UNIVERSIDAD AUTÓNOMA METROPOLITANA

Cloud Computing para el ambiente de programación DLML

Idónea comunicación de resultados
para obtener el grado de
Maestro en Ciencias
(Ciencias y Tecnologías de la Información)
por:
Abraham Martínez Ramírez

Asesores:
Dr. Miguel Alfonso Castro García
Dr. Manuel Aguilar Cornejo

Jurado Calificador:

Presidente:	**Dr. José Guadalupe Rodríguez García**	**CINVESTAV**
Secretario:	**Dr. Miguel Alfonso Castro García**	**UAM-I**
Vocal:	**Ing. Luis Fernando Castro Careaga**	**UAM-I**

UNIVERSIDAD AUTÓNOMA METROPOLITANA - IZTAPALAPA
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

México D.F. 12 de julio de 2013

Resumen

Hoy en día, en Internet se ofrecen todo tipo de servicios que van desde correo electrónico, buscadores, hasta servicios de hardware; a toda esta propuesta de la computación se le conoce como Cloud Computing (La Nube). La idea principal del Cloud Computing es que los usuarios puedan ejecutar servicios de una manera transparente, sin que se preocupen que hay detrás de la nube, esta propuesta ha tomado un gran auge gracias al avance tecnológico de dispositivos móviles, virtualización de recursos, entre otros, pero la gran aceptación se debe a la facilidad que brinda a los usuarios para hacer uso de una gran cantidad de servicios. Por otra parte, otra propuesta de la computación importante es el HPC (High Performance Computing), el cual se conforma de arquitecturas de hardware como los clusters, grids, etc., que en complemento con herramientas para la programación paralela el HPC brinda un cómputo de alto desempeño. El HPC es usado hoy en día, para resolver problemas que requieren grandes cargas de procesamiento, que pueden tardar desde horas hasta días en una computadora normal con una programación secuencial. Gran parte de los usuarios de HPC son investigadores como matemáticos, físicos, químicos, etc., muchos de los modelos de estos investigadores requieren de grandes cantidades de procesamiento. No obstante el punto crítico del HPC, es la complejidad que conlleva realizar una aplicación de este tipo, inclusive para usuarios expertos en programación.

Relacionado a lo anterior, en la nube ha surgido un modelo de servicio denominado HPCaaS (HPC as a Service), en el cual se ofrece HPC como servicio sobre la nube, algunos proveedores son Amazon, Azure, Cyclone, etc., estos ofrecen servicios de infraestructura para que los usuarios desarrollen sus aplicaciones HPC. No obstante los proveedores de HPCaaS no ofrecen como tal, un servicio de software que permita a los usuarios ejecutar una aplicación HPC a través de una interfaz web, por tal motivo la solución ofrecida en la nube, a la fecha, sólo se convierte en un camino alterno para usuarios que no cuentan con una infraestructura de HPC, en el cual la complejidad para los usuarios del HPC sigue siendo la misma o inclusive mayor.
En este trabajo se presenta el proyecto Cloud Computing para el ambiente de programación DLML (Cloud-DLML), para ofrecer servicios HPCaaS, así con esto beneficiar a muchos de los usuarios del HPC, facilitando por medio del Cloud Computing recursos y aplicaciones de HPC. Cloud-DLML permite pasar de un escenario complejo de obtener recursos de hardware y programar aplicaciones paralelas, a un escenario en el cual por medio de un dispositivo con conexión a Internet se puede mandar a ejecutar servicios de HPC.

keywords: Cloud Computing, Cloud-DLML, SOA, Portal-DLML, Web Service, HPC, HPCaaS, DLML y Cluster.

Agradecimientos

Quiero agradecer primeramente a mis padres quienes fueron los que me inculcaron los estudios, a mi madre Patricia Ramírez Jiménez por todo su apoyo incondicional y por todo lo que ha representado para mi hasta este momento de mi vida, de igual manera a mi padre Juan Carlos Martínez Nieto que aunque en estos momentos ya no esta en cuerpo presente, él fue quien me guio con sus consejos y me dio gran parte de mi educación, también quiero agradecer a mi hermana Leylani Martínez Ramírez que también me apoyo con este proyecto de vida. Por otra parte, no puedo dejar de mencionar a todos mis compañeros de la maestría, los cuales de cierta forma me han brindado su apoyo en este camino que hemos elegido, a mis amigos los cuales estuvieron a mi lado y que también me impulsaron.

Claro también agradezco a mis maestros que me han dado parte de su conocimiento, en especial a los profesores Miguel Alfonso Castro García que me ha asesorado durante la maestría y Luis Fernando Castro Careaga del cual aprendí mucho para fines profesionales, todos ellos grandes maestros y excelentes personas.
Por último agradezco a dos grandes instituciones que hacen posible que los estudiantes puedan lograr sus metas, es por eso que agradezco a la Universidad Autónoma Metropolitana en la cual he pasado 7 años de mi vida, lo cual ha sido una gran experiencia en lo personal y profesional, la otra institución a la que le agradezco es al CONACYT por su apoyo económico que me ha brindado en el transcurso de la maestría.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1
Introducción

Actualmente en Internet hay todo tipo de servicios, desde correo electrónico, mapas geográficos en línea, hasta servicios como el uso de hardware para el almacenamiento y procesamiento de información. A todos estos servicios se les ha denominado Cloud Computing conocido como el cómputo en nube o simplemente la nube, esta propuesta engloba desde el software hasta el hardware que es utilizado para proveer servicios en Internet. La idea principal es que el usuario, por medio de un navegador de Internet, ya sea desde su PC, laptop, tablet, celular, o cualquier otro dispositivo con conexión a Internet pueda hacer uso de los servicios. En general los servicios ofrecidos en Internet se clasifican en tres tipos: SaaS (Software as a Service), PaaS (Platform as a Service) e IaaS (Infrastructure as a Service), estos tres tipos de servicios cubren con la demanda de la mayoría de los usuarios en Internet [1].

En contraste al Cloud Computing en cuanto la facilidad de uso por parte de los usuarios, esta el HPC (High Performance Computing), que es usado hoy en día, para aplicaciones o problemas complejos que necesitan de grandes cargas de procesamiento. El HPC se puede dividir en dos partes fundamentales que son el hardware y el software: los recursos de hardware usados para el HPC son los clusters, las grids y más actualmente las GPU's, para la parte del software se cuenta con herramientas como MPI (Message Passing Interface), MapReduce, CUDA, DLML (Data List Management Library), que son usadas para explotar los recursos de hardware [2].

Algunos ejemplos de problemas que requieren de HPC son el estudio del genoma, la predicción del clima, búsqueda de vida extraterrestre, etc. Como se puede ver la gran mayoría de estos problemas radican en problemas de investigación donde colaboran varios científicos o instituciones, en general estos problemas surgen de modelos científicos que pueden ser modelos matemáticos, físicos, químicos, por mencionar algunos.

En relación a estas dos propuestas Cloud Computing y HPC, ha surgido un nuevo servicio en la nube llamado HPCaaS (HPC as a Service), en el cual se busca que desde una interfaz web se pueda operar o utilizar los recursos del HPC, este modelo de servicio es relativamente nuevo en el ámbito del Cloud Computing [3].

Como es de esperarse en la nube hay muy pocos servicios de HPC, en especial para modelos científicos que requieren del HPC. Algunos proveedores como Amazon, Cyclone, Azure entre otros ofrecen servicios de este tipo, pero enfocados en general a los recursos de hard-

ware, que en la mayoría de los casos se hace por medio de clusters virtualizados dejando a un lado las aplicaciones HPC. Además de que la mayoría de proveedores cobran por estos servicios, salvo pocas excepciones como OpenCirrus que brinda a sus usuarios servicios de hardware por medio de clusters de manera gratuita. Como se puede ver la mayoría de los HPCaaS están a nivel de IaaS, ya que solo se ofrece la infraestructura de HPC para que los usuarios desarrollen sus aplicaciones y puedan ejecutarlas sobre estas.

Cabe mencionar que hay pocos casos de HPCaaS a nivel de SaaS, para estos casos el ejemplo más representativo es HPC Advisor Council, que ha llevado aplicaciones HPC sobre la nube, donde además de ofrecer el hardware, también ofrecen el software para que el usuario lo use a través de una interfaz web, facilitando la ejecución de dos aplicaciones paralelas CPMD (Car-Parrinello Molecular Dynamics) y NAMD (Nanoscale Molecular Dynamics), para un grupo de científicos, así con estos servicios los usuarios no se preocupan de la infraestructura o la programación paralela [4].

En resumen, aunque hay grandes proveedores sobre la nube que ofrecen HPCaaS a nivel de IaaS, todavía son muy complejos para los usuarios, ya que el virtualizar un cluster en alguna nube de los grandes proveedores no es nada fácil, más aún el tener que configurar, preparar e implementar las aplicaciones HPC sigue siendo muy complejo para los usuarios; por tal motivo las aplicaciones paralelas expuestas de CPMD y NAMD como servicios, son HPCaaS más completos a un nivel de SaaS, aunque estos son exclusivos para algunos usuarios además de que han sido implementados sobre una arquitectura de software centralizada, sujeta a la infraestructura de HPC, en la cual es difícil escalar a nivel de clusters y agregar otras aplicaciones paralelas como servicios.

En esta tesis se presenta Cloud-DLML que es una plataforma para ofrecer HPCaaS, los cuales sean capaces de proveer ya las aplicaciones HPC como servicios sobre la nube, así como lo ha hecho HPC Advisor Council, pero con una gran diferencia que es la arquitectura, ya que Cloud-DLML ha sido implementado con una arquitectura basada en SOA (Service Oriented Architecture), la cual anexa una capa desacoplada de HPC basada en clusters; permitiendo con esto una arquitectura altamente escalable y flexible, para ofrecer aplicaciones HPC como HPCaaS. En un principio Cloud-DLML ofrece los servicios de N-Reinas, Multiplicación y Suma de Matrices, los cuales explotan la infraestructura HPC compuesta por clusters y la librería DLML para aplicaciones paralelas.

En Cloud-DLML los usuarios no tienen que preocuparse de la infraestructura o la programación, en este escenario los usuarios solo por medio de un dispositivo con una conexión a Internet pueden mandar a ejecutar las aplicaciones HPC, haciendo uso de clusters que en teoría pueden estar en cualquier lugar del mundo.
El resto de la tesis se organiza de la siguiente forma: en el Capítulo 2 se presenta con mayor detalle el contexto del HPC tanto el hardware como el software, incluyendo la herramienta DLML que fue usada para la implementación de las aplicaciones paralelas de Cloud-DLML. Posteriormente en el Capítulo 3 se presenta el contexto del Cloud Computing, así como todo lo que demanda su implementación. Una vez revisados las dos propuestas tanto el Cloud

Computing como el HPC, que son el contexto de esta tesis en el Capítulo 4 nos adentramos al diseño, la arquitectura y los protocolos de comunicación que ha demandado la implementación de Cloud-DLML. En el Capítulo 5 se muestran los resultados obtenidos de Cloud-DLML por medio de una evaluación de atributos de calidad y la manera en que fue puesto en operación Cloud-DLML por medio de dos diferentes despliegues (Deployments). Por último en el Capítulo 6 se muestran las conclusiones y trabajo a futuro de este proyecto.

Capítulo 2
Cómputo de alto desempeño

El Cómputo de alto desempeño o bien High Performance Computing (HPC) como se le es conocido, es sinónimo de supercómputo, el HPC se divide en dos partes principales, el hardware y el software [2]. Por la parte del hardware se cuenta con diferentes arquitecturas como lo son: los clusters, grids, supercomputadoras y más recientemente las GPU's. Todas las arquitecturas de HPC tienen en común el poder de procesamiento, pero cada una de ellas tiene sus ventajas y desventajas: los clusters y las grids proporcionan una alta escalabilidad pero requieren de complejas configuraciones; mientras que las supercomputadora son tal vez las de menor complejidad pero son las de mayor costo monetario y menos escalables; las GPU's son arquitecturas con gran poder de procesamiento de bajo costo pero estas carecen de memoria en comparación con las otras arquitecturas.

Por la parte del software, el HPC cuenta con herramientas que explotan las arquitecturas antes mencionadas: MPI (Message Passing Interface), MapReduce, DLML(Data List Management Library) y CUDA. MPI es un estándar para el envío de mensajes entre los nodos de un cluster, MapReduce es un modelo de programación basado en key's/Value's, y en las funciones Map y Reduce que se ejecutan sobre los nodos de un cluster [5]. DLML (Data List Management Library), es una librería basada en MPI que sigue un modelo de programación basado en listas para arquitecturas de clusters y grids, y por último CUDA que es un lenguaje de programación para la arquitectura de las GPU's [6].

Como se puede ver el HPC hace uso de potentes arquitecturas de hardware y herramientas de software, que complementadas permiten resolver problemas complejos que requieren de grandes recursos de cómputo, en especial grandes cargas de procesamiento. De esta manera dar solución a problemas que necesitan gran capacidad de procesamiento se hace posible. A continuación damos un mayor detalle de las arquitecturas y de las herramientas de programación, que son usadas para el desarrollo de HPC.

2.1. Arquitecturas

En la parte del hardware tenemos diferentes arquitecturas de HPC, entre las cuales destacamos a los clusters y las grids, estas dos arquitecturas de hardware han sido adoptadas para dar solución a los problemas que requieren grandes cantidades de procesamiento, no

obstante para los usuarios que las requieren no es nada fácil obtener estos recursos, además de la configuración y administración que estas implican.

2.1.1. Cluster

Un cluster en términos informáticos, es un conjunto de computadoras conectadas entre sí, que trabajan como si fueran una sola, convirtiéndose así en una supercomputadora (Figura 2.1). Este tipo de tecnología ha surgido por la necesidad de resolver aplicaciones que requieren grandes cargas de procesamiento, que una computadora ordinaria no podría resolver por sus capacidades, o que tal vez tardaría una gran cantidad de tiempo en arrojar los resultados (desde horas hasta años). También hay que decir que gran parte de la aceptación de esta tecnología, se debe a que los clusters son mucho menos costosos que una supercomputadora que proporcione las mismas capacidades de cómputo [7]. En la actualidad los cluster son usados para servidores web, servidores de base de datos de alto rendimiento, aplicaciones de supercómputo, cómputo científico etc., a continuación mencionamos cuatro características de los clusters:

Figura 2.1: Cluster

- Alto rendimiento: los cluster están pensados para resolver problemas que requieren grandes cargas de procesamiento, debido a que todas las computadoras del cluster pueden trabajar en conjunto para resolver un mismo problema.
- Alta disponibilidad: en un cluster, al contar con varios nodos puede haber una alta disponibilidad, en este caso lo que se hace es redundancia de éstos, con esto si falla uno, entonces otro nodo puede ejecutar las tareas del que fallo, con esto se garantiza una alta disponibilidad.
- Eficiencia: las tareas que se ejecutan en diferentes nodos deben de ser lo más independientes entre sí, con esto se asegura una mayor eficiencia del cluster, ya que se eliminan

los tiempos en los que los nodos están sin trabajar por causa de estar esperando que otros nodos terminen su tareas.

- Escalabilidad: un cluster puede escalar, ya que se le pueden ir agregando nodos y así obtener mayor poder de cómputo.

Una organización que se centra en evaluar a las grandes supercomputadoras (clusters) de acuerdo a sus capacidades de procesamiento, almacenamiento y comunicación, es el Top500 [8]. Esta organización cada semestre publica una lista de las 500 supercomputadoras con mayor capacidad de cómputo en el mundo, hasta la fecha de noviembre del 2012 los 5 primeros supercomputadoras (Tabla 2.1) era encabezada por la computadora Titan-Cray Xk7, para darse una idea de las capacidades de estas supercomputadoras, esta tiene la capacidad de procesar hasta 17590.0 petaflop/s usando 560,640 cores.

Rank	Lugar	Cluster
1	DOE/SC/Oak Ridge National Laboratory, Estados Unidos	Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x
2	DOE/NNSA/LLNL, Estados Unidos	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom
3	RIKEN Advanced Institute for Computational Science (AICS), Japon	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect
4	DOE/SC/Argonne National Laboratory, Estados Unidos	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom
5	Forschungszentrum Juelich (FZJ), Alemania	JUQUEEN - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect

Tabla 2.1: Las primeras 5 supercomputadoras del top500 2012

Como vemos un cluster genera un gran valor a los usuarios que hacen uso de éste, pero también hay que mencionar que el obtener un cluster no es nada fácil, ya que no sólo se trata del hardware sino que también requiere una configuración compleja, inclusive para usuarios expertos les puede tomar mucho tiempo para construir un cluster. A la fecha, los usuarios como físicos, matemáticos, químicos, investigadores en general tienen que esperar por los recursos en el mejor de los casos, para que sus modelos puedan ser procesados en un cluster. Por otro lado existen arquitecturas aún más grandes como lo son las grids, las cuales pueden estar formadas por clusters, como es de esperarse estas arquitecturas son más complejas.

2.1.2. Grid

La tecnología de grids puede ser vista como el siguiente paso después de los clusters [9], éstas consisten en interconectar diferentes recursos heterogéneos por medio de un red extensa como el Internet, estos recursos pueden ir desde simples computadoras hasta grandes clusters. Las grids no están sujetas a un control centralizado, a diferencia de los clusters, los recursos de la grid pueden estar en diferentes lugares geográficos con administraciones independientes (Figura 2.2). Hoy en día existen herramientas como Globus y Condor [10], por mencionar algunas, éstos permiten la administración y control sobre los recursos que componen la grid. Lo que se busca de una grid, es hacer uso de estos recursos de manera transparente para resolver problemas que requieren grandes cantidades de procesamiento.

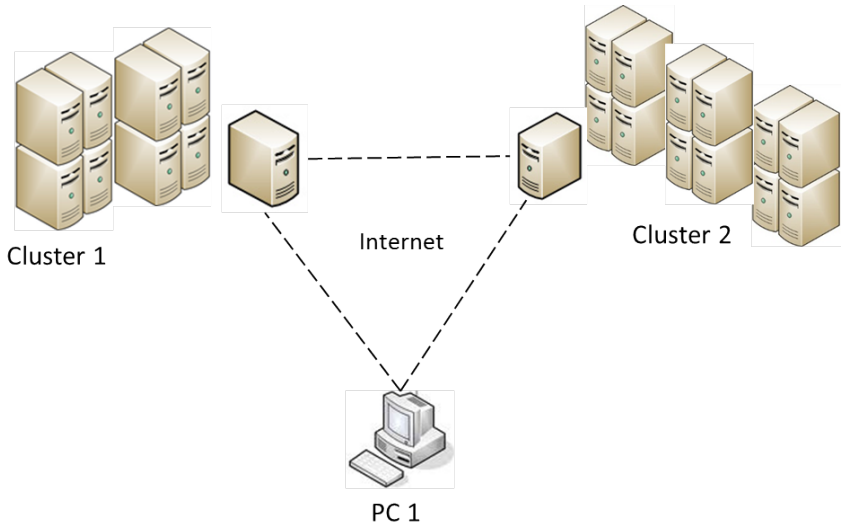

Figura 2.2: Grid formado por dos clusters y una computadora

Los aspectos importantes de una grid son los siguientes:

- Heterogeneidad: una grid puede tener diferentes tipos de recursos con capacidades diferentes, estos recursos van desde una computadora hasta un cluster, por lo cual hay una heterogeneidad de hardware e incluso de software.

- Gestión: la grid debe tener un medio para coordinar todos estos recursos y así hacer uso de estos para distintos fines específicos, este aspecto es muy importante para que haya un control.

- Descentralización: este aspecto se debe a que en la grid se hace uso de múltiples recursos distribuidos geográficamente, donde cada uno de estos recursos puede tener su propia administración y políticas de seguridad.

- Disponibilidad: debido a que la grid hace uso de múltiples recursos en consecuencia hay una alta disponibilidad, aunque esta pueda variar en rendimiento según las capacidades de los recursos disponibles.
- Escalabilidad: la grid al igual que los clusters, se puede escalar en recursos de hardware.
- Comunicación y colaboración: Este aspecto es muy importante, ya que debe haber una estrategia para hacer posible la comunicación y colaboración entre los diferentes recursos de la grid, esto se logra normalmente implementando protocolos y estándares de comunicación.
- Balance de carga: en una grid la carga de trabajo se hace de acuerdo a las capacidades de cada recurso dentro de la grid, es decir una computadora debería tener menos trabajo que un cluster, así con esto se logra que haya un mayor aprovechamiento de los recursos de acuerdo a sus capacidades.

Por lo anterior podemos decir que una grid puede ser vista como una supercomputadora virtual, que hace uso de recursos heterogéneos distribuidos geográficamente. Pero al igual que los clusters la construcción de una grid es muy compleja, inclusive más que la de un cluster. A la fecha se ha optado más por la opción del cluster, esto debido a la complejidad de construcción y administración de una grid. Hay excepciones como el proyecto Seti@Home, que se encarga de buscar vida extraterrestre, este proyecto hace uso de una grid para hacer sus cálculos.

Como se puede ver hasta este punto hemos visto las dos principales arquitecturas del HPC, resaltando sus grandes ventajas al usar cualquiera de estas dos arquitecturas, pero también observamos que la complejidad para construir un cluster o una grid es muy alta, y aunque estas arquitecturas son muy potentes requieren de herramientas de software para que sean explotadas eficientemente.

2.2. Herramientas

La otra parte importante del HPC es el software, como es de esperarse al igual que el hardware, las herramientas de software usadas para la implementación de aplicaciones de HPC son complejas, a la fecha son contadas las herramientas que permiten implementar aplicaciones de HPC, nosotros hacemos énfasis en dos herramientas en particular, como lo son MPI y DLML, la primera porque es la más usada actualmente y la segunda es por la manera en que simplifica hasta cierto punto la programación paralela y es parte importante de esta tesis en particular.

2.2.1. Message passing interface

Message Passing Interface (MPI), es un estándar para la implementación de paso de mensajes entres procesadores, éste define la sintaxis y la semántica que debe contener una

biblioteca de paso de mensajes [11]. Ya que MPI es un modelo de programación basado en el envío y recepción de mensajes, es muy útil para resolver aplicaciones paralelas que se ejecutan sobre los nodos de un cluster. La arquitectura de cluster en complemento con la herramienta MPI, es la combinación de hardware y software más común en el ámbito del HPC, debido a que la mayoría de las aplicaciones paralelas pueden ser implementadas con MPI para explotar los recursos de un cluster.

Para implementar una aplicación o un problema con MPI el desarrollador debe hacer los siguiente:

- Identificar la estructura de los datos a procesar.
- Inicializar los datos.
- Inicializar el ambiente MPI.
- Definir el algoritmo para procesar los datos.
- Hacer el envío y recepción de los datos entre los nodos.
- Sincronizar los diferentes nodos.
- Obtener los resultados de los diferentes nodos.
- Finalizar el ambiente MPI.

Cada uno de los pasos anteriores tiene cierto grado de complejidad, sobre todo la sincronización entre los nodos. Para atacar cada uno de los pasos anteriores, MPI brinda rutinas, las cuales son usadas por los desarrolladores para hacer sus aplicaciones MPI, estas rutinas se pueden clasificar en los siguientes tipos:

- Rutinas utilizadas para inicializar, administrar y finalizar comunicaciones: estas son utilizados para delimitar la aplicación MPI y para administrar los nodos.
- Rutinas utilizadas para transferir datos entre un par de procesos: éstas son usadas para mandar mensajes y recibir de un nodo a otro, por lo regular hay un *send* y un *recv* que se en cargan del envío y recepción entre dos nodos (Figura 2.3).

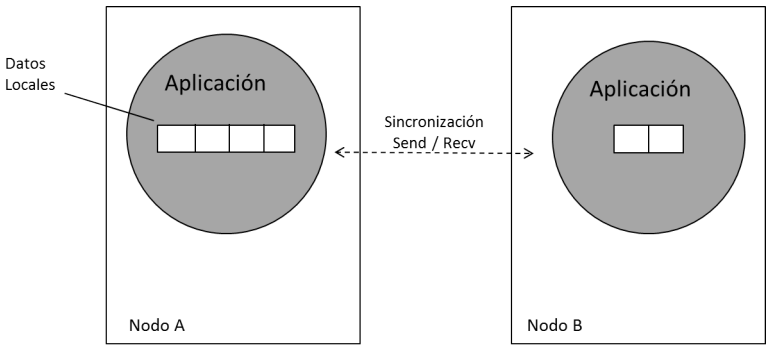

Figura 2.3: Arquitectura MPI

- Llamadas para transferir datos entre varios procesos: hay rutinas que hacen envío y recepción de múltiples mensajes, éstas por ejemplo pueden ser utilizadas si se quiere notificar a distintos nodos al mismo tiempo un cierto mensaje.

- Llamadas utilizadas para crear tipos de datos definidos por el usuario: éstas son usadas para transferir datos especiales definidos por el programador entre los nodos.

A continuación se muestra el programa "Hola Mundo" en MPI:

```
1 /* structure  MPI program*/
2 #include <stdio.h>
3 #include <mpi.h>
4
5 int main (argc, argv)
6 {
7     int argc;
8     char *argv[];
9     int rank, size;
10
11    // init MPI
12    MPI_Init (&argc, &argv);
13
14    // get id core
15    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
16
17    // get numbers of cores for  MPI application
18    MPI_Comm_size (MPI_COMM_WORLD, &size);
19
20    printf( "hi i'am core %d of %d\n", rank, size );
21
22    // ends MPI
23    MPI_Finalize();
24    return 0;
25 }
```

Aunque MPI es un estándar muy completo y con varias implementaciones, la programación con este estándar puede llegar hacer muy compleja, según el tipo de problema o modelo que se quiera implementar (e inclusive para programadores expertos), la programación de un problema con MPI puede llevar varios meses, mas aún para investigadores con poca experiencia, les es todavía más difícil plasmar su modelo sobre este estándar. A la fecha, en general las herramientas de software para HPC son muy complejas, otras como DLML que facilita en parte la programación, ya que su modelo de programación es casi secuencial.

2.2.2. Data list management library

Data List Management Library (DLML), es otra herramienta para el desarrollo de programas que requieran cómputo de alto desempeño [12]. DLML ha sido implementado en lenguaje C usando MPI, DLML surgió en gran parte para disminuir la complejidad de la

programación paralela, ya que su modelo de programación está basado en tipos de datos abstractos ADT (Abstract Data Type's) en particular de las listas, de esta manera se trata de facilitar la abstracción por parte del desarrollador, ya que la programación paralela se torna casi secuencial, además de que DLML proporciona un balance de carga entre los diferentes nodos del cluster.

Para implementar una aplicación o un problema con DLML el desarrollador debe hacer los siguiente:

- Modelar los datos sobre una lista.
- Definir la inicialización de la lista.
- Definir el algoritmo para procesar cada elemento de la lista.
- Obtener los resultados y finalizar la aplicación.

Como se puede ver, con DLML se reducen los pasos para construir una aplicación paralela, para atacar cada uno de los pasos anteriores, DLML cuenta con las siguientes rutinas:

- Rutinas de Señalización: estas se utilizan básicamente para marcar los límites del ambiente DLML.
- Rutinas de Manipulación: estas son las que se encargan de hacer el manejo sobre las listas, como insertar o eliminar de la lista.
- Rutinas de Recopilación: como su nombre lo indica se encargan de recopilar los resultados previos.

Las aplicaciones que se pueden implementar con DLML son aplicaciones estáticas y dinámicas. En las estáticas se conoce desde el inicio el total de los datos a procesar, por ejemplo la Multiplicación de Matrices. Por otra parte en las dinámicas el tamaño de los datos puede aumentar como en el caso del problema de las N-Reinas [12]. A continuación se presenta un ejemplo sencillo de un programa DLML:

```
1 /* structure  DLML program */
2 #include "dlml.h" //Librery DLML
3
4 //define items of list
5 typedef struct DLML_item Info;
6 struct DLML_item
7 {
8    // list items
9 };
10
11 Lista L;
12 Info elem;
13 main()
14 {
```

```
15      //init   DLML
16      DLML_Init();
17
18      //process list items
19      While (noVacia(L)){
20          elem=get(L.elem)
21          elimina(L.elem)
22          procesa(elem)
23      }
24
25      //ends DLML
26      DLML_Finalize();
27  }
```

Como se puede ver en el código anterior, un programa en DLML se puede ver casi secuencial, esto es porque la librería DLML se encarga de todo el paralelismo de manera transparente al desarrollador.
La arquitectura de DLML usa dos procesos por cada nodo (Figura 2.4), uno de estos procesos es la "aplicación" y el otro es el "balanceador", el primero se encarga de hacer las operaciones básicas sobre la lista como insertar, eliminar, etc. el balanceador se encarga de la comunicación y del balance de cargar con otros nodos. Entre la aplicación y el balanceador la comunicación se lleva a cabo por medio de mensajes locales y la comunicación entre balanceadores se lleva acabo por medio de mensajes remotos.

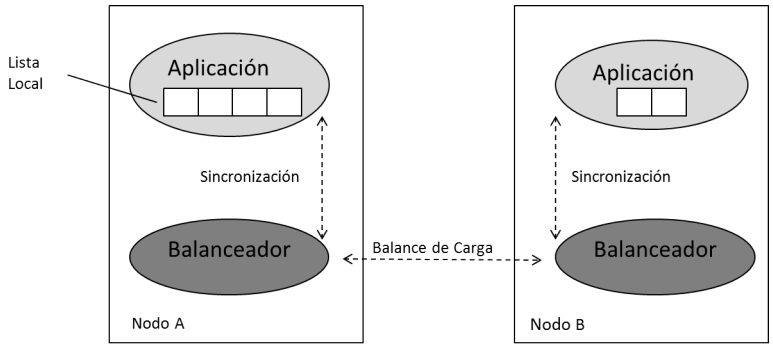

Figura 2.4: Arquitectura DLML

Como se puede ver DLML surgió para beneficiar en gran parte a los usuarios del HPC, haciendo una abstracción de la programación paralela a una forma casi secuencial basada en listas, a la fecha DLML ha sido probado con éxito en las arquitecturas de clusters y grids. No obstante, aunque DLML es una buena opción para la programación paralela, tiene limitantes como el modelado de los datos a una lista, ya que se puede llegar a complicar o bien se puede correr el riesgo de no hacerlo correctamente; sobre todo para programadores no tan expertos puede tornarse muy complicado esta abstracción.

En general observamos que las arquitecturas de hardware y las herramientas de software

del HPC son muy poderosas para el procesamiento de grandes cantidades de datos, pero lo complejidad de éstas es muy grande, sobre todo para programadores que no son expertos en aplicaciones paralelas, vemos que este contexto de la computación es altamente complejo para quienes requieren de éste, en contraste con lo que es el Cloud Computing, un contexto de la computación, que ha surgido con gran auge, gracias a la aceptación de los usuarios, ya que este paradigma ofrece todo tipo de servicios a usuarios de manera transparente.

Capítulo 3
Cómputo en nube

En relación al Cómputo en nube (La nube) o Cloud Computing como se le conoce, hoy por hoy no existe una definición única, muchos autores o personajes tiene su propia definición, pero lo que si es claro es que muchas de estas definiciones tiene algo en común y es lo siguiente: lo que se busca básicamente es ofrecer servicios de todo tipo relacionado a las TI (Tecnologías de la Información), para todo tipo de usuarios, donde los usuarios van desde personas hasta grandes empresas que requieren dichos servicios [13]. La idea principal de este concepto, es que por medio de dispositivos con una conexión a Internet, los usuarios hagan uso de los servicios sin saber que hay detrás de estos (Figura 3.1), esta propuesta a diferencia del HPC trata de facilitar todo a los usuarios haciendo trasparente lo que hay detrás de los servicios.

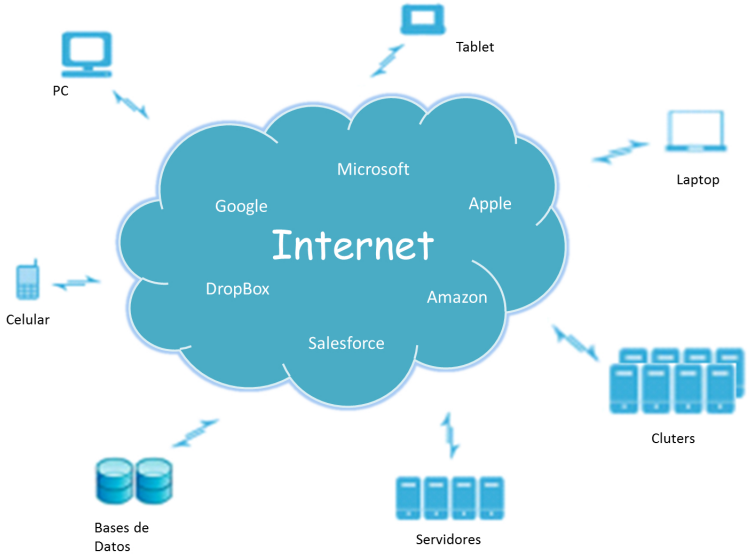

Figura 3.1: Cómputo en nube

Actualmente hay una gran cantidad de proveedores en la nube, que van desde instituciones hasta grandes empresas como lo son: Google, Amazon, Microsoft, por mencionar algunos, a cada implementación de cómputo se le puede nombrar como una "nube". Aunque la idea

del Cloud Computing es muy fácil de entender en la implementación se deben considerar ciertos aspectos importantes, como la arquitectura, el modelo de servicios, la seguridad, por mencionar algunos.

3.1. Aspectos del cómputo en nube

Como es de esperarse, el desarrollo de una nube puede ser tan complejo como el involucrar grandes infraestructuras de hardware, y grandes sistemas de programación todo funcionando con un fin, el proveer servicios a los usuarios [14].

Por esta razón la taxonomía del Cloud Computing se define con los siguientes aspectos que tienen que ser tomados en cuenta para el desarrollo de las nubes:

Arquitectura. La característica más importante a la hora de implementar Cloud Computing es la arquitectura [15], grandes proveedores de la nube tienen sus propias arquitecturas según los servicios que ofrecen y según el tipo de nube, que puede ser pública, privada, comunitaria e híbrida. Los servicios son un factor importante para definir la arquitectura apropiada de la nube, ya que la arquitectura debe adaptarse a las necesidades de los servicios, por ejemplo si tenemos una nube que ofrece un servicio de almacenamiento, entonces su arquitectura de la nube tiene que ser altamente escalable en recursos de almacenamiento.

En la actualidad el tipo de arquitectura más usado es SOA (Services Oriented Architecture), que en un principio surgió como una arquitectura orientada a negocios y por los beneficios en varios atributos de calidad [16], también debido a la gran cantidad de estándares que giran alrededor de ella, es actualmente muy usada en las TI (Tecnologías de la Información) y el Cloud Computing no es la excepción, algunos ejemplos de la implementación de SOA en Cloud Computing son:

- AWS(Amazon Web Services), la plataforma de Amazon para ofrecer servicios de IaaS, esta basada en SOA [17].
- SOCCA(Service Oriented Cloud Computing Architecture), que es una arquitectura que anexa una capa de SOA sobre diferentes nubes, con este tipo de arquitectura se busca que puedan interoperar distintas nubes [18].
- SOACC (Service-Oriented Storage Resource Architecture for Cloud Computing) en ésta los autores hacen énfasis sobre cómo administrar y utilizar eficientemente los recursos de almacenamiento, en particular esta arquitectura trata de resolver el almacenamiento en la nube (Cloud Storage) [19].
- EA (Enterprise Architecture), que es una arquitectura basada en SOA y enfocada a diferentes sistemas empresariales trabajando como una sola nube [20].
- CCOA (Cloud Computing Open Architecture), que es una arquitectura basada en 7 cualidades, integradas con el patrón de arquitectura SOA [21].

Como se puede ver la arquitectura es la parte fundamental del Cloud Computing, en general cada arquitectura de una nube sigue un patrón arquitectónico, anexándole por lo regular ciertas capas, adecuando el patrón de acuerdo al modelo de servicios.

Servicios. En la nube hay que tener muy en cuenta los servicios que se ofrecerán, un servicio se denota como XaaS donde la X puede representar S (Software), P (Platform), I (Infrastructure) [22]. En la actualidad hay muchos modelos de servicios, los más ofrecidos entran en los tres modelos anteriores, en la mayoría de los casos estos servicios, desde el punto de vista del usuario, deben cumplir con otros aspectos importantes como lo son: la seguridad, tolerancia a fallas, escalabilidad de recursos.

Seguridad. Esta característica es un factor muy importante para los usuarios a la hora de optar por el Cloud Computing, más aún si se habla de nubes privadas. La seguridad es muy importante para cualquier sistema de cómputo, hoy en día se cuenta con protocolos seguros como TLS, SSH, SFTP, SSL etc., además de estos protocolos existen mecanismos usados como la autentificación, la autorización, el no repudio entre otros.

El estándar International Standards Organization (ISO) 7498-2 establece los requerimientos (Figura 3.2) que se deben tomar en cuenta para el Cloud Computing [23].

Tipos de Nubes											
Requerimientos de Seguridad		Pública				Privada				Híbrida		
	Identificación & Autenticación	x	x	*		x	x	*		*	x	*
	Autorización	x	x	x		*	x	*		*	x	*
	Confidencialidad	*	x	*		*	x	x		*	x	*
	Integridad	x	x	*		*	x	x		x	x	x
	No-repudio	*	x	*		*	x	*		*	*	*
	Disponibilidad	x	*	x		x	x	x		*	*	*
		IaaS	SaaS	PaaS		IaaS	SaaS	PaaS		IaaS	SaaS	PaaS
Modelos De Servicios												
X = Requerimiento obligatorio												
* = Requerimiento opcional												

Figura 3.2: Requerimientos de seguridad en la nube

La seguridad en el cómputo en nube como en cualquier sistema es muy importante, empresas como IBM tienen sus propios modelos de seguridad para este tipo de cómputo [24].

Tolerancia a Fallas. Por supuesto una característica importante es la tolerancia a fallas,

esto asegura por lo regular que haya un respaldo de los datos, por otra parte hace posible una alta disponibilidad. Esta característica es todavía más importante en los servicios de paga, ya que los clientes esperan que haya un respaldo de su información, además de que haya una alta disponibilidad de los recursos, aunque tal vez la tolerancia a fallas está muy relacionado con el servicio y al tipo de nube [15]. Regresemos al ejemplo de la nube para almacenamiento, en este caso no es lo mismo una tolerancia a fallas para un usuario ordinario, que tal vez use el servicio de manera gratuita, a la tolerancia a fallas que debe haber para los recursos de una empresa que paga por el servicio y que muy probablemente tenga información muy valiosa. Como se puede ver, la tolerancia a fallas es muy importante pero también muy relativa al servicio y a los usuarios.

Escalabilidad de recursos. Ya que la información generada por las organizaciones y usuarios crece en demasía, las nubes tienen que ser altamente escalables en recursos, tanto para almacenamiento como para procesamiento de la información generada a través del tiempo [16]. Hoy en día, la información generada por empresas o instituciones ya no son solo gigabytes o terabytes, sino que ya llegan a cantidades de petabytes de información, es por esto que las nubes deben cumplir con esta característica. La escalabilidad de recursos en el Cloud Computing implica tres aspectos, que son la virtualización de recursos, el balance de carga entre éstos y la elasticidad dentro de la nube.

Virtualización. Esta es la parte de la nube que hace un mapeo de distintos recursos lógicos sobre uno o varios recursos físicos, básicamente es una abstracción lógica de una infraestructura [1]. La virtualización se clasifica en tres tipos, virtualización de servidores, virtualización de almacenamiento y virtualización de las redes. Un ejemplo de una buena virtualización en una nube, puede ser cuando se tiene un servidor con grandes capacidades de cómputo, entonces para ejecutar ciertas aplicaciones que no necesiten de grandes recursos se procede a virtualizar servidores a la medida a partir del servidor físico para cada aplicación correspondiente. La virtualización ha sido un factor muy importante para el Cloud Computing, sobre todo para las nubes que ofrecen servicios de infraestructura (IaaS), ya que permite ofrecer servicios a la medida y así hacer un mejor uso del hardware y de cierta manera protegerlo contra ciertos incidentes.

Balance de Carga. Esta es una característica tal vez no es primaria a diferencia de las anteriores, aunque el balance de carga en una nube repercute a favor del desempeño, además de hacer un mejor uso de los recursos [1]. Esta característica debe ser transparente al cliente, por lo cual se vuelve una característica compleja a la hora de su implementación, pero que da una mayor calidad a la nube. En general el balance de carga implica una capa intermedia o un mecanismos que se encarga de asignar tareas a los distintos recursos de la nube.

Elasticidad. Este concepto ha surgido plenamente por el Cloud Computing y esta directamente relacionada con los IaaS, esta característica es asociada a dos aspectos antes descritos, que son la escalabilidad de recursos y la virtualización. La elasticidad se refiere a qué tan

fácil la nube puede escalar la infraestructura virtualizada según a las necesidades del usuario, visto por el lado del usuario, es qué tan fácil el usuario puede incrementar o disminuir sus recursos virtualizados sobre la nube de acuerdo a sus necesidades. Lo que se busca con la elasticidad es que el usuario pueda escalar dentro de la nube sus recursos con la mínima fricción sobre los que ya tiene y en el menor tiempo posible [26]. La elasticidad es un aspecto propio del Cloud Computing, a diferencia de los aspectos de interoperabilidad y gobierno, éstos son aspectos fundamentales para cualquier sistema de cómputo, y las nubes no son la excepción.

Interoperabilidad. Este es muy importante para cualquier nube, no importa el tipo o servicios que ofrezca, la interoperabilidad va a permitir que la nube sea capaz de interactuar con otras aplicaciones u otros servicios. Esto asegura que haya una flexibilidad y fácil escalabilidad de la nube [18]. La interoperabilidad es un factor en favor del uso de SOA en el Cloud Computing, ya que SOA asegura de cierta manera la interoperabilidad, esto es debido a las tecnologías estándar de SOA.

Gobierno. Este es una parte de la nube, la cual es un tema de gran auge para los proveedores de Cloud Computing, debido a que es muy difícil mantener un control dentro de la nube. Cuando hablamos de gobierno hablamos del control; en este caso el gobierno en la nube se refiere a procesos o mecanismos que permitan tener un control sobre ésta. A la fecha los proveedores cuentan con sus propias políticas para el control de la nube, pero no hay todavía estándares que sigan los proveedores de Cloud Computing para dicho control. Un ejemplo claro del gobierno en la nube son los SLA's (Service Level Agreement) estos son un contrato entre el proveedor y el consumidor. Lo que buscan los proveedores con un buen gobierno es que haya procesos, mecanismos o estructuras de gestión que brinden seguridad y confiabilidad sobre la información, y estos procesos o mecanismos deberían ser reusable, medibles, definibles y con una mejora continúa para todos sus consumidores [27]. El gobierno es tal vez una parte que todavía no es muy clara a la hora de implementar el Cloud Computing, en contraste con SOA, que en su contexto el gobierno es un aspecto ya maduro, y que cuenta con procesos y mecanismos estandarizados para mantener un control de la información [28], caso contrario en el Cloud Computing el gobierno es uno de los puntos a mejorar.

Como se puede ver el concepto de la nube puede ser muy fácil de entender pero a la hora de llevarlo a cabo en la implementación surgen varios aspectos que tienen una gran complejidad para su desarrollo, los aspectos anteriores deben ser tomados en cuenta para implementar Cloud Computing, A la fecha, una nube se puede clasificar por su tipo y por los modelos de servicios que ofrece a los usuarios.

3.2. Tipos de nubes

En la actualidad, las nubes que se ofrecen en Internet se clasifican en cuatro diferentes tipos (nubes públicas, privadas, comunitarias e híbridas) [15]:

Las *nubes públicas*, son abiertas a todo tipo de usuarios, en estas nubes no hay mucha seguridad y por lo regular todo está disponible para otros usuarios, en algunos de estos casos no hay garantía de confidencialidad de la información. Las nubes públicas en la mayoría de los casos son grandes infraestructuras que deben escalar en almacenamiento, ya que la información que se maneja crece en demasía. Algunos ejemplos de nubes públicas son Google, Drop-Box, etc., por lo regular estas nubes son gratuitas [1].

Las *nubes privadas*, como lo indica su nombre no son abiertas a cualquier usuario u organización, por lo regular éstas son para organizaciones que requieran mayor seguridad de su información donde se garantice una confidencialidad de sus datos. La infraestructura de estas nubes pueden ser no muy grandes, a veces son infraestructuras dentro de la misma organización o pueden ser provistas por los grandes proveedores de servicios, las nubes privadas en general implican una mayor calidad en los servicios que ofrece, a diferencia de las públicas. Como es de esperarse los servicios privados en la mayoría de los casos tienen un costo monetario [1].

Las *nubes comunitarias*, este tipo de nube puede ser abierta a todo público o tal vez sólo ciertos usuarios, este tipo de nube se caracteriza por tener su infraestructura segregada en diferentes lugares geográficos, además de que es mantenida y sustentada por un grupo de usuarios u organizaciones, a las cuales les interesa el crecimiento y mantenimiento de la misma. Este tipo de nube puede ser visto como el open source de las nubes, ya que en la mayoría de los casos es mantenida por los mismos usuarios [27].

Las *nubes híbridas*, son aquellas que son una mezcla de públicas, privadas o comunitarias, esto se refiere a que la nube provee servicios a todo público y servicios privados, esto es favorable para organizaciones o usuarios que necesiten sólo algunos servicios con una mayor calidad [27].

Como se puede ver los cuatro tipos de nubes son una clasificación muy acorde a los costos monetarios, a la seguridad que proporcionan y la calidad de los servicios. Para que los usuarios y organizaciones hagan una buena elección del tipo de nube a usar, tienen que tener claras sus necesidades y sus prioridades. En general hablando de costos monetarios los clientes (usuarios, organizaciones, etc.) de las nubes pagan por la demanda de los servicios, esto es mientras más requieran mayor será el pago.

Por otra parte, las nubes también se distinguen por los modelos de servicios que ofrecen a los usuarios, en el Cloud Computing hay tres principales modelos de servicios que son el SaaS, PaaS e IaaS.

3.3. Modelos de servicios

En el Cloud Computing lo primero que se debe tener claro es el ¿Qué?, es decir, qué servicios va a proveer la nube a los usuarios, y por parte de los usuarios que servicios requieren [22]. Al día de hoy, en Internet hay una infinidad de servicios, éstos se representan como XaaS donde X es el modelo de servicio, pudiendo ser SaaS, PaaS, IaaS, (Software, Plataforma e Infraestructura) hay muchos modelos de servicios, pero la mayoría entra en estos tres modelos, que pueden ser vistos como los tres niveles de la pila del Cloud Computing (Figura 3.3).

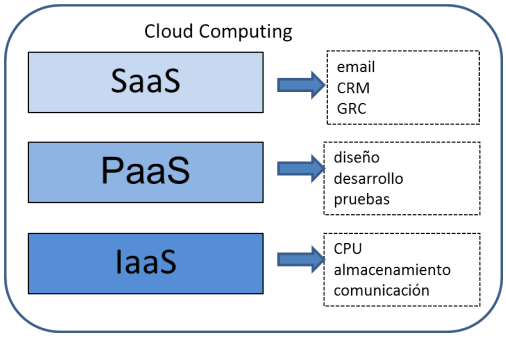

Figura 3.3: Modelos de servicios

3.3.1. Software como un servicio

El Software como un servicio (SaaS), este puede ser visto como el servicio más completo y a la medida para los usuarios. El SaaS es un software sobre Internet, el cual puede ir desde aplicaciones pequeñas como una agenda, hasta grandes sistemas como un CRM (Customer Relationship Management) para empresas [1]. En general, en este nivel de servicio el usuario o el consumidor del servicio no tiene que preocuparse por nada el sólo usa el servicio, este modelo es el primero en la capa de abstracción de modelos, está por encima de los modelos de PaaS e IaaS que más adelante se mencionan. El SaaS describe un modelo que es desplegado sobre Internet, y que puede ser accedido globalmente desde cualquier lugar a través de un navegador web, el SaaS está diseñado para que simplemente lo use el usuario sin que tenga que preocuparse de otros aspectos como la plataforma o la infraestructura, también algo que se busca es que el SaaS pueda ser operado por los usuarios desde dispositivos móviles debido a su gran avance.
Actualmente, la mayoría de las empresas que proveen aplicaciones web o aplicaciones de escritorio, están haciendo una transformación de sus aplicaciones para ser ofrecidas como SaaS para los usuarios. Un ejemplo representativo de un SaaS es el CRM de Salesforce, hoy en día más de 100,000 de consumidores confían en las funciones de este CRM, gran parte de su éxito se debe a que su tecnología es accesible sobre los principales plataformas de los dispositivos móviles [29].

3.3.2. Plataforma como un servicio

La Plataforma como un servicio (PaaS), son aquellos servicios en los cuales los consumidores pueden crear aplicaciones a partir de herramientas que proporciona el proveedor, estas herramientas pueden ser middlewares, librerías, API's , etc. Es común que las aplicaciones desarrolladas con un PaaS sean compatibles con la infraestructura del proveedor [25]. Este nivel de servicio es usado en general por desarrolladores o empresas de TI (Tecnologías de la Información), de esta manera ellos utilizan una tecnología ya probada y con la posibilidad de utilizar la infraestructura del proveedor, matando así dos pájaros de un tiro, el desarrollo y la infraestructura para sus sistemas. En general los PaaS sirven como herramientas para pasar al nivel más alto de la pila del Cloud Computing que son los SaaS, es decir, lo implementado con los servicios PaaS en general pasan hacer SaaS, en cuanto a seguridad; en los servicios PaaS el proveedor ofrece herramientas para que el usuario implemente su propia seguridad [1].
Hoy en día se encuentran muchos proveedores de PaaS en la nube, un ejemplo representativo es Google con GAE (Google App Engine), el cual permite a los usuarios poder desarrollar aplicaciones web escalables, compatibles con la infraestructura de Google [30]. El punto débil de los PaaS, es que en general las aplicaciones implementadas con tales servicios están sujetas al proveedor y en la mayoría de los casos no son portables entre distintas nubes.

3.3.3. Infraestructura como un servicio

La Infraestructura como un servicio (IaaS), es la capa más baja del Cloud Computing. En este nivel de servicio básicamente se proveen servicios de infraestructura como lo son: el ancho de banda, recursos de almacenamiento y procesamiento. En general los IaaS se hacen por medio de recursos lógicos sobre los recursos físicos a través de la virtualización, esto en gran medida se hace para proteger los recursos físicos y explotar mejor las capacidades de estos recursos. En este nivel el usuario tiene mayor responsabilidad sobre la seguridad, ya que el proveedor brinda una seguridad para administrar los recursos, pero lo que se maneja dentro de cada recurso adquirido por el usuario es responsabilidad del usuario [1]. Los IaaS son complejos en contraparte de los SaaS, sobre todo para usuarios no expertos, en la mayoría de lo casos son desarrolladores especializados en el ámbito de infraestructura los que se encargan de hacer uso de estos servicios para sus empresas. Hoy en día este servicio es muy demandado por empresas que buscan virtualizar su infraestructura en las nubes de los proveedores.
El mejor ejemplo de un proveedor de IaaS es Amazon, este es uno de los pioneros del Cloud Computing. Amazon por medio de AWS (Amazon Web Services) provee servicios de IaaS, la nube de Amazon proporciona una infraestructura muy fiable y escalable para la implementación de soluciones referidas a la infraestructura [31]. Haciendo uso de estos servicios los usuarios de Amazon han podido virtualizar su infraestructura en la nube, evitando así gastar grandes cantidades de dinero en superservidores.

En términos generales tenemos tres modelos de servicios, que pueden verse como tres ni-

veles diferentes de abstracción, en el primer nivel de la pila del Cloud Computing esta el IaaS, incluye la capa de los recursos e infraestructura desde instalaciones hasta las estructuras de hardware. En el segundo nivel esta el PaaS que depende directamente del IaaS, este nivel provee de herramientas como API's, middlewares, librerías, etc., para que el usuario pueda desarrollar sus aplicaciones. En el último nivel esta el SaaS, éste es creado a partir de los niveles anteriores de PaaS e IaaS, este nivel proporciona al usuario un entorno completo para ser usado únicamente, sin que el usuario tenga que preocuparse de infraestructura o alguna plataforma. Estos modelos de servicios en general cumplen con las necesidades de los usuarios, ya sean organizaciones, empresas, o usuarios independientes.

No obstante, a la fecha hay un modelo de servicio que ha emergido por la demanda de un cómputo de alto desempeño sobre Internet, a este modelo de servicios se le ha denominado HPCaaS (HPC as a Service), estos son servicios relacionados con el HPC que ofrecen en la mayoría de los casos recursos de clusters virtualizados y en algunos casos herramientas de software como MPI, para los usuarios del HPC.

3.3.4. HPC como un servicio

Anteriormente se revisó con detalle el HPC (Capítulo 2), que se divide en dos partes el hardware y el software; y que es usado a la fecha para resolver problemas complejos que requieren grandes cantidades de procesamiento. En contraste con el Cloud Computing, el HPC es muy complejo para los usuarios, tener que lidiar con arquitecturas de hardware como los clusters o programar aplicaciones paralelas con MPI, por mencionar algunos casos. Son escenarios complejos con los que se enfrentan los usarios del HPC, relacionado a esto, en Internet han surgido servicios enfocados al HPC, a los cuales se les conoce HPCaaS, por medio de estos servicios se busca llevar a usuarios el HPC a través de Internet.

Entre los proveedores de HPCaaS actuales están Amazon, Windows Azure que proveen a los usuarios clusters inclusive con herramientas de software como MPI, éstas dos como nubes privadas. Algunos como Open cirrus [32] y SGI-Cyclon [33], que ofrecen clusters virtualizados para las investigaciones científicas. En general los HPCaaS actuales, en su gran mayoría están enfocados a la parte del hardware, en base a dos aspectos del Cloud Computing que son la virtualización y la elasticidad [34]; es por esto que los HPCaaS ofrecidos hasta la fecha en Internet están a nivel de IaaS, ya que éstos sólo proveen la infraestructura que puede estar formada por clusters virtualizados, acompañados de una herramienta de programación como MPI u otra. A la fecha, los proveedores de HPCaaS no ofrecen como tal, las aplicaciones HPC como servicios, sino que solo proporcionan una infraestructura en la cual el usuario pueda ejecutar una aplicación paralela previamente desarrollada. Por tal razón la complejidad del HPC no es disminuida, al contrario, en algunos casos para los usuarios puede ser más complejo lidiar con una infraestructura virtualizada en la nube de los proveedores y trasladar sus aplicaciones HPC.

Por tal motivo una mejor solución para usuarios del HPC, no expertos en programación paralela o en arquitecturas como clusters, son los HPCaaS a un nivel SaaS, es decir, una

aplicación HPC vista como un servicio web, en el cual el investigador o usuario ya no tiene que preocuparse ni de la infraestructura o la programación, sino que sólo se preocupe por el análisis de los resultados que arroja la aplicación HPC vista como un servicio en Internet, así con esto se busca que el usuario solo la mande a ejecutar a través de un navegador web dicha aplicación.

Aunado a este escenario de HPCaaS a nivel de SaaS, un grupo de desarrolladores del HPC Advisory Council, crearon una nube privada en la cual se ofrecen servicios de aplicaciones implementadas con MPI de los algoritmos paralelos CPMD (Car-Parrinello Molecular Dynamics) y NAMD (Nanoscale Molecular Dynamics) sobre arquitecturas de clusters, para un grupo de investigadores que requieren de estas aplicaciones [4]. Ambas aplicaciones son problemas de la biociencia, estos servicios surgieron para que investigadores no expertos en computación, puedan ejecutar estas aplicaciones y así de manera fácil obtengan los resultados de las ejecuciones, que es lo que a ellos les interesa. También ha surgido por la problemática de clusters dedicados (Figura 3.4), además de que los recursos se encuentran administrados detrás de la nube y puede haber distintas aplicaciones usando los mismos recursos, lo cual favorece a una mejor explotación de los recursos.

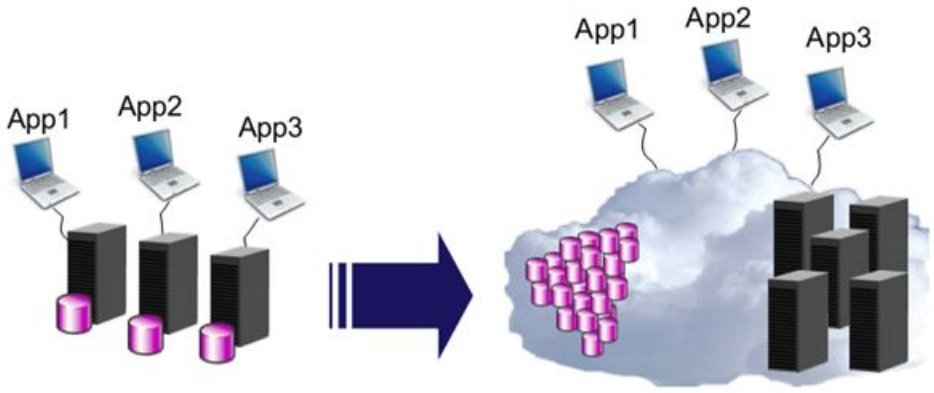

Figura 3.4: Esquema tradicional a esquema HPCaaS

Este es un perfecto preámbulo para el propósito de este trabajo, donde se integran los contextos del Cloud Computing y el HPC, en el caso anterior se observa que la gran aportación fue el crear estos HPCaaS para los científicos que requerían de los algoritmos CPMD y NAMD, en el cual no sólo se engloba los clusters, sino que también se proveen ya las aplicaciones CPMD y NAMD implementadas con MPI como un servicio, en este caso se puede ver como el HPCaaS es llevado a un nivel de SaaS, donde los usuarios ya no se preocupan de la infraestructura o la programación, en caso contrario de los HPCaaS de los grandes proveedores como Amazon o algún otro gran proveedor su HPCaaS son a nivel de IaaS. Algunos otros trabajos muestran escenarios posibles de cómo ofrecer las aplicaciones HPC como servicios a nivel de SaaS [34].

En este trabajo se ha implementado una plataforma con la que se propone ofrecer HPCaaS a

nivel de SaaS, esta plataforma ha demostrado ser escalable en recursos de hardware y flexible a nivel de herramientas de software. A esta plataforma se le ha denominado Cloud-DLML y es un caso parecido al de HPC Advisory Council, a diferencia de que Cloud-DLML esta basado en una arquitectura desacoplada de la infraestructura de HPC, permitiéndole en teoría hacer uso de clusters en cualquier lugar del mundo, además de que brinda una librería core para el desarrollo de HPCaaS.

Capítulo 4
Cloud-DLML

Una vez visto las bases que han impulsado a la creación de Cloud-DLML, por un lado se tiene el HPC que es demasiado complejo para los usuarios, en contraste con el Cloud Computing que es fácil de usar para los usuarios. Cloud-DLML es una plataforma en la cual se integran el HPC y el Cloud Computing para ofrecer servicios de HPCaaS a nivel de SaaS (Figura 4.1), así de esta manera se facilita a los usuarios la ejecución de aplicaciones HPC a través de dispositivos con conexión a Internet.

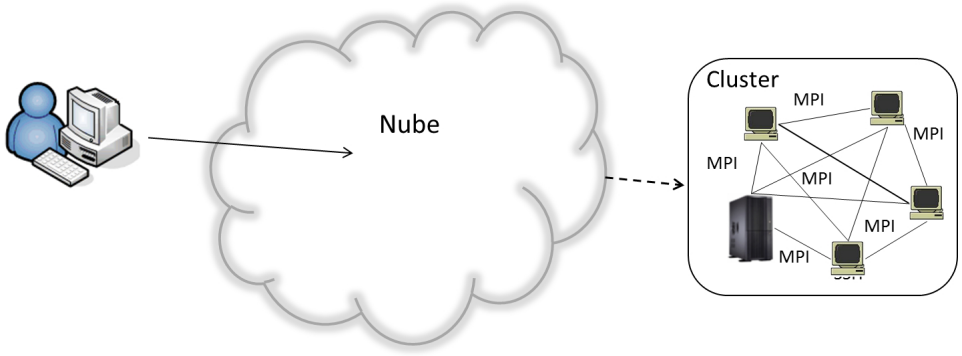

Figura 4.1: Propuesta HPCaaS

En base a los aspectos que se deben considerar para implementar Cloud Computing, los aspectos principales que definen a Cloud-DLML son los siguientes:

- Cloud-DLML es una nube que va dirigida a usuarios que realizan cómputo científico (físicos, matemáticos, químicos, biólogos, desarrolladores de HPC, etc.) principalmente, sobre todo para usuarios que cumplen con alguno de los siguientes puntos: poca habilidad para programar, falta de recursos, quienes deseen reutilizar los servicios existentes en otras aplicaciones, quienes quieran compartir sus aplicaciones HPC que se ejecuten sobre clusters.

- Cloud-DLML se considera una nube pública y comunitaria, abierta al uso y colaboración de la comunidad científica, en un principio está abierta para todo público pero los

servicios en un inicio serán algo básicos. Pero la plataforma está implementada para empezar a ofrecer servicios más especializados para la comunidad científica, de esta manera el Cloud-DLML queda abierto a implementación de nuevos servicios.

- Cloud-DLML ofrece servicios HPCaaS, en principio está pensado para ofrecer servicios basados en aplicaciones DLML usando clusters como infraestructura. Estos servicios serán ejecutados desde interfaces web por medio de algún dispositivo con conexión a Internet.

- Cloud-DLML ha sido implementado sobre una arquitectura SOA (Services Oriented Architecture) anexando una capa HPC. Con el fin de implementar una arquitectura Cloud-DLML basada en tecnología estándar.

- El Cloud-DLML implementa una seguridad basada en protocolos seguros y una autenticación a nivel de cluster por medio de SSH (Secure Shell) y SFTP (Secure File Transfer Protocol).

- En cuanto al control dentro de Cloud-DLML, se lleva un registro de todas las ejecuciones de las distintas aplicaciones en los distintos clusters.

Los anteriores son aspectos que definen a Cloud-DLML en el contexto del Cloud Computing, que en pocas palabras se puede describir como una nube para brindar servicios HPCaaS. Además de que los aspectos anteriores de Cloud-DLML fueron fundamentales para su implementación, ya que estos fueron tomados en cuenta para el diseño, la arquitectura y la comunicación del sistema Cloud-DLML.

4.1. Diseño

Gran parte del diseño de Cloud-DLML, fue enfocado a encontrar la arquitectura adecuada para el sistema. Como se sabe, gran parte del diseño de un sistema se basa en encontrar la arquitectura adecuada para dicho sistema, ya que la arquitectura determina gran parte de la calidad de éste. Por esta razón gran parte del diseño esta enfocado a encontrar la arquitectura adecuada para Cloud-DLML. Durante el transcurso de este trabajo se analizaron muchas arquitecturas de Cloud Computing, en las cuales se observó, que el tipo de arquitectura en diferentes nubes está basado en SOA, lo cual nos daba un panorama de lo que se esperaba para Cloud-DLML.

No obstante hoy en día, la ingeniería de software marca que el diseño de la arquitectura debe ser en base a procesos o métodos, para que ésta pueda ser adecuada con los requerimientos funcionales, que a su vez cumpla con cierta calidad y todo esto bajo ciertas restricciones del

desarrollo del sistema. Es por esto que el diseño de la arquitectura de Cloud-DLML se hizo en base a una adecuación de dos métodos de ingeniaría de software sumamente probados, estos métodos son el QAW (Quality Attribute Workshop) y el ADD (Attribute-Driven Design). A partir de los métodos QAW/ADD, y sin perder de vista las arquitecturas analizadas del Cloud Computing, es como se ha diseñado la arquitectura para Cloud-DLML (Figura 4.2).

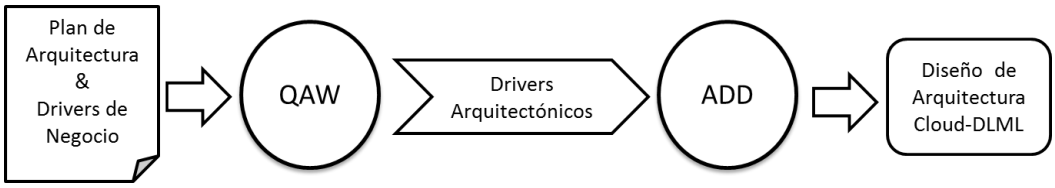

Figura 4.2: Aplicación de QAW/ADD sobre Cloud-DLML

QAW/ADD son dos métodos muy completos proporcionados por el SEI (Software Engineering Institute), para el diseño de arquitecturas de sistemas [35]. Estos métodos ayudan a los equipos de desarrollo a diseñar arquitecturas de calidad, que cumplan con los requerimientos y restricciones del sistema. Para Cloud-DLML se ha adoptado un método con 4 pasos basados en QAW/ADD, en donde se engloban los 8 pasos originales, la adecuación de estos se hizo por las siguientes razones: por la investigación previa del estado del arte que hay acerca del proyecto, la cual abarcó desde la problemática hasta las soluciones que se han ofrecido por las diferentes arquitecturas analizadas del Cloud Computing, la envergadura del sistema ya que este en un principio es un sistema no tan grande, porque se tiene bien definido el alcance del proyecto y la última razón es porque no hay un gran número de involucrados, ya que este proyecto es un problema en una primera instancia un sistema que surge por un proyecto de tesis.

Pasos del Método para Cloud-DLML	Pasos QAW/ADD
1. Plan arquitectónico	1. Introducción y presentación del QAW 2. Presentación de negocios 3. Presentación del Plan arquitectónico
2. Identificación de los driver Arquitectónicos	4. Identificación de los driver Arquitectónicos
3. Consolidación de escenarios	5. Discusión de Escenarios 6. Consolidación de escenarios 7. Priorización de escenarios
4. Refinamiento de escenarios con ADD	8. Refinamiento de escenarios

Tabla 4.1: Método para el diseño de la arquitectura de Cloud-DLML

Para nuestro método de 4 pasos basándose en los pasos originales de QAW/ADD (Tabla 4.1), cada paso tiene su propia finalidad: el paso 1 nos ayuda a saber las metas del sistema

y da un panorama de lo que se espera, el paso 2 identifica lo que se quiere cumplir, cómo se va a cumplir y bajo qué condiciones, el paso 3 nos da una visualización de cómo debería comportarse Cloud-DLML bajo ciertos escenarios, por último el paso 4 nos da posibles soluciones para cumplir con la arquitectura adecuada para Cloud-DLML.
A continuación, se muestra cada uno de los pasos y la manera en que se fueron aplicando cada uno de estos para definir el diseño de Cloud-DLML:

1. Plan arquitectónico: en este caso para Cloud-DLML, se contempla como driver de negocio el proveer HPCaaS, donde los usuarios puedan ejecutar aplicaciones HPC sin tener que preocuparse de la programación o la infraestructura. Para el plan de la arquitectura, se espera una arquitectura basada en SOA, esto por la investigación de las arquitecturas previas y que en muchos de los casos se han probado con éxito, por otra parte se tiene todo el contexto de la problemática y el alcance del proyecto.

2. Identificación de los drivers arquitectónicos: en este paso nos damos a la tarea de encontrar los drivers arquitectónicos de Cloud-DLML que son: los atributos de calidad deseables, los requerimientos del sistema y las restricciones para el sistema (Tabla 4.2).

Atributos de Calidad	Requerimientos Funcionales	Restricciones
Usabilidad	Servicios de aplicaciones DLML	
Escalabilidad	Ejecución de aplicaciones HPC como servicios	Software libre
Interoperabilidad	Envío de notificaciones	
Adaptabilidad	Usar un cluster como recursos	
Portabilidad		

Tabla 4.2: Drivers arquitectónicos de Cloud-DLML

3. Consolidación de escenarios: en este paso encontramos los escenarios de acuerdo a los drivers arquitectónicos (Tabla 4.2). Que según el QAW/ADD debe de haber un escenario por cada atributo de calidad, donde se satisfagan los requerimientos funcionales, y se contemplen las restricciones del sistema. A continuación se muestra en orden de prioridad cada uno de los escenarios, en un formato de una tabla estándar del SEI:

1. Escenario Usabilidad: este escenario se refiere a la interacción que tiene el usuario con Cloud-DLML.

Factor	Valor
Fuente	Usuario
Estímulo	El usuario quiere usar el sistema y hacer uso de los servicios
Artefacto	Portal Web de Cloud-DLML
Entorno	En tiempo de ejecución
Respuesta	El portal web presenta una interfaz agradable e intuitiva para el usuario, exponiendo así una fácil navegación dentro del portal, por otra parte el diseño del portal tiene que ser agradable a la vista de los usuarios, además de que el diseño minimiza errores debido al posible mal uso por parte del usuario.
Medida de respuesta	Tiempo en que tarda el usuario en familiarizarse con el portal y la satisfacción del usuario.

Tabla 4.3: Escenario de usabilidad

2. Escenario de Escalabilidad: este escenario es en relación a la escalabilidad de recursos para Cloud-DLML.

Factor	Valor
Fuente	Administrador del sistema
Estímulo	El administrador desea agregar un cluster al sistema
Artefacto	Sistema
Entorno	En tiempo de desarrollo
Respuesta	El sistema anexa el cluster al catálogo de clusters disponibles dentro del Cloud-DLML, de esta manera el sistema debería poder ejecutar las aplicaciones HPC en este cluster, por otra parte el sistema no necesita que este cluster sea dedicado, además de que la administración del cluster puede ser ajena a la del sistema.
Medida de respuesta	Tiempo que se toma en desarrollo para agregar conexiones al cluster, y el número de clusters que puede soportar.

Tabla 4.4: Escenario de escalabilidad

3. Escenario de Interoperabilidad: este escenario se refiere a la posible interacción que puede existir entre Cloud-DLML con otros sistemas o nubes.

Factor	Valor
Fuente	Sistema externo
Estímulo	Un sistema externo desea ejecutar los servicios del Cloud-DLML
Artefacto	Servicios de Cloud-DLML
Entorno	En tiempo de ejecución
Respuesta	Los servicios reciben la petición por parte del sistema externo y ejecutan la aplicación HPC solicitada en el cluster solicitado, termina la ejecución y regresa la respuesta al sistema externo. Los servicios o módulos del sistema son capaces de invocar otros servicios.
Medida de respuesta	Tiempo de respuesta por parte del servicio y solicitudes de sistemas externos que se pueden atender.

Tabla 4.5: Escenario de interoperabilidad

4. Escenario de Adaptabilidad: este se refiere a la facilidad para modificar o anexar funcionalidad a Cloud-DLML por parte de los desarrolladores.

Factor	Valor
Fuente	Desarrolladores
Estímulo	Se requiere modificar el sistema o aumentar la funcionalidad
Artefacto	Cloud-DLML
Entorno	En tiempo de desarrollo
Respuesta	Los desarrolladores anexan nuevos servicios de aplicaciones HPC sin afectar las que ya están, si se requiere modificar algún servicio en particular se puede modificar sin afectar a otros servicios, el sistema soporta los cambios que se presentan. Hay módulos del sistema que pueden ser reutilizados para el desarrollo de nuevos servicios y funcionalidades.
Medida de respuesta	Tiempo que toma modificar algún servicio o alguna funcionalidad del sistema.

Tabla 4.6: Escenario de adaptabilidad

5. Escenario de Portabilidad: en este se busca que el usuario pueda hacer uso de los servicios de Cloud-DLML sin tener que instalar nada, ni que dependa de alguna plataforma o dispositivo en especial.

Factor	Valor
Fuente	Usuarios
Estímulo	El usuario desea usar el sistema con algún dispositivo con conexión a Internet
Artefacto	Portal Web de Cloud-DLML
Entorno	En tiempo de ejecución
Respuesta	El portal esta disponible y se puede ejecutar en diferentes navegadores web, o inclusive de diferentes dispositivos con conexión a Internet sin perdida de funcionalidad.
Medida de respuesta	Tiempo de carga del portal web sobre el navegador web y el número de los diferentes dispositivos que pueden hacer uso de Cloud-DLML.

Tabla 4.7: Escenario de portabilidad

Por medio de los escenarios anteriores se visualiza como se debería comportar el sistema ante ciertos eventos, así con estos escenarios establecemos el comportamiento esperado de Cloud-DLML, así como las métricas o parámetros de referencia para saber sí Cloud-DLML esta cumpliendo con la calidad esperada.

4. Refinamiento de escenarios con ADD: en este último paso nos enfocamos sólo a los atributos de calidad para encontrar los patrones arquitectónicos o estrategias adecuadas para satisfacer cada atributo, con la finalidad de llegar a una arquitectura de calidad que se adapte perfectamente a las necesidades. Es importante mencionar que en los sistemas no siempre es posible satisfacer todos los atributos de calidad, es por esto que se tiene que priorizar.

Atributo	Patrón arquitectónico	Interpretación en el sistema
Usabilidad	———	En este caso se delega no a un patrón arquitectónico, sino que más bien a un mecanismo que haga amigable la interacción con el usuario, que llevo a un mecanismo asíncrono para las peticiones del usuario al sistema.
Escalabilidad	Broker	Se busca que haya una capa que funja como un Broker hacia los clusters y permita tener desacoplados los recursos, en este caso la capa de servicios es el Broker que permite tener desacoplados los cluster de Cloud-DLML, ya que cada servicio direcciona al cluster solicitado sin que tengan que ser dedicados.
Interoperabilidad	———	Que haya un servicio por cada aplicación y que otros sistemas clientes puedan hacer uso de estos, por eso parte de la implementación se hizo con tecnología estándar de WS (Web Services), siguiendo una arquitectura SOA.
Adaptabilidad	Layers	Encapsular de forma adecuada el sistema en diferentes capas donde los módulos de cada una de ellas vayan de acuerdo a la abstracción, con esto se busca que haya diferentes capas en Cloud-DLML, para facilitar las modificaciones de cada una de ellas sin afectar a las otras capas.
Portabilidad	———	Este atributo se delega al uso de tecnología estándar, buscando con esto independencia de alguna plataforma específica.

Tabla 4.8: Aplicación de patrones y mecanismos arquitectónicos para el diseño de Cloud-DLML

Al terminar el método anterior basado en 4 pasos del QAW/ADD, que fue la guía para tomar decisiones arquitectónicas (Tabla 4.8) y teniendo en cuenta arquitecturas de diferentes nubes, al final obtuvimos como resultado una arquitectura (Figura 4.3) basada en SOA implementando Web Services, y anexando una capa de HPC, de esta manera el sistema Cloud-DLML ofrece servicios de HPCaaS buscando una calidad aceptable y conveniente

para el crecimiento de éste.

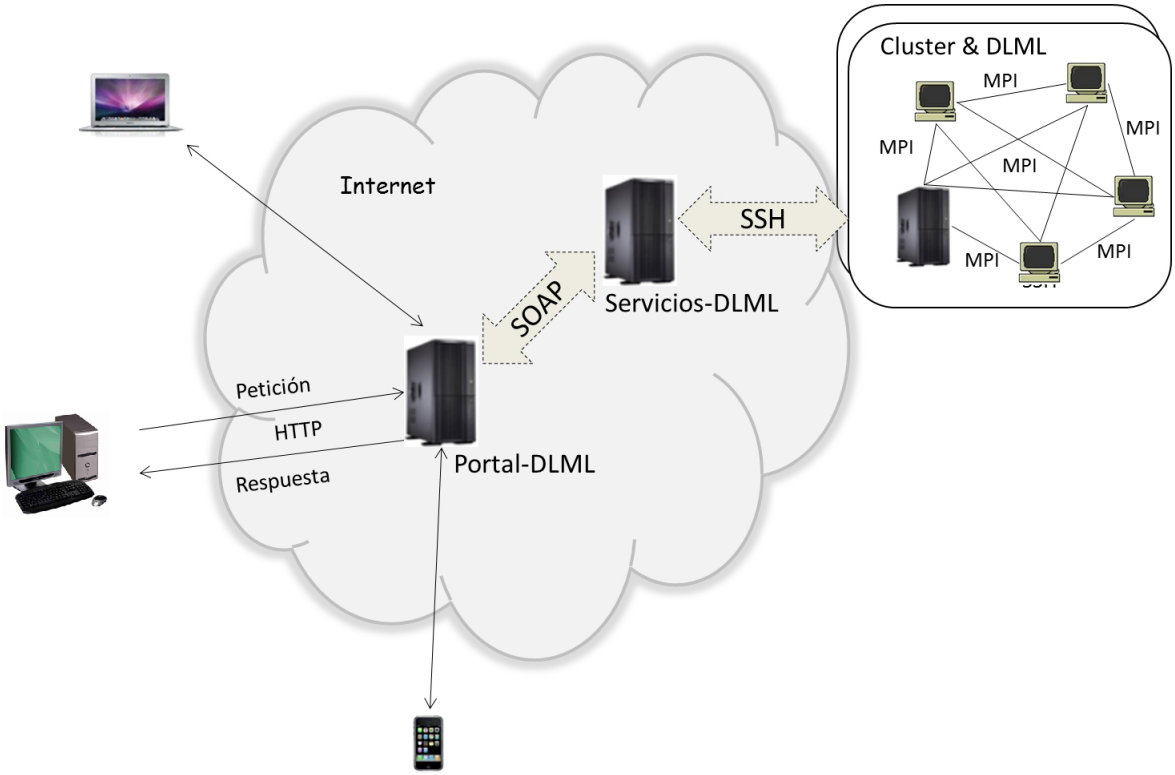

Figura 4.3: Arquitectura de Cloud-DLML

4.2. Arquitectura

La arquitectura de Cloud-DLML (Figura 4.3) se divide en tres capas, éstas son el Portal-DLML que es el que interactúa directamente con el usuario y desde el punto de vista de SOA es el consumidor de los servicios, luego está la capa de los Servicios-DLML que es el nivel donde se encuentran alojados los servicios y por último se encuentra la tercera capa Aplicaciones-DLML donde se concentra todo el HPC. Esta implementación de la arquitectura de Cloud-DLML permite encapsular toda la parte de HPC y con esto el usuario sólo por medio de una conexión a Internet pueda hacer uso de los servicios HPCaaS.

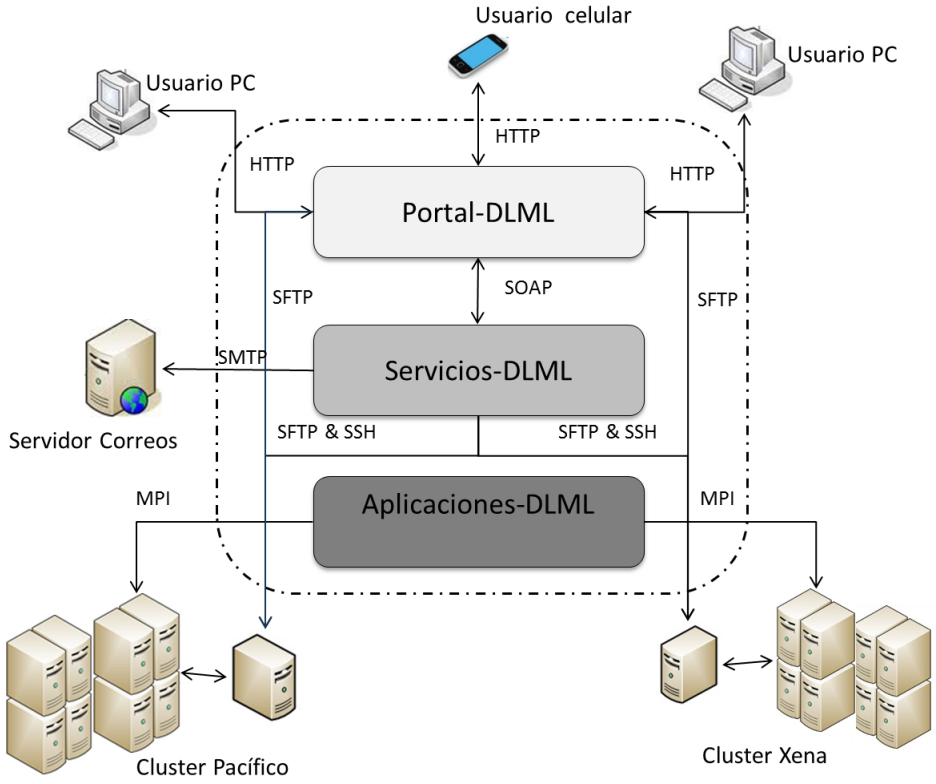

Figura 4.4: Capas de Cloud-DLML

Cada una de estas capas tiene diferentes funcionalidades (Figura 4.4), que se complementan entre sí para hacer posible la ejecución de los servicios HPCaaS (Anexo C).

4.2.1. Capa de Portal-DLML

Esta capa es la que interactúa con el usuario directamente, desde el punto de vista de SOA el Portal-DLML juega el roll del consumidor de los servicios DLML, este recibe las peticiones por parte del usuario para posteriormente mandar a ejecutar los servicios DLML. Para llevar a cabo la implementación del Portal-DLML se tuvo en cuenta un atributo de calidad desde el punto de vista del usuario y este es la usabilidad, es por esta razón que gran parte del desarrollo de las interfaces web se implementó con tecnologías RIA (Rich Internet Applications), es decir, se usaron tecnologías para hacer posible que el Portal-DLML sea una aplicación de Internet enriquecida (Figura 4.5). Las herramientas que destacan en el desarrollo del Portal-DLML son:

- Ajax [36] que permite que el Portal-DLML siga un modelo asíncrono, con esto el usuario no tiene que esperar a que termine la ejecución del servicio para seguir navegando dentro del portal.

- DHTML(Dynamic HTML), por medio de la librería DTHMLGoodies [37] que permite la implementación de portales web sumamente dinámicos y vistosos, esto hace posible un dinamismo agradable entre las interfaces web del Portal-DLML.

- JS(Java Script), por medio de este lenguaje hacemos implementación de validaciones con el fin de facilitar el uso del Portal y así evitar posible errores de los usuarios.

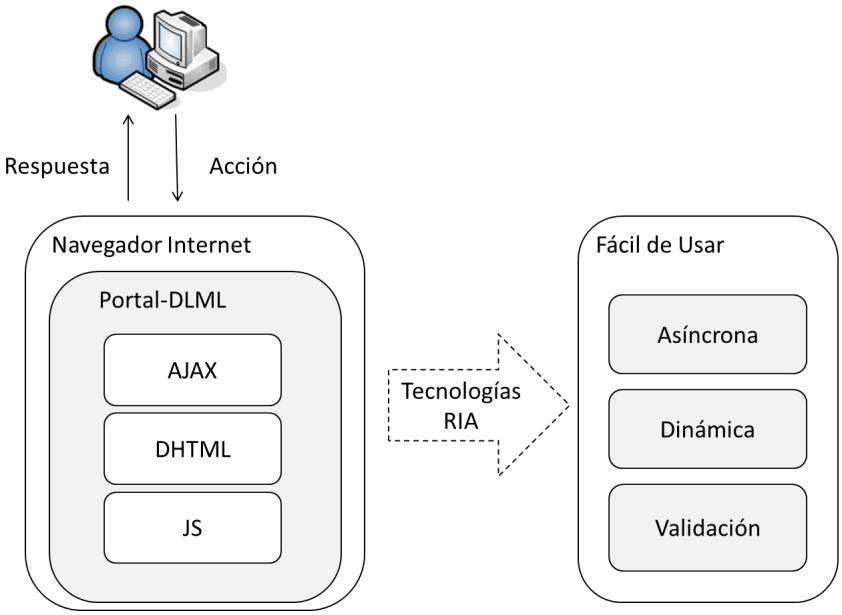

Figura 4.5: Interacción del usuario con el Portal-DLML

Las tecnologías (Ajax, DHTML y JS) usadas para la implementación, son tecnologías estándar que puede ser ejecutada sin problemas en la mayoría de los navegadores web, es por esta razón que el Portal-DLML se puede ejecutar desde varias plataformas (Windows, Linux, Mac OS, iOS y Android).
En cuanto a la construcción lógica del Portal-DLML, ésta se ha basado en el patrón de diseño Factory, este es el patrón de diseño más usado d grupo de patrones de creación [38], el patrón Factory funciona como su nombre lo indica en una fábrica de productos y consiste en tres módulos (Cliente, Fábrica y Producto). En la capa del Portal-DLML se utilizó el patrón Factory para implementar una *Fábrica de Consumidores* (Figura 4.6), es decir, para cada servicio expuesto en Cloud-DLML se crea un consumidor de un servicio, en este caso sólo hay tres posibles consumidores que la *Fábrica de Consumidores* puede crear ya que sólo se cuenta con los servicios N-Reinas, Multiplicación de Matrices y Suma de Matrices.

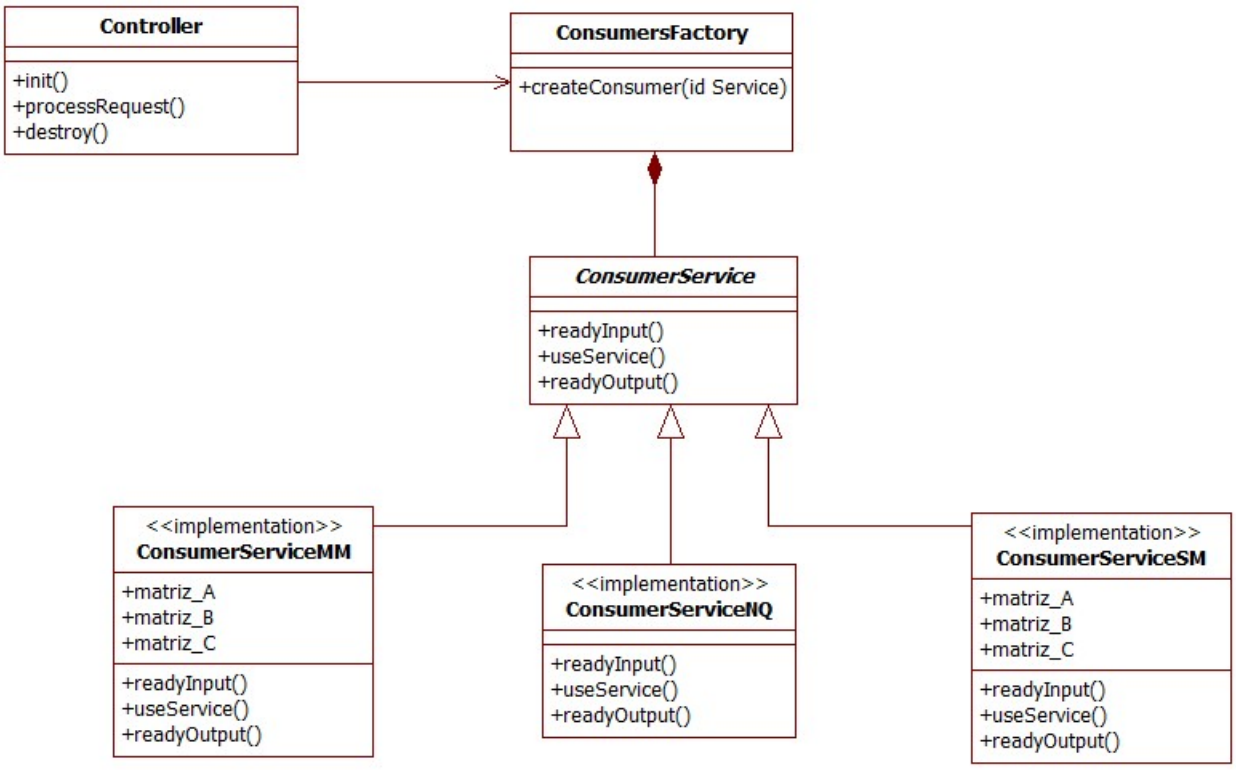

Figura 4.6: Estructura de la Fábrica de Consumidores

Para entender mejor como funciona esta *Fábrica de Consumidores* en el Portal-DLML, describimos cada uno de los tres módulos involucrados (`Contoller`, `ConsumersFactory` y `ConsumerService`), de acuerdo al roll definido por el patrón de diseño Factory:

1. **Controller** (Cliente): este es el Servlet que se encarga de atender las peticiones del usuario. Lógicamente el `Controller` juega el roll de cliente dentro de la *Fábrica de Consumidores*, éste manda a crear el consumidor del servicio a la `ConsumersFactory` de acuerdo a la solicitud del usuario, posteriormente ejecuta los métodos para hacer uso del servicio DLML (Figura 4.7).
   En este caso, para mandar a crear un consumidor el `Controller` nunca cambia su lógica no importa si se agregan más servicios, para éste es transparente la creación de los consumidores para los servicios de Cloud-DLML, quien encapsula la lógica para la creación de los consumidores, es la `ConsumersFactory`.

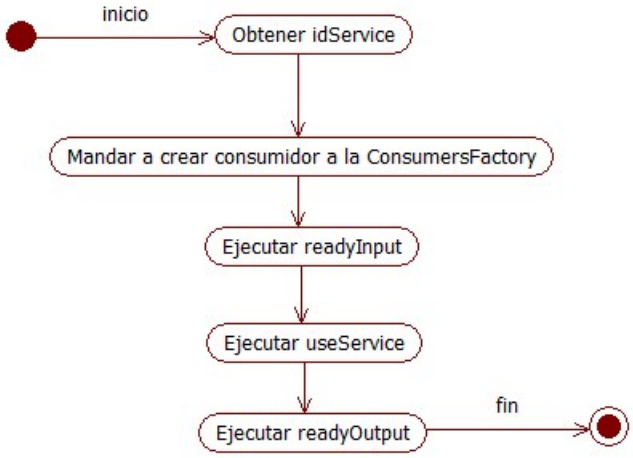

Figura 4.7: Flujo del módulo Controller en la Fábrica de Consumidores

2. **ConsumersFactory** (Fábrica): este módulo se encarga de crear el `Consumidor` adecuado para cada petición de un servicio, por medio del identificador del servicio. El `ConsumersFactory` crea el consumidor correspondiente para el Controller (Cliente), ésta siempre le regresa al `Controller` una especialización de la interface `ConsumerService` según el servicio que se quiera mandar a ejecutar (Figura 4.8).

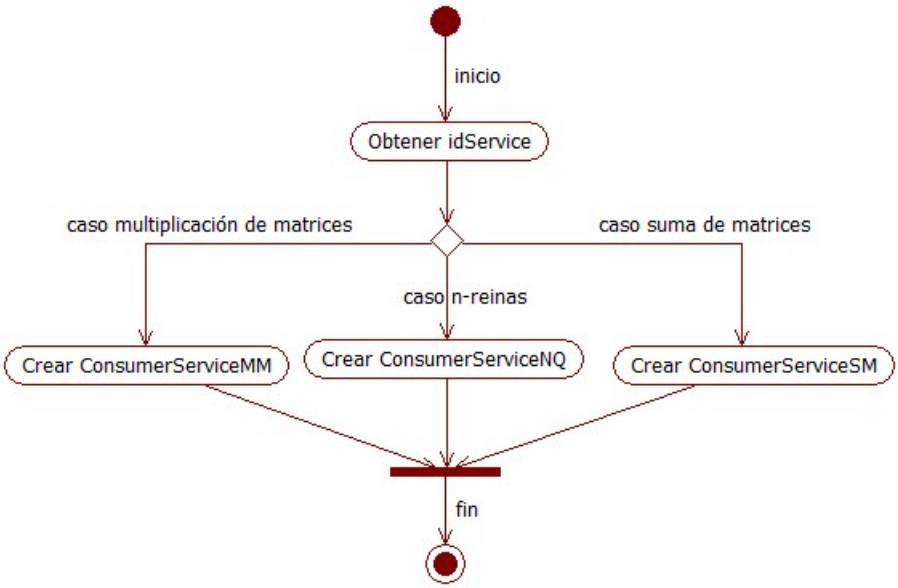

Figura 4.8: Flujo del módulo ConsumersFactory en la Fábrica de Consumidores

Actualmente hay tres opciones de creación, uno por cada servicio. Cada vez que se quiera agregar un consumidor para algún nuevo servicio, se tiene que agregar a las opciones posibles de creación, y asignarle un id al servicio, para que, por medio de éste, pueda ser creado el `ConsumerService` correcto.

3. **ConsumerService** (Producto): Los productos de esta fábrica son los consumidores y éstos son usados para invocar los servicios DLML. Los consumidores están definidos por la interface java `ConsumerService`, la cual tiene tres especializaciones correspondientes a cada consumidor de los servicios de Cloud-DLML, las especializaciones de consumidores son: el `ConsumerServiceNQ` asociado al servicio de las N-Reinas, el `ConsumerServicesMM` asociado al servicio de la Multiplicación de Matrices y por último esta el servicio `ConsumerServiceSM` asociado al servicio de la Suma de Matrices (Figura 4.6). Todos estos consumidores son implementaciones de la interface java `ConsumerService`, que se muestra a continuación:

```
1 public interface ConsumerService {
2
3     /**
4      * this method prepares the input to DLML service.
5      */
6     void readyInput(HttpServletRequest request, HttpServletResponse
          response)
7             throws Exception;
8
9     /**
10     * this method execute the service DLML.
11     */
12    void useService() throws Exception;
13
14    /**
15     * this method prepares the output to user.
16     */
17    void readyOutput(HttpServletRequest request, HttpServletResponse
          response)
18            throws Exception;
19 }
```

El código anterior de la interface `ConsumerService` muestra los métodos que tienen que implementar cada uno de los consumidores (`ConsumerServiceNQ`, `ConsumerServicesMM` y `ConsumerServiceSM`), observamos que hay tres métodos diferentes, en cada implementación la lógica que debe de haber para estos métodos es la siguiente:

- método **readyInput**, en éste se prepara la entrada hacia el servicio a partir de los datos ingresados por el usuario en la interfaz web.
- método **useService**, básicamente lo que se debe implementar en este método es la lógica para localizar y conectar con el servicio por medio del `ProxyGeneric` (Anexo B).

- método **readyOutput**, éste debe tener la lógica para que a partir de la respuesta del servicio, se prepare la respuesta que se le notificará al usuario sobre la interfaz web del Portal-DLML.

Lo que se tiene al final es una *Fábrica de Consumidores* (Figura 4.6), donde de acuerdo a la petición del usuario, se crea el consumidor correspondiente al servicio solicitado por el usuario.

La aplicación de este patrón facilita la creación de consumidores de servicios, tratándolos a todos como subclase de una clase genérica, además esto beneficia al desarrollador para ingresar un nuevo consumidor, sólo debe seguir este patrón y agregar un nuevo consumidor a la fábrica para cada servicio que se agregue, y por medio de este hacer uso del servicio correspondiente. Lo que se obtiene con esta implementación del patrón de diseño Factory es simplificar la lógica de programación siguiendo un patrón altamente probado y que esta lógica sea fácil de modificar.

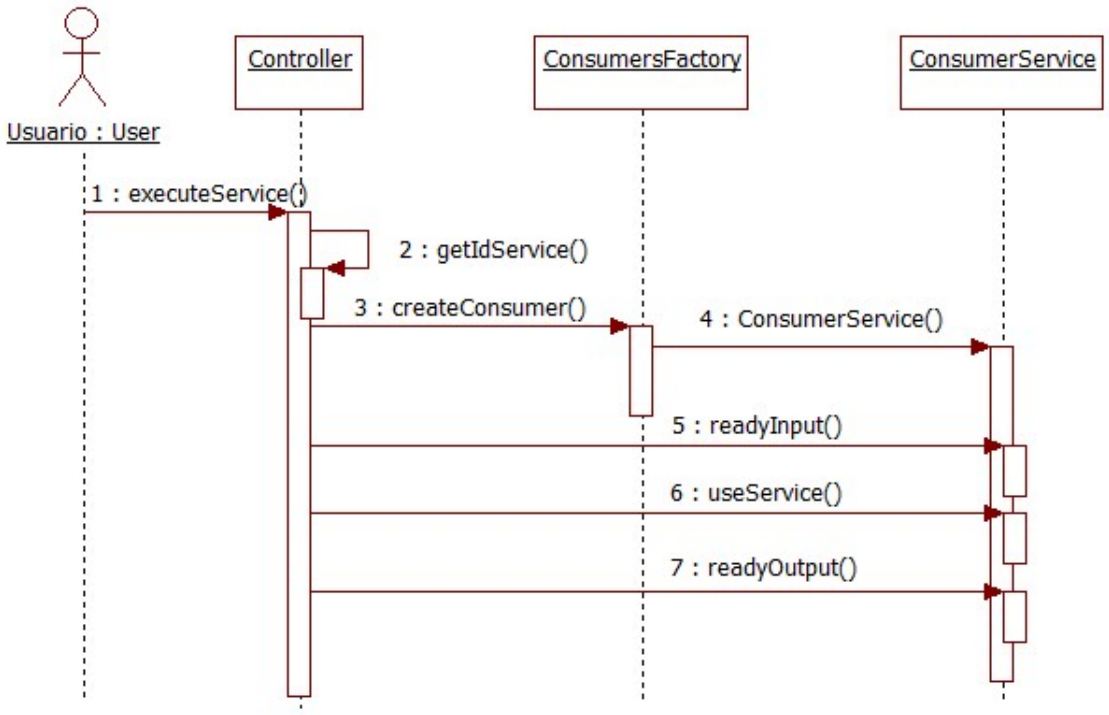

Figura 4.9: Diagrama de secuencia de la Fábrica de Consumidores

De esta manera la programación se estructura, es fácil de entender y escalar, por lo cual la lógica de programación para mandar a ejecutar un servicio a partir de la petición del usuario (Figura 4.9), se simplifica para la capa del Portal-DLML en lo siguiente:

1. Obtener el id del servicio seleccionado por el usuario.

2. Crear el consumidor por medio de la *Fábrica de Consumidores* de acuerdo al id del servicio.

3. El consumidor prepara la entrada al servicio, a partir de los datos ingresados por el usuario.

4. El consumidor manda la petición al servicio.

5. El consumidor recibe la respuesta del servicio y le despliega al usuario en pantalla los resultados.

Con este análisis quedan cubiertos los aspectos importantes de la implementación tanto visuales como lógicos de la capa del Portal-DLML, por una parte lo que se muestra al usuario son las interfaces web, y por otra parte el comportamiento interno que crea los consumidores, para hacer uso de los servicios que se encuentran en la capa de Servicios-DLML.

4.2.2. Capa de Servicios-DLML

En esta capa es donde se encuentran los servicios DLML, esta capa funge como un Broker entre el Portal-DLML y las aplicaciones DLML alojadas en los clusters. La implementación y lógica de la capa Servicios-DLML se basa en dos aspectos importantes: por un lado esta el *WS (Web Service) Genérico* que se implementó usando las librerías de Apache Axis [39] para crear WS, la otra parte fundamental de esta capa es la *Fábrica de Servicios*, que al igual que la *Fábrica de Consumidores* sigue el patrón de diseño Factory.

Por medio del *WS Genérico*, la capa de Servicios-DLML atiende las peticiones provenientes del Portal-DLML, lo importante a revisar son los tipos de entrada `inGeneric` y salida `outGeneric`, ya que estos son los parámetros que se pasan y se obtiene de cada uno de los servicios (N-Reinas, Multiplicación de Matrices y Suma de Matrices) respectivamente, estos contienen la información necesaria para ejecutar las aplicaciones DLML por medio de los datos ingresados por el usuario en el Portal-DLML.
Estos dos grupos de parámetros, tienen un roll importante dentro de la ejecución de los servicios, primero hagamos una revisión de `inGeneric` (Figura 4.10) que encapsula los parámetros de entrada para los servicios:

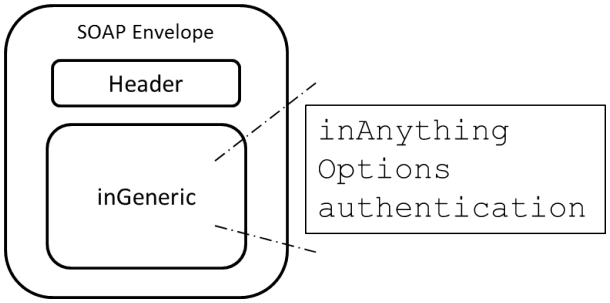

Figura 4.10: Parámetro inGeneric del WS Genérico

- **inAnything**: la idea de este parámetro es exactamente como su nombre lo refleja "cualquier cosa", se busca hacer uso de cualquier dato o estructura de datos, que pueda ser representado con una Struct en lenguaje C o su equivalente en java un objeto de valores VO (Value Object). Para esto en Cloud-DLML se ha implementado un mecanismo de un parser genérico (`ParserGeneric`), que se encarga de aplanar objetos VO's a cadenas XML y viceversa(Figura 4.11), así con este mecanismo hay un polimorfismo de los datos entre el Portal-DLML y Servicios-DLML.

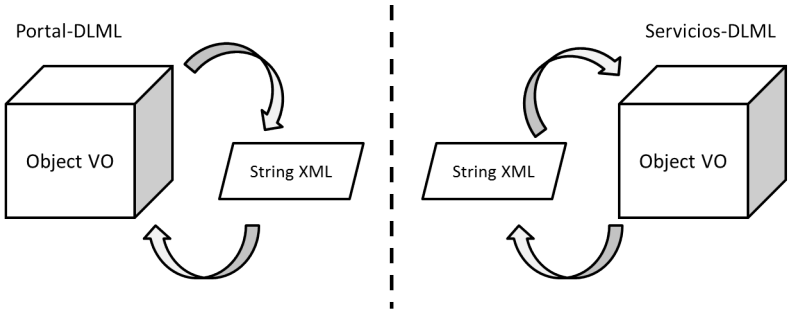

Figura 4.11: Mecanismo de Parser Genérico

Así con la ayuda del `ParserGeneric` el *WS Genérico*, es capaz de hacer uso de cualquier objeto VO en java, que posteriormente puede ser mapeado a un struct en lenguaje C, que usara la aplicación DLML.

- **options**: este parámetro se refiere a las opciones de ejecución que tiene el usuario dentro del Cloud-DLML. En este parámetro se especifica el cluster en el que se va a ejecutar la aplicación, el número de nodos para la ejecución, opcionalmente el correo electrónico al que se le notificará el detalle de la ejecución y cualquier otro parámetro que se requiera.

- **authentication**: este parámetro se pensó para el caso en que se requiera una autentificación con alguna cuenta del usuario, esto al establecer la conexión con los clusters. Actualmente Cloud-DLML tiene una cuenta de conexión para cada cluster, cuando el

usuario manda hacer una ejecución a un cluster no es necesario que tenga una cuenta en dicho cluster, ya que la conexión a los recursos se hace por medio de las cuentas de Cloud-DLML sin importar el usuario que manda a ejecutar un servicio. Es por ende que este parámetro aun no es ocupado, pero podría ocuparse en un futuro o bajo ciertas condiciones.

Una vez revisado con mayor detalle los parámetros de entrada, hagamos una revisión a los parámetros de salida `outAnything` (Figura 4.12), que es lo que regresa cada servicio al finalizar una ejecución, y que contienen los resultados de las aplicaciones DLML:

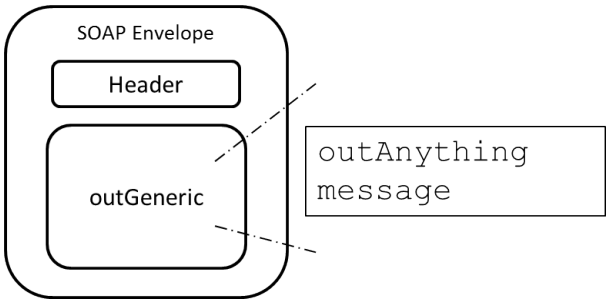

Figura 4.12: Parámetro outGeneric del WS Genérico

- **message**: este parámetro es usado precisamente para mensajes ya sea de error o de algún otro evento durante la ejecución del servicio. Este tiene gran importancia ya que si ocurren anomalías, por medio de este mensaje podríamos saber que paso en la ejecución, o simplemente si se requiere saber algún evento en particular.

- **outAnything**: este tiene la misma idea y contexto que el parámetro de entrada `inAnything`, la idea es poder regresar cualquier cosa desde el servicio hasta la interfaz web del usuario, usando el mismo mecanismo de `ParserGeneric` (Figura 4.11), así de esta manera en este parámetro puede ir desde una matriz hasta un simple número.

Habiendo revisado los parámetros de entrada y salida del *WS Genérico* (encargado de atender y responder las peticiones del Portal-DLML), necesitamos hablar del módulo que esta detrás del *WS Genérico*, al que se le delega la ejecución de los servicios, hablamos de la *Fábrica de Servicios*, este módulo encapsula la creación y ejecución de los servicios (N-Reinas, Multiplicación de Matrices y Suma de Matrices) encargados de ejecutar las aplicaciones DLML correspondientes.

La *Fábrica de Servicios* (Figura 4.13), al igual que la *Fábrica de Consumidores* esta en base al patrón de diseño Factory, sólo que esta fábrica se encarga de crear los servicios que acceden a los diferentes clusters, para ejecutar, monitorizar y obtener los resultados de las aplicaciones DLML.

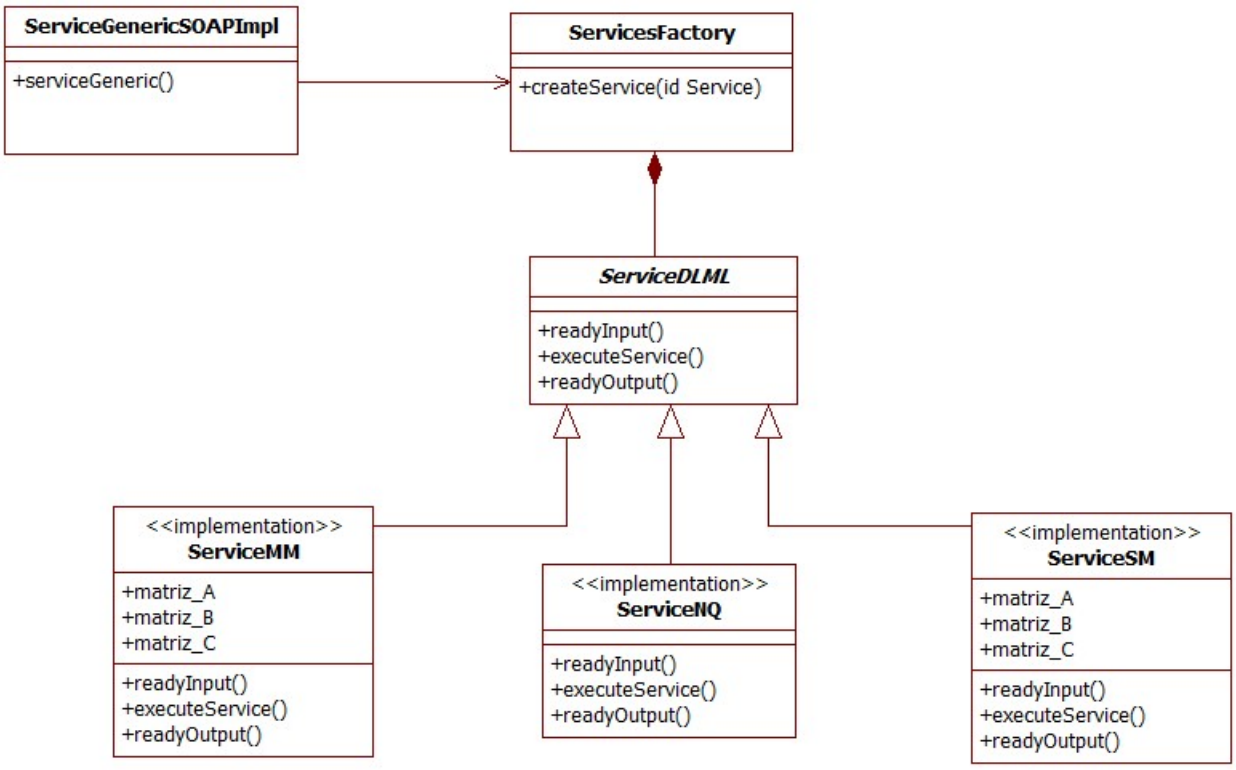

Figura 4.13: Estructura de la Fábrica de Servicios

A continuación se explica como funciona cada uno de los tres componentes (`ServiceGenericSOAPImpl`, `ServicesFactory` y `ServiceDLML`) de esta *Fábrica de Servicios*:

1. **ServiceGenericSOAPImpl** (Cliente): En este caso el cliente de la fábrica es la clase perteneciente al *WS Genérico* que se encarga de mandar a crear los servicios de acuerdo a las peticiones del Portal-DLML. Para el *WS Genérico* es transparente la creación de los servicios, éste sólo manda a llamar a la *Fábrica de Servicios* para obtener el servicio adecuado y así poderlo ejecutar. Al terminar la ejecución del servicio, el *WS Genérico* se encarga de regresar el resultado al Portal-DLML (Figura 4.14).
   Cada vez que se agreguen nuevos servicios, el `ServiceGenericSOAPImpl` no cambia su comportamiento, siempre ejecutara la misma llamada al `ServicesFactory` delegando a esta la creación de un servicio, para posteriormente ejecutarlo.

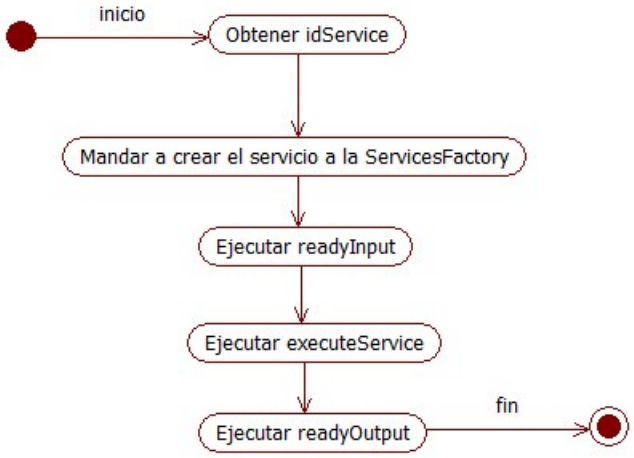

Figura 4.14: Flujo del módulo ServiceGenericSOAPImpl en la Fábrica de Servicios

2. **ServicesFactory** (Fábrica): Ésta se encarga de crear los servicios adecuados para cada petición del cliente. Esta clase de acuerdo al identificador del servicio crea el servicio correspondiente (Figura 4.15), actualmente sólo tiene tres opciones de creación, una por cada servicio de Cloud-DLML.

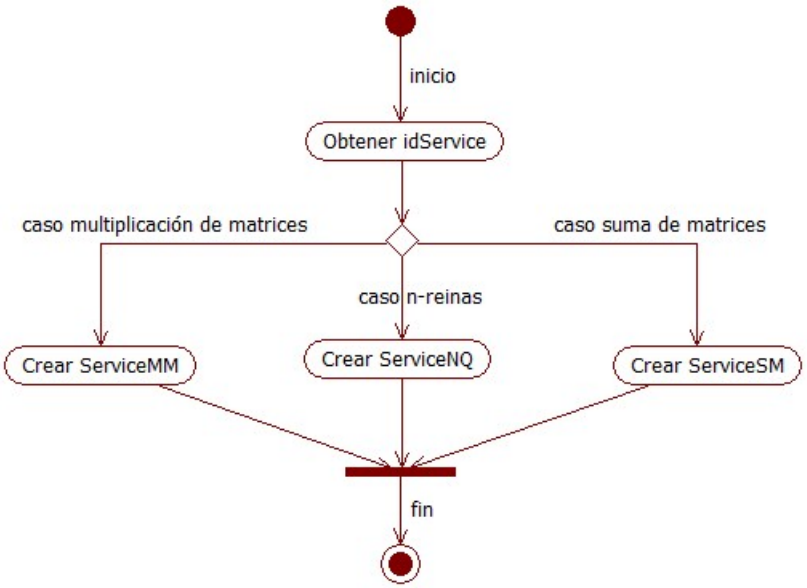

Figura 4.15: Flujo del módulo ServicesFactory en la Fábrica de Servicios

Cada vez que se quiera agregar un nuevo servicio se tiene que agregar a las opciones de posibles productos (*ServiceDLML's*) de la fábrica y agregar la especialización correspondiente del `ServiceDLML`.

3. **ServiceDLML** (Producto): Los productos de esta fábrica son los servicios, que son los que acceden a los clusters para ejecutar las aplicaciones DLML. Actualmente hay tres servicios (Productos): el `ServiceNQ` para el problema de las N-Reinas, `ServiceMM` para la Multiplicación de Matrices y `ServiceSM` para la Suma de Matrices. Todos los servicios desprenden de la clase genérica `ServiceDLML`, que se muestra a continuación:

```
1 public interface ServiceDLML {
2
3       /**
4        * this method prepares the input to DLML application.
5        */
6   void readyInput(String inAnything, String[] option) throws Exception;
7
8       /**
9        * this method execute the DLML application on cluster.
10       */
11  void excuteService() throws  Exception;
12
13      /**
14       * this method prepares the output to Portal-DLML.
15       */
16  void readyOutput(StringHolder outAnything) throws Exception;
17 }
```

Los métodos que tienen que ser implementados por los servicios son tres, que por el nombre de cada método se puede intuir la lógica que debe contener cada implementación de estos:

- método **readyInput**, en éste se tiene que preparar la entrada para la aplicación alojada en el cluster a partir de los datos provenientes del Portal-DLML. La entrada puede abarcar desde parámetros hasta archivos, por ejemplo en el caso de la aplicación N-Reinas sólo basta con hacer llegar los parámetros del número de reinas y número de nodos, por otra parte en el caso de la suma o Multiplicación de Matrices requieren de archivos de entrada que contiene las matrices A y B además del parámetro para el número de nodos.
- método **executeService**, se implementa la lógica para hacer la conexión hacia los cluster por medio del protocolo SSH y mantener esta conexión durante la ejecución de la aplicación.
- método **readyOutput**, en el cual se implementa la lógica para obtener los resultados de la ejecución de la aplicación, para este caso a diferencia de la entrada siempre implica archivos con los resultados de salida, es por esto que siempre se tiene que obtener archivos de salida por medio de conexiones SFTP hacia los

clusters. Posteriormente cada uno de los servicios (`ServiceNQ`, `ServiceMM` y `ServiceSM`) tienen que preparar la salida a partir de estos archivos hacia el Portal-DLML.

Como se puede ver el patrón de diseño Factory implementado en esta capa, sigue la misma idea que en el caso de la capa del Portal-DLML, sólo que en esta capa se implementa una *Fábrica de Servicios* (Figura 4.13) y de esta forma complementar el uso *WS genérico*, que por cada petición de un servicio, por medio del identificador delega la creación del servicio correspondiente a la *Fábrica de Servicios*.

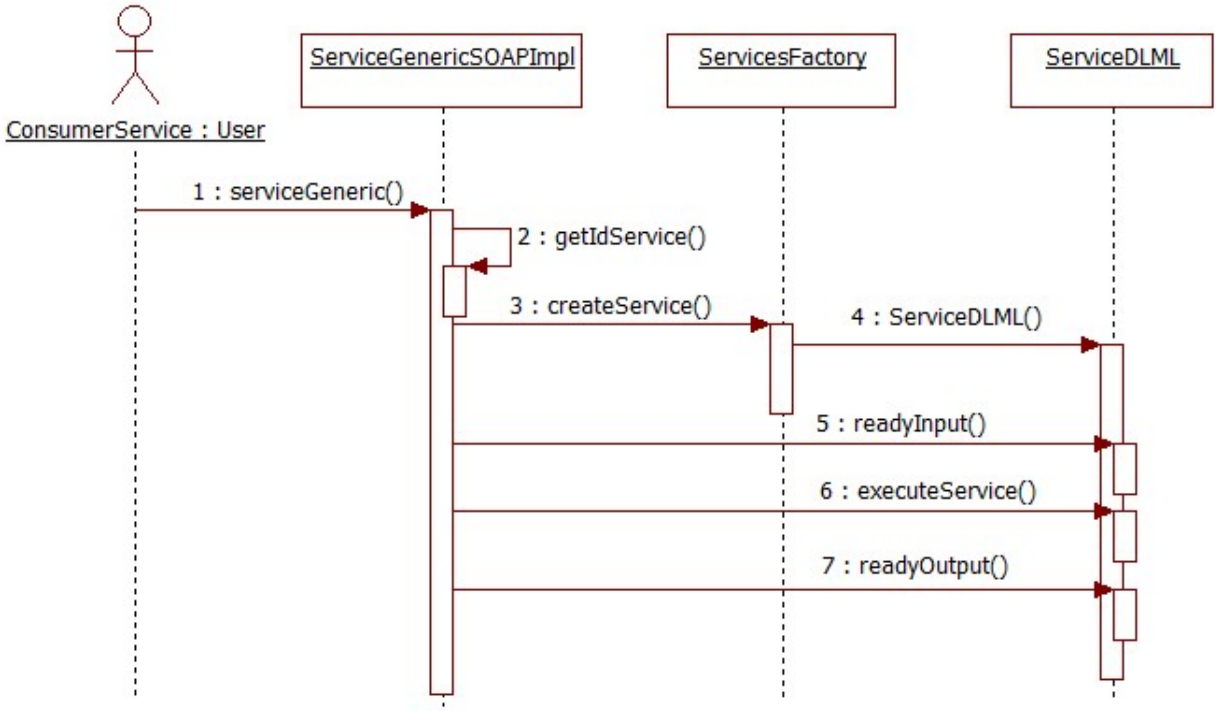

Figura 4.16: Diagrama de secuencia de la Fábrica de Servicios

La *Fábrica de Servicios* hace que la lógica de programación (Figura 4.16) para la creación y ejecución de un servicio sea fácil de entender, por tal motivo la lógica para crear los servicios que mandan a ejecutar las aplicaciones DLML a partir de las peticiones del Portal-DLML se simplifica ya que la capa de Servicios-DLML sólo se hace lo siguiente para ejecutar una aplicación DLML:

1. Obtener el id del servicio solicitado por el consumidor.

2. Crear el servicio por medio de la *Fábrica de Servicios* de acuerdo al id del servicio.

3. El servicio prepara la entrada para la aplicación DLML, a partir de los datos ingresados por el consumidor.

4. El servicio ejecuta y monitorea la aplicación DLML.

5. El servicio obtiene el resultado de la aplicación DLML y la manda al consumidor.

Por el algoritmo anterior, se observa que gracias al *WS Genérico* y la *Fábrica de servicos* la lógica para la ejecución de los servicios se facilita mucho, en este caso la lógica compleja se encuentra en los métodos `readyInput`, `executeService` y `readyOutput`, que implementan cada una de las especializaciones de la interface `ServiceDLML`, los cuales hacen uso de las funciones núcleo de Cloud-DLML (Anexo B).

Una vez analizadas las dos capas superiores Portal-DLML y Servicios-DLML que son los limites del patrón arquitectónico SOA en Cloud-DLML, sólo queda por analizar la capa más baja del sistema, la capa de Aplicaciones-DLML que es la que concentra todo el HPC de esta nube.

4.2.3. Capa de Aplicaciones-DLML

Esta última capa esta conformada por clusters y las aplicaciones DLML, que son el hardware y software correspondiente para el HPC en la nube Cloud-DLML. Para que esta capa funcione correctamente, lo que se necesita es una cuenta SSH en el cluster para Cloud-DLML y alojar las aplicaciones DLML sobre dicho cluster. Hasta ahora Cloud-DLML sólo cuenta con dos clusters, pero se podrían agregar más, ya que Cloud-DLML cuenta con módulos (Anexo B), que le permiten acceder, ejecutar y controlar, hasta cierto punto los clusters que alojan las aplicaciones DLML.

Por la parte de aplicaciones DLML, hay tres para ofrecer como servicios: Multiplicación de Matrices, Suma de Matrices y el problema de las N-Reinas. Recordando un poco DLML es una librería que facilita la programación paralela hasta cierto punto (Capítulo 2). A continuación se describen a un alto nivel de abstracción cada una de las aplicaciones DLML:

Suma de Matrices: esta aplicación necesita de archivos de entrada y salida, el algoritmo es muy simple, en este caso el algoritmo se programó casi secuencial usando las librerías DLML, para esta aplicación cada elemento de la lista DLML se le inserta una fila de la matriz A y B al igual que el índice de la fila (Figura 4.17).

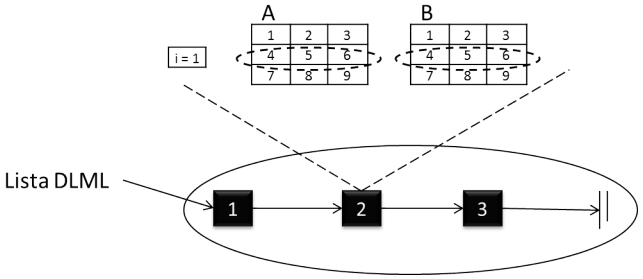

Figura 4.17: Inicialización de la lista DLML para la aplicación Suma de Matrices

Posteriormente se implementó el algoritmo para procesar cada elemento de la lista, es importante decir que DLML se encarga de la distribución y balance de carga de los elementos de la lista entre los nodos [12], a continuación se muestra el algoritmo:

```
Mientras longitud de la lista L > 0
{
   Obtener elemento e de la lista L
   Eliminar elemento e de la lista L
   Obtener la fila a del elemento e
   Obtener la fila b del elemento e
   Obtener índice i del elemento e
   Crea fila c
   Para cada elemento de las filas
     {
       c[k] = a[k]+b[k]
     }
   Escribe la fila c e índice i en archivo
}
Finaliza la aplicación
```

Una vez que todos los elementos de la lista han sido procesados por los nodos, aplicando el algoritmo para obtener las filas de la matriz resultante C, entonces los nodos mandan los resultados a un archivo de salida con el número de fila correspondiente. Lo anterior es necesario para conservar un orden de los resultados ya que los resultados se obtienen de forma concurrente por los nodos, y esto ocasiona que los resultados estén desordenados. El archivo de salida resultante es tomado y procesado por el servicio Suma de Matrices para enviárselo al usuario.

Multiplicación de Matrices: esta aplicación es muy parecida a la Suma de Matrices, en esta se cargan las matrices desde archivos y se inicializa la lista, cada elemento de la lista contiene la fila i de la matriz A, la columna j de la matriz B y los índices i y j (Figura 4.18), estos para saber la posición correspondiente a cada resultado.

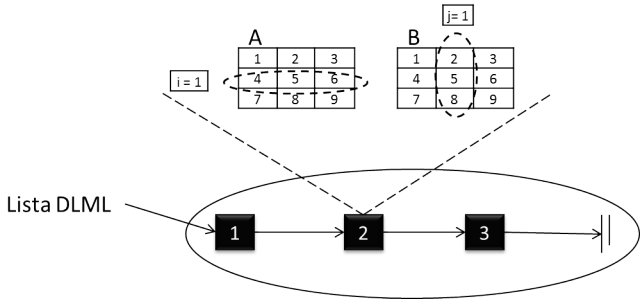

Figura 4.18: Inicialización de lista DLML para la aplicación Multiplicación de Matrices

Una vez inicializada la lista, el algoritmo que se usa para procesar toma cada uno de los elemento de la fila i de la matriz A y lo multiplica por cada uno de la columna j de la matriz B, obteniendo así el elemento c_{ij} de la matriz C, al igual que la Suma de Matrices el algoritmo es casi secuencial:

```
Mientras longitud de la lista L > 0
{
   Obtener elemento e de la lista L
   Eliminar elemento e de la lista L
   Obtener la fila a del elemento e
   Obtener la fila b del elemento e
   Obtener índice i del elemento e
   Obtener índice j del elemento e
   Crear  total_c
   Para cada elemento  de la fila a
   {
      total_c = total_c + a[k]*b[k]
   }
   Escribe el total_c  y las coordenadas i, j en el archivo de salida.
}
Finaliza la aplicación
```

Con este algoritmo cada vez que se procesa un elemento de la lista se obtiene un elemento de la matriz resultante C, el cual se guarda en un archivo de salida junto con las coordenadas dentro de la matriz C. Por último al terminar de procesar toda la lista, se termina la aplicación obteniendo un archivo de salida con los elementos de la matriz C, con sus respectivas coordenadas dentro de esta. Al terminar la aplicación el servicio Multiplicación de Matrices toma el archivo de salida y lo manda al usuario.

N-Reinas: Este es un problema muy conocido y con muchas aplicaciones como control de tráfico aéreo, ruteo de mensajes, entre otros. El problema de las N-Reinas o NAQ (no ataque entre reinas) consiste en colocar N-Reinas en un tablero de `n x n` sin que se ataquen entre sí, este es un problema dinámico a diferencia de los anteriores que son estáticos, es decir, los elementos de la lista se fijan al inicio y no aumenta sólo disminuyen conforme va siendo procesada la lista. Para este caso la lista crece dinámicamente conforme se van generando posibles soluciones, en este problema una solución es un tablero lleno con n-reinas de tal forma que no se coman entre sí, en este caso cada elemento de la lista contiene una posible solución a explorar, conformada por el número de fila que es el número de reina y un vector de tamaño n que contiene las posiciones de las reinas colocadas hasta dicho instante (Figura 4.19).

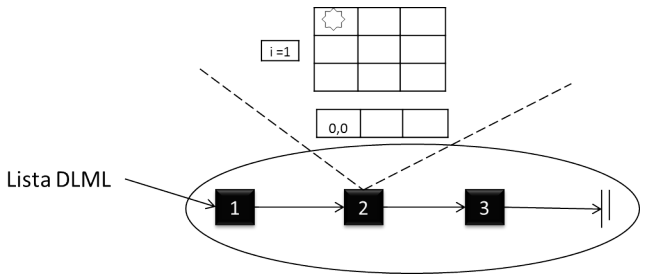

Figura 4.19: Inicialización de la lista DLML para la aplicación N-Reinas

Una vez que se inicializa con alguna configuración inicial para el tablero, los elementos de la lista son procesados siguiendo el algoritmo de las N-Reinas, a continuación se muestra el algoritmo:

```
num_sol =  0
Mientras longitud de la lista L > 0
{
   Obtener elemento e de la lista L
   Eliminar elemento e de la lista L
   Obtener el número de reina del elemento e
   Obtener el vector con las soluciones previas del elemento e
   Para cada columna del tablero
   {
     Verifica (no es_comida)
      {
         Agrega elemento e a la lista L
      }
   }
   Verifica (si número reina = n )
   {
```

```
      num_sol = num_sol +1
   }
}
Escribe num_sol en el archivo de salida
Finaliza la aplicación
```

Al final de la ejecución, la aplicación escribirá en un archivo de salida el número de soluciones para la n-reinas en el tablero `n x n`. Como se puede ver el algoritmo de las N-Reinas también es casi secuencial esto es por la librería DLML, que se encarga de asignar los elementos de la lista a los diferentes nodos. Al finalizar la aplicación el servicio N-Reinas toma el archivo de salida y lee el resultado de este, para notificar al usuario el resultado.

Como era de esperarse la programación para el desarrollo de las aplicaciones paralelas se facilitó en cierta forma, gracias a que fueron implementadas con DLML, en un inicio se contaba con un desarrollo parcial de la aplicación N-Reinas la cual se tuvo que adecuar para que tuviera entrada y salida por medio de archivos (I/O File), posteriormente las otras dos aplicaciones fueron más fáciles de implementar una vez que se estaba familiarizado con la herramienta DLML. Observamos que las aplicaciones pueden recibir las entradas desde parámetros o por medio de archivos, y al finalizar las aplicaciones DLML mandan la salida a un archivo que posteriormente es tomado por el servicio en cuestión, para notificar los resultados al Portal-DLML y enviar un correo electrónico al usuario. También es importante decir que los clusters utilizados pueden estar en lugares geográficos diferentes y no tienen que estar dedicados a Cloud-DLML, en este momento sólo se tienen dos clusters "Pacífico" y "Xena" los cuales tienen su propia administración.

Al finalizar este análisis que hicimos de cada una de las capas, se puede observar que las capas de la arquitectura de Cloud-DLML se encuentran débilmente acopladas, permitiendo que Cloud-DLML no este sujeto a una infraestructura en particular, y se puede observar que cada capa tiene funcionalidades diferentes, pero que se complementan entre sí para ofrecer servicios HPCaaS. Con esta arquitectura débilmente acoplada las capas de Cloud-DLML pueden estar en distintos lugares geográficos sin que afecte la funcionalidad del sistema.
Hasta este punto se ha explicado la funcionalidad de cada capa y como se han implementado, otro aspecto importante son los protocolos de comunicación que hay entre ellas, en la siguiente sección se muestra con mayor detalle los tipos de comunicación entre las capas de Cloud-DLML.

4.3. Comunicación entre capas

La arquitectura de Cloud-DLML esta compuesta por tres principales capas (Portal-DLML, Servicios-DLML y Aplicaciones-DLML) las cuales ya se han revisado. No obstante para comunicar estas capas fue necesario hacer uso de diferentes protocolos y mecanismos de comunicación entre éstas. Así en el flujo de una ejecución de algún servicio de Cloud-DLML,

se hace uso de diferentes mecanismos y protocolos para llevar a cabo la interacción entre las capas de Cloud-DLML.

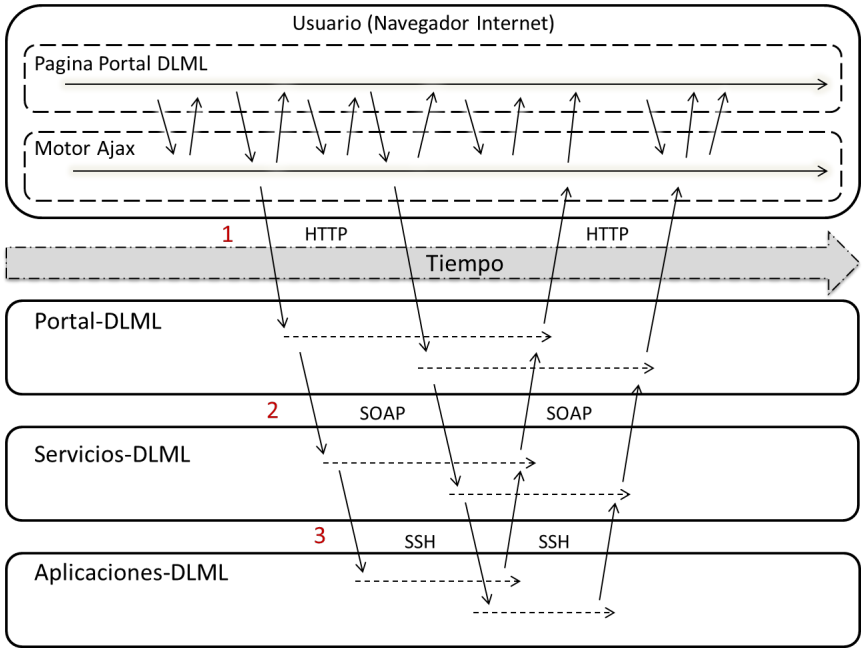

Figura 4.20: Flujo de ejecución de Cloud-DLML

En la figura anterior, se muestra el flujo de ejecución de Cloud-DLML (Figura 4.20), cuando el usuario manda a ejecutar un servicio, en éste flujo se muestra como interactúan las diferentes capas de Cloud-DLML en la línea del tiempo, así como los protocolos que intervienen en cada una de las interacciones entre las diferente capas, estos protocolos son:

1. HTTP, éste se encarga de comunicar el Navegador con el Portal-DLML.

2. SOAP, es el protocolo usado para comunicar la capa de Portal-DLML con la capa de Servicios-DLML.

3. SSH, éste protocolo se usa para establecer la comunicación entre la capa de Servicios-DLML con la capa de Aplicaciones-DLML.

Por otra parte, en la ejecución de un servicio (Figura 4.20), hay que resaltar dos aspectos importantes:

- El usuario al ejecutar un servicio en todo momento esta interactuando con el Portal-DLML, aun sí no ha terminado la ejecución del servicio, esto es posible gracias al Motor de Ajax que hace peticiones asíncronas.

- El tiempo que tarda Cloud-DLML en la ejecución de un servicio, es determinado casi en su totalidad por el tiempo que tarda la aplicación DLML, es decir, el overhead de Cloud-DLML es mínimo.

En lo siguiente explicamos con mayor detalle cada una de las interacciones entre las diferente capas de Cloud-DLML, en la ejecución de los HPCaaS.

Navegador al Portal-DLML: esta comunicación se hace por medio de HTTP, ésta se lleva a cabo cuando un usuario manda a ejecutar un servicio o simplemente se encuentra navegando en el Portal-DLML. Esta es la comunicación más simple, pero lo que hay que resaltar son las peticiones y respuestas que se hacen usando un `XMLRequest` por medio de Ajax, permitiendo peticiones asíncronas (Figura 4.21), así el usuario puede seguir navegando dentro del Portal-DLML, sin que este se bloquee o que tenga que esperar a que termine la ejecución del servicio. En cuanto termina la ejecución del servicio, el Portal-DLML notifica al usuario que ha terminado la ejecución y despliega los resultados en pantalla.

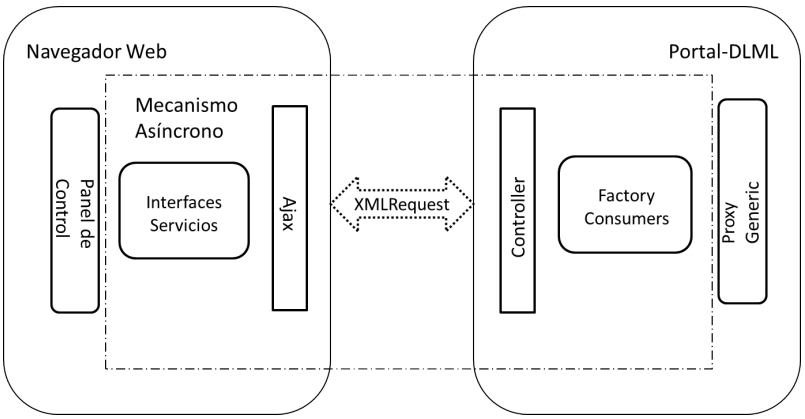

Figura 4.21: Comunicación entre el navegador y Portal-DLML

Con este mecanismo asíncrono, sí una aplicación paralela se tarda demasiado en ejecutar, el usuario no pierde el control de la interfaz web.

Portal-DLML a Servicios-DLML: esta comunicación esta determinada por el *WS Genérico*, en este caso el protocolo que interviene para comunicar las dos capas es SOAP (Simple Object Access Protocol). Para esto el Portal-DLML manda un mensajes SOAP que contiene los parámetros requeridos para ejecutar los servicios DLML, de igual forma al terminar la ejecución de un servicio este regresa la respuesta en un mensaje SOAP al Portal-DLML.

Los mecanismos para la comunicación entre estas dos capas son proporcionados por el *WS Genérico* que por medio del motor de Apache Axis (Figura 4.22) establece la conexión entre ambas capas y se manipulan los mensajes SOAP para su envío y recepción.

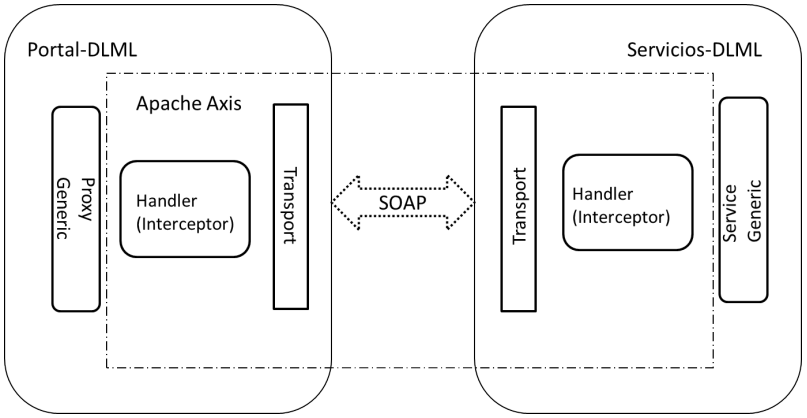

Figura 4.22: Comunicación entre las capas Portal-DLML y Servicios-DLML

Es importante notar que los mensajes SOAP varían en su contenido de acuerdo al servicio que se manda a ejecutar. Como se puede ver la estructura del *WS Genérico*, permite la comunicación entre el Portal-DLML y Servicios-DLML, sin importar el servicio que el usuario haya mandado a ejecutar, de esta forma se simplifica demasiado la comunicación entre ambas capas, ya que reutiliza la misma estructura del mensaje SOAP, cambiando solamente el contenido de acuerdo al servicio solicitado.

Servicios-DLML a las Aplicaciones DLML: esta es la comunicación más compleja dentro del Cloud-DLML, se hace a través de los protocolos SSH (Secure SHell) y SFTP (Secure File Transfer Protocol), ambos son protocolos seguros para establecer una sesión e intercambiar archivos remotamente. Para hacer posible esta comunicación se implementó un mecanismo síncrono, que permite a Cloud-DLML monitorear la conexión entre ambas capas en todo momento y así siempre tener un control de la ejecución de las aplicaciones DLML sobre los clusters. En un flujo de una ejecución de un servicio se lleva a cabo el siguiente proceso de comunicación entre estas dos capas:

1. La capa de Servicios-DLML abre una conexión SFTP en caso de que la aplicación DLML requiera de archivos de entrada.
2. Se establece un túnel SSH entre el servicio y un nodo del cluster (Figura 4.24).
3. Se ejecuta la aplicación DLML la cual es monitorizada por el servicio.
4. Cuando termina la aplicación DLML manda los resultados a un archivo y el servicio cierra la conexión SSH.
5. El servicio DLML toma el archivo de salida a través de una conexión SFTP que posteriormente se cierra.

Para hacer posible este proceso de comunicación entre las capas de Servicio-DLML y Aplicaciones-DLML se necesitaron de muchas pruebas con distintas tecnologías. Al final las

tecnologías usadas fueron Expect y Jcraft; Expect es una librería para sistema Linux la cual nos permite realizar scripts interactivos que esperan respuesta por parte del SO [40], por otra parte Jcraft es una librería Java para conexiones SSH y SFTP [41]. Al complementar estas dos tecnologías se obtuvieron buenos resultados para establecer conexiones entre los servicios y los clusters (Figura 4.23).

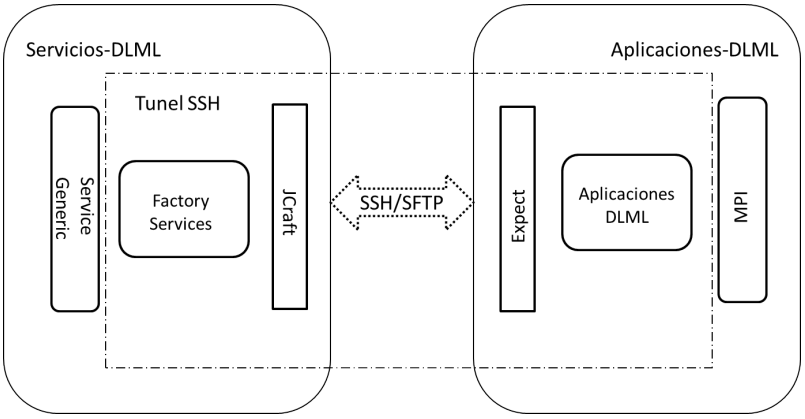

Figura 4.23: Comunicación entre las capas Servicios-DLML y Aplicaciones-DLML

Con Expect y Jcraft trabajando en conjunto, permiten a Cloud-DLML la ejecución de las aplicaciones DLML de una manera interactiva, automatizada y desacoplada en tiempo de ejecución. Por un lado Jcraft se encarga de conectar con el Gateway del cluster y del otro lado Expect se encarga de moverse libremente en los nodos del cluster para ejecutar la aplicación DLML. La implementación de este mecanismo con ambas tecnologías hace posible que Cloud-DLML sea una plataforma multicluster (Figura 4.24).

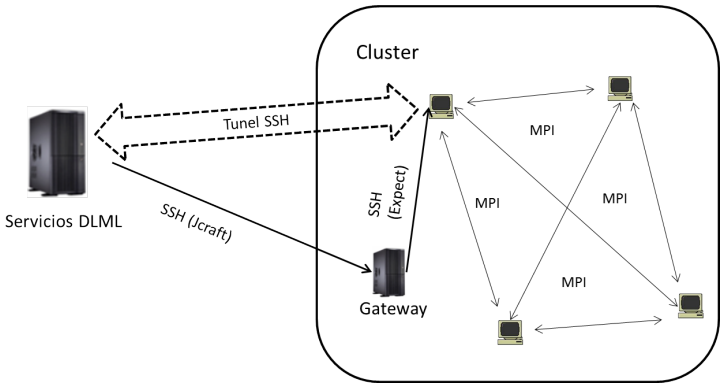

Figura 4.24: Túnel SSH entre el servicio DLML y el cluster

Los puntos anteriores reflejan cómo interactúan entre sí las capas de Cloud-DLML para llevar a cabo la ejecución de los servicios.

Hasta este punto de la tesis se ha mostrado el diseño, la arquitectura y parte del desarrollo del sistema Cloud-DLML, ahora lo que resta de este trabajo es la evaluación y los resultados que se obtuvieron del sistema, para esto se presenta una evaluación de cumplimiento de diferentes atributos de calidad y la manera en que fue puesto en operación Cloud-DLML.

Capítulo 5
Resultados

En resumen se ha visto cómo esta desarrollado el sistema Cloud-DLML, revisamos cada una de sus capas desde las interfaces web hasta las aplicaciones DLML, así como los protocolos de comunicación que existen entre ellas y el flujo de ejecución de los servicios. Se reviso el *WS Genérico* y su complementación con la *Fábrica de Consumidores* y la *Fábrica de Servicios*, que facilitaron el desarrollo de forma elegante, que en conjunto con los mecanismos de comunicación entre las diferentes capas, hacen posible ofrecer las aplicaciones HPC como servicios (Figura 5.1).

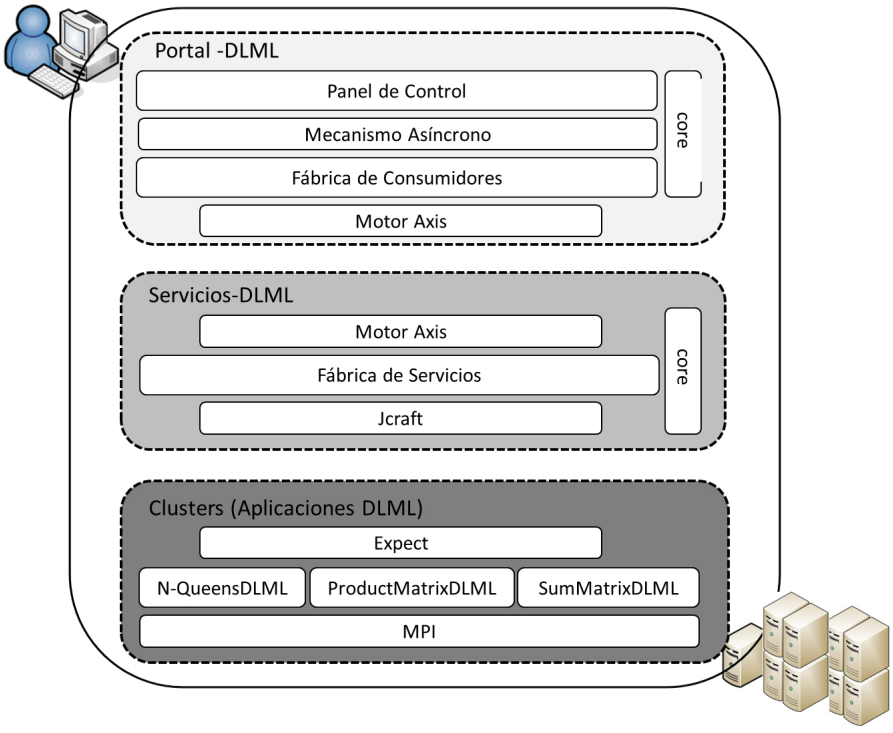

Figura 5.1: Componentes de Cloud-DLML

Al final del desarrollo, se implementaron varios componentes (Figura 5.1) sobre las capas de Cloud-DLML, estos componentes se complementan entre sí, para ofrecer los HPCaaS, a

continuación describimos cada uno de estos, así como la finalidad que tienen cada uno de éstos en Cloud-DLML:

- Panel de Control: este componente es la interfaz web con la que interactúan los usuarios, el "Panel de Control" se encarga de coordinar las tres interfaces web correspondientes a cada uno de los servicios DLML que ofrece Cloud-DLML.

- Mecanismo Asíncrono: éste permite al Portal-DLML ofrecer al usuario llamadas asíncronas, es decir, cada vez que el usuario hace una petición de algún servicio, éste puede seguir navegando sin que tenga que esperar a que termine de ejecutarse dicho servicio, ya que puede haber servicios que tarden inclusive horas.

- Fábrica de Consumidores: este módulo es el encargado de crear los consumidores de los servicios DLML, cada uno de estos consumidores, se encarga de preparar la entrada al servicio, hacer la petición al servicio y preparar la salida hacia el usuario.

- Motor Axis: éste se encuentra en ambas capas tanto Portal-DLML como Servicios-DLML. Del lado del Portal-DLML, el Motor Axis lleva a cabo la creación de los paquetes SOAP, y la conexión con la capa de Servicios-DLML. Por el lado de Servicios-DLML, el Motor Axis se encarga de atender las peticiones desempaquetar los mensajes SOAP, y responder cada una de las peticiones de los servicios.

- Funciones Núcleo (core): este es un módulo compuesto de varías funciones, que tienen como objetivo ejecutar tareas indespensables para la ejecución de los HPCaaS de Cloud-DLML, por ejemplo la carga de los datos de un cluster para la conexión a éste, las conexiones SSH y SFTP, las notificaciones por vía mail, por mencionar algunas de las funciones que se encuentran en este componente.

- Fábrica de Servicios: por medio de éste se crean los servicios que son invocados por el usuario, cada una de las implementaciones de estos servicios se encargan de ejecutar las aplicaciones DLML. Cada vez que se ejecutan los servicios de esta fábrica, éstos siempre tienen el control sobre los clusters, ya que se hacen uso de las funciones núcleo, las cuales algunas hacen uso de la librería Jcraft.

- Jcraft: este componente consiste en una librería de funciones que le premite a los servicios DLML establecer conexiones SSH y SFTP con los clusters, para la ejecución de las aplicaciones DLML, Jcraft trabaja en conjunto con scripts de Expect, para establecer un túnel SSH entre el servicio DLML y los nodos del cluster.

- Expect: éste como tal es un demonio de linux, que permite a los servicios DLML tener el control dentro de los clusters, por medio de scripts de Expect, así de esta manera es como se interactúa directamente con los nodos de los clusters para poder ejecutar las aplicaciones DLML.

- Aplicaciones DLML (N-QueensDLML, ProductMatrixDLML y SumMatrixDLML): éstas son las tres aplicaciones paralelas actuales que Cloud-DLML ofrece como HPCaaS, éstas se encuentran alojadas en los clusters Pacífico y Xena que fueron en los que se probó la arquitectura de Cloud-DLML.
- MPI: éste aunque no se implementó nada directamente con MPI, es considerado como un componente ya que las aplicaciones DLML se ejecutan sobre MPI.

Como se puede ver cada uno de los componentes involucrados en la ejecución de los HPCaaS, son piezas importantes de cada una de las capas de la arquitectura de Cloud-DLML. Por otra parte, para darnos una idea de cómo es que interactúan cada uno de estos componentes, en la ejecución de un servicio, es importante revisar la ejecución a detalle de un HPCaaS en Cloud-DLML.

5.1. Ejecución de HPCaaS

Ya que la ejecución de los tres servicios (N-Reinas, Multiplicación de Matrices y Suma de Matrices) es sumamente similar, hacemos la descripción de cada uno de los pasos involucrados para ejecutar un servicio de forma general, debido a que el diagrama de secuencia de Cloud-DLML es muy grande, se muestra por cada capa (Portal-DLML, Servicios-DLML y Aplicaciones-DLML) el fragmento correspondiente de dicho diagrama, con la explicación de cada uno de los pasos que se realizan para ejecutar los HPCaaS.

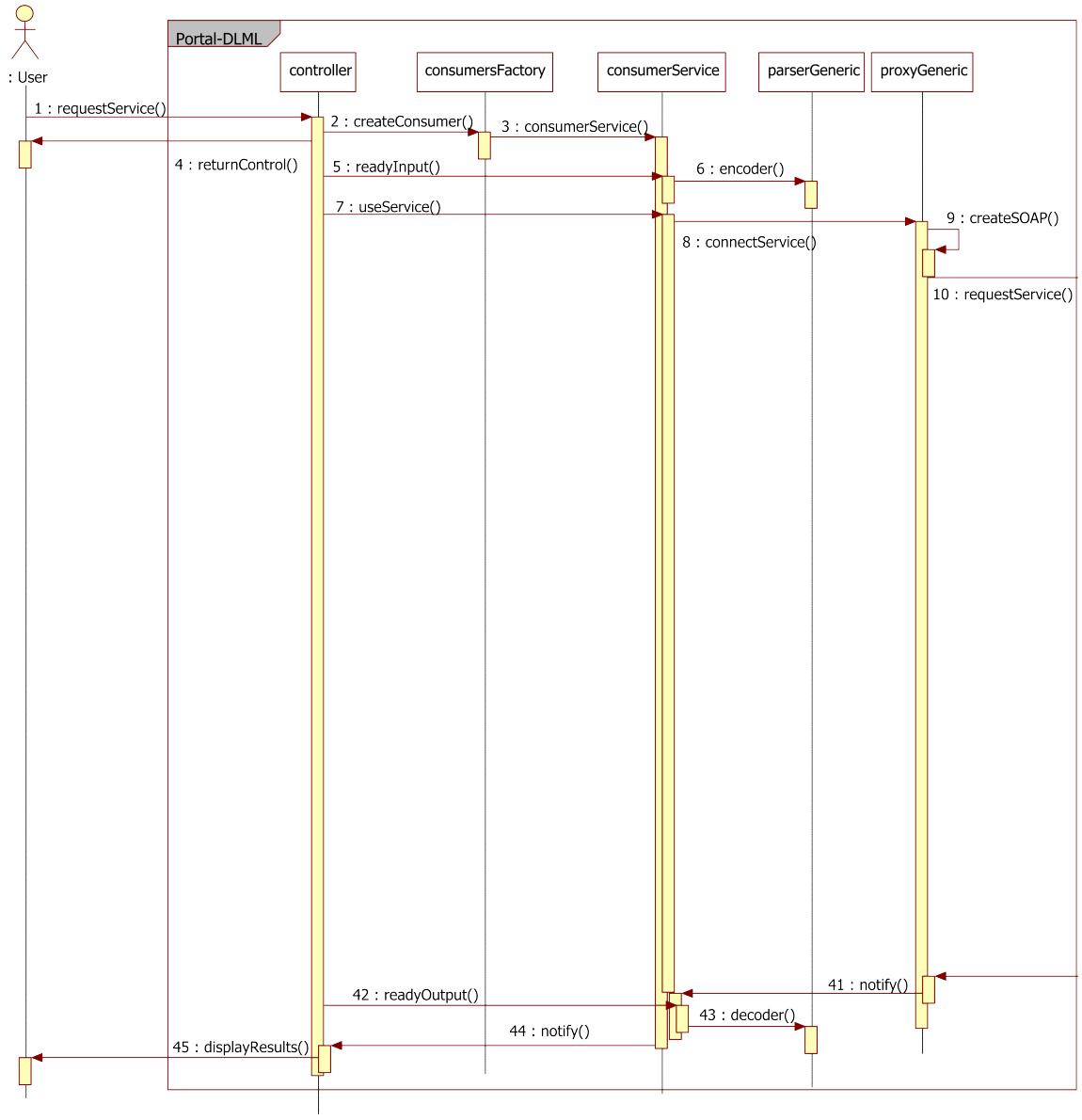

Figura 5.2: Diagrama de secuencia de la capa de Portal-DLML

El diagrama (Figura 5.2), es el fragmento correspondiente al Portal-DLML, a continuación explicamos por bloque de pasos, cada una de las acciones ejecuatadas en esta capa:

- Paso 1, 2, 3 y 4: el usuario manda a invocar un servicio, posteriormente el `Controller` manda a crear el consumidor para dicho servicio a la `ConsumersFactory`, una vez hecho esto el `Controller` regresa el control al usuario, para que pueda seguir navegando mientras es ejecutado el servicio.

- Paso 5 y 6: el `Controller` por medio del `ConsumerService` prepara la entrada del servicio a partir de los datos insertados por el usuario, y encapsula los datos en una cadena XML por medio del `ParserGeneric`.

- Paso 7 y 8: el `Controller` por medio del `ConsumerService` manda la petición del servicio, haciendo uso del `ProxyGeneric` que encapsula funcionalidad del Motor Axis y espera por la respuesta de dicho servicio.

- Paso 41: una vez que el servicio responde al `ProxyGeneric`, éste notifica al `ConsumerService` que ha terminado la ejecución del servicio y desempaqueta el mensaje SOAP por medio del Motor Axis.

- Paso 42 y 43: una vez que el `ConsumerService` tiene la respuesta del servicio, éste prepara la salida que se presentara al usuario, por medio del `ParserGeneric` obtiene el objeto con los datos listos para presentarse al usuario.

- Paso 44 y 45: el ConsumerService notifica al `Controller`, para que éste despliegue en pantalla los resultados al usuario.

Como se puede ver en la capa del Portal-DLML, se muestra como inicia la ejecución de un HPCaaS hasta el despliegue de resultados al usuario, pero falta por ver toda la parte intermedia de la ejecución de un HPCaaS, la cual esta encapsulada en la capa de Servicios-DLML.

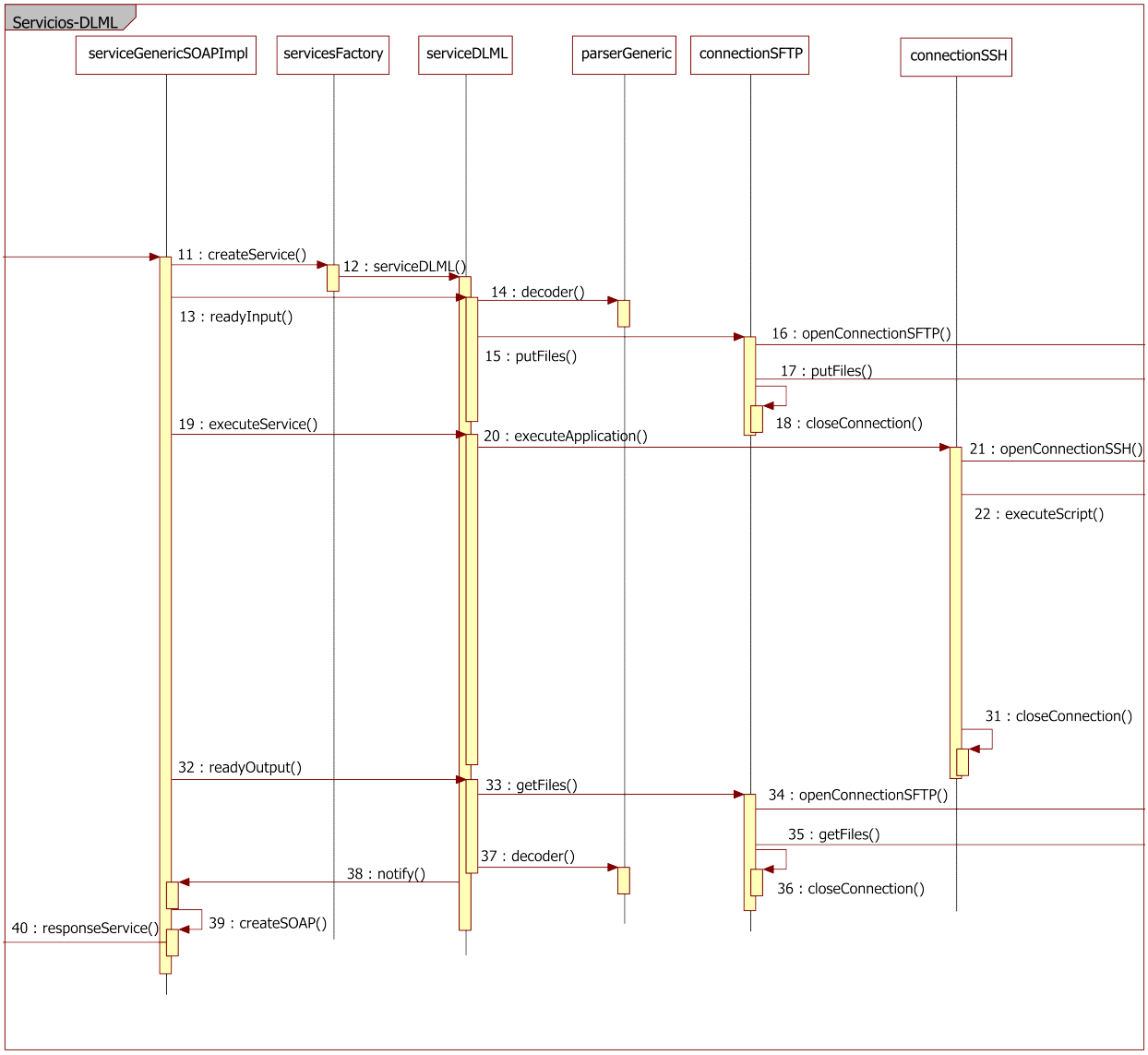

Figura 5.3: Diagrama de secuencia de la capa de Servicios-DLML

En el fragmento del diagrama de secuencia (Figura 5.3), se muestran los pasos involucrados en la capa de Servicios-DLML, que inician cuando el `ConsumerService` manda la petición de un servicio, a continuación hacemos la descripción de las acciones que llevan a cabo estos pasos:

- Paso 11 y 12: el `ServiceGenericSOAPImpl` obtiene la petición del Portal-DLML y manda a crear el ServiceDLML a la `ServicesFactory`.
- Paso 13, 14 y 15: el `ServiceGenericSOAPImpl` por medio del `ServiceDLML`, prepara la entrada a través del `ConnectionSFTP` que hace uso de la librería Jcraft.

- Paso 16, 17 y 18: `ConnectionSFTP` establece una conexión SFTP con el cluster y manda los archivos de entrada para la aplicación DLML y posteriormente se cierra la sesión SFTP. Éstos pasos sólo son ejecutados cuando la aplicación DLML necesita de archivos de entrada por ejemplo multiplicación y suma de Matrices.

- Paso 19 y 20: el `ServiceGenericSOAPImpl` por medio del `ServiceDLML`, manda a ejecutar la aplicación DLML.

- Paso 21, 22 y 31: establece la conexión SSH con el cluster para ejecutar el script de Expect, el cual se encarga de ejecutar la aplicación DLML, cuando termina la ejecución, se cierra la conexión SSH.

- Paso 32 y 33: una vez que terminada la ejecución de la aplicación DLML, el `ServiceDLML` manda a obtener los archivos de salida por medio de `ConnectionSFTP`.

- Paso 34, 35 y 36: el `ConnectionSFTP` establece una conexión SFTP con el cluster y obtiene los archivos de salida de la aplicación DLML, por último cierra la conexión SFTP.

- Paso 37 y 38: el `ServiceDLML` hace uso del `ParserGeneric` para encapsular la respuesta en una cadena XML y notifica al `ServiceGenericSOAPImpl` que ha terminado la ejecución del servicio.

- Paso 39 y 40: el `ServiceGenericSOAPImpl` empaqueta la respuesta del servicio en un mensaje SOAP por medio del Motor Axis, y por último manda el mensaje SOAP al `ConsumerService` del Portal-DLML.

La capa de Servicios-DLML, es la capa que actúa como un Broker entre las peticiones del usuario y los clusters, ésta gestiona casi en su totalidad los HPCaaS. La última capa Aplicaciones-DLML, es donde se encuentra todo el HPC del Cloud-DLML y es la capa final para la ejecución de un HPCaaS en Cloud-DLML.

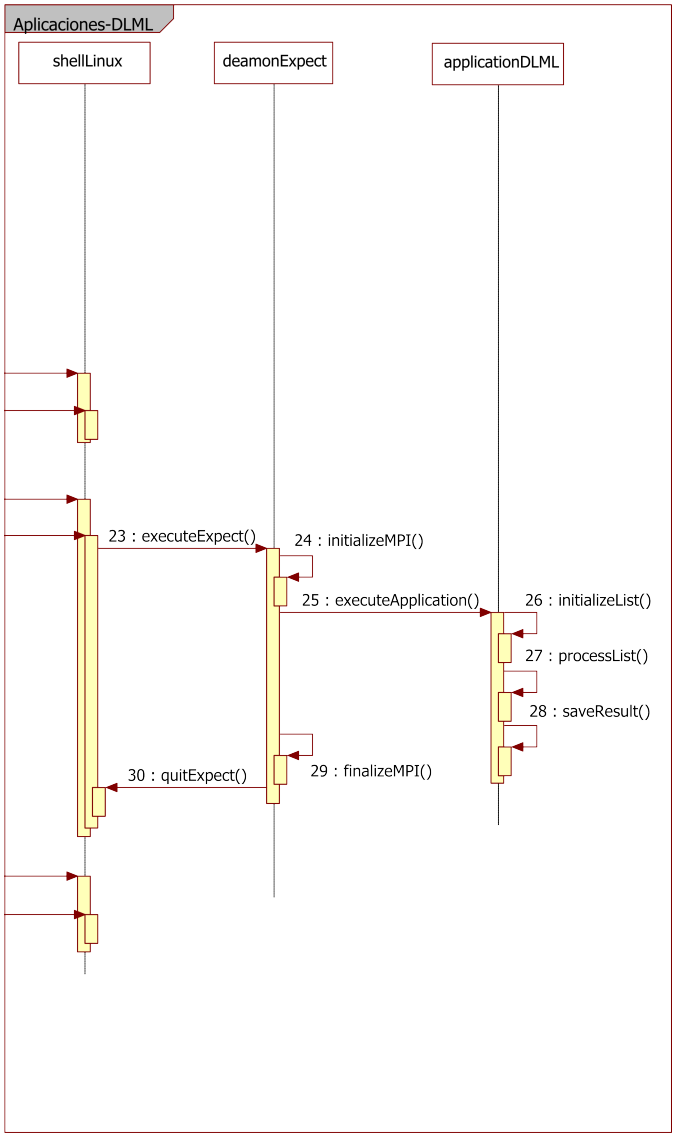

Figura 5.4: Diagrama de secuencia de la capa de Aplicaciones-DLML

En el fragmento del diagrama de secuencia de la capa de Aplicaciones-DLML (Figura 5.4), se muestran los pasos que son ejecutados por los servicios y aplicaciones DLML sobre los clusters, a continuación se describen estos pasos:

- Paso 23: en este paso el `ServiceDLML` una vez que estableció una conexión SSH, ejecuta el script de Expect, el cual lanza un demonio de Expect que permite al `ServiceDLML` mantener una conexión con los nodos del cluster.
- Paso 24 y 25: por medio del `DeamonExpect` se inicializa a MPI con el número de nodos seleccionados por el usuario, una vez inicializado MPI se manda a ejecutar la aplicación DLML.

- Paso 26, 27 y 28: la `ApplicationDLML` (N-Reinas, Multiplicación de Matrices y Suma de Matrices), inicializa la lista DLML con los datos provenientes del `ServiceDLML`, estos datos pueden ser parámetros o archivos de entrada, posteriormente se procesa cada uno de los ítems de la lista, al terminar de procesar toda la lista se mandan los resultados a un archivo de salida.
- Paso 29 y 30: una vez que finaliza la `ApplicationDLML`, se finaliza MPI y se sale del `DeamonExpect`, para que por defecto el `ServiceDLML` cierre la sesión SSH y posteriormente pueda obtener el archivo de salida.

Lo anterior fue el último fragmento del diagrama de secuencia que se ejecuta en un servicio de Cloud-DLML, para ver el diagrama completo ver (Anexo C). Todos estos módulos además que hacen posible los HPCaaS, fueron consecuencia del diseño de una arquitectura para Cloud-DLML que cumpliera con una calidad aceptable, en la siguiente sección se muestra como cumple esta arquitectura con los atributos de calidad planteados al inicio de su diseño.

5.2. Evaluación de atributos de calidad

Una vez ya mencionados los componentes o módulos que integran a Cloud-DLML, lo siguiente es hacer una evaluación de la arquitectura del sistema. Basándonos en el ATAM (Method for Architecture Evaluation), que es un método para la evaluación de arquitecturas de sistemas, este método es similar a los métodos usados para el diseño QAW/ADD, al igual que éstos el ATAM es basado en los atributos de calidad de la arquitectura de un sistema [42]. Así por medio del ATAM hacemos una evaluación del diseño de la arquitectura de Cloud-DLML (Figura 5.5), enfocándonos a cómo es que cumple con ciertos atributos de calidad en Cloud-DLML.

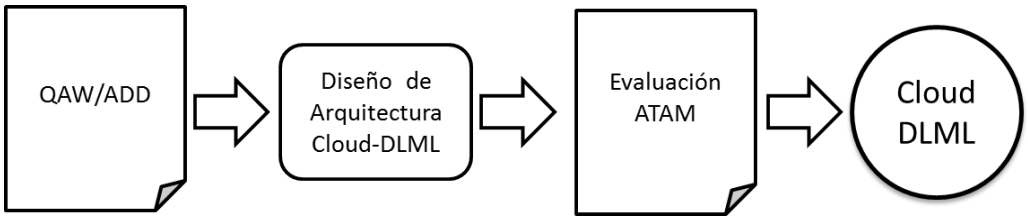

Figura 5.5: Evaluación ATAM sobre Cloud-DLML

A continuación se muestran en orden de prioridad los escenarios, en los cuales se muestra cómo es que Cloud-DLML cumple ciertos atributos de calidad, algunos de éstos fueron tomados en cuenta para su diseño y otros son consecuencia de los patrones arquitectónicos y de diseño implementados para su construcción. A continuación se muestran cómo se comporta Cloud-DLML a cada uno de estos escenarios:

Usabilidad: toda la parte de la interfaces web con el usuario se implementó con tecnologías RIA (Rich Internet Applications), lo cual hace que sea fácil de usar por el usuario ya que son interfaces sumamente dinámicas además de ser asíncronas. La funcionalidad del Portal-DLML es muy intuitiva por parte del usuario, el diseño de éste se basa en un panel de control principal, que coordina las interfaces de los servicios y esto hace que el usuario pueda navegar fácilmente entre los diferentes servicios abiertos por medio de pestañas.

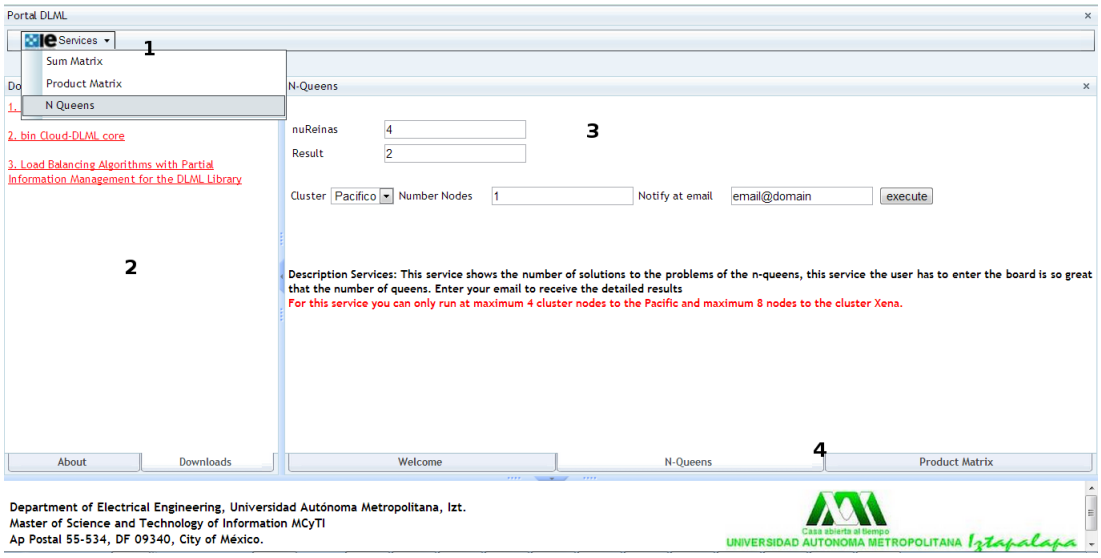

Figura 5.6: Interfaz web del Panel de Control del Portal-DLML

El Panel de Control del Portal-DLML, básicamente cuenta con 4 partes (Figura 5.6), las cuales describimos a continuación:

1. El menú de los servicios que ofrece Cloud-DLML, actualmente se cuenta con tres servicios (N-Reinas, Multiplicación de Matrices y Suma de Matrices), al seleccionar una opción del menú de servicios se despliega su interfaz web correspondiente para insertar los datos de dicho servicio y poderlo mandar a ejecutar.

2. En el área 2, del Portal-DLML se encuentra el "About" del Cloud-DLML, en el cual se presentan la información de los desarrolladores y los correos electrónicos de éstos para un posible contacto por parte de los usuarios. También se presenta el área de "Descargas" en la cual, el usuario puede descargar las funciones núcleo de Cloud-DLML, así como la documentación java de cada una de esta funciones, también se encuentra las URL's importantes del desarrollo de Cloud-DLML.

3. Esta área del Panel de Control, corresponde a la interfaz principal del panel, en ésta se despliegan las interfaces web de los diferentes servicios que selecciona el usuario, para el ejemplo del servicio N-Reinas el usuarios tiene que ingresar los siguientes datos:

   - nuReinas: en este campo el usuario ingresa el número de reinas, es decir, el tamaño del tablero para la aplicación DLML N-Reinas.
   - Result: este campo no es editable por el usuario, ya que sobre éste se despliega el resultado del servicio, una vez que termina de ejecutarse.
   - Cluster: en éste el usuario selecciona el cluster en el que se va a ejecutar el servicio, actualmente sólo hay dos opciones posibles (Pacífico y Xena).

- Number Nodes: el usuario sobre este campo ingresa el número de nodos del cluster seleccionado, sobre los cuales se ejecutara la aplicación DLML. El usuario actualmente sólo puede seleccionar hasta 4 nodos para el cluster Pacífico y hasta 8 nodos para el cluster Xena.
- Notify at email: sobre este campo el usuario ingresa su correo electrónico en caso de que requiera recibir la notificaciones de los resultados vía email, por parte de Cloud-DLML.

4. EL área de pestañas del Portal-DLML, permite al usuario intercambiarse al usuario entre las diferentes interfaces web de los servicios seleccionados por éste, y así interactuar de manera rápida con cada uno de los servicios de Cloud-DLML.

Escalabilidad: Cloud-DLML en un principio estaba pensado para que haga uso de un cluster con lo cual se obtenía escalabilidad a nivel de nodos, pero en el transcurso de la implementación y gracias a la arquitectura, nos da la oportunidad de escalar en recursos para el procesamiento de la aplicaciones, no sólo a nivel de nodos en un cluster sino también escalar a nivel de clusters, que puedan ser utilizados por los servicios DLML; ya que la capa de Servicios-DLML está desacoplada de la infraestructura donde se ejecutan las aplicaciones DLML y esto permite que Cloud-DLML sea multicluster. Para agregar un cluster sólo se necesita una cuenta SSH y que tanga instalado las librerías de Expect que son usadas para interactuar con los recursos del cluster.

Gracias a los mecanismo implementados con Jcraft y Expect, Cloud-DLML es capaz de hacer uso de múltiples clusters, en un principio se pensaba hacer uso de un sólo cluster (Pacífico), pero posteriormente se pudo en más de uno, por lo cual también se anexo a la infraestructura de Cloud-DLML el cluster Xena, dándole la opción al usuario de elegir dos posibles clusters para ejecutar los servicios. Los prerrequisitos que debe tener un cluster, para que pueda ser usado por Cloud-DLML como parte de infraestructura son los siguientes:

- Tener instalado Expect.
- Tener instalado alguna versión de MPI.

Posteriormente cumplidos los prerrequisitos, para anexar el nuevo cluster a Cloud-DLML, se tiene que crear el archivo `nameCluster.properties`, en el cual se especifican las propiedades suficientes para poder establecer las conexiones SSH y SFTP con el cluster, a continuación se muestran las propiedades para el caso del cluster Pacífico:

```
#pacifico
cluster.host=pacifico.izt.uam.mx
cluster.port=22
cluster.user=abraham
cluster.pwd=*********
cluster.node=pacifico
cluster.home=/home/alumnos/abraham
cluster.maxnodes=4
```

Posteriormente en el cluster, habrá que alojar las aplicaciones DLML y crear los scripts de Expect correspondientes a cada una de las aplicaciones alojadas, a continuación se muestra el ejemplo de un script de Expect para el caso del servicio N-Reinas:

```
#!bin/bash
/usr/bin/expect <<EOD
spawn ssh karazu@pacifico1
set timeout -1
expect "$"
send "cd DLML_NQ/\r"
expect "$"
send "sh todo.sh\r"
expect "$"
send "lamboot -v maquinas.txt\r"
expect "$"
send "mpirun -np $2 dlbs_todo $1\r"
expect "$"
send "lamwipe\r"
send "exit\r"
expect eof
EOD
```

El script anterior básicamente lo que hace es inicializar a Expect, desactivar el timeout debido a que la ejecución de una aplicación puede tardar demasiado, posteriormente inicializa MPI y ejecuta la aplicación DLML, esperando a que ésta termine para finalizar a MPI junto con Expect.

Al Hacer lo anterior Cloud-DLML podrá hacer uso de otro cluster para ejecutar HPCaaS (Figura 5.7).

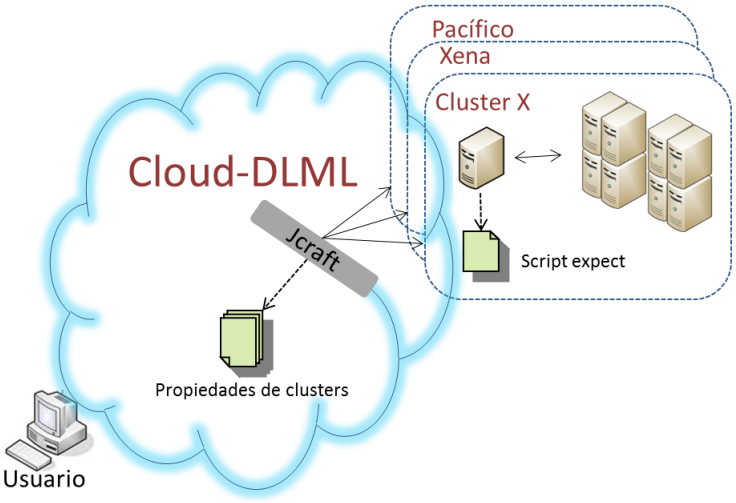

Figura 5.7: Escalabilidad de clusters en Cloud-DLML

Interoperabilidad: esta arquitectura está basada en SOA y su implementación se hizo utilizando WS consiguiendo con esto todas las ventajas de los protocolos estándar de comunicación y conexión. Con lo anterior es posible que el Cloud-DLML pueda interactuar con otras aplicaciones, es decir, tanto la parte del Portal-DLML puede invocar otros servicios ajenos a los servicios DLML, así como los servicios puedan ser invocados por otras aplicaciones ajenas al Cloud-DLML.

Para que los servicios actuales los pueda usar otros clientes (sistemas), por medio del WSDL del *WS Genérico* (Anexo C) podría hacerlo, ya que a través de éste el cliente podría implementar sus propios consumidores lógicos del servicio, encapsular los datos en un mensaje SOAP apropiado y establecer una conexión con los servicios DLML (Figura 5.8).

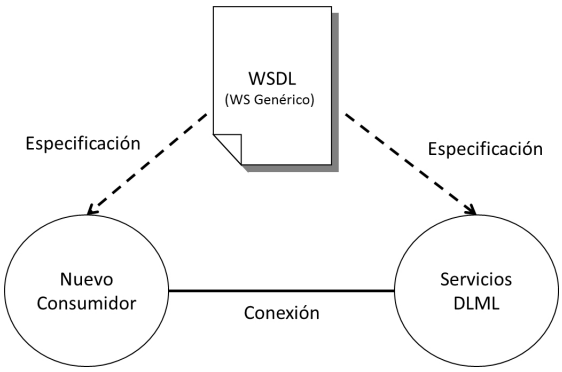

Figura 5.8: Interoperabilidad de la capa de Servicios-DLML

Por otro lado, el Portal-DLML de igual manera podría hacer uso de otros servicios por medio de un WSDL de los servicios externos que si quisieran ejecutar (Figura 5.9).

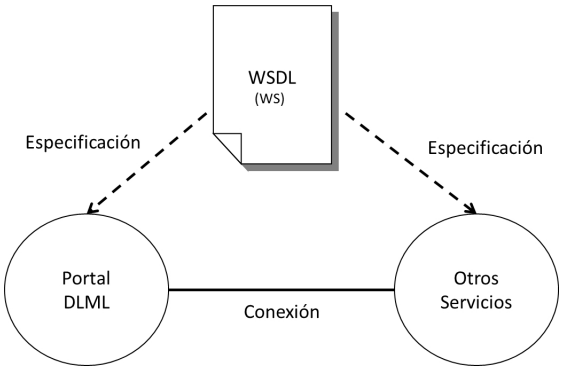

Figura 5.9: Interoperabilidad de la capa de Portal-DLML

Por lo tanto con la arquitectura de Cloud-DLML, éste queda abierto para la posible interoperabilidad con otros sistemas.

Adaptabilidad: esto es consecuencia de que la arquitectura está compuesta por tres capas, donde cada una encapsula funcionalidad de acuerdo al nivel de abstracción. Supongamos se quiere cambiar la funcionalidad de un servicio, esto sería fácil suponiendo que no cambian los parámetros de entrada y de salida, entonces sólo se tendría que modificar el servicio DLML o tal vez la aplicación DLML. Por otra parte la arquitectura permite agregar nuevos servicios de manera fácil, para esto todo lo que se tiene que hacer es la interfaz web del servicios y posteriormente agregarlo en la fábrica de consumidores y servicios para crear los enlaces a la aplicación HPC.

Para agregar un nuevo servicio el desarrollador en la capa del Portal-DLML tiene que crear la interfaz web y agregarla al menú de servicios disponibles de Cloud-DLML, así el Panel de Control, pueda disponer de ésta cuando el servicio sea seleccionado por el usuario.

Posteriormente tendría que implementar el `ConsumerService` con sus tres métodos (`readyInput`, `useService` y `readyOutput`) apoyándose de las funciones núcleo para su correcto funcionamiento. Posteriormente habría que agregarlo a la `ConsumersFactory` para que el `Controller` pueda usarlo (Figura 5.10).

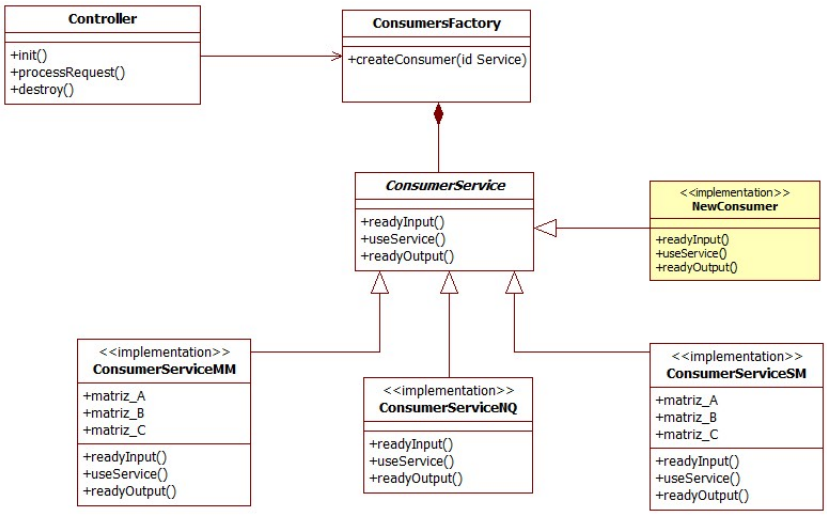

Figura 5.10: Agregar un consumidor en la Fábrica de Consumidores

Una vez echo lo anterior en la capa de Servicios-DLML el desarrollador tendría que agregar la implementación del ServiceDLML (Figura 5.11), con sus métodos correspondientes (`readyInput`, `executeService`, `readyOutput`) para la nueva aplicación DLML, haciendo uso de las funciones núcleo para establecer conexiones SSH y SFTP con el cluster en el que se quiera ejecutar.

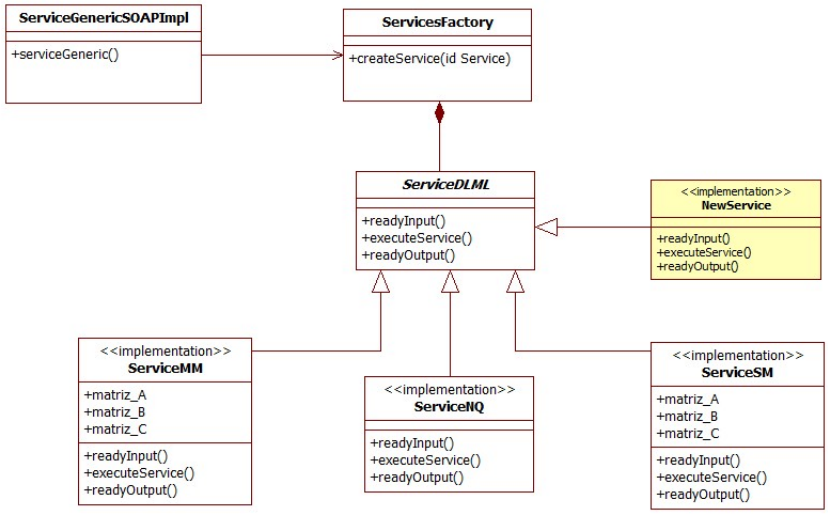

Figura 5.11: Agregar un servicio en la Fábrica de Servicios

Por último se tiene que alojar la nueva aplicación DLML en los clusters disponibles de Cloud-DLML. Con todo lo anterior el usuario podrá disponer de nuevos servicio en cloud-DLML.

Portabilidad: la implementación de Cloud-DLML se hizo con tecnologías y protocolos estándares, precisamente para garantizar la portabilidad. El usuario puede hacer uso de Cloud-DLML desde cualquier dispositivo con conexión a Internet, como puede ser un celular, una tablet, una PC o una laptop, por medio de un navegador web, ya que se usaron tecnologías estándar como Ajax, JS y DHTML que no requieren de algún navegador en especial o instalación de algún plugín sobre éstos.

Al terminar el desarrollo de los servicios se hicieron pruebas con los diferentes navegadores más usados por los usuarios de Internet (Chrome, Mozilla e IE), también hicimos una evaluación sobre cómo se comportaba la aplicación al ejecutarse sobre estos (Tabla 5.1).

Servicio	Chrome	Mozilla Firefox	IE
N-Reinas	éxito	éxito	éxito
Multiplicación de Matrices	éxito	éxito	éxito
Suma de Matrices	éxito	éxito	éxito

Tabla 5.1: Compatibilidad con los navegadores web

También es importante mencionar que gracias a la tecnología estándar con la que fue desarrollada el Portal-DLML, este queda accesible casi desde cualquier dispositivo con una conexión a Internet, por esta razón se ejecutaron una serie de pruebas de los HPCaaS del

Cloud-DLML con las principales plataformas, cada una con distinto sistema operativo (Tabla 5.2).

Servicio	Windows	Linux	iOS	Android
N-Reinas	éxito	éxito	éxito	éxito
Multiplicación de Matrices	éxito	éxito	éxito	éxito
Suma de Matrices	éxito	éxito	éxito	éxito

Tabla 5.2: Compatibilidad con los SO's

Los anteriores fueron atributos en los cuales se basó el diseño y la implementación de Cloud-DLML, a continuación describimos como es que se comporta Cloud-DLML en relación a otros atributos de calidad que de igual manera consideramos importantes para Cloud-DLML:

Flexibilidad: este es muy importante en el Cloud-DLML, ya que esta pensado que con el paso del tiempo se anexen servicios DLML o incluso de otro tipo como puede ser MPI o MapReduce, u otra tecnología controlable a través de SSH. Cloud-DLML esta diseñado para soportar varias herramientas de HPC, por ejemplo si se quisiera agregar servicios MPI lo que se tendría que hacer es diseñar la aplicación para recibir las entradas por medio de parámetros o por archivos, mientras que las salidas de dicha aplicación tendrían que ser enviados a un archivo, sólo estas dos restricciones son suficientes para que pueda ser ofrecido como un servicio de Cloud-DLML.

Disponibilidad: por tener una aplicación web (Portal-DLML) para el usuario, se busca que haya una disponibilidad 24x7 de Cloud-DLML, por otra parte ya que la arquitectura es escalable a nivel de clusters, se puede garantizar una alta disponibilidad global aunque los clusters que integran Cloud-DLML no sean dedicados. Así cuando un usuario no pueda ejecutar un servicio en un cluster de Cloud-DLML, el usuario puede intentar ejecutar el servicio en otro cluster de Cloud-DLML.

Tolerancia a Fallas: actualmente la herramienta DLML usada para las aplicaciones paralelas no es tolerante a fallas, pero en caso de que una aplicación falle por alguna razón, Cloud-DLML sólo notifica al usuario que no se pudo ejecutar la aplicación, permitiendo a éste ejecutar en otro cluster o simplemente volver a intentar. Es muy importante tener en cuenta, que hasta cierto punto Cloud-DLML si es tolerante a fallas, pero esto no implica que las aplicaciones de HPC también lo sean, ya que estas pueden o no ser tolerante a fallas.

Seguridad: Cloud-DLML actualmente es seguro en la parte de las últimas capas (Servicios-DLML y Aplicaciones-DLML), en la capa de los servicios se cuenta con una bitácora que registra todas las peticiones de los servicios, por otra parte en la última capa, toda conexión se hace por medio de los protocolos SSH y SFTP. Para la capa del Portal-DLML en una

primera instancia se diseñó e implementó para que no haya autentificación y que esté abierto para todo usuario, quedando como una mejora un método de autentificación para que ésta sea usado sólo por usuarios autorizados.

Los anteriores son atributos de calidad deseables para cualquier sistema de computación, en los puntos anteriores se menciona cómo Cloud-DLML cumple tales atributos. Con esta evaluación se puede ver que el diseño de la arquitectura hace posible que Cloud-DLML cumpla con una buena calidad, cubriendo atributos de calidad muy importantes entre los cuales para el fin de Cloud-DLML se destacan los atributos de usabilidad y portabilidad para el usuario; para el lado del desarrollo, la adaptabilidad, flexibilidad y escalabilidad, son los atributos a destacar. Lo siguiente por analizar es el comportamiento del sistema ya desplegado para brindar servicios sobre la nube.

5.3. Despliegue de Cloud-DLML

La otra parte en cuanto resultados es la parte del despliegue del sistema (deployment) Cloud-DLML ya funcionando y ejecutando servicios HPCaaS sobre la web. Para el deployment del sistema Cloud-DLML se lleva a cabo en dos diferentes plataformas, la primera por medio de una intranet y la segunda utilizando la plataforma de Windows Azure, a continuación se describe cada una de estas.

5.3.1. Despliegue sobre una intranet

Esta fue la primera plataforma de experimentación dentro de una intranet, en donde los clientes del portal debían estar dentro de la intranet para que pudieran hacer uso de la nube Cloud-DLML. Esta plataforma consistió en dos laptops una para alojar el Portal-DLML y otra para los Servicios-DLML, y en cuanto infraestructura se utilizaron los clusters Pacífico y Xena para la capa de Aplicaciones-DLML.

Portal-DLML	Servicios-DLML	Aplicaciones-DLML
Laptop (Windows 7 64 bits), en la cual se desplegó el Portal-DLML sobre el servidor web Tomcat.	Laptop (Ubuntu 11.04 64 bits), en el cual se levantaron los servicios DLML sobre el servidor web Tomcat.	Cluster Pacífico (1-4 nodos) con LAM 7.1.4/MPI 2 y Cluster Xena (1-8 nodos) con LAM 7.1.4/MPI 2, en los cuales se alojaron las aplicaciones DLML.

Tabla 5.3: Plataforma de experimentación sobre una intranet

Una vez levantado el sistema Cloud-DLML (Anexo A) sobre esta plataforma (Tabla 5.3), se procedió a ejecutar los servicios sobre ambos clusters Pacífico y Xena, en el cual se pudieron ejecutar el primer servicio que hasta ese entonces estaba funcionando, nos referimos al servicio de las N-Reinas, el cual se mandó a ejecutar sobre los cluster Pacífico y Xena, obteniendo buenos resultados en cuanto a la funcionalidad de Cloud-DLML. Posteriormente se mando a ejecutar el servicio de N-Reinas para un tablero de `16 x 16` sobre el cluster Xena usando de 1 hasta 8 nodos (Figura 5.12).

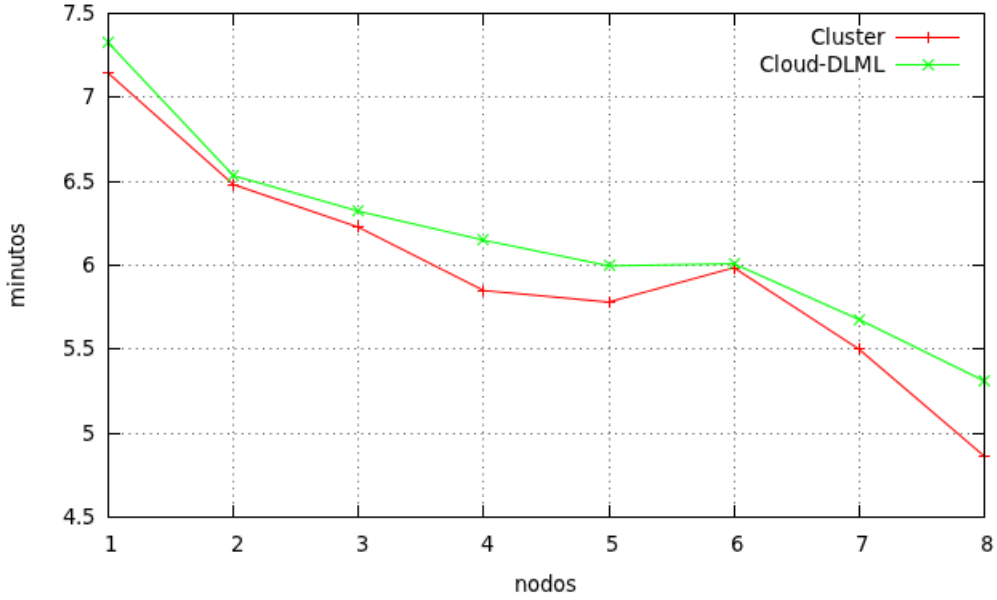

Figura 5.12: Gráfica del servicio N-Reinas para un número de 16 reinas, sobre el despliegue en una intranet

La gráfica anterior muestra los tiempos de ejecución en minutos para diferentes número de nodos, la linea superior representa los tiempos de Cloud-DLML y la linea inferior los tiempos de ejecutar directamente la aplicación DLML sobre el cluster. Después de analizar los datos de la gráfica se puede concluir que el overhead de Cloud-DLML es mínimo y casi constante al menos para este servicio (N-Reinas), aunque con pequeñas fluctuaciones de tiempo, debido a las variaciones de la red de la intranet.
Hasta este punto se tenía una arquitectura Cloud-DLML capaz de ejecutar HPCaaS sobre múltiples clusters, aun que para éste caso el sistema no era totalmente disponible, ya que éste sólo podía ser usado dentro de la intranet en la que estaba desplegado el Cloud-DLML, en el siguiente caso del despliegue sobre la plataforma Azure, esto cambiaría y se terminaría de ofrecer los tres servicios planteados en un inicio a través de Internet.

5.3.2. Despliegue sobre la nube Azure

Posteriormente una vez probado Cloud-DLML dentro de una intranet, el sistema estaba listo para ser desplegado en otra plataforma de experimentación, para este caso se usaron recursos de la nube de Windows Azure, el cual es un proveedor en la nube que permite crear infraestructuras y también provee de herramientas para el desarrollo de aplicaciones, es decir, Azure es un proveedor que ofrece servicios PaaS e IaaS [43]. En este caso nosotros ocupamos los servicios de IaaS de Azure para la virtualización de los servidores del Portal-DLML y Servicios-DLML, mientras que los recursos de clusters para la capa de Aplicaciones-DLML siguieron siendo los mismos (Tabla 5.4).

Portal-DLML	Servicios-DLML	Aplicaciones-DLML
VM (Windows Server R2 64 bits), en la cual se despliega el Portal-DLML sobre el servidor web Tomcat.	VM (Windows Server R2 64 bits), en el cual se instala la parte del sobre el servidor web Tomcat.	Cluster Pacífico (1-4 nodos) con LAM 7.1.4/MPI 2 y Cluster Xena (1-8 nodos) con LAM 7.1.4/MPI 2, en los cuales se alojan las aplicaciones DLML.

Tabla 5.4: Plataforma de experimentación sobre la nube Azure

El servicio que se utiliza para el deployment de Cloud-DLML sobre Azure, es CloudServices de Azure. En Azure actualmente hay tres tipos de roles para estos servicios estos son: el web-role, worker-role y VM-role. En este caso implementamos un worker-role que es un servidor que atiende las peticiones web de los usuarios, en este worker-role se incluyeron tanto el Portal-DLML como los Servicios-DLML, en el caso anterior de la plataforma de experimentación de la intranet, se probó que las tres capas de Cloud-DLML pueden estar en diferentes recursos de hardware, en este caso por cuestiones de costos Portal-DLML y Servicios-DLML se virtualizaron en un solo servidor con dos instancias sobre la nube de Azure, es decir hay una redundancia de servidores aprovechando así el balance de carga que realiza Azure sobre sus servidores virtuales cuando hay más de una instancia. A continuación se describe el proceso con el que se virtualiza el Cloud-DLML basado en las herramientas que proporciona Azure para Java y el servidor Tomcat [44], este se describe en cuatro pasos:

1. Primero se crea el paquete, que contiene las capas del Portal-DLML y Servicios-DLML, este paquete es el que será compatible con Azure.

2. Una vez creado el paquete, se crea el archivo de configuración del servicio, en este archivo se define el número de instancias en la nube, el tipo de rol del servicio, también se crea el certificado para poder acceder al servidor de Azure.

3. Hasta este punto se tienen creados el paquete, el archivo de configuración y el certificado. Una vez teniendo éstos se crean las instancias de los servicios desde la plataforma Azure.

Para ello hay que ingresar al portal de Azure para crear un servicio el cual se crea a partir del paquete, el archivo de configuración y el certificado; al terminar de crear los servicios se crean las dos instancias de servidores virtuales que alojan la capa del Portal-DLML y Servicios-DLML.

4. El último paso es configurar el Portal-DLML y Servicios-DLML, por lo cual lo único que se hace, es acceder a los servidores virtuales por medio del certificado creado, y copiar los archivos de configuración propios de Cloud-DLML (Anexo A).

Los anteriores son los pasos generales para llevar a cabo el deployment de Cloud-DLML sobre Azure, en cuanto a los clusters usados para la capa de Aplicaciones-DLML, son los mismos, el cluster Pacífico y Xena.

A diferencia del despliegue en una intranet, en este despliegue final de Cloud-DLML sobre Azure, se tenían ya listos los tres servicios N-Reinas, Multiplicación de Matrices y Suma de Matrices, los cuales se probaron exitosamente, mandándolos a ejecutar por medio Internet sobre ambos clusters. Para este despliegue también se hizo un análisis del overhead, para saber cómo se comportaba Cloud-DLML en esta plataforma de despliegue, primero se hizo la comparación de tiempos para el servicio de las N-Reinas con un tablero de `16 x 16` sobre el cluster Xena (Figura 5.13).

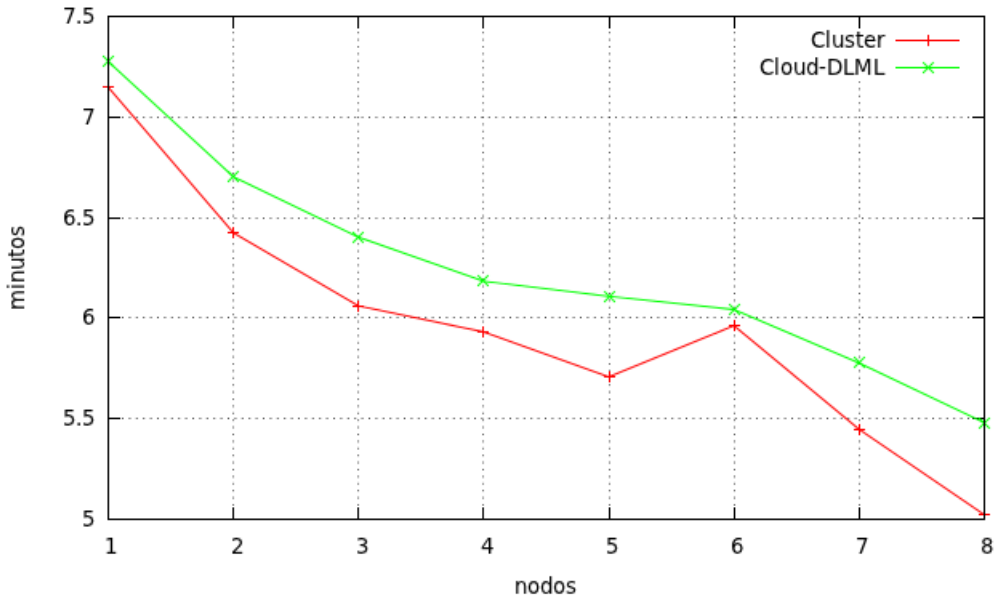

Figura 5.13: Gráfica del servicio N-Reinas para un número de 16 reinas, sobre el despliegue en Azure

El overhead de Cloud-DLML para el servicio de las N-Reinas y sobre este despliegue en Azure, es mayor al que se presentó en el despliegue sobre una intranet, esto es debido a la

distancia geográfica entre el servidor de Servicios-DLML y ambos clusters (Pacífico y Xena), ya que esto ocasiona un mayor delay dentro de Cloud-DLML.

Posteriormente se mando a ejecutar el servicio de la Multiplicación de Matrices, para matrices de `100 x 100`, sobre el cluster Xena usando de 1 a 8 nodos (Figura 5.14).

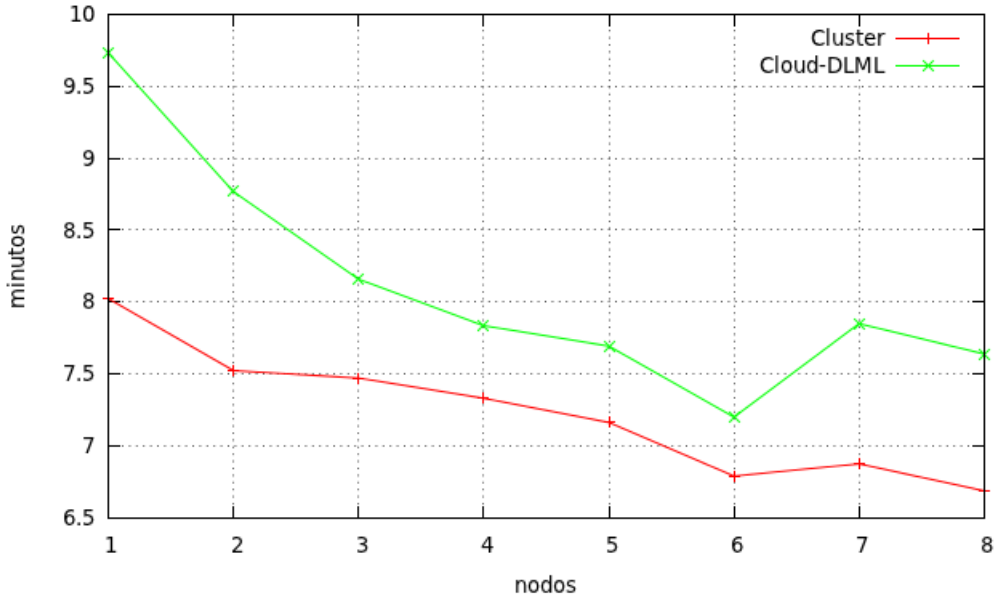

Figura 5.14: Gráfica del servicio Multiplicación de Matrices para matrices de 100 x 100, sobre el despliegue en Azure

De la gráfica anterior se observa, que para este caso, el overhead de la arquitectura de Cloud-DLML, es mucho mayor, esto fue debido a la transferencia de archivos entre el servidor de la capa de Servicios-DLML y los clusters (Pacífico y Xena), ya que la aplicación DLML (Multiplicación de Matrices) tenía como entrada los archivos correspondientes a las matrices A y B, por lo cual para este servicio se tenían que establecer conexiones SFTP a diferencia del servicio de N-Reinas para el cual no era necesario archivos de entrada.

Al final del despliegue sobre Azure, Cloud-DLML se hizo accesible a través de Internet, pudiendo ejecutar los servicios desde cualquier lugar del mundo por medio de una conexión a Internet, con esto también se demuestra la flexibilidad de la arquitectura, por otra parte también se hicieron pruebas con algunos dispositivos móviles como celular y tablet, los cuales ejecutaron con éxito los servicios.

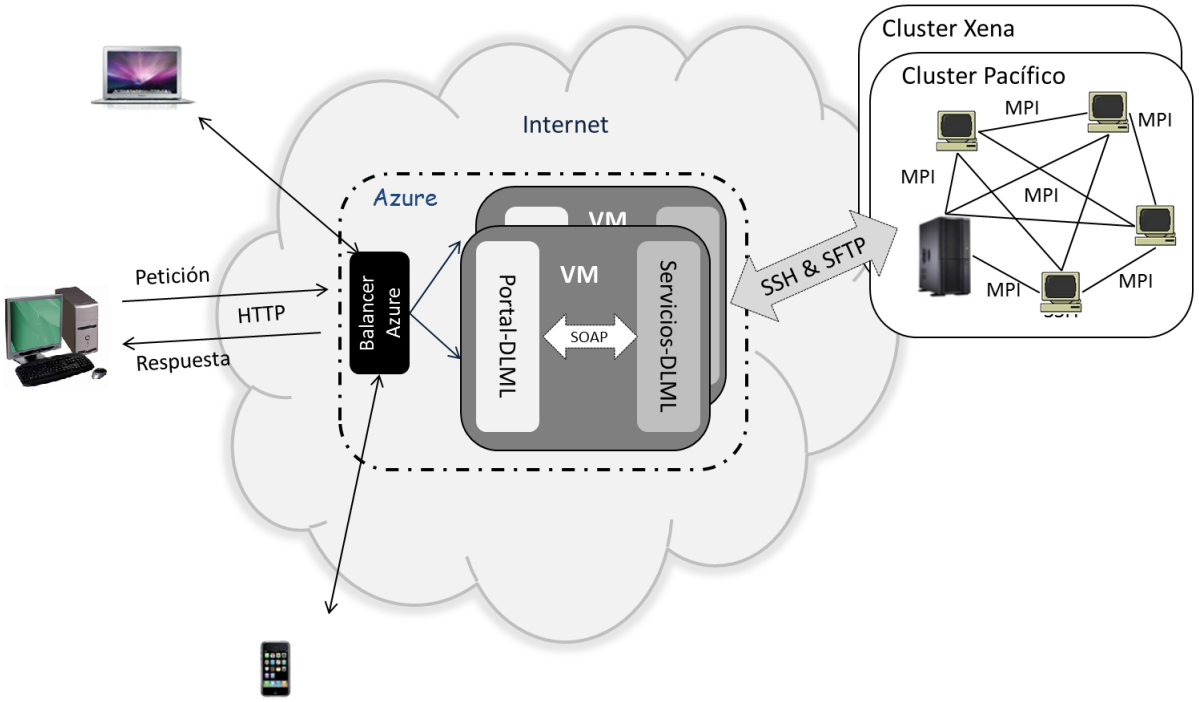

Figura 5.15: Despliegue de Cloud-DLML sobre la nube Azure

Al final de este deployment (Figura 5.15), se obtiene un sistema que brinda HPCaaS usando infraestructura de Azure para las dos primeras capas, el Portal-DLML y los Servicios-DLML. En cuanto a la capa de las Aplicaciones-DLML se usaron los clusters Pacífico y Xena. Con este deployment, Cloud-DLML queda disponible para la ejecución de los HPCaaS, desde cualquier lugar y casi cualquier dispositivo con conexión a Internet.

Capítulo 6
Conclusiones y trabajo a futuro

En esta tesis se presento una nube denominada como Cloud-DLML, la cual ofrece servicios HPCaaS donde no sólo se ofrece la infraestructura, sino que también el software que consiste en una terna de aplicaciones HPC implementadas con DLML. Las aplicaciones que se han ofrecido como los primeros servicios son: N-Reinas, Multiplicación y Suma de Matrices los cuales han sido probados con éxito. En un principio Cloud-DLML sólo contiene estos servicios generales que realmente sirvieron en gran parte para probar la arquitectura del sistema. Por otra parte también es importante decir que Cloud-DLML queda abierto a desarrolladores, sobre todo aquellos que quieran compartir sus aplicaciones HPC para que estas puedan ser ejecutadas como servicios por los distintos usuarios, así con esto se busca que de manera colaborativa Cloud-DLML empiece a escalar en servicios y con esto transformar aplicaciones de HPC a servicios HPCaaS de tal forma que estén disponibles 24 x 7, para los usuarios por medio de un dispositivo con conexión a Internet y así explotar mejor los recursos de cómputo evitando clusters dedicados y disponibles a cualquier hora desde cualquier lugar por medio de una conexión a Internet.

Cloud-DLML surgió como necesidad por parte de los usuarios del HPC, ya que en general la mayoría de los usuarios que requieren de este tipo de cómputo son personajes relacionados con la ciencia y la investigación, como los son los físicos, matemáticos, biólogos, etc., por mencionar algunos, para ellos en la mayoría de los casos, tienen que lidiar con problemas para obtener los recursos de infraestructura, mas aún el lidiar con la programación paralela es sumamente complejo para estos personajes. En el transcurso de la investigación nos dimos cuenta de que el Cloud Computing es una propuesta del cómputo que en general facilita todo a los usuarios, contrariamente a lo que pasa con el HPC otra propuesta del cómputo que implica una gran complejidad para sus usuarios. Aun que el algunos proveedores de Cloud Computing comienzan a ofrecer servicios de HPC, en general se evidencio que el HPC es una necesidad no muy atacada por el Cloud Computing, más aún los HPCaaS actuales están a nivel de IaaS, los cuales son todavía más complicado para la mayoría de los usuarios. Por tal motivo el desarrollo de Cloud-DLML, es una buena opción para quienes necesitan hacer uso del HPC, sobre todo para quienes están involucrados con el cómputo científico.

Para desarrollar Cloud-DLML también fue necesario hacer una investigación del contexto actual del HPC, la cual arrojo tecnologías tan potentes como los clusters, grids, MPI y

DLML, que son usadas para resolver aplicaciones que requieren gran demanda de procesamiento. Posteriormente a la revisión del contexto del HPC, se optó por usar los clusters como hardware y DLML como la herramienta de software, como la infraestructura para el HPC dentro de Cloud-DLML. Por la parte del Cloud Computing fue necesario entender los aspectos fundamentales que se tienen que considerar para la implementación del cómputo en nube. Todo lo anterior complementado con los métodos del QAW/ADD y el ATAM proporcionados por el SEI, hicieron posible el diseño e implementación de una arquitectura que cumple con una calidad aceptable para los propósitos de Cloud-DLML.

El patrón de arquitectura SOA, en el que se basó el desarrollo de Cloud-DLML se adaptó muy bien a las necesidades que se plantearon en un principio, ya que la implementación se realizó con WS, permitiendo a Cloud-DLML que en un futuro posible pueda interactuar con otras aplicaciones o nubes. Este patrón de arquitectura junto con la capa que se le anexó de HPC, hizo posible que la implementación de la arquitectura cumpla con varios atributos de calidad deseables para cualquier sistema de cómputo, además de que Cloud-DLML se implementó usando tecnología estándar y de uso libre, factores que pueden ser determinantes para su crecimiento.

La plataforma de Cloud-DLML queda lista para el desarrollo de nuevos servicios, ya que el *WS Genérico* facilita la creación de servicios, de hecho al inicio del desarrollo se implementaron tres WS correspondientes a cada aplicación DLML pero al final no se utilizaron debido a que no se tenía claro cuáles eran los parámetros de entrada y salida, por lo cual cada vez que cambiaba alguno de éstos, había que rehacer todo WSDL, SOAP y clases del servicio, lo cual era demasiado ineficiente, es por esto que se implemento el *WS Genérico*, la *Fábrica de Consumidores* y la *Fábrica de Servicios* que resolvieron estas problemáticas. Gracias a estos tres componentes, Cloud-DLML es fácil de modificar y más aún hace que el desarrollo sea muy flexible para agregar un mayor número de servicios.

Otro aspecto muy importante es la flexibilidad y escalabilidad que permite Cloud-DLML, ya que a nivel de software, puede crecer en aplicaciones de HPC ofrecidas como servicios usando otras herramientas de programación paralela. Mientras que por el lado de la escalabilidad la arquitectura ha demostrado ser multicluster logrando así una alta escalabilidad, de esta manera la capacidad de infraestructura no queda limitada. Cloud-DLML está planteado para ser una nube híbrida por una parte pública, abierta a cualquier usuario y por otra parte es comunitaria ya que tiene infraestructura de los cluster "Pacífico" por parte de la UAM-I y el cluster "Xena" brindado por el CINVESTAV, queda abierta la posibilidad de que algunas otras instituciones brinden recursos.

Actualmente la arquitectura planteada de Cloud-DLML y la forma en que interactúan cada uno de sus componentes ya ha sido probada con éxito, dando como resultado tres primeros servicios de HPCaaS en operación de Cloud-DLML sobre dos diferentes tipos de deployment, actualmente Cloud-DLML ha quedado accesible desde la Internet y a cumplido los objetivos

planteados del proyecto satisfactoriamente. Este desarrollo ha dejado ya una infraestructura fácilmente escalable y una implementación de una arquitectura flexible para integrar otras tecnologías a demás de DLML y un mayor número de HPCaaS. Por lo cual se espera que sólo sea cuestión de tiempo, para que Cloud-DLML este en operación con un mayor número de servicios e infraestructura para ofrecer a los usuarios del cómputo científico que requieran del HPC.

Cabe mencionar que Cloud-DLML es un código abierto, para que otros desarrolladores puedan ampliar o utilizar la funcionalidad que existe, ya sea para agregar nuevos servicios a Cloud-DLML o para desarrollar su propia nube para ofrecer HPCaaS usando las funciones núcleo de Cloud-DLML (Anexo B).

Cloud-DLML así como cualquier otro sistema puede mejorar, ya que éste debe crecer y también tiene puntos de mejora. Es por esto que se muestran tres posibles mejoras que hemos considerado muy importantes las cuales son: seguridad, escalabilidad de clusters e incrementar el número de servicios para ofrecer a los usuarios.

En cuanto a seguridad, Cloud-DLML puede contar con un mecanismo de autentificación para que así sólo usuarios autorizados hagan uso de los recursos. Otro aspecto importante que se puede mejorar en Cloud-DLML es, usar un protocolo seguro sobre el Portal-DLML, dicho protocolo podría ser SSL o TLS, por medio de la implementación de estos dos mecanismos, los datos de los usuarios estarían más seguros y se evitarían posibles ataques por usuarios mal intencionados. Así con la implementación de los dos mecanismos anteriores, Cloud-DLML estaría brindando total seguridad a los usuarios, en todas las capas de su arquitectura.

Para el crecimiento de Cloud-DLML se desarrollaron módulos que implementan los mecanismos necesarios para agregar mayor infraestructura de HPC, es decir, en Cloud-DLML es posible agregar nuevos clusters además de los que usa actualmente (Pacífico y Xena). Para esto el cluster que se quiere agregar como únicos prerrequisitos, éste tendría que tener instalado las librerías de Expect y una versión de MPI. Por la parte de Cloud-DLML los desarrolladores solo deberían definir las propiedades del cluster y hacer uso de las funciones core de Cloud-DLML para realizar las conexiones SSH y SFTP, a continuación se definen cada una de estas acciones:

- Propiedades del Cluster: éstas se definen a través de un archivo de propiedades (cluster.properties) el cual se carga por medio de la clase ClusterVO, así como se hizo para los cluster Pacífico y Xena, que se define por medio de archivos de propiedades.
- Establecer conexiones: para esto el desarrollador sólo debe hacer uso de las funciones para la conexiones SSH y SFTP, que se encuentra en el core de Cloud-DLML, estas funciones reciben como parámetro un objeto de la clase ClusterVO que se crea a partir de las propiedades de los clusters.

Como se puede ver el agregar nueva infraestructura hasta cierto punto es fácil, por otra parte es importante mencionar, que para el cluster Cloud-DLML es visto como un usuario más que hace uso de los recursos, es por esto que el cluster no necesariamente debe estar dedicado a Cloud-DLML e inclusive éste puede establecer permisos a Cloud-DLML.

El otro factor importante para el crecimiento de Cloud-DLML, es el incremento en servicios. En el desarrollo actual de Cloud-DLML, se ofrecieron los servicios (N-Reinas, Multiplicación de Matrices y suma de Matrices), en el transcurso del desarrolló la arquitectura de Cloud-DLML ha demostrado ser muy flexible para agregar más servicios a Cloud-DLML, ya que después de agregar el primer servicio (N-Reinas) los otros dos fueron muy fácil de agregar a Cloud-DLML. Esta flexibilidad de Cloud-DLML, fue consecuencia de la implementación de la *Fábrica de Consumidores* en la capa del Portal-DLML y la *Fábrica de Servicios* en la capa de Servicios-DLML, el desarrollo del *WS Genérico* y las funciones core de Cloud-DLML. Para que el desarrollador pueda agregar nuevos servicios de HPC solo tendría que hacer lo siguiente:

1. Desarrollar la aplicación HPC, esta pude ser desarrolla en DLML o MPI, la aplicación solo debe tener como restricción cargar los datos de entrada desde archivo o por medio de parámetros, y guardar la salida en archivos.

2. Implementar la interfaz web correspondiente al servicio y anexarla al panel de control principal de Cloud-DLML.

3. Agregar el consumidor y el servicio en las fábricas correspondientes siguiendo el patrón de diseño Factory, para esto solo hay que implementar la interface *ConsumerService* del lado del Portal-DLML y *ServiceDLML* del lado de Servicios-DLML, en cuanto al *WS Genérico* no es necesario hacer cambio o agregar funcionalidad.

4. Modelar los datos de entrada de la aplicación HPC en un objeto VO (Value Object), y hacer uso del parser genérico para manipular el objeto VO en las capas del Portal-DLML y Services-DLML. Posteriormente habrá que hacer uso de las funciones núcleo para establecer las conexiones SSH y SFTP con el cluster sobre el cual se ejecutara dicho servicio.

Al ir agregando más servicios e infraestructura para ejecutar HPCaaS a nivel de software, Cloud-DLML podría convertirse en una plataforma que permita a diferentes involucrados (desarrolladores de HPC, desarrolladores web, usuarios del HPC y otros programadores), colaborar entre sí y brindar servicios con fines científicos, explotando diferentes infraestructuras que puedan estar dispersas, en diferentes lugares geográficos, haciendo esto posible a través de un dispositivo con conexión a Internet.

También es importante mencionar, que del desarrollo actual de Cloud-DLML podría surgir un Framework que permita desarrollar HPCaaS, para esto se necesitaría definir o automatizar aun más algunos procesos de desarrollo, que faciliten a los diferentes desarrolladores la

creación de nuevos HPCaaS dentro del sistema Cloud-DLML o bien fuera de éste en otras nubes de servicios HPCaaS.
De esta manera se explotaría mejor el trabajo que ha dejado esta tesis, para posibles nubes futuras enfocadas a ofrecer HPCaaS, satisfaciendo así parte de la demanda del HPC en el ámbito científico.

Anexo A

Instalación y configuración de Cloud-DLML

En este apartado, se muestra la manera en que debe ser puesto en operación cada una de las capas de Cloud-DLML.

Aplicaciones-DLML

1. Primero hay que revisar si el cluster contiene expect, con el siguiente comando:

   ```
   expect
   ```

   Si se encuentra instalado nos aparecerá el promp de "expect", sino se encuentra instalado habrá que instalarlo, para esto ejecutamos el siguiente comando sobre una terminal:

   ```
   sudo apt-get install expect
   ```

2. Alojar las aplicaciones DLML en el directorio del usuario SSH para Cloud-DLML, por lo cual tendrían las siguientes rutas:

   ```
   /home/userCloudDLML/DLML_NQ
   /home/userCloudDLML/DLML_MM
   /home/userCloudDLML/DLML_SM
   ```

   Para los servicios n-reinas, multiplicación y suma de matrices respectivamente.

   Nota: El cluster debe de contar con una versión instalada de MPI, ya que la librería DLML se ejecuta sobre MPI, si no hay alguna versión instalada puede instalar la versión LAM-MPI `http://www.lam-mpi.org/` *que es con la que se trabajo en Cloud-DLML.*

Servicios-DLML

1. Descargar el servidor web Tomcat puede ser la versión 6 en adelante, en este caso descargamos la versión 6 que se encuentra en la página web:

   `http://tomcat.apache.org/tomcat-6.0-doc/index.html`

2. Hacer un deployment de los Servicios-DLML, para esto necesitamos pegar el archivo ServicesDLML.war en la ruta relativa de apps del servidor Tomcat. En nuestro caso la ruta es:

   `C:\apache-tomcat-6.0.35\webapps`

3. Colocar la carpeta de configuración del portal sobre la unidad c:

   `C:\dlml_configuration`

4. Configurar los Servicios-DLML, por medio de los archivos de configuración:

   - Configurar la cuenta de correo para Cloud-DLML sobre el archivo:

     `mail.properties`

   - Configurar las conexiones a los clusters disponibles, en este caso se debe configurar un archivo para cada cluster disponible, como actualmente se hace uso de dos archivos, hay que configurar los archivos:

     ```
     pacifico.properties
     xena.properties
     ```

     correspondientes a cada cluster.
   - Configurar el archivo para mantener un registro de las ejecuciones de cada servicio para esto hay que editar el archivo:

     `log4j.properties`

5. Levantar el servidor Tomcat, para esto entramos a la carpeta:

   `C:\apache-tomcat-6.0.35\bin`

   posteriormente ejecutamos el archivo:

```
catalina.bat
```

que comenzara a levantar el Tomcat, en caso de estar en Linux hay que ejecutar el archivo catalina.sh.
Una vez que se ha levantado los servicios DLML quedaran listos para ser usados por el Portal-DLML para verificar que los servicios están levantados sobre el servidor verificar por medio de la URL:

```
http://ipserver:8080/ServicesDLML/services/servicegenericSOAP
```

Portal-DLML

1. Descargar el servidor web Tomcat puede ser la versión 6 en adelante, en este caso descargamos la versión 6 que se encuentra en la página web:

   ```
   http://tomcat.apache.org/tomcat-6.0-doc/index.html
   ```

2. Hacer un deployment del Portal-DLML, para esto necesitamos pegar el archivo PortalDLML.war en la ruta relativa de apps del servidor Tomcat. En nuestro caso la ruta es:

   ```
   C:\apache-tomcat-6.0.35\webapps
   ```

3. Colocar la carpeta de configuración del portal sobre la unidad c:

   ```
   C:\dlml_configuration
   ```

4. Configurar el Portal-DLML, por medio de los archivos de configuración:

   - Configurar la URL de los servicios en al archivo:

     ```
     servicios.properties
     ```

   - Configurar las conexiones a los clusters disponibles, en este caso se debe configurar un archivo para cada cluster disponible, como actualmente se hace uso de dos archivos hay que configurar los archivos

     ```
     pacifico.properties
     xena.properties
     ```

correspondientes a cada cluster.

5. Levantar el servidor Tomcat, para esto entramos a la carpeta:

```
C:\apache-tomcat-6.0.35\bin
```

posteriormente ejecutamos el archivo:

```
catalina.bat
```

que comenzara a levantar el Tomcat, en caso de estar en Linux hay que ejecutar el archivo catalina.sh.

6. Una vez que se ha levantado el Portal-DLML estará disponible por medio de la ruta relativa:

```
http://ipserver:8080/PortalDLML
```

Al terminar las configuraciones anteriores y los despliegues sobre los servidores correspondientes se podrá comenzar a usar Cloud-DLML.

Anexo B

Funciones núcleo

Hagamos un repaso de cada una de las funciones núcleo (core) de Cloud-DLML, son los peldaños para ofrecer los HPCaaS. Como ya se menciono anteriormente para agregar un servicio en esta nube, solo hay que seguir el patrón de diseño Factory para la parte del consumidor y para la parte del servicio, pero una vez agregado estos tiene que hacer uso de las funciones core de Cloud-DLML:
Parsers de Objetos, Cloud-DLML tiene 2 funciones core dedicadas a serializar y deserializar objetos de cualquier clase de tipo VO(Value Object), con estas funciones podemos tratar cualquier, con esto sólo se tiene que tener la clase del objeto del lado del Portal y del lado de los servicios para que se puedan hacer envió y recepción de cualquier objeto como si fueran simples cadenas y recuperar los objetos a partir de estas sin afectar su estado previo de cada objeto.
Para la parte de convertir un objeto a una cadena XML se encuentra el método `encoder`(figura 6.1) del `Serealizer` el cual recibe un objeto y lo transforma a una cadena XML, lista para ser enviada dentro del mensaje SOAP del WS genérico como simple string de java.

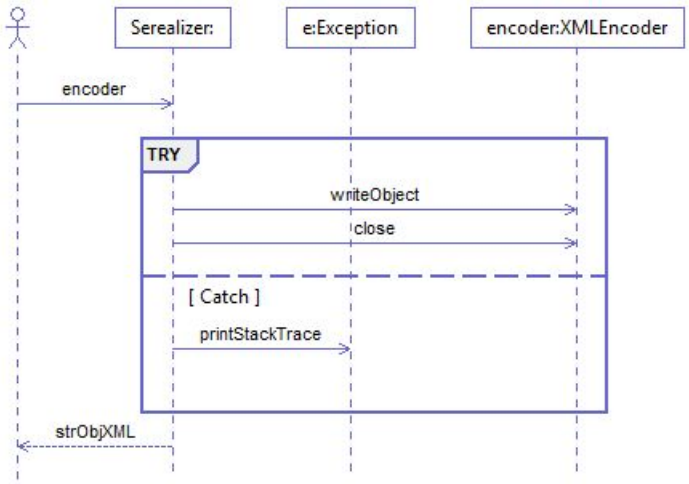

Figura 6.1: Diagrama de secuencia de la función encoder

Para la parte de recuperar el objeto con su estado previo a partir de la cadena XML se usa la función `decoder` (figura 6.2) que brinda el `Deserealizer` esta función método recibe una cadena XML y lo transforma a un objeto Java, posteriormente este objeto puede ser transformado a su clase correspondiente recuperándolo con el ultimo estado antes de ser convertido a cadena XML.

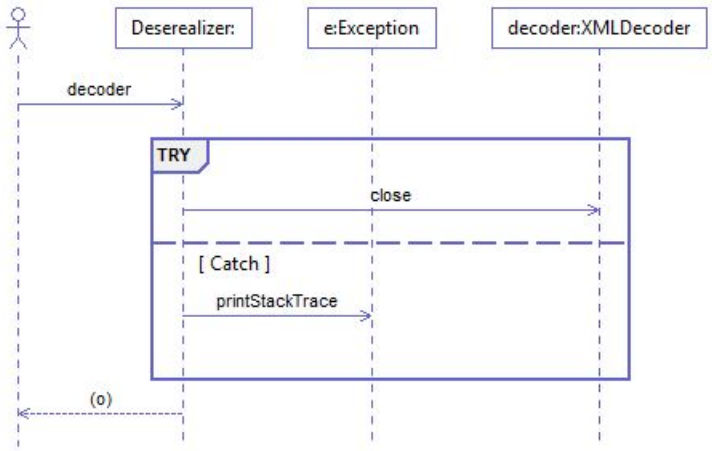

Figura 6.2: Diagrama de secuencia de la función decoder

Proxy Genérico, se tiene la clase core `ProxyGeneric` esta se implemento basada en el patrón de diseño de un Proxy[41], este patrón es de la familia de patrones estructurales, consiste en encapsular la funcionalidad para conectar y controlar objetos o métodos remotos, en este caso el `ProxyGeneric` consiste en conectar con los servicios DLML por medio de la función `connect`(figura 6.3), para esto el `ProxyGeneric` encuentra el punto de acceso a los servicios y posteriormente lo ejecuta, internamente a este se encuentra el motor de Axis que encapsula el empaquetado y des empaquetado del mensaje SOAP así como las localización del servicio.

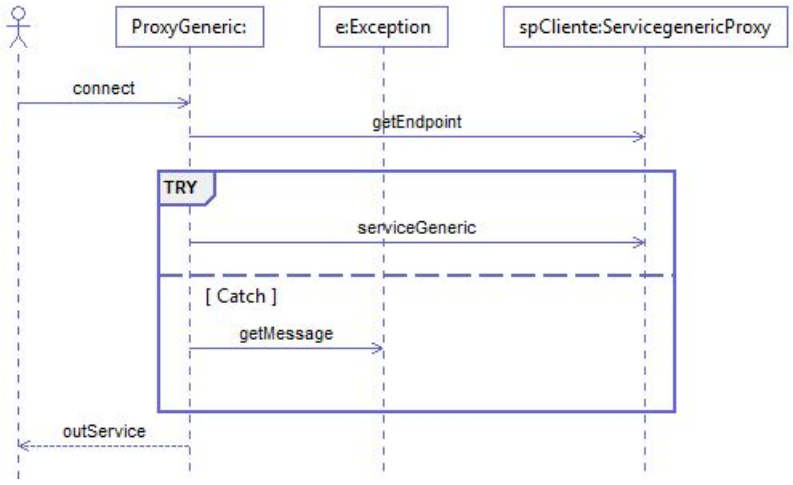

Figura 6.3: Diagrama de secuencia de la función connect

Conexiones SSH, se tiene las funcionalidades para llevar acabo las conexiones SSH, esta permiten conectarse por medio de una cuenta SSH a un cluster, para posteriormente poder ejecutar comandos sobre el Shell de este por medio de `executeCommand` (figura 6.4), recordando un poco los comandos que en general se manda a ejecutar por parte del sistema son script Expect, al igual que para abrir conexiones hay funciones para cerrar las conexiones.

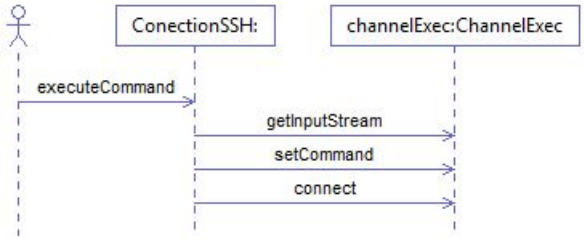

Figura 6.4: Diagrama de secuencia de la función executeCommand

Conexiones SFTP, también se implementaron funcionalidades para intercambio de archivos entre el sistema y los clusters por medio del protocolo SFTP, se cuentan con funciones para mandar archivos a un cluster, esto se hizo porque en general muchas de las aplicaciones requieren de archivos como es el caso de las aplicaciones multiplicación y Suma de Matrices es por esta razón, que en el core de Cloud-DLML también cuenta con estas funcionalidades, en este caso `ConectionSFTP` ofrece el método `putFile`(figura 6.5) para mandar un archivo desde el sistema hacia al cluster

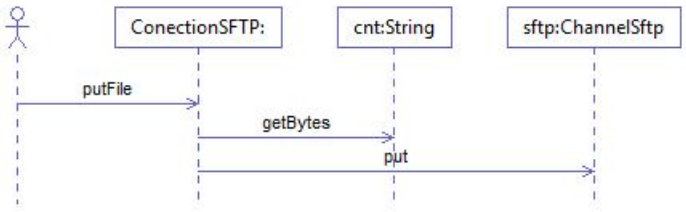

Figura 6.5: Diagrama de secuencia de la función putFile

Otra funcionalidad es la de obtener archivos del sistema de archivos de un cluster, como ya se ha mencionado anteriormente todas las aplicaciones tiene que mandar su salida a un archivo o a varios archivos, es por ende que el sistema por medio de `ConectionSFTP` ofrece el método `getFile` (Figura 6.20) que permite tomar al sistema un archivo del cluster.

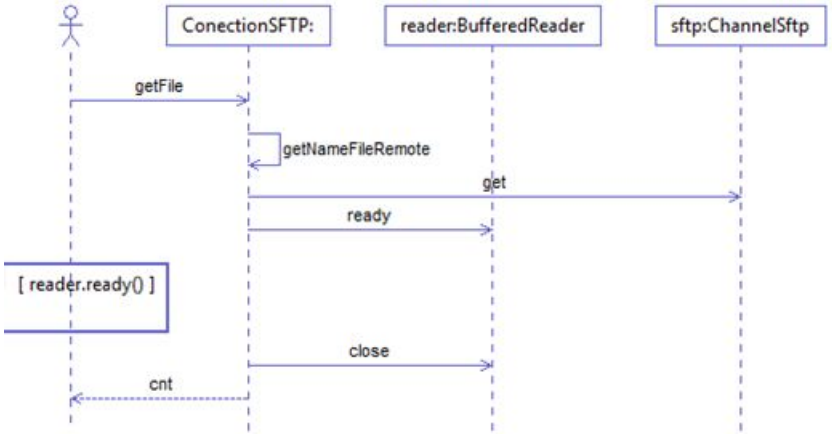

Figura 6.6: Diagrama de secuencia de la función getFile

Notificación de resultados por email, el core también cuenta con la funcionalidad para notificar los resultados los usuarios de los servicios de manera asíncrona, esta notificaciones se hacen por medio de correos electrónicos hacia las cuentas de los usuarios, para esto la clase `SendMail` ofrece el método `sendResult` (figura 6.7), hay que mencionar que esta cuenta de remitente es configurable mediante un archivo de propiedades.

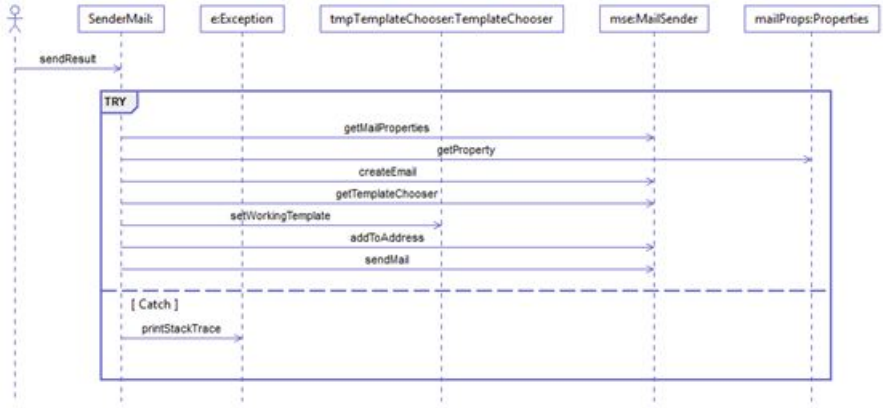

Figura 6.7: Diagrama de secuencia de la función sendResult

Cargar configuración de clusters, otra de las funcionalidades core, es la carga de la configuración de acceso a un cluster a partir de archivos de propiedades de un cluster, así como algunos otros datos iniciales para la conexión a estos.

Esto se hace por medio del método `initialize`(figura 6.8) de la clase `ClusterVO`, a través de un archivo de configuración, el cual contiene los datos suficientes para inicializar una sesión SSH o SFTP, además de la URL o IP del cluster, el acrónimo de los nodos del cluster, el número máximo de nodos en los cuales se puede ejecutar las aplicaciones DLML y la ruta de la carpeta asignada al sistema. Por cada cluster que usa Cloud-DLML, debe haber un archivo de propiedades correspondiente a dicho cluster.

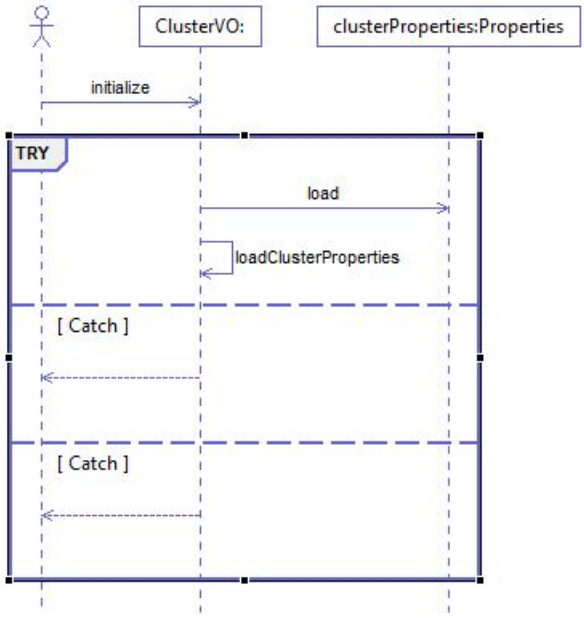

Figura 6.8: Diagrama de secuencia de la función initCluster

Todas las funcionalidades anteriores son pieza fundamental para el desarrollo de los servicios, estas son las partes granulares mas finas del sistema para las capas del Portal-DLML y Servicios-DLML a partir de este core se construyen toda la estructura de los servicios.

Anexo C

WSDL del WS Genérico

El siguiente XML, es WSDL del WS Genérico, a través de éste otros sistemas podrian hacer uso de los servicios de Cloud-DLML.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://dlml.services.hpc.edu/servicegeneric/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"  name="servicegeneric"
   targetNamespace="http://dlml.services.hpc.edu/servicegeneric/">
  <wsdl:types>
    <xsd:schema targetNamespace="http://dlml.services.hpc.edu/servicegeneric/">
      <xsd:element name="serviceGeneric" type="tns:inGeneric">
      </xsd:element>
      <xsd:element name="serviceGenericResponse"
       type="tns:outGeneric">
      </xsd:element>
      <xsd:complexType name="login">
       <xsd:attribute name="user" type="xsd:string"></xsd:attribute>
       <xsd:attribute name="password" type="xsd:string"></xsd:attribute>
      </xsd:complexType>
      <xsd:complexType name="inGeneric">
       <xsd:sequence>
    <xsd:element name="inAnything" type="xsd:string"></xsd:element>
    <xsd:element name="options" type="xsd:string"></xsd:element>
    <xsd:element name="authentication" type="xsd:string"></xsd:element>
       </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="outGeneric">
       <xsd:sequence>
    <xsd:element name="message" type="xsd:string"></xsd:element>
    <xsd:element name="outAnything" type="xsd:string"></xsd:element>
       </xsd:sequence>
      </xsd:complexType>
```

```
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="serviceGenericRequest">
    <wsdl:part element="tns:serviceGeneric" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="serviceGenericResponse">
    <wsdl:part element="tns:serviceGenericResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="servicegeneric">
    <wsdl:operation name="serviceGeneric">
      <wsdl:input message="tns:serviceGenericRequest"/>
      <wsdl:output message="tns:serviceGenericResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="servicegenericSOAP" type="tns:servicegeneric">
    <soap:binding
  style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="serviceGeneric">
      <soap:operation
   soapAction="http://dlml.services.hpc.edu/servicegeneric/serviceGeneric"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="servicegeneric">
    <wsdl:port binding="tns:servicegenericSOAP" name="servicegenericSOAP">
      <soap:address
  location="http://localhost:8080/ServicesDLML/services/servicegenericSOAP"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Secuencia de un HPCaaS

En un flujo de la ejecución de un servicio, hay un conjunto de módulos que interactúan entre sí, para dar como resultado los HPCaaS ofrecidos en Cloud-DLML. A continuación se presenta el diagrama de secuencia, genérico de un HPCaaS en Cloud-DLML (Figura 6.9).

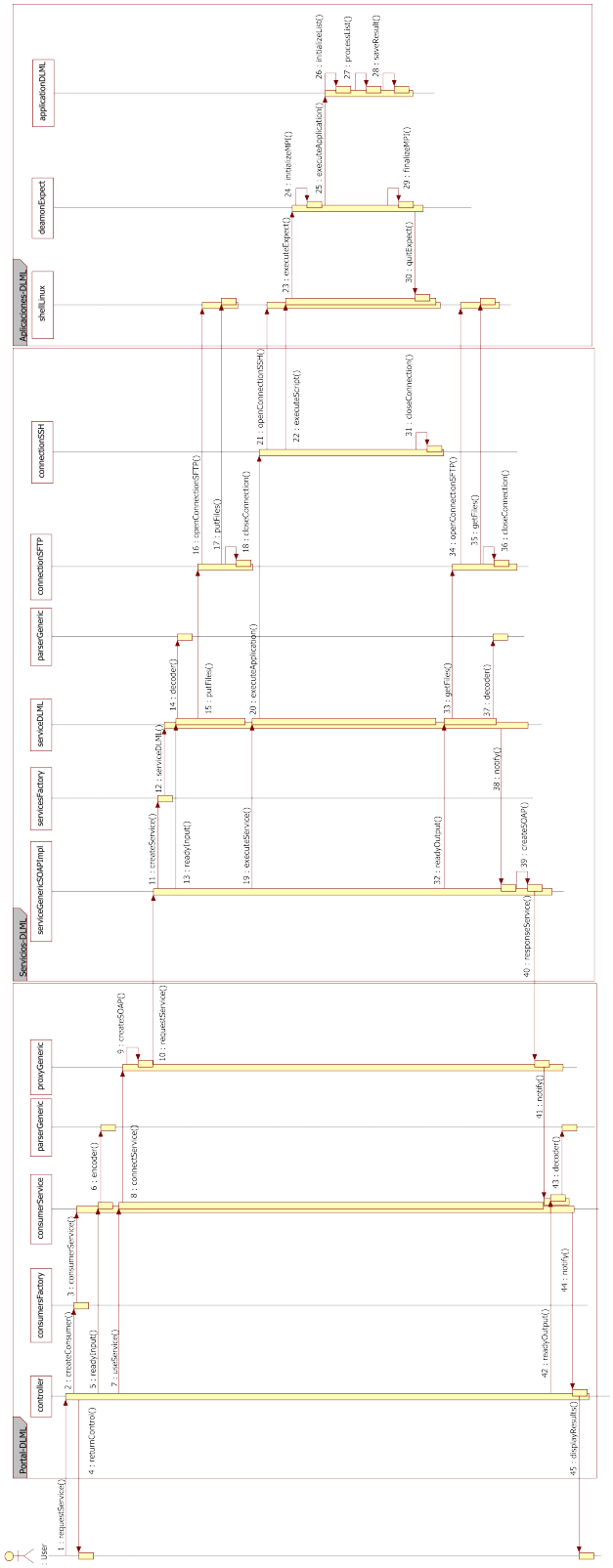

Figura 6.9: Diagrama de secuencia de un HPCaaS

Acrónimos

Cloud-DLML Cloud Computing para el ambiente de programación DLML

TI Tecnologás de la Información

HPC High Performance Computing

HPCaaS HPC as a Service

DLML Data List Management Library

SPMD Single Program, Multiple Data

ADT Abstract Data Type's

MPI Message Passing Interface

API Application Programming Interface

c Lenguaje de programación c

c++ Lenguaje de programación c++

GPU Graphics Processing Unit

SaaS Software as a Service

PaaS Platform as a Service

IaaS Infrastucture as a Service

SSL Secure Sockets Layer

SACS Security Access Control Services

SLA Service Level Agreement

ISO International Standards Organization

SO Sistema Operativo

CRM Customer Relationship Management

AWS Amazon Web Services

GAE Google App Engine

SOA Services Oriented Architecture

WS Web Service

SOCCA Service Oriented Cloud Computing Architecture

SOACC Service-Oriented Storage Resource Architecture

SOAP Simple Object Access Protocol

WSDL Web Service Description Lenguage

IDL Interface Description Lenguage

HTTP Hypertext Transfer Protocol

SMTP Send Mail Transfer Protocol

CPMD Car-Parrinello Molecular Dynamics

NAMD Nanoscale Molecular Dynamics

EA Enterprice Architecture

CCOA Cloud Computing Open Architecture

SEI Software Engineering Institute

QAW Quality Attribute Workshop

ADD Attribute-Driven Design

RIA Rich Internet Applications

Ajax Asynchronous Java Script and XML

HTML Hypertext Markup Lenguage

DHTM Dynamic HTML

XML eXtensible Markup Lenguage

JS Java Script

VO Value Object

TLS Transport Layer Secure

SSH Secure SHell

SFTP Secure File Transfer Protocol

URL Uniform Resource Locator

IP Internet Protocol

ATAM Method for Architecture Evaluation

Referencias

[1] Barrie Sosinsky, *Cloud Computing Bible*, Wiley 2011.

[2] Bob Corey, and John Johnson, *High Performance Tools & Technologies December, Michael Collette*, Computing Applications and Research Department Lawrence Livermore National Laboratory , 2004.

[3] Abhishek Gupta, Dejan Milojicic, *Evaluation of HPC Applications on Cloud*, IEEE 2011.

[4] Gilad Shainer, Tong Liu, Jeffrey Layton, Joshua Mora, *Scheduling Strategies for HPC as a Service (HPCaaS)*, IEEE 2009.

[5] Jeffrey Dean and Sanjay Ghemawat, *MapReduce: Simplifed Data Processing on Large Clusters*, IEEE 2010.

[6] *Whitepaper NVIDIA's Next Generation CUDA Compute Architecture: Fermi*, NVIDIA Corporation 2009.

[7] Mark Baker, *Cluster Computing White Paper*, University of Portsmouth , 2000.

[8] *http://www.top500.org/*, Top500 org. 2012, accesado en el 2012.

[9] Apolo H. Hernández Santos, Graciela Román Alonso, Miguel Alfonso Castro García, *DLML para un Ambiente Grid*, UAM-I, 2011.

[10] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, Steven Tuecke, Condor-G: A Computation Management Agent for Multi-Institutional Grids, IEEE, 2001.

[11] *MPI: A Message-Passing Interface Standard*, Message Passing Interface Forum 2010.

[12] Miguel Alfonso Castro García, *Programación con Listas de Datos para Cómputo Paralelo en Clusters*, CINVESTAV 2007.

[13] Werner Vogels, Greg Olsen, Lew Tucker, Greg Badros, Geir Ramleth, Steve Bourne,Mache Creeger, *CTO Roundtable: Cloud Computing*, ACM 2009.

[14] Bhaskar Prasad Rimal, Eunmi Choi, Ian Lumb, *Taxonomy and Survey of Cloud Computing System*, IEEE 2009.

[15] Wei-Tek Tsai, Xin Sun, Janaka Balasooriya, *Service-Oriented Cloud Computing Architecture*, IEEE 2010.

[16] Mikio Aoyama, Takashi Ikezaki, Noboru Nakamichi, *Attribute-Based Architecture Patterns for Lightweight Service-Oriented Architecture*, IEEE 2009.

[17] Jinesh Varia Technology Evangelist, *Cloud Architectures*, Amazon Web Services 2010.

[18] Wei-Tek Tsai, Xin Sun, Janaka Balasooriya, *Service-Oriented Cloud Computing Architecture*, IEEE 2010.

[19] Zhonglei Fan and Xiangmo Zhao, *Service-Oriented Storage Resource Architecture for Cloud Computing*, IEEE 2010.

[20] Manish Pokharel, YoungHyun Yoon, Jong Sou Park, *Cloud Computing in System Architecture*, IEEE 2009.

[21] Liang-Jie Zhang and Qun Zhou, *CCOA: Cloud Computing Open Architecture*, IEEE 2009.

[22] Ramgovind S, Eloff MM, Smith E, *The Management of Security in Cloud Computing*, IEEE 2010.

[23] Jhon W. Rittinghouse, James F. Ransome, *Cloud Computing implementation, managment and security*, CRC Press 2010.

[24] Axel Buecker, Koos Lodewijkx, Harold Moss, Kevin Skapinetz, Michael Waidner, *Cloud Security Guidance IBM Recommendations for the Implementation of Cloud Security*, IBM Corporation 2009.

[25] Zeng Shu-Qing and Xu Jie-Bin, *The Improvement of PaaS Platform,*, IEEE 2010.

[26] Jinesh Varia, *Arquitectura para la nube: Prácticas recomendadas*, Amazon 2011.

[27] *Guía para la seguridad de áreas criticas de atención en Cloud Computing*, CSA 2009.

[28] Michael P. Papazoglou and Willem-Jan van den Heuvel, *Web Services Management: A Survey*, IEEE 2005.

[29] *The Sales Cloud*, SalesForce 2011.

[30] *http://code.google.com/intl/es-ES/appengine*, Google Inc. 2011, accesado en el 2011.

[31] *Overview of Amazon Web Services*, Amazon Web Services 2010.

[32] Roy Campbell, Indranil Gupta, Mike Heath, Steve Ko, Michael Kozuch, Marcel Kunze, Thomas Kwan, Kevin Lai, Hing Yan Lee, Martha Lyons, Dejan Milojicic, David O'Hallaron, Yeng Chai Soh, *Open Cirrus Cloud Computing Testbed: Federated Data Centers for Open Source Systems and Services Research*, Open Cirrus 2009.

[33] C. Joseph, Steve Conway, Jie Wu, *A New Approach to HPC Public Clouds: The SGI Cyclone HPC Cloud*, IDC 2010.

[34] Philip Church, Adam Wong, Michael Brock, Andrzej Goscinski, *Toward Exposing and Accessing HPC Applications in a SaaS Cloud*, IEEE 2012.

[35] Robert L. Nord, William G. Wood, Paul C. Clements, *Integrating the Quality Attribute Workshop (QAW) and the Attribute-Driven Design (ADD) Method*, SEI 2004.

[36] Greg Murray, *http://www.oracle.com/technetwork/articles/javaee/ajax-135201.html*, Sun Microsystems 2006, accesado en el 2012.

[37] *http://www.dhtmlgoodies.com*, 2011, accesado en el 2012.

[38] Eric Freeman, Elisabeth Freeman, *Head First Design Patterns, Eric Freeman*, O'Reilly 2004.

[39] *http://axis.apache.org/axis/*, The Apache Software Foundation 2005, accesado en el 2011.

[40] *http://www.nist.gov/el/msid/expect.cfm/*, NIST 2010, accesado en el 2012.

[41] *http://www.jcraft.com/jsch/*, JCraft Inc. 2012, accesado en el 2012.

[42] Rick Kazman, Mark Klein, Paul Clements, *ATAM:Method for Architecture Evaluation*, SEI 2000.

[43] *https://www.windowsazure.com/en-us/develop/net/fundamentals/intro-to-windows-azure/*, Windows Azure 2012, accesado en el 2012.

[44] *Windows Azure: Java, Eclipse y Azure*, Windows Azure 2011.

Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA

Cloud Computing para el ambiente de programación DLML

Idónea comunicación de resultados para
obtener el grado de
Maestro en Ciencias
(Ciencias y Tecnologías de la Información)
por:
Abraham Martínez Ramírez

Asesores:
Dr. Miguel Alfonso Castro García
Dr. Manuel Aguilar Cornejo

Jurado Calificador:

Presidente:	**Dr. José Guadalupe Rodríguez García**	**CINVESTAV**
Secretario:	**Dr. Miguel Alfonso Castro García**	**UAM-I**
Vocal:	**Ing. Luis Fernando Castro Careaga**	**UAM-I**

UNIVERSIDAD AUTÓNOMA METROPOLITANA - IZTAPALAPA
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

México D.F. 12 de julio de 2013