(Draft) La Infraestructura de Datos Espaciales de España en 2007.
Proyectos, servicios y nodos. 2007, p. 128-134. ISBN 84-690-8674-2

Un prototipo para acceder a comandos de Grass a través de servicios Web siguiendo la especificación de Web Processing Services

Gonzalo Sancho-Jiménez, Rubén Béjar, David Gayán, Pedro R. Muro-Medrano

Dpto. de Informática e Ingeniería de Sistemas - Universidad de Zaragoza
c/ María de Luna 1, 50018 Zaragoza
e-mail: {gsancho, rbejar, dgayan, prmuro}@unizar.es

Resumen

En este artículo se expone el proceso seguido para ofrecer funciones de tratamiento de datos ráster vía servicios Web siguiendo la especificación WPS de OGC. Para soportar esta funcionalidad se ha utilizado una implementación de tratamiento de datos consolidada y con perspectivas de futuro como Grass. Se estudian las decisiones tomadas en la elección de los programas para soportar la creación de las interfaces Web, la adaptación de los servicios, granularidad, sincronismo y formato de datos empleados para la comunicación.

Palabras clave: WPS, Grass, Deegree, Web Services.

(Draft) La Infraestructura de Datos Espaciales de España en 2007.
Proyectos, servicios y nodos. 2007, p. 128-134. ISBN 84-690-8674-2

1 Introducción

El objetivo de este trabajo es describir un método para permitir el tratamiento de datos ráster por medio de una serie de operaciones vía servicios Web, utilizando una aplicación open source de tratamiento de datos consolidada y con perspectivas de futuro como es Grass [1]. Para facilitar el acceso a estos servicios Web, se ha decidido diseñarlos siguiendo la especificación Web Processing Service (WPS) [2] del Open Geospatial Consortium (OGC).

Un WPS proporciona acceso a operaciones de datos geoespaciales de distinta complejidad mediante servicios Web como interfaces. Puede contener operaciones para tratar tanto datos vectoriales como matriciales provenientes de la red o del propio servidor. A pesar de que todavía se encuentra en fase de desarrollo es una especificación que ya está siendo implementada en diversas bibliotecas de componentes. Entre estas bibliotecas destaca Deegree [3], que es un Framework Java open source que implementa servidores compatibles con diversos estándares OGC (WMS, WCS, WFS, CSW, WAS). Actualmente se encuentra en desarrollo del borrador de la especificación WPS, quedando por implementar algunos detalles como el tratamiento de respuestas asíncronas. Existen otras bibliotecas que implementan esta especificación como la de Tigris [4] o 52n [5], pero se decidió apostar por Deegree porque es un framework muy completo con el que se había trabajado en otras ocasiones.

Para soportar la funcionalidad de tratamiento de datos ráster que se buscaba, se decidió usar Grass. Esta es una herramienta SIG desarrollada en C e inicialmente pensada para un entorno UNIX/Linux como aplicación de escritorio por lo que las instrucciones están orientadas a una interfaz de comandos, a pesar de que en las últimas versiones se ha incluido una interfaz gráfica. Contiene tanto herramientas de tratamiento de datos ráster como vectoriales.

2 Trabajo relacionado

Existen otras implementaciones que exponen funcionalidad de tratamiento ráster a través de servicios Web. A continuación se van a exponer los trabajos con más relevancia relacionados con el tratado en el artículo:

El prototipo creado por Canadian Geospatial Data Infrastructure (CGDI) [6], proporciona métodos basados de acuerdo a las especificaciones OGC de WPS para

operaciones triviales de datos geoespaciales utilizando el lenguaje Geolinked Data Access Service (GDAS). Su intención es reescribir GDAS y GLS con un perfil de aplicación WPS cuando se publique la versión definitiva de WPS. El prototipo presentado realiza operaciones simples que no necesitan de una herramienta específica para el tratamiento de datos.

En el seno del proyecto Geobrain [7] se encuentran en el desarrollo de interfaces de servicios Web SOAP basándose en funciones no interactivas de Grass. Las funciones que incorporan son los comandos básicos Grass, dejando al cliente final la orquestación de estos servicios. Con esto se consigue ofrecer los comandos de Grass vía servicio Web con máxima granularidad. Esto es muy flexible pero puede resultar más lento al aumentar el tráfico de red necesario para completar cualquier operación que involucre una cantidad significativa de comandos Grass. En el trabajo realizado, se desean realizar operaciones de más alto nivel, que sean auto completas, pero que permitan el solapamiento, mejorando de esta forma la velocidad y el tráfico de la red, abstrayendo la capa GRASS al usuario final.

3 Diseño del prototipo

En esta sección se describen los aspectos fundamentales del diseño del prototipo presentado. Primero se analiza la problemática de adaptar una aplicación como Grass a un entorno concurrente y distribuido de servicios Web. Para esto se muestra inicialmente el modelo de trabajo de ambos. A continuación se trata el nivel de granularidad y asincronismo de los servicios ofrecidos acorde con el tráfico de datos y la capacidad de reutilización de las funcionalidades que se quiere ofrecer. Después se muestra el diseño del prototipo, resaltando mecanismos de interacción entre Deegree y Grass, mostrando las distintas alternativas. Finalmente se detallan algunos aspectos de diseño de bajo nivel que han surgido durante la fase de implementación.

3.1 Introducción

El prototipo pretende ser un ejemplo que proporcione dos funcionalidades de tratamiento de datos en formato ráster. Por un lado se pretende obtener el perfil del recorrido producido al seguir una serie de localizaciones con una resolución y en un mapa en concreto. La segunda funcionalidad consiste en obtener la visibilidad desde una localización y con una distancia de visión dadas.

3.2 Modelo de trabajo en Grass y WPS

Grass es una herramienta de trabajo multiusuario diseñada para un acceso desde terminales, y con un sistema de directorios que permiten que varios usuarios puedan acceder a los mismos datos sin problemas. Con este fin se definen tres tipos de variables que especifican el directorio en que se trabaja: *database*, *location* y *mapset*, ***Figura 1***. Estas variables son definidas como variables de entorno en el ordenador cliente desde el que se ejecuta el terminal.

- La variable *database* define el directorio donde se va a trabajar y de donde cuelgan los distintos *location*.
- Un *location* define un proyecto distinto, este tiene un único sistema de proyección.
- Los *mapset* pueden ser definidos como los directorios de trabajo del usuario. Cada usuario solo puede acceder y modificar los datos de su espacio de trabajo. Existe un área de trabajo común de todos los usuarios de un mismo *location*, se trata de un *mapset* denominado *permanent* y del que pueden obtener datos, pero no pueden modificar su contenido.

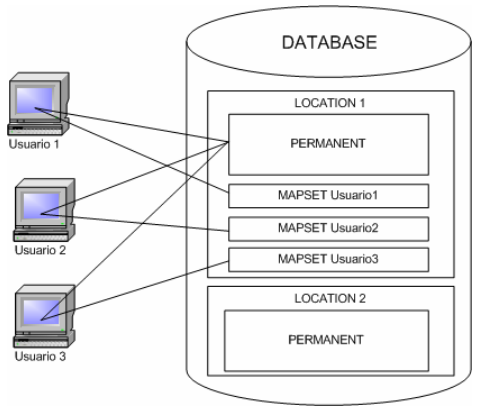

Figura 1: Modelo de trabajo de directorios en Grass.

Por otro lado el modelo de trabajo de WPS está basado en el modelo de los servicios Web que son protocolos sin estado en los que no se mantiene ningún tipo de información entre las distintas peticiones.

Las funcionalidades que se desean ofrecer desde WPS, son independientes del usuario que realiza la petición y no existen unos proyectos de trabajo en los que los

usuarios comparten información, tan solo se piden unos datos, se trabaja con estos y se obtiene un resultado.

3.3 Diseño del prototipo

En las operaciones del WPS se ha escogido un nivel de granularidad y asincronismo acorde con los servicios a prestar. A continuación se comentan los aspectos más relevantes.

La granularidad es el tamaño y división de las operaciones que se ofrecen. El concepto de granularidad parte del principio de que es más fácil reutilizar unidades más pequeñas permitiendo seleccionar aquellas partes que interesen y descartar aquellas que no son adecuadas en un nuevo contexto [8].

En los servicios Web, se debe alcanzar un compromiso entre el nivel de granularidad de las operaciones y el tráfico de información entre estas. Una opción es implementar consultas para cada una de las operaciones de Grass y que el usuario final sea quien pueda construir su propia consulta a partir de ejecutar las instrucciones remotamente. Esto aumenta la reusabilidad de los servicios a cambio de aumentar la complejidad del cliente final, dejándole la tarea de composición del servicio y aumentando el tráfico de datos. De esta forma no se proporciona ninguna solución que el cliente no pudiera resolver con una instancia Grass local. La solución que proporciona mayor reusabilidad y optimización del tráfico son las operaciones mínimas con resultados útiles, pero permitiendo la comunicación y compatibilidad entre ellas. En el caso que ocupa al prototipo, las operaciones a ofrecer son obtener línea de visibilidad y obtener perfil, no viendo útil la descomposición de estas en unidades menores.

Otro factor importante que se ha tenido en cuenta en el diseño ha sido la asincronía. Las tecnologías Web utilizadas hoy en día utilizan respuestas síncronas para las peticiones al servidor, lo que no es muy recomendable cuando los tiempos de respuesta pueden ser largos. En los SIG el tiempo de respuesta puede llegar a ser especialmente largo debido a la cantidad de información a tratar. Para gestionar este tipo de procesos, la especificación WPS tiene un mecanismo de respuesta asíncrona y de obtención de resultados, que puede verse en la ***Figura 2***.

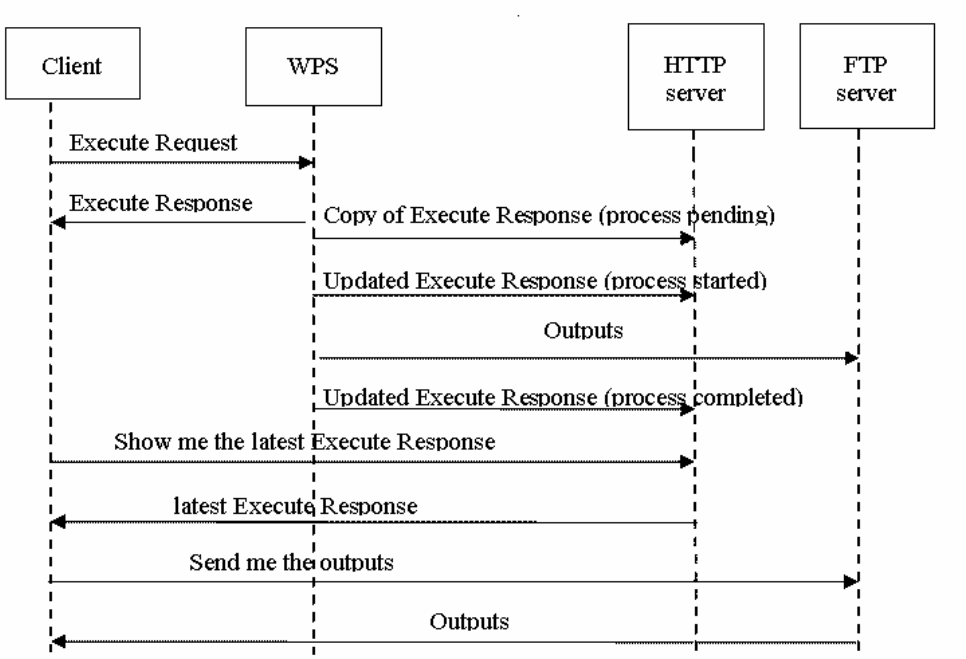

Figura 2: Respuesta asíncrona del servicio WPS.

Según indica la especificación, ante una petición del cliente, se responde síncronamente informándole que su petición está en curso. A lo largo de la ejecución del proceso, el cliente va preguntando por el estado del resultado. Una vez que el se tienen los resultados y ante una petición del cliente, se le informa de la completitud de la operación y la dirección donde puede obtener los resultados. Por último el cliente descarga los resultados de la dirección que se le ha proporcionado. En la implementación de Deegree de los WPS no se incluyen este tipo de respuestas asíncronas por lo que en esta primera versión del prototipo no se incluye.

Al usar las funcionalidades Grass dentro de WPS es necesario realizar una modificación del modo de trabajo de la aplicación, del uso de los espacios de trabajo, manejo de los archivos y control de accesos concurrentes. La idea es ofrecer una serie de funciones independientes del modelo de trabajo Grass.

Las operaciones ofrecidas en el WPS son independientes entre sí, así que pueden considerarse, en el modelo de trabajo Grass, tanto como diferentes proyectos

(*location*), como distintos usuarios de un mismo proyecto (*mapset*). Si se considera que cada una de las operaciones trabaja en el *mapset* de un mismo *location*, se obliga a trabajar en un mismo sistema de referencia y esto puede suponer un incremento en el tiempo de cómputo debido a la proyección de los datos. La elección tomada consiste en trabajar con distintos *location* en cada operación. Este método de trabajo obliga a que cada vez que se realice una petición se cree un nuevo *location* y dentro de este un *mapset*, se trabaje con él y se almacene el resultado fuera para finalmente eliminarlos.

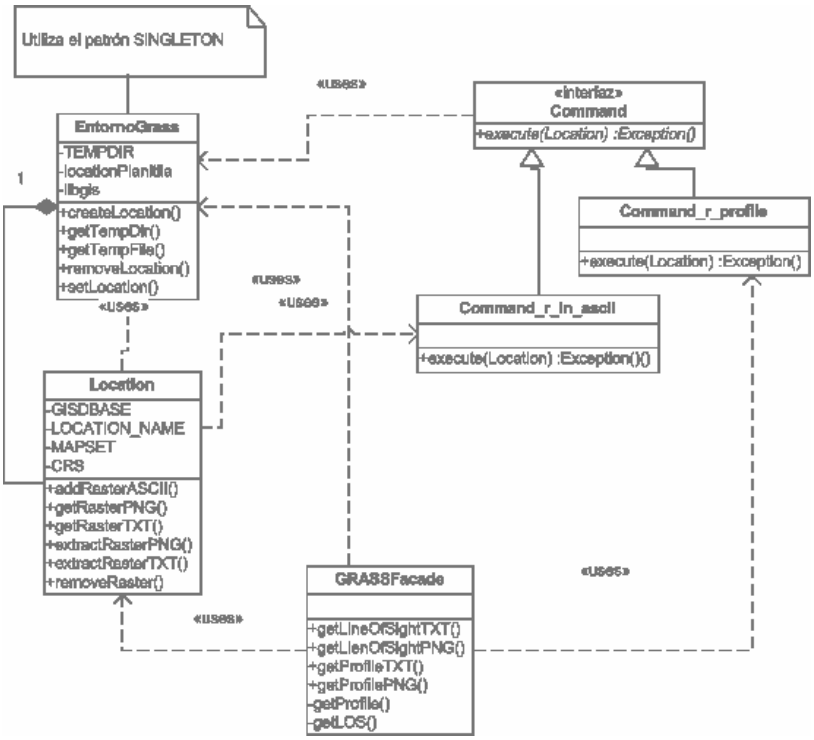

Figura 3: Modelo UML.

Observando el modelado UML de la ***Figura 3*** se pueden diferenciar las distintas partes que componen la abstracción realizada al modelo de trabajo de Grass. La parte de la aplicación que se relaciona con Deegree implementa el patrón *facade* [9], el cual sirve para dar una interfaz unificada y sencilla que haga de intermedia entre los programas. *EntornoGrass* se encarga de la gestión de los *location*, establecer las variables del sistema y mantener una instancia de un *location* denominado plantilla a partir del cual se crean los demás.

Las operaciones que proporciona Grass nativamente se traducen en clases que siguen el patrón *command* [9], consiguiendo que se pueda solicitar una operación a un objeto sin conocer el contenido de la operación ni su receptor. El patrón *command* consiste en encapsular la petición como un objeto parametrizando los métodos. Estas clases realizan todas las comprobaciones necesarias para en última instancia delegar la ejecución en la funcionalidad del programa proporcionado por Grass.

Los comandos Grass leen información de la configuración que está en las variables de entorno del sistema operativo. Por tanto si hay dos comandos ejecutándose concurrentemente y es necesario que lo hagan en distintos entornos (por ejemplo para depositar sus resultados en uno u otro directorio del sistema) es posible que alguno falle. Para solucionar el problema, antes de ejecutar una instrucción Grass se establecen las variables de entorno que redefinen el entorno donde se va a ejecutar. En el diagrama de flujo de la ***Figura 4***, se puede observar que se invoca a *EntornoGrass* desde *Command* para establecer los valores de las variables antes de ejecutar la instrucción, para redefinir estas variables.

Para entender mejor el modelo de trabajo del prototipo con Grass se va a detallar la ejecución de la petición de línea de visibilidad. En la ejecución de esta operación intervienen varias instrucciones Grass: con *r.inAscii* y *r.outAscii* se introducen y se obtienen datos en formato ráster del *location*, para poder realizar operaciones con los datos introducidos se establece la región de trabajo con *g.region* y con la instrucción r.LOS se obtiene la línea de visibilidad sobre los datos introducidos. En la ***Figura 4*** se expone el diagrama de flujo de esta operación. Inicialmente se pide un nuevo *location* al entorno para el almacenamiento de datos. En caso de ser la primera petición, carga el *location* plantilla a partir los valores del fichero de configuración *initial.proporties*. Se mantiene esta instancia para posteriores peticiones bajando la carga de trabajo de las operaciones. A partir de este *location* plantilla, se crea un nuevo *location* temporal en el cual se almacenan los resultados temporales. En algunas operaciones, es posible crear varios *location* con distintos sistemas de referencia para posteriormente intercambiar datos entre estos. Una vez iniciado el entorno de trabajo, se realizan las operaciones Grass comentadas para obtener el resultado. Cuando la operación ha terminado, un hilo independiente es el encargado de eliminar el *location* mientras se proporciona la respuesta.

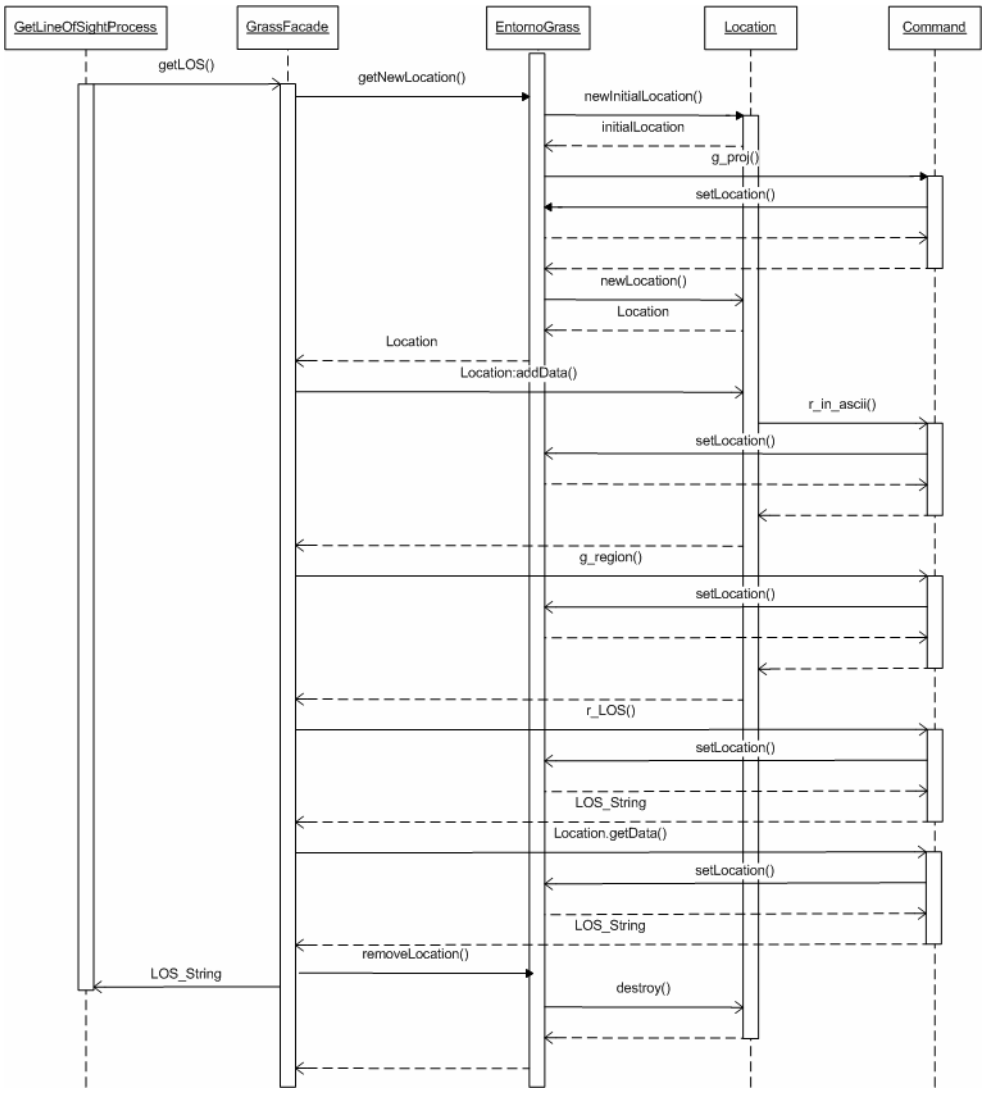

Figura 4: Diagrama de flujo.

Los datos calculados son devueltos a la clase *GetLineOfSightProcess*, que inicialmente realiza la llamada. Esta clase es el punto de conexión con Deegree. Para que el proceso pueda ser incluido en el framework de Deegree, esta clase extiende la clase abstracta Process implementando sus métodos abstractos

ProcessDescription, *ProcessOutputs* y *getProcessDescription*. El proceso también tiene que ser descrito en un fichero XML definiendo sus entradas y salidas.

Los servicios Web están diseñados para recibir peticiones y responderlas de forma concurrente. El modelo de Grass en que se basa no está diseñado para soportarlo, por lo que en esta primera versión, se bloquea el acceso concurrente a los WPS.

3.4 Diseño de bajo nivel

Para poder incluir Grass como aplicación desarrollada en C dentro de un servicio Web desarrollado en Java se pueden tomar varias alternativas. A continuación se discuten estas alternativas y se justifica la elección realizada en el prototipo.

Una adaptación de los servicios C de Grass a Java es dada por Alexandre Sorokine [10] y está incluida en las últimas versiones de Grass en el repositorio *src.grass.garden*. Consiste en la generación de varias DLL con las que, utilizando JNI, el sistema puede usar las funcionalidades Grass como funcionalidades propias. Se trata de una buena solución pero tan solo incluye las funcionalidades más sencillas. La implementación de nuevas funcionalidades corre a cargo del programador teniendo que modificar directamente el código C. Esto le hace perder el atractivo inicial, ya que impide al sistema realizar actualizaciones directas a partir de las nuevas versiones de Grass. A pesar del problema planteado, se trata de una solución muy interesante y en el prototipo se incluyen las funcionalidades básicas ya implementadas.

JGrass [11] es una iniciativa libre consistente en la creación de un entorno clónico de Grass en Java, implementando todas las funcionalidades desde cero. Se trata de una solución que en un futuro puede ser interesante, pero en la actualidad es un sistema poco maduro.

Una tercera propuesta consiste en tener instalada en el equipo servidor una versión completa de Grass compilada y llamar directamente a los ejecutables desde el código Java. Esta solución permite la actualización directa de la aplicación Grass, pero delegar la ejecución del programa en segundas aplicaciones aumenta la complejidad del sistema y puede ser una nueva fuente de problemas.

La solución adoptada consiste en la fusión de dos de las soluciones propuestas. Por un lado se integran las DLL con las funcionalidades básicas, al ser utilidades que no van a mejorar ni cambiar a lo largo del tiempo, y en la parte de las

funcionalidades más específicas, se utilizan directamente los binarios de Grass permitiendo las actualizaciones de nuevas funcionalidades sin necesidad de realizar un gran desarrollo. Para que la ejecución de estas funciones fuera del entorno no provoque errores, se tienen en cuenta todo tipo de excepciones que puedan provocar.

4 Trabajo futuro

El prototipo construido es funcional, pero deja algunos aspectos por implementar. En el caso de la concurrencia, los servicios Web proporcionan soporte para las peticiones concurrentes, pero el prototipo la evita no permitiéndola. Ya se ha indicado que esto es un problema por la estrecha relación de los comandos de Grass con el entorno del sistema. Como trabajo futuro, se propone que en el momento que la instrucción establece el *location* a utilizar y lo carga en las variables de entorno, el sistema bloquee la ejecución de nuevas instrucciones hasta obtener los resultados, haciendo las instrucciones atómicas. A pesar de que estas no se ejecutan a la vez, se evita que interfieran entre sí permitiendo la concurrencia de las operaciones.

5 Conclusiones

Se ha descrito el diseño de un prototipo que demuestra que es factible adaptar las funcionalidades de un sistema como Grass, al entorno de los servicios Web y en concreto a los WPS, los cuales requiere ejecuciones remotas, posiblemente asíncronas y por defecto concurrentes. Sin embargo esta adaptación no es inmediata, y el prototipo tiene un recorrido que realizar hasta poder llegar a un entorno de producción.

De este diseño se han detallado las decisiones tomadas respecto al nivel de granularidad y asincronía adecuados para los servicios WPS y la problemática de las interfaces entre el código Java (Deegree) y el código C de Grass.

Finalmente como resultado del estudio se ha implementado un prototipo funcional de WPS con las funcionalidades completas de línea de visibilidad y de obtención de un perfil.

Referencias.

[1] GRASS DEVELOPMENT TEAM. Grass, Geographic Information System (GIS). [en línea]. Disponible en Web: http://grass.itc.it [Consulta: 20 enero 2007]

[2] OPEN GEOSPATIAL CONSORTIUM. General Description of the CGDI, Web Processing Service (WPS). Draft 0.2.1 Enero 2005. [en línea]. Disponible en Web: http://www.opengeospatial.org/standards/requests/28 [Consulta: 12 enero 2007]

[3] OPEN GEOSPATIAL CONSORTIUM. Deegree. [en línea]. Disponible en Web: http://www.deegree.org/ [Consulta: 14 enero 2007]

[6] TIGRIS. Spring plug-in for OGC's Web Processing Service (WPS) interface. Disponible en Web: http://wpsint.tigris.org [Consulta 20 enero 2007]

[5] 52-NORTH. The 52N Web Processing Service. [en línea]. Disponible en Web: http://52north.org/ [Consulta 20 Mayo 2007]

[6] XIAOYUAN GENG.Agriculture and Agri-Food Canada/Agriculture et Agroalimentaire Canada. [en línea]. Disponible en Web: http://wms1.agr.gc.ca/GeoPS/GeoPSPage.html [Consulta: 20 enero 2007]

[7] UPENDRA DADI. Grass webservices. [en línea]. Disponible en Web: http://geobrain.laits.gmu.edu/products.htm [Consulta: 20 enero 2007]

[8] Lars Bernard, Max Craglia, Michael Gould, Werner Kuhn (2005): Towards an SDI Research Agenda. 11 EC-GIS (Sardinia), June 28- July 1, 2005.

[9] Buschmann, Meunier, Rohnert, Sommerlad, Stal – Wiley. Design Patterns: Elements of Reusable Object-Oriented Software. Publicado 1998 Addison-Wesley.

[10] ALEXANDRE SOROKINE. Grass-JNI (aka grass-java). [en línea]. Disponible en Web: http://sorokine.info/software.html [Consulta: 15 febrero 2007]

[11] CUDAM, HYDROLOGIS AND ICENS. JGrass, free multi platform, open source GIS based on GIS Grass. [en línea]. Disponible en Web: http://www.jgrass.org/ [Consulta: 13 enero 2007]