.
- *<fileStageIn>:* define una lista que consta de dos direcciones. La dirección remota del archivo a ejecutar en el nodo de cómputo, definida por la etiqueta *<sourceUrl>,* y la dirección local del archivo que va a ser ejecutado *<destinationUrl>.*
- *<fileCleanUp>:* se elimina el archivo creado una vez ejecutado el job.

Para solucionar el problema en cuestión se determinó que por las características de los JDL vistos anteriormente y por las prestaciones de la plataforma T-arenal, los problemas serán descritos en un XML, cuya estructura se definirá mediante un XSD. Logrando de esta manera que pueda realizarse una aplicación genérica que pueda distribuir cualquier problema descrito en un XML que cumpla con el XSD definido.

## 1.4 Herramientas y tecnologías

La construcción de la solución propuesta, requiere que sea organizado todo el proceso de desarrollo, con la finalidad de obtener un producto con calidad y que satisfaga las necesidades apremiantes del usuario final. Seguidamente serán definidas las herramientas y tecnologías que permitirán organizar y preparar el proceso de desarrollo de la misma.

### 1.4.1 Metodología de desarrollo de software

Las metodologías ágiles están especialmente preparadas para sufrir cambios durante el proyecto, desarrollan software en cortos períodos de tiempo y exigen poca documentación, permiten un estrecho vínculo con el cliente y están diseñadas para equipos pequeños. OpenUp (Open Unified Process) es una metodología ágil de desarrollo de software; entre sus características principales se encuentran: desarrollo iterativo e incrementar, dirigido por casos de uso y centrado en la arquitectura. Sin embargo a pesar de

ser una metodología ágil respeta la documentación más importante del proceso de desarrollo, por lo que permite tener correctamente documentada las soluciones desarrolladas. Esta metodología guiará el proceso de desarrollo para la solución del problema planteado. (17)

### 1.4.2 Lenguaje de modelado

Para solucionar el problema planteado se utilizará como lenguaje de modelado UML (Unified Modeling Language). Especializado en especificar, visualizar y documentar sistemas de software, incluyendo su estructura y diseño, de manera que cumpla con todos estos requisitos. (18) Permite automatizar determinados procesos y soporta generar código a partir de los modelos y a la inversa, lo cual garantiza que el modelo y el código estén actualizados, manteniendo la visión en el diseño de más alto nivel de la estructura del proyecto. Es un lenguaje para especificar y no para describir métodos o procesos. Se puede aplicar en una variedad de formas para dar soporte a una metodología de desarrollo de software, pero no especifica qué metodología o proceso utilizar.

### 1.4.3 Herramientas CASE

Visual Paradigm 8.0 es una herramienta CASE[2] enmarcada en traducir los requisitos de una aplicación en calidad de software. (19) Tiene como principio inflexible que un software debe ser fácil de usar, incluso cuando se pretende resolver un problema bastante complejo, por lo que garantiza la usabilidad. Brinda soporte para el modelado de UML y modelado de procesos de negocio y genera código para un gran número de lenguajes de programación. La herramienta permite la integración con varias herramientas de Java como son Eclipse, Netbeans, Jbuilder, además es multiplataforma. Se caracteriza por permitir la transformación de diagramas entidad relación en tablas de bases de datos; por la distribución automática de diagramas; la reorganización de figuras y conectores de los diagramas UML y por la importación y exportación de ficheros XML así como por ser generador de informes para generación de documentación y tener editor de figuras. Esta herramienta CASE se utilizará para dar solución al problema en cuestión.

[2] Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora

### 1.4.4 Lenguaje de programación

Para dar solución al problema se hará uso del lenguaje de programación Java. La versatilidad, eficacia, portabilidad de plataformas y seguridad de la tecnología Java la convierte en la tecnología ideal para la informática de redes. (20) Java es un lenguaje simple; orientado a objeto, por lo que es más fácil su manipulación; distribuido, por lo que permite aceptar y establecer conexiones con los servidores o clientes remotos; proporciona numerosas comprobaciones en compilación y en tiempo de ejecución, por lo que se define como robusto; es seguro, y esta es una característica muy importante en Java pues se han implementado barreras de seguridad tanto en el lenguaje como en el sistema de ejecución en tiempo real; indiferente a la arquitectura; portable, multiplataforma, esencialmente debido a su indiferencia con la arquitectura sobre la cual trabaja; interpretado y compilado a la vez, multihilos, dinámico y de alto rendimiento.

### 1.4.5 Herramientas de desarrollo

Para la definición de la estructura del lenguaje de descripción de trabajo, se estudiaron algunas de las siguientes herramientas:

- **Altova XMLSpy**: esta aplicación permite que el usuario pueda visualizar, validar y editar documentos en el lenguaje XML. Es la herramienta para aquellos que necesiten visualizar sus archivos en el lenguaje XML, DTD, esquemas XML, XSLT y también archivos XQuery. El software posee características como: atractivas impresiones, ayuda con las entradas sensibles al contexto, le auxiliará al completar código, posee una robusta validación y también generación de instancias de ejemplo en el lenguaje XML, y éstas son algunas de las útiles propiedades que le proporciona el programa Altova XMLSpy Standard Edition para realizar una edición básica en el lenguaje XML. (21)

- **Adivo**: TechWriter for XML Schemas es una herramienta de documentación XML que genera documentación para definiciones de esquemas XML (XSDs) y XML DTDs. Utilizando el generador de documentación para esquemas XML de Adivo, usted puede ahorrarse tiempo y dinero, sincronizar la documentación con su esquema, y dirigirse fácilmente a diferentes públicos. El

generador de documentación de TechWriter aprovecha los esquemas XML para producir de forma automática documentos de referencia en los formatos PDF, CHM, RTF y HTML. (22)

- **Liquid Technologies Ltd:** Editor Gráfico XSD y XML. Es un Estudio de Desarrollo de XML completo, incluyendo: un Editor de Esquemas XML gráfico y textual con validación, un editor XML con resaltamiento sintáctico, validación e "intellisense" (captación y sentido inteligente). (23)

Todas estas herramientas tienen varios puntos en común debido a que son especializadas en edición de XML mediante XSD, la seleccionada para la generación del lenguaje XML necesario para solucionar el problema en cuestión es Altova XMLSpy.

- Para dar solución al problema, se va a hacer uso del **NetBeans** 6.9 como herramienta de desarrollo. El IDE NetBeans es un entorno multiplataforma, de código abierto y permiten a los desarrolladores crear rápidamente tanto web como empresa, escritorio y aplicaciones móviles utilizando la plataforma Java, así como PHP, JavaScript, Ajax, Groovy y Grails, y C / C + +.Ofrece un rendimiento y experiencia de codificación, con capacidades de análisis de código estático en el Editor de Java y una inteligente explotación de los proyectos. (24)

## 1.5 Conclusiones parciales

Para dar solución al problema en cuestión se desarrollará la definición de un lenguaje XML, basando su selección en los estudios realizados a los JDL de los distintos middleware existentes para el procesamiento distribuido, tomando parámetros de estos que permitirán la descripción de los problemas. Este lenguaje será procesado, para la realización de cálculos sobre la plataforma T-arenal, por una aplicación genérica. Los problemas descritos en el XML una vez generados en la plataforma serán descompuestos mediante la técnica de descomposición de datos Output para crear los lotes que serán enviados en cada una de las tareas a distribuir. El proceso de desarrollo de la solución propuesta estará guiado por herramientas y tecnologías que permitirán un mejor desarrollo del mismo: como metodología de desarrollo de software, OpenUP; como lenguaje de modelado UML; como herrameinta CASE, Visual Paradigm; como lenguaje de programación Java; se utilizará como IDE de desarrollo, NetBeans. XML Spy será la herramienta utilizada para el diseño del esquema.

# CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

En el presente capítulo se realiza una descripción del sistema a desarrollar para dar solución al problema en cuestión además de un estudio del ambiente en el cual se desarrolla el negocio y será utilizada la aplicación a desarrollar. Partiendo de este estudio se presenta el modelo de dominio con las clases necesarias para reflejar dicho negocio, así como la descripción de estas. Se realiza la especificación de los requisitos tanto funcionales como no funcionales que debe cumplir la solución propuesta. Se obtienen y describen, sobre la base de las entrevistas realizadas a los especialistas del proyecto, la estructura del modelo de Casos de Uso del Sistema (CUS), los actores, los Casos de Uso (CU) y las relaciones que existen entre estos, para luego realizar una descripción detallada de cada uno de ellos.

## 2.1 Descripción del sistema

Partiendo del análisis realizado en el capítulo anterior, referente a determinados problemas del campo de la Bioinformática, se define que debido a las características comunes que presentan, esencialmente reflejadas en que: necesitan de un ejecutable para llevarse a cabo, necesitan de ficheros de entrada que representan los datos que conformaran los lotes de cada tarea, pueden tener ficheros adicionales a los de entrada, y generan ficheros de salida para guardar la información correspondiente a la solución encontrada; las aplicaciones independientes necesarias para generar las tareas de cada problema, problemas similares, en T-arenal tienen características comunes. Por tanto el proceso de desarrollo de cada aplicación es similar, lo que trae consigo emplear tiempo de desarrollo innecesario.

Partiendo de todo esto, la aplicación que se propone como solución al problema en cuestión y que cumplirá con el objetivo general del presente trabajo de diploma, garantiza la ejecución de los problemas anteriormente mencionados sin la necesidad de implementar aplicaciones independientes, partiendo del lenguaje de descripción de trabajo en el que se describirán los problemas, mediante XML (definido luego de un estudio a las diferentes formas de describir problemas, abordado este estudio en el capítulo 1), por lo que la solución propuesta solamente no disminuye el tiempo de desarrollo sino también la complejidad que representa hoy generar una tarea para la plataforma. Aparejado a esto, para facilitar al usuario la descripción de los problemas se propone el desarrollo de una herramienta de apoyo, capaz de generar a

partir de una interfaz amigable, el XML de cualquier problema; evitando así que el usuario necesite de conocimientos referentes al lenguaje XML.

## 2.2 Modelo de dominio

Un modelo de dominio es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Representa clases conceptuales del dominio del problema y los conceptos del mundo real, no de los componentes del software. Una clase conceptual puede ser una idea o un objeto físico (símbolo, definición y extensión). Es un diagrama de clases en el que se muestran (25):

- Conceptos u objetos del dominio del problema: clases conceptuales.
- Asociaciones entre las clases conceptuales.
- Atributos de las clases conceptuales.

### 2.2.1 Representación del modelo de dominio

Luego de haber realizado un estudio del entorno, se determina realizar un modelo de dominio teniendo en cuenta los conceptos u objetos más importantes y las relaciones que existen entre estos.

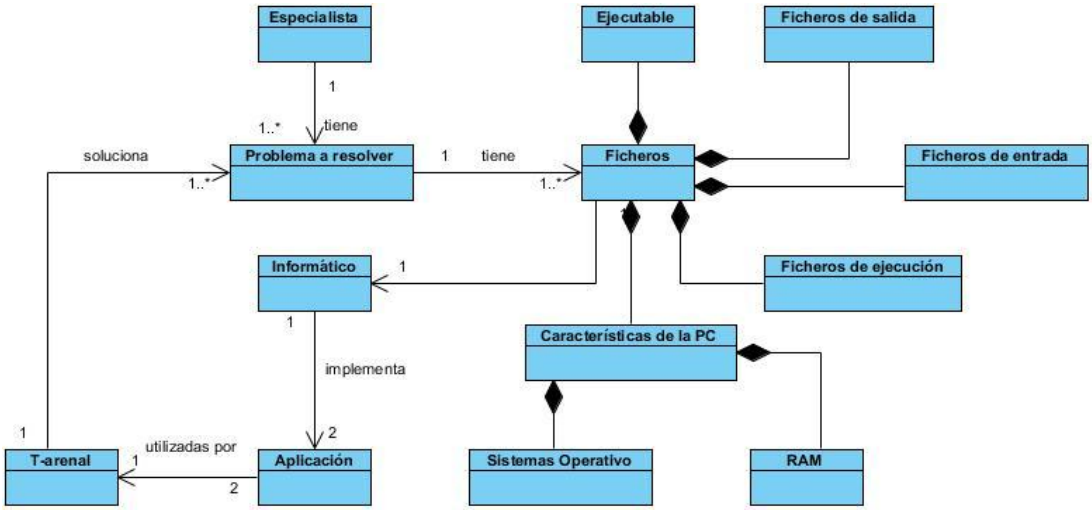


Figura 1: Representación del modelo de dominio.

.

**Definición de las clases del modelo de dominio**

- **Especialista:** persona que tiene los problemas a solucionar sobre la plataforma T-arenal.
- **Problema a resolver:** problema de la Bioinformática que necesitan ser resueltos, y demandan gran capacidad de cómputo para esto.
- **Ficheros:** conjunto de ficheros que necesita el problema para ejecutarse.
- **Ejecutable:** fichero que permite ejecutar el problema.
- **Ficheros de entrada:** conjunto de ficheros de entrada que necesita el problema para su ejecución.
- **Ficheros de ejecución:** ficheros adicionales que necesita el problema para su ejecución independientemente de los ficheros de entrada.
- **Características de la PC:** características que debe cumplir la PC cliente donde se va a ejecutar el problema.
- **Sistema operativo:** sistema operativo que requiere el problema para poder ser ejecutado.
- **RAM:** RAM mínima o conjuntos de estas que demanda el problema para su ejecución.
- **Informático:** persona que, a partir de un estudio realizado a los ficheros que tiene el problema, implementa las clases requeridas por T-arenal, para distribuir cada uno, en búsqueda de su solución.
- **Clases:** la clase DataManager y la clase Task, implementadas por un informático, a partir del análisis realizado a los ficheros del problema, necesarias para la ejecución del mismo, en búsqueda de su solición.
- **T-arenal:** Plataforma de Tareas Distribuidas, encargada de distribuir los problemas a resolver, a partir de las clases DataManager y Task, y de los ficheros de estos, para darles solución.

## 2.3 Especificación de requisitos

Frecuentemente, los ingenieros en el desarrollo de software se encuentran enfrentando situaciones en las que los usuarios no saben lo que quieren, ni cómo detallar lo que conocen de forma precisa y muchas veces desconocen qué partes de su trabajo pueden transformarse en software.

Sobre la base del análisis de la representación de los conceptos u objetos del mundo real mediante el modelo de dominio, las entrevistas realizadas a los especialistas del proyecto, expectativas del desarrollo del sistema, la necesidad de realizar una correcta captura de los requisitos, para evitar demoras en la

construcción del módulo y errores en los mismos, fueron identificados los requisitos que deben cumplir tanto la aplicación genérica como la herramienta de apoyo, los cuales se presentan en los subepígrafes que se relacionan a continuación.

### 2.3.1 Requisitos funcionales

Los requisitos funcionales son las capacidades o condiciones que el sistema debe cumplir, para que el usuario resuelva su problema o cumpla sus objetivos, es decir, especifican el comportamiento de entrada y salida del sistema y surgen de la razón fundamental de la existencia del producto. Deben estar claros y libres de ambigüedades, asegurando que los involucrados comprendan claramente el significado de cada uno.

**Requisitos funcionales relacionados con el procesamiento de los problemas tipo sobre la plataforma T-arenal**

- **RF1** Validar la estructura de un XML.
- **RF2** Guardar los datos del XML correcto.
- **RF3** Chequear compatibilidad de características.
- **RF4** Crear una unidad de trabajo.
- **RF5** Procesar una unidad de trabajo.
- **RF6** Procesar resultados de una unidad de trabajo.
- **RF7** Terminar ejecución.

**Requisitos funcionales relacionados con el proceso de generación del XML.**

- **RF8** Adicionar ejecutable.
- **RF9** Adicionar argumentos.
- **RF10** Adicionar ficheros de entrada.
- **RF11** Adicionar ficheros de ejecución.
- **RF12** Adicionar ficheros de salida que serán creados
- **RF13** Adicionar ficheros de salida que serán guardados
- **RF14** Adicionar RAM.

- **RF15** Adicionar sistema operativo.

### 2.3.2 Requisitos no funcionales

Los requisitos no funcionales son las propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido y confiable. Normalmente están vinculados a requerimientos funcionales, es decir, una vez que se conozca lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser. Los requerimientos no funcionales también añaden funcionalidad al producto, pues hacen que un producto sea fácil de usar, seguro, o demanda cierta cantidad de procesamiento.

- **RNF1. Transparencia**

  **Aplicación genérica:** el usuario tiene una abstracción completa del procesamiento que realiza la aplicación para dar solución al problema que este desea ejecutar, por lo que solamente tiene que especificar los parámetros necesarios para dicho procesamiento.

  **Herramienta de apoyo:** el usuario tiene una abstracción completa del proceso que realiza la herramienta para generar el XML, solamente tiene que ir especificando los parámetros que requiere el mismo mediante una interfaz amigable.

- **RNF2. Fiabilidad**

  **Aplicación genérica:** asegura al 100% que el problema del usuario será resuelto, independientemente de los problemas ajenos que existan, ya sean fallos de red, electricidad, apagado de máquinas, entre otros.

- **RNF3. Software:** para el uso del sistema se deberá disponer de la máquina virtual de Java 1.5 en todas las PCs o superior, y T-arenal deberá estar instalado en cada una de las PCs que se utilizarán.

- **RNF4. Apariencia o interfaz externa**

  **Herramienta de apoyo:** se desea que la interfaz externa del producto sea de fácil navegación por el usuario, sencilla y legible, que mantenga los estándares y características de la plataforma soberana T-arenal como son: los colores, la estructura de los botones, el estilo y tamaño de letra.

- **RNF5. Portabilidad:** ambas soluciones serán multiplataforma, razón por la cual podrán ser utilizados en cualquier sistema operativo y arquitectura de hardware.

- **RNF6. Usabilidad:** se debe lograr un diseño adaptable, con la capacidad de poder soportar funcionalidades adicionales o modificar las funcionalidades existentes sin impactar el resto de los requerimientos contemplados en el sistema.

- **RNF7. Soporte:** ambas soluciones estarán bien documentadas de forma tal que el tiempo de mantenimiento sea mínimo en caso de necesitarlo, permitiendo implementar cambios, ya sea de corrección, mejora o adaptación del software.

## 2.4 Descripción del modelo de Casos de Uso del Sistema.

El modelo de Casos de Uso del Sistema es un modelo que contiene actores, casos de uso y sus relaciones; describe lo que el sistema debería hacer por sus usuarios y bajo qué restricciones. Permite que los desarrolladores y clientes lleguen a un acuerdo sobre los requerimientos, y proporciona la entrada fundamental para el análisis, diseño y las pruebas.

### 2.4.1 Actores del sistema

Un actor es un usuario del sistema. Esto incluye usuarios humanos y otros sistemas computacionales. El conjunto de casos de uso al que un actor tiene acceso define un rol en el sistema y el alcance de su acción. A continuación, se mencionan los actores que van a interactuar con el sistema a construir, definiendo el rol que le ocupa dentro del mismo.

Tabla 1: Descripción de los actores.

| Actor | Descripción |
|---|---|
| **Especialista** | Representa los usuarios que van a interactuar con las soluciones propuestas, para iniciar la ejecución de un problema y/o generar el XML. |
| **Cliente de T-arenal** | Representa la aplicación encargada de realizar solicitudes de tareas a un servidor de peticiones para su procesamiento. |

### 2.4.2 Listado de casos de uso del sistema

La forma en que los actores usan un sistema es representada a través de los casos de uso, los cuales van a ser fragmentos de funcionalidad que ofrece el sistema para aportar un resultado de valor para sus actores, y especifican una secuencia de acciones que el sistema debe llevar a cabo mediante la interacción con sus actores, incluyendo alternativas dentro de una secuencia.

Tabla 2: Casos de Uso.

| Referencia a requisitos | Nombre del caso de uso | Prioridad |
|---|---|---|
| **RF: 1 y 2** | Almacenar descripción del problema. | Crítico |
| **RF: 3, 4, 6 y 7** | Atender tarea asignada. | Crítico |
| **RF: 5** | Realizar procesamiento. | Crítico |
| **RF: 8 - 15** | Crear XML. | Secundario |

Tabla 3: Breve descripción de los CU.

| Orden | Nombre | Prioridad | Breve descripción |
|---|---|---|---|
| 1 | Almacenar descripción del problema. | Crítico | El caso de uso se inicia cuando el especialista desea ejecutar un problema sobre la plataforma. Termina cuando se encuentra la descripción correcta y se almacenan los datos |

| | | | |
|---|---|---|---|
| | | | de esta. |
| 2 | Atender tarea asignada. | Crítico | El caso de uso inicia cuando un cliente hace, al servidor de peticiones correspondiente, una solicitud de procesamiento o devuelve los resultados obtenidos. Termina cuando es creada una anidad de trabajo. |
| 3 | Realizar procesamiento. | Crítico | El caso de uso inicia cuando un cliente hace una solicitud de unidad de trabajo al servidor de peticiones correspondiente para luego realizar su procesamiento. Termina cuando el cliente ha terminado de procesar la unidad de trabajo. |
| 4 | Crear XML. | Secundario | El caso de uso inicia cuando el especialista desea generar la descripción correcta de un problema. Termina cuando el usuario ha introducido todos los datos correctamente y es creado el XML. |

### 2.4.3 Diagrama de Casos de Uso del Sistema

Un diagrama de Casos de Uso del Sistema (CUS) contiene actores, Casos de Uso (CU) y las relaciones existentes entre los mismos. En la siguiente figura se muestra el diagrama de CUS correspondiente a la solución del problema planteado.

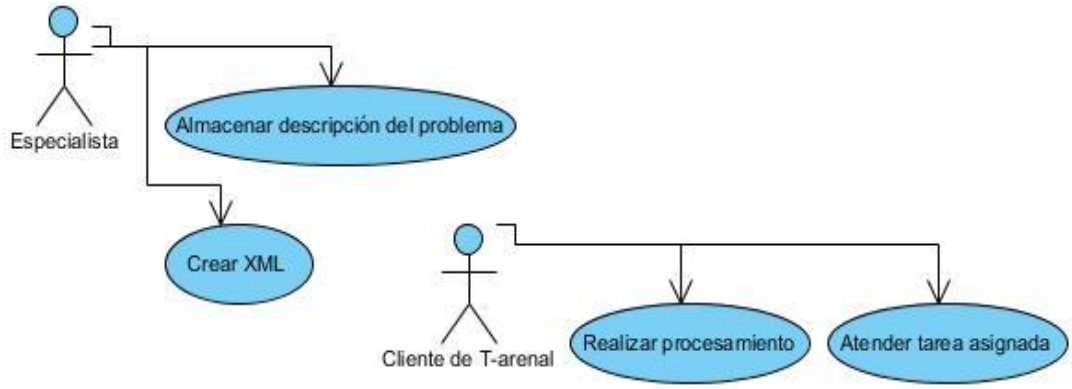


Figura 2: Diagrama de CU.

### 2.4.4 Descripción de Casos de Uso del Sistema

Debido a la que las descripciones textuales de los CU son extensas, se propone la descripción del caso de uso Atender tarea asignada y la de los restantes en los Anexos.

Tabla 4: Descripción del CU Atender tarea asignada.

| **Caso de Uso:** | Atender tarea asignada |
|---|---|
| **Actores:** | Cliente de T-arenal |
| **Propósito** | Atender en el servidor de peticiones la tarea asignada por el servidor central. |
| **Resumen:** | El caso de uso inicia cuando un cliente hace, al servidor de peticiones correspondiente, una solicitud de procesamiento o devuelve los resultados obtenidos. Termina cuando es creada una anidad de trabajo. |
| **Precondiciones:** | Tiene que existir un XML que describa correctamente el problema. |
| **Referencias** | RF3, RF4, RF6 y RF7. |
| **Prioridad** | Crítico |
| **Flujo Normal de Eventos** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| **1.** El cliente realiza una solicitud de:<br>• Obtener una unidad de trabajo. Ver sección Crear unidad de trabajo. | |

| | |
|---|---|
| • Enviar el resultado del procesamiento realizado. Ver sección Procesar resultado de una unidad de trabajo. | |
| **Sección: Crear unidad de trabajo** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| **1.** El cliente solicita una unidad de trabajo para su procesamiento. | **2.** El servidor de peticiones valida que las características del cliente satisfagan las demandadas por el problema para su ejecución.<br>**3.** El servidor de peticiones selecciona el ejecutable de la unidad de trabajo a crear según las características del cliente que la solicita.<br>**4.** El servidor de peticiones almacena:<br>• Si el ejecutable es descargable o no.<br>• Los argumentos que determinarán la línea de comando que ejecutará el problema en cada cliente.<br>• Los ficheros de entrada que serán enviados al cliente, para la ejecución de la unidad de trabajo.<br>• Los ficheros de ejecución que necesita el problema.<br>• El nombre de los ficheros de salida que debe crear el sistema para recoger los datos generados por el procesamiento de la unidad de trabajo.<br>• Los ficheros de salida que el Especialista desea sean guardados por el sistema.<br>**5.** El servidor de peticiones guarda en un vector la información referente a la unidad de trabajo creada y se la envía al cliente que la solicitó.<br>**6.** El servidor de peticiones adiciona la unidad creada a la lista de unidades pendientes de respuesta. |

| | |
|---|---|
| | **7.** El servidor de peticiones finalmente retorna la unidad generada. Finaliza el caso de uso. |
| **Flujo Alterno** | |
| | **2.1** Si las características del cliente no satisfacen las demandadas por el problema, el servidor de peticiones retorna una unidad de trabajo nula.<br>**2.2** Si no se puede generar ninguna unidad de trabajo el servidor de peticiones termina la ejecución. Finaliza el caso de uso. |
| **Sección: Procesar resultado de una unidad de trabajo** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| **1.** El cliente solicita enviar el resultado obtenido del procesamiento de una unidad de trabajo. | **2.** El servidor de peticiones crea una carpeta en el directorio asignado.<br>**3.** El servidor de peticiones recibe el nombre de los ficheros de salida que debe crear en esa carpeta y los datos de estos ficheros.<br>**4.** El servidor de peticiones crea los ficheros de salida en el directorio y escribe los datos en ellos.<br>**5.** El servidor de peticiones elimina la unidad de trabajo del conjunto de unidades pendientes.<br>**6.** El servidor de peticiones verifica que no queden unidades pendientes.<br>**7.** El servidor de peticiones termina la ejecución y pasa a comprimir el directorio de trabajo asignado a la ejecución. Finaliza el caso de uso. |
| **Flujo Alterno** | |
| | **6.1** El servidor de peticiones espera a procesar todas las unidades pendientes.<br>**6.2** El servidor de peticiones termina la ejecución y pasa a |

| | |
|---|---|
| | comprimir el directorio de trabajo asignado a la ejecución. Finaliza el caso de uso. |

## 2.5 Conclusiones parciales

La presentación de la solución tecnológica tanto para la aplicación que permita ejecutar tareas sobre la plataforma así como la herramienta de apoyo que permita generar el XML de cualquier problema, realizada en el presente capítulo, y partiendo del análisis del negocio, así como de los artefactos generados en los flujos de trabajo pertenecientes a la metodología utilizada, han permitido que se puedan arribar a las siguientes conclusiones: 1) la captura e identificación correcta de los requerimientos funcionales del sistema permite que se obtenga un producto, lo más consecuente con las solicitudes del cliente; 2) la identificación adecuada de los requisitos no funcionales permite que no ocurran retrasos en la construcción de las soluciones por fallos en algunas de las aplicaciones o servidores, se mantenga la seguridad de la información y en caso de pérdida de la misma, se pueda restablecer y recuperar; 3) la identificación de los casos de uso a partir de los requerimientos funcionales y la descripción de los mismos, permite que se tenga una mayor visión de la solución que se quiere construir, el comportamiento de la misma y la secuencia de actividades que debe de realizar el usuario para lograr su objetivo.

# CAPÍTULO 3: DISEÑO DEL SISTEMA

Cuando se inicia la construcción de un sistema de software, luego de conocer cuáles son las necesidades básicas del usuario final, el ambiente donde será desarrollado el mismo, las condiciones para ello, las tecnologías y recursos necesarios para lograr un ambiente de trabajo adecuado, entonces se sientan las bases del esqueleto arquitectónico del sistema. En el presente capítulo se realiza una propuesta de la vista arquitectónica tanto para la aplicación genérica para ejecutar tareas sobre la plataforma como para la herramienta de apoyo para generar XML; basada fundamentalmente en los patrones de diseño de GRASP. Luego son creadas las condiciones para pasar directamente al modelo de diseño del sistema, justificando de manera concreta que el uso de la metodología OpenUp permite que se pueda prescindir del modelo de análisis, si se ha logrado que se haya adquirido una comprensión de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia. Se provee una propuesta del diagrama de clases del diseño que permite representar las clases y las relaciones entre estas.

## 3.1. Arquitectura de software

Según la IEEE la Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución. La arquitectura del software de un programa o sistema de cómputo es la estructura o las estructuras del sistema, que incluyen los componentes del software, las propiedades visibles externamente de esos componentes y las relaciones entre ellos. Es una representación que permite analizar la efectividad del diseño para cumplir con los requisitos establecidos, considerar opciones arquitectónicas en una etapa en que aún resulta relativamente fácil hacer cambios al diseño y reducir los riesgos asociados a la construcción del software.

### 3.1.1 Estilo arquitectónico

Un estilo arquitectónico o variante arquitectónica define a una familia de sistemas informáticos en términos de su organización estructural. Un estilo arquitectónico describe los componentes y las relaciones entre ellos con las restricciones de su aplicación, la composición asociada y el diseño para su construcción (26).

**Aplicación genérica**

Debido a que la aplicación genérica a desarrollar como parte de la solución propuesta, es un aplicación para la Plataforma de Tareas Distribuidas, se ajusta al estilo de arquitectura de la misma, que es el cliente – servidor, así como el patrón arquitectónico utilizado que es Mediador (Mediator), los cuales son explicados detalladamente en (4).

**Herramienta de apoyo**

El estilo arquitectónico que sigue la herramienta de apoyo propuesta es el de llamada y retorno. Unas de las principales ventajas que enfatiza esta familia de estilos son la modificabilidad y la escalabilidad, lo cual se ajusta a las necesidades de la investigación ya que se desea desarrollar una herramienta que permita a partir de la distribución de sus componentes y las relaciones existentes entre ellos, realizar modificaciones, de ser necesarias, en alguna de las capas sin que sean afectadas las restantes, contribuyendo a la reutilización de las clases que forman parte de la herramienta en cuestión, así como la posibilidad de agregarle funcionalidades sin que exista algún tipo de afectación. Dentro del mismo, se utiliza la arquitectura en capas una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.

Específicamente se presenta la arquitectura en dos capas: presentación y lógica de negocio. En la capa de presentación se encuentra la clase que representa la interfaz gráfica de usuario y en la lógica de negocio las clases controladoras y de lógica de negocio. La capa intermedia (lógica de aplicación) es básicamente el código al que recurre la capa de presentación para recuperar los datos deseados. La capa de presentación recibe entonces los datos y los formatea para su presentación.

Esta separación entre la lógica de aplicación de la interfaz de usuario añade una enorme flexibilidad al diseño de la aplicación. Pueden construirse y desplegarse múltiples interfaces de usuario sin cambiar en

absoluto la lógica de aplicación siempre que está presente una interfaz claramente definida a la capa de presentación.

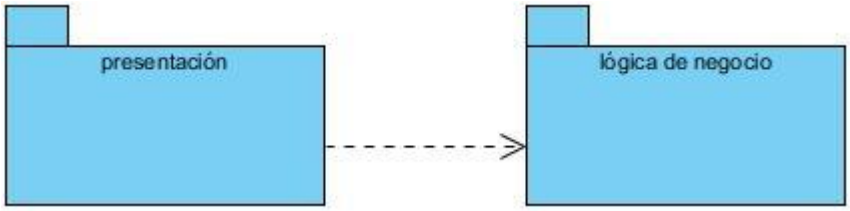


Figura 3: Estilo de arquitectura en 2 capas aplicado a la herramienta de apoyo.

### 3.1.2 Patrones de diseño

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular. Identifica: clases, instancias, roles, colaboraciones y la distribución de responsabilidades.

Es una solución estándar para un problema común de programación. Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios. Un proyecto o estructura de implementación que logra una finalidad determinada. Un lenguaje de programación de alto nivel. Una manera más práctica de describir ciertos aspectos de la organización de un programa.

Los patrones de diseño pretenden:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas de software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

En el diseño de la solución propuesta se aplican los patrones GRASP[3]. A continuación se mencionan los patrones utilizados y algunos ejemplos por cada uno de ellos:

**Experto:** Se aplica para asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. En este caso se tienen las clases:

- La clase ReadingXML, que es la responsable de leer y almacenar todos los valores de las etiquetas existentes en el XML que describe el problema analizado y la clase ValidatingXML, experta en validar que el XML analizado cumpla con las características definidas. Clases definidas para la aplicación genérica a implementar.

- Files, experta en manejar los datos necesarios de todos los ficheros que requiere el problema, para crear las etiquetas correspondientes en el XML; la clase FeaturesPC, responsable de manejar todos los datos necesarios de las características que demanda el problema, para crear las etiquetas correspondientes a dichas características en el XML que lo describe y la clase GeneralConfiguration, experta en manejar los datos referentes al ejecutable y los argumentos del problema, para crear las etiquetas correspondientes a estas descripciones del mismo en el XML. Clases definidas para el generador de XML.

El uso de este patrón permite que se conserve el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento.

**Creador:** Se aplica para la asignación de responsabilidades a las clases relacionadas con la creación de objetos, de forma tal que una instancia de un objeto sólo pueda ser creada por el objeto que contiene la información necesaria para ello. En este caso se tienen las clases:

- GenericDataManager, encargada de crear una instancia de las clases ValidatingXML, ReadingXML, CreateWU y Filter, clases que describen las variables que contienen la información

[3] GRASP es un acrónimo que significa General Responsibility Asignment Software Patterns (Patrones Generales de Software para Asignar Responsabilidades)

necesaria para el desarrollo de la clase Creador. Clases definidas para la aplicación genérica a implementar.

- CreateXML, encargada de crear una instancia de las clases Files, FeaturesPC y GenaralConfiguration, clases que describen las variables que contienen información necesaria para el desarrollo de la clase Creador. Clase definida para el generador de XML.

**Bajo Acoplamiento:** El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, de cómo se conocen y de cómo recurre a ellas. En este caso se tienen las clases:

- Task, ErrorH y Filter, son fáciles de entender por separado y fáciles de reutilizar. Clases definidas para la aplicación genérica a implementar.

**Alta Cohesión.** La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. De manera que todas posean la característica de tener las responsabilidades estrechamente relacionadas y que no realicen un trabajo enorme. El uso de este patrón permite que se pueda mejorar la claridad y facilidad con que se entiende el diseño, se simplifique el mantenimiento y existan mejoras de funcionalidad. En este caso se tienen las clases:

- ValidatingXML, enmarcada solamente a realizar la validación del XML analizado. Clases definidas para la aplicación genérica a implementar.

- CreateXML, enmarcada solamente en generar los datos y etiquetas genéricas del XML. Clase definida para el generador de XML.

**Controlador:** Un controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema. Define además el método de su operación. En este caso se tienen las clases:

- GenericDataManager, ya que es la encargada de iniciar las ejecuciones de casi todas las restantes clases, así como administrar los resultados que dichas ejecuciones arrojan. Clases definidas para la aplicación genérica a implementar.

- CreateXML, encargada de ejecutar las restantes clases del generador, y manejar las funcionalidades de las mismas. Clase definida para el generador de XML.

## 3.2 Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso, centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Los subsistemas de diseño y las clases de diseño representan abstracciones del subsistema y componentes de la implementación del sistema.

### 3.2.1 Diagrama de clases

Las clases del diseño representan una abstracción de una o varias clases (o construcción similar) en la implementación del sistema. El lenguaje utilizado para especificar una clase del diseño es el mismo que el lenguaje de programación utilizado, los métodos tienen correspondencia directa con el correspondiente método de la implementación de clases, puede aparecer como un estereotipo que se corresponde con una construcción en el lenguaje de programación dado. Teniendo en cuenta lo antes expuesto, se propone el diagrama de clases del diseño para la aplicación genérica y la herramienta de apoyo.

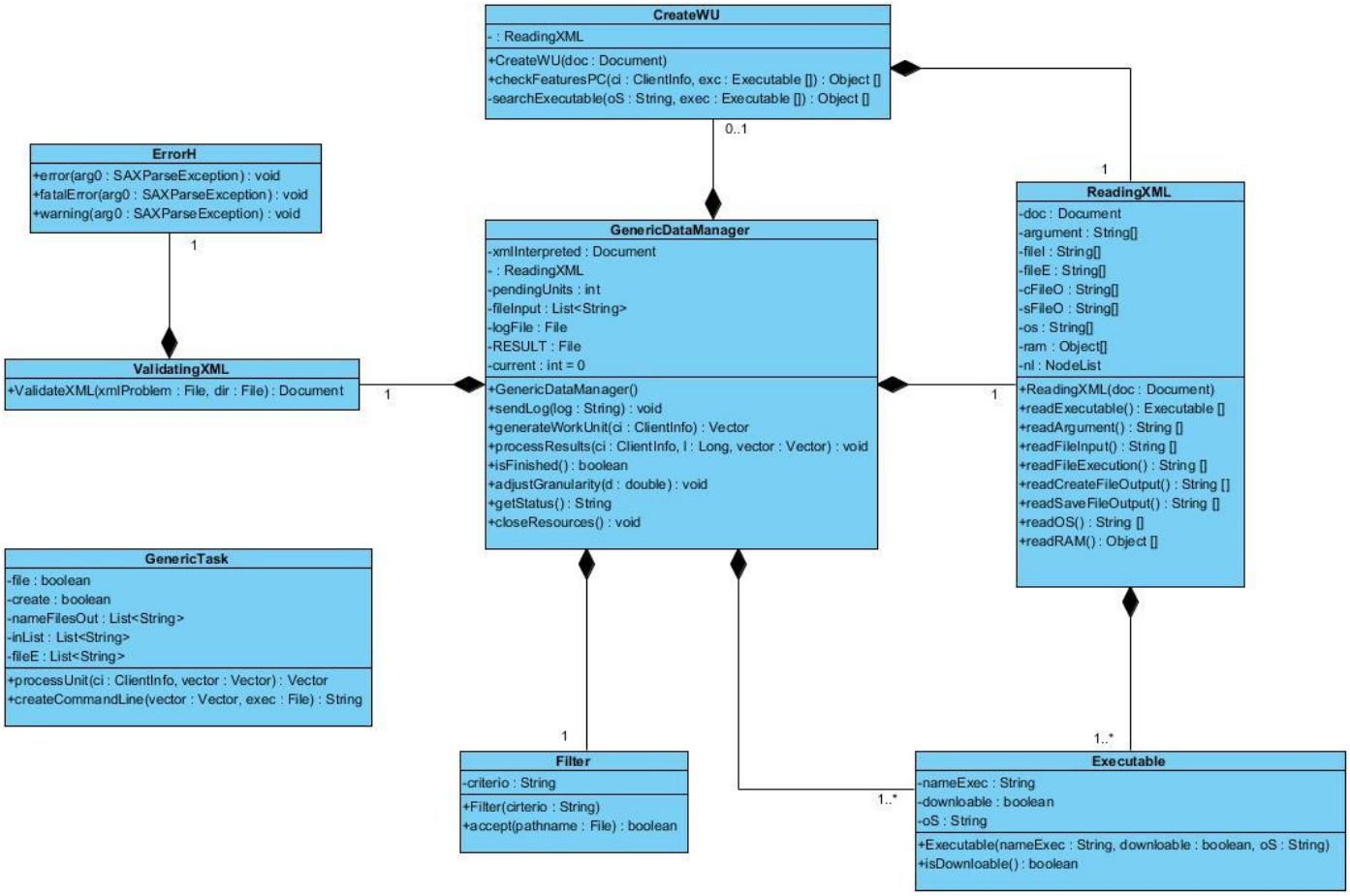


Figura 4: Diagrama de clases de la aplicación genérica.

El diagrama de clases del diseño de la Figura 4 está integrado por la clase GenericDataManager que es la encargada de guiar todo el proceso para la generación de tareas para la plataforma. Esta clase está relacionada estrechamente con casi todas las restantes. La misma instancia la clase ValidatingXML, para iniciar el proceso de búsqueda y validación del XML correcto, posteriormente utiliza el atributo de tipo ReadingXML, para leer el XML encontrado, y así poder instanciar la clase CreateWU en búsqueda de compatibilidad de características entre el problema a procesar y la PC cliente que llevará a cabo este proceso, utiliza además una instancia de la clase Executable para poder enviar este objeto hacia el cliente que ejecutará la tarea; esta clase también utiliza un objeto de la clase Filter para definir los parámetros

necesarios de utilización de los filtros. A su vez, ReadingXML tiene como atributo un objeto de la clase Executable, que le permite crear todos los ejecutables tenga el problema.

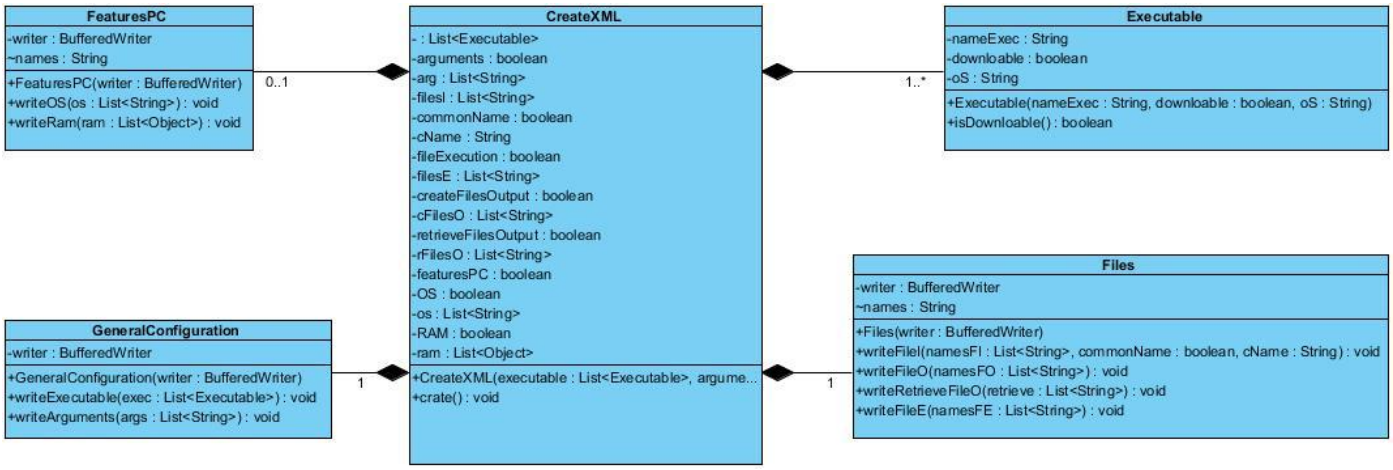


Figura 5: Diagrama de clases de la herramienta de apoyo.

El diagrama de clases del diseño de la Figura 5 está integrado por la clase CreateXML, responsable de instanciar las clases Files, FeaturesPC y GeneralConfiguration, con el objetivo de utilizar estas instancias para generar el XML correspondiente al problema que el usuario está describiendo. Esta clase también cuenta con un atributo de tipo Executable, que garantiza escribir correctamente en el XML los datos especificados por el usuario.

## 3.3 Conclusiones parciales

Como resultados importantes del presente capítulo se tienen: 1) el haber definido primeramente los aspectos fundamentales desde el punto de vista arquitectónico para la aplicación a desarrollar, como lo son los patrones de diseño, que permitieron analizar la efectividad del diseño y cumplir con los requisitos establecidos; 2) el haber presentado el modelo de diseño, dónde se define el diagrama de clases del diseño de toda la aplicación a desarrollar y además de la herramienta de apoyo, el generador de XML. En ambos casos el papel más importante del modelo presentado es que permitió mostrar una abstracción de las clases que posteriormente serán definidas durante la implementación.

# CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA

El presente capítulo está enfocado a la implementación y prueba de las funcionalidades para desarrollar un sistema completo. Partiendo de los resultados obtenidos en capítulos anteriores, el presente tiene como propósito clarificar los requisitos restantes y completar el desarrollo de la aplicación, por lo que se puede ver como un proceso de fabricación, en el que se pone el énfasis en la gestión de los recursos y el control de las operaciones para optimizar la planificación y la calidad, asegurando esta con la realización de pruebas, puesto que son un elemento crítico para la garantía de la misma, y representan además una revisión final de todo el proceso de diseño y desarrollo.

## 4.1 Diagrama de componentes

Un componente es una unidad modular que puede reemplazarse en su propio entorno. Sus elementos internos quedan ocultos, pero tiene una o varias interfaces proporcionadas bien definidas a través de las cuales se puede obtener acceso a sus funciones. El diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes de software, ayuda al equipo de desarrollo a entender un diseño existente y crear uno nuevo. Al pensar en el sistema como una colección de componentes con interfaces proporcionadas y necesarias, bien definidas, se mejora la separación entre los componentes. Esto, a su vez, facilita la comprensión y los cambios cuando se modifican los requisitos.

En la figura 6 se muestra la integración de los componentes de la solución propuesta con los componentes de T-arenal que intervienen en la misma, el resto de los componentes de la plataforma, así como la descripción detallada del diagrama de componentes de la misma se puede encontrar en (4).

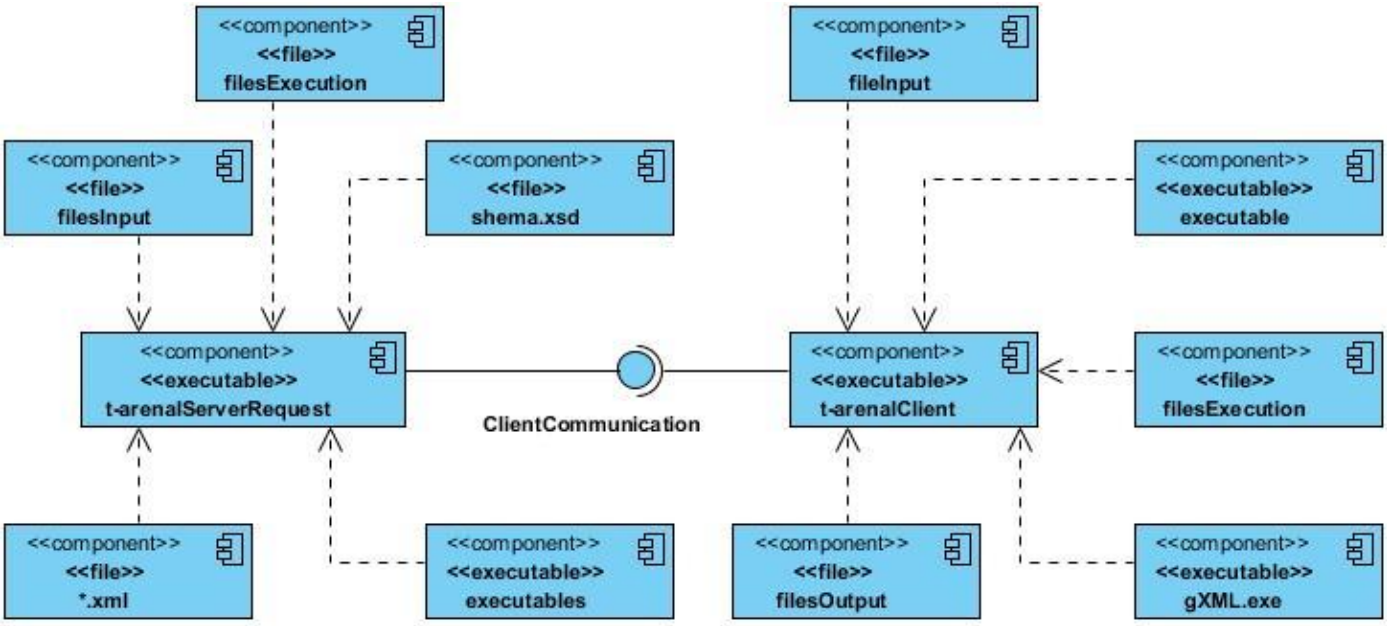


Figura 6: Diagrama de componentes del sistema.

La descripción de diagrama presentado en la figura 6, y que se presenta a continuación, estará dividida en dos partes, cada una agrupada por los componentes pertenecientes a la plataforma, en cada una de las partes se explicarán las descripciones de los componentes pertenecientes a la aplicación genérica a desarrollar.

El *t-arenalServerRequest* (servidor de peticiones) es un componente ejecutable en el cuál se encuentran los componentes *schema.xmd* (esquema), que representa el fichero que establece el esquema correcto para validar la estructura de la descripción del problema a ejecutar; esta descripción se encontrará especificada en un fichero con extensión xml, se podrán encontrar además otros ficheros con esta misma extensión, representando configuraciones de los problemas, estos tipos de ficheros se encuentran agrupados en el componente **.xml*. Otro de los componentes relacionados con *t-arenalServerRequest* es el *filesInput* (ficheros de entrada), que especifica un conjunto de ficheros de entrada que necesita el problema para ser ejecutado; de la misma manera se encuentran relacionados con el *t-arenalServerRequest* todos los ejecutables que pueda tener un problema, representados en el componente *executables* (ejecutables); y el último de los componentes relacionados es el *filesExecution*

(ficheros de ejecución), que representa todos los ficheros que necesita el problema para su ejecución, sin incluir los de entrada.

En el *t-arenalClient*, mencionado con anterioridad, se encuentran los componentes *filesOutput* (ficheros de salida), que representa el conjunto de ficheros de salida que genera cada tarea al ser ejecutada; otro componente relacionado es el *filesExecution* (ficheros de ejecución), que se refiere a todos los ficheros que necesita una tarea para ejecutarse, que no se encuentran dentro de los ficheros de entrada (componente *filesInput*); el componente *fileInput* (fichero de entrada), referente a un fichero de entrada de la lista de ficheros que necesita una tarea para ejecutarse, en este caso, como estamos en el componente a ejecutar en el cliente (*t-arenalClient*) se trata solamente de un fichero de entrada el requerido; el componente *executable* (ejecutable), que representa el ejecutables del problema a ejecutar en el cliente; y finalmente el componente *gXML.exe,* referente a la herramienta de apoyo capaz de generar el XML de determinados problemas.

## 4.2 Diagrama de despliegue

El diagrama de despliegue es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos, permitiendo modelar mediante el mismo la vista de despliegue estática, describir la arquitectura en tiempo de ejecución de procesadores, dispositivos y los componentes de software que ejecutan esta arquitectura, además de la topología del sistema, estructura de hardware y el software que se ejecuta en cada unidad. Seguidamente se representan los nodos del diagrama de despliegue de T-arenal (4) en los que estarán ambas soluciones.

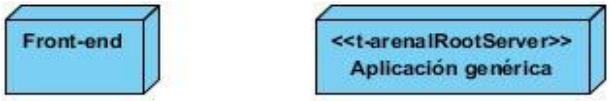


Figura 7: Representación del despliegue.

### 4.2.1 Descripción de los nodos

- Nodo Front-end: es el nodo donde una persona, software o componente de software externo al sistema interactúa con este. En este nodo se encontrará la herramienta de apoyo para que el usuario genere el XML del problema que desea resolver.

- Nodo Aplicación genérica: es el nodo donde se encuentra la aplicación genérica que generará tareas por cada uno de los problemas a resolver en la plataforma, este nodo representa al servidor central de T-arenal; para un mejor entendimiento se encuentra detallado el diagrama de despliegue de la plataforma en (4).

## 4.3 Pruebas al sistema

El flujo de trabajo correspondiente a las pruebas es uno de los más importantes del ciclo de desarrollo del producto de software, porque es el encargada de validar si el producto desarrollado cumple las expectativas funcionales, proporcionando la información necesaria sobre la calidad del producto, detectando los posibles errores que se puedan generar al hacer uso del mismo antes de que sea entregado a un usuario final. Seguidamente se exponen tres pruebas realizadas: caja blanca, correspondiente a la aplicación genérica; caja negra, correspondiente a la herramienta de apoyo; y estudio de caso, correspondiente al conjunto de ambas soluciones, puesto que es realizada con el fin de verificar que la solución desarrollada contribuye a disminuir el tiempo de desarrollo, objetivo del presente trabajo de diploma.

### 4.3.1 Pruebas de caja blanca

Las pruebas de caja blanca se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente, básicamente se escogen distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa y cerciorarse de que se devuelven los valores de salida adecuados, ya que su cometido es probar los flujos de ejecución dentro de cada unidad. Se comprueban los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado.

Para realizar las pruebas de la aplicación en cuestión se utilizó la prueba del camino básico que es una técnica de prueba del método de caja blanca. Este método permite obtener una medida de la complejidad lógica de un diseño procedimental y usarla como guía para la definición de un conjunto básico de caminos de ejecución, que permite realizar por lo menos una vez cada sentencia del programa. A continuación se presenta el fragmento del código seleccionado.

```
public String createCommandLine(Vector vector, File exec) throws IOException {

        String cmd = " ";
        int count = 0;
        String[] arguments = (String[]) vector.get(2);

        for (int i = 0; i < arguments.length; i++) {

          if (arguments[i].contains("$TARENAL_HOME")) {

              cmd = cmd + PROBLEMDIRECTORY + " ";

          } else if (arguments[i].contains("$FILE_INPUT")) {

              cmd = cmd + new File(PROBLEMDIRECTORY, inList.get(count)).getName() + " ";
              count++;

          } else {

              cmd = cmd + arguments[i] + " ";

          }
        }
        .
        .
        .
        cmd = cmd.substring(0, cmd.length() - 1);
        return cmd;
    }
}
```

1
2
3
4
5
6
7
8

Figura 8: Método *createCommandLine*

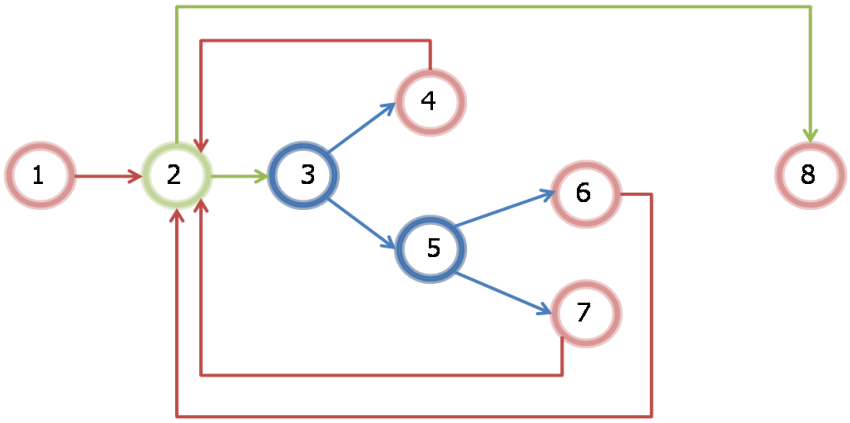


Figura 9: Grafo de flujo del método *createCommandLine.*

Teniendo ya el código seleccionado, con los nodos identificados y el grafo que define el método del camino básico, se procede a definir la complejidad ciclomática.

La complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y nos da un límite inferior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez.

Un **camino independiente** es cualquier camino del programa que introduce por lo menos un nuevo conjunto de sentencias de procesamiento o una nueva condición. En términos del grafo de flujo, un camino independiente se debe mover por lo menos por una arista que no haya sido recorrida anteriormente a la definición de un camino.

La **complejidad ciclomática V (G)** se puede calcular de tres formas:

**1.** El número de regiones del grafo de flujo coincide con la complejidad ciclomática.

**2.** Aristas - Nodos + 2, es decir **V (G) = A – N + 2**.

**3.** Nodos Predicado + 1 (un nodo predicado es el que representa una condicional if o case, es decir, que de él salen varios caminos).

**Por tanto:**

1. **V (G)** = 3

2. **V (G)** = 10 – 8 + 2 = 3

3. **V (G)** = 3

**Caminos de prueba**:

Camino 1: 1 – 2 – 3 – 4 – 2 – 8
Camino 2: 1 – 2 – 3 – 5 – 6 – 2– 8
Camino 3: 1 – 2 – 3 – 5 – 7 – 2– 8

Tabla 5: Descripción de los Casos de prueba.

| Camino | Descripción | Entrada | Salida |
|---|---|---|---|
| **1** | Mientras se está cumpliendo la condición del ciclo, este | El valor del argumento analizado es: $TARENAL_HOME | Escribir en la línea de comando: *PROBLEMDIRECTORY* |

| | | | |
|---|---|---|---|
| | camino se realiza si el valor que se está leyendo hace referencia al directorio de T-arenal. | | |
| **2** | Mientras se está cumpliendo la condición del ciclo, este camino se realiza si el valor que se está analizando hace referencia a un fichero de entrada. | El valor del argumento analizado es: $FILE_INPUT | Escribir en la línea de comando el nombre del fichero de entrada que corresponde analizar. |
| **3** | Mientras se está cumpliendo la condición del ciclo, este camino se realiza si el valor que se está analizando es una cadena cualquiera. | El valor del argumento analizado es una cadena de texto cualquiera. | Escribir en la línea de comando la cadena de texto que corresponde. |

### 4.3.2 Pruebas de caja negra

Las pruebas de caja negra permiten comprobar la operatividad de cada función, es decir, los valores de entradas se aceptan de forma adecuada y se produce una salida correcta. Estas pruebas se llevan a cabo sobre la interfaz del software, obviando el comportamiento interno y la estructura del programa. Los errores encontrados están dirigidos a funciones incorrectas o ausentes, errores en la interfaz, errores en estructuras de datos o en accesos a bases de datos externas, errores de rendimiento y errores de inicialización y de terminación.

Dentro del método de caja negra, fue aplicada la técnica de partición de equivalencia, la cual divide el dominio de entrada de un programa en clases de datos, a partir de las cuales se derivan los casos de prueba.

Seguidamente se muestran tanto las variables utilizadas en el CU al que se le realizará la prueba así como el caso de prueba correspondiente al mismo.

Tabla 6: Variables utilizadas en el CU Crear XML.

| No | Nombre del campo | Clasificación | Valor nulo | Descripción |
|---|---|---|---|---|
| **1** | Nombre del ejecutable | TextField | No | Permite especificar el nombre del ejecutable. |
| **2** | Sistema operativo del ejecutable | RadioButton | Si | Permite especificar el o los Sistemas Operativos de cada ejecutable. |
| **3** | Descargable | CheckBox | Si | Permite especificar si el ejecutables es descargable o no. |
| **4** | Memoria RAM | TextField | Si | Permite especificar el o los requerimientos de RAM de cada problema. |
| **5** | Argumentos | TextPane | Si | Permite especificar los argumentos |

| | | | | |
|---|---|---|---|---|
| | | | | de cada problema. |
| **6** | Ficheros de entrada | List | No | Permite especificar los ficheros de entrada. |
| **7** | Existencia de ficheros de ejecución | CheckBox | Si | Permite especificar si existen ficheros de ejecución. |
| **8** | Ficheros de ejecución | List | Si | Permite especificar los ficheros de ejecución. |
| **9** | Existencia de nombre común | CheckBox | Si | Permite especificar si existe nombre común. |
| **10** | Nombre común | TextField | Si | Permite especificar el nombre común. |
| **11** | Existencia de ficheros de salida | CheckBox | Si | Permite especificar si se crearán ficheros de salida. |
| **12** | Ficheros de salida | TextPane | Si | Permite especificar los ficheros de salida que serán creados. |
| **13** | Existencia de ficheros a recuperar | CheckBox | Si | Permite especificar sise recuperarán ficheros de |

| | | | | |
|---|---|---|---|---|
| | | | | salida. |
| **14** | Ficheros a recuperar | TextPane | Si | Permite especificar los ficheros de salida a recuperar. |

Tabla 7: Caso de prueba Crear XML

| Escenario | Descripción | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | V10 | V11 | V12 | V13 | V14 | Respuesta del Sistema | Flujo Central |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| EC: Crear XML | Adicionar todos los parámetros de un XML | V | V | V | V | V | V | V | V | V | V | V | V | V | V | El sistema crea un XML. | Especificar el valor de todas las variables. Hacer clic en Aceptar. |
| | | V | I | I | I | V | V | I | I | I | I | I | I | I | I | El sistema crea un XML. | Especificar el ejecutable, los ficheros de entrada y los argumentos del problema. Hacer clic en Aceptar. |
| | | V | I | I | I | V | V | V | I | V | I | V | I | V | I | Muestra mensaje de error indicando campos vacíos. | Especificar los valores correspondientes a las variables de existencia seleccionadas. Hacer clic en Aceptar |

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | V | I | I | I | I | I | I | I | I | I | I | I | I | I | Muestra mensaje de error, indicando campos obligatorios vacíos. | Especificar Los valores correspondientes a los campos obligatorios. Hacer clic en Aceptar. |
| | | I | I | I | I | V | I | I | I | I | I | I | I | I | I | Muestra mensaje de error, indicando campos obligatorios vacíos | Especificar Los valores correspondientes a los campos obligatorios. Hacer clic en Aceptar. |
| | | I | I | I | I | I | V | I | I | I | I | I | I | I | I | Muestra mensaje de error, indicando campos obligatorios vacíos | Especificar Los valores correspondientes a los campos obligatorios. Hacer clic en Aceptar. |

### 4.3.3 Estudio de caso

Dentro de las pruebas realizadas a la aplicación se encuentra una dirigida a demostrar la disminución del tiempo de desarrollo que hoy conlleva la realización de aplicaciones independientes para generar tareas sobre T-arenal. Este tipo de prueba se realiza mediante el método empírico estudio de caso que tiene como objetivos: (27)

- Producir un razonamiento inductivo a partir del estudio, la observación y recolección de datos establece hipótesis o teorías.
- Puede producir nuevos conocimientos al lector, o confirmar teorías que ya se sabían.
- Hacer una crónica, un registro de lo que va sucediendo a lo largo del estudio.
- Describir situaciones o hechos concretos.
- Proporcionar ayuda, conocimiento o instrucción a caso estudiado.
- Comprobar o contrastar fenómenos, situaciones o hechos.
- Pretende elaborar hipótesis.

Es decir, el estudio de caso pretende explorar, describir, explicar, evaluar y/o transformar. El estudio de caso se llevó a cabo en el departamento de Bioinformática de la facultad 6, escenario donde se desarrolla una aplicación por cada uno de los problemas que se necesitan ejecutar en la Plataforma de Tareas Distribuidas.

Se presentarán dos casos correspondientes a dos de estas aplicaciones Vina y Dock, enfocadas en resolver problemas de Acoplamiento, abordados en el capítulo 1 del presente trabajo de diploma. Para la selección de estos casos de estudio se tuvo en cuenta que los mismos no fueran ficticios y que mostrasen la utilidad del modelo propuesto. Se realizaron entrevistas a los desarrolladores de estas dos aplicaciones, para establecer diferentes variables que determinarán en factor tiempo para ambos casos, tomando en cuanto el flujo de eventos antes y después del desarrollo de la solución propuesta. A continuación se muestran estos flujos.

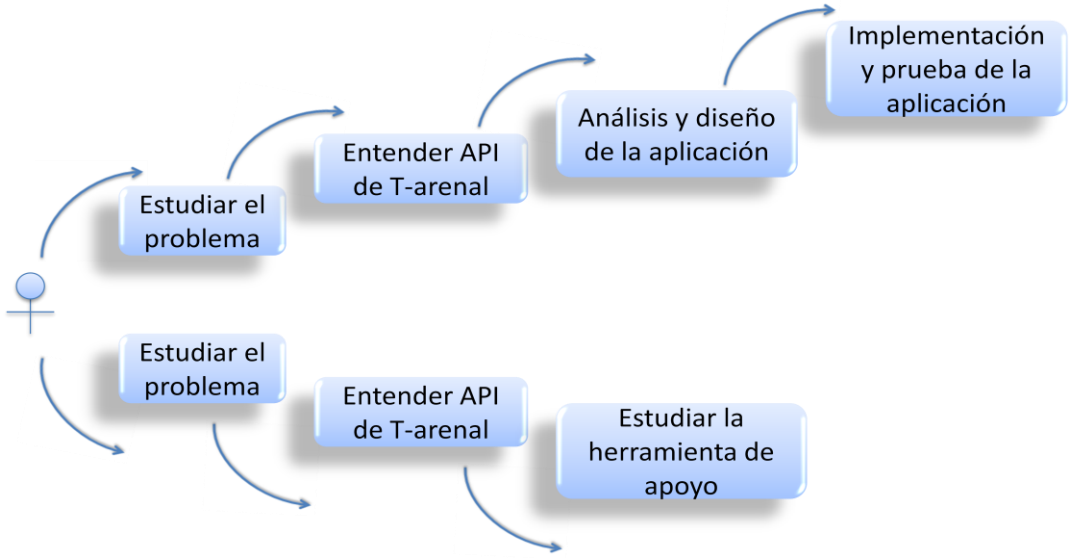


Figura 10: Flujos para resolver un problema con la utilización de T-arenal.

En la figura 10 se muestran los dos flujos existentes para resolver un problema con la utilización de la plataforma, el flujo 1 es el correspondiente al desarrollo de una aplicación para cada problema, y está reflejado en la parte superior de la figura; y el flujo 2 es el correspondiente a la utilización de la solución propuesta, y está reflejado en la parte inferior de la figura.

Del análisis de estos flujos se seleccionan como variables a tener en cuenta:

- **Estudiar el problema**: esta variable define el estudio correspondiente al problema a resolver, que debe hacerse en cualquier caso, para poder dar solución a un problema, puesto que se hace necesario comprender las características del mismo de igual manera para ambos flujos.
- **Entender API de T-arenal**: esta variable define el estudio que debe llevarse a cabo para comprender el funcionamiento de la plataforma, tiene una particularidad, y es que para el flujo 1 este estudio es más profundo pues debe entenderse la plataforma tanto en el rol de usuario como en el rol de desarrollador, mientras que para el flujo 2 solamente debe estudiarse el funcionamiento de la misma como usuario.
- **Análisis y diseño de la aplicación**: esta variable define, luego de conocer las características del problema y de la plataforma, la propuesta de la aplicación que permitirá crear el problema en T-arenal; esta variable solamente se encuentra en el flujo 1.

- **Implementación y prueba de la aplicación**: esta variable define el proceso de implementación y prueba para obtener finalmente la aplicación correspondiente a cada problema, esta variable solamente se encuentra en el flujo 1.
- **Estudiar la herramienta de apoyo**: esta variable define el estudio a llevar a cabo para comprender la herramienta que será capaz de generar el XML de cualquier problema que cumpla con los parámetros necesarios, esta variable solamente se encuentra en el flujo 2.

Luego de determinar las variables a medir, se define que el estudio de caso y los resultados del mismo estarán guiados a calcular el tiempo de desarrollo de la aplicación para generar una tarea. Es importante mencionar que los estudiantes entrevistados trabajan cinco días a la semana, dedicando en total 40 horas semanales.

**Estudio de caso para Vina**.

Tabla 8: Total de horas por variables dedicadas al desarrollo de Vina.

| Identificador de la variable. | Variables. | Total de horas. |
|---|---|---|
| **V1** | Estudiar el roblema. | **40h** |
| **V2** | Entender API de T-arenal | **60h** |
| **V3** | Análisis y diseño de la aplicación | **50h** |
| **V4** | Implementación y prueba de la aplicación | **120h** |
| **V5** | Estudiar la herramienta de apoyo | **-** |

Resultados arrojados del estudios de caso referente a la aplicación Vina.

- Total de horas dedicadas: **270**
- Total de sesiones de trabajo dedicadas: **67.5**

- Promedio de horas dedicadas: **67.5**

**Estudio de caso para el Dock**.

Tabla 9: Total de horas por variables dedicadas al desarrollo del Dock.

| Identificador de la variable. | Variables. | Total de horas. |
|---|---|---|
| **V1** | Estudiar el problema | **40h** |
| **V2** | Entender API de T-arenal | **80h** |
| **V3** | Análisis y diseño de la aplicación | **40h** |
| **V4** | Implementación y prueba de la aplicación | **120h** |
| **V5** | Estudiar la herramienta de apoyo | **-** |

Resultados arrojados del estudios de caso referente a la aplicación Dock.

- Total de horas dedicadas: **280**
- Total de sesiones de trabajo dedicadas: **70**
- Promedio de horas dedicadas: **70**

**Estudio de caso para la utilización de la solución propuesta**.

En el caso de la siguiente tabla las horas se muestran de la siguiente manera: horas dedicadas por el primer desarrollador entrevistado/ horas dedicadas por el segundo desarrollador entrevistado.

Tabla 10: Total de horas por variables dedicadas a la utilización de la solución propuesta.

| Identificador de la variable. | Variables. | Total de horas. D1/D2 |
|---|---|---|
| **V1** | Estudio de problema a resolver | **40h/40h** |

| | | |
|---|---|---|
| **V2** | Entender API de T-arenal | **20h/30h** |
| **V3** | Análisis y diseño de la aplicación | **-** |
| **V4** | Implementación y prueba de la aplicación | **-** |
| **V5** | Estudiar la herramienta de apoyo | **12h/8h** |

Resultados arrojados del estudios de caso referente a la aplicación genérica y la herramienta de apoyo.

- Total de horas dedicadas por el desarrollador 1: **72**
- Total de horas dedicadas por el desarrollador 2: **78**
- Total de sesiones de trabajo dedicadas por el desarrollador 1: **24**
- Total de sesiones de trabajo dedicadas por el desarrollador 2: **26**

A continuación se presentan diferentes gráficas que ayudarán a comprender los resultados arrojados.

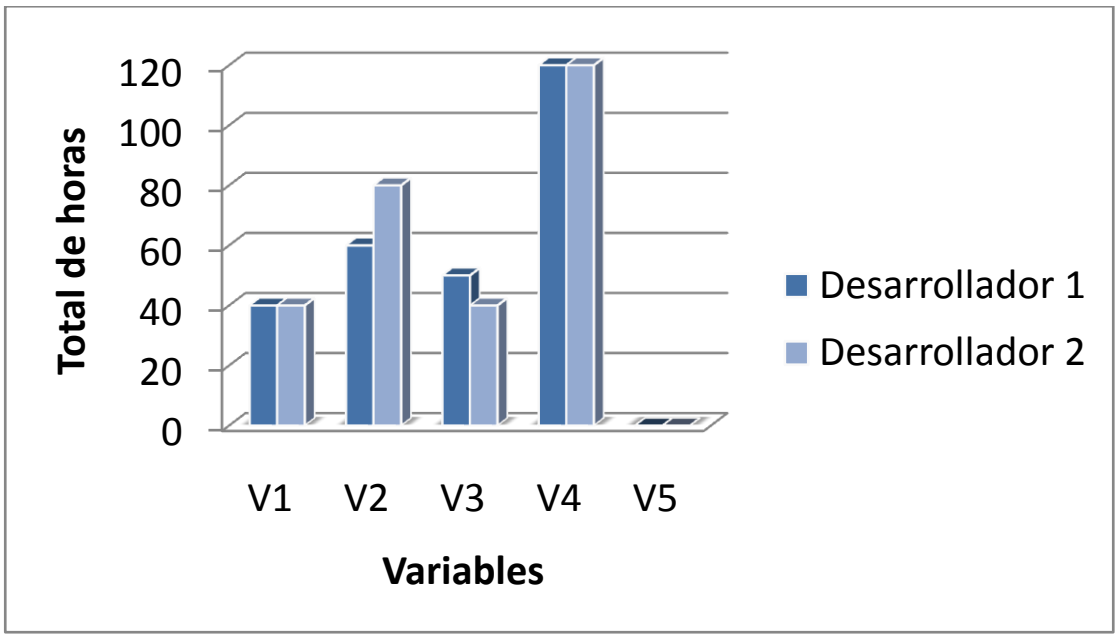


Figura 11: Total de horas dedicadas a cada una de las variables en el flujo 1 por cada desarrollador.

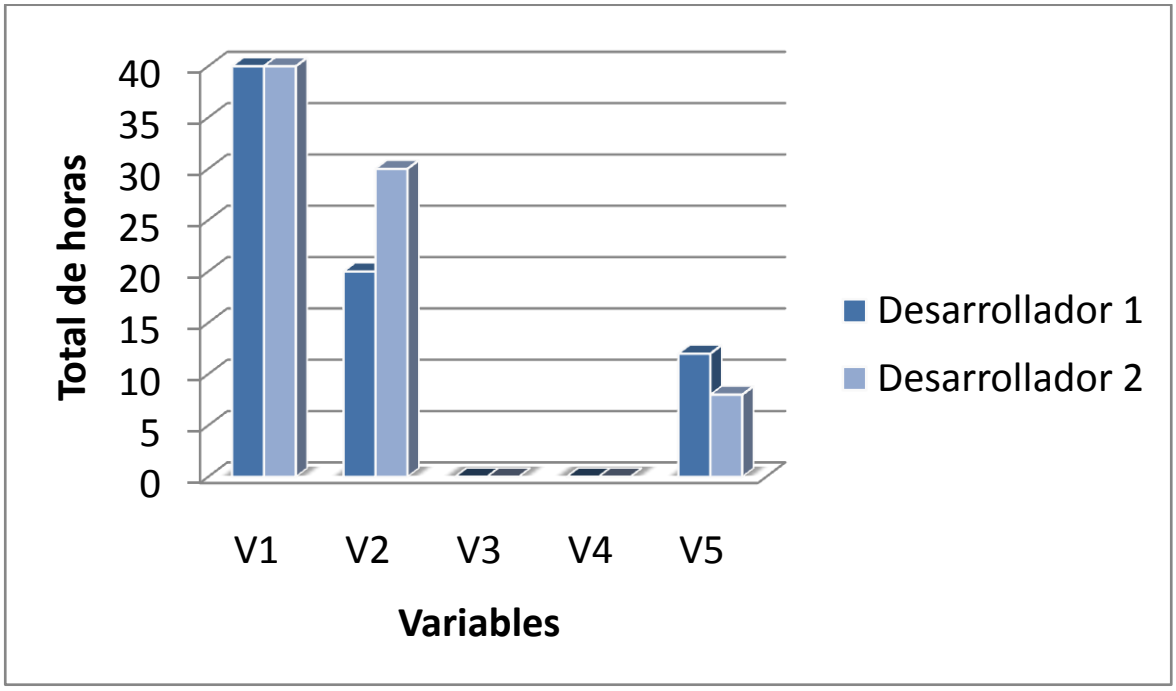


Figura 12: Total de horas dedicadas a cada una de las variables en el flujo 2 por cada desarrollador.

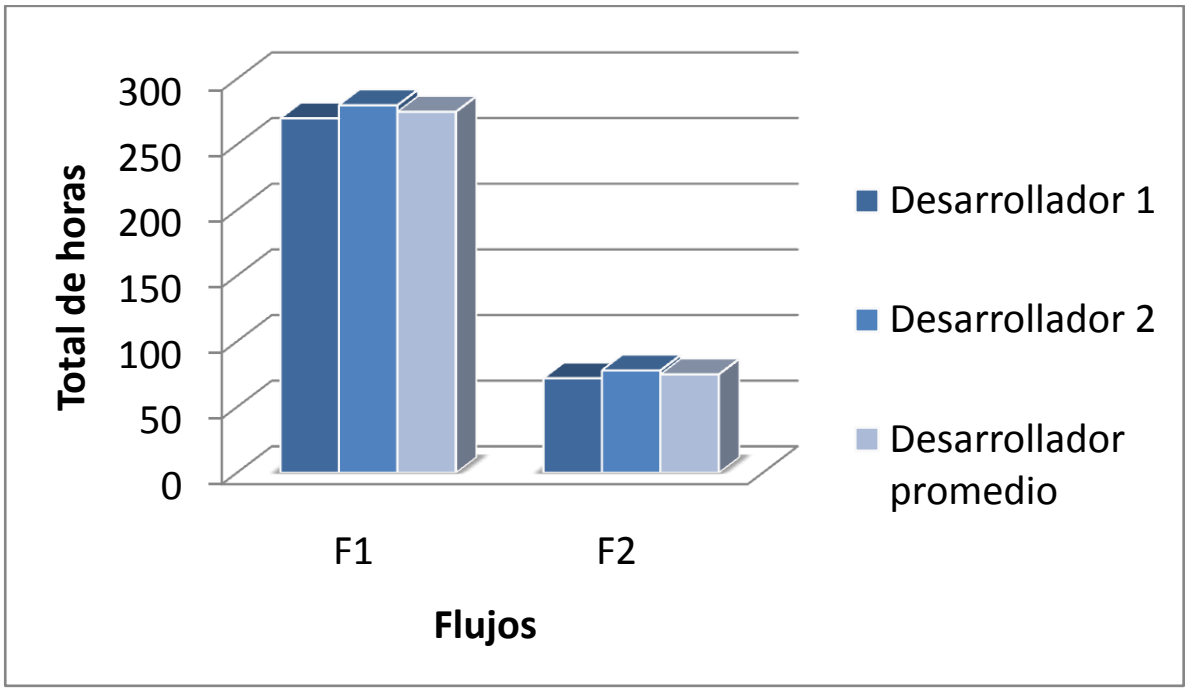


Figura 13: Total de horas dedicadas a cada uno de los flujos.

Luego del análisis de las gráficas mostradas se concluye que el desarrollo de una aplicación genérica y una herramienta de apoyo contribuyen a disminuir el tiempo de desarrollo de aplicaciones similares, así como el tiempo de llegar a generar un problema en la plataforma.

## 4.4 Conclusiones parciales

En el presente capítulo, se sientan las bases para dar paso a la implementación de la aplicación para ejecutar tareas sobre la plataforma, luego de haber definido el diagrama de componente que facilitó el entendimiento de las organizaciones y dependencias lógicas entre componentes de software, la comprensión del diseño existente para crear uno nuevo. También se obtuvo como resultado del capítulo la representación del despliegue que permitió determinar la ubicación de las soluciones propuestas. Los resultados arrojados por la prueba de caja blanda realizada al código fuente de la aplicación hicieron referencia a la correctitud de dicho código; así como lo hicieron, con respecto a la interfaz, los arrojados por la prueba de caja negra realizada a la herramienta de apoyo. Finalmente los resultados de una última prueba realizada, el estudio de caso, permitieron determinar que la solución propuesta cumple con el objetivo del presente trabajo de diploma.

# CONCLUSIONES GENERALES

**1.** El estudio y análisis de las soluciones existentes permitió determinar el lenguaje de descripción de trabajo para describir los problemas a ejecutar por lotes en la plataforma T-arenal.

**2.** El modelo de CU del sistema, permitió describir las funcionalidades que proveen tanto la aplicación como la herramienta de apoyo, en correspondencia, con las necesidades del usuario final.

**3.** El modelo de diseño, proporcionó una entrada para la implementación pues permitió describir cómo quedarían implementadas las clases y la comunicación existente entre ellas.

**4.** La aplicación genérica para la ejecución por lote de tareas sobre la plataforma de T-arenal, brinda la posibilidad de, a partir de la descripción de problemas mediante el lenguaje formal XML, ejecutar estos sin la necesidad de aplicaciones independientes para cada uno; disminuir el tiempo de desarrollo que anteriormente se empleaba en la construcción de aplicaciones independientes y facilitar la generación de tareas para la plataforma.

**5.** Se obtuvo una herramienta de apoyo que permite generar el XML para cualquier problema que pueda ser descrito con bajo la información que demanda este XML para ser creado.

**6.** La validación de la solución propuesta, a partir de las pruebas de caja blanca y caja negra, permitió detectar los posibles errores y obtener un producto con la calidad adecuada.

**7.** La realización de estudios de casos, permitió demostrar que se cumple el objetivo y por lo tanto se resuelve el problema planteado en el presente trabajo de diploma, referente a la necesidad de disminuir el tiempo de desarrollo de aplicaciones que generen tareas para T-arenal.

## RECOMENDACIONES

- Se recomienda adicionar varios mecanismos de lectura de formatos biológicos que el usuario final no tenga que dividir la data de entrada inicial.
- Se recomienda adicionar un mecanismo de sincronización de procesos que permita la ejecución por lotes de varios programas.

## REFERENCIAS BIBLIOGRÁFICAS

1. **G. S., Almasi and A., Gottlieb.** *Highly parallel computing.* Inc. Redwood City, CA, USA : Benjamin-Cummings Publishing Co., 1989. ISBN:0-8053-0177-1.

2. **Department, Bioinformatics and Genomics.** 7th Spanish Symposium on Bioinformatics and Computational Biology . España : s.n., 2006.

3. *Bioinformatics in the post-sequence era.* **Center, Bioinformatics and Laboratory, European Molecular Biology.** Japan; Germany : Minoru Kanehisa1 & Peer Bork2, 2003.

4. **García Jacas, César Raúl and Millares Taset, Daniel Marino.** "T-arenal v2.0: Desarrollo del back - end". La Habana : s.n., 209.

5. *A Multi-Server Approach for Distributing Tasks.* **Garcia Jacas, Cesar Raul, Resin Geyer, Claudio Fernando and Aguilera Mendoza, Longendri.** La Habana : s.n.

6. **Young, David C.** Computational Chemistry: A Practical Guide for Applying Techniques to Real-World Problems. 2001.

7. **García Jacas, César Raúl and Aguilera Mendoza, Longenri.** Manual del Usuario. *Front-end de la Plataforma de Tareas Distribuidas.* La Habana : s.n., 209.

8. **Toledo, Miguel L and Philips, George.** *Molecular Docking: A Problem With Thousands Of Degrees Of Freedom.* 2001.

9. **Vinuesa, Pablo.** *Introducción a la Inferencia Filogenética Molecular.* México : s.n., 2007.

10. HTC Condor. [Online] Update 2012. http://research.cs.wisc.edu/htcondor/.

11. gLite - Lightweight Middleware for Grid Computing. [Online] http://glite.cern.ch/.

12. Globus Toolkit. [Online] Update 2012. http://www.globus.org/.

13. **Roy, Alain.** ClassAds: Present and Furute. Wisconsin : s.n.

14. **Motoska, V., et al.** Condor as a resource for finalcial manket analysis. *FACADE – Financial Analysis Computing Architecture in Distributed Environment.* 2011.

15. *Programming the Grid with gLite.* 2006.

16. **Globus, Alliance, Globus and Toolkit, Globus.** *GT: Jod Description Schema Doc.* Chicago : s.n.

17. **Balduino, Ricardo.** Introduction to OpenUP (Open Unified Process). Brasil : s.n., 2007.

18. **Pinheiro da Silva, Paulo and W. Paton, Norman.** User Interface Modelling with UML. UK : s.n.

19. Visual Paradigm. [Online] http://www.visual-paradigm.com.

20. **Belmonte Fernández, Oscar.** Introducción al lenguaje de. 2004. Vol. III.

21. Altova. [Online] 2005. http://www.altova.com.

22. Adivo. [Online] 2002. http://www.adivo.com.

23. liquid Technologies. [Online] 2012. http://www.liquid-technologies.com.

24. NetBeans. [Online] 2012. http://netbeans.org.

25. **Martínez, Ivette Carolina.** *Modelo Conceptual/ Modelo de Dominio.* Venezuela : s.n.

26. **Reynoso, Carlos and Kicillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires : s.n., 2004.

27. **Yacuzzi, Enrique.** *El estudio de caso como metodología de investigación.*

**BIBLIOGRAFÍA**

1. **G. S., Almasi and A., Gottlieb.** *Highly parallel computing.* Inc. Redwood City, CA, USA : Benjamin-Cummings Publishing Co., 1989. ISBN:0-8053-0177-1.

2. **Department, Bioinformatics and Genomics.** 7th Spanish Symposium on Bioinformatics and Computational Biology . España : s.n., 2006.

3. *Bioinformatics in the post-sequence era.* **Center, Bioinformatics and Laboratory, European Molecular Biology.** Japan; Germany : Minoru Kanehisa1 & Peer Bork2, 2003.

4. **García Jacas, César Raúl and Millares Taset, Daniel Marino.** "T-arenal v2.0: Desarrollo del back - end". La Habana : s.n., 209.

5. *A Multi-Server Approach for Distributing Tasks.* **Garcia Jacas, Cesar Raul, Resin Geyer, Claudio Fernando and Aguilera Mendoza, Longendri.** La Habana : s.n.

6. **Young, David C.** Computational Chemistry: A Practical Guide for Applying Techniques to Real-World Problems. 2001.

7. **García Jacas, César Raúl and Aguilera Mendoza, Longenri.** Manual del Usuario. *Front-end de la Plataforma de Tareas Distribuidas.* La Habana : s.n., 209.

8. **Toledo, Miguel L and Philips, George.** *Molecular Docking: A Problem With Thousands Of Degrees Of Freedom.* 2001.

9. **Vinuesa, Pablo.** *Introducción a la Inferencia Filogenética Molecular.* México : s.n., 2007.

10. HTC Condor. [Online] Update 2012. http://research.cs.wisc.edu/htcondor/.

11. gLite - Lightweight Middleware for Grid Computing. [Online] http://glite.cern.ch/.

12. Globus Toolkit. [Online] Update 2012. http://www.globus.org/.

13. **Roy, Alain.** ClassAds: Present and Furute. Wisconsin : s.n.

14. **Motoska, V., et al.** Condor as a resource for finalcial manket analysis. *FACADE – Financial Analysis Computing Architecture in Distributed Environment.* 2011.

15. *Programming the Grid with gLite.* 2006.

16. **Globus, Alliance, Globus and Toolkit, Globus.** *GT: Jod Description Schema Doc.* Chicago : s.n.

17. **Balduino, Ricardo.** Introduction to OpenUP (Open Unified Process). Brasil : s.n., 2007.

18. **Pinheiro da Silva, Paulo and W. Paton, Norman.** User Interface Modelling with UML. UK : s.n.

19. Visual Paradigm. [Online] http://www.visual-paradigm.com.

20. **Belmonte Fernández, Oscar.** Introducción al lenguaje de. 2004. Vol. III.

21. Altova. [Online] 2005. http://www.altova.com.

22. Adivo. [Online] 2002. http://www.adivo.com.

23. liquid Technologies. [Online] 2012. http://www.liquid-technologies.com.

24. NetBeans. [Online] 2012. http://netbeans.org.

25. **Martínez, Ivette Carolina.** *Modelo Conceptual/ Modelo de Dominio.* Venezuela : s.n.

26. **Reynoso, Carlos and Kicillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires : s.n., 2004.

27. **Yacuzzi, Enrique.** *El estudio de caso como metodología de investigación.*

28. **Grau, Ricardo, Correa, Cecilia and Rojas, Mauricio.** *Metodología de la investigación.* Colombia : El POIRA Editores S. A., 2004. ISBN: 958-8028-10-8.

29. Estrategias de prueba del software. [book auth.] Pressman.

30. **Sommerville, Ian.** *Ingeniería de software. Séptima edición.* Madrid : Pearson Educación S.A, 2005. ISBN: 84-7829-074-5.

31. **Compensation, Office of the Voice Chancellor for Human Resources.** UIC Human Resources. *Writing Effective Job Description.* 2009.

32. **DataGrid.** *Job Description Language How To.* 2001.

33. **it-Mentor.** *Capacidad y guía para el desarrollo de software.*

# ANEXOS

Tabla 11: Descripción del CU Almacenar descripción del problema.

| | |
|---|---|
| **Caso de Uso:** | Almacenar descripción del problema |
| **Actores:** | Especialista |
| **Propósito** | Este caso de uso se lleva a cabo con el objetivo de validar y almacenar la descripción de un problema. |
| **Resumen:** | El caso de uso se inicia cuando el especialista desea ejecutar un problema sobre la plataforma, para lo cual especifica los ficheros de dicho problema entre los cuales se encuentra la descripción del mismo, termina cuando se encuentra la descripción correcta y se almacenan los datos de esta. |
| **Precondiciones:** | El usuario tiene que estar autenticado. Tiene que existir la descripción del problema. |
| **Referencias** | RF1, RF 2 |
| **Prioridad** | Crítico |
| **Flujo Normal de Eventos** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| **1.** El caso de uso se inicia cuando el especialista especifica los ficheros que necesita el problema para ejecutarse, incluyendo la descripción de este. | **2.** El sistema busca todos los ficheros con extensión xml, que se encuentren en el directorio.<br>**3.** El sistema valida la estructura de cada fichero (XML) mediante un esquema XSD, hasta que encuentra el correcto.<br>**4.** El sistema realiza la interpretación del XML, obteniendo los valores de cada una de las etiquetas que lo componen y que describen el problema a resolver. Finaliza el caso de uso. |
| **Flujo Alterno** | |
| | **3.1** Si no encuentra ningún XML que cumpla con el esquema manda un mensaje de error: *No existe un XML* |

| | |
|---|---|
| | *que cumpla con las características*. Finaliza el caso de uso. |

Tabla 12: Descripción del CU Realizar procesamiento.

| **Caso de Uso:** | Realizar procesamiento. |
|---|---|
| **Actores:** | Cliente |
| **Propósito** | Realizar la tarea que recibe desde el servidor de peticiones correspondiente. |
| **Resumen:** | El caso de uso inicia cuando un cliente hace una solicitud de unidad de trabajo al servidor de peticiones correspondiente para luego realizar su procesamiento. Termina cuando el cliente ha terminado de procesar la unidad de trabajo. |
| **Precondiciones:** | Tener una tarea asignada por el servidor de peticiones correspondiente. |
| **Referencias** | RF5 |
| **Prioridad** | Crítico |
| **Flujo Normal de Eventos** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| **1.** El cliente realiza una petición al servidor de peticiones correspondiente. | **2.** El servidor de peticiones retorna la unidad de trabajo creada para el cliente (ver caso de uso Atender tarea asignada. Sección Crear una unidad de trabajo). |
| **3.** El cliente obtiene desde el servidor de peticiones correspondiente, la tarea a realizar. | **3.1** El servidor de peticiones retorna la tarea a procesar y los ficheros que serán utilizados para la realización de la misma. |
| **4.** El cliente carga la tarea y los ficheros retornados desde el servidor de peticiones. | |
| **5.** El cliente crea la línea de comando necesaria para la ejecución del problema. | |
| **6.** El cliente da permiso de ejecución a los ficheros necesario para el procesamiento del problema. | |
| **7.** El cliente realiza la tarea especificada en la | |

| | |
|---|---|
| unidad de trabajo. | |
| **8.** El cliente recoge los ficheros de salida especificados en la unidad de trabajo creada, si no se especifica ninguno en particular, el cliente recoge todos los ficheros de salida generados. | |
| **9.** Si culminó el procesamiento, el cliente retorna el resultado obtenido hacia el sistema. | **9.1** El servidor de peticiones procesa el resultado enviado por el cliente (ver caso de uso Atender tarea asignada). |
| **Flujo Alterno** | |
| **9.1** Si el cliente no ha culminado la tarea, retorna a la actividad número 9 del flujo normal de eventos. | |

Tabla 13: Descripción del CU Crear XML.

| | |
|---|---|
| **Caso de Uso:** | Crear XML |
| **Actores:** | Especialista |
| **Propósito** | Crear el XML que describa el problema que necesita ser procesado. |
| **Resumen:** | El caso de uso inicia cuando el especialista desea generar la descripción correcta de un problema. Termina cuando el usuario ha introducido todos los datos correctamente y es creado el XML. |
| **Precondiciones:** | Que existan los ficheros obligatorios que necesita el XML. |
| **Referencias** | RF8, RF9, RF10, RF11, RF12, RF13, RF14 y RF15 |
| **Prioridad** | Secundario |
| **Flujo Normal de Eventos** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| **1.** El caso de uso se inicia cuando el especialista ejecuta el generador con el fin de crear un XML. | **2.** El sistema muestra una ventana con los campos que debe llenar el usuario correspondientes a:<br>• El ejecutable, este campo es de carácter obligatorio, y puede añadirse una o varias veces. |

| | |
|---|---|
| | • El sistema operativo, este campo es opcional y se refiere al sistema operativo correspondiente al ejecutable.<br>• Si el ejecutable es descargable o no.<br>• Los argumentos, este campo es opcional, pueden escribiste varias líneas, y hace referencia a la línea de comando que ejecutará el problema.<br>• Los ficheros de entrada, este campo es obligatorio, hace referencia a los ficheros que necesita el programa para ejecutarse, puede ser uno o varios.<br>• Los ficheros de ejecución, este campo es opcional, hace referencia a ficheros adicionales a los de entrada que necesita el problema para ejecutarse, puede ser uno o varios.<br>• Ficheros de salida a crear, este campo es opcional, hace referencia a los ficheros de salida que el programa necesita que sean creados o que el especialista lo desea.<br>• Ficheros de salida a recuperar, este campo es opcional, hace referencia a los ficheros de salida que el especialista desea recoger, de no especificar ninguno se recogen todos.<br>• Memoria RAM, este campo es opcional, hace referencia a la(s) memoria(s) RAM que demanda el problema para su ejecución. |
| **3.** El especialista llena los campos correspondiente y selecciona la opción Crear. | **4.** El sistema valida que los campos obligatorios estén no estén vacíos.<br>**5.** El sistema almacena todos los datos leídos, y pasa a crear el XML en el directorio especificado, o en el directorio |

| | |
|---|---|
| | por defecto. Finaliza el caso de uso. |
| **Flujo Alterno** | |
| | **4.1** El sistema muestra un mensaje de error indicando cuál campo vacío obligatorio encontró, y retorna a la actividad 3 del flujo normal de los eventos. |