PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DESARROLLO DE UN FRAMEWORK WEB PARA EL ENVÍO REMOTO DE TAREAS, MONITOREO Y RECUPERACIÓN DE RESULTADOS PARA DESKTOP GRIDS USANDO UNA ARQUITECTURA ORIENTADA A SERVICIOS: CASO BOINC

Tesis para optar por el Título de Ingeniero Informático, que presenta el bachiller:

Pablo Alejandro Fonseca Arroyo

ASESOR: Genghis Ríos Kruger

Lima, 29 de enero del 2014

Historial de Revisiones

Historial de revisiones				
Ítem	**Fecha**	**Versión**	**Descripción**	**Equipo**
1	04/04/10	1.0	Versión inicial.	Pablo Fonseca
2	05/05/10	1.1	Agregados	Pablo Fonseca
3	14/07/10	2.0	Correcciones finales correspondientes al curso.	Pablo Fonseca
4	12/09/11	3.0	Actualizaciones mayores después de la versión 1.0 de Legión Framework presentada en el BOINC Workshop 11.	Pablo Fonseca
5	18/11/11	3.1	Actualizaciones después de la liberación de la versión 1.1 de Legión Framework.	Pablo Fonseca
6	13/09/12	4.0	Actualizaciones después de la liberación de la versión 1.4 de Legión Framework.	Pablo Fonseca
7	29/01/14	4.1	Versión Final	Pablo Fonseca

Tabla de Contenidos

1. Generalidades....8
1.1. Definición del problema....8
1.2. Marco Conceptual....10
1.2.1. Computación en Grid....10
1.2.2. Desktop Grids y Cómputo Voluntario....10
1.2.3. BOINC....11
1.3. Estado del Arte....13
1.3.1. Visu@lGrid....13
1.3.2. App Wrapper....13
1.3.3. Rboinc (GPU Grid)....14
1.3.4. Comparación....15
1.4. Plan de Proyecto....15
1.4.1. SCRUM....15
1.4.2. Análisis de la adaptabilidad de SCRUM....17
1.4.3. Adaptación de SCRUM al proyecto....17
1.4.4. Línea de tiempo del proyecto....18
1.5. Descripción y sustentación de la solución....22
1.5.1.Descripción de la solución....22
1.5.2.Costo del desarrollo....23
2. Análisis....24
2.1. Metodología aplicada para el desarrollo de la solución....24
2.1.1. Rational Unified Process (RUP)....25
2.1.2.Agile Unified Process (AUP)....25
2.1.3.Extreme Programming (XP)....27
2.1.4. Comparación....28
2.1.5. Adaptación de la metodología escogida....28
2.2.Identificación de requerimientos....29
2.2.1.Listado de Requerimientos....29
2.3.Análisis de la Solución....32
2.3.1.Usuarios del sistema....32
2.3.2.Nociones del sistema....34
2.3.3.Capa Web....34
3. Diseño....39
3.1.Arquitectura de la solución....39
3.2.Legión Web Interface 41
3.2.1.Template Interpreter....41
3.2.2.TaskCreator....45
3.2.3.FormRenderer....45
3.2.4.Editor de formularios del lado del cliente: “Legion Form Editor”....46
3.2.5. Web Interface....48
3.3. Legión Web Services....48
3.3.1.Interacción del sistema....48
3.3.2.Lineamientos para interacción de las capas....48
3.4. Diseño de interfaz gráfica....49
3.4.1. Lineamientos de interfaz gráfica....49
3.4.2. Prototipos de interfaz gráfica....50
3.5. Arquitectura de información....51
3.5.1. Representación de formularios....51

3.5.2. Especificación del descriptor de tareas....51
3.5.3. Especificación de servicios web....53
4. Construcción y pruebas....55
4.1.Construcción....55
4.1.1.Selección de tecnología para el cliente....55
4.1.2.Selección de tecnología para Servicios Web....56
4.1.3.Selección de tecnología para Servicios Web SOAP....56
4.1.4.Comparación con Servicios Web REST....57
4.1.5. Selección de tecnología para la interfaz web....58
4.1.6.Implementación de un framework web simplificado....59
4.1.7.Implementación del Template Interpreter....60
4.1.8.Implementación del Task Creator....61
4.1.9.Implementación de un hilo de creación de tareas....61
4.1.10.Implementación de un hilo de actualización de tareas....62
4.1.11.Implementación de un hilo de eliminación de tareas....62
4.1.12.Ventajas del lenguaje de programación escogido....63
4.2.Pruebas....64
4.2.1.Pruebas de integración....64
5. Observaciones, conclusiones y recomendaciones....68
5.1.Observaciones....68
5.2.Conclusiones....69
5.3.Recomendaciones y trabajos futuros....69
5.3.1.Barrido de parámetros en Legión Web Interface....69
5.3.2.Uso de técnicas de Machine Learning para predecir tiempo de ejecución....70
5.3.3.Analizar integración con OurGrid....70
5.3.4.Analizar integración con Condor....70
5.3.5.Implementar un Account Manager para trabajar con Legión....70
5.3.6.Implementar una Grid Nacional / Latinoamericana....70
Bibliografía....72
Anexos....75

Índice de imágenes

Figura 1.1: Interacción en BOINC [Ries 2010].....9
Figura 1.2: Interacción de componentes en BOINC [Berkeley 2010].....12
Figura 1.3: Visual Grid.....13
Figura 1.4: App Wrapper [Nikitina 2011].....14
Figura 1.5: Arquitectura de RBoinc [Giorgino 2010].....14
Figura 1.6:Esquema de trabajo según SCRUM [Fuente propia].....17
Figura 1.7:Línea de tiempo del proyecto.....22
Figura 2.1:AUP [Ambler 2010].....26
Figura 2.2: XP [Wells 2000] 27
Figura 2.3: Ámbito del sistema.....33
Figura 2.4: Ámbito del proyecto.....33
Figura 2.5: Ámbito de la tarea.....33
Figura 2.6: Diagrama de Casos de Uso – Paquete Administración.....35
Figura 2.7: Diagrama de clases de análisis (Capa Legión Web Interface).....37
Figura 2.8: Diagrama de estados – Clase Task.....38
Figura 3.1:Diagrama de la arquitectura del sistema.....40
Figura 3.2:Funcionamiento del Template Interpreter.....43
Figura 3.3:Diseño simplificado del Template Interpreter.....44
Figura 3.4: Diseño de TaskCreator.....45
Figura 3.5: Diseño de Form Renderer.....46
Figura 3.6: Diseño de Form Editor.....47
Figura 3.7: Prototipo de pantalla principal.....47
Figura 3.8:Diagrama de paquetes de LWI.....48
Figura 3.9: Pantalla del sistema 49
Figura 3.10:Distribución de la pantalla.....50
Figura 3.11:Procedimiento para crear descriptor de tareas.....53
Figura 4.1:Proceso de creación de tareas.....61
El account manager podría ser un sistema independiente que brinde servicios web para Legión para conocer: Cantidad de nodos, porcentajes asignados, etc.....70
Figura 5.2: Propuesta de arquitectura 71

Índice de Tablas

Tabla 1.1:Comparación de Aplicaciones.....15
Tabla 1.2 : Costo del proyecto.....23
Tabla 1.3: Costo del desarrollo de una interfaz después del proyecto.....23
Tabla 2.1: Comparación de metodologías.....28
Tabla 2.2: Resultado de comparación de metodologías.....28
Tabla 2.3: Requerimientos funcionales.....31
Tabla 2.4: Requerimientos no funcionales.....32
Tabla 3.1: Lista de métodos de la capa LWS.....54
Tabla 5.1:Mapeo de REST sobre los métodos HTTP.....58

1. Generalidades

El presente capítulo muestra el problema que a resolver de manera detallada. Adicionalmente, se revisan los principales intentos de solución a nivel internacional. Posteriormente, se desarrolla la solución propuesta y se explica porqué resulta importante, viable y útil tanto en el contexto universitario de la Pontificia Universidad Católica del Perú como globalmente para los usuarios de sistemas de cómputo de alto rendimiento, en especial los de cómputo voluntario.

1.1. Definición del problema

Hoy en día los científicos necesitan de aplicaciones de cómputo para realizar cálculos, validar modelos matemáticos o hacer simulaciones; requiriendo para esto del procesamiento de un gran volumen de datos. En ocasiones, los requerimientos computacionales para realizar estas tareas exceden las prestaciones que les puede brindar una computadora personal. En vista de esto, es necesario contar con una infraestructura de mayor escala, la cual permita realizar cálculos intensivos.

Hay muchas alternativas para implementar la infraestructura necesaria. Por un lado, se puede adquirir una supercomputadora (con altas prestaciones de memoria, procesador) pero a un alto costo. El otro camino viable es construir un sistema de Grid o un Cluster a partir de hardware de bajo costo y que se encuentre ampliamente disponible. Escoger una u otra opción depende de las necesidades particulares de la organización, así como del presupuesto disponible. Si se opta por la segunda opción, existe software open-source para gestionar los recursos del sistema, programar tareas computacionales y ejecutarlas. Un ejemplo de este tipo de sistemas middleware es Berkeley Open Infraestructure for Network Computing (BOINC) desarrollado para dar soporte al cómputo voluntario siendo ampliamente usado en el mundo.

Si bien es cierto que las operaciones de bajo nivel son delegadas a software intermedio como BOINC, no se debe dejar de tener en cuenta que la configuración y uso de estos no son tareas triviales. Enviar una tarea a ejecutarse a la Grid es poco intuitivo y demanda un alto nivel técnico. Para ilustrar la complejidad, un diagrama de las actividades necesarias para crear un proyecto en BOINC (previo al envío de tareas) se muestra en la Figura 1.1. Por esta razón, una capa de abstracción es necesaria para ocultar al científico las complejidades del sistema. Esta capa podría presentarse como una interfaz que permita al usuario enviar tareas, monitorear su avance y recuperar los resultados.

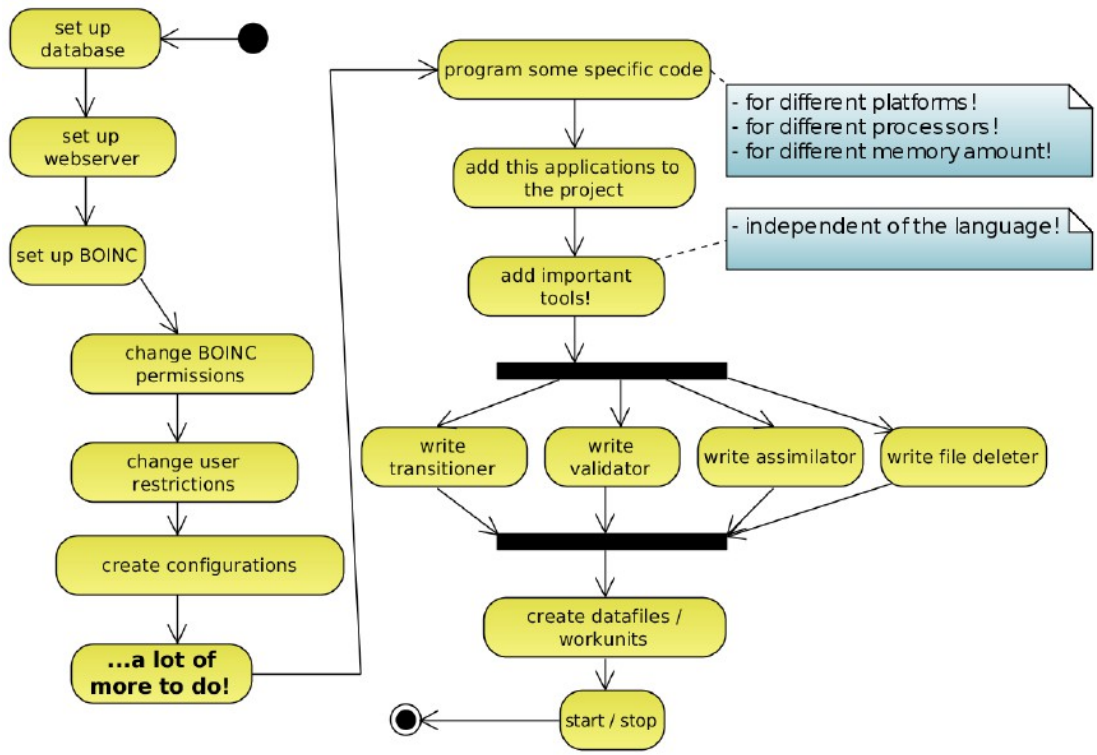


Figura 1.1: Interacción en BOINC [Ries 2010]

Los puntos mencionados anteriormente evidencian un primer intento de solución que fueron plasmados en la primera versión del sistema Legión [Ríos 2009]. Sin embargo, dependiendo del grado de especificidad de la interfaz, podría existir un problema mayor -el desarrollo de ésta- ya que cada proyecto de investigación puede ser muy diferente. Además, estas deben ser lo suficientemente específicas para evitar fallas de seguridad e impedir que los usuarios cometan errores involuntarios. Cuanto más específica es la interfaz, mayor trabajo implica por parte de la unidad u organización que administra el sistema de cómputo. Las estadísticas de tiempos de desarrollo en el área de Infraestructura de la Dirección de Informática Académica (DIA) de la Pontificia Universidad Católica del Perú (PUCP) parecen indicar que esta tarea tiene un alto costo, ya que tomaba más del 50% del tiempo total de adaptación de los programas para que puedan ser ejecutados en el sistema Grid, lo que dificulta la atención a nuevos usuarios de cómputo intensivo.

Resumiendo, el problema radica principalmente en la dificultad que representa desarrollar las interfaces que tendrían el propósito de ocultar las complejidades del sistema a los usuarios finales. En la PUCP, la infraestructura y el software de supercómputo se denominan en conjunto “Sistema Legión”. Este sistema debe suplir los recursos de cómputo intensivo para la comunidad universitaria y ser de fácil uso tanto para los administradores del sistema, como para los usuarios finales.

1.2. Marco Conceptual

1.2.1. Computación en Grid

Existen varias definiciones de lo que es y de lo que debería ser una Grid; sin embargo, se tomará la siguiente definición para los propósitos del presente trabajo. Una Grid permite unir muchos recursos computacionales distribuidos geográficamente para tratar problemas grandes y trabajos como si todos los servidores y recursos estuvieran en un solo sitio [Abbas 2004]. A diferencia de los Clusters, una Grid podría incluir recursos computacionales en distintas ubicaciones y pertenecientes a diferentes organizaciones [Emmen 2010]. Incluso, algunos de estos recursos podrían ser Clusters [Anderson 2004]. El principal problema con las Grids ocurre precisamente cuando se cruzan las barreras organizacionales.

1.2.2. Desktop Grids y Cómputo Voluntario

La intención original de los investigadores en el área de Grid, según enuncian Peter Kacsuk et al en [Kacsuk 2006] era que cualquiera pudiera unirse a una Grid ofreciendo sus recursos, así como solicitarlos dinámicamente para algún requerimiento de procesamiento intensivo. Hay dos caminos que se han seguido, por un lado están los Grids dedicados donde el hardware es administrado por una organización y tiene una gran cantidad de usuarios. El segundo camino, un grupo de usuarios limitado, donde cualquiera puede unirse para donar recursos computacionales. Está última definición pertenece a las Desktop Grids.

Existen dos tipos de Desktop Grids según Emmen [Emmen 2010], las Desktop Grids de voluntariado, donde los recursos son generalmente computadoras en hogares y las Desktop Grid organizacionales como el sistema Legión [Ríos 2009] de la Pontificia Universidad Católica o la Desktop Grid Local de la Universidad de Westminster (WLDG) usando las computadoras con las que ya contaban las instituciones mencionadas. Winter [Winter 2011] estima que los 1800 nodos que conforman WLDG equivaldrían aproximadamente a un cluster que implicaría una inversión de 500,000 libras esterlinas y cita otras ventajas: Alta disponibilidad, vida más larga, actualización continua y facilidad de uso.

Anderson en [Anderson 2004] contrasta a las Grids tradicionales con el cómputo de recursos voluntarios. Indica que mientras las Grids implican recursos que son propiedad de las organizaciones participantes (que pueden ser supercomputadoras, clusters o computadoras comunes) siguen siendo administrados por profesionales de tecnologías de información, por lo que son generalmente confiables. Por otro lado, en el cómputo voluntario los recursos pertenecen a donantes anónimos dispersos mundialmente, donde no se puede asumir necesariamente que sean recursos confiables, ni se puede garantizar su disponibilidad. Diversos estudios sobre la disponibilidad han sido realizados, incluído los realizados por Andrezjak [Andrzejak 2010A] y [Andrzejak 2010B] donde utiliza técnicas estadísticas para analizar la disponibilidad de recursos volátiles en el contexto de acuerdos de nivel de servicios.

Una Desktop Grid presenta una ventaja significativa debido a las características mencionadas. El beneficio que salta a la luz inmediatamente es que puede escalar rápidamente a un poder de cómputo inmenso manteniendo bajos costos. Sin embargo, esta solución aparentemente adecuada, presenta una limitación: tradicionalmente brinda solución solo a los problemas que pueden resolverse bajo un paradigma master-worker [Kacsuk 2006] o del tipo bag-of-tasks. Sin embargo, en la Universidad de Houston se está desarrollando Volpex, un framework para proveer un soporte de similar a MPI [Rohit 2011] en un ambiente de cómputo voluntario como BOINC.

1.2.3. BOINC

BOINC es un sistema middleware para Desktop Grid o Cómputo de Recursos Públicos como lo enuncia Anderson en [Anderson 2004]. El uso de ambos términos para referirse a software como BOINC es intercambiable. Este sistema presenta una arquitectura con dos componentes principales que interaccionan entre si, llevando a cabo el cómputo de las tareas o unidades de trabajo.

El primer componente es el servidor, el cual se encarga de recibir las tareas de parte del usuario y distribuirlas entre los nodos de ejecución; de coordinar la comunicación de red necesaria tanto para el envío de tareas y archivos de entrada; y para la recuperación de resultados. El servidor es accesible a través del protocolo

HTTP, que por lo general, es de libre acceso tanto en redes domésticas como empresariales. Un servidor BOINC puede albergar uno o más proyectos.

El segundo componente es el cliente, el cual se instala en los nodos de ejecución. Este tiene versiones para varias plataformas conocidas como Windows, Mac OS X y Linux. Incluso existen clientes para consolas de juego ya que poseen un alto poder de cómputo. El cliente es el que inicia la comunicación, solicitando tareas al servidor para ejecutarlas. Este tipo de comunicación presenta la ventaja de que el servidor no tiene que monitorear el estado de los clientes ya que estos pueden ser muchos. Es importante notar que un cliente puede estar relacionado con más de un proyecto BOINC, además permite especificar que porcentaje de los recursos libres (CPU y memoria) se asigna a cada proyecto al que se encuentra suscrito.

La Figura 1.2 muestra de manera general como sucede la comunicación entre estos dos componentes. La PC cliente solicita tareas al servidor, el servidor responde con instrucciones y se realiza la descarga de las aplicaciones y archivos de entrada. Luego, se ejecuta el cálculo correspondiente a la unidad de trabajo asignada. Al terminar, se suben los archivos y reportan los resultados.

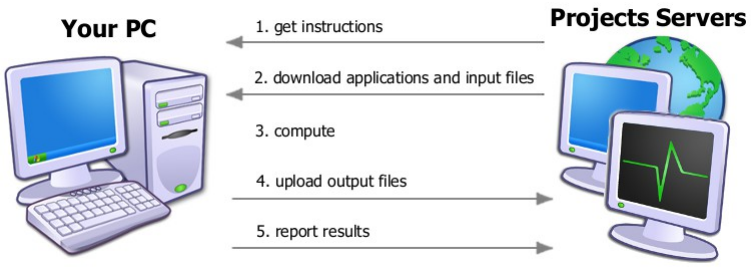


Figura 1.2: Interacción de componentes en BOINC [Berkeley 2010]

BOINC presenta una API tanto para aplicaciones cliente, como para aplicaciones del servidor. Para comenzar, en la aplicación cliente, es necesario construir un programa que haga el procesamiento necesario y a la vez informe al cliente BOINC de su estado. Para conseguir esto se pueden seguir dos caminos: desarrollar una aplicación nativa o usar un contenedor. Se puede desarrollar una aplicación que haga uso de la API de BOINC, para esto existen librerías en C++ y Python; sin embargo, esto no es siempre factible, ya que no se dispone en todos los casos del código fuente de esta que se requiera para ejecutar. La otra posibilidad es usar una aplicación denominada wrapper que se encarga de hacer las llamadas a la API de BOINC y ejecutar internamente la aplicación requerida (de manera práctica realiza una llamada “exec”). Esto implica que el ejecutable sea un programa desarrollado y compilado por un tercero. Este último caso es el más extendido en los proyectos de los científicos de la PUCP. Sin embargo se han presentado casos que la aplicación a ejecutarse necesita ser recompilada en el mismo cliente: esto también es factible utilizando el componente wrapper genérico provisto por BOINC. Cabe resaltar que

también hay wrappers desarrollados por terceros como VMWrapper [Segal 2010] desarrollado por el equipo de Ben Segal en el CERN.

1.3. Estado del Arte

Para tener un panorama claro respecto al estado del arte es necesario un análisis de los principales intentos internacionales y nacionales de desarrollo de interfaces para BOINC. Por este motivo se ha seleccionado algunos casos particulares: Visu@IGrid, App Wrapper, Rboinc.

1.3.1. Visu@IGrid

Visu@IGrid [Ries 2010] tiene por objetivo principal ser un IDE para desarrollar aplicaciones que se ejecuten en BOINC por medio de modelado textual o gráfico con generación completa de código dentro de un marco de proceso de Model Driven Engineering. En la figura 1.3 se puede observar una pantalla del sistema.

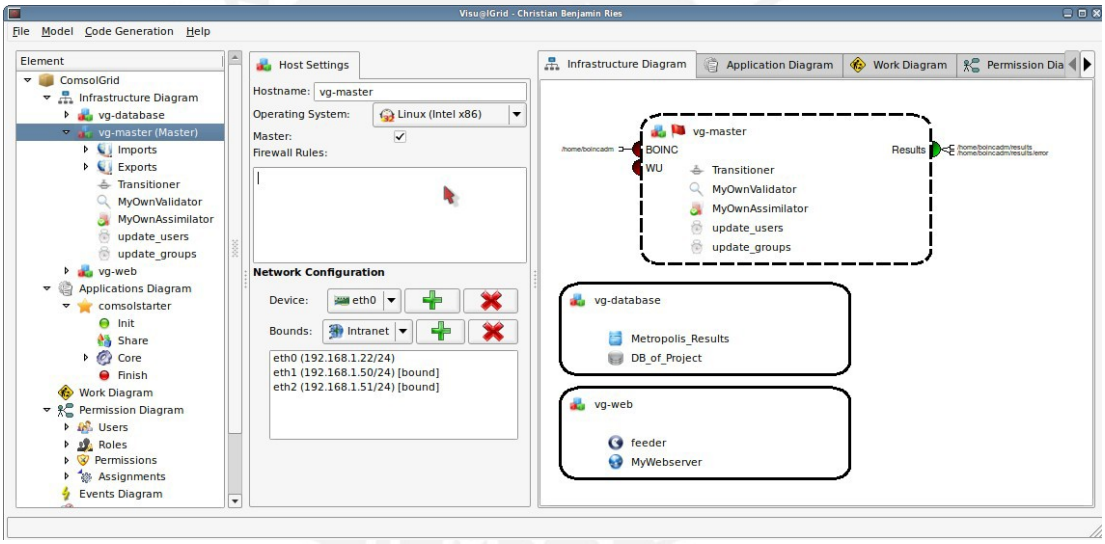


Figura 1.3: Visual Grid

La mayor contribución de este proyecto es proveer a BOINC con una generación de proyectos automatizada (mediante modelado gráfico y textual). Al momento de escribir el presente documento, el proyecto no estaba terminado y formaba parte de la tesis doctoral de Christian Benjamin Ries.

1.3.2. App Wrapper

App Wrapper es un proyecto del Institute of Applied Mathematical Research de Karelian Research Centre de la Academia Rusa de Ciencias. Este sistema fue presentado por primera vez en el BOINC Workshop 11 [Nikitina 2011]. Su objetivo principal es realizar la creación remota de aplicaciones en un servidor BOINC mediante una interfaz web (que puede ser observada en la figura 1.4).

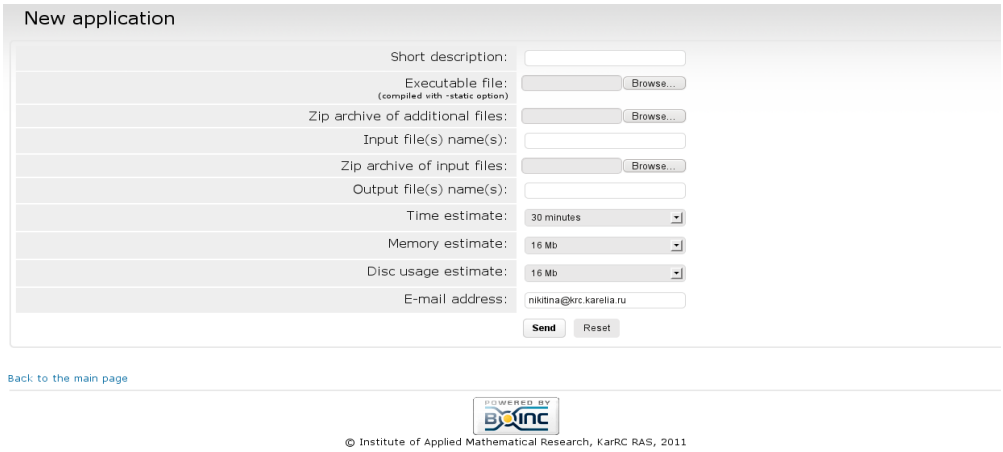


Figura 1.4: App Wrapper [Nikitina 2011]

1.3.3. Rboinc (GPU Grid)

RBoinc es una interfaz para los científicos que extiende las funcionalidades de BOINC, permite el envío y monitoreo de las tareas de manera remota. Presenta dos componentes: un cliente y un servidor (ambos escritos en Perl). La coordinación del proyecto BOINC decidió incluirlo como parte del núcleo [Giorgino 2010]. Este sistema fue presentado por primera vez en el BOINC Workshop 09. Cabe resaltar que Rboinc ya no está siendo actualizado tan constantemente.

Características de RBoinc

- Permite envío remoto de tareas a un servidor BOINC.
- Permite monitorear el estado de las tareas.
- Arquitectura Cliente – Servidor.
- Interfaz de línea de comandos.
- El servidor funciona como una instancia del servidor web Apache.

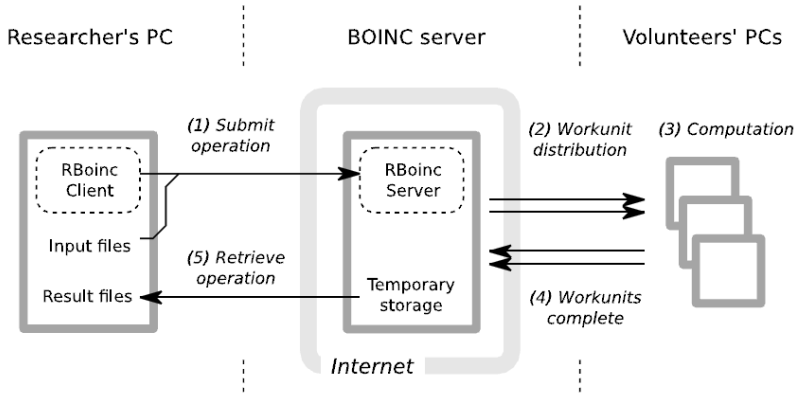


Figura 1.5: Arquitectura de RBoinc [Giorgino 2010]

1.3.4. Comparación

Para realizar la comparación usaremos una tabla de doble entrada evaluando ciertas características.

	Visu@lGrid	App Wrapper	RBoinc
Interfaz/Portal Web	No	Sí	No
Interfaz de Línea de comandos	No	No	Sí
Interfaz de escritorio	Sí	No	No
Lenguaje de programación	C++	PHP	Perl
Envío remoto de tareas	No	Sí	Sí
Creación asistida de aplicaciones	Sí	Sí	No
Licencia	Privado (por el momento)	Privado	Open Source

Tabla 1.1:Comparación de Aplicaciones

1.4. Plan de Proyecto

El presente proyecto se realiza en el contexto de una organización de educación superior; de manera concreta en una oficina administrativa del área de informática académica. Debido a esto, se debe tomar en cuenta los lineamientos que se establecen en la organización para la gestión de proyectos. En la Dirección de Informática Académica de la Pontificia Universidad Católica del Perú se utiliza SCRUM para la gestión de proyecto; sin embargo, es conveniente evaluar si esta metodología es adecuada para las características del presente proyecto.

1.4.1. SCRUM

SCRUM señala 3 roles principales, siendo estos Product Owner, ScrumMaster y equipo.

Product Owner (Dueño del producto)

El dueño del producto es quien financia el proyecto con el fin de obtener un resultado esperado. Si se compara con PMBOK, este rol se puede *mapear* al de Sponsor (Patrocinador).

ScrumMaster

El ScrumMaster es el encargado de gestionar todo el proceso, de enseñar Scrum a los involucrados en el proyecto, de implementar Scrum de acuerdo a las características de la organización y de velar para que se cumplan los lineamientos establecidos [Schwaber 2004].

Equipo

Es la unidad responsable de desarrollar las funcionalidades, debe ser auto-manejado y auto-organizado.

Los artefactos de SCRUM

Product Backlog

Al comenzar un proyecto se listan los requerimientos funcionales y no funcionales en el Product Backlog. No se asume que se encuentra completo, más bien que se trata de un documento dinámico, lo que favorece el desarrollo en un contexto de cambios constantes.

Sprint Backlog

El Sprint Backlog define el trabajo o tareas que es necesario realizar para lograr un producto funcional al final de un Sprint.

Incremento de una funcionalidad del Producto Potencialmente entregable

Al finalizar un Sprint se debe entregar un incremento que debe ser funcional y potencialmente entregable. Es por esto que debe estar probado y bien estructurado.

Flujo de SCRUM

Un proyecto basado en SCRUM comienza con la visión del sistema que quiere ser desarrollado. Esta visión se plasma en el Product Backlog, que es un listado "dinámico" de los requerimientos. Se realiza la estimación del Product Backlog para cuantificar el trabajo. A continuación se ingresa a la parte iterativa, todo el trabajo se hace en Sprints, donde cada Sprint tiene una duración variable de dos semanas a dos meses, aunque se recomienda un mes. Se seleccionan las características más importantes para realizarse y se estima junto al equipo qué parte se podrá completar en un Sprint. El Sprint genera un incremento en el producto, y tras este se realiza una presentación del incremento, lo que permite obtener feedback del Product Owner. Este esquema de trabajo se puede observar en la figura 1.6.

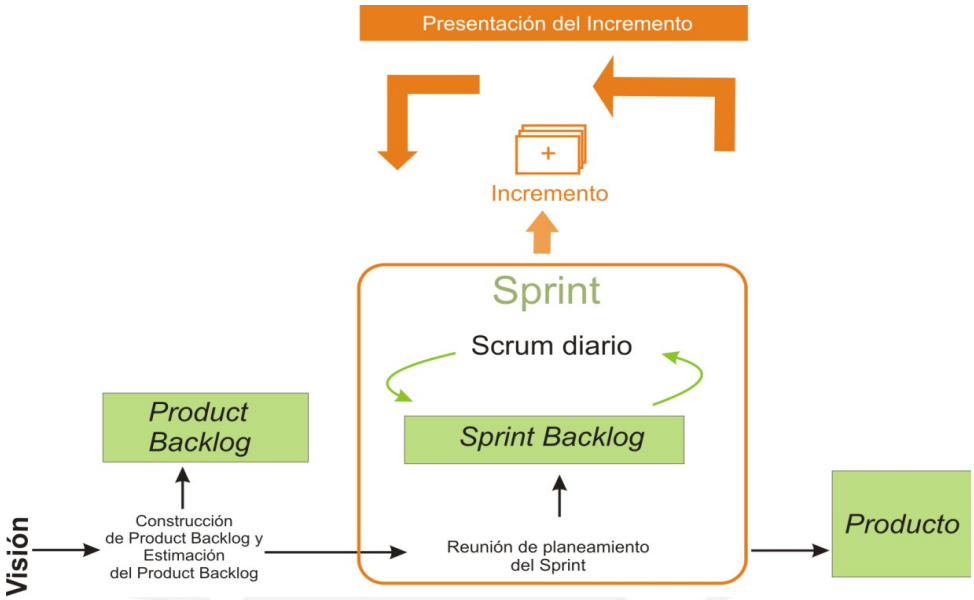


Figura 1.6:Esquema de trabajo según SCRUM [Fuente propia]

1.4.2. Análisis de la adaptabilidad de SCRUM

La metodología ágil SCRUM tiene algunas características que la hacen adecuada para este tipo de proyecto.

- Manejo dinámico de requerimientos y de la planificación – La interfaz desarrollada para cada proyecto de cómputo intensivo depende de los requerimientos de los científicos. Pueden surgir nuevas necesidades no contempladas durante la etapa de análisis. Adicionalmente, se trata de un sistema grande, que debe ser escalable y robusto; esto originará un ciclo de pruebas que retroalimenten el desarrollo del sistema.
- Funciona bien con equipos pequeños – En este caso, el equipo es de dos personas (la parte operativa del proyecto).
- Tolerancia a fallos - Por la naturaleza del proyecto, y debido a su integración con operaciones de bajo nivel – en muchas ocasiones, en asuntos relacionados a los sistemas operativos, protocolos de red, e iteracción con el middleware BOINC, podrían surgir complicaciones técnicas inesperadas.

Por estas razones se considera que SCRUM sí es una metodología adecuada para la gestión del proyecto. A continuación se presenta como se adaptará a este proyecto en particular.

1.4.3. Adaptación de SCRUM al proyecto

1. El product owner será el supervisor del proyecto quién es un ingeniero de la Dirección de Informática Académica.

2. El rol de equipo y de Scrum Master recae sobre el tesista.
3. El Sprint tendrá una duración de 4 semanas, donde se tendrán reuniones con el product owner.
4. La presentación del incremento se hará al Product Owner y a un interesado más, en este caso el Asesor de la tesis.

1.4.4. Línea de tiempo del proyecto

División en Sprints de duración de un mes.

Sprint 0-A (un mes)	**Inicio: Enero 2010 | Fin: Febrero 2010**
Requerimientos	
Prototipos de pantalla	
Pruebas de Web Services ZSI	
Sprint 0-B (un mes)	**Inicio: Febrero 2010 | Fin: Marzo 2010**
Requerimientos	
Pruebas de Web Services Soaplib	
Pruebas de Web Services PHP	
Configuración de BOINC	
Sprint 0-C (un mes)	**Inicio: Marzo 2010 | Fin: Abril 2010**
Análisis	
Pruebas con Web Services Axis	
Pruebas con Web Services WSF/PHP	
Configuración de BOINC	
Sprint 0-D (un mes)	**Inicio: Mayo 2010 | Fin: Junio 2010**
Diseño: Arquitectura	
Pruebas de arquitectura, transferencia de archivos binarios	
Monitoreo de comunicación con Wireshark	
Configuración de BOINC	
Implementación de MVC con Base Controller	
Sprint 1 (un mes)	**Inicio: Junio 2010 | Fin: Julio 2010**
Implementación Base Controller	
Creación del proyecto de Netbeans	
Mapeo de la Base de Datos con MyBatis	
Implementación de Login	
Sprint 2 (un mes)	**Inicio: Julio 2010 | Fin: Agosto 2010**
Implementación de gestión de permisos por roles	

Implementación de Legión Table	
Implementación de registro de usuarios	
Definición de métodos que expondrá LWS	
Sprint 3 (un mes)	**Inicio: Agosto 2010 | Fin: Septiembre 2010**
Implementación de listado de usuarios	
Implementación de eliminación de usuarios	
Implementación de modificación de usuarios	
Modificaciones a Legión Table	
Sprint 4 (un mes)	**Inicio: Septiembre 2010 | Fin: Octubre 2010**
Implementación de listado de roles (sin uso)	
Implementación de modificación de roles (sin uso)	
Implementación de creación de roles (sin uso)	
Implementación de paginación para Legión Table	
Sprint 5 (un mes)	**Inicio: Octubre 2010 | Fin: Noviembre 2010**
Implementación de modificación del perfil	
Implementación de Legión Form Renderer	
(Inicio) Configuración del Pool de Conexiones	
Implementación de recuperación de contraseña	
Implementación de Servicio de Envío de Emails	
Sprint 6 (un mes)	**Inicio: Noviembre 2010 | Fin: Diciembre 2010**
Creación de Bindings de Legión Web Services usando Axis 1.5	
Pruebas manuales de envío de tareas remotamente	
Implementación de formulario de envío de tareas	
Preparación Paper CICIC	
VACACIONES (Ponencia en la Universidad Nacional de Trujillo)	
Sprint 7 (un mes)	**Inicio: Enero 2011 | Fin: Febrero 2011**
Implementación de listado de tareas	
Implementación de Progress bar Ajax para monitoreo de tareas usando Legión Table	
Implementación de Hilo de Actualización de tareas	
Sprint 8 (un mes)	**Inicio: Febrero 2011 | Fin: Marzo 2011**

Implementación del Task Creator	
Implementación del Template Interpreter	
Sprint 9 (un mes)	**Inicio: Marzo 2011 | Fin: Abril 2011**
Implementación del Wizard de Creación de Proyectos (Datos generales del proyecto, Ingreso de Formulario XML)	
Implementación de tareas	
Ponencia en Linux Week 2011 (PUCP, Lima - Perú)	
Sprint 10 (un mes)	**Inicio: Abril 2011 | Fin: Mayo 2011**
Implementación del Wizard de Creación de Proyectos	
Pruebas de integración (LWI + LWS)	
Sprint 11 (un mes)	**Inicio: Mayo 2011 | Fin: Junio 2011**
Implementación del Wizard de Creación de Proyectos	
Pruebas de Integración (LWI + LWS)	
Sprint 12 (un mes)	**Inicio: Junio 2011 | Fin: Julio 2011**
Implementación de la búsqueda avanzada de tareas	
Implementación del Wizard de Creación de Proyectos (subir archivos)	
Sprint 13 (un mes)	**Inicio: Julio 2011 | Fin: Agosto 2011**
Implementación del Editor gráfico de formularios	
Implementación de propiedades del sistema	
Manual de usuario	
Empaquetado del primer release Web Interface 1.0 _ beta	
Ponencia en el BOINC Workshop 11 (Instituto Max Planck para la Física Gravitacional, Hannover - Alemania)	
Sprint 14 (un mes)	**Inicio: Agosto 2011 | Fin: Septiembre 2011**
Análisis de cambio al esquema de base de datos para incluir: Backends, y múltiples Backends por proyecto.	
Pruebas y correcciones	
Sprint 15 (un mes)	**Inicio: Septiembre 2011 | Fin: Octubre 2011**
Implementación de funcionalidad de descarga de tareas	
Mapeo de nuevas tablas en la BD	
Pruebas y correcciones tras cambio mayor en la BD	
Sprint 16 (un mes)	**Inicio: Octubre 2011 | Fin: Noviembre 2011**

Sprint 17 (un mes)	**Inicio: Noviembre 2011 | Fin: Diciembre 2011**
Refactorización de Legión Form Renderer (valores ocultados por seguridad)	
Pruebas para migrar a Axis2	
Refactorización de Legión Form Editor	
Pruebas y correcciones	
VACACIONES	
Sprint 18 (un mes)	**Inicio: Enero 2012 | Fin: Febrero 2012**
Migración a Axis2	
Implemetación de hilo de creación de tareas	
Refactorización de hilo de monitoreo	
Refactorización de TaskCreator	
Pruebas y correcciones	
Sprint 19 (un mes)	**Inicio: Febrero 2012 | Fin: Marzo 2012**
Refactorización de Template Interpreter	
Implementación de límite de tamaño de archivos	
Adaptación de ACE Editor para archivos XML	
Pruebas y correcciones	
Sprint 20 (un mes)	**Inicio: Marzo 2012 | Fin: Abril 2012**
Implementación de ordenamiento en Legión Table	
Implementación de validación de Template del descriptor de tarea usando DTD	
Implementación soporte para grupo de tareas	
Pruebas y correcciones	
Sprint 21 (un mes)	**Inicio: Abril 2012 | Fin: Mayo 2012**
Implementación de búsqueda en línea de usuarios	
Implementación de bloqueo de usuarios (acceso y de envío de tareas)	
Implementación de página de cambio de contraseña	
Pruebas y correcciones	
Sprint 22 (un mes)	**Inicio: Mayo 2012 | Fin: Junio 2012**
Implementación de popups para mostrar ruta de resultados	
Pruebas y correcciones	
Sprint 23 (un mes)	**Inicio: Junio 2012 | Fin: Julio 2012**
Implementación de hilo de Eliminación de tareas	
Pruebas y correcciones	

Sprint 24 (un mes)	Inicio: Julio 2012 | Fin: Agosto 2012
Manual de usuario	
Inicio proceso para registrar Legión como Propiedad Intelectual	
Taller de Legión + BOINC en Conferencia Latinoamericana de Cómputo de Alto Rendimiento (Panamá)	
Pruebas y correcciones	

Figura 1.7:Línea de tiempo del proyecto

1.5. Descripción y sustentación de la solución

1.5.1.Descripción de la solución

La solución propuesta es desarrollar un framework para la generación de interfaces web para proyectos de cómputo intensivo de manera dinámica: que sea a la vez un framework que permita generarlas (mediante modelado gráfico y textual) y también la plataforma que las hospedará, proveyendo herramientas anexas para propiciar un entorno colaborativo para los investigadores, y que permita mantener a los usuarios informados sobre el estado del sistema (tanto administradores – que estarán al tanto de un estado global, como a los usuarios finales – que estarán pendientes de las tareas que han enviado ellos o sus colaboradores registrados en el mismo proyecto).

Para lograr lo mencionado en el párrafo anterior, la solución debe ser poseedora de una arquitectura adecuada que facilite el mantenimiento y que admita extensiones – de todas formas, este tipo de sistemas están siempre en evolución para mantenerse a la par de las innovaciones en el área. Una arquitectura modular permitiría la eventual integración con otros sistemas de manejo de grids, así como la extesión de las funcionalidades en un futuro sin incurrir en cambios costosos que atraviesen toda la plataforma.

Para que esto sea posible y sea de fácil mantenimiento a través del tiempo, es necesario que haya poco acoplamiento entre los componentes. Por esta razón se propone una división de dos capas: la primera es la capa “Legión Web Interface”, que debe incluir el framework de generación de interfaces y las herramientas anexas para posibilitar el trabajo colaborativo y hospedar a estas últimas. La segunda es la capa intermedia denominada “Legión Web Services for BOINC”, que es la que finalmente ejecutará las operaciones sobre el servidor BOINC y proveerá la posibilidad de ejecutar cálculos en BOINC.

La presente tesis tiene el alcance siguiente respecto a estos dos componentes:

-Análisis, diseño e implementación de “Legión Web Interface”.
-Análisis de “Legión Web Services for BOINC” - el diseño y desarrollo estará a cargo de un ingeniero de la Dirección de Informática Académica, miembro del equipo de “Legión Framework”.

1.5.2.Costo del desarrollo

Se presenta el costo de desarrollo del sistema correspondiente a la presente tesis.

	Monto	Cantidad	Subtotal
Sueldo x mes	2000	33	66000
Impresiones	0,1	700	70
Internet	100	33	3300
Electricidad	50	33	1650
Reuniones	100	20	2000
Computadora	2500	1	2500
		TOTAL	75520,0

Tabla 1.2 : Costo del proyecto

Luego del desarrollo del proyecto, se presenta el costo de desarrollo de una interfaz.

	Monto	Cantidad	Subtotal
Sueldo x mes (tiempo: 1 día)	2000	0.03	66.67
		TOTAL	66.7

Tabla 1.3: Costo del desarrollo de una interfaz después del proyecto

Lo que representa un ahorro significativo del desarrollo de nuevas interfaces. Adicionalmente, el sistema permite reducir el tiempo de administración de la Desktop Grid. Un beneficio colateral es un mejor posicionamiento de la universidad por las investigaciones que se pueden realizar con supercómputo; en el caso particular de la PUCP, el sistema Legión ha sido extensivamente usado por el grupo de Física de Altas Energías que ha trabajado en colaboración con el CERN.

De manera cuantitativa, usar el Sistema Legión en lugar de un cluster con poder de cómputo equivalente (usando el ahorro calculado por Winter respecto al Desktop Grid de la Universidad de Westminster) indica un ahorro de cerca de 600 mil nuevos soles. Por otro lado, Legión Framework funcionaría con Desktop Grids de distintos tamaños y en distintas organizaciones, por lo que la inversión en el desarrollo es solo un costo fijo.

2. Análisis

A continuación se presentan los resultados del análisis del proyecto, incluido el proceso de seleccionar una metodología para el desarrollo. Se indican los requerimientos, los mismos que fueron establecidos a raíz de la experiencia con una primera interfaz web para el sistema Legión en la Pontificia Universidad Católica del Perú. Para finalizar, el capítulo incluye los diagramas de clases y de estados que plasman de manera más detallada los requerimientos. Para la validación de requerimientos se utilizaron prototipos de pantalla que también son considerados en el capítulo 3 y pertenecen al anexo A).

2.1. Metodología aplicada para el desarrollo de la solución

Para seleccionar una metodología de desarrollo se hace una comparación entre Agile Unified Process (AUP), Extrem Programming (XP) y Rational Unified Process (RUP). Se analizan las ventajas y desventajas que presentan estas metodologías en vista de la naturaleza del proyecto.

Las metodologías pesadas proveen de robustez al proceso de software ante los riesgos más comunes, mayormente a través de documentación extensiva. Esto ha probado ser muy útil, debido a que uno de los principales riesgos en el desarrollo de software es la comunicación: por un lado con los clientes e interesados, y por otro dentro del mismo equipo de desarrollo. Las metodologías ligeras poseen un mayor grado de flexibilidad, que podría ser un requerimiento importante debido a que por la naturaleza del proyecto, los requerimientos podrían cambiar y el curso del desarrollo podría derivar en múltiples modificaciones debido a la incertidumbre propia de una arquitectura compleja y no probada anteriormente.

Para evaluar la metodología se comparan las siguientes características:

- Adaptación a requerimientos cambiantes.
- Costo de adaptación.
- Costo / Beneficio.
- Que el ciclo de vida se adapte a la naturaleza del proyecto.
- Manejo del riesgo tecnológico.

2.1.1. Rational Unified Process (RUP)

RUP es un framework de procesos adaptable. Rational insta a adaptar y recortar el proceso de acuerdo a las necesidades del proyecto y producto. Es actualmente la metodología más usada.

Características de RUP [NIH 2010] :

- Necesita ser configurado y adaptado.
- Contiene fases, iteraciones y flujos de trabajo.
- Tiene artefactos, plantillas y guías.
- Ayuda a una organización a alcanzar niveles de madurez 2/3.

RUP define 4 fases en su ciclo de vida [Pow-Sang 2008] y a su vez una serie de disciplinas. Se puede observar en el gráfico el uso de las disciplinas durante las distintas fases.

- Incepción - Definir el objetivo del proyecto y elaborar el modelo de negocio.
- Elaboración – Planificar el proyecto, especificar los modelos y dar la base para las arquitecturas.
- Construcción – Construir el producto.
- Transición – Transición de los usuarios al nuevo producto.

2.1.2.Agile Unified Process (AUP)

AUP está basado en RUP. El ciclo de vida intenta ser serial en los bloques grandes e iterativo en los más pequeños. Los entregables son incrementales en el tiempo [Ambler 2010]. Y al igual que en RUP define 4 fases.

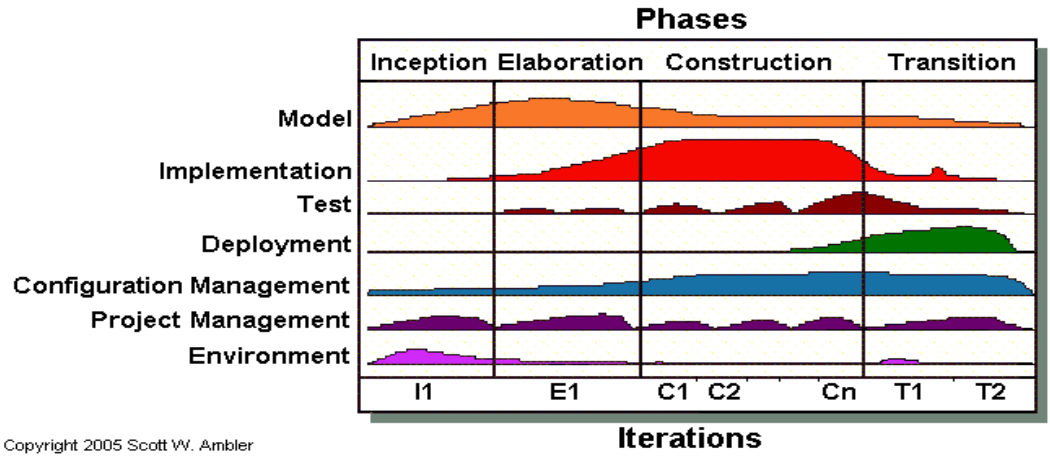


Figura 2.1:AUP [Ambler 2010]

Disciplinas [Ambler 2010]:

- **Modelado** - El objetivo de esta disciplina es entender el negocio de la organización, el problema de dominio que se abordan en el proyecto, y determinar una solución viable para resolver el problema de dominio.
- **Implementación** - El objetivo de esta disciplina es transformar el modelo en código ejecutable y llevar a cabo un nivel básico de pruebas.
- **Pruebas** - El objetivo de esta disciplina es ejecutar una evaluación objetiva para asegurar la calidad. Esto incluye la detección de defectos, validaciones del sistema diseñado, y verificar que se cumplan los requerimientos.
- **Despliegue** - El objetivo de ésta disciplina es planificar la entrega del proyecto de desarrollo y ejecutar el plan para dejar disponible el sistema al usuario final.
- **Gestión de la Configuración** - La meta de esta disciplina es manejar el acceso a sus productos de trabajo de proyecto. Esta no sólo incluye el rastreo de versiones del trabajo del producto en el tiempo, sino que también el control y administración de los cambios de estos productos.
- **Gestión de Proyecto** - El objetivo de esta disciplina es dirigir las actividades a lo largo del proyecto; esto incluye la administración del riesgo, dirección del personal (asignación de tareas, rastreo del progreso, etc.), y coordinación con personas y sistemas fuera del alcance del proyecto para asegurar su liberación a tiempo y dentro del presupuesto.
- **Ambiente** - El objetivo de esta disciplina es dar soportarte al resto del esfuerzo, asegurando que el proceso apropiado, las guías (normas y directrices), y herramientas (hardware y software) estén disponibles para cuando el equipo las necesite.

Características de AUP:

- Es una adaptación de RUP.
- Es más ligero que RUP y más pesado que XP.

- Enuncia que la documentación debe producir valor, hacer lo necesario.
- Exige algunos entregables.

Entregables a presentar:

- Sistema.
- Código Fuente.
- Scripts de instalación.
- Documentación del Sistema (Manual de Usuario)
- Notas de la versión (ChangeLog)
- Modelado de requerimientos (Prototipos de pantalla)
- Documento de Arquitectura.

2.1.3.Extreme Programming (XP)

Xtreme Programming es una metodología ágil que se centra en la parte de desarrollo del producto y se podría integrar fácilmente con SCRUM.

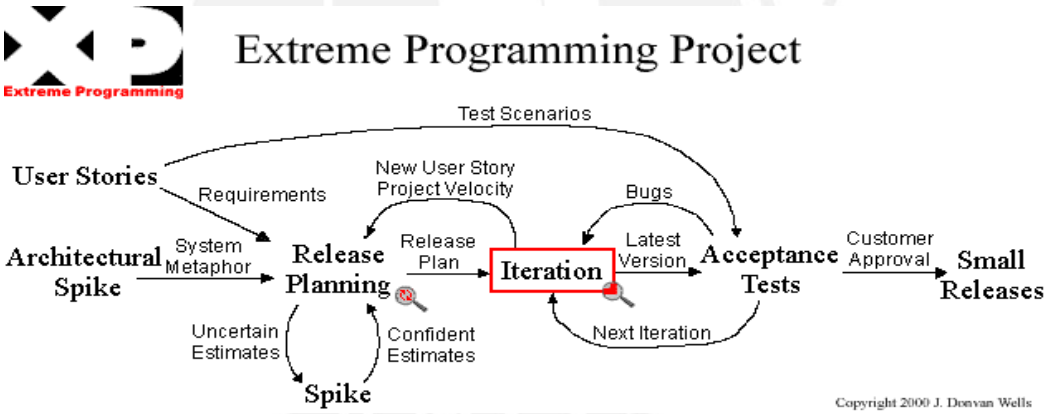


Figura 2.2: XP [Wells 2000]

Algunos principios de XP integrado a SCRUM [Kniberg 2007]

1. Programación de pares
2. Test Driven Development
3. Diseño incremental
4. Integración continua
5. Propiedad de código compartida
6. Espacio de trabajo informativo
7. Estándar de código
8. Ritmo sostenido

2.1.4. Comparación

Se presenta una tabla comparativa de las metodologías analizadas previamente.

Característica / Metodología	RUP	Agile UP	XP
Adaptación a cambios	*Regular*	*Sí*	*Sí*
Costo de adaptación adecuado	*No*	*Sí*	*Sí*
Que el ciclo de vida se adapte a la naturaleza del proyecto	*No*	*Sí*	*Sí*
Manejo del riesgo tecnológico	*Excelente*	*Bueno*	*Métodos no tradicionales – Adecuado.*

Tabla 2.1: Comparación de metodologías

Para decidir cuál será la metodología para el desarrollo del producto, se hará un realizará un análisis costo beneficio basado en la tabla 2.1. El resultado se observa en la tabla 2.2.

	RUP	Agile UP	XP
Ratio Costo / Beneficio	*Medio*	*Alto*	*Medio*

Tabla 2.2: Resultado de comparación de metodologías

Por lo tanto, se escoge AUP.

2.1.5. Adaptación de la metodología escogida

Se escogió AUP como metodología para el desarrollo del producto porque presenta una alternativa que tiene las bondades de las metodologías pesadas y ágiles. Se presenta la siguiente adaptación: entregables y notas.

Entregables:

- Requerimientos: Product Backlog / Sprint Backlog (propios de SCRUM) correspondientes a la disciplina de modelado de AUP.
- Modelado de Requerimientos: Diagramas de caso de uso / Descripción
- Modelado de Diseño: Diagramas de secuencia que muestren la interacción a alto nivel.
- Notas de versión o “Changelog”.
- Scripts de instalación
- Código Fuente

- Manual de usuario

Notas acerca de algunas disciplinas:

- Administración de la configuración : Uso de repositorio SCM Subversion.
- Administración del proyecto : Uso de SCRUM / Listado de riesgos.

2.2.Identificación de requerimientos

La Dirección de Informática Académica de la PUCP viene usando el Sistema Legión desde el 2008. En el transcurso del uso del sistema, se identificó la necesidad de contar con un framework para el diseño de la interfaz de los proyectos. Para obtener los requisitos se evaluó el desempeño pasado y presente del Sistema Legión, los requerimientos de las aplicaciones de cómputo intensivo y las necesidades futuras.

2.2.1.Listado de Requerimientos

Requerimientos funcionales

Id	Descripción	Componente
R01	El framework debe permitir la creación asistida de interfaces para un proyecto de cómputo distribuido por medio de modelamiento textual y/o gráfico.	Framework
R02	El framework debe permitir registrar los archivos ejecutables y adicionales que servirán para la creación de tareas.	Framework
R03	El framework debe permitir modelar el formulario de envío de tareas tanto de manera textual como gráfica.	Framework
R04	El framework debe permitir modelar la plantilla para descriptor de tareas de manera textual y finalmente convertirla en un descriptor de tareas de una instancia en particular usando un “intérprete de plantillas” que debe incluir un lenguaje de modelado suficientemente descriptivo.	Framework
R05	El framework debe permitir extender el comportamiento del “intérprete de plantillas” para instancias particulares de creación de tareas.	Framework
R06	El framework debe permitir modificar: el formulario de envío de tareas (parámetros requeridos para la creación de tareas), archivos ejecutables, descriptor de creación de tareas y roles de usuarios asociados	Framework

	a un proyecto de cómputo distribuido sin necesidad de romper la compatibilidad con las tareas previamente enviadas.	
R07	El framework debe permitir configurar una conexión a un *backend* de procesamiento en particular para cada proyecto de cómputo distribuido.	Framework
R08	El framework debe permitir la descarga de archivos resultados mediante retrasmisión de archivos binarios o redirección.	Framework
R09	El framework debe permitir seleccionar cuál será la forma de descarga de archivos resultados.	Framework
R10	El framework debe permitir gestionar una base de usuarios, agruparlos en proyectos y permitirles compartir tareas y resultados.	Framework
R11	El framework debe permitir notificar a los usuarios mediante correo electrónico sobre la finalización satisfactoria o debido a errores de una tarea.	Framework
R12	El framework debe permitir a los usuarios seleccionar las preferencias para la recepción de correo electrónico respecto a notificación de tareas propias o compartidas por otros usuarios de manera separada.	Framework
R13	El framework debe permitir listar y realizar una búsqueda sobre todas las tareas registradas en el sistema.	Framework
R14	La interfaz web debe permitir crear tareas usando un formulario de envío con validación para algunos tipos de datos: entero, punto flotante, valores mínimos y máximos, longitud de cadenas y expresiones regulares tanto en el lado del servidor como en el cliente.	Interfaz generada
R15	La interfaz web debe permitir gestionar la suscripción de usuarios del mismo grupo de trabajo o proyecto a los resultados de una tarea.	Interfaz generada
R16	La interfaz web debe permitir cancelar tareas.	Interfaz generada
R17	La interfaz web debe permitir monitorear el estado de las tareas (porcentaje completado, tiempo de ejecución, cantidad de nodos en uso).	Interfaz generada

R18	La interfaz web debe notificar al usuario sobre la finalización de una tarea, si este así lo ha estipulado.	Interfaz generada
R19	La interfaz web debe permitir recuperar los resultados de las tareas de acuerdo a la configuración realizada por el administrador.	Interfaz generada
R20	La interfaz web debe mostrar información sobre la economía de créditos en caso de estar activada para un proyecto en particular.	Interfaz generada
R21	La interfaz web debe permitir al usuario actualizar su perfil: nombre, correo electrónico y password.	Interfaz generada
R22	La interfaz web debe permitir al usuario recuperar su contraseña.	Interfaz generada
R23	La capa LWS debe ser una especificación extendible a otros sistemas de manejo de grids.	Legión Web Services (LWS)
R24	La capa LWS debe exponer un método para las creación de tareas: que reciba un descriptor de tarea, los archivos ejecutables y los archivos adicionales requeridos.	Legión Web Services (LWS)
R25	La capa LWS debe interpretar un descriptor de tarea que admita la creación de tareas bajo el esquema de barrido de parámetros, repetición y modo básico de listado de líneas de comando.	Legión Web Services (LWS)
R26	La capa LWS debe poder abstraer el concepto de tarea como colección de unidades computacionales más pequeñas.	Legión Web Services (LWS)
R27	La capa LWS debe exponer un método para la recuperación de resultados de tareas.	Legión Web Services (LWS)
R28	La capa LWS debe exponer un método para obtener el estado de las tareas.	Legión Web Services (LWS)
R29	La capa LWS debe exponer un método para la cancelación de tareas.	Legión Web Services (LWS)

Tabla 2.3: Requerimientos funcionales

Requerimientos no funcionales

Id	Descripción	Componente
RN01	El framework y las interfaces generadas deben tener una arquitectura web.	Framework
RN02	El framework y las interfaces deben ser compatibles con Google Chrome, Firefox e Internet Explorer 9.	Framework
RN03	El framework debe estar desarrollado en Java EE.	Framework
RN04	El framework debe admitir múltiples idiomas y debe proveer al menos: Inglés y Español.	Framework
RN05	La capa “Legión Web Services” debe exponer los métodos a través de servicios web.	Legión Web Services (LWS)
RN06	La Capa “Legión Web Services” debe estar desarrollada en Python 2.x.	Legión Web Services (LWS)
RN07	El código, los nombres de las clases, métodos y atributos, y la documentación debe estar escrita en el idioma inglés.	Framework y Legión Web Services (LWS)
RN08	El código y estructura del proyecto debe estar preparado para ser liberado como software libre y solo deben usarse componentes open source.	Framework y Legión Web Services (LWS)

Tabla 2.4: Requerimientos no funcionales

2.3.Análisis de la Solución

2.3.1.Usuarios del sistema

Existen dos tipos de usuario principales, los usuarios finales y los administradores. Sin embargo es posible distinguir tres ámbitos de permisos y roles: ámbito del sistema, ámbito del proyecto, ámbito de la tarea.

Ámbito del sistema

En el ámbito del sistema existen dos tipos de usuarios: los Usuarios Finales y los administradores. Esta distinción es muy importante, ya que los usuarios finales sólo podrán realizar actividades relacionadas a los proyectos en lo que estén efectivamente registrados, y de acuerdo al rol que tomen en estos.

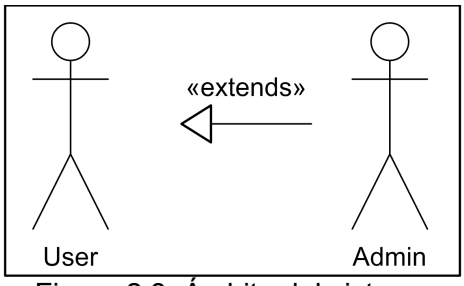


Figura 2.3: Ámbito del sistema

Ámbito del proyecto

En el ámbito del proyecto (grupo de usuarios que ejecuta una aplicación en particular en la grid) existe una distinción importante: los usuarios suscriptores, que pueden monitorear y descargar los resultados de las tareas que reciban mediante la opción de compartir y los usuarios creadores, que sí están capacitados para enviar tareas. Es importante notar, que un usuario creador también puede recibir una tarea creada por otro usuario por medio de la opción de compartir.

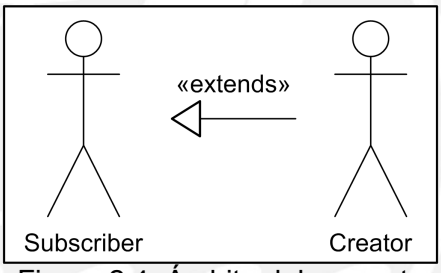


Figura 2.4: Ámbito del proyecto

Ámbito de la tarea

En el ámbito de la tarea (particular a cada instancia) existen dos roles de usuarios: los que están relacionados con la tarea porque la crearon, y los que están relacionados debido a que otro usuario, quién creó esta tarea, se las compartió. El dueño de la tarea puede eliminarla y compartirla, mientras el que la recibe por medio de la función de compartir sólo puede monitorearla y descargar sus resultados.

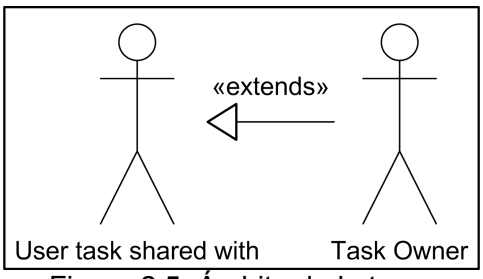


Figura 2.5: Ámbito de la tarea

2.3.2.Nociones del sistema

A continuación se presentan algunos conceptos necesarios para entender el sistema.

- **Usuario** – Generalmente es un investigador que requiere realizar cálculos que implican cómputo intensivo. Está relacionado directamente a un proyecto de investigación, y puede pertenecer a un grupo de investigación que será integrado por otros usuarios también asociados al proyecto. Muchas veces requiere ejecutar código propio compilado o escrito en un lenguaje interpretado.
- **Cliente BOINC** – Es el componente que es instalado en los nodos de ejecución. Es el encargado de ejecutar finalmente las unidades de trabajo de los proyectos BOINC a los que se encuentre suscrito.
- **Proyecto** – Es la asociación de un grupo de usuarios con una aplicación que se ejecuta en la desktop grid, por lo que es posible abstraer su operación más general en un formulario de envío de tareas.
- **Proyecto BOINC** – Incluye una o más aplicaciones que serán ejecutadas en los distintos clientes. Puede incluir versiones para múltiples plataformas como Windows, Linux, Max OS X, etc en distintas arquitecturas como: 32 bits y 64 bits.
- **Tarea** – Es un concepto que no existe en la versión estable de BOINC al momento de escribir esta tesis. Representa a la ejecución de un cálculo de cómputo intensivo como colección de unidades de trabajo más pequeñas pero masivas (este esquema se conoce como *bag-of-tasks* o bolsa de tareas). Sin embargo, la siguiente versión estable de BOINC probablemente incluirá esta funcionalidad. Por otro lado, muchos proyectos desplegados en el mundo usaron el campo “batch” de la tabla workunit de BOINC para simular el concepto de tarea.
- **Workunit o unidad de trabajo** – Representa una unidad, generalmente indivisible, de cálculos. Una unidad de trabajo es enviada a uno o varios nodos o clientes BOINC, donde serán ejecutadas. Pueden ser enviadas a múltiples clientes por redundancia, método que se usa para salvar la no confiabilidad de los recursos de cómputo voluntario (lo que no ocurre en un ambiente controlado como el Sistema Legión).
- **Tarea completa** – Una tarea está completa cuando todos los workunits que la conforman han sido ejecutados satisfactoriamente y se ha recibido de manera correcta sus resultados.
- **Account Manager** – Un account manager es un componente propio de la arquitectura de BOINC en la que una organización poseedora de computadoras (*pool*), conocida como *Supplier* o un individuo que tiene una computadora conocido como *Volunteer*, delegan la tarea de seleccionar a que proyectos BOINC colaborarán sus computadoras – y en qué porcentajes – a una institución conocida como *Allocator*.

2.3.3.Capa Web

Los requerimientos del sistema han sido evaluados y tras el análisis correspondiente se ha obtenido los siguientes casos de uso.

Paquete Administración

Este paquete contiene los casos de uso relacionados a tareas de administración del sistema.

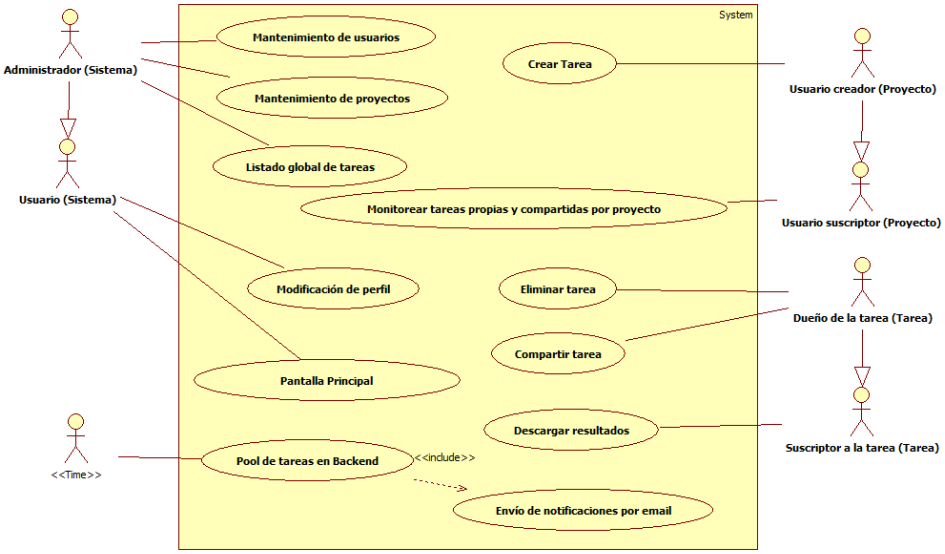


Figura 2.6: Diagrama de Casos de Uso – Paquete Administración

Listado de casos de uso:

Id	Nombre	Descripción
CU01	Mantenimiento de proyectos	Incluye todas las actividades relacionadas a los proyectos: Listado, Creación, Modificación y Eliminación.
CU02	Mantenimiento de usuarios	Incluye las actividades relacionadas a los usuarios del sistema: Listado, Creación, Modificación y Eliminación.
CU03	Listado global de tareas	Incluye un listado detallado y búsqueda de todas las tareas enviadas en el sistema.
CU04	Pool de tareas en Backend	Es un proceso periódico que actualiza el estado de las tareas en la base de datos del sistema, obteniendo esta información de consulta al *backend* registrado por cada tarea.
CU05	Modificación de perfil	Modificación de los datos del usuario: nombre, apellidos, contraseña y correo electrónico.

CU06	Pantalla principal	Muestra los datos relevantes del sistema dependiendo si el usuario es un administrador o un usuario final (ámbito del sistema).
CU07	Crear tarea	Un formulario dinámico es mostrado al usuario: éste ingresa los datos, parámetros y opcionalmente archivos de entrada para crear una tarea.
CU08	Monitorear tareas propias y compartidas por proyecto	Muestra el listado de tareas enviadas por el propio usuario o que ha recibido por medio de la opción de compartir. Pueden ser filtradas por estado: finalizadas, ejecutando, falladas o por las opciones de compartir: compartidas para mí, compartidas por mí.
CU09	Eliminar tarea	Permite eliminar una tarea ya ejecutada o cancelar una tarea que se encuentre en ejecución.
CU10	Compartir tarea	Permite “suscribir” a otros usuarios al progreso y resultados de una tarea en particular. Sólo el usuario que la creó puede realizar esta operación.
CU11	Envío de notificaciones por email	Permite que los usuarios sean notificados tras una actualización del estado en la base de datos.

Diagrama de Clases

A continuación se presenta el diagrama de clases de análisis de la Legión Web Interface, particularmente la parte correspondiente al lado del servidor.

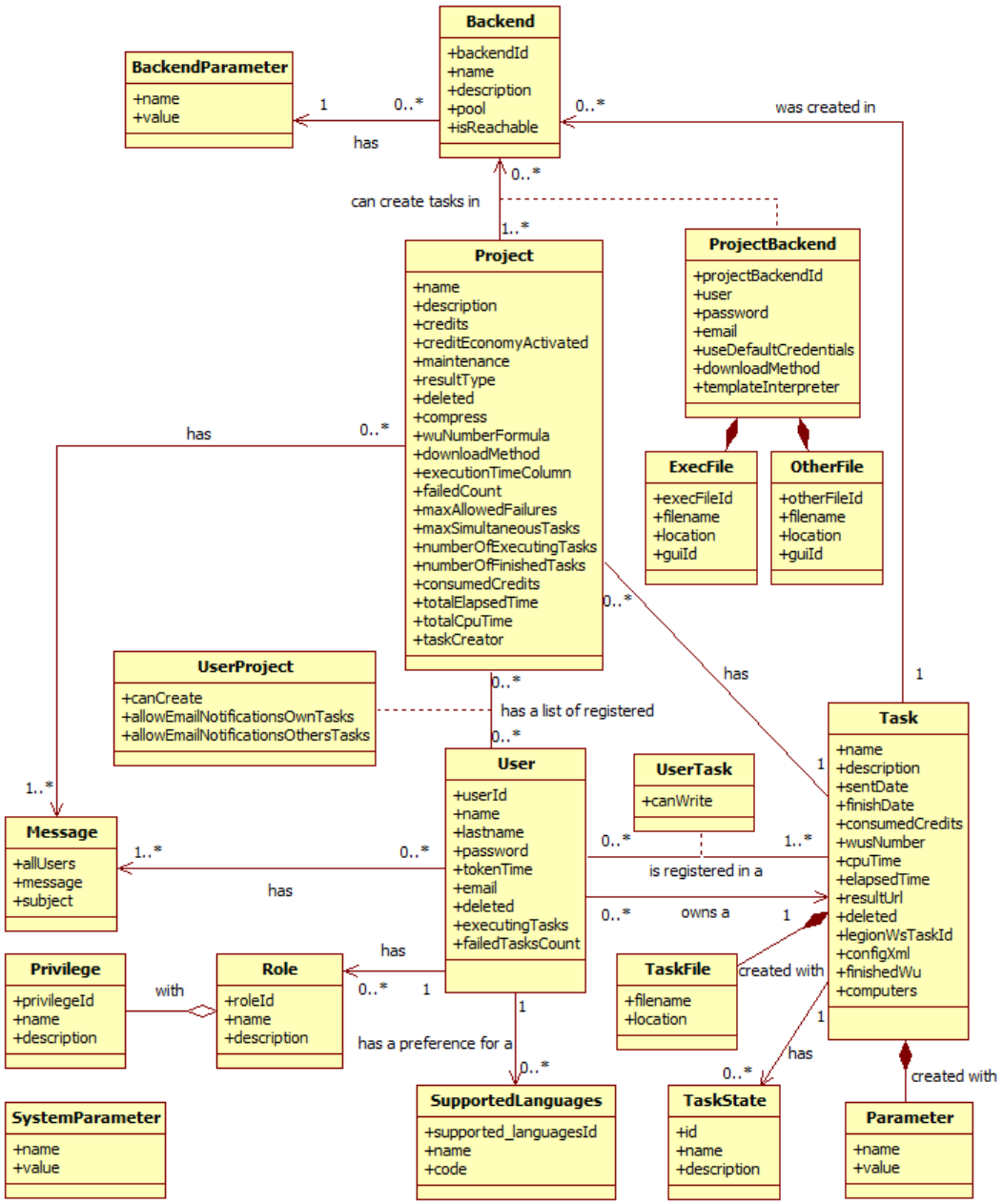


Figura 2.7: Diagrama de clases de análisis (Capa Legión Web Interface)

Estados de la clase principal: Task

La clase más importante en el sistema es la que representa el concepto de tarea (bag-of-tasks), en parte porque esta noción no existe en BOINC y es uno de los principales valores agregados que brinda este sistema.

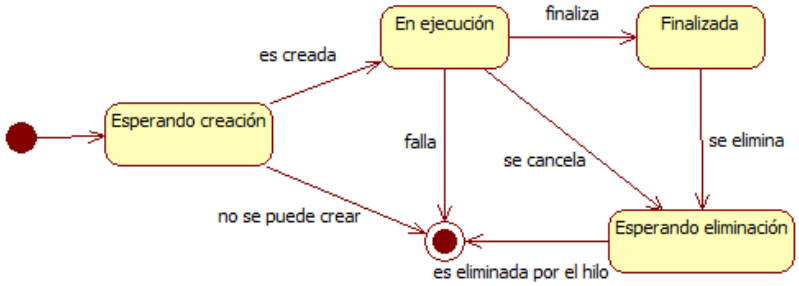


Figura 2.8: Diagrama de estados – Clase Task

Cabe resaltar que un fallo en una tarea ocurre cuando el número de intentos hechos por el Servidor BOINC de ejecutar un workunit superan el valor del parámetro correspondiente. Este parámetro se puede configurar para admitir una redundancia más robusta. Para definir el valor de este parámetro se debe considerar también el ambiente de ejecución y cuán confiable es.

3. Diseño

En el presente capítulo se presentan las consideraciones de diseño para el sistema. Se comienza con la arquitectura y las razones que llevaron a la especificación mostrada en adelante. Más detalles se pueden obtener en el anexo B "Arquitectura". Posteriormente, se explican las consideraciones para el diseño de la interfaz gráfica y se muestra la arquitectura de la información.

3.1.Arquitectura de la solución

A continuación se presenta la arquitectura del sistema. Cómo principio de diseño se consideró que los componentes del sistema deberían estar lo suficientemente desacoplados, de modo que bajo determinadas circunstancias sea posible intercambiarlos a conveniencia. Esto permitite que el producto sea más mantenible; y además, los desarrolladores de cada componente podrían trabajar independientemente y en forma paralela.

Para lograr esto se escoge una arquitectura orientada a servicios. Para resumir, se establece una lista de métodos que debería exponer el sistema manejador de Grid a través de un componente que se inserte sobre este. Esta lista de métodos sirve a su vez como una especificación para la construcción de los componentes sin

necesidad de interactuar con el componente mismo, ya que puede estar plasmada en un documento estándar como el WSDL. El alcance de la presente tesis considera la interacción y especificación de un componente que pueda gestionar un servidor BOINC, pero no su implementación: lo que quedan a cargo de un ingeniero de la Dirección de Informática Académica.

En lineas generales el componente que se inserta sobre el sistema manejador de Grid debe proveer la implementación de conceptos que eventualmente podrían no estar presentes. En el caso de BOINC, al momento de empezar el trabajo no existía el concepto tarea como colección de unidades de cómputo más pequeñas e independientes entre sí (bag-of-tasks). Por eso fue necesario implementar este concepto. Sin embargo, en el BOINC Workshop 11 se anunció los “Batch Jobs” o tareas compuestas por muchas unidades de trabajo. Esto era previsible debido a que en la tabla workunit del sistema BOINC ya existía hace buen tiempo un campo “batch” que se usaba, por lo general, para agrupar tareas. Este es el mismo uso que se le da en Legión Web Services para BOINC.

Esta arquitectura permite separar los componentes lo suficiente, de modo incluso que una falla en uno de los servidores no llegue a comprometer a los otros y a su vez la carga de recursos computacionales como memoria, procesador y acceso a red sean independientes. Los nodos que conforman el sistema son: Nodo cliente – que ese la computadora del usuario final con un browser con javascript habilitado, Legión Web Interface (LWI) que es un servidor web, Legión Web Services (LWS) + Sistema manejador de Grid y los múltiples nodos de ejecución conforme a la arquitectura del sistema manejador de Grid.

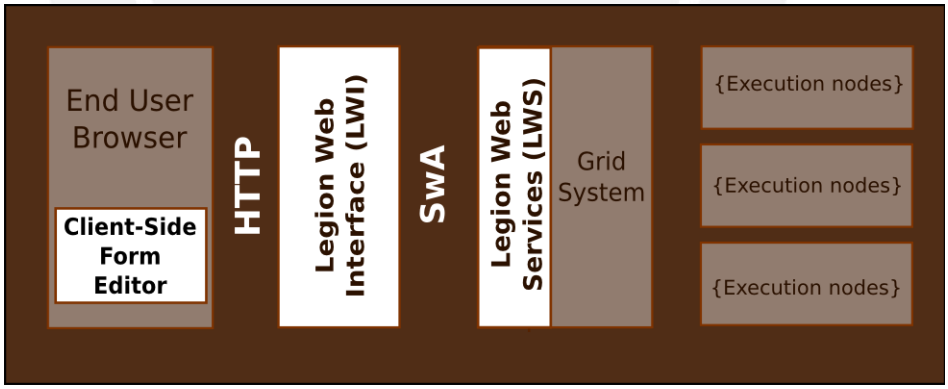


Figura 3.1:Diagrama de la arquitectura del sistema

Nodo Cliente

Es la computadora del usuario final del sistema y contiene únicamente un navegador web, el cual deberá tener habilitadas funcionalidades que permitan ejecutar código javascript, lo que es algo muy común hoy en día.

Nodo Legión Web Interface (LWI)

Es el encargado de recibir y atender las peticiones del cliente a través del protocolo HTTP. Contiene adicionalmente, un sistema administrador de base de datos, del que cual se vale para obtener y persistir la información generada durante la interacción con el usuario final y con los múltiples sistemas manejadores de Grid.

Nodo Legión Web Services (LWS)

Es el nodo que expone servicios web SOAP que serán consumidos por el componente LWI con el fin de acceder y manejar el sistema de Grid. En el caso de la implementación para BOINC es importante notar que este sistema no ofrecía servicios web al momento de comenzar el presente trabajo. Es solo en el BOINC Workshop 2011 que se anuncia una capa de servicios web, pero que no siguen el estandar SOAP.

Nodo de ejecución

Es el nodo dependiente del sistema manejador de Grids en donde se lleva a cabo la ejecución de las unidades de cómputo correspondientes a las tareas enviadas por el usuario final del sistema. La arquitectura y protocolos de comunicación son particulares al sistema manejador de Grid correspondiente.

Es importante notar que todos estos nodos y componentes deben interactuar entre sí de manera transparente al usuario final quién solo podrá ver la interface web que se le presenta. Incluso, estos componentes deberían poder funcionar sin la intervención de un administrador.

3.2.Legión Web Interface

Legión Web Interface es el principal componente diseñado en la presente tesis. En líneas generales debe ser una meta-interfaz web para múltiples sistemas manejadores de Grid. Para conseguir esto no debería ser necesario modificar el código fuente de la aplicación. Por esta razón, se especificarán los siguientes componentes: TemplateInterpreter, TaskCreator, FormRenderer, Form Editor descritos a continuación.

3.2.1.Template Interpreter

Legión Framewok debe incluir un "**DefaultTemplateInterpreter**" para la generación del archivo descriptor de la tarea que es requerido por Legión Web Services. Este descriptor indica como se creará la tarea; es por ello que debe contener información que solo estará disponible en el momento del envío de estas. Se deben proporcionar etiquetas para acceder a los parámetros enviados en el formulario para la construcción del archivo config.xml. En la etapa de creación del proyecto se debe especificar la plantilla y qué intérprete de plantilla se va a utilizar.

Remplazo de variables de formulario

Si se ingresa un componente HTML en el formulario y se identifica, por ejemplo, por el id "txtSomething"; entonces, sólo se debería utilizar la etiqueta **{Form:txtSomething}** en la plantilla o "template" para obtener el valor de este componente al realizar el remplazo de la etiqueta.

La evaluación matemática

Si fuera necesario calcular un valor antes de sustituirlo, solo se debería añadir una etiqueta de evaluación matemática: **{Math:operación matemática}**. Se debe poder utilizar los operadores básicos: suma (+), resta (-), multiplicación (*) y división (/) en conjunción con el remplazo de valores del formulario. Debería ser válido también el uso de paréntesis como signos de colección.

Por ejemplo:

{Math:({Form:txtLast} - {Form:txtFirst}) / {Form: txtStep}}

Debería calcular:

$Reemplazo = (txtLast - txtFirst) / txtStep$, usando valores enviados por el usuario final a través del formulario de envío de tareas.

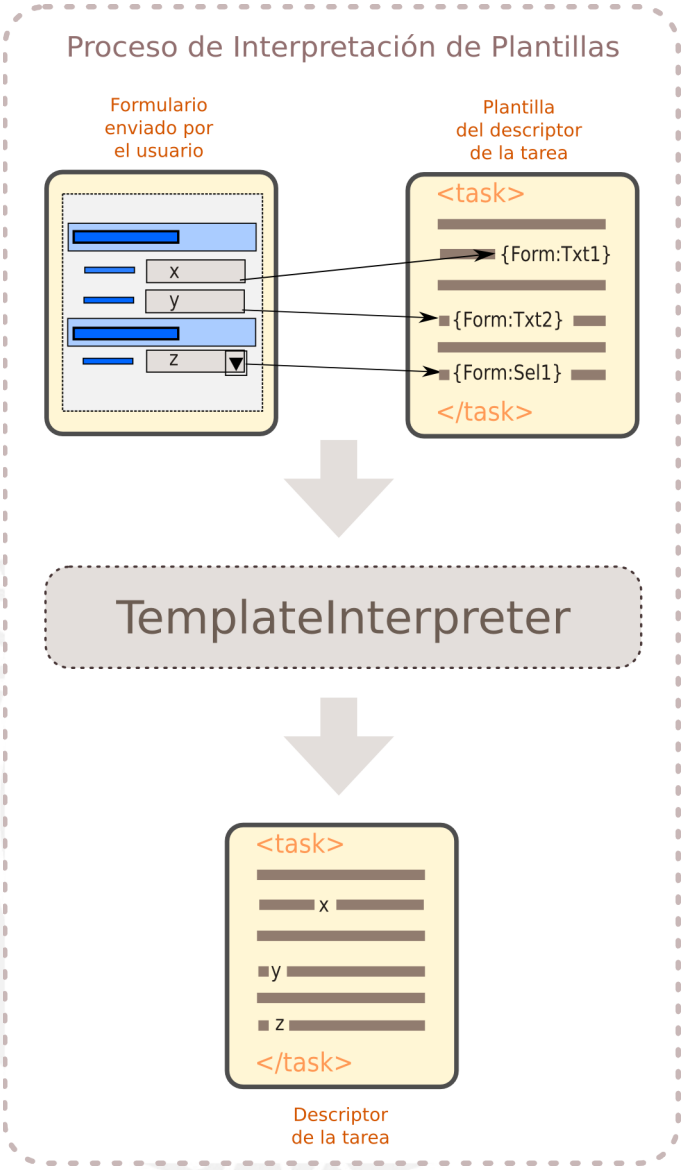


Figura 3.2:Funcionamiento del Template Interpreter

Operador "Split"

A veces será necesario separar los valores de un componente HTML. Por ejemplo, si se tuviera un conjunto de parámetros para ejecutar en cada línea de un "TextArea", probablemente sería necesario separar el valor devuelto por este campo, usando el carácter de cambio de línea **"\n"** como separador. Para conseguir esto, si el campo se llamara por ejemplo: **txtCommands**, bastaría usar la etiqueta "Split" de la siguiente forma:

<workunit>**{Split:txtCommands: \ n}**</ workunit>

Cabe resaltar que toda la línea debería repetirse. Es decir, si la etiqueta está rodeada por texto, como es el caso de la etiqueta "workunit" en el ejemplo, este también se repetiría. Además, se debería usar cualquier carácter o cadena de carácteres como separador.

```
<workunit> línea 1 </ workunit>
<workunit> línea 2 </ workunit>
<workunit> línea 3 </ workunit>
 ...
```

El uso de esta etiqueta puede requerir conocer la cantidad de líneas, por lo que se debe proveer una etiqueta "SplitCount", que permita contar la cantidad de elementos generados al realizar la operación de separación o "Split".

Por ejemplo:

{SplitCount:txtCommands:\n}

Esta etiqueta será remplazada por el valor numérico de la cantidad de líneas no vacías en el componente txtCommands.

El template interpreter sigue un patrón "Factory" debido a que el lenguaje especificado para el intérprete podría ser insuficiente en algunas aplicaciones, por lo que este componente debe ser fácilmente reemplazable. Se utiliza un método basado en inspección para instanciar la clase factory, de modo que no es necesario modificar el código fuente, ni base de datos de la aplicación para agregar un nuevo Template Interpreter.

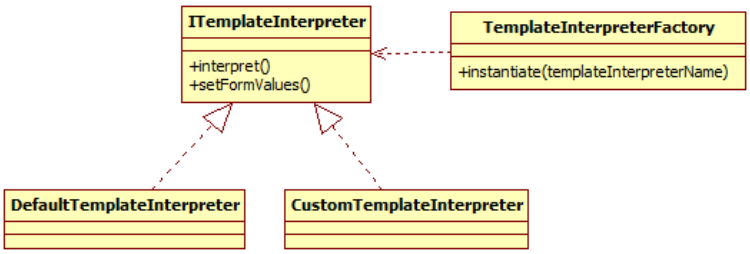


Figura 3.3:Diseño simplificado del Template Interpreter

3.2.2.TaskCreator

El componente de creación de tareas o “TaskCreator” realiza la creación de tareas y es instanciado al recibir el formulario de creación de tareas de un proyecto en particular. Sigue un patrón “Factory”, debido a que su funcionamiento por defecto podría ser reemplazado a conveniencia para flujos de trabajo muy particulares. Cabe resaltar que los proyectos tienen registrado que TaskCreator se usará para crear sus tareas. Es importante notar que se los TaskCreators se instancian por inspección, de modo que no es necesario registrarlos en el código o base de datos: basta tener el nombre.

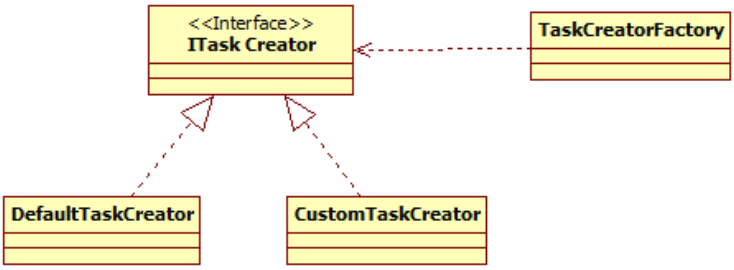


Figura 3.4: Diseño de TaskCreator

Las tareas en Legión Framework son enviadas a través de una interfaz web. Sin embargo, su creación en el servidor BOINC no es inmediata. Las tareas una vez enviadas por el usuario, son almacenadas en un “buffer” de tareas por Backend esperando su creación. Sin embargo, el buffer en mención es un concepto lógico, porque las tareas finalmente residen en la base de datos. Un hilo es iniciado por cada Backend que tenga tareas en espera de ser creadas. Este hilo las va ir creando, una por una, en el Servidor BOINC a través de los servicios SOAP ofrecidos por Legión Web Services.

Proceso de creación de tareas

El proceso de creación de una tarea se inicia en el momento en que el usuario, tras haber abierto el formulario de envío de tareas, hace click en crear. Los parámetros son recibidos por el TaskCreator designado en el paso 2 de la creación de proyectos.
Por defecto se usa “DefaultTaskCreator”. Sin embargo, de requerirse una modificación del comportamiento estándar, es posible usar una implementación propia.

3.2.3.FormRenderer

El “*renderizador*” de formularios o FormRenderer es un componente que tiene por tarea realizar las operaciones relacionadas a los formularios dinámicos para el envío de tareas. Una de sus tareas principales es generar el código HTML del

formulario, así como generar el código javascript para validación del lado del cliente en el mismo navegador. También se encarga de validar el formulario en el lado del servidor y cargar la estructura del formulario de un documento XML.

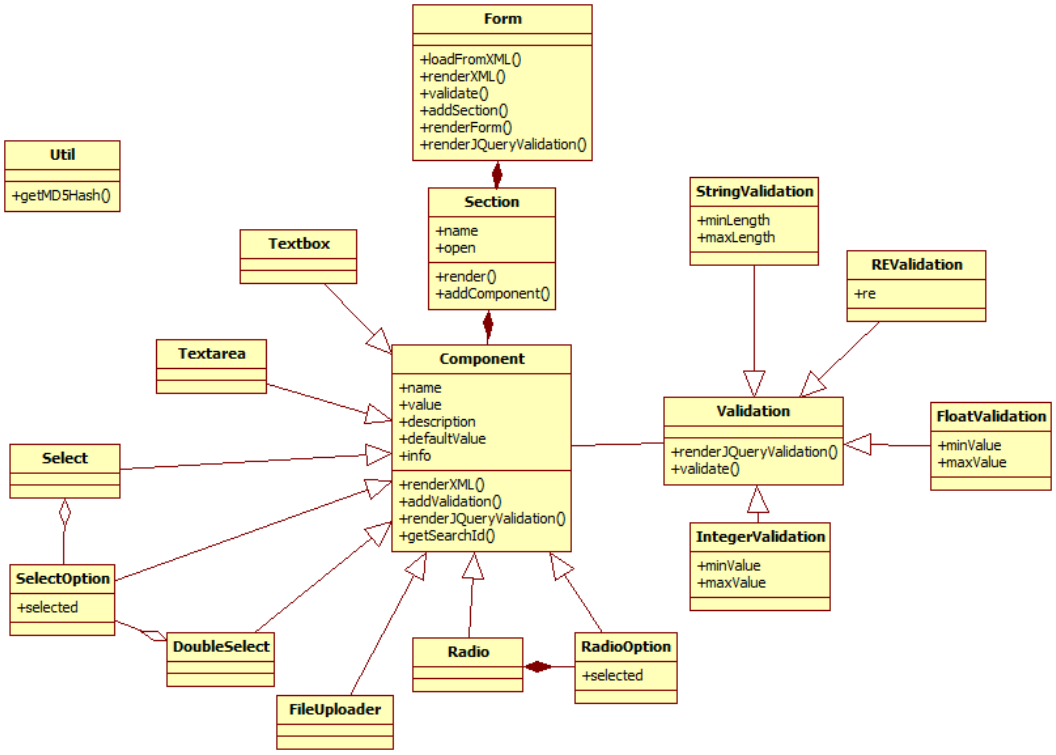


Figura 3.5: Diseño de Form Renderer

3.2.4.Editor de formularios del lado del cliente: “Legion Form Editor”

El editor de formularios en el lado del cliente estará desarrollado en el lenguaje Java usando el framework Google Web Toolkit que permitirá compilarlo a código javascript. Dado que Java es un lenguaje orientado a objetos se presenta a continuación un diagrama de clases de diseño para el mencionado componente. Cabe resaltar que será necesario implementar una clase “Helper” para interpretar los documentos XML correspondientes a los formularios que se editarán.

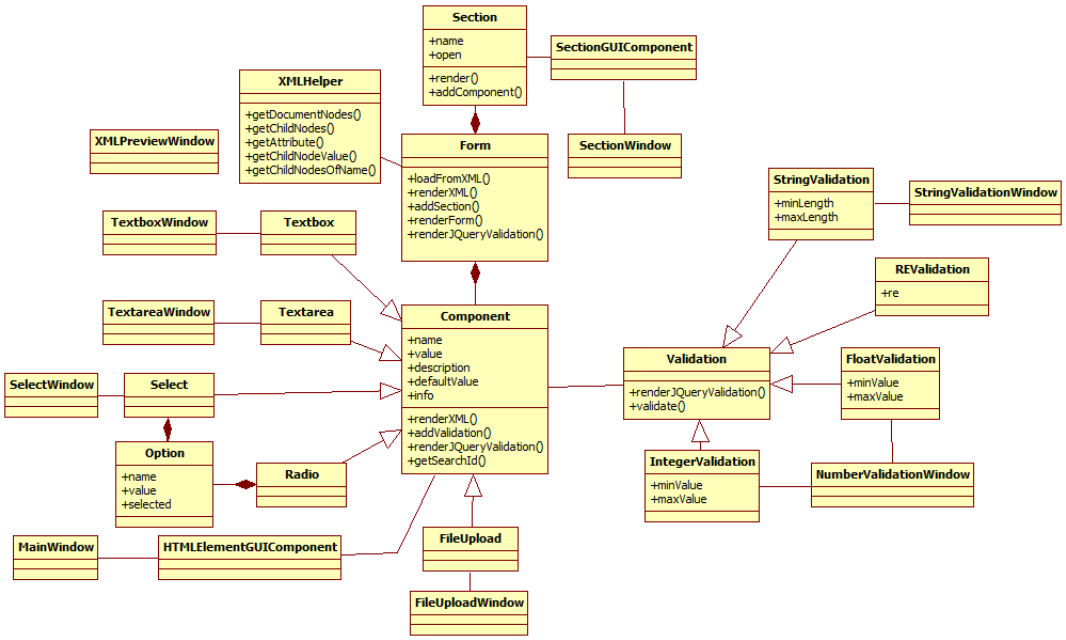


Figura 3.6: Diseño de Form Editor

Adicionalmente, será necesario definir dos nuevos "Widgets": ***SectionGUIComponent*** y ***HTMLElementGUIComponent*** para mostrar las secciones y componentes HTML en el formulario.

A continuación se puede apreciar el prototipo de la pantalla principal, donde en cada fila se observa el componente ***HTMLElementGUIComponent.***

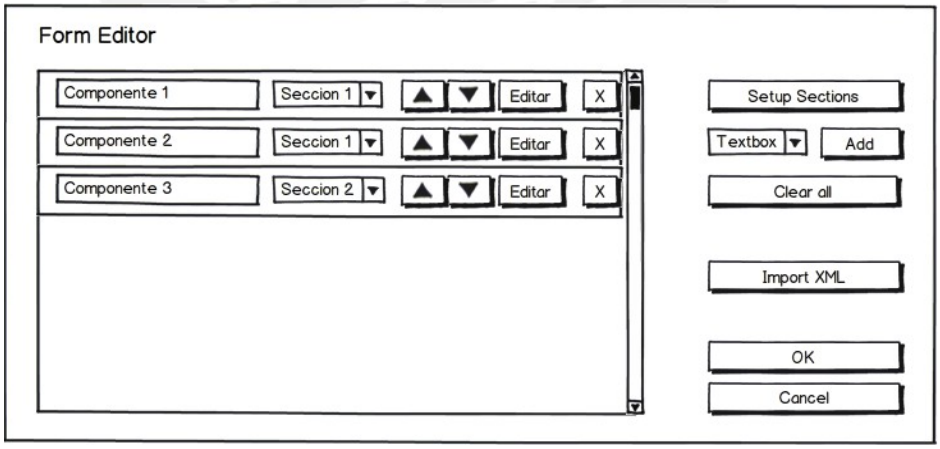


Figura 3.7: Prototipo de pantalla principal

Para ver el resto de prototipos refiérase al anexo A.

3.2.5. Web Interface

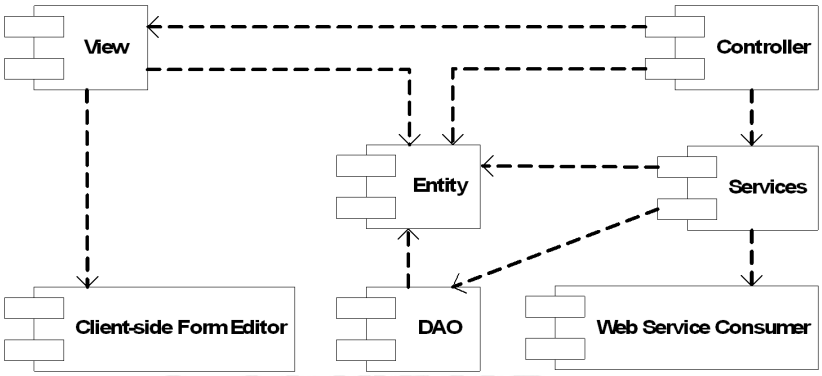


Figura 3.8:Diagrama de paquetes de LWI

3.3. Legión Web Services

La capa LWS expondrá servicios web. Estos servicios serán consumidos por la capa web. Sin embargo, esta capa representa una capa “abstracta” y representa una especificación más que una implementación particular. Una de las ideas principales al diseñar esta capa es brindar a la comunidad de usuarios BOINC una capa que pueda integrarse con sistemas middleware más grandes como Globus Toolkit, aunque la integración propiamente dicha queda fuera del alcance de la presente tesis.

Finalmente estos servicios harán uso de la API de BOINC para ejecutar acciones sobre el servidor.

3.3.1.Interacción del sistema

Es importante validar que la arquitectura propuesta soporta los flujos principales del sistema. Consideramos importantes:

- Flujo de envío de tareas
- Flujo de recuperación de resultados

Para lograr esto se presenta en el ANEXO B los diagramas de secuencias que permiten observar la interacción entre las capas en los flujos que pueden presentar problemas en el sistema.

3.3.2.Lineamientos para interacción de las capas

Se presentan algunos lineamientos para la interacción de los componentes del sistema.

- El usuario, cuando recupera el resultado de una tarea, puede hacerlo por streamming desde la capa web en caso de tratarse de un solo archivo o

también, puede recibir una dirección ftp, de donde puede descargar los archivos.

- La capa web recibe una URL a la que puede acceder con el fin de retrasmitir el contenido de algún archivo hacia el usuario.
- Cuando se envía una tarea, dependiendo del número de unidades de trabajo que implique, puede demorar mucho en el proceso de creación. Por eso no es factible confirmar la creación de la tarea, debido a eso se considera un estado “En creación”.

3.4. Diseño de interfaz gráfica

3.4.1. Lineamientos de interfaz gráfica

El desarrollo de la interfaz gráfica para el presente proyecto está en función a los lineamientos establecidos en una versión previa del sistema Legión. Es por esto que solo se presentan algunas mejoras.

Figura 3.9: Pantalla del sistema

Distribución de la pantalla

Se presenta la distribución de la pantalla y algunos lineamientos de diseño.

Figura 3.10:Distribución de la pantalla

Barra superior – Es el componente que identifica al sistema, y sigue el lineamiento de los servicios que ofrece la Dirección de Informática Académica de la PUCP. Dimensiones 760 x 90 pixeles.

Imagen – Es una imagen que representa el servicio ofrecido. Dimesiones 760 x 140 pixeles.

Barra Informativa – Es una barra que contiene ayuda de navegación e identificación del usuario. Su color es Gris.

Barra - Es una barra complementaria, de color RGB #FFCF23.

Barra lateral de menú - Es una barra que contiene acceso directo a las principales funcionalidades.

Área de contenido – Es el área donde se posicionarán los diferentes componentes correspondientes a cada funcionalidad.

3.4.2. Prototipos de interfaz gráfica

Ver anexo A. Se utilizó la herramienta Balsamiq Mockups.

3.5. Arquitectura de información

3.5.1. Representación de formularios

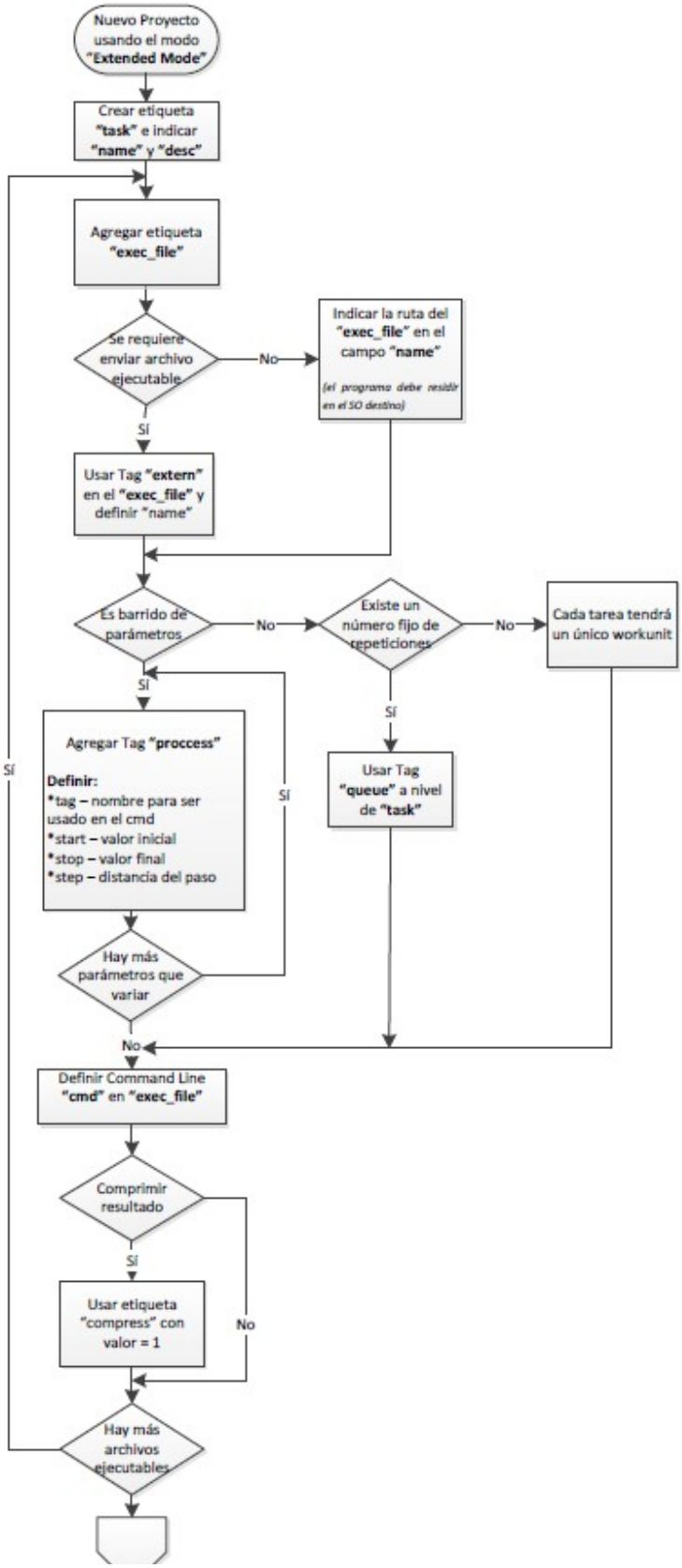
Los formularios se representan en archivos con formato XML.

Los componente posibles son:

- OptionGroup
- TextBox
- Button
- FileUpload
- Select
- TextArea
- Checkbox

3.5.2. Especificación del descriptor de tareas

La lista de las etiquetas admitidas en el Descriptor de Tareas se presentan en un anexo. Para más detalle de como crear este archivo, se puede consultar el siguiente diagrama de flujo.



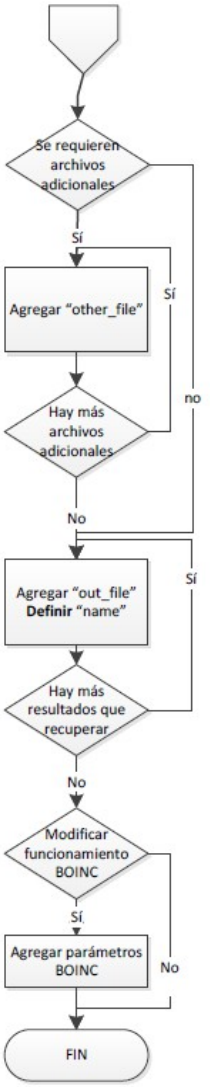


Figura 3.11:Procedimiento para crear descriptor de tareas

3.5.3. Especificación de servicios web

A continuación se muestra la lista de métodos que expone la capa LWS y que son consumidos por Legión Web Interface.

Nombre	Entrada	Salida	Descripción
User_Validate	• Username • Password	• Authenticator	Valida los credenciales de un usuario, otorgando un "Authenticator" para realizar operaciones subsiguientes.

Task_Status	• Task_id • Authenticator	• Nombre • Descripción • Tiempo CPU • Tiempo Real • Status • URL Resultado • Wus finalizados • Cantidad de Pcs	Permite conocer el estado detallado de la tarea que se está consultando.
Task_Create	• Nombre • Descripción • config XML • Archivos ejecutables • Otros archivos • Authenticator	• Status	Permite crear una tarea, devuelve un flag con el resultado de la operación.
Task_Cancel	• Task_id • Authenticator	• Status	Permite cancelar una tarea, devuelve un flag con el resultado de la operación.

Tabla 3.1: Lista de métodos de la capa LWS

4. Construcción y pruebas

En el presente capítulo se presentan las consideraciones para el desarrollo del sistema, así como las pruebas ejecutadas.

4.1.Construcción

4.1.1.Selección de tecnología para el cliente

Para el desarrollo de funcionalidades del lado del cliente se requeriría de alguna tecnología que permita brindar respuestas muy rápidas al usuario. Esto último fue lo que motivó la idea de implementar el editor gráfico de formularios en el lado del cliente. Las dos opciones principales fueron: flash y javascript. El criterio para escoger una u otra estuvo relacionado con qué tan bien soportados estaban y estarían en la variedad de dispositivos heterogéneos y sus navegadores. Ya que Flash estaba contando cada vez con menos soporte en plataformas móviles y las basadas en linux (ampliamente usado en la ciencia), se escogió javascript. Sin embargo, incluso el código javascript requiere ser adaptado para funcionar en múltiples navegadores; e incluso, también para que funcione en distintas versiones de un mismo navegador. Una forma de delegar esta adaptación a un tercero es usando un framework “cross-plataform”. GWT permite escribir código Java y “compilarlo” a javacript. Para el desarrollo de un editor gráfico de formularios se requerirían ventanas, definir nuevos widgets y el paradigma orientado a objetos sería de gran utilidad. Para todo esto último, GWT probó ser muy útil.

Funcionalidades de GWT (útiles para el proyecto):

- Programación en java, se compila a javascript.
- Soporte de programación orientada a objetos.
- Soporte de ventanas.
- Soporte para definir nuevo widgets.

- Código optimizado para diferentes navegadores.
- Helpers para interpretar “XML”

El editor gráfico de formularios debía poder cargar una descripción textual (en formato xml) del formulario en creación en el mismo navegador, permitir su modificación de manera gráfica y producir un nuevo documento XML: todo esto, sin intervención del servidor.

4.1.2.Selección de tecnología para Servicios Web

Se decidió usar servicios web SOAP para permitir que los componentes del framework interaccionen, esto frente a tecnologías como REST. La razón para preferir al primero es que las librerías que dan soporte a SOAP, e incluso los estándares mismos, están mejor definidos que para servicios REST. Al desarrollar un framework, es importante facilitar y animar la extensión de las características. Al proveer una especificación estricta de los métodos que expondrán y consumirán los componentes, se facilita una mejor integración ya que se “regulan” estas comunicaciones. Así el esfuerzo humano para integrar el framework con nuevos componentes se ve mejor canalizado. Vale la pena notar que el estándar SOAP exige la trasmisión de información adicional lo que genera un *overhead* mayor que el producido por REST; sin embargo, se entiende que los beneficios provistos y la naturaleza de la comunicación (baja frecuencia de comunicación simultanea) justifican su uso.

4.1.3.Selección de tecnología para Servicios Web SOAP

A pesar de que Python es un lenguaje muy usado hoy en día, el soporte para servicios web no es tan completo como en Java. Para la presente tesis se hicieron pruebas con dos lenguajes principales: Java y Python, siendo el objetivo demostrar que la implementación de servicios web era interoperable entre estas dos plataformas. Adicionalmente se hicieron pruebas con el Lenguaje PHP. Sin embargo, es importante notar que Legión Web Services for BOINC debía ser desarrollado en Python, porque BOINC tiene una API para este lenguaje.

The Zolera Soap Infraestructure (Librería Python)

La librería ZSI 2.0 fue una de las alternativas, aunque fue descartada debido a que la documentación oficial era poca y la última versión oficial databa de finales del año 2007. Al comenzar el desarrollo de Legión Framework en el 2010 y según lo manifestado en la lista de correos no existían los recursos necesarios para planear una siguiente versión oficial.

Cabe resaltar que ZSI sigue la filosofia de Desarrollo “TopDown”. Se parte de un documento WSDL (Web Service Description Language) para generar el código tanto para el desarrollo del componente cliente y del servidor, esto podría representar un problema si durante la etapa temprana de pruebas de interacción de Legión Web Interface y Legión Web Services se hubiera tenido que redefinir los

métodos (escribir un documento WSDL sin la asistencia de herramientas es complicado).

Tras pruebas realizadas se comprobó que no soporta el estándar SOAP with Attachments (SwA) y que la transferencia de archivos binarios no estaba optimizada; esto último era un requerimiento importante, porque al crear una tarea en Legión Web Interface era imprescindible enviar archivos ejecutables y otros archivos adicionales para crear la tarea en Legión Web Services.

Soaplib (Librería Python)

Soaplib es la librería que tiene una mayor frecuencia de actualización y la comunidad alrededor es muy activa. Para desarrollar servicios web se sigue el método bottom up, esto quiere decir que el WSDL es generado a partir de los métodos conforme se van programando. Esto resulta muy conveniente en el contexto del presente proyecto ya que el desarrollo de los servicios web fue progresivo y los métodos variaron ligeramente durante la etapa inicial.

Las pruebas que se realizaron con Soaplib incluyen: interacción con clientes en PHP, Java y Python. Haciendo uso de Wireshark para analizar la trama HTTP, se comprobó el correcto funcionamiento. Se pudo, además, afinar los parámetros de las librerías. La compatibilidad con Java se da en la medida que existe un alto interés en la integración con Axis, iniciativa que proviene de la misma comunidad de Soaplib. La compatibilidad con PHP se da ya que existe una versión de Axis para PHP llamada WSF/PHP.

Las pruebas se realizaron con el servidor HTTP propio de Python. Sin embargo, en producción se debería usar en conjunto con el servidor Apache con el módulo mod_wsgi y cherrypy. En la interacción con estos dos componentes de software, es posible que el servidor acepte conexiones HTTPS.

Axis2 (Librería Java)

Apache Axis es la librería más extendida para servicios web en Java. Hay estudios que analizan su desempeño en el campo de la e-ciencia, por lo que fue una opción casi por defecto. Además, hay herramientas para generar los bindings dado un WSDL. Además, estos bindings automáticamente permiten configurar número de reintentos y timeouts, lo que resulta práctico para implementar la interacción de los componentes de Legión Framework. Sin embargo, lo más importante fue el soporte de envío de archivos binarios de manera optimizada. Si se requiriera proteger el contenido, Axis2 permite la conexión sobre HTTPS. Vale la pena recordar que Axis2 será usado como una librería para “consumir” servicios web.

4.1.4.Comparación con Servicios Web REST

Los web services REST se originaron como parte la tesis de doctorado de Roy Fielding. El acrónimo REST hace referencia a REpresentational State Transfer. Se debe ver REST como un patrón de arquitectura más que como una tecnología.

REST intenta resolver algunos problemas presentes en otros estándares de Web Services como SOAP, sobre todo pretende simplificar el uso y favorecer el consumo de los servicios por una gama amplia de dispositivos.

El patrón de arquitectura de REST aprovecha los métodos existentes en el protocolo HTTP y una de las nociones más importantes es el *mapeo* de recursos. Hay 4 operaciones que se ejecutan generalmente sobre los recursos, y estas son "CREAR", "RECUPERAR", "ACTUALIZAR", "ELIMINAR". Estas operaciones estarán relacionadas con 4 métodos HTTP, algunos de los cuales no son muy usados por los navegadores.

Acción sobre el Recurso	Método HTTP asociado
CREAR	POST
RECUPERAR	GET
ACTUALIZAR	PUT
ELIMINAR	DELETE

Tabla 5.1:Mapeo de REST sobre los métodos HTTP

Las respuestas de los servicios web REST pueden ser enviadas en XML o JSON. Ambos métodos representan alguna ventaja en función del contexto. Por ejemplo si el consumo va a hacerse desde un navegador, probablemente una respuesta en JSON sea más adecuada pues es interpretado directamente por Javascript.

Otro punto importante es el *mapeo* de los recursos por URL, técnica que ya es conocido por su amplio uso en los *frameworks* que implementan el patrón MVC en web. Por ejemplo en el contexto de un sistema que programa tareas, se puede tener lo siguiente:

```
http://dominio/tarea/5
```

El resultado será un reporte de la tarea identificada por 5, que no necesariamente tendría que ser el identificador en la Base de datos, porque podría representar un problema de seguridad.

De este modo todas las operaciones sobre las tareas serán ejecutadas sobre la URL http://dominio/tarea. Solo que las diferentes operaciones serán llamadas con diferentes métodos HTTP. Esta discriminación la debe hacer el servidor.

4.1.5. Selección de tecnología para la interfaz web

Para implementar la interfaz web se consideró usar portlets porque esta alternativa estuvo muy extendida en el mundo Grid. Sin embargo, las funcionalidades de Legión Framework hacían poco factible el uso de tecnología como esta por ser algo restrictiva. La idea detrás de los portlets es extender un contenedor o portal con

componentes reutilizables, esto ya imponía adecuarse al manejo de usuarios, sesiones y podría complicar la gestión de procesos invisibles de pooling a los recursos de cómputo o Backends. Por lo que se optó por desarrollar un pequeño framework web más acorde a los requerimientos del proyecto.

PORTLETS

La tecnología de portlets ha sido ampliamente utilizada en el mundo de las Grids. Un portlet es una pequeña aplicación web que existe en el contexto de un portal. De manera precisa existe dentro de un contenedor de portlets y este es una extensión del concepto de *Servlet* en Java Enterprise Edition (JEE).

El primer estándar de portlets es el JSR-168 y es conocido como Portlets 1.0. Fue ideado por IBM y desarrollado como parte del Java Community Process. La siguiente versión del estándar es la JSR-286 y es conocida como Portlets 2.0. Esta versión tiene compatibilidad binaria respecto a la anterior, sin embargo, incluye mejoras en cuanto a la interacción entre portlets y con el contenedor.

Se realizaron pruebas con GridSphere y Liferay usando el contenedor de Servlets Apache Tomcat 6.x. GridSphere es compatible con IBM WebSphere, que es el contenedor de Portlets de IBM. La última versión consultada es la 3.1 y data del 2010. Es usado en el mundo de Grid pues se originó en un proyecto de la Comunidad Europea para este fin a comienzos de la década del 2000. Un proyecto conjunto le brinda capacidades para integrarse a Globus Toolkit 4 y se llama GridPortlets, sin embargo este proyecto no se ejecuta correctamente en Tomcat6 a diferencia del contenedor. Otro paquete más reciente que brinda interoperabilidad con Globus Toolkit y Glite es VineToolkit.

Liferay es el contenedor open source más usado, sin embargo su principal motivo de desarrollo no fueron las aplicaciones de Grid como en el caso de GridSphere. Por otro lado la suite VineToolkit se adapta a este contenedor.

4.1.6.Implementación de un framework web simplificado

Se implementó un framework web simplificado para dar soporte a las características particulares de Legión Web Interface. Esto incluye el desarrollo de componentes reutlizables como los mencionados a continuación.

Legión Table

Para facilitar la presentación de datos comunes como texto, fechas, pero también de información menos estándar como barras de progreso actualizadas por ajax (para mostrar el progreso de las tareas), botones de descarga, componentes de entrada de texto, archivos entre otros.

Base Controller

El BaseController fue desarrollado como una clase que extiende al Servlet y que incluye métodos que implementan el patrón MVC y de la que deben heredar los Controladores del sistema. A continuación se listan algunas de las funcionalidades desarrolladas.

- Redirigir a los usuarios dentro del sistema ocultando la estructura interna mediante llamadas usando un hash MD5.
- Anotaciones para indicar que métodos se van a exponer por "GET" o "POST" y redirección automática para estos métodos.
- Validación de permisos para acceder a un método en particular.
- Registrar en el log las acciones del usuario en el sistema.
- Método para obtener rápidamente: Nombre de usuario, dirección de IP, página de la que proviene el request.

4.1.7.Implementación del Template Interpreter

Con el fin de que el TemplateInterpreter sea reemplazable, y así se pueda variar el comportamiento por defecto del intérprete de plantillas que se provee "DefaultTemplateInterpreter", se especificó una "interfaz" que deben implementar las clases que realicen la tarea de interpretación de plantillas.

```
public interface ITemplateInterpreter {
    public String interpret(String template);
    public void   setForm(Form form);
    public void   setFormValues(HashMap<String,String> formValues);
    public void   setProjectValues(HashMap<String,String> projectValues);
    public void   setOtherValues(HashMap<String,Object> otherValues);
}
```

La clase que implemente la interfaz ITemplateInterpreter deberá estar en el paquete **pe.edu.pucp.legion.webinterface.templateinterpreter.extensions**, el nombre de esta clase debe terminar en "TemplateInterpreter" ej. "**Extended*TemplateInterpreter***". Después de esta operación, la clase estará disponible para ser seleccionada, pues la lista es cargada por inspección del mencionado paquete.

Esta operación permite que se implementen un conjunto de etiquetas que puedan ser usadas en un tipo particular de creación de tareas, o también podría ser útil al momento de implementar la integración con otro tipo de sistemas de computación en Grid.

4.1.8.Implementación del Task Creator

El TaskCreator fue desarrollado para convertir la entrada original de los usuarios a través del formulario en una representación universal de la tarea a ser creada. Se mencionó anteriormente que el TaskCreator es una interfaz, y aunque se provee una clase por defecto, esta puede ser extendida o incluso re implementada en su totalidad. Como no es adecuado crear la tarea en el backend de cómputo de manera inmediata (tiempo de espera), esta representación universal de la tarea se almacena en un buffer. La tarea se encontrará entonces en estado "Waiting Creation", lo que indica que el backend de computación (donde finalmente se ejecutará la tarea) no ha sido contactado hasta ese momento.

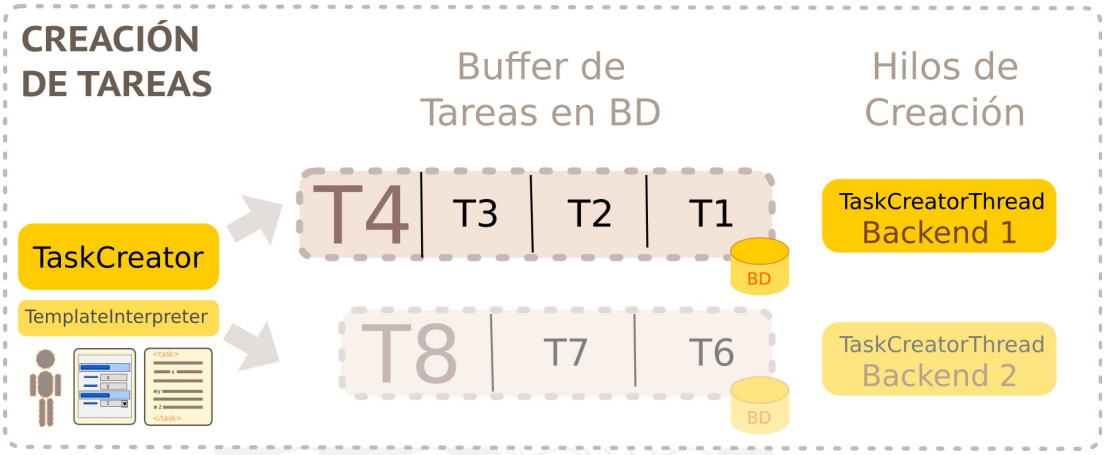


Figura 4.1:Proceso de creación de tareas

La decisión de implementar el buffer en la base de datos tiene que ver con que esta no es volátil; si por alguna razón la tarea no hubiera sido creada, no se perdería aun si el servidor se apagara tanto de manera sorpresiva como programada.

Un hilo es ejecutado periódicamente para cada backend que contenga tareas en espera de creación. El acceso a los backends es controlado mediante un semáforo de modo que no se saturen las conexiones a la capa de servicios web. Para ejemplificar lo importante de esta medida de control, se puede observar tareas compuestas por workunits en el orden de los miles: esto puede tomar unos cuantos minutos en el servidor BOINC.

4.1.9.Implementación de un hilo de creación de tareas

El hilo de creación de tareas se inicia a través de un *Listener* que se lanza al momento de creación del contexto (un ámbito común a toda la aplicación web en Java EE). Además, se guarda una referencia al hilo como atributo del contexto, de modo que se pueda monitorear su estado, detenerlo, y eventualmente volverlo a iniciar, desde la misma aplicación web.

El hilo de creación de tareas tiene el propósito principal de dosificar las operaciones de creación en un backend determinado, de modo que se evite cualquier tipo de

saturación de Legión Web Services. Sin embargo, las operaciones realizadas sobre un backend no implican congestión en otro, por lo que, la creación de tareas en backends distintos puede darse en paralelo. Es por esto que el hilo en mención ejecuta periódicamente “Task Creator Workers” , uno por cada backend que tenga tareas en estado de “Waiting Creation” en el buffer. Cabe resaltar que un pequeño retardo entre peticiones es establecido.

Para garantizar un acceso restringido a los backends, se implementó una clase “SemaphoreHelper” que mantiene referencias a los semáforos que regulan el acceso a los backends. Estos semaforos son compartidos por el hilo de creación de tareas, el hilo de eliminación de tareas y el hilo que consulta el progreso de estas, ya que todas estas operaciones requieren de Legión Web Services.

Cabe resaltar que la operación de creación de tareas podría fallar por diversas razones, como por ejemplo: un archivo descriptor con errores semánticos (la sintaxis es validada a discreción del usuario usando una especificación DTD), caso por el cuál se entiende que la operación de creación de una tarea puede derivar en dos resultados posibles: la tarea pasa a un estado de ejecución (tras una creación satisfactoria), o la tarea pasa a estado fallado.

4.1.10.Implementación de un hilo de actualización de tareas

El hilo de actualización de tareas también se inicia desde un *Listener* que monitorea la creación y destrucción del contexto de la aplicación web. El propósito de este componente es mantener la coherencia entre los datos almacenados en Legión Web Interface y los de Legión Web Services. Si una tarea tuviera un porcentaje de avance diferente, o hubiera finalizado (ya sea satisfactoriamente o fallando), este cambio se debería reflejar en la aplicación web.

Para evitar mútiples consultas al servicio web y tiempos elevados de respuesta, cuando el usuario consulta el estado de una tarea, se usa como fuente de información a la base de datos de la aplicación web. A diferencia de los web services, esta está especialmente preparada para soportar altos niveles de carga.

Dependiendo el nivel de resolución deseado se puede variar la frecuencia de consulta. Sin embargo, esto no es crítico en un ambiente de High Throughput Computing (HTC), ya que las tareas generalmente son de alta duración.

Al igual que el hilo de creación de tareas, este también inicia periódicamente un hilo para cada Backend que tenga tareas en el estado “Executing”, haciendo consultas espaciadas por un retardo que permite no sobrecargar al servicio web. Es importante notar que en este caso también se usa la clase “SemaphoreHelper” para regular el acceso a la sección crítica.

4.1.11.Implementación de un hilo de eliminación de tareas

El hilo de eliminación de tareas al igual que los dos anteriores, está diseñado para controlar el acceso al servicio web en términos de concurrencia. Este hilo permite que tareas que son eliminadas por el usuario sean canceladas en sus respectivos

Backends sin saturar a los servicios web y sin exigir al usuario que espere por el resultado de la operación.

Este hilo también tiene una referencia como parámetro del contexto, de modo que puede ser accedido para consultar su estado, detenerlo o volverlo a iniciar. Así, el administrador puede restringir los periodos en los que el hilo, al igual que con los anteriores, deba ejecutarse.

Una característica distintiva de este componente es que no se lanza un hilo por cada Backend que tenga tareas con el estado “Waiting Deletion”, sino que el hilo consulta por todas las tareas que tengan este estado y las va eliminado progresivamente. Si el resultado de la eliminación fuera satisfactorio, entonces el estado de la tarea (que por cierto, ya es visible a los usuarios) pasaría a “Deleted”; por el contrario, si el resultado de la eliminación no fuera satisfactorio, el estado permanecerá en “Waiting Deletion”, de modo que fuera eliminado en un próximo bucle de este hilo.

4.1.12.Ventajas del lenguaje de programación escogido

Al escoger el lenguaje de programación (Java) y consigo, las librerías que darían soporte a los distintos componentes de Legión Web Interface, se obtuvo una lista de beneficios que se detalla a continuación:

POOL de Conexiones a la BD

Al usar el framework MyBatis en conjunto con el Contenedor Servlet Apache Tomcat 7, se pudo configurar el acceso a datos mediante un pool de conexiones a la BD para facilitar la programación. La capa de acceso a datos (DAO) resultaría más sencilla de este modo (miles de líneas de código ahorradas respecto a JDBC). Aún así, el mayor beneficio se observa al desplegar Legión Web Services, pues en este pool de conexiones está especialmente preparado para que el acceso a la base de datos sea restringido a una cantidad, que se entienda manejable, de conexiones a la Base de Datos. Cabe resaltar que en caso de requerir una nueva conexión el sistema permite que se reutilicen conexiones, reduciendo la saturación. Además, siempre mantiene un número mínimo de conexiones abiertas e inmediatamente disponibles para el sistema: de este modo, los tiempos de respuesta del sistema se ven reducidos significativamente.

Librería interoperable para servicios web

Otro beneficio de la plataforma escogida es la disponibilidad de librerías para servicios web muy desarrolladas, y sobre todo ampliamente usadas. Aunque ya se mencionaron las ventajas de trabajar con Axis, la librería open source de Apache, cabe resaltar que esta no esta disponible en otras plataformas open source para aplicaciones web.

4.2.Pruebas

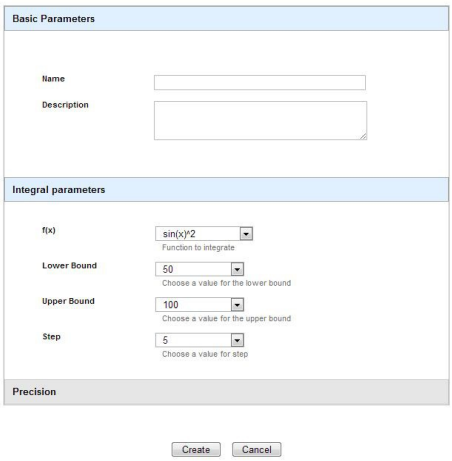
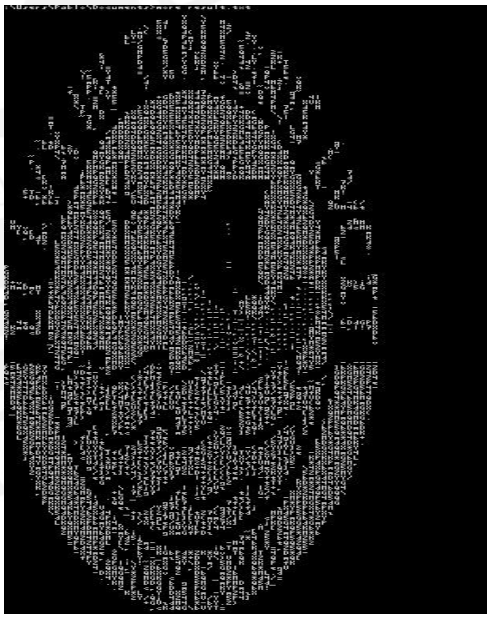
Legión Web Interface ha sido ampliamente probado, sin embargo las pruebas de integración permitieron comprobar que toda la arquitectura funcionaba en conjunto:

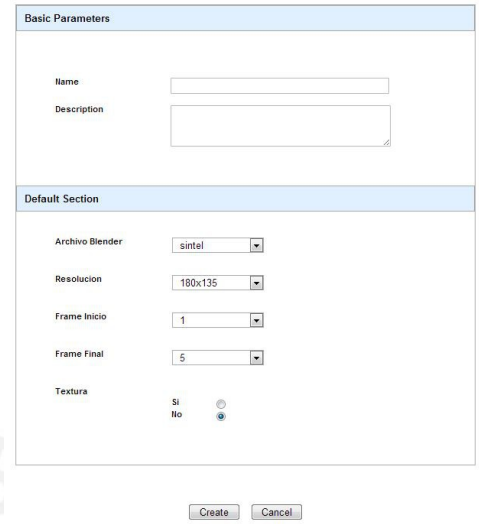
4.2.1.Pruebas de integración

Flujo 1: Crear proyecto y enviar una tarea

Seq	Detalle	Notas
1	Nuevo proyecto	-
2	Paso 1 de creación: Datos generales	Combinar opción: FTP / HTTP
3	Paso 2 de creación: Formulario	Crear formulario gráficamente / textualmente
4	Paso 3 de creación: Seleccionar usuarios	Seleccionar tanto usuarios suscriptores como creadores.
5	Paso 4 de creación: Configurar Web Services	Usar el botón de “Prueba de Conexión”
6	Paso 5 de creación: Crear plantilla del descriptor de tareas	Probar el botón “Verificar” (validación DTD)
7	Paso 6 de creación: Subir archivos	Subir archivos ejectutables / otros archivos
8	Guardar y cargar el nuevo proyecto	-
9	Seleccionar nueva tarea y enviarla, esperar unos segundos hasta que se cree en el servidor LWS.	La creación no es inmediata, pero sí su transformación a una descripción universal de la tarea.
10	Esperar a que la tarea termine (se comprueba que se actualiza, consultando a LWS) y se descarga el resultado.	

Resultados de la prueba

Proyecto	Descripción	Resultado
Integrador	Se programó un programa en C, que es capás de calcular una integral definida numéricamente provisto que recibe la función "f(x)", un límite inferior, y la cantidad de unidades que se integrarán a partir de este último. Se desarrolló una Interfaz para que cada parte de una integral se calcule como una unidad de cómputo independiente.	 Resultado: OK
Image Script Python	Se usó un código que convertía imágenes a texto en Python y se adaptó para que una sección de la imagen sea procesada por cada unidad de trabajo. Lo que se quería conseguir era demostrar la integración con aplicaciones instaladas en el nodo de ejecución como Python. Esta demostración es válida para todo tipo de scripts (Python, Octave, R, etc).	 Resultado: OK (División en 5 workunits).

Renderizar videos 3D en Blender	Se creó un proyecto para renderizar videos 3D en Blender, de modo que en cada unidad de trabajo se cree una imagen, luego podría ser unidas teniendo en cuenta cuantos FPS (frames per second) se están usando.	  Resultado: OK Nota: Los nodos de ejecución requieren al menos 4GB para renderizar videos.
Validación de modelos estadísticos con R	Se creó un proyecto para dar soporte a las necesidades de simulación de un profesor del departamento de Matemáticas. Se trataba de una simulación de montecarlo usando el	(Sin Imagen)

	software R, se barrieron 5 parámetros para tamaños distintos de muestra. Hubiera tardado cerca de dos semanas en una sola computadora. En el sistema Legión con 100 computadoras tomó 12 horas. *El código de este proyecto no está disponible en la presente tesis.*	Resultado: OK

Nota: Los archivos fuente de estos proyectos se incluyen en el cd adjunto.

Flujo 2: Eliminación de tareas

Seq	Detalle	Notas
1	Seleccionar una tarea, seleccionar “Eliminar”	-
2	Verificar que desaparece de la lista de tareas	-
3	Verificar que cambia a estado “2 – Cancelado” después de hacer la llamada al web service.	Usar cliente MySQL Verificar Log Legión Web Interface / Legión Web Services.

Resultado: OK

5. Observaciones, conclusiones y recomendaciones

En el presente capítulo se detallan las observaciones y conclusiones tras la construcción del sistema. Se explica las principales razones por las cuales el proyecto y sus aportes son importantes para proveer de capacidad de cómputo a las universidades. Al finalizar se brindan recomendaciones y trabajo futuro en el área de la presente tesis.

5.1.Observaciones

El presente proyecto resulta importante, en su conjunto, por dos razones principales. La primera es que se diseña una capa intermedia que provee servicios web que pueden ser consumidos por cualquier aplicación. Esto representa un paso importante para la comunidad de proyectos que usan BOINC e incluso a los que no lo usan. Por ejemplo, es posible que una universidad en cuyas instalaciones exista hardware de alto desempeño pueda ampliar su gama de *backends* usando la misma infraestructura con la que ya cuenta. La ventaja de BOINC es que puede escalar fácilmente a un gran poder de cómputo con bajo costo, ya que es posible contar con donantes de capacidad de cómputo alrededor de todo el mundo.

La segunda razón por la que el presente proyecto resulta importante es que se reduce de manera significativa el tiempo de construcción de la interfaz para el investigador. Un administrador con experiencia de pocos meses en Legión Framework podría crear un nuevo proyecto de cómputo distribuido en menos de un día, y sin necesidad de programar nada explícitamente, lo cual quiere decir que Legion es un proyecto replicable en cualquier parte del mundo.

5.2.Conclusiones

Tomando como base Legión Framework, se ha propuesto un modelo para cómputo de alto rendimiento de bajo costo. Este modelo, incluye una interfaz web que permite el fácil acceso a tecnologías de Desktop Grid (el framework es open source) y el uso de virtualización para lograr aislamiento (por seguridad) con máquinas homogéneas (menor costo de adaptación de aplicaciones). Es así como los costos de administración se ven también reducidos. Este modelo es muy atractivo en situaciones en las cuales un cluster dedicado no es asequible. Esto último es muy común en países en desarrollo, lo que limita en muchos casos las áreas de investigación en la que pueden participar los investigadores de estas instituciones.

5.3.Recomendaciones y trabajos futuros

5.3.1.Barrido de parámetros en Legión Web Interface

Una forma muy común de crear tareas del tipo *bag-of-tasks* es el barrido de parámetros. Esta operación consiste en combinar todos los posibles valores que podrían tener los parámetros, usualmente para generar los argumentos para ejecutar una aplicación del tipo batch. En Legión Framework esta tarea es delegada al componente LWS. Sin embargo, incluir esta funcionalidad en el componente LWI puede resultar muy útil en una posible integración con los servicios web de BOINC. Ya que en el BOINC Workshop 11 se decidió delegar tareas de alto nivel como el barrido de parámetros y manejo de dependencias a las aplicaciones de envío remoto de tareas.

5.3.2.Uso de técnicas de Machine Learning para predecir tiempo de ejecución

Se puede usar técnicas como Neural Networks (NN), que capturan el comportamiento de funciones no lineales, para pronosticar el tiempo de duración de las tareas al momento de ser enviadas. Si fueran muchos parámetros la aplicación de técnicas de reducción de la dimensionalidad como Principal Component Analysis (PCA) puede ser estudiada. Se debería usar los parámetros que son registrados en la tabla t_task_parameter para entrenar los clasificadores (respetando su naturaleza dinámica ya que al modificar el formulario de envío de tareas, los parámetros podrían variar).

5.3.3.Analizar integración con OurGrid

OurGrid es un middleware para cómputo Peer-to-peer desarrollado en Brasil, especialmente desarrollado para trabajar tareas del tipo Bag-of-Tasks, por lo que es muy similar. El equipo que lo desarrolla se puso en contacto, manifestando su interesés en crear un capa LWS para OurGrid.

5.3.4.Analizar integración con Condor

Condor es un sistema HTC muy utilizado en el mundo, es más, la especialidad de Física de la PUCP tiene un servidor con este sistema. Para integrarlo se podría crear una LWS for Condor.

5.3.5.Implementar un Account Manager para trabajar con Legión

La arquitectura de BOINC incluye un componente llamado Account Manager. De manera general, permite delegar la configuración del comportamiento de cliente de BOINC a un tercero que actúa en representación del dueño de la computadora.

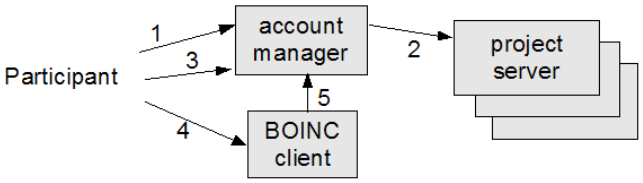


Figura 5.1:Diagrama de funcionamiento de un Account Manager [BOINC 2011]

El account manager podría ser un sistema independiente que brinde servicios web para Legión para conocer: Cantidad de nodos, porcentajes asignados, etc.

5.3.6.Implementar una Grid Nacional

Se propone que en la integración de un account manager y Legión Framework + BOINC se pueda construir una Grid Latinoamerica usando las redes avanzadas como CLARA.

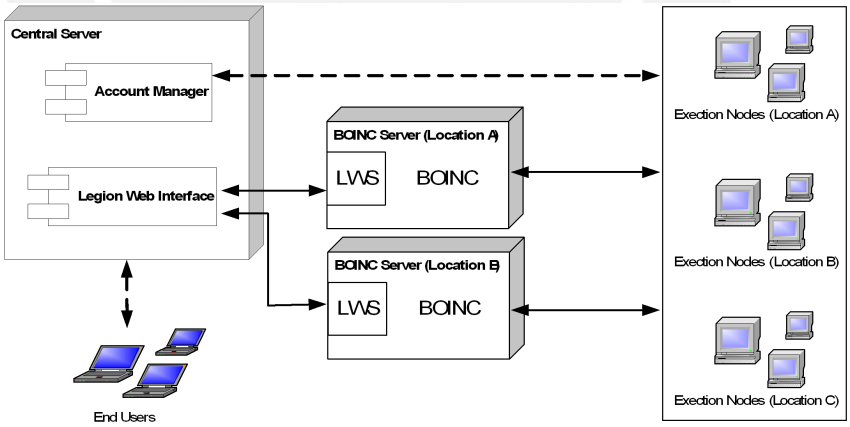


Figura 5.2: Propuesta de arquitectura

Bibliografía

[Ries 2010] C.B. Ries, Visu@IGrid - Realization of a development environment for model-based design and code generation of heterogeneous client server applications, Code Generation Conference, England, Cambridge, 17. June, 2010

[Andrzejak 2010A] Artur Andrzejak, Derrick Kondo, and Sangho Yi : Decision Model for Cloud Computing under SLA Constraints, 18th Annual Meeting of the IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2010), Miami Beach, Florida - August 17-19, 2010.

[Andrzejak 2010B] Artur Andrzejak, Derrick Kondo, David P. Anderson: Exploiting Non-Dedicated Resources for Cloud Computing, 12th IEEE/ IFIP Network Operations and Management Symposium (NOMS 2010), Osaka, Japan, 19-23 April 2010.

[Segal 2010] Ben Segal, Jarno Rantala, David Weir. LHC Volunteer Cloud Computing Project. VMWrapper. Disponible en: http://code.google.com/p/boincvm/wiki/VMWrapper Consultado 18/11/2011.

[Kacsuk 2006] Kacsuk, Podhorszki, Kiss. Scalable Desktop Grid System, High Performance Computing for Computational Science - VECPAR 2006, Springer, 2006.

[Anderson 2004] Anderson. BOINC: A system for public-resource computing and storage. Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, 2004.

[Schwaber 2004] Schwaber, Agile Project Management with Scrum, Microsoft Press, 2004.

[Domingues 2008] Celio Domingues Goncalves, Servico de Seguranca de Middlewares, http://www.facom.ufu.br/posgrad/wd2008/artigos/celio.pdf , 2008.

[Schulze 2007] Schulze. Projeto GT VCG. SBRC - Simpósio Brasileiro de Redes de Computadores, VIII WRNP - Workshop da Rede Nacional de Pesquisa, 2007.

[Bazinet 2009] Bazinet. The Lattice Project: A Multi-Model Grid Computing System, Thesis Presentation. 2009 http://lattice.umiacs.umd.edu/files/bazinet_thesis_presentation.pdf Consultado el: 30 de marzo de 2010

[Giorgino 2010] Giorgino, Harvey, de Fabritiis. Distributed computing as a virtual supercomputer: tools to run and manage large-scale BOINC simulations. Dynamics, High Performance Computing. Elsevier. 2010.

[Pow-Sang 2008] Pow-Sang, Flores. Diapositivas de Clases del Curso Ingeniería de Software, Ciclo: 2008-2.

[NIH 2010] RUP Fundamentals. Fuente: http://era.nih.gov/docs/rup_fundamentals.htm ,Consultado 30 de marzo 2010

[Ambler 2010] Ambler, Traducción del AUP. Fuente: http://cgi.una.ac.cr/AUP/index.html Consultado 30 de marzo 2010

[Kniberg 2007] Kniberg. Scrum and XP from the trenches. C4Media, 2007.

[Berkeley 2010] Wiki del sitio oficial de BOINC de la Universidad de California en Berkeley. http://boinc.berkeley.edu/wiki/How_BOINC_works Consultado 30 de marzo 2010.

[Wells 2000] Extreme Programming. http://www.extremeprogramming.org/map/project.html Consultado 30 de marzo 2010.

[Soaplib 2010] Soaplib, sitio oficial. http://wiki.github.com/jkp/soaplib/ Consultado 14 de julio 2010.

[Rohit 2011] Rohit, E., Nguyen, H., Kanna, N., Subhlok, J., & Gabriel, E. (2011). A Robust Communication Framework for Parallel Execution on Volunteer PC Grids. *Computer*.

[Emmen 2010] Emmen, A. (2010). Introduction to Desktop Grid Computing. IDGF International Desktop Grid Federation.

[Winter 2011] Winter, S. C., Kiss, T., Terstyanszky, G., Farkas, D., Delaitre, T., & Reynolds, C. (2011). *Experiences with the University of Westminster Desktop Grid*. Meeting of the International Desktop Grid Federation.

Anexos