Implementación de geoprocesos como servicios web

por ***Pablo Javier Zader***

Presentado ante la Facultad de Matemática, Astronomía y Física
y el Instituto de Altos Estudios Espaciales Mario Gulich
como parte de los requerimientos para la obtención del grado de

MAGÍSTER EN APLICACIONES ESPACIALES DE ALERTA
Y RESPUESTA TEMPRANA A EMERGENCIAS

UNIVERSIDAD NACIONAL DE CÓRDOBA

Septiembre, 2016

Director: Mgr. Estefania Aylén de Elia
Co-Director: Dr. Ernesto G. Abril

A los Compañeros y Amigos que me entregó la vida a través de la experiencia en esta maestria, porque junto a ellos viví momentos únicos que quedarán por siempre en mi memoria, y siempre será muy grato volver a encontrarlos...

A mi Mamá especialmente por su ejemplo de Preseverancia y Amor, porque nunca dejó de luchar para que sus hijos crezcamos fuertes, nobles y honestos a pesar de los grandes dolores que esta vida le presentó...

A mis hermanas, sobrinos y amigos de la vida...

Agradecimientos

A la Universidad Nacional de Córdoba y a CONAE por haberme dado nuevamente la oportunidad de seguir formándome y de ese modo, seguir brindando lo mejor de mi en mis tareas laborales. A la Agencia Espacial Italiana que gracias al convenio con CONAE, me permitió realizar una experiencia internacional en Italia.A la Fundazione Edmund Mach de la provincia de Trento - Italia donde realicé mi pasantía de investigación. Al Instituto Gulich por haberme formado con los mejores profesionales de la Argentina en el ámbito geoespacial. A los docentes del instituto por haberme brindado sus conocimientos tan generosamente. A Marta y Sara del Gulich por hacer con su precencia que la estadía en el instituto sea mas agradable aún. A Carlos por facilitar los medios tecnológicos para avanzar día a día en el estudio. A Vanina y Gastón porque con sus aportes en la gestión de mi paso por la maestría. A la FCA por haber facilitar los medios para realizar esta maestría. A mis directores, Estefanía y Ernesto, por el tiempo dedicado y su guía en la realización de este trabajo que hicieron que salga aún mejor. A Ernesto principalmente por su apoyo desde año 2007 que nos conocimos, hasta el día de hoy. Al Dr. Andrés Ravelo del CREAN por abrirme las puertas de su instituto en aquel 2007 y hasta el día de hoy por sus aportes en la tesis de Maestria; por su siempre cordial y respetuoso trato hacia mi. Quiero agradecer especialmente al Dr. Marcelo Scavuzzo (el Marce), por la calidad humana con la que siempre me trató y por haberme mostrado con sus presencia un Ejemplo de profesional y persona y porque siempre estuvo en cada momento de mi formación, con una palabra justa, una recomendación precisa y una gran visión. Ringrazio il mio tutore della Fondazione Edmund Mach, Markus Neteler, per avermi accettato nel loro gruppo. Grazie al suo team: Luca Delucci, Roberto Zorer, Markus Metz e Duccio Roccini, per l'amicizia e la formazione ricevuta. E tutte le altre belle persone che questa esperienza mi ha messo sulla mia strada.

A cada uno de los MAEARTES de mi cohorte y las otras cohortes con las que me crucé, por una cantidad innumerable de momentos vividos dentro y fuera del Gulich, gracias por Su Amistad. La experiencia de vida recolectada en esos dos años quedará por siempre en mis mejores memorias. A los LEONES con quienes compartí la pasión que tenemos los argentinos... el futbol. A mi familia y amigos de la vida, de quienes recibí siempre su apoyo incondicional para seguir adelante.

En África, existe el concepto de Ubuntu, que encierra el profundo sentido
de que somos humanos solo a través de la humanidad de otros;
que si conseguimos cualquier cosa en este mundo, se deberá en igual medida
al trabajo y a los logros de otros...
Nelson Mandela

Resumen

La utilización de la tecnología Web Processing Service (WPS), promueve el inicio de una nueva generación de aplicaciones web en el dominio geoespacial. Los aplicativos SIG y de procesamiento de imágenes, están pasando de ser software ejecutable en computadoras personales a sistemas funcionando en la nube. Estas capacidades de geoprocesamiento, vienen a sumarse a las actuales Infraestructuras de Datos Espaciales (IDE's), nutriéndose de procesos geoespaciales que son invocados como servicios. En el presente trabajo, se especifica un procedimiento general, que permite desarrollar sistemas web para la publicación y ejecución de geoprocesos como servicios, con el fin de brindar un conjunto de herramientas que le permita al usuario generar productos a demanda. La metodología presentada, se basa en el desarrollo de un software de aplicación con dos componentes principales. (a) un servidor de mapas para el acceso y entrega de datos geográficos y un visualizador web de mapas para la interacción con el servidor; (b) un componente de geoprocesamiento que deberá ser vinculado al servidor de mapas y una interfaz de acceso web a los geoprocesos, con procesos predefinidos, para efectuar cálculos espaciales sobre los datos disponibles. Como resultado, obtenemos una infraestructura informática que permite publicar, describir y ejecutar geoprocesos, como servicios web. Se han implementado para tal fin, dos geoprocesos: el primero que opera sobre datos raster (r_nbr) y el segundo que opera sobre datos vectoriales (Buffer), ambos accesibles a través de una aplicación web. También se incorpora un tercer servicio web, r_what, para satisfacer los requerimientos de un proyecto del mundo real aplicado al agro. Obtenemos de este modo, un sistema íntegramente compatible con los estándares actuales promovidos por la Open Geospatial Consortium para almacenar, presentar, publicar e invocar datos y procesos como servicios (software como servicio), mediante el uso de los protocolos WMS, WCS, WFS y WPS, logrando así la Interoperabilidad de información geográfica y geoprocesos. Por último, los geoprocesos implementados quedan accesibles como servicios web, lográndose una solución de cómputo en la nube.

Palabras claves: WPS, Servicios Web, WMS, WCS, WFS, OGC, OSGeo, IDE, ZOO-Proyect, GRASS GIS, raster, vector, WebGIS.

Abstract

The use of the Web Processing Service (WPS) protocol is promoting the startup of a new generation of web applications in the geospatial domain. GIS and image processing applications have moved from being software that runs on personal computers to being systems that work in the cloud. These geoprocessing capabilities have come to expand the current Spatial Data Infrastructures (SDIs), extending their scope by means of geospatial processes which are invoked as services. This paper specifies a general procedure for the development of web systems that will publish and execute geoprocesses as services, with the aim of providing a set of tools which will allow users to generate products on demand. The methodology here presented is based on the development of application software made up of two main components, namely (a) a map server for access to and delivery of geographical data, including a web display of maps for interaction with the server; and (b) a geoprocessing component which will have to be linked to the map server, including a web interface for access to geoprocesses, with preset processes, in order to perform spatial calculations on the available data. As a result, we will obtain an IT infrastructure which will allow us to publish, describe and execute geoprocesses as web services. To this end, two geoprocesses have been implemented: a first one which operates on raster data (r_nbr) and a second one which operates on vector data (Buffer), both of which are accessible via a web application. A third web service, r_what, has also been incorporated in order to meet the requirements of a real-world project applied to agriculture. In this way we will have a system entirely compliant with the current standards furthered by the Open Geospatial Consortium to store, present, publish and invoke data and processes as services (software as service) through the use of WMS, WCS, WFS and WPS protocols, thus achieving an interoperability of geographical information and geoprocesses. The implemented geoprocesses will, lastly, remain accessible as web services, which will mean a cloud computing solution.

Keywords: WPS, Web Service, WMS, WCS, WFS, OGC, OSGeo, SDI, ZOO-Proyect, GRASS GIS, raster, vector, WebGIS

Índice general

Índice de figuras

Índice de tablas

Acrónimos

AEARTE	**A**plicaciones **E**spaciales de **A**lerta y **R**espuesta **T**emprana a **E**mergencias
AJAX	*Asynchronous JavaScript and XML*
API	*Application Programming Interface*
CONAE	**Co**misión **N**acional de **A**ctividades **E**spaciales
CRS	*Coordinate Reference System*
EVI	*Enhanced Vegetation Index*
FaMAF	**Fa**cultad de **A**stronomía, **M**atemática y **F**ísica
GIS	*Geographic Information System*
GML	**Geographic Marked Language**
GPL	**L**icencia **P**ública **G**eneral
GRASS	**G**eographic **R**esources **A**nalysis **S**upport **S**ystem
HTTP	**Hypertext Transfer Protocol**
HTTPS	**Hypertext Transfer Protocol Secure**
IDERA	**Infraestructura de Datos Espaciales de la República Argentina**
IG	**I**nstituto de Altos Estudios Espaciales Mario **G**ulich
JSON	**Javascript Object Notation**
NASA	**N**ational **A**eronautics and **S**pace **A**dministration
NDVI	**N**ormalized **D**ifference **V**egetation **I**ndex
OGC	**O**pen **G**eospatial **C**onsorsium
OLI	**Operational Land Imager**
RESTful	**REpresentational State Transfer**
SIG	**Sistema de Información Geográfica**
TIRS	**Thermal Infrared Sensor**
SRS	*Spatial Reference System*
SWIR	**Short-wave Infrared**
UML	**Unified Modeling Language**
UNC	**U**niversidad **N**acional de **C**órdoba
WFS	**Web Feature Service**
WCS	**Web Coverage Service**
WKT	*WKT Well–Known Text*
WMS	**Web Map Service**
WPS	**Web Processing Service**
XML	*Extensible Markup Language*

Capítulo 1

Introducción General

La Web representa y proporciona una infraestructura tecnológica de amplio desarrollo y alta accesibilidad. La evolución a la Web 2.0 trae consigo la entrega de contenido y el trabajo interactivo, ayudado por la disponibilidad de los navegadores que soportan contenido dinámico. Precisamente la disponibilidad de éstos hace posible que usuarios de todas partes del mundo puedan ejecutar una aplicación o Servicio común accediendo desde cualquier computadora. Un ejemplo de la Web 2.0 son los Servicios Web (*Web Service*) (Keen and Coutinho, 2014). Un Servicio Web es un sistema de software diseñado para soportar interacción interoperable máquina a máquina sobre una red (W3C, 2014). La implementación de ellos proporciona algunas ventajas como por ejemplo acceder, mediante la utilización de protocolos y estándares, a aplicaciones disponibles en internet que hayan sido inclusive desarrolladas en distintos lenguaje de programación y sean independientes de la plataforma de ejecución.
El interés por publicar y compartir datos a través de mapas en la Web, principalmente por parte de instituciones estatales, ha crecido en el último tiempo (IDERA, 2014). El concepto WebGIS nace haciendo referencia a las aplicaciones Web que realizan operaciones tales como consulta, navegación y manipulación entre otras, sobre datos georeferenciados provistos por servidores de mapas del estilo de MapServer, Geoserver, ArcGIS Server. Este avance ha requerido que se definan y adopten estándares y protocolos; rol que viene cumpliendo la OGC (*Open Geospatial Consortium*) desde el año 1994 (Open Geospatial Consortium, 2015). La implementación de los servicios WMS (*Web Map Service*), WFS (*Web Feature Service*) y WCS (*Web Coverage Service*) ampliaron la funcionalidad de los servidores de mapas permitiendo compartir de forma estandarizada los mapas que ellos generan. Es decir, proveen un interfaz de fácil acceso a los datos espaciales como servicios (Data as Service). Un paso mas en la WEB 2.0 es la posibilidad de exponer a los procesos como Servicios (en ingles, *Web Service*). Para tal fin, en el año 2007, la OGC crea un nuevo estándar denominado WPS (*Web Processing Service*) con el objetivo de homogeneizar la forma en que un cliente solicita o ejecuta un proceso como Servicio (OpenGIS WPS, 2007), que luego publicaría una corrección del protocolo en el año 2009 (Corrección WPS, 2009). En particular, el Servicio puede ser un cálculo espacial implementado sobre algún software SIG (Sistema de Información Geográfica) funcionando como *background* de un sistema WebGIS.

1.1. Motivación

La utilización de tecnología como WPS abre un nuevo mundo de posibilidades para programadores y usuarios SIG. Debido a que el estándar WPS es relativamente nuevo no ha sido ampliamente utilizado o adoptado, sin embargo muestra un gran potencial como desarrollo de aplicaciones (Shekhar and Xiong, 2007). La implementación de sistemas Web, complementados con herramientas que permiten a los usuarios ejecutar (usando protocolos como WPS) procesos estandarizados sobre datos raster y vectoriales, tales como algoritmos de análisis, clasificación, cálculos de reproyección, etc, generan una serie de ventajas:

- Fácil acceso a funcionalidades SIG: el usuario accede a través de un navegador a funcionalidades de un SIG de escritorio.
- Publicación de Servicios SIG: muchos usuarios accediendo a los servicios SIG definidos e implementados una sola vez.
- Sin necesidad de instalar nada en la PC personal: toda funcionalidad es gestionada por el navegador a través de los protocolos de Internet.
- Interoperabilidad: servidor-servidor, cliente-servidor
- Integración de diversos software de análisis y procesamiento de imágenes: permiten la generación de algoritmos y funcionalidades implementados en sus propios lenguajes haciéndolos disponibles al usuario SIG a través de un lenguaje en común.
- Generación de productos a demanda: cruzando capas disponibles en diversos servidores de mapas a través de protocolos WMS, WFS y WCS y datos locales.
- Creación de nodos para conformar una red de Servicios SIG.

Combinar distintas herramientas para lograr un sistema completo que integre Servicio de mapas y Servicio de procesamiento es un gran un desafío en la actualidad. Esta combinación permite concentrar las capacidades y cualidades de sistemas ya implementados, testeados y ampliamente utilizados, en aplicaciones accesibles desde internet enfocadas a proyectos que utilizan como unidad de información la imagen satelital y datos vectoriales.
Algunas entidades ya brindan algunos Servicios de geoprocesamiento los cuales son accedidos a través de *plugins* en diversos SIG de escritorio. Tales son los casos de la Fundazione Edmund Mach del Instituto Agrario de San Michele Trento – Italia; el proyecto Zoo (Zoo-Proyect.org); 52 North (http://52north.org/).
El Programa Regional de Empleo de Información Satelital para la Productividad Agrícola ATN/OC-12483-RG es un proyecto del BID conjuntamente con Argentina, Chile, Paraguay y Uruguay y cuyo objetivo es contribuir al aumento de la productividad agrícola mediante el uso de información de origen satelital para la toma de decisiones de las actividades agrícolas y actividades comerciales relacionadas (BID, 2014). Este programa propone la generación de distintos productos como humedad de suelo, cultivos, sequías, incendios y bosques nativos, a

partir de información satelital. Tales productos son el resultado de la ejecución de distintos algoritmos y operaciones entre imágenes satelitales. Mucha de esta información puede permanecer estática sin necesidad de recalcularla y alguna otra cambia a medida que cambian las condiciones ambientales y climáticas, por lo que es de utilidad proveer al usuario la posibilidad de solicitar un nuevo cálculo sin depender del administrador del sistema. Es por estas razones que un sistema como el que se plantea en el presente trabajo, es de suma utilidad y necesidad. Cabe mencionar que si bien la infraestructura informática planteada estará particularmente enfocada a los requerimientos del proyecto BID, la misma puede ser aplicada en diversos proyectos de teledetección, como por ejemplo aquellos relacionados a emergencias ambientales y epidemiología.
La solución de computo en la nube que se propone trae con sigo varios beneficios como la reutilización de geoprocesos, reducción del costo de mantenimiento de los sistemas informáticos (se hace en los servidores del proveedor y nada debe hacerse en la pc del usuario), reducción en el gasto de equipos informáticos (solo se requieren servidores centrales, minimizando el gasto en equipos potentes para el usuario), seguridad en los datos (a cargo del proveedor del servicio), optimización en el uso de recursos de forma automática, entre otros.

1.2. Objetivos

1.2.1. Objetivo general

El principal objetivo de este trabajo fue especificar un procedimiento general que permita desarrollar sistemas web para la publicación y ejecución de geoprocesos como servicios, con el fin de brindar un conjunto de herramientas web que le permita al usuario generar productos a demanda. La metodología presentada se basa en el desarrollo de un software de aplicación con dos componentes principales. (a) un servidor de mapas para el acceso y entrega de datos geográficos y un visualizador Web de mapas para la interacción con el servidor; (b) un componente de geoprocesamiento que deberá ser vinculado al servidor de mapas y una interfaz de acceso web a los geoprocesos, con procesos predefinidos, para efectuar cálculos espaciales sobre los datos disponibles.

1.2.2. Objetivos específicos

- Revisar y aplicar estándares geoespaciales internacionales promovidos por la OGC (Open Geospatial Consortium)
- Estudiar y elegir herramientas gratuitas y de código abierto (Open Source) para la generación de geoprocesos en la web y funcionalidades de los procesos en el servidor
- Incorporar un mecanismo que entregue datos geoespaciales en la web.
- Describir/establecer la forma en que deben interactuar los componentes y tecnologías para la publicación de geoprocesos en la web

- Definir e implementar los algoritmos y geoprocesos requeridos y las funcionalidades como Servicios de procesamiento y generar un cliente para la interacción con los mismos
- Definir e implementar los algoritmos y geoprocesos requeridos en GRASS GIS y las funcionalidades como Servicios de procesamiento.
- Configurar, instalar y poner de forma operativa un servidor con el sistema desarrollado

1.3. Estructura de la tesis

En el capítulo 2 hacemos un análisis de los objetivos específicos que se deben cumplir para la implementación del trabajo propuesto. Además, se presenta una Arquitectura modular de la solución y la relación entre las entidades del mismo, el cual será construido en el capítulo 5.
En el capitulo 3 introduciremos los estándares en el campo de la tecnología geoespacial, promovidos por la OGC y que son necesarios para el desarrollo del trabajo que se plantea en esta tesis.
En el capítulo 4 describiremos que tecnologías hay disponibles para lograr el fin, cuales fueron seleccionadas y porque.
En el capítulo 5 se describe como se implementó el módulo de procesamiento que incluye: servidor WPS, GRASS GIS, modulo cliente y geoprocesos, lo que responde al procedimiento general que se plantea.
En el capítulo 6 se describe la aplicación de la infraestructura planteada en el proyecto PREISPA.
En el capítulo 7 presentamos conclusiones y trabajos futuros.

Capítulo 2

Planteo del problema

2.1. Introducción

Este capítulo describe un análisis de los objetivos específicos que se deben cumplir para la implementación del objetivo general propuesto en esta tesis. Además, se presenta un modelo conceptual de la solución y la relación entre las entidades del mismo, el cual será construido en el capítulo 5.

2.2. Análisis de los objetivos

Para cumplir con el objetivo general propuesto, un conjunto de objetivos específicos fueron planteados. A continuación analizaremos cada uno, con el fin de detallar la potencialidad del sistema desarrollado.

1. Revisar y aplicar estándares geoespaciales internacionales promovidos por la OGC (Open Geospatial Consortium):
   Con el fin de lograr la interoperabilidad, se deberán revisar cuales son los protocolos necesarios para el trabajo con datos raster, vectoriales y procesos.

2. Estudiar y elegir herramientas gratuitas y de código abierto (Open Source) para la generación de geoprocesos en la web y funcionalidades de los procesos en el servidor:
   Este punto es central y se refiere a las herramientas que debieron ser estudiadas, adquiridas y utilizadas para lograr que un geoproceso que se ejecuta habitualmente en una computadora personal, se ejecute en un servidor y que además, el acceso a este proceso este disponible desde Internet de una forma estandar como un servicio.

3. Incorporar un mecanismo que entregue datos geoespaciales en la web:
   Lo primero que necesitaremos es contar con un sistema que provea datos raster y vectoriales, accesibles desde la web, en formatos estándares y que permitan una interfaz de fácil acceso. Un servidor de mapas es la tecnología adecuada para cumplir con este objetivo, ya que presta servicios de datos mediante los protocolos WMS, WFS y WCS.

4. Describir/establecer la forma en que deben interactuar los componentes y tecnologías para la publicación de geoprocesos en la web:
   Se deberá entender como se relacionan los software de procesamientos de imágenes, los software GIS y/o librerías espaciales, con servidores web y servidores de mapas para publicar los procesos en la web.

5. Definir e implementar los algoritmos y geoprocesos requeridos mediante la implementacion de algoritmos para los servicios de procesamiento o geoservicios y generar un cliente para la interacción:
   Se deberán definir un conjunto de procesos a implementar para luego ser publicados como servicios y accesibles para el usuario común. Además, la interacción con tales servicios deberá realizarse mediante un cliente que debe ser implementado.

6. Configurar, instalar y poner de forma operativa un servidor con el sistema desarrollado:
   Un sistema web para visualización, consulta y procesamiento de datos raster y vectoriales deberá estar montado en un servidor y accesible públicamente. Mediante la aplicación real al proyecto PRESIPA se cumplirá con este objetivo.

2.3. Arquitectura del sistema: Modelo conceptual

En esta sección planteamos un modelo (Figura 2.1) que conceptualiza los objetivos específicos; la implementación de dicho modelo será detallada en el capítulo 5. En el capítulo 3 examinamos los estándares de comunicación, herramientas que son centrales en la construcción de sistemas como el que se plantea en este trabajo, pero también en sistemas de publicación de datos espaciales. A partir de este modelo se despreden una serie de requerimientos (principalmente del sistema) que se deberán satisfacer para cumplir con el objetivo.

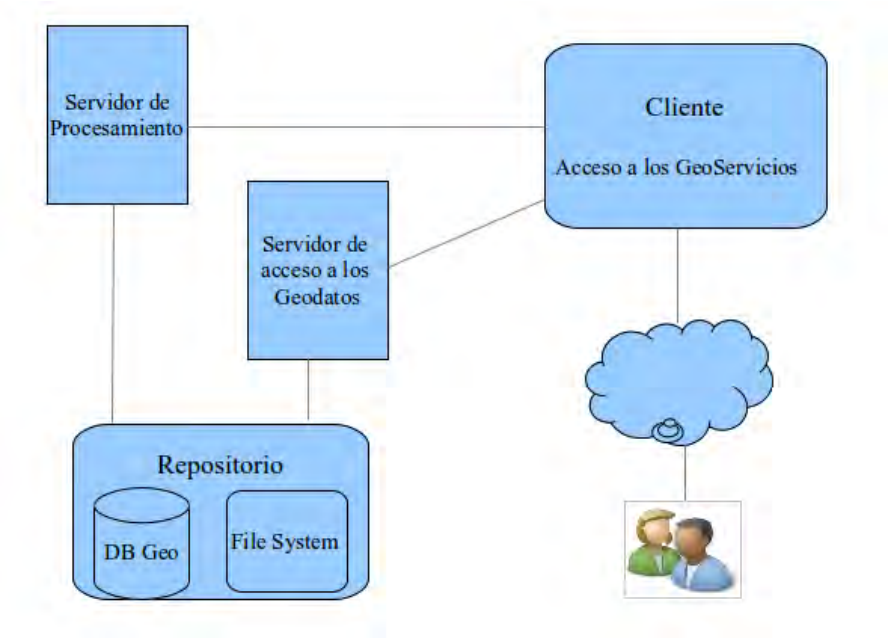


Figura 2.1: Modelo Conceptual de la infraestructura informática planteada

2.4. Requerimientos generales

Un conjunto de requerimientos son necesarios y recomendandos para trabajar con sistemas de datos geográficos y sistemas de procesamiento en la web. Incluyen protocolos estándares internacionales (Open Geospatial Consortium, 2015), formatos estándares para la representación de datos geográficos, software geoespacial certificado (OSGeo, 2015) y estándares para la web (W3C, 2014). En el Capitulo 3 presentamos conceptos geoespaciales y estándares.
En esta sección describimos a modo general, un conjunto de requerimientos necesarios y deseables con lo que debería contar un sistema de geoprocesamiento.

2.4.1. Requerimientos del servidor de acceso a los geodatos

El servidor de geodatos deberá proveer los datos mediante servicios web que cumplan con especificaciones internacionales de la OGC (*Open Geoespatial Consortium*): WMS (*Web Map Service*), WFS (*Web Feature Service*) y WCS (*Web Coverage Service*). Deberá entregar los datos en al menos dos formatos: GeoTiff para datos raster y ESRI Shapefile para datos vectoriales. Además, para este último es deseable proveer conexión con el sistema de base de datos espacial PostGIS. Deberá contar con funcionalidad de reproyección de datos contemplando el conjunto de sistemas de referencias de coordenadas dado por la EPSG[1]. Se

[1] http://www.epsg.org/

requiere como mínimo contar con el sistema de referencia EPSG:4326[2] y EPSG:900913. Es deseable contar con un administrador de capas para facilitar la creación de estilos, generación de grupos de capas, carga/edición/borrado de capas raster y vectoriales. El acceso a los geodatos deberá estar disponible a través de una aplicación web. Es deseable el acceso a los mismos a través de aplicaciones GIS de escritorio. Deberá ser un sistema basado en software libre (*Open Source*).

2.4.2. Requerimientos del servidor de procesamiento

Deberá proveer la generación de algoritmos geoespaciales mediante software y/o librerías de geoprocesamiento certificados por la OSGeo (*Open Geospatial Consortium*). El acceso a los algortimos espaciales deberá estar disponibles a través de la web utilizando el protocolo estandar WPS (*Web Processing Service*) de la OGC. Es deseable proveer acceso a través de aplicaciones GIS de escritorio. El sistema deberá obtener acceso a los datos geoespaciales provistos por el servidor de geodatos, a través de los protocolos WFS y WCS. Se deberán proveer al menos dos algoritmos geoespaciales: uno sobre datos raster y uno sobre datos vectoriales. Los resultados de los geoprocesamientos deberán estar disponibles para el usuario al menos para la descarga. Es deseable entregar los resultados en un visualizador de mapas. Para el caso de procesos de largo tiempo, se deberá proveer un mecanismo que se independice del cliente de modo tal que informe mas tarde el resultado del procesamiento. Deberá ser un sistema basado en software libre (*Open Source*).

2.4.3. Requerimientos de la aplicación web

Se deberá generar una aplicación web para acceder a los datos y algortimos geoespaciales. Se deberán desarrollar componentes en el cliente con el fin de interactuar con los datos provisos por el servidor de procesamiento y el servidor de geodatos mediante WPS, WCS y WFS. Deberá contar con funcionalidades básicas: visualización de mapas; consulta de atributos; herramientas de zoom y desplazamiento; herramientas de medición de área; visualización dinámico de coordenadas geográficas en latitud y longitud y barra de escalas con un conjunto de niveles de escalas predefinido, con el fin de explorar los mapas generados. Deberá proveer un menú de capas sobre las cuales se aplicarán los algortimos implementados, y podrán ser activadas (y desactivadas) para su visualización. Se deberán proveer herramientas visuales para el acceso a los geoprocesos implementados (raster y vectoriales), tales como botones, formularios, selección de *features*, para que el usuario interactue de forma directa con el geoproceso. Para la ejecución de procesos de largo tiempo, se deberá proveer un mecanismo para que el usuario tenga acceso al resultado del procesamiento, una vez que éste finalice. Deberá ser un sistema basado en software libre (*Open Source*).

[2] http://spatialreference.org/ref/epsg/wgs-84/

Capítulo 3

Fundamentos Geoespaciales

3.1. Introducción

En este capítulo introducimos los estándares en el campo de la tecnología geoespacial, promovidos por la OGC y que son necesarios para el desarrollo del trabajo que se plantea en esta tesis.
En este capítulo se cumple el objetivo:

- Revisar y aplicar estándares geoespaciales internacionales promovidos por la OGC (Open Geospatial Consortium).

3.2. Open Geospatial Consortium

El OGC es un consorcio de industrias internacionales de mas de 500 miembros entre empresas, agencias de gobiernos y universidades que participan mediante consenso, en el desarrollo de estándares disponibles publicamente. Los estándares facultan a los desarrolladores para que los datos complejos y servicios geoespaciales sean fácilmente accedidos por todo tipo de aplicación. En este sentido, un estándar es un documento generado y aprobado por los miembros del OGC, que proporciona normas y pautas, dirigido al grado optimo de Interoperabilidad en un contexto dado. Estos documentos son elaborados a partir de requerimientos de la comunidad, requerimientos de los miembros, tendencias del mercado y tendencias tecnológicas. Hasta que los estándares esten disponibles a través de productos de software, estos no resuelven el problema identificado.

3.3. Protocolos para la Interoperabilidad

Entre los estándares que mantiene la OGC existen los denominados estándares *Web service* categorizados como OWS (*OGC Web Service*). Ellos definen protocolos de comunicación para operaciones y datos geoespaciales. Por ejemplo la implementación de estos protocolos son útiles para solicitar información sobre un *feature* específico de un vector; para devolver una imagen de un mapa; para llamar a un servicio que realice el cálculo del NDVI (*Indice de Vegetación de Diferencia Normalizada*). Entre los mas utilizados encontramos WMS, WFS, WCS y WPS (protocolo central en el desarrollo de la presente tesis) los cuales estan construidos para interactuar sobre el protocolo HTTP[1] (*Hypertext Transfer Protocol*). Este último, esta basado en un modelo *peticion/respuesta* entre un cliente y un servidor, lo cual determina un modelo cliente/servidor donde un cliente es una aplicación que utiliza el lenguaje que define el protocolo, mediante las solicitudes (comunmente llamados, verbos) POST y GET, y el servidor es cualquier servidor web. Para cada protocolo antes descripto debe existir un componente *servidor* que lo implemente.
A continuación describimos los protocolos y las operaciones obligatorias y algunas opcionales soportadas, y se mostraran tablas descriptivas con los parámetros de cada operación. Es preciso resaltar que en las tablas decribimos los parámetros de versiones puntuales de los protocolos, a los fines de clarificar el uso de los mismos, pero existen vigentes diversas versiones implementadas y el programador deberá consultar la documentación específica[2] de la versión que desee utilizar.

3.3.1. Web Map Service

El WMS es un estándar definido en OpenGIS WMS (2006) y acompaña su específicación un documento de buenas prácticas OpenGIS (2014). El componente servidor que implementa WMS sirve mapas (un mapa no es el dato en si) generados a partir de los datos geográficos, generalmente en formatos JPEG, PNG, etc. en forma de mosaicos que son aptos para su visualización en las aplicaciones clientes GIS. La específicación del protocolo describe 3 (tres) operaciones para WMS: *getCapabilities, getMap, y getFeatureInfo*. Mientras que las dos primeras son obligatorias, *getFeatureInfo* es opcional en la implementación del servidor WMS. Una extensión denominada SLD (*Styled Layer Descriptor*) se define en OpenGIS WMS-SLD (2002) para ampliar a WMS agregando estilo a *features* y coberturas. Para lograr esto, se definen operaciones *DescribeLayer y GetLegendGraphic* entre otras, pero estas últimas resultan de especial interés.

[1] protocolo usado para cada transacción sobre la World Wide Web
[2] http://www.opengeospatial.org/

3.3.1.1. Petición GetCapabilities

El propósito de esta petición, es obtener el metadato del servicio el cual es una descripción (legible por el humano y por la computadora) de las operaciones, servicios y datos que ofrece el servidor WMS (Tabla 3.1).

Parámetros	Oblig.(M)/Opcionales(O)	Descripción
VERSION=version	O	Request version (Versión del protocolo o servicio)
SERVICE=WMS	M	Service type (Tipo de servicio)
REQUEST=GetCapabilities	M	Request name (Nombre de la petición)
FORMAT=MIME_type	O	Output format of service metadata (Formato de salida del metadato del servicio)

Tabla 3.1: Los parámetros de la petición GetCapabilities (OpenGIS WMS, 2006).

3.3.1.2. Petición GetMap

El propósito de esta petición, es obtener un mapa. Una vez realizada la solicitud, el servidor retornará un mapa o emitirá un reporte de excepción (cuyo código es definido por el estándar) en el caso que no lo encuentre o que el mismo sea innaccesible (Tabla 3.2).

Parámetros	Descripción
layers	List (comma–separated) of map layers (lista de capas)
styles	List (comma–separated) of styles (lista de estilos)
SRS	Spatial reference system (sistema de referencia espacial)
BBOX	Bounding box coordinates (coordenadas del rectángulo geográfico)
width	Pixel width of the map image (ancho en pixeles de la imágen del mapa)
height	Pixel height of the map image (alto en pixeles de la imágen del mapa)
format	Output format of the map image (formato de salida de la imágen del mapa)

Tabla 3.2: Web Map Service – Parámetros de la petición GetMap (OpenGIS WMS, 2006).

3.3.1.3. Petición GetFeatureInfo

Esta petición es opcional. Es soportada por capas para las cuales el atributo queryable=1 ha sido definido o heredado.(por ejemplo las capas base tienen este atributo queryable=0). La operación *getFeatureInfo* esta diseñada para proveer a los clientes WMS la capacidad de obtener información acerca de los *features* en la imagen del mapa que fue previamente retornado en un solcitud *getMap*. El uso típico, consiste en que el cliente elije una coordenada sobre el mapa (por ejemplo, haciendo click sobre él) sobre la cual obtener información (tabla 3.3).

Parámetros	Oblig.(M)/Opcional(O)	Descripción
VERSION=1.3.0	M	Request version (Versión del protocolo o servicio)
REQUEST=GetFeatureInfo	M	Request name (nombre de la petición)
QUERY_LAYERS=layer_list	M	Comma-separated list of one or more layers to be queried (lista de capas a consultar)
INFO_FORMAT=output_format	M	Return format of feature information (MIME type) (formato de salida de la información solicitada)
FEATURE_COUNT=number	O	Number of features about which to return information (default=1) (máximo número de feature a retornar)
I=pixel_column	M	i coordinate in pixels of feature in Map CS (coordenada I del punto a consultar sobre el mapa,en pixeles)
J=pixel_row	M	j coordinate in pixels of feature in Map CS (coordenada J del punto a consultar sobre el mapa,en pixeles
EXCEPTIONS=exception_format	O	The format in which exceptions are to be reported by the WMS (default= XML) (formato en el cual se reportan las excepciones)

Tabla 3.3: Los parámetros de la petición GetFeatureInfo OpenGIS WFS (2005)

3.3.1.4. Petición DescribeLayer

Esta petición se utiliza en el caso que se deseen aplicar estilos definidos por el usuario, para lo cual es necesario conocer al menos el tipo: *feature* o cobertura. Si este tipo de funcionalidades se requieren implementar en un cliente GIS, es necesario que el WMS soporte esta operación (Tabla 3.4). Esta operación se define en OpenGIS WMS-SLD (2002).

3.3.2. Web Feature Service

El WFS es un estándar definido en OpenGIS WFS (2005) para el intercambio de información geográfica en la web sobre HTTP. Mientras que WMS genera y transmite imágenes como mapas, WFS permite devolver y actualizar datos codificados en GML (*Geography Markup Language)*. GML es un lenguaje basado en XML, es un estándar y se describe en la sección *Formatos para la Interoperabilidad*. En su versión simple, implementa las operaciones *GetCapabilities y GetFeature*, y en su versión WFT-Transacional, se agregan las peticiones *LockFeature y Transaction*. En otras palabras, WFS permite que los *feature* sean consultados, actualizados, creados o borrados.

Nombre	Definición	Tipos de datos y valores	Multiplicidad
service	Service type identifier (Identificador del tipo de servicio)	Character String type, not empty. Value is OWS type abbreviation (“WMS”) (caracter de tipo string, no vacio, identificando el tipo de servicio)	One (obligatorio)
request	Operation name (nombre de la petición)	Character String type, not empty (“DescribeLayer”) (caracter de tipo string, no vacio.)	One (obligatorio)
version	Specification version for operation (WMS) (version del servicio WMS)	Character String type, not empty (caracter de tipo string, no vacio)	One (obligatorio)
layers	names of layers description of requested layers (nombres de las capas para la descripción solicitada)	Character String type,not empty. Multiple layer names separated by ’,’ (caracteres de tipo string, no vacios, separados por coma)	Multiple (uno es obligatorio)
sld_version	Specification version for SLD-specification (version del servicio SLD)	Character String type, not empty (1.1.0) (caracter de tipo string, no vacio.)	One (obligatorio)

Tabla 3.4: Los parámetros de la petición DescribeLayer (OpenGIS WMS-SLD, 2002).

3.3.2.1. Petición GetCapabilities

Esta petición genera un documento de metadatos describiendo el servicio WFS provisto por el servidor así como las operaciones, parámetros válidos y el listado de capas vectoriales.

3.3.2.2. Petición GetFeature

Esta operación permite consultar o devolver un *feature* o un conjunto de ellos desde la fuente de datos incluyendo la geometría y valores de atributos. Es decir, es una operación que se utiliza para transferir datos vectoriales en la red en formatos como GML, GeoJSON y algunos otros (Tabla 3.5), pero también para consultar los atributos de una geometría. Es una operación mas flexible que *getFeature* de WMS ampliando el tipo de entradas y salidas.

Parámetros	Descripción
service	Service type identifier Character String type, not empty. Value is OWS type abbreviation (“WFS”)
version	Specification version for operation (WFS) String type, not empty(”1.1.0”)
request	Operation name Character String type, not empty (GetFeatures)
typeName	Layer Name
srsName	Layer Projection
outputFormat	Output format of the feature

Tabla 3.5: Web Feature Service 1.1.0 – Parámetros de la petición GetFeatures (OpenGIS WFS, 2005).

3.3.2.3. Petición DescribeFeatureType

Esta petición devuelve la descripción (tabla de atributos), de los tipos de *feature* soportados por el servidor WFS.

3.3.3. Web Coverage Service

El WCS es un estándar definido en OpenGIS WCS (2008) creado por la OGC y cuyo objetivo es proveer información geoespacial como *cobertura* para su transferencia. Ejemplos de coberturas son: datos climáticos, modelos de elevación digital, coberturas de suelo, índices históricos, frecuencia de ocurrencia de eventos, en general, información geoespacial de fenomenos variando en el espacio-tiempo. El protocolo WCS es el homólogo al protocolo WFS simple, en el sentido que trabaja sobre datos raster y como contraparte del WFS-T, existe el protocolo WCPS (*Web Coverage Processing Service*) definido en OpenGIS WCPS (2009).
WCS se distingue también del WMS, debido a que mientras el último retorna un mapa, el WCS retorna el dato raster que puede tener como fin la descarga o bien ser utilizado como entrada de modelos o algoritmos de procesamiento. Una capacidad importante con la que cuenta el protocolo, es que permite retornar un *subset* a partir del dato completo. Define 3 (tres) operaciones básicas *getCapabilities, describeCoverage y getCoverage*.

3.3.3.1. Petición GetCapabilities

Su comportamiento es similar a la operación *getCapabilities* del protocolo WMS, excepto que solo devuelve la lista de coberturas.

3.3.3.2. Petición DescribeCoverage

El objetivo de esta operación es proporcionar información adicional acerca de una o mas Coberturas que estan siendo consultadas por un cliente. A esta operación la precede un *getCapabilities*. La información que devuelve es variada incluyendo crs (*coordinate reference system*), el metadata, el dominio de definición espacial del raster, formatos disponibles (por ejemplo, GeoTiff). La respuesta viene contenida en un documento XML.

3.3.3.3. Petición GetCoverage

El objetivo de esta operación es obtener desde el servidor WCS, la Cobertura solicitada (precede a esta operación una del tipo *getCapabilities* de WMS o WCS). Es la operación mas

importante ya que permite transferirle al usuario o a la aplicación, el dato raster y es útil para implementar por ejemplo: funcionalidades de Descarga de Capas Raster o bien servir de entradas para servicios Web de procesamiento de información espacial. Esta operación es la homóloga a la operación *getFeature* definida sobre WFS. Debido a que el objetivo final es la trasnferencia del dato raster, se deben específicar en la solicitud, parámetros obligatorios como el *BoundingBox* y el sistema de proyección *GridBaseCRS*. En la tabla 3.6 se muestran algunos parámetros de la llamada y en OpenGIS WCS (2008) se encuentra la lista completa.

Parámetros	Opcionalidad	Definición y formato
service=WCS	Mandatory	Service name, shall be "WCS" (nombre del servicio)
version=1.1.1	Mandatory	Request protocol version (vesión del protocolo solicitado)
request=GetCoverage	Mandatory	Name of the operation, shall be "GetCoverage" nombre de la petición
identifier=identifier	Mandatory	Unique identifier of an available coverage (nombre de la capa raster disponible)
BoundingBox= 47,-71,-51,66, urn:ogc:def:crs :OGC:1.3:CRS84	Opcional, include when spatial subset desired (define un rectangulo espacial) Request a coverage subset defined by the specified bounding box in the referenced coordinate reference system	
GridBaseCRS= EPSG:4326	Optional	Reference to baseCRS of desired output GridCRS, a URN (define el sistema de referencia para la salida)
format=image/geotiff	Mandatory	Requested output format of Coverage. Shall be one of those listed in the description of the selected coverage (define formato del raster de salida)
store=true	Optional	Specifies whether response coverage should be stored, remotely from client at network URL, a boolean value (define si el raster de salida debe o no almacenarse en el servidor)

Tabla 3.6: Web Coverage Service 1.1.1 – Parámetros de la petición GetCoverage

3.3.4. Web Processing Service

El protocolo WPS originalmente fue llamado *Geoprocessing Service* pero el nombre cambió a *Web Processing Service* para evitar confusión con el acronimo GPS (*Global Positioning System*). La versión final del protocolo es WPS 1.0 del año 2007 y se publicó una corrección en

el año 2009 (Corrección WPS, 2009).
Es un estándar definido en OpenGIS WPS (2007) por la OGC y su objetivo es proveer una interfaz estándar para facilitar la publicación de procesos (en particular, geoprocesos), descubrimiento y conectividad con un cliente. El término ”publicación”hace referencia con hacer que la información este disponible mediante los metadatos, para humanos y máquinas, la cual permite descubrir y usar el servicio.
Un WPS se puede configurar para ofrecer funcionalidades GIS (cálculo de buffer, union, intersección) a través de la red, como así también de algoritmos pre-programados (operaciones entre capas raster) o modelos computacionales (calculo de índices, modelos de inundación, generación de datos climáticos) que trabajen sobre datos georeferenciados.
La específicación de la interfaz provee mecanismos para identificar los datos gereferenciados requeridos, iniciar el cálculo y manejar las salidas asi el cliente puede acceder al mismo.
La específicación WPS esta diseñada para permitirle a un proveedor de servicios, exponer los procesos en la web, tales como intersectar polígonos, calcular buffer, de una manera que posibilita al cliente colocar los datos de entrada y ejecutar el proceso sin tener conocimientos de como está implementando ni como está definida la API de acceso al proceso subyacente.
La interfaz de WPS estándariza la forma en que los procesos y sus entradas/salidas son descriptos, cómo un cliente puede solicitar la ejecución de un procesos y cómo es manejada la salida desde un proceso.
Los datos de salida puede ser entregados a través de la red o quedar disponibles en el servidor. Estos datos pueden incluir imágenes en formato GeoTiff o en formato intercabiable estándar tal como GML (*Geography Markup Language*), entre otros. La entradas pueden ser peticiones a servicios OGC como ser, un dato de entrada para una operacion de intersección podría ser un polígono entregado como respuesta a una solcitud WFS, en tal caso el dato de entrada al WPS es la cadena de consulta WFS. Lo mismo sucede con coberturas, para el cálculo de un índice, los datos de entrada podrían provenir como respuestas a consultas WCS y los datos de entrada del WPS serían las cadenas de consultas WCS.
La interfaz WPS 1.0 específica tres operaciones que pueden ser solicitadas por un cliente y realizadas por el servidor WPS, todas obligatorias: *getCapabilities, describeProcess y execute*.

3.3.4.1. Petición GetCapabilites

Esta operación permite solicitar y recibir el documento de metadato de los servicios (capacidades) que describe las capacidades específicas de la implementación del servidor. En el metadato encontramos nombre y descripción general de cada proceso ofrecido por el instancia WPS. Esta operación también soporta negociación de la específicación de la versión que esta siendo usada para la interacción cliente-servicio.

3.3.4.2. Petición DescribeProcess

Esta operación permite a un cliente solicitar y recibir información detallada acerca del proceso que corre sobre la instancia del servidor, incluyendo las entradas requeridas, formatos

disponibles y las salidas que pueden ser producidas. La utilidad principal de esta petición se ve reflejada en las interfaces de descripción del servicio desarrolladas por el cliente, que son el producto de la interpretación del archivo recibido en la consulta.

3.3.4.3. Petición Execute

Esta operación permite a una aplicación cliente, ejecutar un proceso especifico provisto por el WPS, usando valores provistos como parámetros de entrada y devolviendo las salidas producidas.
Estas operaciones tienen muchas similitudes con otros servicio OWS, incluyendo los WMS, WFS y WCS. En la Figura 3.1 observamos el diagrama UML de la interfaz de WPS.

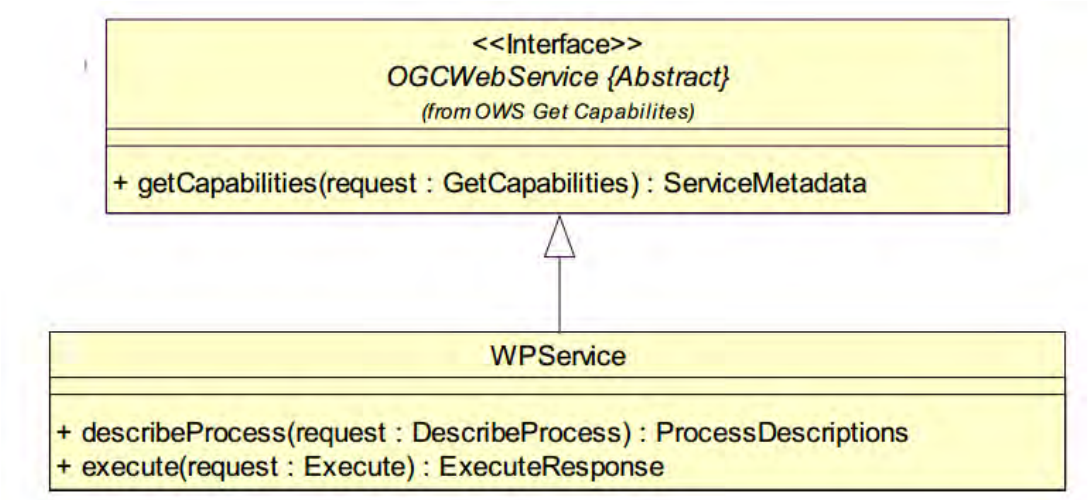


Figura 3.1: Diagrama UML de la interfaz de WPS (OpenGIS WPS, 2007)

3.3.5. Otros aspectos del WPS

WPS puede verse como un modelo abstracto de un servicio Web, para el cual deben desarrollarse perfiles. Si bien, WPS permite desarrollar servicios para reutilizar significantes cantidades de código en el desarrollo de interfaces web, facilitando al mismo tiempo la comprensión entre los desarrolladores de aplicaciones web, el OGC propone el uso de perfiles estándarizados para conseguir una interoperabilidad completa. El empleo de un perfil permite la optimización del comportamiento de interoperabilidad de la interface del usuario cliente, así como del paradigma publicar/encontrar/vincular. Para alcanzar una alta interoperabilidad, cada proceso estará específicado en un perfil de aplicación de esa específicación. Respecto a los mecanismos para encontrar y enlazar servicios, WPS sigue el modelo OGC de WMS y WFS, de manera que define la operación *getCapabilities* que puede ser solicitada mediante el protocolo HTTP y SOAP con las operaciones GET (pares valor-clave), POST (mediante un XML) y SOAP/WSDL (Michaelis and Ames, 2008). WPS describe la interface del servicio la

cual define:

- Codificar la petición para la ejecución del proceso
- Codificar la respuesta para la ejecución del proceso
- Embeber los datos y metadatos en las entradas y salidas de la ejecución del proceso
- Referenciar los datos de entrada y salida accesibles vía Web
- Devolver información del estado del proceso
- Devolver errores del proceso
- Obtener la respuesta de las salidas del proceso

Para ello, WPS propone una serie de operaciones mediante las que se envían y se recibe la información y los datos de los procesos. Los tipos de datos de entrada y salida que se utilizan pueden ser de tres tipos diferentes:

- LiteralData: Cadenas de caracteres, numérico entero y numérico doble
- ComplexValue y ComplexValueRefence: El primero corresponde a archivos raster, vectores y otros archivos de datos, como mapas en varios formatos. En el caso de ComplexValueReference se trata de la URL donde se encuentran los datos.
- BoundingBox: Son pares de coordenadas

WPS permite para la obtención de datos de entrada dos métodos diferentes. Mediante la codificación de datos en la petición *Execute*, actuando como un servicio stand-alone, o haciendo referencia a los datos desde un recurso accesible vía Web, actuando como un servicio *middleware* que obtiene datos desde un recurso externo con el fin de ejecutar un proceso en la implementación local. Además, la ejecución de los procesos puede realizarse de manera síncrona y asíncrona. Los cálculos geoespaciales pueden durar mucho tiempo, en termino de horas, días e incluso semanas. En estos casos, es posible realizar el seguimiento del progreso del proceso chequeando el estado del mismo. Otro aspecto importante es el encadenamiento de servicios con WPS. Generalmente, un proceso WPS es una función atómica que realiza un cálculo geoespacial específico. El encadenamiento de procesos WPS facilita la creación de *workflows* repetitivos. A partir de la versión 1.0.0 de WPS, el estándar es compatible con WSDL y SOAP (Simple Object Access Protocol). WSDL puede utilizarse para un proceso individual WPS, así como para una instancia WPS entera que puede incluir varios procesos.

3.4. Seguridad sobre los protocolos

Los *Web Service* antes descriptos, son ofrecidos comúnmente usando el protocolo HTTP. El proveedor del servicio podría ofrecer el acceso al *Web Service* mediante el protocolo HTTP seguro (HTTPS)[3], realizando las configuraciones adecuadas en el servidor y activando el uso de HTTPS.
Para el caso del protocolo WFS, en OpenGIS WFS (2005) se especifica que se podría ofrecer el servicio WFS sobre HTTPS sin afectar la descripción de la solicitud ni la respuesta, pero acciones adicionales se requieren en el cliente y en el servicio con el objetivo de iniciar una comunicación segura. Para los protocolos WMS, WCS no se encontró documentación oficial que especifique concretamente el uso de ellos sobre un canal seguro. Para el caso de WPS, SOAP puede utilizarse para empaquetar las peticiones y las respuestas WPS permitiendo incluir certificado de seguridad al invocar los servicios WPS (OpenGIS WPS, 2007).

3.5. Arquitectura ampliada

Los protocolos anteriormente descriptos son la clave para la interacción entre los distintos componentes del modelo conceptual dado en la Figura 2.1.
La implementación de los protocolos WMS, WFS, WCS y WPS, hacen posible que los subsistemas que integran la solución completa (geodatos + geoprocesos), se comuniquen entre si logrando Interoperabilidad, lo que nos lleva a ampliar el modelo conceptual antes definido, como podemos observar en la Figura 3.2.

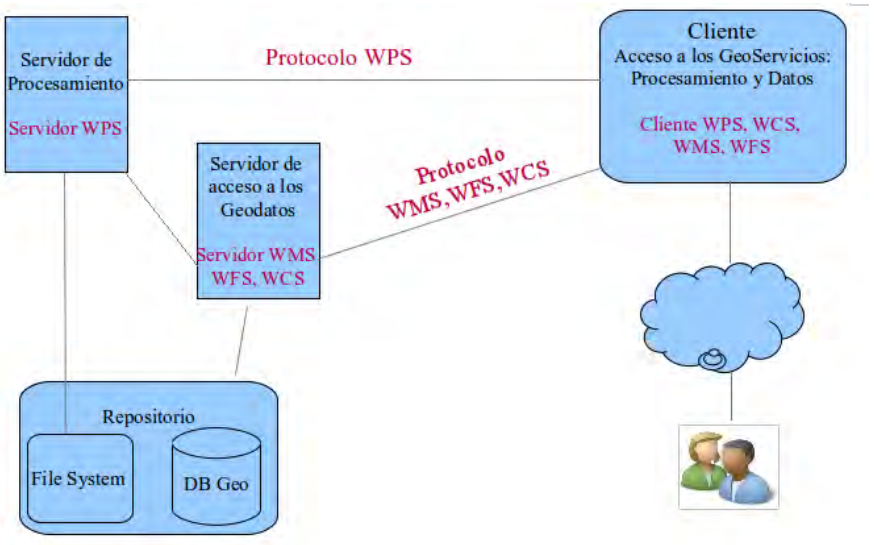


Figura 3.2: Modelo Conceptual Ampliado de la infraestructura informática planteada

[3]HTTPS es HTTP sobre un canal de comunicación seguro, el cual permite que la información encriptada sea transferida entre máquinas sobre la World Wide Web (WWW)

3.6. Formatos para la Interoperabilidad

Se describen los formatos geoespaciales KML and GeoJSON and GML and WKT, los cuales son parte de la estándarizacion que logra la OGC para la interoperabilidad de los sistemas de información geografica. Mediante estos formatos se codifica la información geoespacial que se transfiere en red.

3.6.1. Geography Markup Language

Es una codificación para *features* geográficos basada en XML (OGC-GML, 2015) y es la codificación por defecto usada por los servicios OWS tales como WFS (este protocolo, lee y escribe datos GML). Los documentos GML pueden ser validados por esquemas provistos por la OGC (OpenGIS, 2012).

3.6.2. GeoJSON

Es un estándar abierto (Butler et al., 2008) para el intercambio de información espacial basado en JSON (*JavaScript Object Notation*) y adoptado e implementado por servidores de mapas (aunque no es un estándar de la OGC). Un objeto GeoJSON puede representar un *feature* o una colección de *features*. Soporta los siguientes tipos: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon y GeometryCollection. Un *feature* en GeoJSON contiene un objeto de geometría y propiedades adicionales y una colección de *features* representa una lista de*features*.

3.6.3. Well Known Text

El WKT, es un estándar de la OGC que provee una forma compacta de representar objetos geograficos facilmente legibles por la máquina y el por el usuario. Permite representar una variedad de geoometrías: POINT, LINESTRING, MULTILINESTRING, POLYGON (simple multipoligono), GEOMETRYCOLLECTION (puntos, lineas, polígonos), POINT EMPTY y MULTIPOLYGON EMPTY (OpenGIS WKT, 2010). También es utilizado para describir la definiciones de los elementos de un sistema de referencia de coordenadas (CRS) (OpenGIS WKT-CRS, 2013).

3.6.4. Keyhole Markup Language

El KML es un formato de archivo utilizado para mostrar datos geográficos en un navegador de mapas. Fue creado por Google (OGC and Google, 2015) y posteriormente presentado a la OGC para su revisión e inclusión como un estándar.
KML es un lenguaje enfocado en la visualización, incluyendo anotación de mapas e imágenes. La visualización geográfica incluye no solo la representación de datos grográficos sobre el globo, sino también el control de la navegación del usuario en el sentido de, donde ir y donde mirar. Desde este punto de vista, KML complementa a los estándares existentes como GML, WFS y WMS. Pero en las versiones actuales, incluye características del GML como la representación de puntos, lineas y polígonos (OpenGIS KML, 2008).

Capítulo 4

Tecnologías de base

En este capítulo se describen las tecnologías existentes basadas en software libre, que implementan los estándares y componentes necesarios para lograr el objetivo, además de documentarlas, explicamos las elecciónes tomadas.
En este capítulo se cumple el objetivo:

- Estudiar y elegir herramientas gratuitas y de código abierto (Open Source) para la generación de geoprocesos en la web y funcionalidades de los procesos en el servidor.

4.1. Servicios Web

Una defínción estándar de los *Web Service* esta dada por la W3C (W3C Working Group, 2004): *Un Web Service es una sistema de software diseñado para lograr la interacción interoperable máquina a máquina sobre la red. Tiene una interfaz descripta en un formato procesable por una máquina (específicamente, WSDL (Web Service Description Language)). Los demás sistemas interactúan con los Web Service de la forma establecida por su descripción usando mensajes SOAP, por lo general usan HTTP con serialización mediante XML conjuntamente con otros estándares web.*
Los *Web Service* describen un paradigma el cual permite integrar distintos componentes de software y aplicaciones, creadas en diferentes lenguajes de programación e interfaces completamente distintas, mediante estándares que funcionen en Internet o bien en una Intranet. Los *Web Service* son accedidos por software cliente especializado, lo cual se hace posible debido a que ésta tecnología consiste en componentes que se auto-describen generando así la información suficiente para que un componente de software cliente determine la funcionalidad que provee y/o los datos diposibles para el acceso. En la práctica, los *Web Service* son implementados para usar XML ya sea para la descripción de la interfaz como así también para la transferencia de datos. En otras palabras, se utiliza WSDL (que consiste de un archivo en

formato XML) para describir el servicio e indicar como acceder a él y a sus métodos (Suda, B., 2003). El objetivo fundamental de los *Web Service* es generar un alto grado de Interoperabilidad entre distintos componentes de software.

4.2. Servicio de Procesamiento Web: WPS

La utilización de un Servicio de Procesamiento Web (en sus siglas en inglés, WPS) es útil si se desea que los servicios de geoprocesamiento esten disponibles en forma abierta, reconocida y accesible por distintas plataformas y clientes. Existen diversas implementaciones de WPS Server que hacen posible la publicación de servicios WPS en la web y la interacción con componentes clientes WPS. A continuación describiermos las opciones y las elecciónes realizadas.

4.2.1. Implementación del estándar WPS: pyWPS

pyWPS (*Python Web Processing Service*) (Cepicky, 2015) es una implementación en Python, del estándar WPS 1.0.0 (hasta el momento de la escritura de este trabajo) de la OGC (OGC-pyWPS, 2015) y es un proyecto incubado por la OSGeo. Sus inicios fueron en el 2006 como un proyecto financiado por una Fundación Alemana de Medio Ambiente[1] y desde el 2009 principalmente esponsoreada por una empresa de aplicación GIS [2]. La última versión es la pyWPS 3.0.0. Esta plataforma consiste en un entorno que implementa el servidor WPS y permite ejecutar procesos o calculos espaciales implementados en Python. Además, provee soporte nativo para utilizar GRASS GIS y sus funcionalidades como servicios (Cepicky and Becchi, 2006). Basado en tecnología *open source*, requiere una serie de componentes que permiten construir un entorno completo: servidor web Apache, Python, Python-XML (obligatorios), GRASS GIS, proj4, GDAL/OGR, Mapserver y R (recomendados).

Algunas cuestiones que se consideraron

- Escasos proyectos implementan esta tecnología
- La documentación carece de ejemplos, aunque es muy clara
- Debido a que es una herramienta basada en Python, es muy alta la probabilidad de crecimiento debido a la facilidad para programar en este lenguaje
- Es *Open Source*
- Solo permite invocar geoprocesos escritos en Python

[1] https://www.dbu.de
[2] http://www.bnhelp.cz/

4.2.2. Implementación del estándar WPS: Geoserver-WPS

Es la implementación WPS 1.0.0 que extiende al servidor de mapas Geoserver, para proveer análisis espacial. Debido a que está disponible como una extensión, no es parte del núcleo de Geoserver. La principal ventaja de Geoserver WPS sobre una instalación independiente, es la integración con los servicios de Geoserver y el catálogo de datos: el dato para realizar un cálculo espacial está disponible directamente en el servidor de mapas. En este sentido, WPS actua como una herramienta geoespacial completa capaz de leer y escribir datos hacia y desde Geoserver (Geoserver, 2015).
Implementa las operaciónes *getCapabilites, describeCoverage, execute and dismiss*. Ésta última sirve para detener o cancelar la ejecución de un proceso iniciado por el cliente. También, a la par de esta operación implementa una pseudo-operación WPS, *getExecutionStatus*, que permite llevar hacer un seguimiento del estado de avance del proceso iniciado.

Administración del servicio

Geoserver ofrece una interfaz de administración de procesos (cliente WPS) accesibles mediante WPS, permitiendo configurar el metadato del servicio (Figura 4.1), las opciones de administración y ejecución de los recursos (Figura 4.2) y hacer un seguimiento del estado del servicio (Figura 4.3).

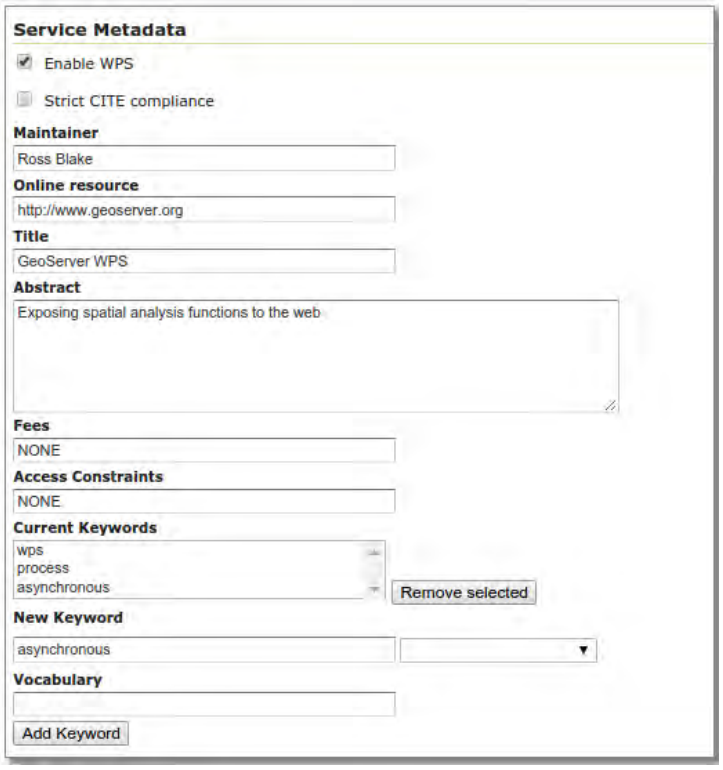


Figura 4.1: Metadato del Servicio WPS (WPS, 2015)

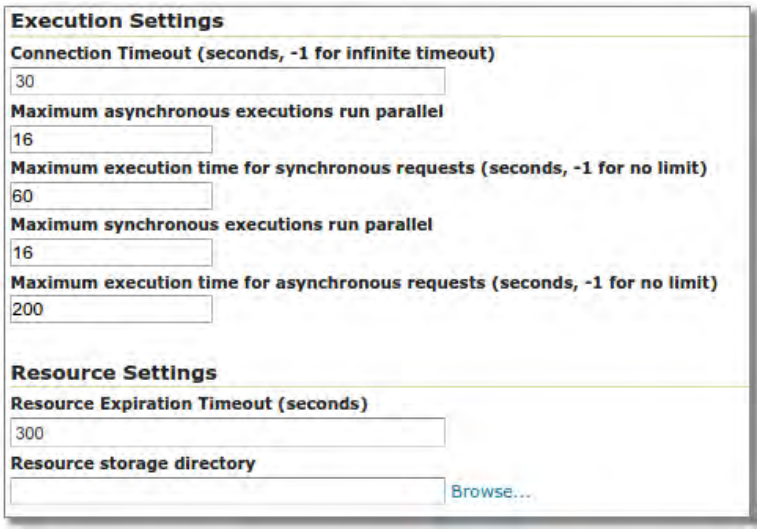


Figura 4.2: Administración de los recursos de ejecución (WPS, 2015)

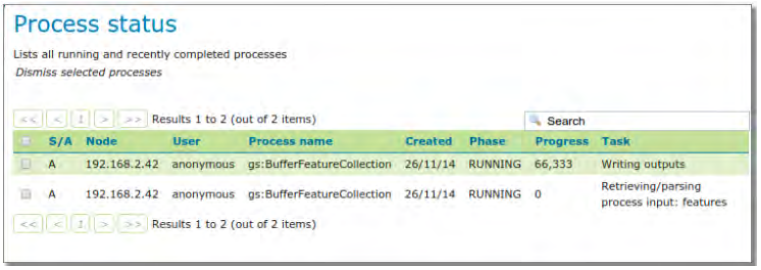


Figura 4.3: Estado de los procesos (WPS, 2015)

Geoprocesos

Geoserver implementa procesos espaciales de dos tipos:

- JTS Topology Suite processes:
  JTS Topology Suite es una librería JAVA de funciones para procesar geometría en dos dimensiones. JTS es conforme a la especificación *Simple Feature* para SQL publicado por la OGC, similar a PostGIS. También incluye funciones comúnes como área, buffer, intersección y simplificar.

- GeoServer-specific processes:
  Geoserver WPS incluye una serie de procesos creados para usar con Geoserver, algunos son reproyección y límites. Estos usan conexiones internas con WCS y WFS para leer y escribir datos.

Algunas cuestiones que se consideraron

- No encontramos registros de cuanto se usa el módulo de procesamiento de Geoserver accedido como WPS.
- Es una herramienta actualizada ya que tiene un mantenimiento frecuente de toda la plataforma Geoserver
- La documentación no es completa en cuanto a ejemplos, tampoco se encontraron foros que ayuden en caso de problemas, pero si existe la lista de usuarios de Geoserver que responde a preguntas sobre toda la plataforma Geoserver.
- Es una herramienta que implementa procesos en JAVA y se pueden ampliar con el lenguaje Jython, el cual provee una opción mas accesible debido a que hoy en día el desarrollo de librerías espaciales con Python/Jyhton es muy común.
- No es sencillo generar y agregar procesos propios a Geoserver WPS.
- Es *Open Source*

4.2.3. Implementación del estándar WPS: 52north

El nombre 52°North hace referencia al grado de latitud que intersecta con el lugar de fundación de la organización, la ciudad de Münster, Alemania. El proyecto 52ºNorth esta compuesto de una red internacional abierta de *partners* de investigadores, insdustrias y administración pública. Su principal propósito es fomentar la innovación en el campo de la GeoInformática. Todo el software desarrollado es publicado bajo la licencia GNU GPL V.2 (52north, 2015).

Entre sus características principales encontramos:

- Esta implementado en JAVA
- Implementa el protocolo WPS 1.0.0, sorportando todas las características
- Integra JTS, geotools, servlets, derby
- Soporta registro de actividad de servicios y almacena resultados de la ejecución
- Provee un cliente WPS
- Invocación de procesos síncronos y asíncronos, soporta datos crudos y llamadas HTTP-GET y HTTP-POST
- Soporta formatos GeoTiff, ArcGRID, GML2, GML3, Shapefile, KML y WKT.

- Extensiones existentes: WPS4R (backend de R), GRASS GIS, más de 200 procesos SEXTANTE y conector con ArcGIS Server
- Los resultados vectoriales y raster pueden ser almacenados como WFS y WCS.

Algunas cuestiones que se consideraron

- No se encontraron trabajos significativos usando esta tecnología
- La documentación no es muy clara y es escasa
- Muestra versatilidad en cuanto a las opciones de conexión con diversos software GIS y de procesamiento
- Permite agregar procesos en Python pero teniendo como requisito ArcGIS server instalado, siendo este último software privativo
- Es una librería *Open Source* basado en la licencia GNU GPL V.2

4.2.4. Implementación del estándar WPS: ZOO-Proyect

ZOO es un proyecto *Open Source* que implementa el estándar WPS 1.0.0 (y WPS 2.0.0 pero en su versión beta hasta el momento de escribir este trabajo) ha sido liberado bajo una licencia al estilo de MIT/X-11 (A.1) la cual es compatible con GPL en el sentido que esta última permite la combinación y redistribución con el software que usa licencia MIT[3]. Provee un *framework* de desarrollo amigable compatible con OGC WPS para crear y encadenar procesos. Su primer versión fue desarrollada en el año 2010, llevando un ritmo de crecimiento de una versión por año, en Julio pasado se liberó la versión ZOO-Proyect 1.5 la cual incluye entre sus características mas importantes, una primera implementación (beta) de WPS 2.0 con la operación *dismiss*. El proyecto desde su nacimiento ha sido incubado por la OSGeo.
ZOO-Proyect esta diseñado para comúnicarse con librerías espaciales promovidos por la OSGeo y logra una comúnicación estandar. Entre los proyectos privados que actualmente estan haciendo uso de ZOO se encuentran: GeoLabs, Cleolys, Cartogenic y Cartworks.
Implementa las operaciones *getCapabilities, describeProcess y execute* en la versión WPS 1.0 y agrega las operaciónes *getStatus, getResult y dismiss* en la versión WPS 2.0. EL objetivo principal de ZOO, es lograr que librerías y software geoespacial, corran sobre un servidor como servicios WPS. Entre los software y librerías que soporta se encuentran:

- GDAL (*Geospatial Data Abstraction Library*). Los servicios ZOO están disponibles para *gdalinfo, gdal_grid, gdal_translate, gdal_extractProfile, ogrinfo, ogr2ogr, gdal_warp* y algunos mas. Es posible generar algoritmos usando pyGDAL.

[3] http://www.gnu.org/licenses/license-list.html#Expat

- Mapserver. Zoo-kernel tiene soporte opcional para Mapserver y es capaz de generar archivos *mapfile* al vuelo para las salidas generadas por el modulo de servicios de ZOO a través de los protocolos WFS, WMS y WCS.
- GRASS 7.0+. El *kernel* es capaz de correr la mayoría de los módulos de vectores y raster de GRASS GIS 7.0+ usando un componente de software que actúa como puente entre él y ZOO (no soporta procesos nativos de GRASS GIS aún).
- CGAL (*Computational Geometry Algorithms Library*). Los servicios ZOO que están disponibles son Cgal_Delaunay y Ggal_Voronoi.
- OTB (*Orfeo Toolbox*). A través del soporte que provee ZOO, se pueden vincular los servicios y los algoritmos definidos con esta librería.
- SAGA GIS (*System for Automated Geoscientific Analyses*).

ZOO esta compuesto por tres partes y en la figura (4.4) podemos ver la arquitectura. La descripción de estos tres componentes son presentados a continuación:

- ZOO Kernel: es el componente *kernel* del lado del servidor, desarrollando en el lenguaje C y que tiene como funciones fundamentales manejar y encadenar *Web Services* codificados en diferentes lenguajes (C/C++,Fortran, Java, Javascript, Perl, PHP y Pyhton).
- ZOO Services: un conjunto de *Web Services* basados en librerías *Open Source*. Un ZOO-Service esta compuesto por un archivo de configuración que describe el servicio y el código que implementa la funcionalidad a ofrecer luego como servicio.
- ZOO API: Una API server-side desarrollada en JavaScript con la capacidad de llamar y encadenar los servicios ZOO, la cual facilita el proceso desarrollo.

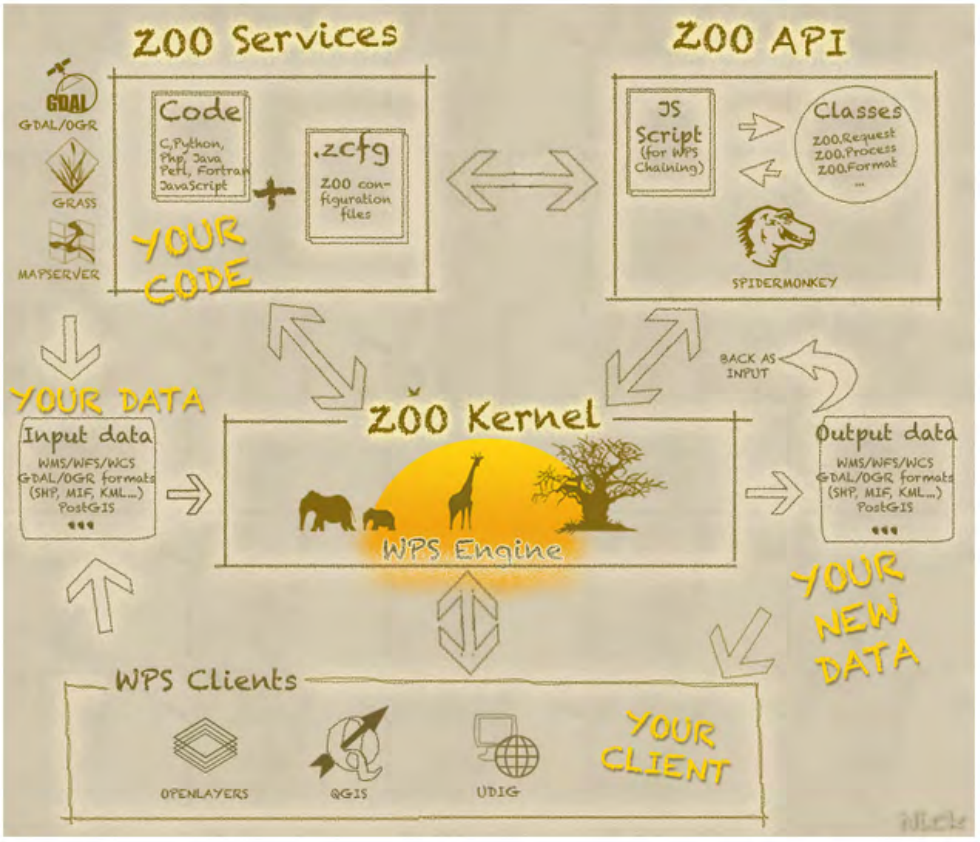


Figura 4.4: Architectura de ZOO-Proyect (Open WPS (2014))

Algunas cuestiones que se consideraron

- ZOO-Proyect tiene una amplia difusión, principalmente se ha presentado cada año, desde el 2010 hasta la actualidad, en las conferencias FOSS4G (*Free and Open Source Software for Geospatial*)
- La documentación es clara, presentando ejemplos en el sitio web del proyecto como así también a través de una serie de *Workshops* que se presentan cada año en el FOSS4G[4].
- Permite conectar software GIS y de procesamiento (GRASS GIS, SAGA GIS, GDAL, OTB) lo cual suma en versatilidad permitiendo además que un amplio espectro de programadores accedan a la tecnología.
- Esta fuertemente ligado a GRASS GIS y MapServer, logrando entre los tres componentes, una poderosa herramienta de procesamiento en la Web.
- Debido a que en su última versión implementa el protocolo WPS 2.0, podemos considerar esta tecnología de punta, mostrando asi un alto grado de competitividad con Geoserver-WPS y ArcGIS Server.

[4]http://wiki.osgeo.org/wiki/FOSS4G

- Es *Open Source* basado en MIT/X-11

4.2.5. Implementación del estándar WPS: ArcGIS Server WPS

Es un componente para la publicación de mapas en las Web, desarrollado por la empresa ESRI. Trabaja con otros componentes de ArcGIS (ArcGIS Desktop, ArcGIS Mobile) y contiene un módulo para geoprocesamiento. Desde la versión ArcGIS 10.1 podemos encontrar la implementación de WPS 1.0, el cual implementa las operaciones *getCapabilities, describeProcess y execute* (ESRI, 2015). Debido a que esta solución es de tipo propietaria, no se considera como opción para el presenta trabajo.

4.2.6. Elección realizada: ZOO-Proyect

De las implementaciones antes mencionadas, tanto Geoserver-WPS como ZOO-Proyect presentan opciones bastantes completas, ZOO-Proyect muestra mayor versatilidad en el momento de elegir un software de procesamiento para incorporar como proveedor funcionalidades gis y de procesamiento: acepta un amplio conjunto de software GIS de gran reconocimiento y aceptación entre los usuarios de tecnología geoespacial y permite incorporar servicios cuya implementación esta basada en el lenguaje Python. ZOO-Proyect es un componente *standalone*, lo que permite interactuar tanto con Geoserver como con MapServer al momento de acceder a los datos para los procesamientos, mientras que Geoserver es una solución que integra en si mismo su propio servidor de mapas.

4.3. Servidores de Mapas

Los servidores de mapas son componentes de software que tienen como función principal, entregar datos raster y vectoriales en la Web en forma de mapas y disponibles a través de los protocolos OWS. Permiten centralizar la información espacial evitando la duplicación de la información y promoviendo la reutilización de la misma por distintos usuarios en ubicaciones geográficas diferentes. Son un componente principal de las IDE (Infraestructura de Datos Espaciales), implementan los protocolos WMS, WFS y WCS (entre otros) ofreciendo a través de estos, los datos como *Web Service*.
Existen en diferentes lenguajes de implementación, destacándose en la comunidad de software libre, los servidores MapServer implementado en C y Geoserver en Java.

4.3.1. MapServer

MapServer es una plataforma *Open Source* para publicar datos espaciales y mapas interactivos en la web, bajo licencia MIT-Style corre sobre las plataforma Windows, Linux y MAC OS X. Esta desarrollado en el lenguaja C y en la actualidad es un proyecto de la OSGeo. Soporta ambientes de desarrollo basados en los lenguajes PHP, Python, Perl, Ruby, Java, and .NET. y numerosos estándares de la OGC como son WMS (cliente/servidor), WFS no transaccional (client/server), WMC, WCS, SLD y GML entre otros. Soporta en susversiónes mas recientes los protocolos: WMS 1.3.0, WFS 2.0, WCS 2.0. Está provisto para manejar una multitud de formatos raster y vectoriales, entre los mas populares el GeoTiff, ESRI Shapefile y conexión a bases de datos espaciales como PostGIS, MySQL y Oracle Spatial (MapServer, 2015a). El método de defínción de nuevas capas esta basado en archivos MapFile, que consisten en archivos de texto plano donde se definen las características de acceso al dato (raster o vectorial) y los estilos que se aplicarán sobre los mismos al momento del renderizado (MapServer, 2015b).

4.3.2. Geoserver

GeoServer es una plataforma basada en JAVA que implementa varios estándares de la OGC, es *Open Source* y esta liberado bajo la lincencia GPL2. Permite a los usuarios compartir y editar datos geoespaciales a través de las implementaciones de los protocolos de la OGC: WMS 1.3.0, WFS 2.0.0 y WCS 2.0.1 (en susversiónes más recientes) (GeoServer, 2015). También soporta WPS 1.0.0.
Provee una Interfaz de Administración Web muy completa, que permite manejar cada uno de los recursos (capas,espacios de trabajo, estilos, usuarios, roles, configuraciones de protocolos, procesos, etc.) de forma gráfica. Para los casos en que se requiera acceder programaticamente a los recursos de Geoserver, tal plataforma dispone de una interfaz RESTful (*REpresentational State Transfer*) a través de la cual los programas pueden subir y obtener información y hacer cambios de configuración haciendo llamadas HTTP (mediante operaciónes GET, PUT,POST, DELETE).

4.3.3. Elección realizada: Geoserver

Ambos servidores de mapas (Mapserver y Geoserver) presentan soluciones completas de acuerdo a los requerimientos para el presente trabajo. La diferencia que ayudo a tomar la desición es la interfaz RESTful y la interfaz de administración web que provee Geoserver. En cuanto a *performance*, si bien se tenderia a pensar que la implementación de Mapserver es mejor debido a que esta escrita en C, en contra posición de Geoserver escrito en JAVA, dependerá de la aplicación en particular. Así mismo, haciendo una adecuada asignación de

memoria RAM a la JVM (*Java Virtual Machine*) y estableciendo diferentes parámetros de forma adecuada (Boundless, 2015), se puede lograr un muy buen desempeño de Geoserver en entornos de producción.

4.4. Software GIS y de procesamiento

4.4.1. GRASS GIS

GRASS (*Geographic Resources Analysis Support System*) (Neteler and Mitasova, 2008), comúnmente llamado **GRASS GIS**, es un software GIS (Geographic Information System) Open Source usado para análisis y manejo de datos espaciales, procesamiento de imágenes, producción de mapas y gráficos, modelado espacial y visualización. Liberado bajo la licencia GPL v2+, es usado en el ámbito académico y comercial por muchas agencias gubernamentales (NASA, NOAA, USDA, DLR, CSIRO, the National Park Service, the U.S. Census Bureau, USGS) y empresas de consultoría ambientales. Es un proyecto oficial de la OSGeo (GRASS GIS Development Team, 2015b).
GRASS GIS contiene cerca de 350 módulos para renderizar mapas e imágenes; manipula archivos raster y vectoriales incluyendo redes vectoriales; procesa imágenes multiespectrales; crea, maneja y almacena datos espaciales. Ofrece dos interfaces de trabajo: un interfaz gráfica intuitiva y la linea de comandos para realizar las operaciónes. GRASS GIS puede hacer de interfaz con impresoras, plotters y bases de datos para crear nuevos datos como asi para manejar datos existentes. Esta desarrollado en el lenguaje C lo cual lo hace muy veloz a la hora de procesar la información, cuenta con API's en C (GRASS Development Team, 2015a) y Python ((GRASS Development Team, 2015b), (Zambelli et al., 2013)) para la generación de nuevos módulos o funcionalidades.

Estructura interna

Para almacenar la información, utiliza una estructura en árbol como podemos observar en la Figura (4.5), la cual es presentada para GRASS GIS 6 pero que se mantiene en GRASS GIS 7. Dicha estructura es creada y manipulada por GRASS GIS una vez que el usuario definió el *DATABASE* al momento de la instalación del software. El componente *LOCATION* define un nombre al cual se le asocia una proyección geográfica y el *datum*. El componente *MAPSET* define un directorio de trabajo sobre el cual corre una sesión de GRASS GIS. Se permiten varios *MAPSETS* por *LOCATION*, logrando así una estructura de organización para el trabajo de varios usuarios en red sobre el mismo *LOCATION*. En este sentido y con el fin de compartir mapas entre usuarios, existe por defecto el *MAPSET PERMANENT*.

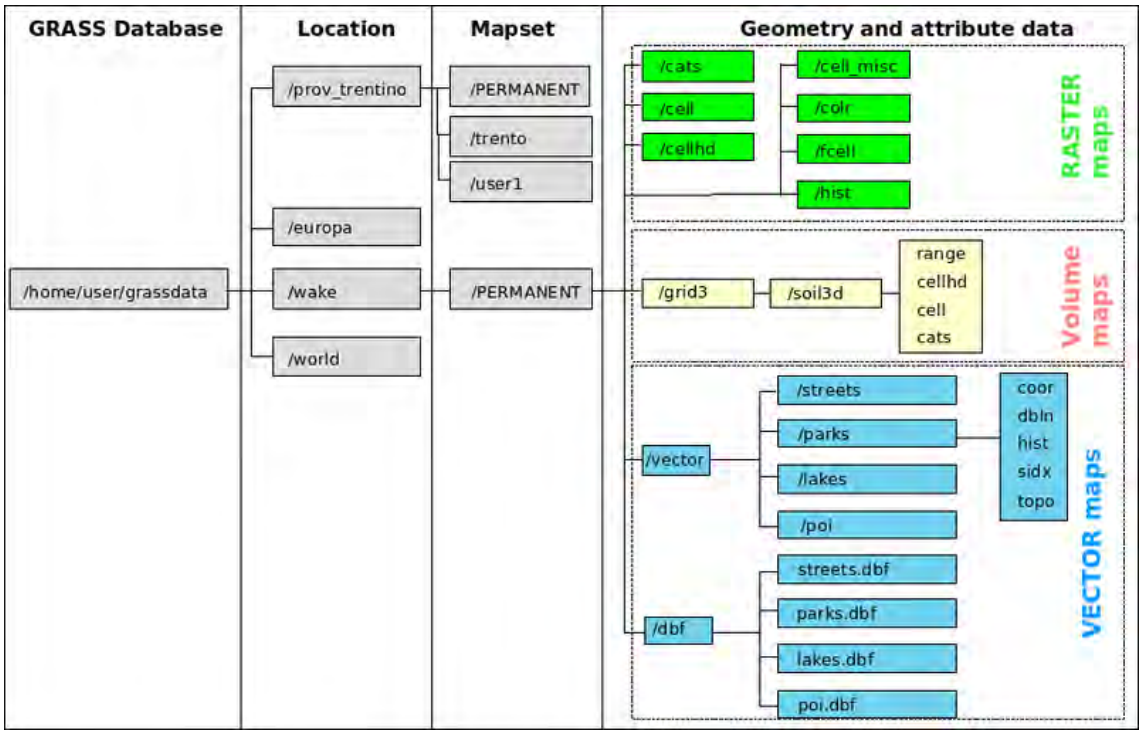


Figura 4.5: Organización de los datos en GRASS GIS
(GRASS Development Team, 2015a)

Arquitectura

En la figura 4.6 observamos el diagrama de los componentes internos de GRASS GIS y como se relacionan entre si. Los módulos de funcionalidades son agupadas según el tipo de recursos sobre el cual se trabaja:

- r.* corresponde a las funcionalidades nativas sobre datos RASTER 2D
- v.* corresponde a las funcionalidades nativas sobre datos VECTORIALES
- db.* corresponde a las funcionalidades nativas para el manejo de las bases de datos soportadas por GRASS GIS (tales como sqlite, postgreSQL)
- r3.* corresponde a las funcionalidades nativas para el manejo de RASTER 3D
- i.* corresponde a las funcionalidades nativas para el manejo de imágenes
- t.* corresponde a las funcionalidades nativas para el manejo de series de tiempo
- g.* corresponde a las funcionalidades nativas para funciones de propósito general (por ejemplo, g.region, g.list)

Las librerías de manipulación de datos raster y vectoriales son implementadas sobre GDAL/OGR y el manejo de proyecciones geográficas con Proj4, ambas librerías *Open Source*.

Por último, los datos sobre los que trabaja GRASS GIS pueden ser almacenados para su manipulación dentro del *DATABASE*, dentro de bases de datos y/o enlazado a directorios externos al *DATABASE*.

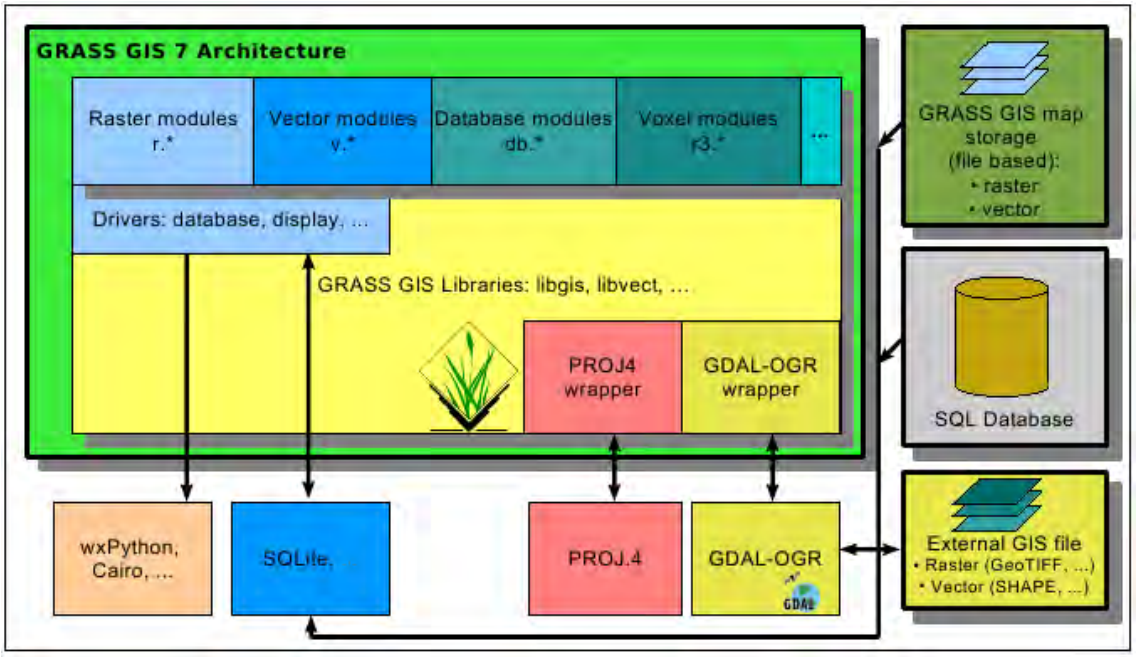


Figura 4.6: Architectura de GRASS 7 (GRASS Development Team, 2015a)

Módulos adicionales para GRASS GIS

Existen un gran número de extensiones o *addons* desarrollados por terceros para ampliar funcionalidades de GRASS GIS (GRASS GIS Development Team and third parties, 2015) (encontramos funciones de manipulación y procesamiento de datos MODIS *r.modis*; landsat *i.landsat* entre otras). Por otro lado, GRASS GIS y R se unen para generar una solución completa de analisis estadístico espacial y de buen rendimiento a través del módulo *rgrass7* (Bivand et al., 2009) para GRASS GIS 7, el cual permite ejecutar R desde dentro GRASS GIS o bien en un *LOCATION* temporal estableciendo la configuración adecuada.

4.4.2. GDAL/OGR en Python

Ambas librerías, GDAL y OGR, se utilizan para la manipulación de datos raster y vectoriales respectivamente. Sobre éstas, se implementa una API comúnmente referenciada como pyGDAL que contiene prácticamente las mismas clases y métodos implementados en GDAL y OGR, contenido en el módulo *osgeo* de Python. GDAL es un proyecto de la OSGeo con licencia al estilo X/MIT (OSGeo, 2015).

4.4.3. OSSIM

OSSIM (Open Source Software Image Map) es un software muy potente para teledetección para procesamiento, análisis, sistemas de información geográfica y fotogrametría. Diseñado como una serie de librerías de alta performance, esta escrito en C++ y emplea las últimas técnicas de diseño orientado a objetos (Community Software Open Source, 2015). Con el paquete OSSIM viene una gran variedad de utilidades de linea de comando, aplicaciones con interfaz gráfica y otros sistemas que han sido integrados. Ha sido liberado bajo la licencia LGPL y es un proyecto de la OSGeo.

Características

- Capacidad de procesamiento en paralelo con la librería MPI (Message Passing Interface).
- Modelos de sensores rigurosos.
- Modelos de sensores universales.
- Amplio rango de proyecciones de mapas y Datums soportados.
- Non-destructive, parameter based image chains.
- Acceso a archivos nativos.
- Corrección de terreno precisa y Ortorectificación.
- Soporte de elevaciones...y muchas funciones mas.
- Expone funconalidades para ser usadas a través de una interfaz SWIG como *wrapper*, utilizando diferentes lenguajes (Javascript, Perl, PHP, Python, Tcl,Ruby y otros).

4.4.4. Elección realizada: GRASS GIS y GDAL/OGR en Python

GRASS GIS y pyGDAL generan una solución suficientemente completa para proveer funcionalidades GIS y de procesamiento. GRASS GIS es un proyecto que coopera con ZOO-Proyect y hace que la elección sea directa, sumado a que ofrece un amplio conjunto de funcionalidades nativas y la posibilidad de desarrollar nuevos procesos y funciones en pyhton, lenguaje de gran flexibilidad y resultados rápidos. pyGDAL ofrece una solución que permite el desarrollo de geoprocesos, independientes de un software de procesamiento y por lo tanto se vuelve mas versatil.

4.5. Cliente Web GIS

4.5.1. librerías para Web Mapping

Web mapping es el proceso de diseñar, implementar, generar y entregar mapas y sus productos sobre la Web (Hazzard, 2011).

OpenLayers

Openlayers permite generar mapas dinámicos en la web. Permite visualizar *tiles*[5] de mapas y marcadores cargados desde diversas fuentes. Es una librería *Open Source* basada en Javascript del lado del cliente (en el modelo cliente/servidor) y ha sido liberada bajo la licencia FreeBSD. En los últimos meses fue liberada la versión Openlayers 3.8. Es un proyecto de la OSGeo Ol Community Software (2015).
Una característica importante es que soporta navegación en dispositivos móviles (Figura 4.7). Interactúa con servidores de mapas para devolver al usuario las peticiones que realiza: cargar un mapa, realizar zoom, consultar información y muchas otras. Por cada acción sobre el mapa se realiza una consulta al servidor de mapas, si esta fuera solicitar un mapa, Openlayers reune los diversos *tiles* devueltos en la petición y construye un mapa completo para ser visualizado (Hazzard, 2011).
Un característica importante es que Openlayers provee clases para la interacción con el servidor WPS (OpenLayers.WPSClient y OpenLayers.WPSProcess), además de las clases que permiten interactuar con el servidor WMS y WFS.

Browser	Touch events	Multiple touches	Accelerometer	geolocation
iOS (4.x)	yes	yes	yes	yes
iOS (3.x)	yes	yes	no	yes
iOS (2.x)	yes	yes	no	?
iOS (1.x)	yes	no	no	?
Android	yes	no(2)	no	yes
Opera Mobile	no	no	no	yes
Symbian	no	no	no	no
IE7 (WP7)	no	no	no	no
Firefox 4(1)	no	no	no	yes

Figura 4.7: Soporte en navegadores de dispositivos móviles. Tabla obtenida de Ol Developer Team (2015)

[5]Un tile se entiende como la unidad mas pequeña de la imágen de mapa, generalmente son imágenes cuadradas y en el contexto de visualizadores de mapas, son útiles para optimizar el tiempo de carga del mapa en el visor

Leafleat

Es una librería Open Source, que tiene como principal ventaja y característica la creación de mapas interactivos para dispositivos móviles. Luego, también es utilizada para desarrollo de aplicaciones de mapas para la Web, compitiendo con Openlayers en ambos aspectos.
Leafleat esta diseñada con simplicidad y facilidad de uso, funciona de manera eficiente con las principales plataformas de escritorio y móviles, se puede ampliar con una gran cantidad de *plugins*, tiene una API bien documentada y un código de fuente simple y legible. Fue diseñada por Vladimir Agafonkin de MapBox y por un equipo de contribuyentes de diversas partes del mundo (Agafonkin, 2015). Es una solución moderna debido a que toma ventaja de las implementaciones HTML5 y CSS3, tecnologías que forman de las carcaterísticas de los navegadores modernos.

GeoExt

ExtJS es un *framework* Javascript utilizado por GeoExt. Existen versiones *Open Source* y con licencia (JS, 2015).
GeoExt es una librería compuesta por Openlayers y ExtJS, que permite construir poderosas aplicaciones GIS sobre la web, al estilo gráfico de los software GIS de escritorio, utilizando Javascript. Por ejemplo, la creación de una grilla conteniendo el resultado de una consulta WMS Getcapabilities con Openlayers; la creación de un control para dibujar geometrías que se asocia a un botón y se agrega en la interfaz web; y muchos otros.
GeoExt, es extendida mediante la libería GXM (GeoExt Mobile), la cual es basada en Sencha Touch[6] y Openlayers para soporte en dispositivos móviles. Trabaja sobre los navegadores de iOS (iPad, iPhone, etc.), el navegador nativo de Android y los navegadores conocidos de escritorio como Apple Safari o Google Chrome. Aunque GeoExt ofrece un conjunto de herramientas para aplicaciones dinámicas de cartografía en la web, no provee funcionalidades completas cuando es accedida sobre dispositivos móviles (Eijnden, B. and Jansen, M. and Monnerat, M., 2015).
GeoExt esta disponible bajo la licencia BSD y es mantenida por una amplia comunidad de individuos, empresas y organizaciones (GeoExt, 2015).

GXP

La OpenGeo Suite es una plataforma geoespacial para el manejo de datos y la construcción de mapas y aplicaciones web, desktop y móviles (OpenGeoSuite, 2015). GXP son componentes de alto nivel que extienden funcionalidades relacionadas a mapas. Las clases de estos componentes estan construidos como extensión de objetos GeoExt y/o OpenLayers. La API hace referencia a propiedades y métodos que son extensiones o modificaciones de clases de la

[6] Es un framework Javascript basado en MVC para la creación de aplicaciones web para móviles multiplataforma

librería ExtJS (OpenGeo, 2015). GXP es un framework, Boundless SDK, que forma parte de la OpenGeo Suite. Provee un conjunto de plugins y widgets que pueden ser utilizados para generar una aplicación de mapas básica.

Geoexplorer

Es un *template* desarrollado como parte de la OpenGeo Suite y esta basado en componentes ExtJS, Openlayers, GeoExt y GXP. Debido a que es construido principalmente con la librería ExtJS, tiene la apariencia de una aplicación de escritorio (Desktop) y no presenta un diseño adaptable (*Responsive Web Design*) para dispositivos móviles.

4.5.2. Elección realizada: Geoexplorer

Debido a que Geoexplorer integra las librerías, Openlayers y GeoExt, esta construido con el framework GXP, además de proveer un entorno de trabajo pre-desarrollado con funcionalidades básicas como consulta de atributos y agregado de capas con estilos pre-definidos entre otros, elegimos esta herramienta para ampliarla utilizando componentes GXP. Además, Openlayers provee un conjunto de clases que permiten la interacción con servidores WPS, tema central en este trabajo.

4.6. Otros software

La instalación de los software elegidos en las secciones anteriores se realiza sobre Ubuntu 12.04 LTS, debido a que es una distribución que mayor cantidad de software geoespacial soporta (hasta el momento de escribir este trabajo), tal como lo demuestra el proyecto OSGeo Live (Community OSGeoLive, 2015) el cual esta basado en Ubuntu.

4.7. Modelo con tecnologías

Una vez seleccionadas las tecnologías y habiendo presentado en el Capítulo 3 el modelo conceptual (ver 3.2) del sistema, elaboramos un nuevo modelo actualizado donde reemplazamos los estándares de la OGC con las tencnologías elegidas. La figura 4.8 presenta el modelo. ZOO-Proyect es la implementación del servidor WPS, el cual recibe las peticiones del cliente a través del servidor web e invoca los servicios definidos. GRASS GIS y pyGDAL son software GIS y de procesamiento que implementan un conjunto de algoritmos y los dejan disponibles para que ZOO puedan encontrarlos mdiante servicios, publicarlos e invocarlos.

Geoserver es la aplicación que accede a los datos geográficos, exponiendo los mismos a través de WMS, WFS y WCS. El cliente esta basado en GXP, GeoExt, Openlayers y ExtJS.

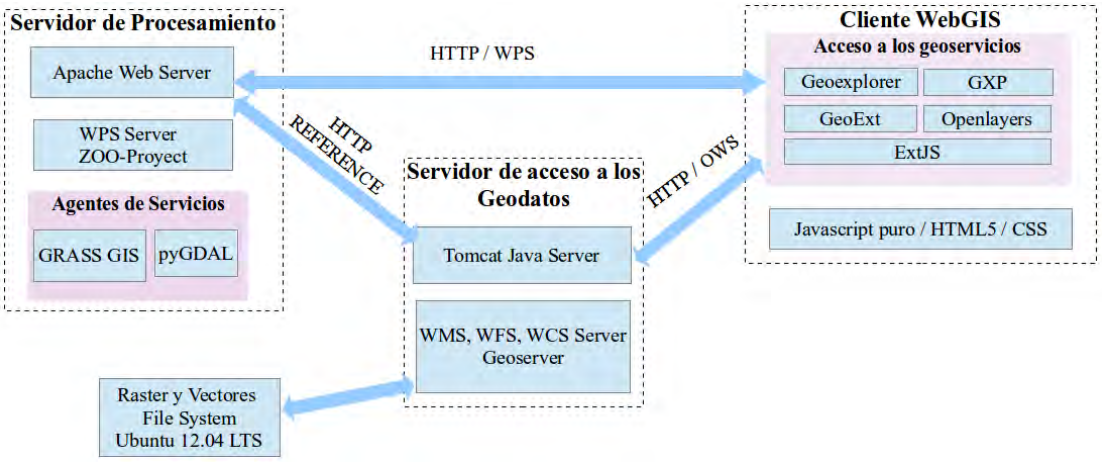


Figura 4.8: Arquitectura con las tecnologías elegidas

Notemos que en la arquitectira anterior, no se considera el uso de bases de datos y se decidió trabajar unicamente con archivos en el servidor como fuente de datos. Las razones de esta desición, responden a las siguiente consideraciones:

- El acceso a los datos espaciales es responsabilidad del servidor de mapas y transparente para el servidor de procesamiento, hecho que no modifica el procedimiento que se describe en este trabajo.
- Escases de tiempo: el trabajo con bases de datos espaciales requeriría mas tiempo de implementación y configuración, motivo por el cual se debió acotar el trabajo en esta tesis.

Capítulo 5

Diseño e Implementación

Este capítulo describe como se diseñó e implementó el módulo de procesamiento que incluye: servidor WPS, GRASS GIS y pyGDAL, modulo cliente y geoprocesos. En este capítulo se cumplen los siguientes objetivos:

1. Incorporar un mecanismo que entregue datos geoespaciales en la web.
2. Describir/establecer la forma en que deben interactuar los componentes y tecnologías para la publicación de geoprocesos en la web.
3. Definir e implementar los algoritmos y geoprocesos requeridos mediante la implementación de algoritmos para los servicios de procesamiento o geoservicios y generar un cliente para la interacción con los mismos.

5.1. Publicación de datos geoespaciales en la web

Para el procesamiento es necesario que los datos raster y vectoriales sean accesibles por el cliente WebGIS, para lo cual existen dos opciones:

- A través de los protocolos WCS y WFS (para datos raster y vectoriales respectivamente)
- A través de un directorio accesible para el servidor web

Sin embargo, se decidió trabajar con la primer forma debido a que presenta una solución completa de acuerdo a los objetivos planteados.

5.1.1. Datos accesibles por protocolos WCS y WFS: Geoserver

Se instaló Geoserver con el fin de publicar los datos espaciales, raster y vectores, mediante los protocolos WCS y WFS. El almacenamiento de dichos datos reside en el sistema de archivos del servidor desde el cual son leidos por Geoserver.

5.1.2. Configuración de Geoserver

En Geoserver el objeto central es la **capa**(*layer*), la cual se organiza dentro de un **almacén de datos** (*stores*) y éste en un **espacio de trabajo** (*workspace*), a su vez, a una capa se le asocia un estilo definido con SLD.
El conjunto de capas se pueden cargar a travéz de la interfaz de administración web de Geoserver o por medio de *scripts* que interactuan con el servicio RESTfull. Cualquiera sea el método de carga, debemos asegurarnos que los protocolos WCS y WFS esten activos (Figura 5.1) y que las capas cargadas queden publicadas (Figura 5.2).
Una vez efectuadas estas configuraciones, podemos acceder a las capacidades (operaciones y datos) que ofrecen estos servicios, a través de la solicitud *GetCapabilities* y la URL:

```
http://mydomain/geoserver/[NAME\_SERVICE]?request=GetCapabilities
```

donde NAME_SERVICE es alguno de la lista WMS, WFS, WCS

Figura 5.1: Servicios WCS y WFS habilitados

Figura 5.2: Capas habilitadas para los servicios

5.2. Servidor de procesamiento

Se compone de un servidor web (Apache), un servidor de datos espaciales (Geoserver), un servicio que implementa WPS (ZOO-Proyect) y un software de procesamiento de imágenes y funcionalidades SIG (GRASS GIS y librerías Python). Además, para la generación de Servicios ZOO implementados en GRASS GIS utilizamos las librerías wps-grass-bridge y PyXB 1.1 para vincular con ZOO-Proyect. Los pasos de instalación de cada uno de estos software pueden consultarse en la documentación oficial de cada sitio. En las siguientes secciones, iremos describiendo paso a paso la metodología para generar un servidor de procesamiento web de datos geoespaciales. Tales pasos, se deducen de la manera en que se deben interconectar los paquetes de software antes listados.

5.2.1. Generación de geoprocesos en GRASS GIS y PyGDAL

Se implementaron dos cálculos sobre datos geoespaciales, uno sobre datos raster y otro sobre datos vectoriales. El trabajo realizado sobre GRASS GIS es diferente al realizado sobre Python, mientras que el primero requiere pasos extras, el segundo es mas simplificado.

5.2.1.1. Geoproceso sobre datos raster: índice de area quemada

Se implemento en GRASS GIS el cálculo del índice de area quemada NBR (*Normalized Burn Ratio*) utilizando imágenes Landsat 8 (l8) (Solorza et al., 2014). La ecuación correspondiente a este índice se muestra en la figura 5.3.

$$NBR_i = \frac{\rho_{i,NIR} - \rho_{i,SWIR}}{\rho_{i,NIR} + \rho_{i,SWIR}}$$

Figura 5.3: Ecuacion del ínidice Normalized Burn Ratio

donde

$$\rho_{i,SWIR}$$

representa la reflectividad del pixel i en la banda del SWIR (banda 7 en Landsat 8 OLI/TIRS) y

$$\rho_{i,NIR}$$

representa la reflectividad del pixel i en la banda del NIR (banda 5 en Landsat 8 OLI/TIRS). En la Figura A.1 del Anexo se presenta una lista completa de las bandas Landsat 8.
El geoproceso se desarrollo en Python y se incorporó a GRASS GIS como un comando interno, con el nombre *r.nbr*. El nombre del comando sigue la convención de GRASS GIS en la forma en que clasifica sus funcionalidades y debido a que *r.nbr* es un cálculo sobre datos raster, debe estar precedido por la letra *r*.
Para la incorporación del proceso *r.nbr* como comando interno de GRASS GIS, se implementó el *script* usando la librería *Python Scripting Library* (GRASS Development Team (2015b)) con los comandos que definen el proceso. Por último, la defínción de las cabeceras del script, sigue las convenciones definidas en (GRASS GIS Development Team, 2015a).

Generación del comando r.nbr

Asumimos que las imágenes Landsat 8, que son *inputs* del cálculo, ya estan calibradas, por lo que este proceso no forma parte del comando en sí.
La implementación en GRASS GIS del cálculo del NBR, la realizamos en dos pasos: en el primero, escribimos la secuencia de comandos GRASS GIS que formaran parte del procesos completo y generamos un script Bash[1] que será luego ejecutado desde dentro de GRASS GIS. A continuación presentamos el *script*:

```
1 #Normalized Burnt Ratio (NBR)
2 #importar bandas 7 y 5 (swir y nir de l8)
3 r.in.gdal -o input=B5.TIF output=B5
4 r.in.gdal -o input=B7.TIF output=B7
5 #definir la region de trabajo
6 g.region rast=B5
7 #generacion del indice NBR
8 #nbr=nir-swir/nir+swir
9 r.mapcalc --o expression="NBR=(B5 - B7)/(B5 + B7)"
10 #Exportamos el resultado
11 r.out.gdal --o input=NBR output=NBR.tif format=GTiff
```

[1] http://www.gnu.org/software/bash/

En el código anterior, las lineas 3 y 4 se encargan de importar a la estructura de directorios de GRASS GIS, las bandas de la imagen Landsat. La linea 6 establece la extensión geográfica de trabajo y la resolución espacial, obteniendo estos datos de una de las bandas recien importadas (podemos utilizar cualquiera de las dos: B5 o B7). La linea 9 efectua el cálculo del índice mediante la aplicación de la formula dada en la linea 8. Y por último, la linea 11 exporta al sistema de archivos, el raster NBR.tif que contiene el resultado final.

El paso a continuación, consiste en ejecutar el código y chequear que genere el resultado esperado, para luego realizar el paso 2 que consiste en generar un comando interno de GRASS GIS mediante la traducción del código anterior a un *script* Python usando la API *Python Scripting Library*. Este paso es el previo a convertir el comando en un servicio. El script en Pyhton se muestra a continuación:

```
#############################################################################
#
# MODULE:       r.nbr
# AUTHOR(S):    Pablo Zader
#
# PURPOSE:      implements command r.nbr of GRASS
# COPYRIGHT:    (C) 2015 by Pablo Zader
#               This program is free software under the GNU General
#               Public License (>=v2). Read the file COPYING that
#               comes with GRASS for details.
#
# TODO: add sudo support where needed (i.e. check first permission to write into
#       $GISBASE directory)
#############################################################################
#%module
#% label: Implementa el programa r.nbr de GRASS GIS.
#% description: Genera un mapa raster del indice NBR(normalized burn ratio)
#%a partir de una imagen Landsat 8. Utiliza las bandas nir (banda5) y swir (banda7).
#% keywords: raster
#%end
#%option G_OPT_R_INPUT
#%key: nir
#%type: string
#%key_desc: nir
#%description: Nombre del raster de entrada que corresponde a la Banda del NIR
#%gisprompt: old,dir,input
#%required: yes
#%end
#%option G_OPT_R_INPUT
#%key: swir
#%type: string
#%key_desc: swir
#%description: Nombre del raster de entrada que corresponde a la Banda del SWIR
#%gisprompt: old,dir,input
#%required: yes
#%end
#%option G_OPT_R_OUTPUT
#% key: nbr
#% description: Nombre del mapa raster de salida que describe el NBR
#%end
import os
import sys
import grass.script as grass

def main():
    input_file1 = options['nir']
```

```
    input_file2 = options['swir']
    output_file = options['nbr']
    if not grass.find_file(input_file1):
                grass.fatal(_("Raster map <%s> not found") % input_file1)
    if not grass.find_file(input_file2):
                grass.fatal(_("Raster map <%s> not found") % input_file2)
    ret0 = grass.run_command("g.region", rast=input_file1)
    ret1 = grass.mapcalc("$NBR=($toar5 - $toar7)/($toar5 + $toar7)",
                                                        NBR=output_file,
                                                        toar5=input_file1,
                                                        toar7=input_file2
                                                    )
if __name__ == "__main__":
    options, flags = grass.parser()
    sys.exit(main())
```

En la implementación de este *script*, es importante generar los encabezados que definen los parámetros de entrada y salida del proceso, como así también el tipo de datos y la descripción de cada parámetro. Esta información es utilizada por GRASS GIS para generar la salida del comando con la opción *–help*.

Instalación del proceso r.nbr en GRASS GIS

Una vez generado el algoritmo en Python, es necesario compilar el código para incluirlo como un comando de GRASS GIS sumándose así al conjunto de comandos nativos del mismo, para luego ser publicado como un servicio ZOO. La estructura necesaria para crear el comando de GRASS GIS consiste de 3 (tres) archivos que requiere este software para incorporar el algoritmo o cálculo generado, en forma de comando. Estos archivos deben ser creados por el programador o usuario GIS y se describen a continuación (Ejemplos de definción de estos archivos, pueden encontrarse en el código fuente de GRASS GIS obtenido por SVN):

r.nbr.py: contiene el código escrito en Python, que ejecuta el proceso NBR
nbr.html: contiene documentación sobre el uso del proceso NBR
Makefile: contiene las instrucciones necesarias para compilar

Luego de crear estos archivos, ejecutamos la orden *make* dentro del directorio que los contiene, obteniendo un nuevo comando nombrado como *r.nbr*, y que pasa a formar parte de GRASS GIS como comando nativo.
Para chequear que se generó correctamente, probamos el comando desde la terminal de comando de GRASS GIS:

```
r.nbr --help
```

obteniendo en caso de éxito una salida mostrando la ayuda de uso de la función. Notemos que la informacion de salida generada con la opción *help* es leida por GRASS GIS desde la cabecera que hemos definido en el script r.nbr. Algunas opciones extras las incorpora GRASS GIS automáticamente (Código 5.1). En el caso que falle, GRASS GIS genera en la salida

estándar el mensaje de error correspondiente.

```
GRASS 7.1.svn (tempo):~ > r.nbr --help
Description:
 Implementa el programa r.nbr de GRASS.
 Genera un mapa raster del indice NBR (normalized burn ratio) a
 partir de una imagen Landsat 8. Utiliza las bandas nir (banda5) y swir (banda7).
Keywords:
 raster
Usage:
 r.nbr nir=nir swir=swir nbr=name [--overwrite] [--help] [--verbose]
   [--quiet]
Flags:
 --o    Allow output files to overwrite existing files
 --h    Print usage summary
 --v    Verbose module output
 --q    Quiet module output
Parameters:
    nir    Nombre del raster de entrada que corresponde a la Banda del NIR
   swir    Nombre del raster de entrada que corresponde a la Banda del SWIR
    nbr    Nombre del mapa raster de salida que describe el NBR
```

Código 5.1: Ayuda del comando r.nbr

De la misma manera que desde la linea de comando podemos obtener una descripción del uso del comando *r.nbr*, GRASS GIS ofrece una ventana gráfica que se genera automáticamente por cada proceso definido nativamente (Figura 5.4). Otras vistas gráficas las podemos observar en la Figura (2) del Anexo.

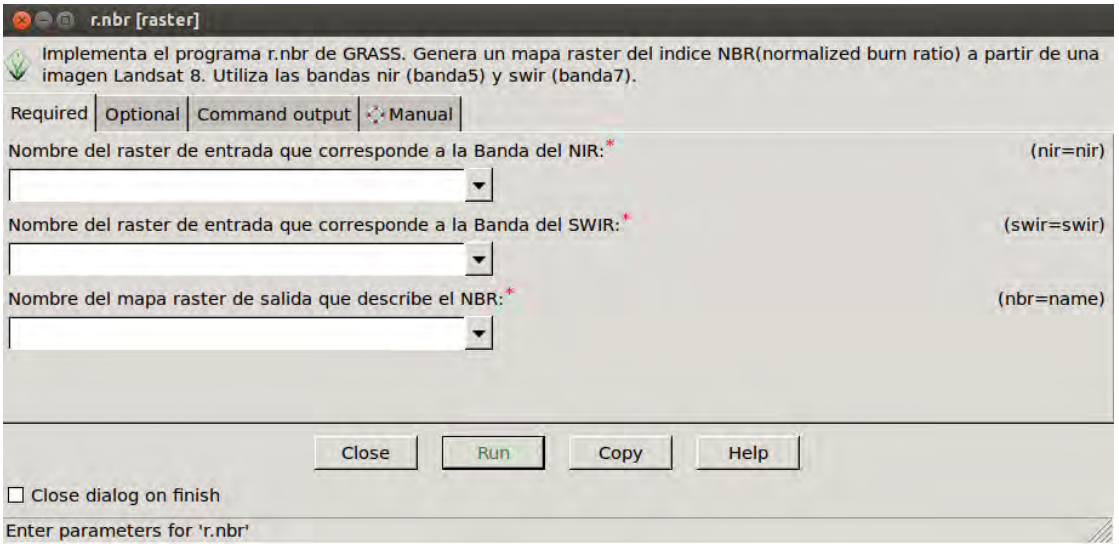


Figura 5.4: Comando r.nbr en GRASS GIS

5.2.1.2. Generación del Servicio r.nbr a partir del geoproceso

La defincón del servicio para el trabajo con ZOO-Service es relativamente directo cuando se trabaja con GRASS GIS ya que contamos con facilidades del software para generar el archivo de descripción del proceso; con un módulo que permite generar el archivo de configuración ZCFG del proceso y la interfaz en Python para interactuar con el comando ya definido, ambos con la estructura que requiere ZOO-Service. Se desprenden tres pasos para generar el servicio a partir de *r.nbr*:

1. Generación del archivo de descripción del proceso
   Una vez incorporado el proceso *r.nbr* como comando de GRASS GIS, hacemos uso de la opción *–wps-process-description* que automáticamente esta disponible para todo comando GRASS GIS. Con esta opción, generamos como salida el archivo de descripción del servicio en formato XML (Código 5.2), cuyo contenido es retornado cuando se realizan las llamadas *describeProcess* desde el cliente mediante el protocolo WPS.

```
GRASS 7.0.svn:~> r.nbr --wps-process-description > $HOME/wps-grass-bridge-read-only
/gms/Testing/Python/XML/r.nbr.xml
```

Código 5.2: Archivo de descripcion del proceso r_nbr

   La invocación del comando anterior, genera el archivo de descripción del proceso r.nbr.xml el cual es generado dentro de un directorio especial de la libería wps-grass-bridge, encargada de vincular GRASS GIS con ZOO WPS.

   El archivo XML de salida, contiene por defecto las secciones:

   ProcessDescription: define el nombre del servicio y una descripción del mismo sobre su funcionamiento.
   DataInputs: define el conjunto de entradas que el servicio recibe, describiendo para cada una si es de tipo obligatorio u opcional, el nombre del parámetro, el tipo de dato (literalData, ComplexData) y una descripción del parámetro.
   ProcessOutputs: define el nombre del parámetro de salida, tipo de dato y una descripción de la misma.
   El archivo completo puede consultarse en el código 3 del Anexo.

2. Creación del servicio ZOO NBR
   En este paso, creamos ambos archivos: el archivo que contiene el código a ser ejecutado y el archivo de configuración con extensión ZCFG que acompaña al mismo (Código 5.3). La generación de ambos archivos se logra usando el programa wps-grass-bridge (Google Summer Code, 2015) provisto para tal fin.

```
1 GRASS 7.0.svn:/wps-grass-bridge-read-only> python GrassXMLtoZCFG.py
2 -x gms/Testing/Python/XML/r.nbr.xml
3 -z zoo_services/r_nbr.zcfg -p r_nbr.py
```

Código 5.3: Generacion del Servicio ZOO: r_nbr

Mediante el programa GrassXMLtoZCFG.py en Python, en la linea 3 obtenemos los archivos *r_nbr.zcfg* y *r_nbr.py* que definen un servicio ZOO.

3. Publicación del servicio NBR
   Para finalizar debemos copiar ambos archivos generados en el paso anterior, en la carpeta definida en la instalación de Apache para la ejecución de programas externos CGI. En nuestro caso, se definió en /usr/lib/cgibin, pero podría variar.

Estructura de los archivos r_nbr.py y r_nbr.zcfg

Los archivos que son parte de un servicio ZOO tienen cierta estructura que deben respetarse. En este caso, como los servicios son creados a partir de comandos GRASS GIS, la estructuras y contenidos son automáticamente generados por librerías como describimos antes. En el caso de servicios generados a partir de código Python puro, las estructuras y contenidos deben ser creados manualmente en el momento del desarrollo, como se muestra en la subsección siguiente. El código completo de ambos archivos son presentados en los Código 7 y 8 del Anexo.

5.2.1.3. Geoproceso sobre datos vectoriales: Buffer

Un *buffer* es una función que genera una geometría vectorial a partir de otra geometría y distancia dada, haciendo que la geometría generada sea concéntrica a la dada y espaciada a la distancia especificada (Figura 5.5). Podemos generar un buffer sobre un punto, línea o polígono. En nuestro caso, definimos un *buffer* sobre un polígono con el objetivo de mostrar como se trabaja con datos vectoriales en llamadas WPS y ademas porque es un cálculo útil del cual disponen todos los software SIG.
La implementación de este cálculo se desarrolló con pyGDAL. Tal implementación ha sido provista a través del conjunto de ejemplos que dispone ZOO-Proyect, por lo tanto, describimos a continuación la estructura de la implementación y el formato que debe tener cualquier implementación similar pero para otros procesos.
La especificación de la función *buffer* en lenguaje coloquial, la expresamos de la siguiente manera: la función deberá tomar dos entradas, una geometría G y un numero D positivo y retornará una nueva geometría R concéntrica a G, con extensión geográfica mayor o igual a la original y cuyos contornos estarán separados unos de otros por una distancia D.

5.2.1.4. Generación del servicio Buffer a partir del geoproceso

La implementación de la función requiere respetar cierta estructura para convertirse posteriormente en un servicio. Al igual que con el proceso *r_nbr.py*, la función a ser invocada como servicio debe contener la parametrización adecuada como se muestra en el código 5.4.

```
Buffer(conf, inputs, outputs)

donde,

conf: recibe el entorno de configuraci n definido en el archivo main.cfg de ZOO-Proyect
inputs: corresponde a un arreglo de entradas recibidas mediante el metodo GET o POST
desde el cliente.
outputs: corresponde a un arreglo que contendr  las salidas producidas por la
ejecucion del algoritmo subyacente al Servicio.
```

código 5.4: Estructura de definicion de un servicio ZOO

La implementación completa de la función, la podemos encontrar en el código 2 del Anexo. Un vez implementado el código del servicio, generamos el archivo de configuración ZCFG que acompaña a dicha función. La estructura del archivo define una cabecera con paramétros generales sobre el proceso: nombre del archivo que implementa el proceso, el lenguaje en el que se implementó y variables para inidicar si el resultado del proceso puede almacenarse en el servidor, entre otros. Luego define una sección con las entradas del proceso, salidas y tipo de datos permitidos para ambas tal como se muestra en el código 5.5.
Una descripción total para este proceso puede encontrarse en el código 6 del Anexo. También, en el sitio web oficial de ZOO-Proyect pueden encontrarse distintos ejemplos de libre accceso.

```
[InputPolygon]
  Title = Polygon to compute boundary
  Abstract = URI to a set of GML that describes the polygon.
  minOccurs = 1
  maxOccurs = 1
  <MetaData>
      Test = My test
  </MetaData>
<ComplexData>
 <Default>
   mimeType = text/xml
   encoding = UTF-8
 </Default>
 <Supported>
   mimeType = application/json
   encoding = UTF-8
 </Supported>
</ComplexData>
```

Código 5.5: Definición en ZCFG: definición del entrada InputPoligon del Servicio buffer

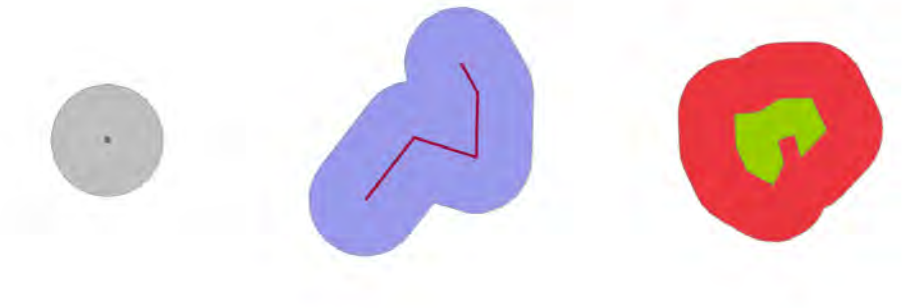


Figura 5.5: Buffers sobre punto, linea y polígono

5.2.2. Testing de los servicios web

Una vez creados los servicios web, es necesario chequear su correcto funcionamiento para asegurar que las aplicaciones clientes (en general) puedan utilizarlos sin falla alguna y en particular para asegurarnos que para la implementación del cliente que se define en la siguiente sección hemos eliminado cualquier error o falla del lado del servidor.
Se define un procedimiento (Figura 5.6) que deberá ser ejecutado cada vez que se agregue a la infraestructura un nuevo servicio web de procesamiento.

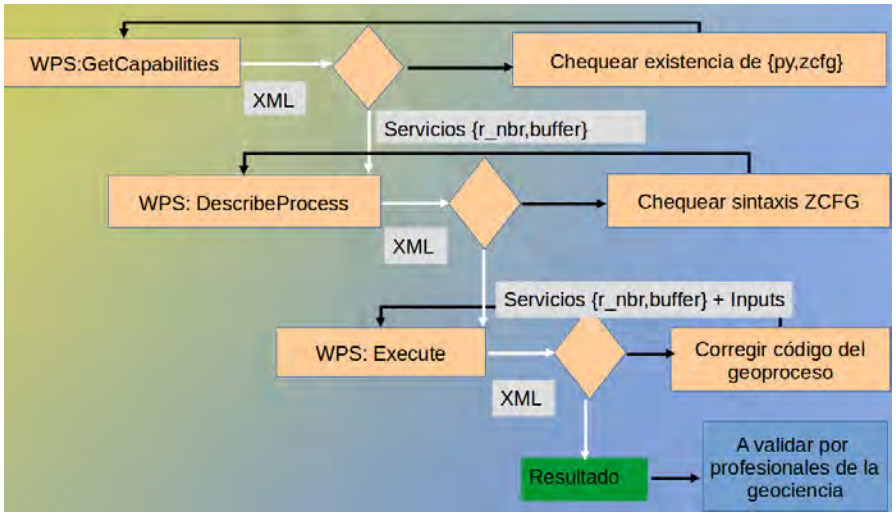


Figura 5.6: Procedimiento para testear los servicios web de procesamiento.

5.3. Diseño e implementación del Cliente

Una vez puesto en marcha el entorno del servidor de procesamiento y generados los geoprocesos, implementamos un cliente *front–end* que permite al usuario acceder a los geoprocesos definidos en el servidor.
En base a los geoprocesos definidos se generaron requerimientos generales para la aplicación cliente WPS y se relevaron requerimientos de usuario, ya que la interacción con el geoproceso *r_nbr* es diferente al *buffer*. Por otro lado, no es trivial generar un cliente web WPS debido a la diversidad de opciones que se deben considerar para la interacción con los geoprocesos. Si nos enfocamos en las opciones de ejecución que permite el protocolo, las salidas admiten dos formas de obtener los resultados: *RawDataOutput y ReponseDocument*. Para la última, se despreden una serie de posibilidades: obtener el dato crudo embedido en un XML; almacenar el resultado en el servidor como un recurso web (storeSupported=True); almacenar el resultado en el servidor y mantener actualizado el estado de ejecución del proceso almacenando un docuemento XML de estado (storeSupported=true, statusSupported=true). Si la respuesta es solicitada mediante *asReference*, entonces el resultado sera devuelto mediante un enlace contenido en un documento XML. Cuando se utiliza el parámetro *statusSupported=True*, las solicitudes hechas por el cliente son asíncronas; caso contrario corresponde a solicitudes síncronas.

5.3.1. Requerimientos del usuario

- El usuario deberá interacturar a través del cliente web WPS con los geoprocesos publicados por el servidor de procesamiento (mediante WPS).
- La invocación a los servicios web WPS deberá realizarse mediante botones, formularios y eventos disparados desde acciones sobre la interfaz y algunos otros elementos de diseño web necesarios.
- Las entradas de los servicios web WPS deberan elegirse desde la interfaz web: desde el panel de capas se deberán seleccionar capas raster y vectoriales; y mediante herramientas de dibujo se deberán generar polígonos para procesos que requieran este tipo de entrada.
- Los resultados deberán ser visualizados en el visor de mapas y/o quedarán almacenados en el servidor y se proveera un enlace donde podrán ser descargados.
- Los resultados de los procesos, deberán proveerse en formato GeoTiff y ShapeFile y en proyección Latitud-Longitud.

5.3.2. Requerimientos del cliente

- El cliente deberá implementar las llamadas a las operaciones *getCapabilites, describeProcess, execute y getProcess* para interactuar con los servicios WPS de geoprocesamiento implementados en el servidor.
- El cliente deberá implementar el acceso a dos tipos de servicios web WPS: uno que trabaje sobre datos raster y otro sobre datos vectoriales.
- Los tipos de resultados podran ser: capas raster, capas vectoriales, geometrías y valores literales.
- El cliente deberá ser capaz de invocar procesos síncronos y asíncronos.
- El cliente deberá ser robusto al tratar con procesos asíncronos.
- Todas las funcionalidades que se implementen deberán adherirse como *plugins y/o widgets* al cliente GIS Geoexplorer elegido para este trabajo.
- Disponibilidad de los datos. Para el caso de *r_nbr* los datos raster de entrada deberán estar disponibles a través del protocolo WCS y desde el servidor local únicamente. Para el caso del *Buffer*, el polígono debe ser dibujado sobre el mapa mediante una herramienta provista para tal fin y el valor entero ingresado en un campo de formulario.
- El tipo de dato de las entradas. Para el proceso *r_nbr* los raster de entrada se establecen en formato GeoTiff. Para el caso del *buffer*, las entradas son un polígono serializado en algunos de los formatos estándares definidos por la OGC, XML o JSON y un entero positivo.
- Resultado de los procesos. Para el caso de *r_nbr* el resultado debe estar disponible para la descarga en la interfaz del cliente y/o almacenado en el servidor para su posterior descarga en formato GeoTiff y sistema de referencia (SRS) EPSG:4326. Para el caso del *Buffer*, la geometría resultante debe ser incorporada sobre el mapa, superpuesta en el polígono de entrada, con la opción de ser descargada en formato KML y sistema de referencia (SRS) EPSG:4326.

5.3.3. Diseño de clases

La implementación de nuevas funcionalidades en la Suite OpenGeo se realiza mediante la incorporación de *plugins* usando la API GXP del SDK de Boundless. Estos *plugins* o herramientas, pueden tener acciones y/o salidas. Con una acción hacemos referencia a un botón o item de menú, mientras que con salidas nos referimos a algo visible sobre la pantalla como una ventana emergente o un panel[2].
A continuación presentamos dos diagramas de clases: un diagrama (5.7) que muestra las clases creadas para la interacción con geoprocesos mediante el llenado de un formulario web (interacción con el geoproceso *r.nbr*), donde las entradas deben ser completadas por el usuario;

[2] http://suite.opengeo.org/docs/latest/webapps/gxp/plugin/index.html

el otro diagrama (5.8), muestra las clases creadas para la interacción con geoprocesos que seran invocados mediante la utilización de herramientas extras como dibujar polígonos e invoca al servicio Buffer.

En los diagramas aparecen clases con bordes rojos y con bordes azules, las primeras son las que se implementaron completas en este trabajo; las segundas son clases ya existentes de la librerías Openlayers, GeoExt, ExtJS y GXP y son invocadas por las clases construidas. Cabe destacar que las clases existentes que se expresan en el diagrama UML, son las que implementan métodos para la interacción con WPS y se incluyen para dar al lector una visión extendida de la estructura que compone la solución.

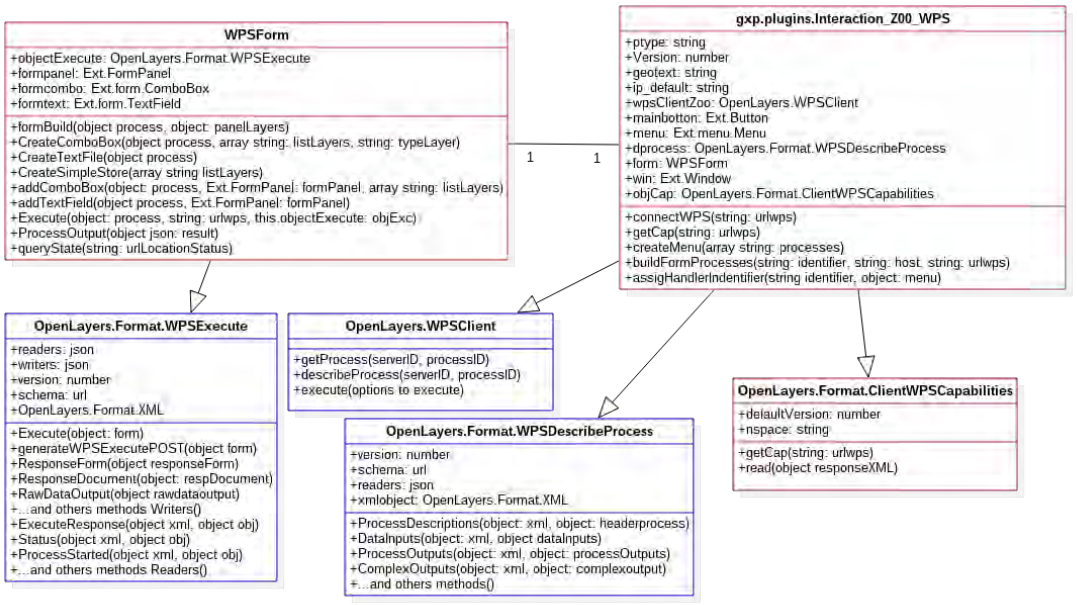


Figura 5.7: Diagrama de clases UML (para r_nbr)

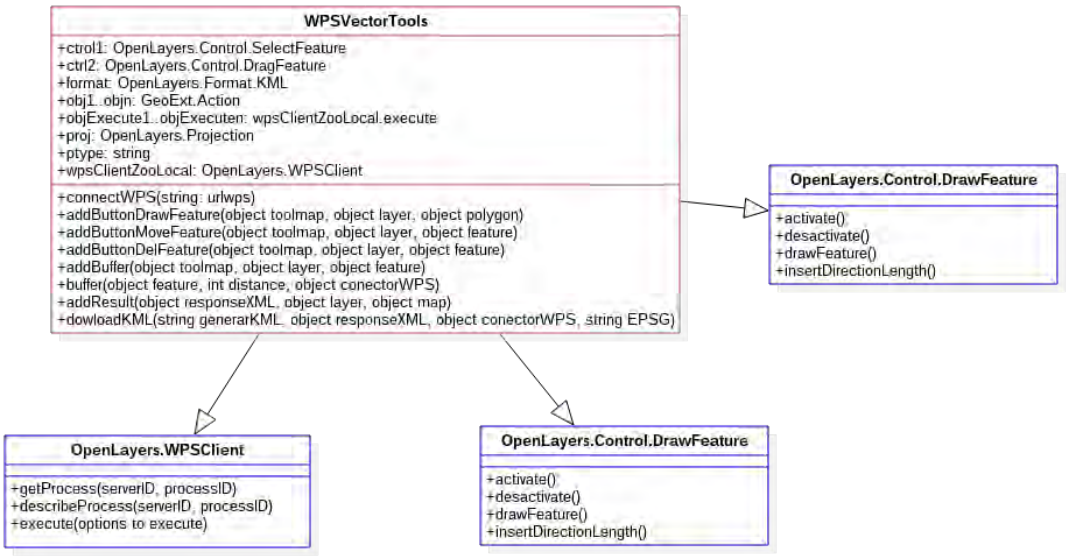


Figura 5.8: Diagrama de clases UML (para Buffer)

5.3.3.1. Descripción de las clases

Clase: gxp.Plugins.Interaction_ZOO_WPS
Dependencias
require plugins/Tool.js require OpenLayers/Format/ClientWPSCapabilities.js require OpenLayers/WPSClient.js require OpenLayers/Format/WPSForm.js
Descripcion
Implementa un cliente WPS que interactua con el servidor WPS (ZOO-Proyect) mediante las operaciones *getCapabilities, describeLayers, execute y getStatus*. Estas operaciones son invocadas mediante los métodos HTTP POST y HTTP GET. Los *inputs* de los geoprocesos invocados mediante el método *Execute* pueden ser de tipo WCS y WFS con acceso únicamente al servidor de geodatos local (localhost). Al cargar el cliente, esta clase hace una llamada WPS *getCapabilities* al servidor y genera un menú con los geoprocesos devueltos. Además, asigna a cada item del menú una función *handler* registrando así la información necesaria para la operación *describeProcess* que se invocará en llamadas subsiguientes. Una vez seleccionado un geoproceso de la lista, se invoca la operación *describeProcess* con el *identifier* del mismo, se toma la respuesta XML desde el servidor, se procesa el XML generando un objeto con la estructura del mismo y luego se invoca al método *buildForm* con este objeto, el que retornará un formulario cuyos campos corresponden a las entradas y salidas del geoproceso seleccionado; y un campo especial para informar el email donde se enviaran los resultados. Dicho formulario una vez llenado por el usuario y enviado, invoca a la operación *Execute* la cual retorna al cliente el resultado del geoprocesamiento para su descarga y además es enviado por email.

TABLA 5.1: Descripción de la clase gxp.Plugins.Interaction_ZOO_WPS

Clase: WPSForm
Dependencias
require OpenLayers/Format/WPSExecute.js
Descripcion
Implementa un formulario que se genera dinámicamente a partir de una respuesta *describeProcess* invocada por el cliente al momento de elegir un geoproceso para ser ejecutado. La llamada a *formBuild* es iniciada en *gxp.Plugins.Interaction_ZOO_WPS*. Cuando el método *formBuild* es invocado, se procesa el objeto XML describeProcess para conocer el tipo de cada una de las entradas del geoproceso (como parte del nodo *DataInputs*); cada una de las salidas del geoproceso (como parte del nodo processOutputs) y el tipo de datos de cada una de las entradas y salidas. Información adicional también es recuperada como el resumen de lo que hace el geoproceso. En cuanto a los tipos de datos de entrada, es capaz de manejar: *application/wkt, text/xml y image/tiff* para el objeto *ComplexData*; *bool, float, integer y string* para el objeto *LiteralData*; y *BoundingBox* para el objeto *BoundingBox*. Si el geoproceso acepta otro tipo de entradas, inmediatamente se rechaza su ejecución. En cuanto al tipo de datos de las salidas, se implementa **Respuesta Asíncrona con referencia**, lo que significa que el cliente recibirá un documento XML conteniendo un enlace al resultado del geoprocesamiento y dicho resultado permanecerá por un tiempo **t** en el servidor de datos y luego será borrado automáticamente. Los datos de entrada de tipo raster(image/tiff) y vectorial(text/xml) son capas cargadas en el panel de capas del cliente Geoexplorer; el usuario, mediante listas desplegables, selecciona las deseadas. Las capas permitidas para entrada de procesos solo pueden ser locales ya que se asegura la disponibilidad de las mismas mediante WCS y WFS. Para cada capa elegida como entrada de algun geoproceso, se genera automáticamente la URL de consulta (WCS y/o WFS) al servidor de geodatos con la operación *getCoverage* para el caso de rasters y *getFeature* para el caso de vectores. Se genera la validación de la información ingresada en campos de tipo LiteralData usando información que se obtiene en la respuesta *describeProcess*. Una vez cargado el formulario, el envio del mismo genera una consulta *Execute* de tipo HTTP POST con los datos del formulario. A partir del documento XML de respuesta obtenido, se genera un objeto y se analizan los indicadores del estado del proceso: *processStarted, processSucceeded y exceptionReport*. Si la ejecución del proceso fue iniciada exitosamente (*processStarted*), el cliente dispara un proceso **endprocess.py**, el cual se encargará de chequear cada 10 segundos (esto es configurable) el estado de avance del proceso, corriendo como proceso *background* en el servidor. Cuando el estado cambie a *processSucceeded*, el resultado será enviando por email al usuario. Mientras tanto, el cliente también lanza un chequeo cada 10 segundos y el cual devolverá al cliente en un popup el enlace del resultado del procesamiento. En el caso que la ejecución falle, se notificara tal resultado por email y el proceso se detiene. Ambos procesos, endprocess.py y el chequeo desde el cliente, hacen llamadas al servicio WPS *getStatus* retornando en cada llamada un documento XML con la información de estado de avance del proceso.

TABLA 5.2: Descripción de la clase WPSForm_ZOO_WPS

Clase: Openlayers.Format.ClientWPSCapabilities
Dependencias
requires OpenLayers/Format/XML/VersionedOGC.js requires OpenLayers/Format/XML.js
Descripción
Implementa lo métodos *getCap y read* y su función es trabajar con el documento de capacidades devuelto en la llamada *getCapabilities*, iniciada por *gxp.Plugins.Interaction_ZOO_WPS*. El método *getCap* invoca un operación *getCapabilities* mediante una llamada HTTP GET al servidor WPS. El resultado es convertido a un objeto XML mediante el método *read*. El resultado de este proceso, es pasado como parámetro para la construcción del menú de geoprocesos.

TABLA 5.3: Descripción de la clase Openlayers.Format.ClientWPSCapabilities

Clase: WPSVectorTools
Dependencias
require plugins/Tool.js require GeoExt/widgets/Action.js require OpenLayers/Control/DrawFeature.js require OpenLayers/Control/DragFeature.js require OpenLayers/Control/SelectFeature.js require OpenLayers/Handler/Polygon.js require OpenLayers/Handler/Path.js require OpenLayers/WPSClient.js require OpenLayers/Format/KML.js
Descripcion
Implementa la llamada al geoproceso *Buffer*, tomando como entrada un polígono dibujado sobre el mapa y un numero entero para la distancia. Esta clase crea un *toolbar* a partir del objeto *GeoExt.Action* de la librería GeoEx e inserta controles Openlayers como botones que definen el conjunto de herramientas para la llamada *Buffer*. Utilizando las librerías Openlayers, genera los botones para dibujar, borrar, mover y calcular el *Buffer* sobre la geometría dibujada en el mapa. La llamada al geoproceso es realizada utilizando el método *OpenLayers.WPSClient.execute* el cual ofrece una excelente abstracción incluso permitiendo encadenar geoprocesos (*chaining of WPS*). El resultado exitoso del proceso, invoca al método *addResult* encargado de agregar al mapa, el polígono recien creado. Seguido de ésto, una ventana de dialogo consulta al usuario sobre la posibilidad de descargar el polígono como formato KML, de ser afirmativa la respuesta se invoca al proceso WPS (en este caso no es un geoproceso, sino solo un proceso) *generarKML*, con el fin de devolver al usuario un archivo KML con el contenido calculado. Esta función, antes de hacer la llamda *Execute* reproyecta la capa vectorial que contiene el polígono de EPSG:90001 (proyección del mapa del visualizador) a EPSG:4326 (Latitud-Longitud), con el fin de ofrecer al usuario un sistema de coordenadas estándar. Por defecto, la llamada *Execute* se realiza de forma síncrona, mediante una llamada AJAX POST, indicando ésto mediante la opción *async:False* del objeto AJAX.

TABLA 5.4: Descripción de la clase WPSVectorTools

5.3.3.2. Diagrama de Secuencia

El diagrama 5.9 muestra la interacción entre los distintos componentes de la solución de geoprocesamiento para el caso de uso: **Solicitud *Execute WPS* asíncrona con la respuesta conteniendo el vínculo al resultado, almacenando el resultado en el servidor y enviando por mail el enlace al usuario**.

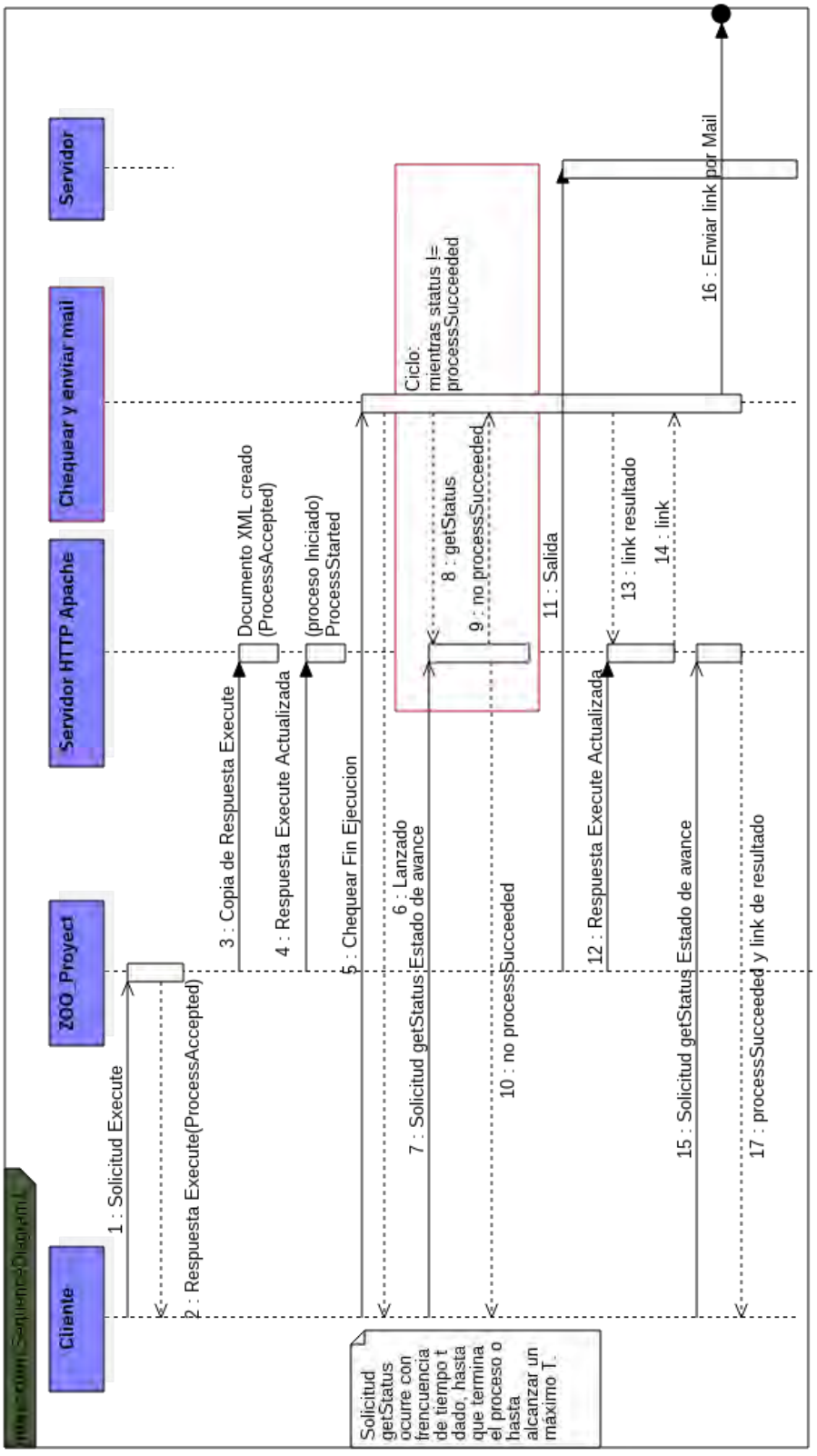


Figura 5.9: Interacción cliente-servidor en una solicitud Execute asíncrona con envio de resultados por mail

5.4. Resultados obtenidos

Presentamos las interfaces que se desarrollaron como parte del cliente para la interacción con los servicios r_nbr y Buffer y los resultados de las llamadas a ambos.

5.4.1. Interfaces de llamadas a los Servicios

Los servicios se acceden a través de la interfaz del cliente como se visualiza en la figura 5.10, donde se presenta el acceso a la lista de geoprocesos, mediante el botón Geoprocesos, que son invocados por medio del formulario de descripción del servicio; y por otro lado, un conjunto de botones: Dibujar, Mover, Borra y Buffer, que forman parte de la herramienta Buffer para acceder al geoproceso subyacente.

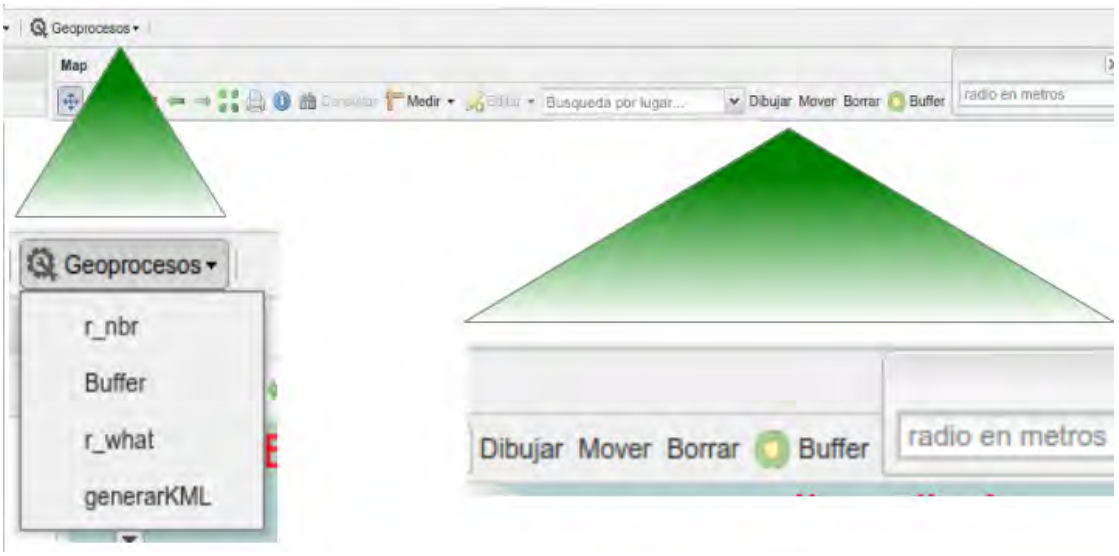


Figura 5.10: Acceso a los Geoprocesos desde la interfaz del cliente GIS

5.4.1.1. Caso: cálculo del índice de incendio NBR

El acceso al servicio r_nbr presenta al usuario el formulario de descripción del proceso (Figuras 5.11 y 5.12) , el cual un vez completado con los parámetros indicados, entregará al usuario el enlace al raster resultante del cálculo NBR y presentará un aviso de finalización del proceso.
La llamada al servicio *r_nbr* se realizó con una imágen Landsat8 del Parque Nacional Los Alerces, ubicado en Esquel, provincia de Chubut (Argentina) del incendio registrado el 23 de Marzo de 2015. El incendio dejó una superficie quemada de al menos 2000ha. Los datos técnicos de la imagen Landsat8 descargada para cubrir la zona de estudio se muestran en la Tabla 5.5. Se realizó una recorte de la misma, para acotar el cálculo a la zona del Parque Nacional, obteniendo una imagen de 4044 x 3291 pixeles, lo que da una superfice aproximada

de 12.000

$$Km^2$$

. Así mismo, el resultado no es significativo, ya que para obtener un resultado que realmente detecte el área quemada se debe hacer un procesamiento mas completo que incluye tres pasos (Jason Karl, 2015): un r_nbr pre-incendio con una imagen previa al incendio de la zona; un r_nbr post-incendio con una imagen del área quemada; y por último la diferencia entre el resultado del r_nbr pre-incendio y r_nbr post-incendio. Estos cálculos no se implementaron debido a que no son parte del objetivo de esta tesis pero podrían incorporarse sin demasiado problemas, generando un nuevo servicio que incluya los tres cálculos antes descriptos.

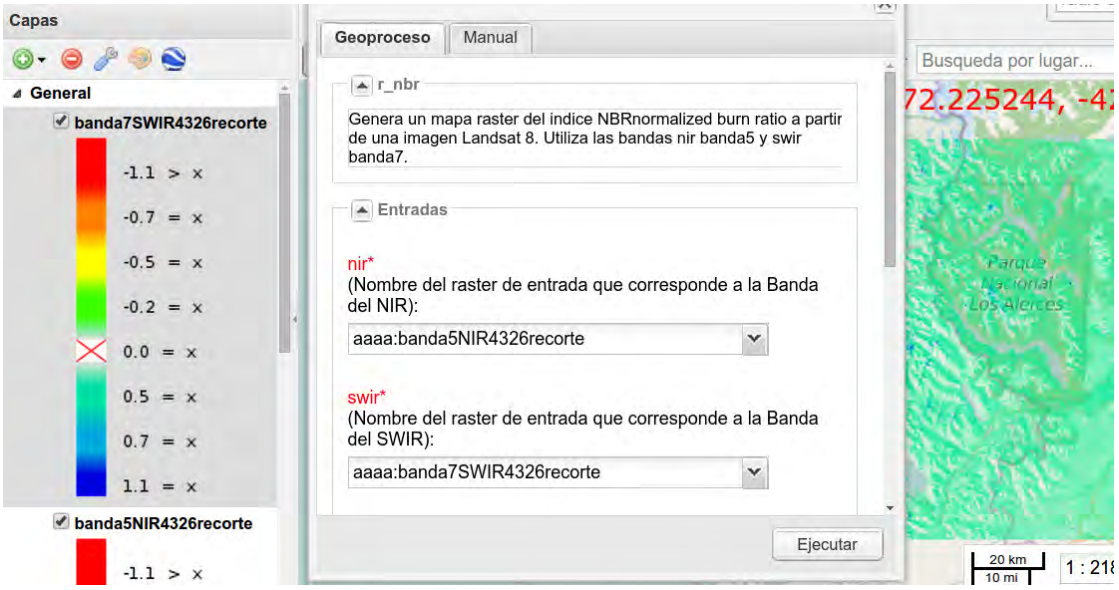


Figura 5.11: Primera parte del formulario de descripción del proceso r_nbr con la acción de Ejecutar(*Execute WPS*) el proceso actual

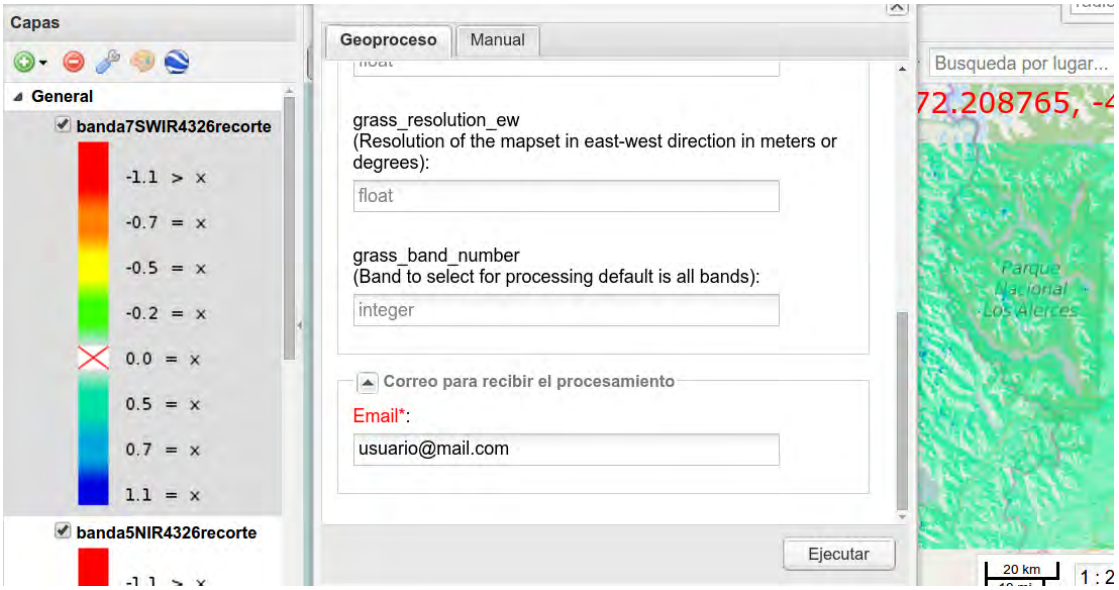


Figura 5.12: Segunda parte del formulario de descripción del proceso r_nbr con la acción de Ejecutar(*Execute WPS*) el proceso actual

Datos técnicos imagen Landsat8
Path/Row: 232/90
Fecha: 26 de Marzo 2015
Zona: Parque Nacional Los Alerces, Esquel - Chubut
Cobertura completa: ver figura 5.13
Tamaño original: 8041 x 8061. Recorte 4044 x 3291
Resolución espacial: 30mtrs para las bandas 5 y 7 utilizadas para el índice NBR
Fuente de la imagen: http://earthexplorer.usgs.gov/

TABLA 5.5: Datos técnicos

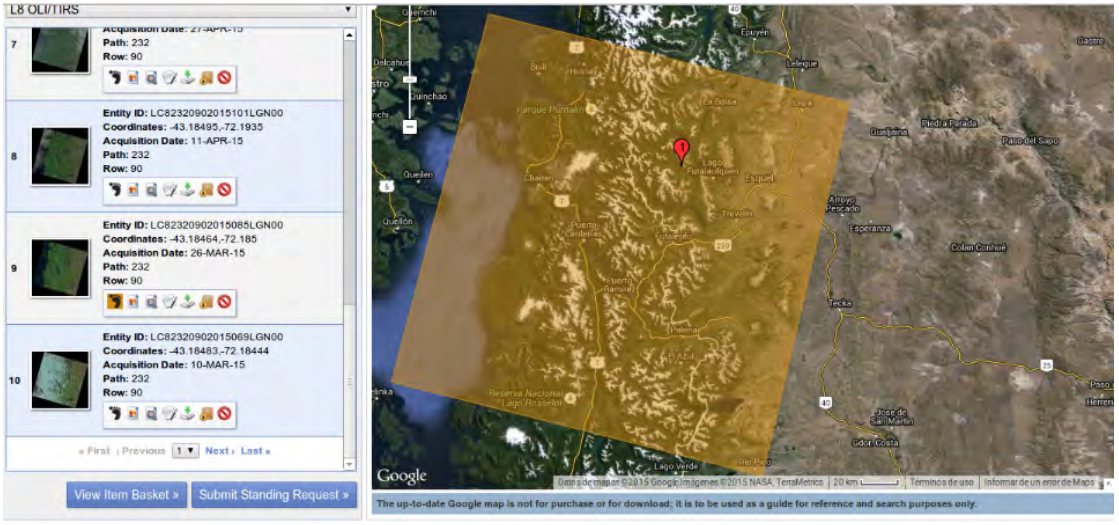


Figura 5.13: Cobertura de la escenca Landsat8 con pathrow 232/90

Cuando el cliente hace la llamada *Execute* al servicio r_nbr, al ser éste asíncrono, lanzará cada 10 segundos (estos es configurable) consultas al servidor para conocer el estado de avance del proceso, hasta que el proceso termine o hasta alcanza un máximo (Código 5.6). El máximo se define para finalizar el cliclo de consultas AJAX al proceso *getStatus* para el caso que algun error no controlado ocurriece y evitar que el cliente permaneciera consultando infinitamente.

```
//consultas cada 10000 ms
frecuency=10000
//Ponemos 30 intervalos, lo que equivale a 5 minutos de cota m xima de consultas
intervals=30
cota=frecuency * intervals
```

Código 5.6: Configuracion en el cliente

El resultado de la ejecución del proceso permanecerá en el servidor por un tiempo acotado y predefinido por ZOO-Proyect. El enlace es enviado por mail como se muestra en la Figura 5.14. La capa resultante del procesamiento se entrega en Latitud-Longitud, debido a que las bandas SWIR y NIR solicitadas mediante WCS, se hacen en Latitud-Longitud (código EPSG:4326), indicando esto en la petición.

Figura 5.14: Mensaje recibido por el usuario con el enlace al resultado

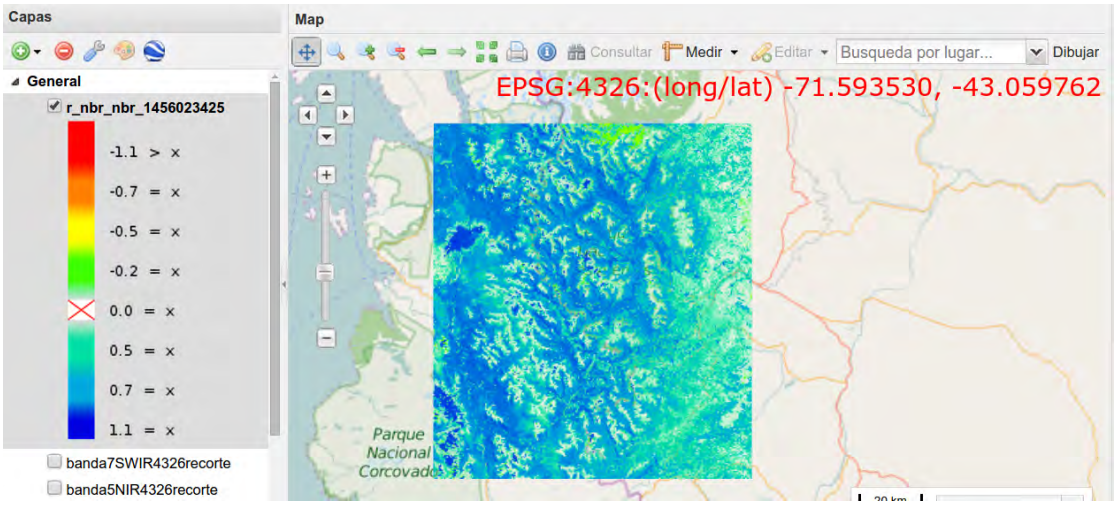


Figura 5.15: Cálculo del NBR

5.4.1.2. Caso: generación de un Buffer

El acceso a este servicio se realiza mediante un botón **Buffer** agregado en la interfaz de Geoexplorer. Como ejemplo funcional, se creó un Buffer sobre la Laguna Mar Chiquita - Córdoba. El usuario dibuja un polígono irregular con la herramienta **Dibujar** también presente en la interfaz. Una vez finalizada la acción, se ingresa un tamaño de Buffer en el campo disponible para tal fin, se selecciona el botón Buffer y se realiza un click sobre el polígono recien dibujado (Figura 5.16). El resultado de esta acción, un polígono concéntrico en el original pero de radio mayor al del polígono original dado, se superpone automáticamente sobre el original y da la opción al usuario de descargarlo en formato KML, con el objetivo de visualizarlo en Google Earth si así lo desea (Figuras 5.17 y 5.18).
Debido a que la llamada a Buffer es síncrona, no se implementa ningún mecanismo de control de estado del proceso en ejecución, y por el tipo de cálculo (un cálculo rápido sobre pocos datos) la respuesta vuelve inmediatamente.

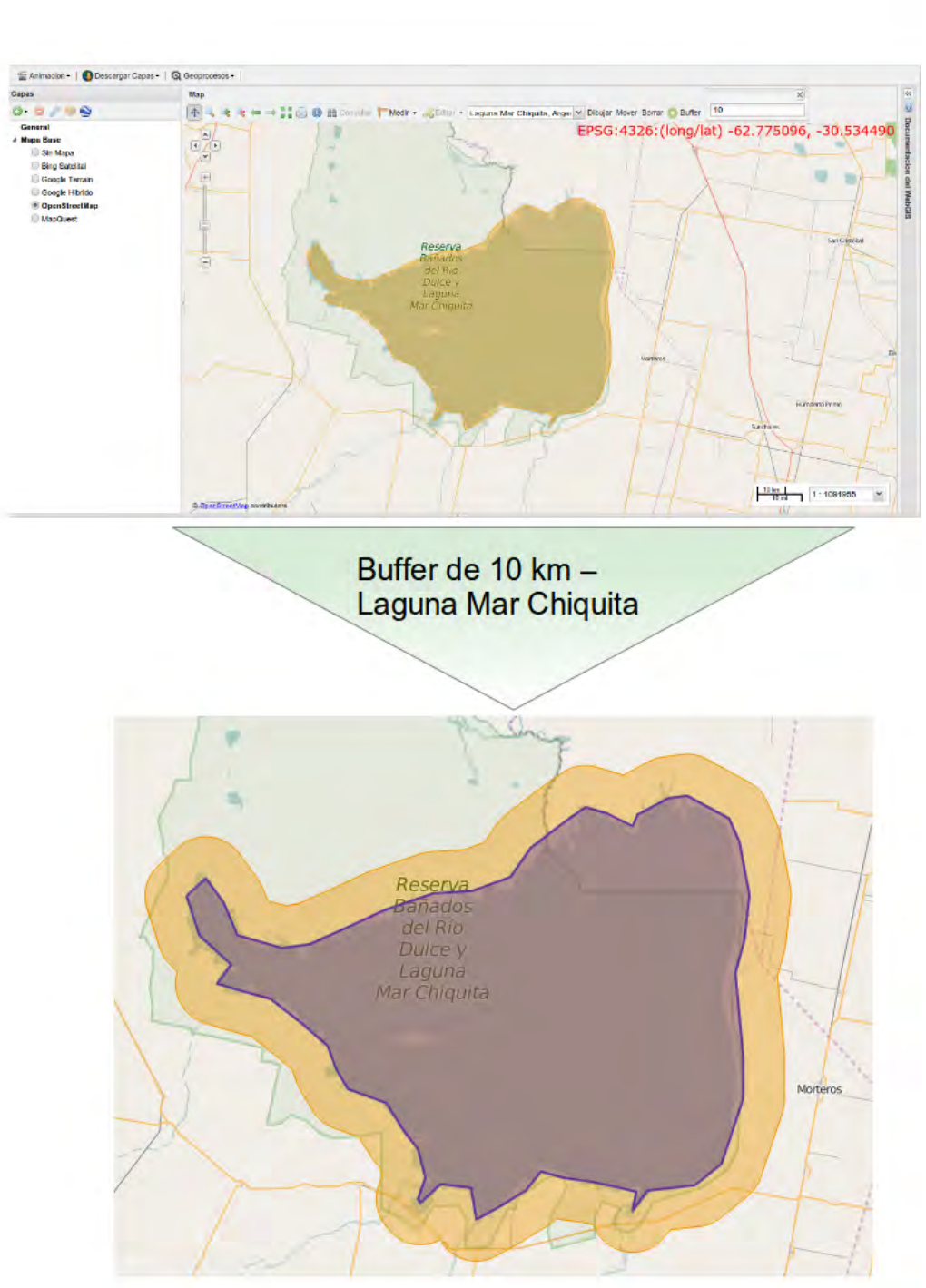


Figura 5.16: Calculo de Buffer de 10 km sobre la Laguna Mar Chiquita - Cordoba

Figura 5.17: Buffer resultante disponible para descargar como KML

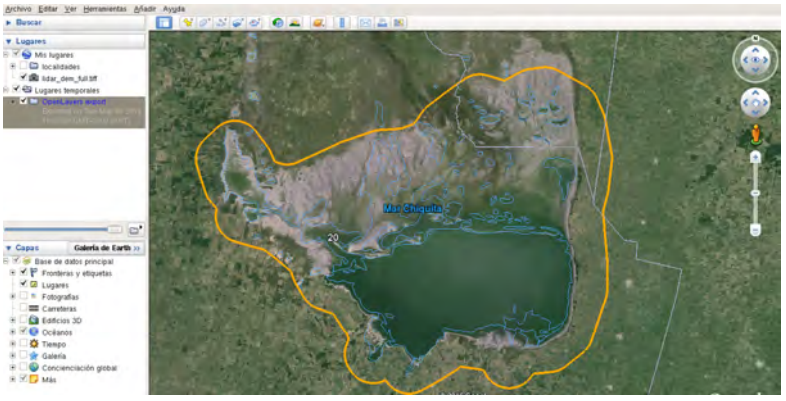


Figura 5.18: Buffer en formato KML visualizado en Google Earth

Capítulo 6

Aplicación de la tecnología de geoprocesamiento mediante WPS

En el presente capítulo se describe el desarrollo realizado para proyecto ISAGRO (Información Satelital para el Agro), que consiste de una IDE y la aplicación de la tecnología de geoprocesamiento usando WPS. Por una lado se requirió el desarrollo de la IDE para la publicación de una variedad de productos satelitales que se generan diariamente en el servidor del proyecto, y por otro lado, fue un ámbito de aplicación real de la tecnología de geoprocesamiento usando WPS, para lo cual se implementa el servicio r_what que provee acceso a los productos publicados, haciendo llegar la información satelital a través de una interfaz web simplificada a usuarios aún sin conocimientos en GIS. El trabajo requirió de la puesta en marcha de un servidor con las tecnologías mencionadas en esta tesis.
Se consibe el ISAGRO como un sistema de alerta temprana de riesgo productivo, donde el riesgo productivo se refiere todos aquellos eventos que afectan el rendimiento, la productividad, y por consiguiente los ingresos esperados del rubro. La actividad agrícola está sujeta a mucha incertidumbre derivada de factores climáticos y enfermedades (INIA, 2009).

En este capítulo se cumple el objetivo:

1. Configurar, instalar y poner de forma operativa un servidor con el sistema desarrollado.

   Cabe destacar que para la utilización de la tecnología que se describe en ésta tesis en el proyecto ISAGRO, hizo falta poner en marcha una IDE tal como se planteó en el apartado (a) del objetivo general.

El contenido del capítulo inlcuye las secciones:

1. **Introducción** En esta sección hacemos una breve introducción a los objetivos del proyecto PREISPA y área de estudio del mismo.

2. **Requerimientos del sistema** En esta sección proveemos la lista de requerimientos del sistema planteados para la generación de la IDE

3. **Arquitectura del sistema**

4. **Productos disponibles y proceso de elaboración** Describimos el proceso que se siguió para la elaboración de productos, software utilizado para la generación de los mismos y fuentes de datos. Los productos desarrollados forman parte del insumo de la IDE como así tambíen la visualización de capas disponibles en otras IDEs (IGN, Google, OpenStreetMap, etc) enlazadas mediante WMS.

5. **Uso del WPS: servicio r_what y Utilización del protocolo WPS por la aplicación web ISAGRO** Estas dos secciones son las mas importantes y relevantes en este capítulo. En ellas se describe la aplicación real de la tecnología de geoprocesamiento mediante servicios web, habiéndose implementado el servicio r_what el cual satisface un requerimiento del proyecto PRESIPA, lograndosé con esta solución una mayor difusión de la información disponible en la IDE y permitiendo que aplicaciones web externas puedan acceder al servicio y mediante él a los productos disponibles en la IDE.

6.1. Introduccion

El ”Programa Regional de Empleo de Información Satelital para la Productividad Agrícola”(PREISPA), es un proyecto que tiene como objetivo generar una red de información regional para difundir de un modo efectivo y amigable, información de origen satelital a los distintos actores públicos y privados interesados en utilizar la misma para la toma de decisiones agrícolas en cada uno de los países: Argentina, Chile, Paraguay y Uruguay. El PREISPA, nombrado como el ”Sistema de Información Regional ISAGRO, fue presentado en Agosto de 2015 (Uruguay, 2015) y tuvo una duración de 4 años (BPR-BID (2016)).
El PREISPA fue financiado por el BID (Banco Interamericano de Desarrollo), genernciado por la OEA (Organización de Estados Americanos) y ejecutado por la CONAE (Comisión Nacional de Actividades Espaciales de Argentina), MGAP (Ministerio de Ganadería Agricultura y Pesca de Uruguay), INIA (Instituto Nacional de Investigaciones Agropecuarias de Uruguay) y el CIREN (Centro de Información de Recursos Naturales de Chile)
La etapa final del proyecto, requirió el desarrollo de una IDE (Infraestructura de Datos Espaciales) para la publicación de productos generados durante el desarrollo del programa, que integró geodatos y geoprocesos para ser invocados a demanda, obteniendo así un WebGIS completo. La incorporación de geoprocesos como servicios es el tema central de esta tesis y el proyecto PREISPA es una aplicación de esta tecnología. Los requerimientos para el desarrollo de la IDE se describen en la sección 6.2. El area de estudio definida en el proyecto PREISPA es a escala regional y comprende los países de Sudamérica: Argentina, Chile, Paraguay y Uruguay (Figura 6.1).

Figura 6.1: Area de estudio PREISPA

6.2. Requerimientos del sistema

- Disponibilidad de información vectorial geográfica, tablas relacionales y datos asociados sobre una Base de Datos Geoespacial (GeoDB)
  Detalles
  El sistema debe aceptar información de tipo vectorial almacenada en archivos en disco y en bases de datos relacionales (PostgreSQL/PostGIS). En el caso de archivos vectoriales almacenados en disco, debe soportar al menos el formato ESRI Shapefile (shp). Ejemplos archivos vectoriales: carreteras, capitales de provincias, departamentos, localidades, provincias de un país.

- Disponibilidad de información raster (imágenes satelitales)
  Detalles
  El sistema debe aceptar información de tipo raster almacenada en archivos en disco. Ejemplos de archivos raster: imágenes satelitales (radar y óptico), mapas de riesgo.

- Disponibilidad de servicios web de información según estándares del OGC®: WMS, WFS, WCS.
  Detalles

El sistema debe contar con la capacidad de proveer y aceptar información vectorial y raster, a través de los servicios (protocolos) WMS(Web Map Service), WFS(Web Feature Service) y WCS(Web Coberage Service). La información de tipo vectorial es provista y recibida a través del protocolo WFS. La información de tipo raster es provista y recibida a través del protocolo WCS. A través del protocolo WMS se proveerán y recibirán datos raster y vectoriales en forma de imágenes. El sistema actúa como cliente y servicios en lo que respecta al uso e implementación de los protocolos antes descriptos.

- Disponibilidad de servicios de información comerciales (Google Maps, Bing Maps, Yahoo Maps)
  Detalles
  El sistema debe contar con la capacidad de aceptar información provista por los servicios de mapas web comerciales pero de libre uso: Google Maps, OpenStreetMap, Bing Maps y MapQuest. Esta información debe ser capturada con el protocolo WMS y usada como capas base de las demás. Estas capas son excluyentes entre ellas, es decir, solo se podrá visualizar una a la vez.

- Herramientas SIG para visualización, navegación, edición y consulta.
  Detalles
  El sistema debe proveer al usuario visualización y trabajo interactivo (navegación, zoom-in, zoom- out, desplazamiento, medición de áreas y longitud) con los mapas que el mismo almacena. Además, para el caso de capas vectoriales debe proveer funciones de modificación y agregado de geometrías como así también consulta sobre los atributos asociados. Para el caso de datos raster, se debe proveer la posibilidad de consultar del valor del pixel seleccionado.

- El sistema debe ser implementado en su totalidad solo usando software libre.
  Detalles
  Apache Tomcat, tecnología libre o GNU/Linux

- Alimentación de la GeoDB local con bases de datos externas.
  Detalles
  El sistema debe tomar datos de fuentes externas: ASCAT, AQUARIUS, SMOS

- Adaptación de diversos formatos a GeoTiff y ESRI Shapefile.
  Detalles
  El sistema debe estandarizar la información entrante (datos raster y vectoriales) que viene en formatos HDF5, DBL, NetCDF y GRIB a formato GeoTiff, para el caso de datos raster y a ESRI shapefile para el caso de datos vectoriales.

- Servicio de mapas con velocidad.
  Detalles
  El sistema debe presentar al usuario un acceso rápido a la información, sobre todo cuando se trata de archivos raster de gran peso como imágenes satelitales. Se debe realizar optimizaciones a nivel raster como la generación de subsets espaciales para los casos que sea posible y además, la generación de un cache o memoria intermedia para almacenar capas que son accedidas frecuentemente.

- Animación para la visualización de datos de series temporales.
  Detalles
  Se debe proveer una funcionalidad que permita visualizar los datos de series temporales de una forma amigable y útil para el usuario, para lograr entender el cambio de la variable de información en el tiempo

- Mecanismo para la recepción, conversión y subida automática de datos AQUARIUS, SMOS, ASCAT y meteorológicos provisto por distintas agencias espaciales
  Detalles
  El sistema debe contar con la capacidad de recibir información de fuentes de datos externas de forma automática, para alimentar la GeoDB (base de datos geoespacial) local. Las fuentes definidas hasta el momento son sitios FTP de Aquarius, SMOS, ASCAT. También datos meteorológicos provistos por el Instituto Gulich - CONAE que incluyen variables de temperatura, precipitaciones, vientos y humedad relativa, de la región de Sudamérica que contiene a los 4 (cuatro) países involucrados en el proyecto. En este último caso, no existe un mecanismo para capturar lo datos por lo que se debe implementar alguno. provisto.

- Interfaz de descarga de datos en formato estándar.
  Detalles
  Se debe proveer una forma sencilla al usuario que permita descargar una capa de información dada, desde la interfaz cliente del sistema. Para el caso de datos vectoriales el formato de descarga debe ser ESRI Shapefile; para el caso de datos raster el formato en que se debe proveer es GeoTiff.

- Generación del árbol de capas en la interfaz cliente de acuerdo a un criterio de organización.
  Detalles
  La sección de la interfaz del sistema que contenga el listado de capas disponibles, debe estar organizado en categorias de acuerdo al tipo de producto: Humedad de suelo, Fuego, Meteorología,Sequía,Precipitaciones,Heladas, NDVI, NDWI y Paises.

- Generar una sección en la interfaz de usuario donde estarán disponibles para la descarga, los *Algorithm Theoretical Basis Document*(ATBD)
  Detalles
  Se deberá definir una sección en la interfaz del usuario donde estarán disponibles los documentos ATBD de cada producto existente en la GeoDB local. Los documentos se deberán poder descargar en formato PDF. La carga de los mismos debe ser ingresada por el usuario al momento que carga una capa.

- Interfaz de llamadas a geoprocesos
  Detalles
  El sistema deberá publicar servicios que aplicaran sobre los datos almacenados en la GeoDB y cuyos resultados serán presentados al usuario. Estas funciones, llamados geoprocesos, serán invocados de acuerdo al modo de acceso al mismo.

- Geoservicio de consulta del valor de un pixel dada una coordenada geográfica.
  Detalles

Se debera proveer un servicio web que permita consultar las capas raster almacenadas en el servidor local, dada una o un conjunto de coordenadas geográficas.

- Disponer de un sistema que genere información estadística de acceso al sistema
  Detalles
  Se debera proveer una funcionalidad que permita medir el acceso a los distintos productos del WebGIS y generar estadísticas a partir de ellos, como ser: a) registrar que tipo de productos consume el usuario del sistema, b) desde donde se consumen, es decir, desde que IP se esta accediendo y c) fecha de acceso. Solo usuarios con privilegios de administrador podrán acceder a esta información.

6.3. Arquitectura del sistema

En la Figura 6.2 presentamos la arquitectura planteada para la IDE implementada, mostrando los distintos componentes de sofware desarrollados, componentes de software externos que se incorporaron a la solucion, catalogos de geodatos y geoservicios y la forma en que todos ellos se relacionan.

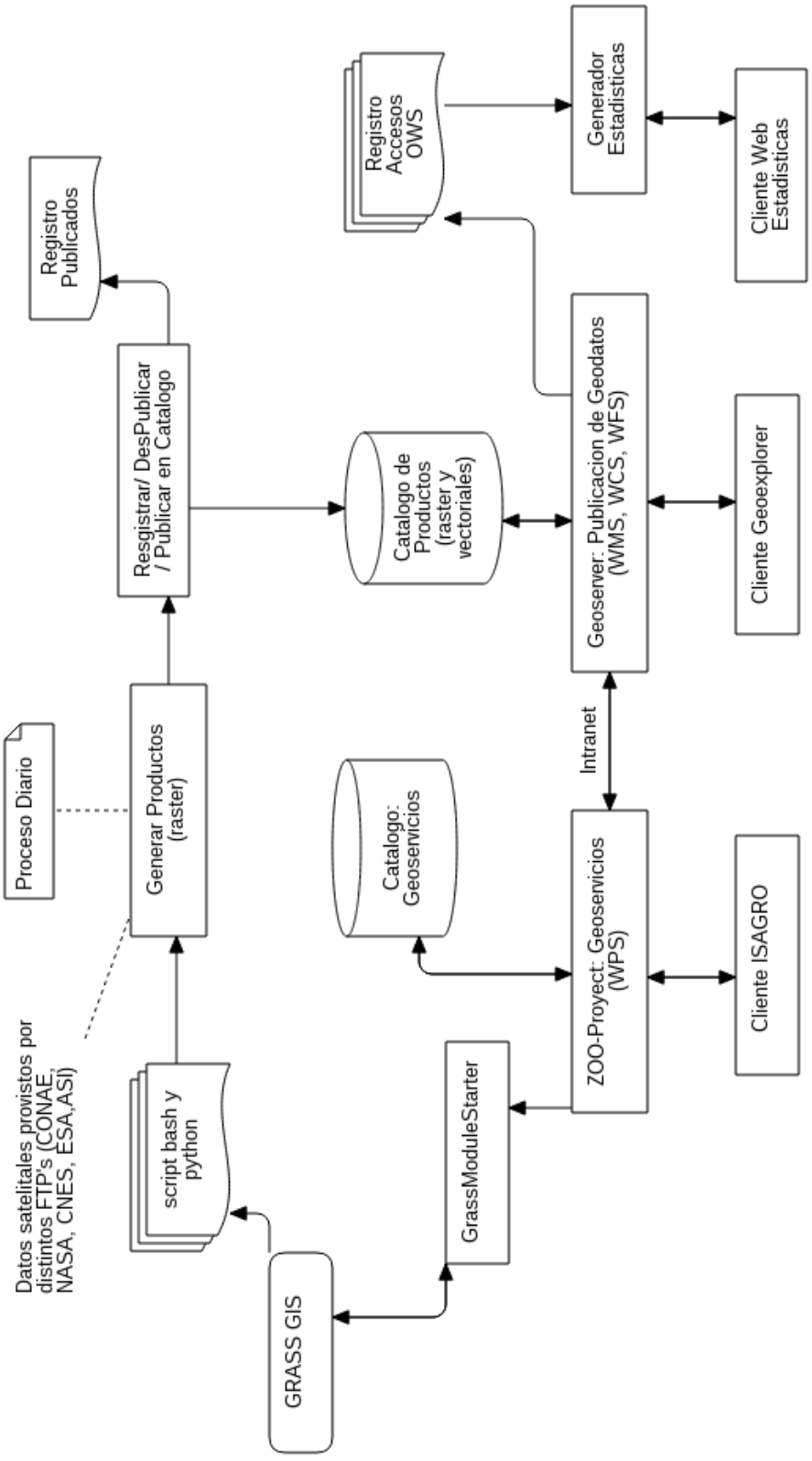


Figura 6.2: Arquitectura del WebGIS ISAGRO

6.4. Productos disponibles y proceso de elaboración

Los datos de entrada al sistema provienen de la ejecucion de algoritmos que corren diariamente en el servidor del proyecto generando nuevas capas raster. El resumen de los productos disponibles en la IDE se presentan en el cuadro 6.1. Los algoritmos para la generacion de los mismos han sido desarrollados en GRASS GIS 7.1, pyGDAL y GrADS por las distintas instituciones involucradas en el proyecto PREISPA y han sido elaborados los Documentos con la Base Teórica de los Algoritmos (en sus siglas en ingles, ATBD) que puede consultarse en las fuentes citadas en el cuadro de productos.
Los algoritmos desarrollados fueron incorporados en un proceso de automatizacion, descripto en los siguientes pasos:

- **Descarga de datos**
  Los datos utilizados como *inputs* de los algoritmos, provienen de distintos sitios FTP de diferentes Agencias Espaciales el mundo. Esta estapa del proceso realiza la descarga programada diariamente, obteniendo los datos necesarios para la elaboracion de los productos.

- **Conversion de formatos**
  Los datos descargados de los FTP provienen en formatos de datos científicos (para mencionar algunos, HDF, NetCDF, GRIB y DBL). La mayoría de estos formatos son utilizados por organismos del ámbito público y privado por su flexibilidad e independencia de las plataformas de software, ademas al ser formatos abiertos, el usuario desarrolla sus propias herramientas para extraer y analizar su contenido. En este paso del proceso, se extraen las variables de interés de cada archivo y se convierten a GeoTiff para ser manipulados y procesados con mayor facilidad por los distintos componentes de software utilizados en este trabajo (GRASS GIS, Geoserver, pyGDAL).

- **Reproyeccion**
  Este paso se aplica a los datos MODIS que provienen en proyeccion sinusoidal y son transformados a Latitud-Longitud.

- **Cálculo del producto**
  Se ejecutan los algoritmos para elaborar los distinos productos y se exportan para su publicación.

- **Publicacion en la IDE**
  Los raster obtenidos en el paso anterior, son publicados en el servidor de mapas para su posterior visualizacion y consulta. Luego, los datos quedan disponibles para ser accedidos a través de los protocolos WCS, WFS y WMS.

Tabla 6.1: Lista de productos disponibles en la IDE

Producto	Res. Espacial	Fuente	Autor
Areas Forestadas	100m	Alos Palsar de banda L	JAXA (2015)
Bosque(raster)	250m	MODIS	UNC-CONAE (2015b)
Fuego: FFDI	15km	WRF-CONAE	UNC-CONAE (2015a)
Focos de calor	-	MODIS	CONAE (2015b)
Heladas Actual	1km	MODIS	CONAE (2015a)
Humedad de Suelo	25km	ASCAT	ESA (2015b)
Humedad de Suelo	43km	SMOS	ESA (2015a)
Humedad Relativa	25km	WRF	CAEARTE-CONAE (2015)
Mapa Interface	500m	Landsat/Meris	CONAE (2015d)
NDVI	250m	MODIS	CONAE-MGAP (2015)
NDWI	250m	MODIS	CONAE-MGAP (2015)
Nubosidad	25km	WRF	CAEARTE-CONAE (2015)
Precipitaciones(pron)	25km	WRF	CAEARTE-CONAE (2015)
Precipitaciones actual	10km	GPM	CONAE (2015c)
Sequia	250m	MODIS	CONAE-MGAP (2015)
Temperatura	25km	WRF	CAEARTE-CONAE (2015)
Viento	25km	WRF	CAEARTE-CONAE (2015)

6.5. Uso del WPS: servicio r_what

Los productos satelitales publicados en la IDE estan a disposición para los usuarios y para otros clientes GIS, accedidos como servicios mediante los protocolos WCS, WFS y WMS. La IDE presenta los productos en forma de mapas y la información que estos proveen puede ser explorada de diferentes formas: usando herramientas de **Navegación** (zoom-in, zoom-out, paneo); herramientas de **Consulta** (consultar el valor de un punto del mapa clickeando sobre el mismo); mediante herramientas de medición de área y longitiud para conocer la magnitud de la distribución de cierta variable en el espacio; y por último a través de herramientas como **Descarga de Capas** que le permite al usuario de Sistemas de Información Geográfica utilizarla para alimentar sus modelos y generar nuevos productos(Figura 6.3). Otros de los usuarios principales de las IDE son los decisores políticos quienes obtienen a través de un mapa una rápida idea de una situación que sucedio, esta sucediendo o sucederá en el territorio. Para el proyecto ISAGRO, se suman nuevos actores como son asociaciones de productores, proveedores de servicios e insumos agrícola, interesados directamente en el uso de la informacion satelital. Debido a que este tipo de usuarios en general no tiene formación sobre interpretación de imágenes, conceptos de manejo de mapas y Sistemas de Información Geográfica (que integran una IDE), se requirió el desarrollo de nuevas funcionalidades que permitan presentar la informacion satelital producida, de una forma simple y de rápida interpretación. Dado que los productos presentes en la IDE muestran información climatológica y geofisica del día (otros son pronósticos hasta 3 días y algunos son históricos), se le posibilita al usuario obtener información tal como el valor de la sequía actual de su campo, el pronóstico de precipitaciones a 3 días, el valor de heladas actual y así sucesivamente, brindando de esta forma una herramienta sumamente útil, eficaz y de acceso simple.

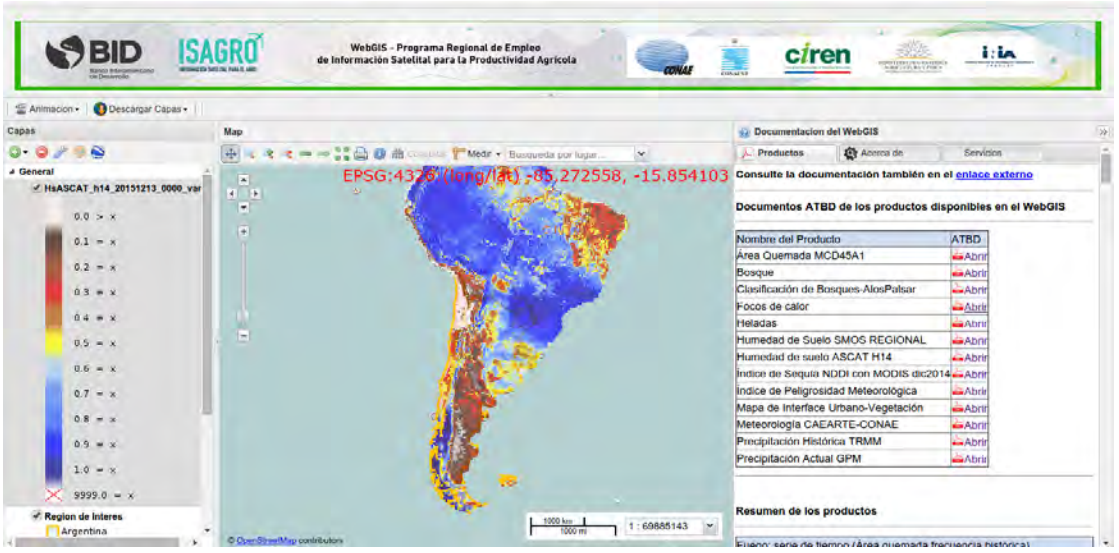


Figura 6.3: Aspecto de la IDE - Proyecto PREISPA

Para lograr las funcionalidades antes descriptas, surge un nuevo requerimiento de usuario:

Se deberá proveer una funcionalidad que permita extraer el valor de los pixeles de las capas raster almacenadas en la IDE para una parcela seleccionada por el usuario.

Para implementar este requerimiento geomático y acceder a la informacion solicitada, hacemos uso de las bondades de la IDE: el acceso a las capas de informacion alli almacenadas se logra mediante los servicios WMS y WCS. Accedemos entonces a las capas raster mediante el protocolo WCS y al listado de las existentes en la IDE a través de WMS.
Para la solución completa planteamos la implementacion de una funcion que tome como entradas una capa raster y un par de coordenadas geográficas y devuelva el valor del pixel en ese lugar. El siguiente paso, es incorporar esta funcionalidad como un servicio (o mas bien, geoservicio) y ser invocada mediante el protocolos WPS, tal como se describe en el Capítulo 5.

> ***Nota:*** La solución completa requiere dos partes: implementar un servicio WPS que cumpla el requerimiento antes descripto y desarrollar una funcionalidad en el cliente que acceda al servicio. En este trabajo, se genera el entorno de procesamiento en el servidor y se incorpora el servicio. El cliente de acceso al servicio fue desarrollado por consultores del proyecto PREISPA y no forma parte del presente trabajo

6.5.1. Defínción del geoservicio: r_what

La tecnología elegida para implemenar la solución, ha sido descripta en el Capítulo 4. En particular, entre los software GIS y de procesamiento, incorporamos a GRASS GIS ya que nos

provee entre sus procesos nativos, el comando r_what, que permite consultar el valor de los pixeles de una o varias capas a través de coordenadas geográficas. El prototipo se muestra en el código a continuacion.

```
GRASS 7.1.svn(mapset):~ > r.what --help

Description:
 Queries raster maps on their category values and category labels.

Keywords:
 r ster, consultar, posici n

Usage:
 r.what [-nfric] map=name[,name,...] [coordinates=east,north]
   [points=name] [null_value=string] [output=name] [separator=character]
   [cache=value] [--overwrite] [--help] [--verbose] [--quiet] [--ui]

Flags:
  -n   Cabecera de fila de salida
  -f   Mostrar las etiquetas de categor a de la celda(s) de cuadr cula
  -r   Producir valores de color como RRR:GGG:BBB
  -i   Producir valores de categor a enteros, no valores de celda
  -c   Activar informe de cach
 --o   Permite a los archivos de salida sobrescribir los archivos existentes.
 --h   Print usage summary
 --v   Salida detallada del m dulo.
 --q   Salida "silenciosa" del m dulo.
 --ui  Force launching GUI dialog

Parameters:
         map   Nombre de mapa(s) r ster existente a consultar
  coordinates  Coordenadas para consulta
      points   Name of vector points map for query
               Or data source for direct OGR access
   null_value  String representing NULL value
               preestablecido: *
      output   Nombre para el archivo de salida (si se omite o "-" salida a stdout)
    separator  Separador de campos.
               Special characters: pipe, comma, space, tab, newline
               preestablecido: pipe
       cache   Tama o del cach  de punto
               preestablecido: 500
```

Este comando permite generar consultas de los valores de los pixeles de una o varias capas raster y a partir de un par o un conjunto de pares de coordenadas geográficas. El resultado siempre serán los valores de los pixeles consultados o el simbolo * en el caso de que las coordenadas solicitadas caigan fuera de la extensión del mapa, o bien, el pixel no contenda datos. Por ejemplo:

```
{Consulta}

r.what map=HumedadDeSuelo coordinates=-65.0,-35.0 -f

{Resultado}

-65.0|-35.0||0.04865
```

El comando *r.what*, siguiendo los pasos descriptos en el Capítulo 5 , se incorporó a la IDE como un servicio WPS para luego invocarse por el cliente. En el codigo (5) se puede consultar el archivo de descripcion del servicio.

6.5.2. Acceso al Servicio r_what

El acceso al servicio WPS r_what implementado en la IDE, se logra mediante las dos operaciones *describeProcess y Execute*. A continuación describimos ejemplos de llamadas de tipo GET (ZOO-Proyect también implementa llamadas POST).

```
{describeProcess}

http://webgis.isagro.org.ar:8080/cgi-bin/zoo_loader.cgi?
request=describeprocess&
service=wps&
version=1.0.0&
identifier=r_what

{execute}

1- http://webgis.isagro.org.ar:8080/cgi-bin/zoo_loader.cgi?
2- request=Execute&
3- service=WPS&
4- version=1.0.0&
5- Identifier=r_what&
6- DataInputs=map=Reference@xlink:href=
7-    http://webgis.isagro.org.ar/geoserver/wcs?&
8-       service=WCS&
9-       version=1.0.0&
10-       request=GetCoverage&
11-       crs=EPSG:4326&
12-       format=GeoTIFF&
13-       coverage=Humedad_De_Suelo:HsAQUARIUS_2015.01.01&
14-       width=600&
15-       height=600&
16-       bbox=-180,-90,180,90;
17- coordinates=-65.0,-35.0;
18- grass_band_number=1&
19- responsedocument=output
```

La llamada *Execute* invoca al servicio r_what (linea 5), con la capa raster de entrada HsAQUARIUS_2015.01.01 obtenida desde la IDE por Referencia a través del protocolo WCS (linea 6 a linea 16) y con el par de coordenadas lat-long (-65.0, -35.0). El resultado es un documento XML que se muestra a continuación:

```
<wps:ExecuteResponse xmlns:wps="http://www.opengis.net/wps/1.0.0"
xmlns:ows="http://www.opengis.net/ows/1.1"
xmlns:xlink="http://www.w3.org/1999/xlink
"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd"
service="WPS" version="1.0.0" xml:lang="en-US"
serviceInstance="http://webgisisagro/cgi-bin/zoo_loader.cgi">
<wps:Process wps:processVersion="2">
```

```
        <ows:Identifier>r_what</ows:Identifier>
        <ows:Title>
        Queries raster maps on their category values and category labels.
        </ows:Title>
        <ows:Abstract>http://grass.osgeo.org/grass71/manuals/r.what.html</ows:Abstract>
</wps:Process>
<wps:Status creationTime="2015-05-02T10:50:41Z">
        <wps:ProcessSucceeded>Service "r_what" run successfully.</wps:ProcessSucceeded>
</wps:Status>
        <wps:ProcessOutputs>
                <wps:Output>
                        <ows:Identifier>output</ows:Identifier>
                        <ows:Title>
                        Name for output file if omitted or "-" output to stdout
                        </ows:Title>
                        <wps:Data>
                                <wps:ComplexData mimeType="text/plain">
                                        -65.0|-35.0||0.04865
                                </wps:ComplexData>
                        </wps:Data>
                </wps:Output>
        </wps:ProcessOutputs>
</wps:ExecuteResponse>
```

La parte mas importante de la respuesta XML es el bloque final (mostrado a continuación), donde se puede leer el valor del pixel (0.04865) para la coordenada dada (-65.0,-35.0):

```
<wps:ProcessOutputs>
                <wps:Output>
                        <ows:Identifier>output</ows:Identifier>
                        <ows:Title>
                        Name for output file if omitted or "-" output to stdout
                        </ows:Title>
                        <wps:Data>
                                <wps:ComplexData mimeType="text/plain">
                                        -65.0|-35.0||0.04865
                                </wps:ComplexData>
                        </wps:Data>
                </wps:Output>
</wps:ProcessOutputs>
```

6.5.3. Integración de WPS al cliente ISAGRO

La implementación del servicio sirvió de interfaz para construir una aplicación web que forma parte del ISAGRO. Dicha aplicación fue implementada por consultores del proyecto y cuya interfaz puede visualizarse en la Figura (6.4).
Cuando el usuario selecciona una parcela sobre un mapa, la aplicación web invoca el servicio tantas veces como coordenadas geográficas haya en la parcela y por cada raster que se deba consultar. Se visualizan las distintas variables provenientes de las capas raster de la IDE, presentadas en un formato amigables y cuya existencia complementa a la IDE y genera un valor agregado a la información satelital producida en el proyecto.
Algunas de las variables visualizadas son Heladas Actuales, Sequias Actuales, NDVI, NDWI, Humedad de Suelo, Precipitacion Actual, Temperatura, Vientos, Nubosidad y Humedad

Relativa. Nuevas capas podrían agregarse a la IDE e inmediatamente quedar accesibles por el servicio r_what.

Figura 6.4: Web ISAGRO (http://www.isagro.org.ar)

Tabla 6.2: Resumen de características de la IDE

Tema	Descripción
URL de acceso al WebGIS	http://webgis.isagro.org.ar/geoexplorer
URL de acceso al ISAGRO	http://www.isagro.org.ar
Tecnología utilizada	GRASS GIS, Geoserver, pyGDAL, Apache Tomcat, Apache Web Server, Openlayers, GeoExt, GXP, ExtJS, ZOO-Proyect, RestFull
Capacidades WPS	Servicio(geoproceso) r_what
Cantidad de capas publicadas	1350($raster + vectores$)
Geodatos	Accedidos por WCS y WFS
Características del servidor físico	2 procesadores XEON, 32GB de RAM, 12 discos de 2TB c/u, RAID5, VirtualMachine's

6.6. Ventanas de la IDE

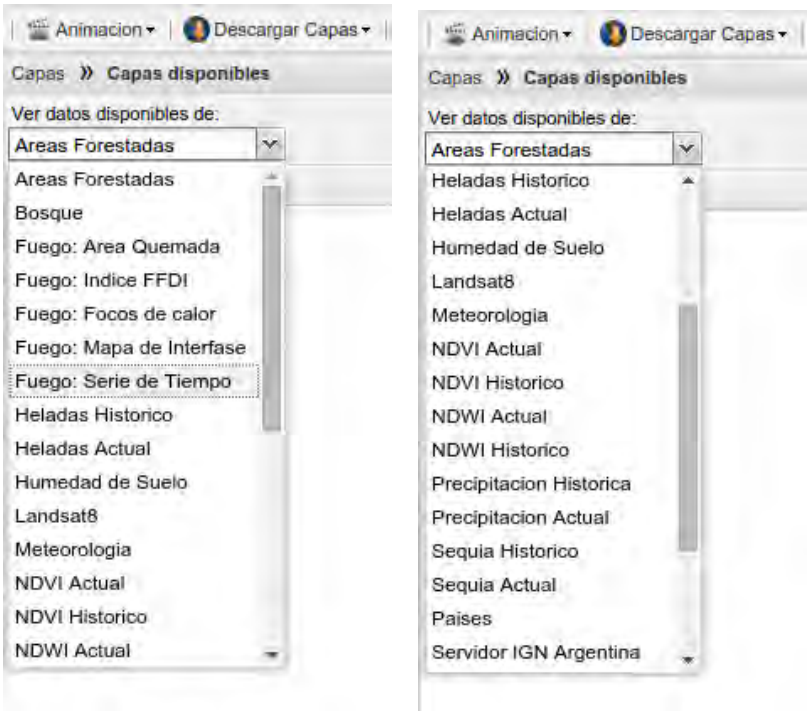


Figura 6.5: Listado de prodcutos: capas raster y vectoriales

Figura 6.6: Documentación de los productos publicados y sobre el proyecto

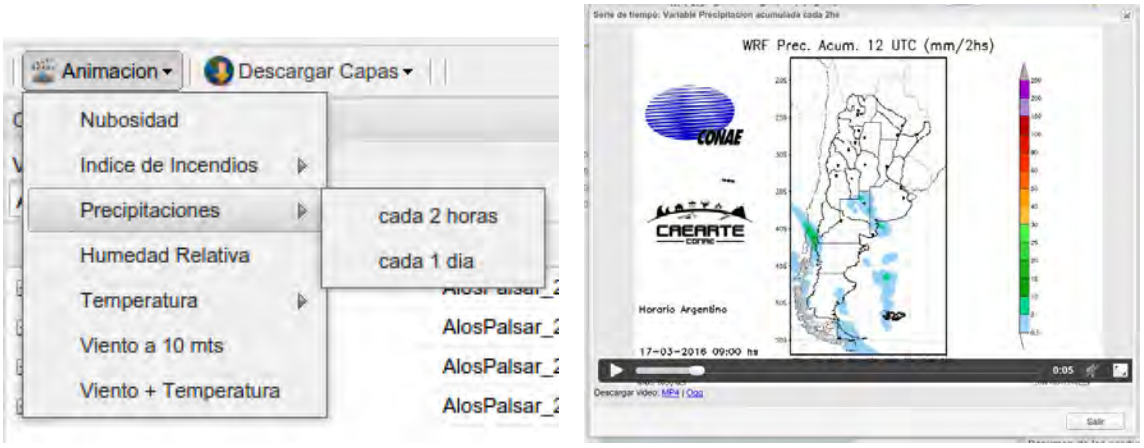


Figura 6.7: Funcionalidad: Animación de variables Meteorológicas

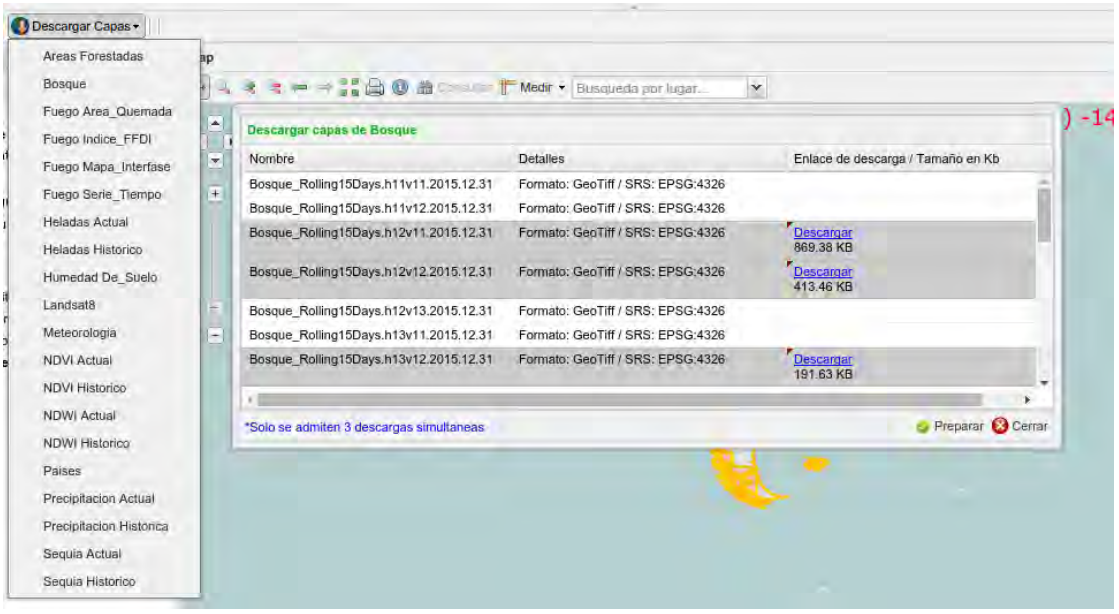


Figura 6.8: Funcionalidad: Descargar Capas

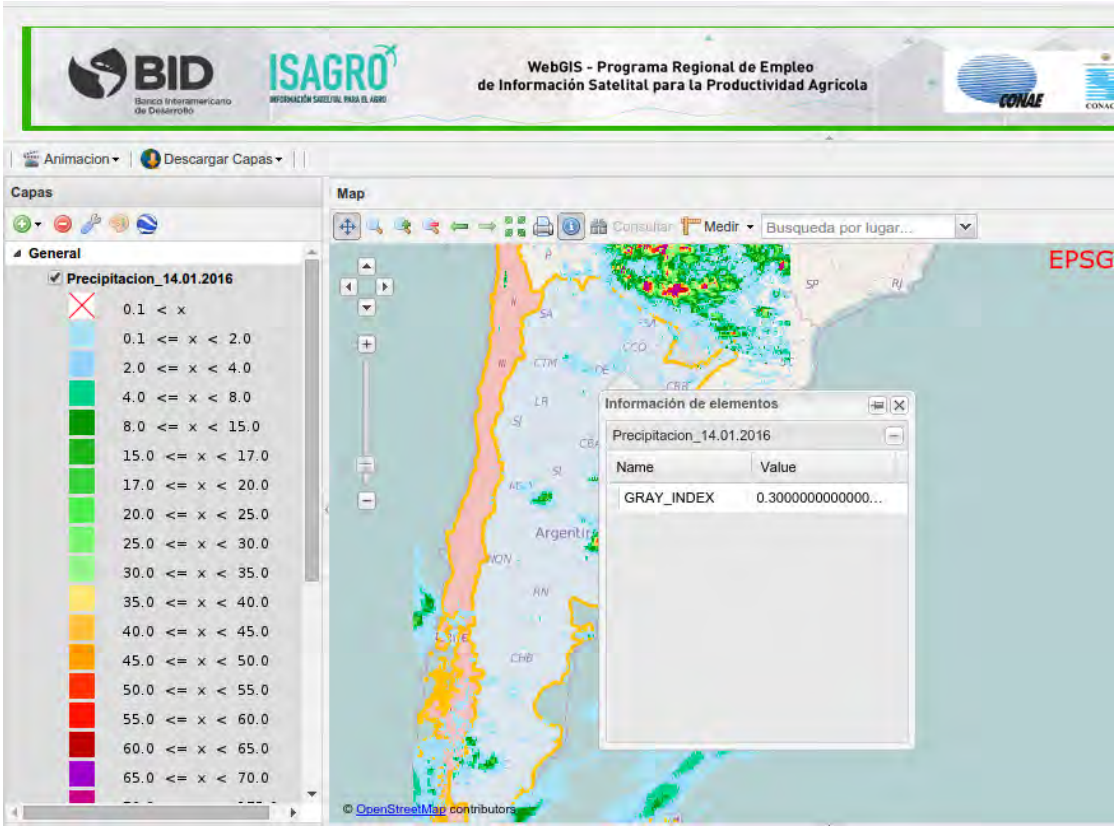


Figura 6.9: Funcionalidad: Consultar un pixel

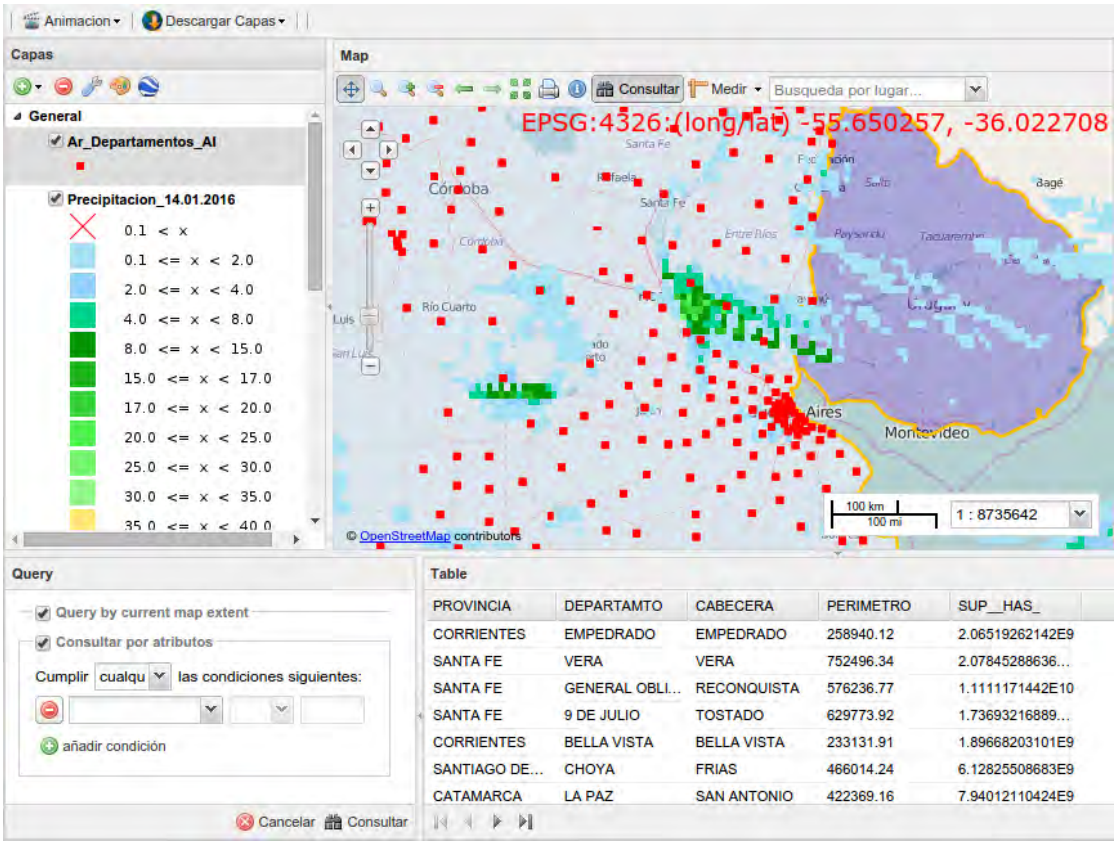


Figura 6.10: Funcionalidad: Consultar atributos vectoriales

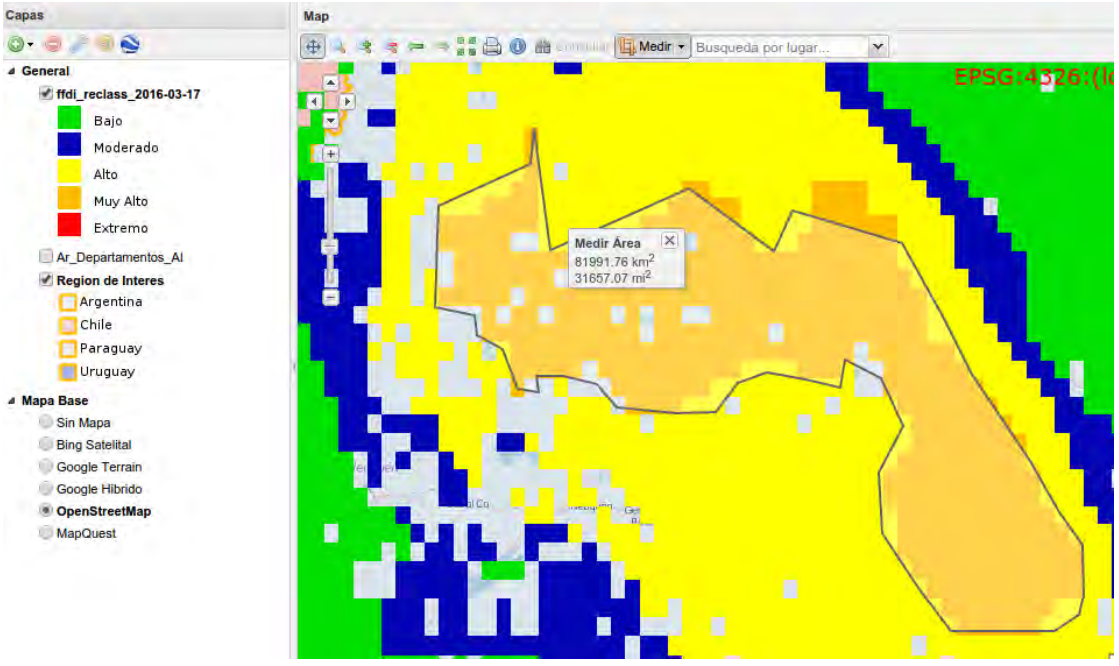
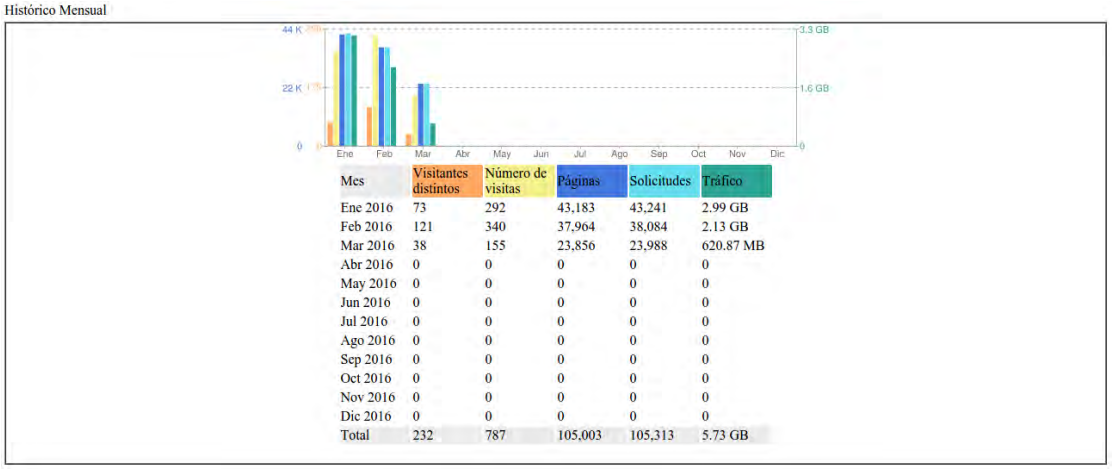


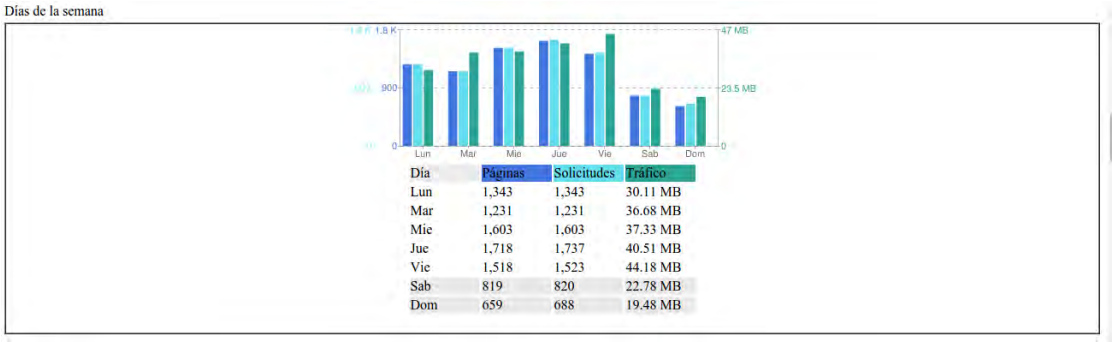
Figura 6.11: Funcionalidad: Midiendo area con riesgo de incendio categorizado como Muy Alto

Figura 6.12: Funcionalidad: Estadísticas de acceso a la IDE



Mes	Visitantes distintos	Número de visitas	Páginas	Solicitudes	Tráfico
Ene 2016	73	292	43,183	43,241	2.99 GB
Feb 2016	121	340	37,964	38,084	2.13 GB
Mar 2016	38	155	23,856	23,988	620.87 MB
Abr 2016	0	0	0	0	0
May 2016	0	0	0	0	0
Jun 2016	0	0	0	0	0
Jul 2016	0	0	0	0	0
Ago 2016	0	0	0	0	0
Sep 2016	0	0	0	0	0
Oct 2016	0	0	0	0	0
Nov 2016	0	0	0	0	0
Dic 2016	0	0	0	0	0
Total	232	787	105,003	105,313	5.73 GB

Figura 6.13: Funcionalidad: Estadísticas de acceso a la IDE - Histórico Mensual



Día	Páginas	Solicitudes	Tráfico
Lun	1,343	1,343	30.11 MB
Mar	1,231	1,231	36.68 MB
Mie	1,603	1,603	37.33 MB
Jue	1,718	1,737	40.51 MB
Vie	1,518	1,523	44.18 MB
Sab	819	820	22.78 MB
Dom	659	688	19.48 MB

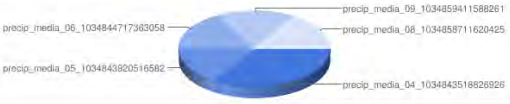
Figura 6.14: Funcionalidad: Estadísticas de acceso a la IDE - Días de la semana

Visitas por Dominios/Países (Top 10) - Lista completa

	Dominios/Países		Páginas	Solicitudes	Tráfico	
?	Desconocido	ip	13,764	13,782	529.49 MB	
	Argentina	ar	8,966	9,079	74.56 MB	
	Uruguay	uy	597	597	5.62 MB	
	Paraguay	py	489	490	10.17 MB	
	Network	net	40	40	1.03 MB	
	Otros		0	0	0	

Figura 6.15: Funcionalidad: Estadísticas de acceso a la IDE - Mapa de accesos

Downloads (Top 10) - Lista completa



Downloads: 20	Solicitudes	206 Solicitudes	Tráfico	Tamaño medio
/geoserver/temp/wcs/precip_media_04_1034843518826926 7.zip	2	0	159.53 KB	79.77 KB
/geoserver/temp/wcs/precip_media_05_1034843920516582 1.zip	1	0	80.00 KB	80.00 KB
/geoserver/temp/wcs/precip_media_06_1034844717363058 8.zip	1	0	76.12 KB	76.12 KB
/geoserver/temp/wcs/precip_media_09_1034859411588261 8.zip	1	0	77.02 KB	77.02 KB
/geoserver/temp/wcs/precip_media_08_1034858711620425 1.zip	1	0	75.12 KB	75.12 KB
/geoserver/temp/wcs/Anomalia_NDDI_03.03.2016_h12v12_103623813175...	1	0	56.67 MB	56.67 MB
/geoserver/temp/wcs/precip_media_10_1034870186707184 9.zip	1	0	77.28 KB	77.28 KB
/geoserver/temp/wcs/HsSMOS_Regional_10.03.2016_1114759306691045 8...	1	0	501.62 KB	501.62 KB
/geoserver/temp/wcs/HsSMOS_Regional_10.02.2016_1114759148047978 4...	1	0	105.05 KB	105.05 KB
/geoserver/temp/wcs/precip_media_03_1034826602740551 4.zip	1	0	80.97 KB	80.97 KB

Figura 6.16: Funcionalidad: Estadísticas de acceso a la IDE - Gráfico de productos descargados

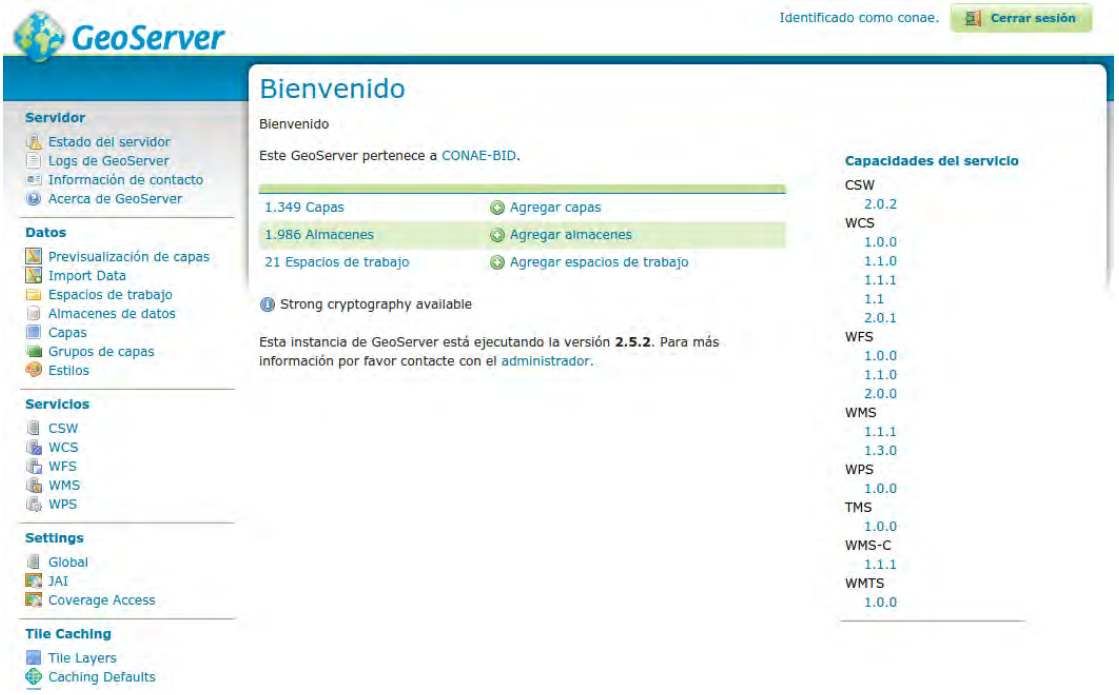


Figura 6.17: Acceso al servidor de mapas: GeoServer

Capítulo 7

Conclusiones y Trabajos futuros

7.1. Conclusiones

Se describió un procedimiento general que permite construir sistemas con características de IDE para proporcionar geoprocesamiento en la web como servicios. Dicha metodología se probó construyendo un sistema que se compone de: (a) Geoserver como servidor de mapas, (b) ZOO-Proyect como servidor WPS, (c) GRASS GIS y pyGDAL como software GIS y de procesamiento y (d) un cliente web utilizando Geoexplorer y el framework GXP conjuntamente con librerías Openlayers, GeoEXT y ExtJS. Se implementaron dos servicios a modo de ejemplo, uno que trabaja sobre datos raster, r_nbr y otro sobre datos vectoriales, *Buffer*. Las funcionalidades de acceso a los geoservicios se incorporaron a Geoexplorer como una serie de plugins.
El sistema es integramente compatible con los estándares actuales para almacenar, presentar, publicar e invocar datos y procesos como servicios. Los estándares geoespaciales utilizados son principalmente WMS, WCS, WFS y WPS. La arquitectura que lo rige, es cliente-servidor, lográndose también la incorporación de componentes relacionados con una arquitectura orientada a servicios como son los *Web Service* mediante WPS de ZOO-Proyect. Se destaca que el empleo del protocolo WPS constituye una forma novedosa de procesamiento de geodatos desde la web, con los beneficios del web service que se especifican seguidamente:

- Fácil acceso a funcionalidades SIG: el usuario accede a través de un navegador a funcionalidades de un SIG de escritorio.
- Publicación de servicios SIG: muchos usuarios accediendo a los servicios SIG definidos e implementados una sola vez.
- Interoperabilidad: incorporando los estándares de la OGC, se logra que los productos y procesos disponibles, funcionen en otras IDEs existentes o futuras. Del mismo modo, los

productos de otras IDEs (que cumplen con los estándares de la OGC) funcionan en la IDE construida.

- Sin necesidad de instalar nada en la PC personal: toda funcionalidad es gestionada por el navegador a través de los protocolos de la Internet.
- Integración de diversos programas de análisis y procesamiento de imágenes: permiten la generación de algoritmos y funcionalidades implementados en sus propios lenguajes y poniéndolos a disposición del usuario final SIG a través de un lenguaje en común.
- Generación de productos a demanda: cruzando capas disponibles en diversos servidores de mapas a través de protocolos WMS, WFS y WCS.
- Creación de nodos para conformar una red de servicios SIG y de procesamiento.

El software utilizado, los estándares elegidos, la metodología de trabajo y la experiencia adquirida en la construcción de este sistema son descriptos y documentados en la tesis. Se considera, que la manera en que ha sido planteado este trabajo, proporciona el conocimiento (*know-how*) para la construcción de sistemas similares. Su potencial aplicabilidad futura abarca proyectos que incluyen la teledetección como fuente informativa, destacándose los relacionados con emergencias ambientales y epidemiológicas, entre otros.
El hecho de haberse trabajado paralelamente en la aplicación de este recurso en el proyecto PRESIPA, que tiene aplicación en el ámbito agropecuario, implicó poner a disposición un conjunto de productos satelitales a través de la IDE construida a tales efectos y la funcionalidad r_what, ambos accesibles en forma de servicios, lográndose la interoperabilidad planteada originalmente. Como resultado de esta aplicación paralela, el sistema fue integrado como nodo a la red de IDEs mundiales. Destacamos además, que la incorporación de procesos como servicios, habilita a que los desarrolladores de web mapping nutran sus interfaces haciendo uso de los servicios publicados.
Se han logrado incorporar aprendizajes relacionados con requerimientos reales específicos, destacándose el trabajo con un conjunto de programas facilitados por la OSGeo para la construcción de una solución de cómputo en nube (cloud computing).
La combinación de los programas y herramientas citados y la experiencia sumada con el empleo de protocolos y estándares geoespaciales construyen un conjunto que se considera valioso per-se y transferible.

7.2. Trabajos Futuros

7.2.1. Control de cuota de disco y recursos de procesamiento en el servidor

La incorporación de geoservicios que requieren mucho tiempo de cálculo (procesos asíncronos) como así tambíen de espacio de almacenamiento, se convierte en un problema si el entorno del

servidor no es controlado. Por ejemplo, consideremos el caso del servicio r_nbr: Este servicio se aplica sobre dos bandas landsat8 de aproximadamente 100MB cada una. Ambas imágenes estan almacenadas en el servidor de geodatos. Cuando el servicio es ejecutado, ZOO-Proyect invoca mediante el protocolo WCS las bandas que son *inputs* del cálculo, ocupando adicionalmente 200MB (temporario) mas de disco hasta que el cálculo finaliza. El resultado del cálculo son otros 100MB en disco.
Por otro lado, el cálculo del índice utiliza el módulo r.mapcalc de GRASS GIS, el cual se ejecuta concurrentemente y ejecutando el proceso en un cpu core i3 con 2GB de RAM, el resultado del cálculo llega a los 3.5 min aproximadamente, sin ejecutar otros procesos.
Ahora, suponiendo que tenemos varios usuarios realizando estas peticiones simultaneamente tendríamos un problema de saturación de recursos.
Para dar solución a situaciones como la descripta aquí, el sistema debería escalar a una solución que mejore por un lado, el tiempo de acceso a los datos mediante el uso de bases de datos espaciales (por ejemplo, PostGIS sobre datos raster y vectoriales) y por otro lado, el uso de clusters o supercomputadores para agilizar el tiempo de computo, principalmente útil cuando se requiera utilizar la solución de geoprocesamiento en sistemas de alerta y respuesta temprana.
También, se hace necesario implementar un sistema de autenticación por usuario y asociar a cada sesión una cuota de disco en el servidor y una cantidad de solicitudes de procesamiento por lapsos de tiempo, es decir, definir concurrencia y máxima cantidad de recursos.
La implementación de una solución como la descripta aquí, permitiría escalar la solución de procesamiento de tal forma que es posible pensar en llevar un software GIS y de procesamiento, completamente a la web. Por lo cual, estaríamos mejorando el rendimiento y los beneficios para el usuario GIS proveyendo de un conjunto mas amplio de servicios de procesamiento sobre datos espaciales.

7.2.2. Desarrollo de nuevos servicios

La incorporación de nuevos servicios de procesamiento al servidor es una tarea que se debe satisfacer cada vez que haya un nuevo requerimiento de usuario. La implementación de los servicios puede ser incorporada por GRASS GIS o pyGDAL como se muestra en este trabajo, pero también es posible incorporar otros software GIS y de procesamiento que son soportados por ZOO-Proyect como CGAL, OTB y SAGA GIS. Para lograr esto, se deberán generar las instalaciones de dichos software antes mencionados en el servidor, la instalación de los software (específicos de cada GIS) que permiten construir los servicios para ZOO-Proyect y la generación de los servicios para las funcionalidades que se deseen proveer.

7.2.3. Desarrollo del lado del cliente

En general, la incorporación de nuevos servicios de procesamiento y GIS sobre el servidor requiere el desarrollo de funcionalidades del lado del cliente. Si bien el formulario de descripción de los servicios que se implementó es lo suficientemente general, es útil para casos donde el usuario parametriza las entradas mediante el rellenado de campos y selección de capas

desde el panel del webgis. Ahora bien, la incorporación de funcionalidades como por ejemplo, recorte de capas, consulta del valor de un pixel sobre datos históricos, cálculo de variables ambientales sobre un polígono dado, etc. requerirán del desarrollo de herramientas en la interfaz del cliente (dibujar polígonos, seleccionar areas, elegír un punto sobre el mapa, etc) para la interacción del usuario con el mapa, tal como se mostró en el caso del buffer.

7.2.4. Interfaz de usuario gráfica (Plugin) para el servicio r_what

En el capítulo 6, sección **Uso del WPS: servicio r_what**, se implementó el servicio r_what en el servidor, pero no se implementó el acceso desde el cliente. Una tarea que queda pendiente es el desarrollo de un *plugin* en el framework GXP del SDK de Boundless, para interactuar con el servicio. Este *plugin* luego debe ser incorporado al cliente geoexplorer.

7.2.5. Acceso a capas raster y vectoriales remotas

El cliente WPS implementado en este trabajo, solo accede a capas raster y vectoriales (mediante WCS y WFS) almacenadas en el servidor local. Un desarrollo útil es modificar el cliente para traer datos desde otros servidores de mapas disponibles en la web y usarlos como entradas de los servicios implementados, parametrizando las URL en el objeto *Reference* de las llamadas *Execute* (en el protocolo WPS). Sin embargo, se requerirá para tal solución los chequeos necesarios para saber a priori si los servidores remotos proveen los datos espaciales mediante WCS y WFS.

Apéndice A

Anexo A

A.1. Códigos desarrollados y documentos XML

```
ZOO Kernel MIT/X-11 License
Copyright     2010-2011 GeoLabs
Permission is hereby granted, free of charge, to any person obtaining a copy of this
software and associated documentation files (the "Software"), to deal in the Software
without restriction, including without limitation the rights to use, copy, modify, merge,
publish, distribute,sublicense, and/or sell copies of the Software, and to permit persons
to whom the Software is furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all copies or
substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM,OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```

Código A.1: Licencia ZOO-proyect

Bandas espectrales del sensor OLI	Longitud de onda	Resolución
Banda 1 (Costero-Aerosol)	0.433-0.453 μm	30 m.
Banda 2 (Azul)	0.450-0.515 μm	30 m.
Banda 3 (Verde)	0.525-0.600 μm	30 m.
Banda 4 (Rojo)	0.630-0.680 μm	30 m.
Banda 5 (Infrarrojo cercano- NIR)	0.845-0.885 μm	30 m.
Banda 6 (Infrarrojo de onda corta - SWIR1)	1.560-1.660 μm	30 m.
Banda 7 (Infrarrojo de onda corta - SWIR2)	2.100-2.300 μm	30 m.
Banda 8 (Pancromática)	0.500-0.680 μm	15 m.
Banda 9 (Cirrus)	1.360-1.390 μm	30 m.
Banda 10 (Infrarrojo termal o de onda larga)	10.30-11.30 μm	100 m.
Banda 11 (Infrarrojo termal o de onda larga)	11.50-12.50 μm	100 m.

Figura A.1: Características espectrales de los sensores OLI y TIRS de Landsat 8 (Solorza et al. (2014))

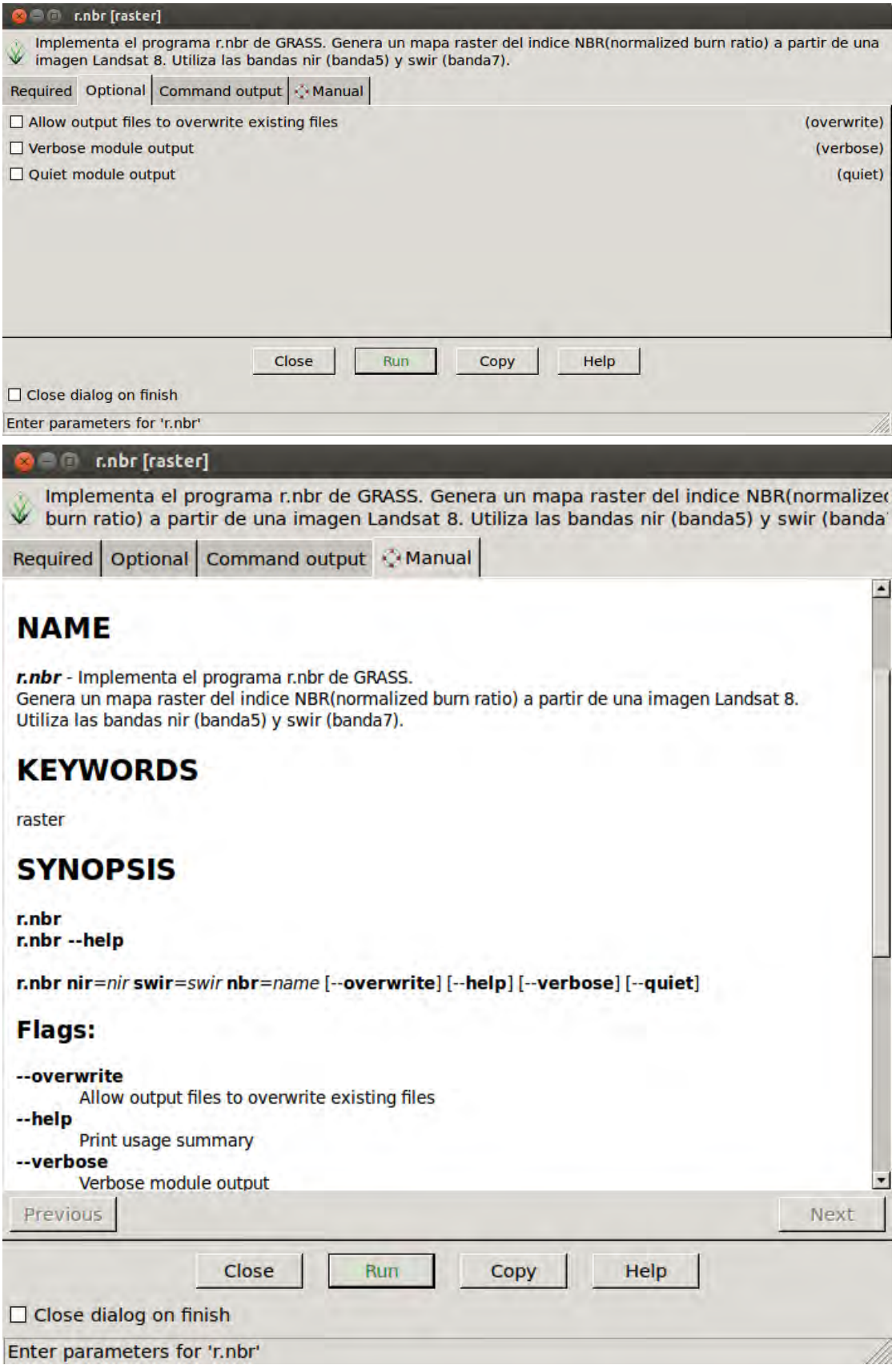


Figura 2: Comando r.nbr incorporado a GRASS GIS

```
        import osgeo.ogr
        import libxml2
        def extractInputs(obj):
                if obj["mimeType"]=="application/json":
                        return osgeo.ogr.CreateGeometryFromJson(obj["value"])
            else:
                    return createGeometryFromWFS(obj["value"])
            return null

        def outputResult(obj,geom):
                try:
                        if obj["mimeType"]=="application/json":
                        obj["value"]=geom.ExportToJson()
                        obj["mimeType"]="application/json"
                else:
                        obj["value"]=geom.ExportToGML()
            except:
                            obj["value"]=geom.ExportToGML()

        def Buffer(conf,inputs,outputs):
                geometry=extractInputs(inputs["InputPolygon"])
            try:
                    bdist=int(inputs["BufferDistance"]["value"])
                except:
                bdist=10
                rgeom=geometry.Buffer(bdist)
            outputResult(outputs["Result"],rgeom)
            geometry.Destroy()
            rgeom.Destroy()
            return 3
```

Código 2: Código que implementa el Servicio Buffer (Web Site Oficial ZOO-Proyect)

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ProcessDescriptions xmlns:wps="http://www.opengis.net/wps/1.0.0"
xmlns:ows="http://www.opengis.net/ows/1.1"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
 http://schemas.opengis.net/wps/1.0.0/wpsDescribeProcess_response.xsd"
 service="WPS" version="1.0.0" xml:lang="en-US">
<ProcessDescription wps:processVersion="1" storeSupported="true"
    statusSupported="true">
<ows:Identifier>r.nbr</ows:Identifier>
<ows:Title>Genera un mapa raster del indice NBR(normalized burn ratio) a
    partir de una
imagen Landsat 8. Utiliza las bandas nir (banda5) y swir (banda7).</ows:Title
    >
<ows:Abstract>http://grass.osgeo.org/grass71/manuals/r.nbr.html</ows:Abstract
    >
<ows:Metadata xlink:title="raster" />
<DataInputs>
<Input minOccurs="1" maxOccurs="1">
<ows:Identifier>nir</ows:Identifier>
<ows:Title>Nombre del raster de entrada que corresponde a la Banda del NIR</
    ows:Title>
<LiteralData>
<ows:DataType ows:reference="xs:string">string</ows:DataType>
<ows:AnyValue/>
</LiteralData>
```

```
</Input>
<Input minOccurs="1" maxOccurs="1">
<ows:Identifier>swir</ows:Identifier>
<ows:Title>Nombre del raster de entrada que corresponde a la Banda del SWIR</ows:Title>
<LiteralData>
<ows:DataType ows:reference="xs:string">string</ows:DataType>
<ows:AnyValue/>
</LiteralData>
</Input>
<Input minOccurs="0" maxOccurs="1">
<ows:Identifier>grass_resolution_ns</ows:Identifier>
<ows:Title>Resolution of the mapset in north-south direction in meters or degrees
</ows:Title>
<ows:Abstract>This parameter defines the north-south resolution of the mapset in meter
or degrees, which should be used to process the input and output raster data. To enable
this setting, you need to specify north-south and east-west resolution.</ows:Abstract>
<LiteralData>
<ows:DataType ows:reference="xs:float">float</ows:DataType>
<UOMs>
<Default>
<ows:UOM>meters</ows:UOM>
</Default>
<Supported>
<ows:UOM>meters</ows:UOM>
<ows:UOM>degrees</ows:UOM>
</Supported>
</UOMs>
<ows:AnyValue/>
</LiteralData>
</Input>
<Input minOccurs="0" maxOccurs="1">
<ows:Identifier>grass_resolution_ew</ows:Identifier>
<ows:Title>Resolution of the mapset in east-west direction in meters or degrees</ows:Title>
<ows:Abstract>This parameter defines the east-west resolution of the mapset in meters or
degrees, which should be used to process the input and output raster data. To enable this
setting, you need to specify north-south and east-west resolution.</ows:Abstract>
<LiteralData>
<ows:DataType ows:reference="xs:float">float</ows:DataType>
<UOMs>
<Default>
<ows:UOM>meters</ows:UOM>
</Default>
<Supported>
<ows:UOM>meters</ows:UOM>
<ows:UOM>degrees</ows:UOM>
</Supported>
</UOMs>
```

```
<ows:AnyValue/>
</LiteralData>
</Input>
</DataInputs>
<ProcessOutputs>
<Output>
<ows:Identifier>nbr</ows:Identifier>
<ows:Title>Nombre del mapa raster de salida que describe el NBR</ows:Title>
<ComplexOutput>
<Default>
<Format>
<MimeType>image/tiff</MimeType>
</Format>
</Default>
<Supported>
<Format>
<MimeType>image/tiff</MimeType>
</Format>
<Format>
<MimeType>image/geotiff</MimeType>
</Format>
<Format>
<MimeType>application/geotiff</MimeType>
</Format>
<Format>
<MimeType>application/x-geotiff</MimeType>
</Format>
<Format>
<MimeType>application/x-erdas-hfa</MimeType>
</Format>
<Format>
<MimeType>application/netcdf</MimeType>
</Format>
<Format>
<MimeType>application/x-netcdf</MimeType>
</Format>
</Supported>
</ComplexOutput>
</Output>
</ProcessOutputs>
</ProcessDescription>
</wps:ProcessDescriptions>
```

Código 3: XML DescribeProcess de r_nbr

```
import osgeo.ogr
import libxml2

def extractInputs(obj):
    if obj["mimeType"]=="application/json":
        return osgeo.ogr.CreateGeometryFromJson(obj["value"])
    else:
        return createGeometryFromWFS(obj["value"])
    return null

def outputResult(obj,geom):
```

```
    try:
        if obj["mimeType"]=="application/json":
            obj["value"]=geom.ExportToJson()
            #obj["mimeType"]="text/plain"
     obj["mimeType"]="application/json"
        else:
            obj["value"]=geom.ExportToGML()
    except:
        obj["value"]=geom.ExportToGML()

def Buffer(conf,inputs,outputs):
    geometry=extractInputs(inputs["InputPolygon"])
    try:
        bdist=int(inputs["BufferDistance"]["value"])
    except:
        bdist=10
    rgeom=geometry.Buffer(bdist)
    outputResult(outputs["Result"],rgeom)
    geometry.Destroy()
    rgeom.Destroy()
    return 3
```

Código 4: Servicio ZOO buffer

```
[r_what]
Title = Queries raster maps on their category values and category labels.
Abstract = http://grass.osgeo.org/grass71/manuals/r.what.html
processVersion = 2
storeSupported = true
statusSupported = true
serviceProvider = r_what
serviceType = Python
<DataInputs>
  [map]
  Title = Name of existing raster maps to query
  Abstract =
  minOccurs = 1
  maxOccurs = 1024
 <ComplexData>
   <Default>
     mimeType = image/tiff
   </Default>
   <Supported>
     mimeType = image/tiff
   </Supported>
   <Supported>
     mimeType = image/geotiff
   </Supported>
   <Supported>
     mimeType = application/geotiff
   </Supported>
   <Supported>
     mimeType = application/x-geotiff
   </Supported>
   <Supported>
     mimeType = image/png
```

```
33      </Supported>
34      <Supported>
35        mimeType = image/gif
36      </Supported>
37      <Supported>
38        mimeType = image/jpeg
39      </Supported>
40      <Supported>
41        mimeType = application/x-erdas-hfa
42      </Supported>
43      <Supported>
44        mimeType = application/netcdf
45      </Supported>
46      <Supported>
47        mimeType = application/x-netcdf
48      </Supported>
49    </ComplexData>
50    [coordinates]
51    Title = Coordinates for query
52    Abstract =
53    minOccurs = 0
54    maxOccurs = 2
55    <LiteralData>
56      DataType    = float
57      <Default>
58      </Default>
59    </LiteralData>
60    [points]
61    Title = Name of vector points map for query
62    Abstract = Or data source for direct OGR access
63    minOccurs = 0
64    maxOccurs = 1
65    <ComplexData>
66      <Default>
67        mimeType = text/xml
68        encoding = UTF-8
69        schema = http://schemas.opengis.net/gml/3.1.1/base/gml.xsd
70      </Default>
71      <Supported>
72        mimeType = text/xml
73        encoding = UTF-8
74        schema = http://schemas.opengis.net/gml/3.1.1/base/gml.xsd
75      </Supported>
76      <Supported>
77        mimeType = application/xml
78        encoding = UTF-8
79        schema = http://schemas.opengis.net/gml/3.1.1/base/gml.xsd
80      </Supported>
81      <Supported>
82        mimeType = text/xml
83        encoding = UTF-8
84        schema = http://schemas.opengis.net/gml/2.1.2/feature.xsd
85      </Supported>
86      <Supported>
87        mimeType = application/xml
88        encoding = UTF-8
```

```
89        schema = http://schemas.opengis.net/gml/2.1.2/feature.xsd
90      </Supported>
91      <Supported>
92        mimeType = text/xml
93        encoding = UTF-8
94        schema = http://schemas.opengis.net/kml/2.2.0/ogckml22.xsd
95      </Supported>
96      <Supported>
97        mimeType = application/dgn
98      </Supported>
99      <Supported>
100       mimeType = application/shp
101     </Supported>
102     <Supported>
103       mimeType = application/x-zipped-shp
104     </Supported>
105   </ComplexData>
106   [null]
107   Title = String to represent no data cell
108   Abstract =
109   minOccurs = 0
110   maxOccurs = 1
111   <LiteralData>
112     DataType   = string
113     <Default>
114       value = *
115     </Default>
116   </LiteralData>
117   [separator]
118   Title = Field separator
119   Abstract = Special characters: pipe, comma, space, tab, newline
120   minOccurs = 0
121   maxOccurs = 1
122   <LiteralData>
123     DataType   = string
124     <Default>
125       value = pipe
126     </Default>
127   </LiteralData>
128   [cache]
129   Title = Size of point cache
130   Abstract =
131   minOccurs = 0
132   maxOccurs = 1
133   <LiteralData>
134     DataType   = integer
135     <Default>
136       value = 500
137     </Default>
138   </LiteralData>
139   [-n]
140   Title = Output header row
141   Abstract =
142   minOccurs = 0
143   maxOccurs = 1
144   <LiteralData>
```

```
145     DataType    = boolean
146     <Default>
147       value = false
148     </Default>
149   </LiteralData>
150   [-f]
151   Title = Show the category labels of the grid cells
152   Abstract =
153   minOccurs = 0
154   maxOccurs = 1
155   <LiteralData>
156     DataType    = boolean
157     <Default>
158       value = false
159     </Default>
160   </LiteralData>
161   [-r]
162   Title = Output color values as RRR:GGG:BBB
163   Abstract =
164   minOccurs = 0
165   maxOccurs = 1
166   <LiteralData>
167     DataType    = boolean
168     <Default>
169       value = false
170     </Default>
171   </LiteralData>
172   [-i]
173   Title = Output integer category values, not cell values
174   Abstract =
175   minOccurs = 0
176   maxOccurs = 1
177   <LiteralData>
178     DataType    = boolean
179     <Default>
180       value = false
181     </Default>
182   </LiteralData>
183   [-c]
184   Title = Turn on cache reporting
185   Abstract =
186   minOccurs = 0
187   maxOccurs = 1
188   <LiteralData>
189     DataType    = boolean
190     <Default>
191       value = false
192     </Default>
193   </LiteralData>
194   [grass_resolution_ns]
195   Title = Resolution of the mapset in north-south direction in meters or
        degrees
196   Abstract = This parameter defines the north-south resolution of the mapset
        in
197   meter or degrees, which should be used to process the input and output
        raster
```

```
198   data. To enable this setting, you need to specify north-south and east-west
199   resolution.
200   minOccurs = 0
201   maxOccurs = 1
202   <LiteralData>
203     DataType    = float
204     <Default>
205     </Default>
206   </LiteralData>
207   [grass_resolution_ew]
208   Title = Resolution of the mapset in east-west direction in meters or
        degrees
209   Abstract = This parameter defines the east-west resolution of the mapset in
         meters
210   or degrees, which should be used to process the input and output raster
        data.  To
211   enable this setting, you need to specify north-south and east-west
        resolution.
212   minOccurs = 0
213   maxOccurs = 1
214   <LiteralData>
215     DataType    = float
216     <Default>
217     </Default>
218   </LiteralData>
219   [grass_band_number]
220   Title = Band to select for processing default is all bands
221   Abstract = This parameter defines band number of the input raster files
        which should
222   be processed. As default all bands are processed and used as single and
        multiple
223   inputs for raster modules.
224   minOccurs = 0
225   maxOccurs = 1
226   <LiteralData>
227     DataType    = integer
228     <Default>
229     </Default>
230   </LiteralData>
231 </DataInputs>
232 <DataOutputs>
233   [output]
234   Title = Name for output file if omitted or "-" output to stdout
235   Abstract =
236   <ComplexData>
237     <Default>
238       mimeType = text/plain
239     </Default>
240     <Supported>
241       mimeType = text/plain
242     </Supported>
243   </ComplexData>
244 </DataOutputs>
```

Código 5: Documento ZCFG del proceso r_what

```
[BufferPy]
Title = Compute Buffer.
Abstract = Returns the buffer of the geometry on which the method is invoked.
processVersion = 1
storeSupported = true
statusSupported = true
serviceProvider=ogr_sp
serviceType=Python
<MetaData>
    Title = Demo
</MetaData>
 <DataInputs>
[InputPolygon]
  Title = Polygon to compute boundary
  Abstract = URI to a set of GML that describes the polygon.
  minOccurs = 1
  maxOccurs = 1
  <MetaData>
      Test = My test
  </MetaData>
<ComplexData>
 <Default>
   mimeType = text/xml
   encoding = UTF-8
 </Default>
 <Supported>
   mimeType = application/json
   encoding = UTF-8
 </Supported>
</ComplexData>
[BufferDistance]
 Title = Buffer Distance
 Abstract = Distance to be used to calculate buffer.
 minOccurs = 0
 maxOccurs = 1
 <LiteralData>
  DataType = float
  <Default>
   uom = degree
   value = 10
  </Default>
  <Supported>
   uom = meter
  </Supported>
 </LiteralData>
</DataInputs>
 <DataOutputs>
[Result]
 Title = The created geometry
 Abstract = The geometry containing the boundary of the geometry on which the
     method
 was invoked.
 <MetaData>
   Title = Result
 </MetaData>
 <ComplexData>
```

```
56    <Default>
57      mimeType = application/json
58      encoding = UTF-8
59    </Default>
60    <Supported>
61      mimeType = text/xml
62      encoding = UTF-8
63    </Supported>
64    <Supported>
65      mimeType = image/png
66      useMapServer = true
67    </Supported>
68  </ComplexData>
69  </DataOutputs>
```

Código 6: Documento ZCFG del proceso Buffer

```
1  [r_nbr]
2  Title = Genera un mapa raster del indice NBRnormalized burn ratio a partir de
       una imagen
3  Landsat 8. Utiliza las bandas nir banda5 y swir banda7.
4  Abstract = http://grass.osgeo.org/grass71/manuals/r.nbr.html
5  processVersion = 2
6  storeSupported = true
7  statusSupported = true
8  serviceProvider = r_nbr
9  serviceType = Python
10 <DataInputs>
11   [nir]
12   Title = Nombre del raster de entrada que corresponde a la Banda del NIR
13   Abstract =
14   minOccurs = 1
15   maxOccurs = 1
16   <LiteralData>
17     DataType    = string
18     <Default>
19     </Default>
20   </LiteralData>
21   [swir]
22   Title = Nombre del raster de entrada que corresponde a la Banda del SWIR
23   Abstract =
24   minOccurs = 1
25   maxOccurs = 1
26   <LiteralData>
27     DataType    = string
28     <Default>
29     </Default>
30   </LiteralData>
31   [grass_resolution_ns]
32   Title = Resolution of the mapset in north-south direction in meters or
       degrees
33   Abstract = This parameter defines the north-south resolution of the mapset
       in meter
34   or degrees, which should be used to process the input and o utput raster
       data. To
```

```
35    enable this setting, you need to specify north-south and east-west
        resolution.
36    minOccurs = 0
37    maxOccurs = 1
38    <LiteralData>
39      DataType    = float
40      <Default>
41  </Default>
42    </LiteralData>
43    [grass_resolution_ew]
44    Title = Resolution of the mapset in east-west direction in meters or
        degrees
45    Abstract = This parameter defines the east-west resolution of the mapset in
         meters
46    or degrees, which should be used to process the input and output raster
        data.
47    To enable this setting, you need to specify north-south and east-west
        resolution.
48    minOccurs = 0
49    maxOccurs = 1
50    <LiteralData>
51      DataType    = float
52      <Default>
53      </Default>
54    </LiteralData>
55  </DataInputs>
56  <DataOutputs>
57    [nbr]
58    Title = Nombre del mapa raster de salida que describe el NBR
59    Abstract =
60    <ComplexData>
61      <Default>
62        mimeType = image/tiff
63      </Default>
64      <Supported>
65        mimeType = image/tiff
66      </Supported>
67      <Supported>
68        mimeType = image/geotiff
69      </Supported>
70      <Supported>
71        mimeType = application/geotiff
72      </Supported>
73      <Supported>
74        mimeType = application/x-geotiff
75      </Supported>
76      <Supported>
77        mimeType = application/x-erdas-hfa
78      </Supported>
79      <Supported>
80        mimeType = application/netcdf
81      </Supported>
82      <Supported>
83        mimeType = application/x-netcdf
84      </Supported>
85    </ComplexData>
```

```
</DataOutputs>
```

Código 7: Documento ZCFG del proceso r.nbr

```
########################################################
# This service was generated using wps-grass-bridge #
########################################################
import ZOOGrassModuleStarter as zoo
def r_nbr(m, inputs, outputs):
    service = zoo.ZOOGrassModuleStarter()
    service.fromMaps("r.nbr", inputs, outputs)
    return 3
```

Código 8: Servicio ZOO r_nbr

Bibliografía

52north (2015). 52°north geoprocessing community - professional open source geoprocessing. http://52north.org/communities/geoprocessing/wps/.

Agafonkin, V. (2015). Leaflet. http://leafletjs.com/.

BID (2014). Programa regional de empleo de información satelital para la productividad agricola. http://www.iadb.org/es/proyectos/project-information-page,1303.html?id=RG-T1820.

Bivand, R., Krug, R., Neteler, M., and Jeworutzki, S. (2009). Package rgrass7. https://cran.r-project.org/web/packages/rgrass7/rgrass7.pdf.

Boundless (2015). Geoserver in production. http://boundlessgeo.com/whitepaper/geoserver-production-2/.

BPR-BID (2016). Información satelital para el agro. https://www.isagro.org.ar.

Butler, H., Daly, M., Doyle, A., Gillies, S., Schaub, T., and Schmidt, C. (2008). The geojson format specification. http://geojson.org/geojson-spec.html.

CAEARTE-CONAE (2015). Manual descriptivo sobre la implementacion experimental del modelo numerico de prediccion del tiempo wrf y sus productos. http://webgis.isagro.org.ar/data/pdf/Meteorologia_CAEARTE-WRF-MAN-ESP-001.pdf.

Cepicky, J. (2015). Web site oficial pywps. http://pywps.org//.

Cepicky, J. and Becchi, L. (2006). Geospatial processing via internet on remote servers – pywps. *OSGeo Journal*.

Community OSGeoLive (2015). Welcome to osgeo-live 8.0 — osgeo-live 8.0 documentation. http://Live.osgeo.org.

Community Software Open Source (2015). Ossim suite geospatial. https://trac.osgeo.org/ossim/.

CONAE (2015a). Documento teórico de los productos de heladas del programa de bienes públicos regionales. http://webgis.isagro.org.ar/data/pdf/Heladas.pdf.

CONAE (2015b). Focos de calor a partir de modis. http://http://catalogos.conae.gov.ar/focos/Instructivo-Visualizacion-Focos-De-Calor.pdf.

CONAE (2015c). Precipitación actual mediante datos gpm. http://webgis.isagro.org.ar/data/pdf/Precipitacion_Actual.pdf.

CONAE (2015d). Procedimiento metodológico para la generación de mapa de interfase urbano-vegetaciÓn rural. http://webgis.isagro.org.ar/data/pdf/Mapa_de_Interface_Urbano-Vegetacion.pdf.

CONAE-MGAP (2015). Aplicacion del indice de sequia (nddi) para monitoreo y alerta con imagenes modis. http://webgis.isagro.org.ar/data/pdf/Indice_de_Sequia_NDDI_con_MODIS_dic2014.pdf.

Corrección WPS, O. (2009). Corrigendum for opengis implementation standard web processing service (wps) 1.0.0. Technical Report 08-091r6, Open Geospatial Consortium.

Eijnden, B. and Jansen, M. and Monnerat, M. (2015). Geoext mobile (gxm). http://geoext.github.io/GXM/.

ESA (2015a). Algorithm theoretical basis document (atbd) for the smos level 2 soil moisture. http://webgis.isagro.org.ar/data/pdf/Humedad_de_Suelo_SMOS_REGIONAL.pdf.

ESA (2015b). Soil moisture profile index in the roots region by scatterometer data assimilation. http://webgis.isagro.org.ar/data/pdf/Humedad_de_suelo_ASCAT_H14.pdf.

ESRI (2015). Web processing service. http://resources.arcgis.com/en/help/main/10.1/index.html#//015400000327000000.

GeoExt, C. S. (2015). Geoext. http://geoext.org/.

GeoServer (2015). Geoserver user manual. http://docs.geoserver.org/latest/en/user/.

Geoserver (2015). Web processing service. http://docs.geoserver.org/stable/en/user/extensions/wps/index.html.

Google Summer Code, P. (2015). Google summer code proyect. https://code.google.com/archive/p/wps-grass-bridge/.

GRASS Development Team (2015a). Grass gis 7 programmer's manual. http://grass.osgeo.org/programming7/.

GRASS Development Team (2015b). Grass gis python scripting with script package. http://grass.osgeo.org/grass71/manuals/libpython/script_intro.html.

GRASS GIS Development Team (2015a). Grass gis python parser_standard_options.c. http://grass.osgeo.org/programming7/parser__standard__options_8c_source.html.

GRASS GIS Development Team (2015b). Grass gis web site ofitial. https://grass.osgeo.org/.

GRASS GIS Development Team and third parties (2015). Grass gis addons. https://grasswiki.osgeo.org/wiki/AddOns.

Hazzard, E. (2011). Openlayers 2.10. Technical Report ISBN: 978-1-849514-12-5, Open Source Geospatial Foundation.

IDERA (2014). Infraestructura de datos espaciales de la república argentina. http://www.idera.gob.ar/.

INIA (2009). Gestón de riesgo agropecuario. Technical Report ISSN: 0717-4829, INIA.

Jason Karl (2015). Normalized burn ratio. http://wiki.landscapetoolbox.org/doku.php/remote_sensing_methods:normalized_burn_ratio.

JAXA (2015). Palsar mosaic and forest/nonforest map (2007-2010), japan aerospace exploration agency,. http://webgis.isagro.org.ar/data/pdf/Clasificacion_de_Bosques-AlosPalsar.pdf.

JS, S. E. (2015). Extjs. https://www.sencha.com/products/extjs/#overview.

Keen, M. and Coutinho, R. (2014). Developing web services applications. .

MapServer (2015a). About mapserver. http://mapserver.org/about.html#about.

MapServer (2015b). Introduction mapserver. http://mapserver.org/introduction.html.

Michaelis, C. D. and Ames, D. P. (2008). Evaluation and implementation of the ogc web processing service for use in client-side gis. *GeoInformatica*.

Neteler, M. and Mitasova, H. (2008). *Open Source GIS: A GRASS GIS Approach, 3rd edn. Springer, NY*. Springer, New York.

OGC and Google (2015). Keyhole markup language. https://developers.google.com/kml/documentation/kmlreference?hl=es.

OGC-GML (2015). Geography markup language. http://www.opengeospatial.org/standards/gml.

OGC-pyWPS (2015). Ogc implementing/compliant product details. http://www.opengeospatial.org/resource/products/details/?pid=706.

Ol Community Software (2015). Openlayers: Free maps for the web. http://openlayers.org/.

Ol Developer Team (2015). Mobile browsing. http://docs.openlayers.org/library/mobile.html.

Open Geospatial Consortium, O. (2015). Open geospatial consorcium. http://www.opengeospatial.org.

Open WPS, P. (2014). Welcome to the zoo-project. http://www.zoo-project.org.

OpenGeo (2015). Opengeo sdk client api reference. http://gxp.opengeo.org/master/doc/.

OpenGeoSuite (2015). Opengeo suite platform. http://boundlessgeo.com/solutions/opengeo-suite/.

OpenGIS, O. (2012). Ogc geography markup language (gml) — extended schemas and encoding rules. Technical Report 10-129r1, Open Geospatial Consortium.

OpenGIS, O. (2014). Ogc best practice for using web map services (wms) with time-dependent or elevation-dependent data. Technical Report 12-111r1, Open Geospatial Consortium.

OpenGIS KML, O. (2008). Ogc kml. Technical Report 07-147r2, Google-OGC.

OpenGIS WCPS, O. (2009). Web coverage processing servicespecification. Technical Report OGC 08-068r2, Open Geospatial Consortium.

OpenGIS WCS, O. (2008). Web coverage service implementation standard. Technical Report 07-067r2, Open Geospatial Consortium.

OpenGIS WFS, O. (2005). Web feature service implementation specification. Technical Report 04-094, Open Geospatial Consortium.

OpenGIS WKT, O. (2010). Geographic information - well-known text representation of coordinate reference systems. Technical Report 06-104r4, Open Geospatial Consortium.

OpenGIS WKT-CRS, O. (2013). Geographic information - well-known text representation of coordinate reference systems. Technical Report 12-063r5, Open Geospatial Consortium.

OpenGIS WMS, O. (2006). Opengis web map server implementation specification. Technical Report 06-042, Open Geospatial Consortium.

OpenGIS WMS-SLD, O. (2002). Styled layer descriptor profile of the web map service implementation specification. Technical Report 01-068r3, Open Geospatial Consortium.

OpenGIS WPS, O. (2007). Opengis web processing service. Technical Report 05-007r7, Open Geospatial Consortium.

OSGeo (2015). Gdal/ogr in python. https://trac.osgeo.org/gdal/wiki/GdalOgrInPython.

OSGeo (2015). Open source geospatial foundation. http://www.osgeo.org/.

Shekhar, S. and Xiong, H. (2007). Encyclopedia of gis, isbn: 978-0-387-35973-1. *Geoinformatica*.

Solorza, R., Argañaraz, J., and Landi, M. (2014). Metodología para la obtención del mapa de área quemada de media resolución en el marco de la elaboración de productos operativos para el proyecto presipa. Technical report, CONAE-IG.

Suda, B. (2003). Soap web services. thesis: Master of science computer science school of informatics. Technical Report Pag 25., University of Edinburgh.

UNC-CONAE (2015a). Manual descriptivo sobre la implementacion experimental del indice de peligrosidad meteorol´ogico ffdi y sus productos. http://webgis.isagro.org.ar/data/pdf/Indice_de_Peligrosidad_Meteorologia(FFDI).pdf.

UNC-CONAE (2015b). Procedimiento metodológico para la generación de mapa de perdida de vegetación. http://webgis.isagro.org.ar/data/pdf/Bosque.pdf.

Uruguay (2015). Isagro: Presentacion. http://www.montevideo.com.uy/auc.aspx?280633.

W3C, G. W. (2014). Web services glossary. http://www.w3.org/TR/ws-gloss/.

W3C Working Group, w. (2004). Web services architecture. http://www.w3.org/TR/ws-arch.

WPS, G. (2015). Web processing service administration. http://docs.geoserver.org/stable/en/user/extensions/wps/administration.html.

Zambelli, P., Gebbert, S., and Ciolli, M. (2013). Pygrass: An object oriented python application programming interface (api) for geographic resources analysis support system (grass) geographic information system (gis). *Geo-Information*, Vol. 2:201–219.