Introduzione pratica a ZOO: The Open WPS Platform

GFOSSDAY 2010 WORKSHOP

Giovedì 18 Nov - C6.S309
Centro studi di Foligno
Aula informatica

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

Informazioni generali

Luogo dove si è tenuto il Workshop:	Centro Studi di Foligno
Aula:	Aula informatica
Durata complessiva:	2h
Istruttore:	Luca DELUCCHI (Fondazione Edmund Mach) luca.delucchi@iasma.it

Prerequisiti

DVD con l'immagine di OSGeoLive Virtual Machine (che contiene anche la versione precompilata di ZOO 1.0 e i dati coi quali fare l'esercitazione)

Conoscenza base di OGC Web Processing Service specification (WPS 1.0)

Conoscenza base dei linguaggi di programmazione C e/o Python

Conoscenza base di JavaScript e OpenLayers

Link utili

Versione in HTML di ZOO Workshop: http://zoo-project.org/trac/wiki/ZooWorkshopFoss4g2010

Versione in PDF di ZOO Workshop: http://zoo-project.org/dl/foss4g2010ws.pdf

Sito ufficiale di ZOO Project:	http://zoo-project.org
Trac ufficiale di ZOO Project:	http://zoo-project.org/trac
Requisiti specifici di OGC WPS 1.0:	http://opengeospatial.org/standards/wps
Sito ufficiale di OSGeoLive:	http://live.osgeo.org/
Sito ufficiale di GDAL/OGR:	http://www.gdal.org/
Sito ufficiale di OpenLayers:	http://openlayers.org

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

Contenuti

1) Introduzione....4
1.1) Cos'è ZOO?....4
1.2) Come funziona ZOO?....4
1.3) Cosa faremo in questo workshop?....5
2) Usare ZOO da un OSGeoLive 6
2.1) Descrizione dell'ambiente di sviluppo....6
2.2) Testare l'installazione di ZOO con GetCapabilities....8
2.3) Preparare la vostra directory per ZOO ServiceProvider....9
3) Creating WPS compliant OGR based Web Services....10
3.1) Introduzione....10
3.2) Preparazione dei file di configurazione di ZOO....10
3.3) Sviluppare processi su geometrie singole....14
3.3.1) Boundary....14
3.3.1.1) Versione C....14
3.3.1.2) Versione python....20
3.3.1.3) Testare il Servizio usando la richiesta Execute....21
3.3.2) Creare servizi per altre funzioni (ConvexHull e Centroid)....24
3.3.2.1) Versione C 24
3.3.2.2) Versione python....26
3.3.3) Creare il servizio Buffer....28
3.3.3.1) Versione C....28
3.3.3.2) Versione python....30
3.3.3.3) Il file MetadataFile di Buffer....30
4) Costruire un client WPS usando OpenLayers....31
4.1) Creare una semplice mappa che mostri il set di dati come un WMS....31
4.2) Recuperare il layer dei dati come WFS e aggiungere il controllo di selezione....32
4.3) Richiamare i processi su geometrie singole con JavaScript....33
4.4) Richiamare i processi su geometrie multiple con JavaScript....35
5) Esercizio....37
5.1) Versione C....37
5.2) Versione python....38
5.3) Testare i servizi....38

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

1 Introduzione

1.1 Cos'è ZOO?

Zoo è un progetto open source WPS (Web Processing Service), recentemente rilasciato con licenza di tipo MIT/X-11. Fornisce un semplice framework WPS compatibile con OGC per creare e concatenare servizi WPS. ZOO è composto da tre parti:

- ✔ ZOO Kernel: un potente kernel server-side scritto in C, che permette di gestire e concatenare servizi Web scritti in diversi linguaggi di programmazione.
- ✔ ZOO Services: una serie di esempi di servizi Web, basati su diverse librerie open source.
- ✔ ZOO API: un API JavaScript server-side in grado di chiamare e concatenare dei Servizi ZOO, che rende più facile lo sviluppo e la concatenazione dei processi.

ZOO è progettato per rendere più facile lo sviluppo WPS sul lato server, fornendo un potente sistema in grado di capire ed eseguire interrogazioni conformi allo standard WPS. Supporta diversi linguaggi di programmazione, il che permette di creare i servizi Web nel linguaggio che si preferisce e di utilizzare codici esistenti. Maggiori informazioni sul progetto sono disponibili sul sito ufficiale del progetto ZOO.

1.2 Come funziona ZOO?

ZOO è basato su un 'WPS Kernel Service' che costituisce il core del sistema ZOO (aka ZOO Kernel). Quest'ultimo permette di caricare librerie dinamiche e di trattarle all'occorrenza come servizi web. Il Kernel ZOO è scritto in C ma supporta anche i più comuni linguaggi di programmazione per creare i Servizi di ZOO.

Un Servizio ZOO è un collegamento costituito da un file di configurazione ZOO (. zcfg) e dal codice per l'applicazione corrispondente. Il file di configurazione descrive tutte le funzioni disponibili che possono essere chiamate tramite una Richiesta WPS Exec e i formati di input/output. I Servizi contengono gli algoritmi e le funzioni, e attualmente possono essere implementati in C/C++, Fortran, Java, Python, PHP e JavaScript.

ZOO Kernel funziona con Apache e può comunicare con i motori cartografici e i clients Web mapping. Zoo Kernel semplicemente aggiunge il supporto WPS all'infrastruttura spaziale dei dati e all'applicazione di Web mapping. Accetta come dati di input tutti i formati supportati da GDAL/OGR e genera in uscita dati vettoriali o raster che si possono utilizzare nel motore cartografico e/o per le applicazioni client web-mapping.

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

1.3 Cosa faremo in questo workshop?

Lo scopo del presente workshop è di presentare il Progetto ZOO e le sue caratteristiche e di mostrare le sue potenzialità rispetto alle specifiche WPS 1.0.0. In 2 ore i partecipanti impareranno come utilizzare lo ZOO Kernel, come creare i Servizi ZOO e i loro file di configurazione e infine come collegare il servizio creato ad una applicazione client-side webmapping.

All'interno di OSGeoLive, il Live DVD ufficiale di OSGeo, è presente una versione pre-compilata di ZOO 1.0. Essa verrà usata durante questo workshop. Così i partecipanti non dovranno compilare ed installare ZOO Kernel manualmente. L'esecuzione e la verifica del funzionamento dello ZOO Kernel dal DVD OSGeoLive è dunque il primo passo del workshop. Ogni partecipante dovrebbe riuscire a disporre di un ZOO kernel funzionante in meno di 30 minuti.

Per prima cosa ZOO Kernel verrà testato su un browser Web, utilizzando le richieste di tipo GetCapabilities. Poi i partecipanti saranno invitati a creare uno ZOO Service Provider basato su OGR, per effettuare semplici operazioni spaziali su dati vettoriali. I partecipanti dovranno scegliere se creare il servizio utilizzando il linguaggio C oppure Python. Tutti i passaggi di programmazione del Servizio ZOO Provider ed i relativi servizi saranno spiegati in dettaglio sia in linguaggio C che in Python.

Quando i Servizi ZOO saranno pronti e richiamabili dallo ZOO Kernel, i partecipanti impareranno ad utilizzarne le diverse funzioni, utilizzando la semplice applicazione OpenLayers. Un set di dati campione, distribuito da Geoserver, verrà visualizzato su una mappa semplice mediante gli standard WMS / WFS e utilizzato come dati di input da parte dei Servizi ZOO. Quindi, alcuni specifici pulsanti di selezione ed esecuzione verranno aggiunti nel codice JavaScript in modo da eseguire modifche sui poligoni visualizzati con geometrie semplici e multiple.

L'intera procedura verrà spiegata in maniera dettagliata, passo dopo passo, fornendo anche pezzi di codice e le loro rispettive spiegazioni. Gli istruttori controlleranno che il Kernel ZOO funzioni su ogni macchina e assisteranno i partecipanti durante lo sviluppo del codice. Naturalmente le domande tecniche sono ben accette durante il workshop.

Aiuti nella lettura:

```
Gli esempi di codice sono riportati su sfondo giallo
```

```
I cambiamenti di codice sono evidenziati su sfondo grigio
```

`Esempi di codice` inclusi nel testo sono mostrati `in questo formato`

```
Il linguaggio HTTP e XML è riportato su sfondo blue
```

Partiamo!

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

2 Usare ZOO da OSGeoLive

2.1 Descrizione dell'ambiente di sviluppo

OSGeoLive è un live DVD basato su Xubuntu che permette di testare una grande varietà di software geospaziali senza doverli installare sulla propria macchina. Comprende solo programmi a licenza libera e quest'anno include anche ZOO 1.0.

Usare il LiveDVD è il modo più semplice per utilizzare lo ZOO Kernel e sviluppare i Servizi ZOO a livello locale. Ha il vantaggio di essere pronto all'uso e di contenere tutto il necessario per compilare i Servizi in C o eseguire i Servizi di Python. Il materiale necessario e il codice sorgente di ZOO sono localizzati nella cartella `/home/user/zoows`. Durante questo workshop si lavorerà in questa cartella. La versione binaria dello ZOO Kernel è già compilata e contenuta in `/home/user/zoows/sources/zoo-kernel`, che contiene due file (`zoo_loader.cgi` e `main.cfg`) che vanno copiati nella cartella `/usr/lib/cgi-bin` utilizzando i seguenti comandi:

```
sudo cp ~/zoows/sources/zoo-kernel/zoo_loader.cgi /usr/lib/cgi-bin
sudo cp ~/zoows/sources/zoo-kernel/main.cfg /usr/lib/cgi-bin
```

Durante questo workshop parleremo di ZOO Kernel e dello script `zoo_loader.cgi` senza fare alcuna distinzione.

Il filve `main.cfg` contiene informazioni per la configurazione dell'identificazione e del provider oltre ad alcune importanti impostazioni. Il file è composto da diverse sezioni, denominate di default `main`, `identification` e `provider`. Ovviamente, se c'è bisogno di sviluppare un servizio specifico, ognuno è libero di aggiungere a proprio piacimento nuove sezioni a file di configurazione. È indispensabile sapere che il nome `env` viene utilizzato in un determinato modo. Consente di definire le variabili d'ambiente richieste dal servizio durante la sua esecuzione. Ad esempio, se il servizio richiede di accedere ad un server X in esecuzione su framebuffer, è possibile aggiungere nella sezione `env` la riga `DISPLAY = 1` per prendere in considerazione questa specifica.

Si esamini questo file. Contiene tre parametri importanti, commentati di seguito:

- ✔ `serverAddress` : url per accedere al ZOO Kernel;
- ✔ `tmpPath` : percorso completo per archiviare i files temporanei;
- ✔ `tmpUrl` : percorso relativo al serverAddress per accedere alla cartella temporanea.

I valori del file `main.cfg` che attivano la macchina virtuale sono i seguenti:

```
serverAddress=http://localhost/zoo
tmpPath=/var/www/temp
```

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

```
tmpUrl=../temp/
```

Il `tmpUrl` è il percorso relativo dal `serverAddress`, quindi deve essere una directory. Anche se ZOO Kernel può essere utilizzato con l'url completo dello script `zoo_loader.cgi`, per una sua migliore leggibilità e funzionalità, è necessario modificare la configurazione di default di Apache, in modo da utilizzare direttamente http://localhost/zoo.

Per prima cosa creare una cartella `zoo` nella cartella esistente `/var/www` che è usata da Apache come `DirectoryIndex`. Poi il file di configurazione `/etc/apache2/sites-available/default` e aggiungere le seguenti linee dopo il blocco `Directory` relativo alla directory `/var/www` :

```
<Directory /var/www/zoo/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride All
        Order allow,deny
        allow from all
</Directory>
```

Ora creare un file `htaccess` in `/var/www/zoo` contenente le seguenti linee:

```
RewriteEngine on
RewriteRule (.*)/(.*) /cgi-bin/zoo_loader.cgi?metapath=$1 [L,QSA]
RewriteRule (.*) /cgi-bin/zoo_loader.cgi [L,QSA]
```

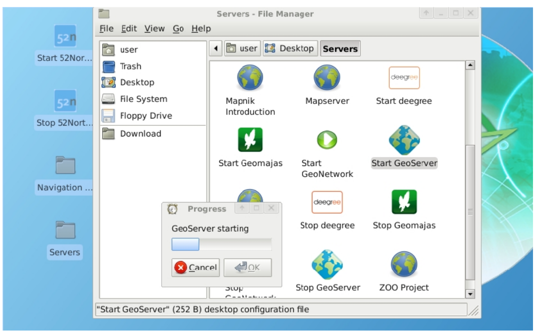
Per questo ultimo file (che deve essere preso in considerazione da Apache), è necessario attivare il modulo rewrite di Apache copiando un file di carico, come mostrato di seguito:

```
sudo cp /etc/apache2/mods-available/rewrite.load /etc/apache2/mods-enabled/
```

Ora dovrebbe essere possibile accedere allo ZOO Kernel semplicemente riavviando il Web Server Apache:

```
sudo /etc/init.d/apache2 restart
```

Due altri software dell'ambiente OSGeoLive verranno utilizzati nel corso di questo workshop. Per prima cosa si utilizzerà Geoserver per fornire i dati WFS di input al Servizio ZOO che si sta sviluppando. Poi, nella sezione 3 un dataset di esempio fornito da Geoserver (poligoni degli Stati Uniti) verrà utilizzato per provare il Servizio. Ora si avvii Geoserver usando il corrispondente bottone nella cartella `Servers`, come illustrato nel seguente screenshot:



La libreria OpenLayers library è anch'essa disponibile su OSGeoLive, e sarà usata durante la sessione 4 per creare una semplice applicazione WPS client capace di interrogare i Servizi ZOO appena sviluppati.

Poiché si è deciso di usare OGR C-API ed il modulo Python delle librerie GDAL, si deve avere a disposizione i corrispettivi file di intestazione, le librerie e i file associati. Fortunatamente tutto quello che serve è disponibile di default su OSGeoLive ed è pronto per l'uso.

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

2.2 Testare la corretta installazione di ZOO con GetCapabilities

Ora è possibile interrogare lo ZOO Kernel usando la seguente richiesta da un browser Internet:

```
http://localhost/cgi-bin/zoo_loader.cgi?Request=GetCapabilities&Service=WPS
```

Si ottiene il seguente documento `Capabilities` XML:


```
-<wps:Capabilities xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net
/wps/1.0.0/wpsGetCapabilities_response.xsd" service="WPS" version="1.0.0" xml:lang="en-US">
 -<ows:ServiceIdentification>
    <ows:Title>The Zoo WPS Server</ows:Title>
    <ows:Abstract>FOSS4G 2010 - WorkShop ZooWPS.</ows:Abstract>
    <ows:Fees>None</ows:Fees>
    <ows:AccessConstraints>none</ows:AccessConstraints>
  -<ows:Keywords>
      <ows:Keyword>WPS</ows:Keyword>
      <ows:Keyword>GIS</ows:Keyword>
      <ows:Keyword>buffer</ows:Keyword>
    </ows:Keywords>
    <ows:ServiceType>WPS</ows:ServiceType>
    <ows:ServiceTypeVersion>1.0.0</ows:ServiceTypeVersion>
  </ows:ServiceIdentification>
```


Nessun nodo `Process` compare nella sezione `ProcessOfferings`, perché non ci sono ancora Servizi ZOO disponibili. Si può eseguire una richiesta `GetCapabilities` anche dalla linea di comando, usando:

```
./zoo_loader.cgi "request=GetCapabilities&service=WPS"
```

Il browser restituirà lo stesso risultato, come mostrato nel seguente screenshot:

```
Terminal - user@user: /usr/lib/cgi-bin
File  Edit  View  Terminal  Go  Help
user@user:/usr/lib/cgi-bin$ ./zoo_loader.cgi "request=getcapabilities&service
=wps" 2> /home/user/zoo_debug.log
Content-Type: text/xml; charset=utf-8
Status: 200 OK

<?xml version="1.0" encoding="utf-8"?>
<wps:Capabilities xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:wps="http:
//www.opengis.net/wps/1.0.0" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns
:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://w
ww.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsGetCapabilit
ies_response.xsd" service="WPS" version="1.0.0" xml:lang="en-US">
  <ows:ServiceIdentification>
    <ows:Title>The Zoo WPS Server</ows:Title>
    <ows:Abstract>FOSS4G 2010 - WorkShop ZooWPS.</ows:Abstract>
    <ows:Fees>None</ows:Fees>
    <ows:AccessConstraints>none</ows:AccessConstraints>
    <ows:Keywords>
      <ows:Keyword>WPS</ows:Keyword>
      <ows:Keyword>GIS</ows:Keyword>
      <ows:Keyword>buffer</ows:Keyword>
    </ows:Keywords>
    <ows:ServiceType>WPS</ows:ServiceType>
    <ows:ServiceTypeVersion>1.0.0</ows:ServiceTypeVersion>
  </ows:ServiceIdentification>
```

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

2.3 Preparare la cartella per lo ZOO ServiceProvider

Per prima cosa occorre spiegare come deve essere strutturata la cartella quando si crea un nuovo Services Provider:

- ✔ La directory principale del Services Provider include:
  - ✔ una cartella `cgi-env` che conterrà tutti i file di configurazione zcfg e librerie dinamiche (librerie dinamiche C o moduli Python);
  - ✔ il `Makefile` e i file `*c` necessari per compilare il Services Provider.

Nell'esistente cartella `zoo-services` situata in `/home/user/zoows/sources/` creare una cartella principale denominata `ws_sp` per il Services Provider, rispettando il seguente albero:

```
mkdir -p /home/user/zoows/sources/zoo-services/ws_sp/cgi-env
```

Il `Makefile` e il codice del Service Shared Object in C o Python saranno spiegati nella prossima sezione.

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

3 Creare un WPS basato su un Servizio Web che supporti OGR

3.1 Introduzione

In questa parte si andrà a creare un ServiceProvider ZOO che utilizza i servizi (basati su OGR API per C oppure sul modulo OGR per Python) contenuti nell'installazione ZOO sul DVD OSGeoLive. L'obiettivo è di usare OGR e alcune sue semplici funzioni spaziali GEOS per i Servizi WPS.

Inizieremo con la funzione Boundary. Ogni singolo passo verrà spiegato, codificato e infine testato. Saranno poi spiegate le funzioni Buffer, Centroid e Convex Hull. Infine si sperimenteranno alcune funzioni su geometrie multiple (Intersection, Union, Difference e Symmetric Difference).

Come già anticipato, è necessario scegliere se scrivere i Servizi in linguaggio C oppure in Python (o entrambi!). Le spiegazioni saranno prevalentemente basate sul linguaggio C, ma si riveleranno utili anche per coloro che sceglieranno di utilizzare Python. Si decida in base alle abitudini e preferenze. I risultati saranno gli stessi in entrambi i casi.

3.2 Preparazione dei file di configurazione di ZOO

Un Servizio ZOO è una combinazione del file di configurazione ZOO (.zcfg) e del modulo runtime per l'applicazione corrispondente (comunemente chiamata ZOO Service Provider). Per prima cosa è necessario preparare il file .zcfg: utilizzando un editor di testo creare un file (che chiameremo `Boundary.zcfg`) nella cartella `/home/user/zoows/sources/zoo-services/ws_sp`. Per prima cosa, all'inizio del file si deve inserire tra parantesi quadre il nome del Servizio che si vuole usare, come segue:

```
[Boundary]
```

Questo nome è importante, si tratta del nome del Servizio e quindi il nome della funzione definita nel Services Provider. Un titolo e un riassunto possono essere aggiunti per informare gli utilizzatori di cosa fa il servizio:

```
Title = Compute boundary.
Abstract = Returns the boundary of the geometry on which the method is invoked.
```

Queste informazioni di configurazione saranno visualizzate quando si fa una richiesta `GetCapabilities`.

Si possono aggiungere anche altre informazioni, come il processVersion. Si può decidere di far archiviare i risultati allo ZOO Service (impostando il parametro storeSupported su true) oppure che la funzione sia eseguita in background, mostrando il suo stato corrente (impostando statusSupported su true):

```
 processVersion = 1
 storeSupported = true
 statusSupported = true
```

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

Nella sezione main del file metadati del Servizio ZOO, è necessario anche specificare due fattori molto importanti:

- ✔ `serviceProvider`, che è il nome della libreria C condivisa contenente la funzione del Servizio, oppure il nome del modulo Python.
- ✔ `ServiceType`, che definisce il linguaggio di programmazione usato per il Servizio. (il valore può essere C oppure Python a seconda del linguaggio scelto)

Questo è l'esempio di come deve essere settato il ServiceProvider se si scrive in C:

```
serviceProvider=ogr_ws_service_provider.zo
serviceType=C
```

In questo caso si ottiene, posizionato nella directory dove c'è ZOO Kernel, il file della libreria dinamica `ogr_ws_service_provider.zo` contenente la funzione `Boundary`.

Questo è l'esempio di come deve essere settato il ServiceProvider se si scrive in Python:

```
serviceProvider=ogr_ws_service_provider
serviceType=Python
```

In questo caso, si ottiene un file `ogr_ws_service_provider.py` contenente il codice Python della funzione `Boundary`.

Nella sezione main, si possono aggiungere anche altri metadati, come ad esempio:

```
<MetaData>
 Title = Demo
</MetaData>
```

A questo punto abbiamo definito le principali informazioni di metadata. Ora è necessario definire quali dati di input utilizzerà il Servizio ZOO. È possibile definire qualsiasi input necessario al Servizio. Da notare che è possibile effettuare un numero maggiore di richieste allo ZOO Kernel rispetto ai dati di input definiti nel file .zcfg. Questi valori verranno semplicemente passati al servizio senza venire utilizzati. Nel Servizio di esempio Boundary, un singolo poligono sarà usato come input, che verrà utilizzato per la funzione Boundary.

La dichiarazione dei dati utilizzati in input è contenuta nel blocco `DataInputs`. Viene usata la stessa sintassi del Servizio, col nome dei dati di input messo tra parentesi quadre. È inoltre possibile inserire titolo, abstract e una sezione di metadati. È necessario impostare i parametri `minOccurs` e `maxOccurs` per riuscire ad eseguire la funzione del Servizio.

```
[InputPolygon]
 Title = Polygon to compute boundary
 Abstract = URI to a set of GML that describes the polygon.
 minOccurs = 1
 maxOccurs = 1
 <MetaData lang="en">
  Test = My test
 </MetaData>
```

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

I metadati definiscono la tipologia di dati supportata dal Servizio. Nell'esempio del Boundary, il poligono di input può essere passato come un file GML o una stringa JSON. Il passo successivo è quello di definire i formati di input di default e quelli supportati. Entrambi i formati devono essere dichiarati a seconda della loro tipologia nel blocco `LitteralData` o in quello `ComplexData`. Per questo primo esempio useremo solo il blocco `ComplexData`.

```
<ComplexData>
 <Default>
  mimeType = text/xml
  encoding = UTF-8
 </Default>
 <Supported>
  mimeType = application/json
  encoding = UTF-8
 </Supported>
</ComplexData>
```

A questo punto occorre attribuire gli stessi metadati ai dati in output del Servizio, all'interno di un blocco `DataOutputs`:

```
[Result]
 Title = The created geometry
 Abstract = The geometry containing the boundary of the geometry on which the method
was invoked.
 <MetaData lang="en">
   Title = Result
 </MetaData>
 <ComplexData>
  <Default>
   mimeType = application/json
   encoding = UTF-8
  </Default>
  <Supported>
   mimeType = text/xml
   encoding = UTF-8
  </Supported>
 </ComplexData>
```

Una copia completa di questo file .zcfg può essere trovata all'URL:

http://zoo-project.org/trac/browser/trunk/zoo-services/ogr/base-vect-ops/cgi-env/Boundary.zcfg

Una volta modificato il file metadati di ZOO, occorre copiarlo nella stessa cartella del Kernel di ZOO (`/usr/lib/cgi-bin`). A questo punto si è in grado di eseguire la seguente richiesta:

http://localhost/zoo/?Request=DescribeProcess&Service=WPS&Identifier=Boundary&version=1.0.0

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

Con la richiesta ProcessDescriptions dovrebbe generarsi il seguente documento XML:

```
-<wps:ProcessDescriptions xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.
 /wpsDescribeProcess_response.xsd" service="WPS" version="1.0.0" xml:lang="en">
  -<ProcessDescription wps:processVersion="1" storeSupported="true" statusSupported="true">
     <ows:Identifier>Boundary</ows:Identifier>
     <ows:Title>Compute boundary.</ows:Title>
   -<ows:Abstract>
       A new geometry object is created and returned containing the boundary of the geometry on which the me
     </ows:Abstract>
     <ows:Metadata xlink:Test="Demo"/>
   -<DataInputs>
     -<Input minOccurs="1" maxOccurs="1">
         <ows:Identifier>InputPolygon</ows:Identifier>
         <ows:Title>Polygon to compute boundary</ows:Title>
         <ows:Abstract>URI to a set of GML that describes the polygon.</ows:Abstract>
       -<ComplexData>
```

Si noti che `GetCapabilities` e `DescribeProcess` hanno bisogno solo del file .zcfg per essere completati. **Semplice, vero?** A questo punto, si faccia una richiesta `Execute` al Kernel ZOO. Si ottiene una risposta `ExceptionReport`, simile alla seguente:

```
-<ows:ExceptionReport xsi:schemaLocation="http://www.opengis.net/ows/1.1 http://schemas.opengis.net/ows/1.1.0
 /owsExceptionReport.xsd" xml:lan="en" service="WPS" version="1.0.0">
  -<ows:Exception exceptionCode="NoApplicableCode">
    -<ows:ExceptionText>
       C Library can't be loaded /usr/lib/cgi-bin//ogr_ws_service_provider.zo: cannot open shared object file: No such file or directory
     </ows:ExceptionText>
   </ows:Exception>
 </ows:ExceptionReport>
```

Se si proverà ad eseguire il Python Service, verrà generato il seguente messaggio di errore:

```
-<ows:ExceptionReport xsi:schemaLocation="http://www.opengis.net/ows/1.1 http://schemas.opengis.net/ows/1.1.0
 /owsExceptionReport.xsd" xml:lan="en" service="WPS" version="1.0.0">
  -<ows:Exception exceptionCode="NoApplicableCode">
    -<ows:ExceptionText>
       Python module ogr_ws_service_provider cannot be loaded.
     </ows:ExceptionText>
   </ows:Exception>
 </ows:ExceptionReport>
```

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

3.3 Sviluppare processi su geometrie singole

Per spiegare in maniera chiara il funzionamento del Services Provider, si fornisce una descrizione dettagliata (passo per passo) di come utilizzare la funzione `Boundary` , che è una delle più semplici. La stessa procedura verrà poi utilizzata per implementare funzioni più complesse, come `Buffer`, `Centroid` e `ConvexHull`.

Ora che i metadati sono completi, è necessario creare il codice del Servizio. La cosa più importante da sapere quando si sviluppa un Servizio ZOO è che la funzione corrispondente al Servizio richiede in entrata tre parametri (`maps` datatype interno o dizionari Python) e in uscita fornisce un valore intero che rappresenta lo stato dell'esecuzione (`SERVICE_FALLITO` or `SERVICE_TERMINATO_CON_SUCCESSO`):

- ✔ `conf` : la configurazione dell'ambiente principale (corrispondente al contenuto di `main.cfg`);
- ✔ `inputs` : i dati di input richiesti o di default;
- ✔ `outputs` : i dati di output richiesti o di default.

3.3.1 La funzione Boundary

3.3.1.1 Versione in C

Come spiegato in precedenza, il Kernel di ZOO fornisce i parametri alla funzione del Servizio in uno specifico formato chiamato `maps`. Per codificare il Servizio con il linguaggio C è necessario sapere come accedere a questo tipo di dati in modalità lettura/scrittura.

Le `maps` sono named linked list di mappe (`map`) contenenti un `name`, una `content map` e un puntatore alla mappa successiva della lista (`next map` oppure `NULL` se non c'è più nessun'altra `map` nella lista). Di seguito la definizione del formato come lo si trova nel file `zoo-kernel/service.h` :

```
typedef struct maps{
      char* name;
      struct map* content;
      struct maps* next;
 } maps;
```

The `map` inserita nella `maps` è anche una semplice linked list e viene usata per immagazzinare i valori Key Value Pair. Una `map` contiene quindi la coppia `name` e `value` e il puntatore all'elemento successivo della lista. Di seguito la definizione del formato dei dati che si può trovare nel file `zoo-kernel/service.h`:

```
typedef struct map{
   char* name;        /* The key */
   char* value;       /* The value */
   struct map* next; /* Next couple */
} map;
```

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

Siccome la struttura dei dati viene parzialmente o interamente completata dallo ZOO Kernel del Servizio, non serve perdere tempo nella creazione della `maps`, ma basta lavorare direttamente sull'esistente `map`, in altre parole il contenuto di ogni `maps`. La prima funzione che bisogna conoscere è `getMapFromMaps` (definita nel file `zoo-kernel/service.h`) che permette di accedere ad una specifica `map` di una `maps`.

Questa funzione ha tre parametri, elencati qui sotto:

- ✔ `m` : un puntatore in `maps` che rappresenta la `maps` utilizzata per cercare una specifica `map`
- ✔ `name` : un `char*` , che rappresenta il nome della `map` che si sta cercando
- ✔ `key` : una chiave specifica nella `map` chiamata `name`

Per esempio, il codice C che estrae dalla `maps` chiamata `inputs` il valore della `map` InputPolygon avrà la seguente sintassi:

```
map* tmp=getMapFromMaps(inputs,"InputPolygon","value");
```

Una volta che si ha la `map`, è possibile accedere ai campi `name` o `value`, usando la seguente sintassi:

```
tmp->name
tmp->value
```

Una volta appreso come leggere ed accedere ai campi della `map` da una `maps`, si procede imparando come scrivere in questo tipo di datastructure. Si può fare usando la pratica funzione `addToMap` definita sempre nel file `zoo-kernel/service.h`. Anche la funzione `addToMap` richiede tre parametri:

- ✔ `m` : un puntatore alla `map` che si vuole aggiornare,
- ✔ `n` : il nome della `map` di cui si vuole aggiungere o aggiornare il valore,
- ✔ `v` : il valore che si vuole attribuire alla `map`.

Di seguito un esempio di come aggiungere o creare la `content map` di una `maps` chiamata `outputs` :

```
addToMap(outputs->content,"value","Hello from the C World !");
addToMap(outputs->content,"mimeType","text/plain");
addToMap(outputs->content,"encoding","UTF-8");
```

La funzione `addToMap` è in grado di creare o aggiornare una `map` esistente. Infatti, se non esiste una `map` chiamata «value» essa viene creata. Altrimenti il suo valore viene sovrascritto automaticamente.

Questo datatype è davvero importante in quanto è usato in tutti i Servizi di ZOO basati su C. È inoltre la stessa rappresentazione utilizzata in altri linguaggi, ma usando i rispettivi datatype. Per esempio in Python, si usa un datatype `dizionario`, che facilita la digitalizzazione.

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

Di seguito si riporta un esempio di `maps` usando il linguaggio Python (si tratta di una versione ridotta della configurazione principale di `maps`):

```
main={
   "main": {"encoding": "utf-8",
            "version": "1.0.0",
            "serverAddress": "http://www.zoo-project.org/zoo/",
            "lang": "fr-FR,en-CA"},
   "identification": {"title": "The Zoo WPS Development Server",
                      "abstract": "Development version of ZooWPS.",
                      "fees": "None",
                      "accessConstraints": "none",
                      "keywords": "WPS,GIS,buffer"}
}
```

Ora che sono chiari i concetti di `maps` e `map`, è possibile iniziare a comporre il codice del primo Servizio ZOO usando la funzione OGR Boundary.

Come anticipato nell'introduzione, si utilizza il server Geoserver WFS disponibile sul DVD OSGeoLive, in modo che le WFS Response vengano usate come valori di input. Siccome si utilizzeranno funzioni OGR di geometria semplice come `OGR_G_GetBoundary`, saranno usato solo le Geometry object, piuttosto che l'intero WFS Response. La prima cosa da fare è di scrivere una funzione che definisca la geometria dall'intero WFS Response. Questo verrà denominato `createGeometryFromWFS`.

Di seguito il codice di questa funzione:

```
OGRGeometryH createGeometryFromWFS(maps* conf,char* inputStr){
   xmlInitParser();
   xmlDocPtr doc = xmlParseMemory(inputStr,strlen(inputStr));
   xmlChar *xmlbuff;
   int buffersize;
   xmlXPathContextPtr xpathCtx;
   xmlXPathObjectPtr xpathObj;
   char * xpathExpr="/*/*/*/*/*[local-name()='Polygon' or local-name()='MultiPolygon']";
   xpathCtx = xmlXPathNewContext(doc);
   xpathObj = xmlXPathEvalExpression(BAD_CAST xpathExpr,xpathCtx);
   if(!xpathObj->nodesetval){
     errorException(conf, "Unable to parse Input Polygon","InvalidParameterValue");
     exit(0);
   }
   int size = (xpathObj->nodesetval) ? xpathObj->nodesetval->nodeNr : 0;
   xmlDocPtr ndoc = xmlNewDoc(BAD_CAST "1.0");
   for(int k=size-1;k>=0;k--){
      xmlDocSetRootElement(ndoc, xpathObj->nodesetval->nodeTab[k]);
   }
   xmlDocDumpFormatMemory(ndoc, &xmlbuff, &buffersize, 1);
   char *tmp=strdup(strstr((char*)xmlbuff,"?>")+2);
   xmlXPathFreeObject(xpathObj);
   xmlXPathFreeContext(xpathCtx);
   xmlFree(xmlbuff);
   xmlFreeDoc(doc);
   xmlCleanupParser();
   OGRGeometryH res=OGR_G_CreateFromGML(tmp);
   if(res==NULL){
       map* tmp=createMap("text","Unable to call OGR_G_CreatFromGML");
       addToMap(tmp,"code","NoApplicableCode");

       exit(0);
 }
 else
```

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

```
 return res;
}
```

La sola cosa sulla quale è importante soffermarsi è la funzione `errorException` compresa nel corpo della funzione. Tale funzione è dichiarata nel file `zoo-kernel/service_internal.h` e definita in `zoo-kernel/service_internal.c`. Può assumere tre parametri:

- ✔ il principale ambiente della `maps`,
- ✔ un `char*` che rappresenta il messaggio d'errore da visualizzare,
- ✔ una `char*` che rappresenta il codice dell'errore (come definito nelle specifiche WPS – Tabella 62).

In altre parole, se il WFS Response non può essere analizzato correttamente, si genera un documento `ExceptionReport` che informa il client che si è verificato un problema.

Ora che si è scritta la funzione per estrarre la geometry object dal WFS, è possibile iniziare a definire il Servizio Boundary. Di seguito il codice completo per il Servizio Boundary:

```
int Boundary(maps*& conf,maps*& inputs,maps*& outputs){
   OGRGeometryH geometry,res;
   map* tmp=getMapFromMaps(inputs,"InputPolygon","value");
   if(tmp==NULL)
     return SERVICE_FAILED;
   map* tmp1=getMapFromMaps(inputs,"InputPolygon","mimeType");
   if(strncmp(tmp1->value,"application/json",16)==0)
     geometry=OGR_G_CreateGeometryFromJson(tmp->value);
   else
     geometry=createGeometryFromWFS(conf,tmp->value);
   res=OGR_G_GetBoundary(geometry);
   tmp1=getMapFromMaps(outputs,"Result","mimeType");
   if(strncmp(tmp1->value,"application/json",16)==0){
       addToMap(outputs->content,"value",OGR_G_ExportToJson(res));
       addToMap(outputs->content,"mimeType","text/plain");
   }
   else{
       addToMap(outputs->content,"value",OGR_G_ExportToGML(res));
   }
   outputs->next=NULL;
   OGR_G_DestroyGeometry(geometry);
   OGR_G_DestroyGeometry(res);
   return SERVICE_SUCCEEDED;
}
```

Come si può vedere dal seguente codice, per prima cosa viene controllato il `mimeType` dei dati in ingresso:

```
   map* tmp1=getMapFromMaps(inputs,"InputPolygon","mimeType");
   if(strncmp(tmp1->value,"application/json",16)==0)
     geometry=OGR_G_CreateGeometryFromJson(tmp->value);
   else
     geometry=createGeometryFromGML(conf,tmp->value);
```

Se si sceglie di impostare il `mimeType` in ingresso su `application/json`, si utilizza la `OGR_G_CreateGeometryFromJson` , negli altri casi si utilizza la funzione locale `createGeometryFromGML`.

Si noti che i dati di input non sono tutti dello stesso tipo. Visto che si è usato direttamente `OGR_G_CreateGeometryFromJson` questo significa che la stringa JSON include solamente l'oggetto della geometria e non l'intera stringa GeoJSON. Tuttavia è possibile andare a modificare questo codice in modo che vada ad utilizzare l'intera stringa GeoJSON, creando semplicemente una funzione che estragga l'oggetto della geometria dalla stringa GeoJSON (usando per esempio la json-c library, che è usata anche dall'OGR GeoJSON Driver).

Definiti gli oggetti della geometria in ingresso, è ora possibile usare la funzione `OGR_G_GetBoundary` e salvare il risultato nella variabile `res` geometry. A questo punto si deve solamente salvare il valore nel giusto formato: GeoJSON per default oppure GML, se lo si era scelto come formato per i dati in uscita.

Da notare che lo ZOO Kernel fornisce valori `outputs` già precompilati. Così basta solo inserire il valore della key denominata `value`. Nell'esempio qui proposto si è invece scelto di sovrascrive il `mimeType` usando il valore `text/plain` piuttosto che il `application/json` (per dimostrare che è possibile andare a modificare anche altri campi della `map`). Infatti, a seconda del formato richiesto dal client (o per default) si ottiene la rappresentazione della geometria in JSON o in GML.

```
tmp1=getMapFromMaps(outputs,"Result","mimeType");
if(strncmp(tmp1->value,"application/json",16)==0){
  addToMap(outputs->content,"value",OGR_G_ExportToJson(res));
  addToMap(outputs->content,"mimeType","text/plain");
}
else{
  addToMap(outputs->content,"value",OGR_G_ExportToGML(res));
}
```

A questo punto il Servizio Boundary è pronto, basta solo compilarlo per produrre una Libreria Dinamica. Siccome si utilizzano le funzioni definite in `service.h` (`getMapFromMaps` e `addToMap`), è necessario includere questo file nel codice C. La stessa operazione deve essere fatta se si vuole utilizzare anche la funzione `errorException` dichiarata in `zoo-kernel/service_internal.h`. È necessario anche collegare la libreria allo `zoo-kernel/service_internal.o` per utilizzare in tempo reale `errorException`. Infine occorre includere i file necessari ad accedere alla libxml2 e alle C-API di OGR.

Per soddisfare la Libreria Dinamica bisogna inserire il codice in un blocco chiamato `extern "C"`. Il codice del Servizio finale viene salvato nel file `service.c` nella cartella root del Servizio Provider (quindi in `/home/zoows/sources/zoo-services/ws_sp`). Dovrebbe essere il seguente:

```
#include "ogr_api.h"
#include "service.h"
extern "C" {
#include <libxml/tree.h>
#include <libxml/parser.h>
#include <libxml/xpath.h>
#include <libxml/xpathInternals.h>
<YOUR SERVICE CODE AND OTHER UTILITIES FUNCTIONS>
}
```

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

A questo punto si procede generando la corrispondente Libreria Dinamica compilando il codice come una Libreria Dinamica. Questo può essere fatto usando il seguente comando:

```
g++ $CFLAGS -shared -fpic -o cgi-env/ServiceProvider.zo ./service.c $LDFLAGS
```

Attenzione che bisogna indicare i valori della variabili ambientali `CLFAGS` e `LDFLAGS`.

`CFLAGS` deve contenere tutti quei percorsi richiesti per trovare gli headers, come il percorso alle cartelle che contengono la cartella `libxml2` e i file `service.h`, `ogr_api.h` e `service_internal.h`. Sfruttando l'ambiente di OSGeoLive, è possibile usare due strumenti (`xml2-config` and `gdal-config` , entrambi usati con l'argomento `–cflags`) per recuperare tali valori. Questi generano il percorso richiesto.

Se si sono seguite le istruzioni su come creare la cartella principale del Servizio Provider di Zoo nello `zoo-services`. A questo punto dovrebbe essere possibile visualizzare gli headers del Kernel di ZOO e l'albero sorgente nella cartella `../../zoo-kernel` relativa al percorso corrente (`/home/user/zoows/sources/zoo-services/ws_sp`). Si noti che è possibile utilizzare anche il percorso completo alla cartella `zoo-kernel` , ma il percorso relativo è preferibile in quanto permette di spostare dovunque l'albero delle sorgenti e conservare il codice aggiornato utilizzando la stessa linea di comando. A questo punto bisogna aggiungere `-I../../zoo-kernel` allo `CFLAGS` per permettere al compilatore di trovare i file `service.h` e `service_internal.h` .

La completa definizione `CFLAGS` dovrebbe apparire come la seguente:

```
CFLAGS=`gdal-config --cflags` `xml2-config --clfags` -I../../zoo-kernel/
```

Una volta inseriti nel `CFLAGS` i percorsi correttamente settati, occorre concentrarsi sulla libreria alla quale ci si è già collegati (definita nella variabile ambientale `LDFLAGS`). Per collegare nuovamente gdal e le librerie libxml2, si possono sfruttare gli stessi strumenti usati in precedenza usando l'argomento `--libs` invece di `--cflags`. La definizione `LDFLAGS` completa deve essere la seguente:

```
LDFLAGS=`gdal-config --libs` `xml2-config --libs` ../../zoo-kernel/service_internal.o
```

Creare ora il `Makefile` di supporto alla compilazione del codice. Si scriva un breve `Makefile` nella root del Servizio Provider di ZOO contenente le seguenti linee:

```
ZOO_SRC_ROOT=../../zoo-kernel/
CFLAGS=-I${ZOO_SRC_ROOT} `xml2-config --cflags` `gdal-config --cflags`
LDFLAGS=`xml2-config --libs` `gdal-config --libs`${ZOO_SRC_ROOT}/service_internal.o

cgi-env/ogr_ws_service_provider.zo: service.c
	g++ ${CFLAGS} -shared -fpic -o cgi-env/ogr_ws_service_provider.zo ./service.c $
{LDFLAGS}

clean:
	rm -f cgi-env/ogr_ws_service_provider.zo
```

Usando questo `Makefile`, è possibile eseguire make dalla cartella principale del Servizio Provider di ZOO e salvare il risultante file `ogr_ws_service_provider.zo` nella cartella `cgi-env` .

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

Il file metadata e le Librerie Condivise si trovano ora nella cartella `cgi-env` . Per distribuire il nuovo Servizio Provider basta solo copiare le Librerie Condivise e il corrispettivo file metadata nelle cartella dove si trova il Kernel ZOO, cioè in `/usr/lib/cgi-bin`. Per fare ciò basta usare il seguente comando `sudo`:

```
sudo cp ./cgi-env/* /usr/lib/cgi-bin
```

L'utilità del codice sorgente del Servizio Provider di ZOO dovrebbe essere ora più chiara! La cartella `cgi-env` directory consente di propagare i nuovi Servizi o quelli di Provider in modo semplice, copiando semplicemente l'intero contenuto di `cgi-env` nella cartella `cgi-bin` . È possibile aggiungere le seguenti linee al `Makefile` per riuscire ad editare la cartella `make install` e rendere disponibile il nuovo Servizio Provider al Kernel di ZOO:

```
install:
	sudo cp ./cgi-env/* /usr/lib/cgi-bin
```

Il Servizio Provider di ZOO è a questo punto pronto per esser utilizzato tramite una richiesta `Execute` eseguita dal Kernel di ZOO.

3.3.1.2 Versione in Python

Per coloro che hanno deciso di usare Python per sviluppare i loro ZOO Services Provider, di seguito si riporta il codice completo, che deve essere copiato nel file `ogr_ws_service_provider.py` della cartella `cgi-env`. Siccome Python è un linguaggio interpretato, non occorre compilare nulla prima di rendere disponibile il servizio, il che semplifica questo passaggio.

```
import osgeo.ogr
import libxml2

def createGeometryFromWFS(my_wfs_response):
	doc=libxml2.parseMemory(my_wfs_response,len(my_wfs_response))
	ctxt = doc.xpathNewContext()
	res=ctxt.xpathEval("/*/*/*/*/*[local-name()='Polygon' or local-   name()
='MultiPolygon']")
	for node in res:
		geometry_as_string=node.serialize()
		geometry=osgeo.ogr.CreateGeometryFromGML(geometry_as_string)
		return geometry
	return geometry

def Boundary(conf,inputs,outputs):
	if inputs["InputPolygon"]["mimeType"]=="application/json":
		geometry=osgeo.ogr.CreateGeometryFromJson(inputs["InputPolygon"]["value"])
	else:
		geometry=createGeometryFromWFS(inputs["InputPolygon"]["value"])
	rgeom=geometry.GetBoundary()
	if outputs["Result"]["mimeType"]=="application/json":
		outputs["Result"]["value"]=rgeom.ExportToJson()
		outputs["Result"]["mimeType"]="text/plain"
	else:
		outputs["Result"]["value"]=rgeom.ExportToGML()
	geometry.Destroy()
	rgeom.Destroy()
	return 3
```

Non ci si sofferma a spiegare le funzioni presenti nel codice (sono state già spiegate in precedenza nel dettaglio) e il codice è stato scritto volutamente nello stesso modo.

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

Come già detto, si deve solamente copiare i file `cgi-env` nella cartella `cgi-bin` :

```
sudo cp ./cgi-env/* /usr/lib/cgi-bin
```

Un semplice `Makefile` contenente la sezione d'installazione può essere scritto nel seguente modo:

```
install:
	sudo cp ./cgi-env/* /usr/lib/cgi-bin/
```

Infine, per rendere operativo lo ZOO Service Provider, si esegue il comando `make install` dalla cartella principale di ZOO Services Provider.

3.3.1.3 Testare il Servizio tramite la richiesta Execute

3.3.1.3.1 Modo semplice e non leggibile

Ciascuno a questo punto dovrebbe disporre della propria copia dell'OGR Boundary Service nello ZOO Services Provider, chiamata `ogr_ws_service_provider` e resa disponibile nello ZOO Kernel. Per testare il Servizio si può usare la seguente richiesta `Execute`:

```
http://localhost/cgi-bin/zoo_loader.cgi?
request=Execute&service=WPS&version=1.0.0&Identifier=Boundary&DataInputs=InputPolygon=Reference@xlink:href=
http%3A%2F%2Flocalhost%3A8082%2Fgeoserver%2Fows%3FSERVICE%3DWFS%26REQUEST%3DGetFeature
%26VERSION%3D1.0.0%26typename%3Dtopp%3Astates%26SRS%3DEPSG%3A4326%26FeatureID%3Dstates.15
```

Come si può vedere dall'url sopra, si usa una richiesta WFS URLEncoded al server WFS di GeoServer disponibile su OSGeoLive usando una chiave `xlink:href` nel valore KVP `DataInputs`, e si setta `Reference` al campo `InputPolygon`. La corrispondente richiesta WFS non codificata è la seguente:

```
http://localhost:8082/geoserver/ows/?
SERVICE=WFS&REQUEST=GetFeature&VERSION=1.0.0&typename=topp:states&SRS=EPSG:4326&FeatureID=state
s.15
```

Se si vuole ottenere informazioni riguardo ai valori in input utilizzati per attivare il Servizio è possibile aggiungere `lineage=true` alla precedente richiesta. Inoltre è anche possibile salvare il documento `ExecuteResponse` di risposta del Servizio ZOO per riutilizzarlo successivamente. In questo caso si deve aggiungere `storeExecuteResponse=true` alla richiesta precedente. Quando questo parametro è settato su true il comportamento dello ZOO Kernel non è esattamente lo stesso rispetto a quando viene lanciato senza: in tal caso, ZOO Kernel rilascia un `ExecuteResponse` documento che contiene anche l'attributo `statusLocation`, che informa il client su dove sarà salvato l'ExecuteResponse in corso o quello finale.

Ecco esempio di come appare l'ExecuteResponse se nella richiesta si usa `storeExecuteResponse=true`:

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

```
-<wps:ExecuteResponse 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" serviceInstance="http://localhost/zoo/" statusLocation="http://localhost/zoo/..
 /temp/ogr_service.zo_21487.xml">
 -<wps:Process wps:processVersion="1">
    <ows:Identifier>Boundary</ows:Identifier>
    <ows:Title>Compute boundary.</ows:Title>
  -<ows:Abstract>
     A new geometry object is created and returned containing the boundary of the geometry on which the method is invoked.
    </ows:Abstract>
  </wps:Process>
 -<wps:Status creationTime="2010-09-02T09:36:16Z">
    <wps:ProcessAccepted/>
  </wps:Status>
 </wps:ExecuteResponse>
```


Se si usa `statusLocation` si ritorna all'ExecuteResponse generato con la richiesta precedente. Questo può rivelarsi estremamente utile per fornire sistemi di caching per una applicazione client.

Nella precedente richiesta non è stato necessario specificare il ResponseForm, in quanto non è richiesto. Usando `application/json` `mimeType` come stabilito nel file zcfg si dovrebbe ottenere di default un `ResponseDocument`. In ogni caso, è possibile specificate allo ZOO Kernel quali tipi di dati si vogliono visualizzare nel risultato della query aggiungendo l'attributo `mimeType=text/xml` ai parametri del `ResponseDocument`. Aggiungendo questo parametro alla precedente richiesta il risultato sarà la seguente rappresentazione GML:

```
http://localhost/cgi-bin/zoo_loader.cgi?
request=Execute&service=WPS&version=1.0.0&Identifier=Boundary&DataInputs=InputPolygon=Reference@xlink:href=
http%3A%2F%2Flocalhost%3A8082%2Fgeoserver%2Fows%3FSERVICE%3DWFS%26REQUEST%3DGetFeature
%26VERSION%3D1.0.0%26typename%3Dtopp%3Astates%26SRS%3DEPSG%3A4326%26FeatureID
%3Dstates.15&ResponseDocument=Result@mimeType=text/xml
```

Poiché è previsto nelle specifiche WPS, per ottenere solo i dati senza l'intero `ResponseDocument` è possibile richiedere un `RawDataOutput`. Per fare questo basta mettere nella richiesta `RawDataOutput` al posto di `ResponseDocument`, come mostrato di seguito:

```
http://localhost/cgi-bin/zoo_loader.cgi?
request=Execute&service=WPS&version=1.0.0&Identifier=Boundary&DataInputs=InputPolygon=Reference@xlink:href=
http%3A%2F%2Flocalhost%3A8082%2Fgeoserver%2Fows%3FSERVICE%3DWFS%26REQUEST%3DGetFeature
%26VERSION%3D1.0.0%26typename%3Dtopp%3Astates%26SRS%3DEPSG%3A4326%26FeatureID
%3Dstates.15&RawDataOutput=Result@mimeType=application/json
```

Si noti che si utilizza il default `mimeType` per ottenere direttamente la stringa JSON. Nella sessione successiva di questo workshop, questo tipo di richiesta verrà utilizzata per sviluppare una applicazione client.

3.3.1.3.2 Semplificazione e leggibilità delle richiesta

Come si è potuto vedere dal semplice esempio utilizzato fino ad ora, talvolta non è semplice fare la richiesta `Execute` usando il metodo GET, che genera URLs davvero lunghi e complessi. Nella prossima richiesta d'esempio, si userà in alternativa il metodo di richiesta POST XML. Di seguito è riportata la richiesta in XML, che corrisponde esattamente a quella fatta in precedenza con Execute:

```
<wps:Execute service="WPS" version="1.0.0" 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
../wpsExecute_request.xsd">
	<ows:Identifier>Boundary</ows:Identifier>
```

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

```
        <wps:DataInputs>
                <wps:Input>
                        <ows:Identifier>InputPolygon</ows:Identifier>
                        <ows:Title>Playground area</ows:Title>
                        <wps:Reference xlink:href="http://localhost:8082/geoserver/ows/?
SERVICE=WFS&amp;REQUEST=GetFeature&amp;VERSION=1.0.0&amp;typename=topp:states&amp;SRS=EPSG:43
26&amp;FeatureID=states.15"/>
                </wps:Input>
        </wps:DataInputs>
        <wps:ResponseForm>
                <wps:ResponseDocument>
                        <wps:Output>
                                <ows:Identifier>Result</ows:Identifier>
                                <ows:Title>Area serviced by playground.</ows:Title>
                                <ows:Abstract>Area within which most users of this playground will live.</ows:Abstract>
                        </wps:Output>
                </wps:ResponseDocument>
        </wps:ResponseForm>
</wps:Execute>
```

Per facilitare l'esecuzione della richiesta in XML, nella cartella `/var/www` è disponibile un semplice HTML form chiamato `test_services.html`. Vi si accede tramite il seguente link : http://localhost/test_services.html.

Aprire questa pagina in un browser. Mettere il contenuto della richiesta XML in un campo di tipo textarea e cliccare il pulsante d'invio « run using XML Request ». Si ottiene esattamente lo stesso risultato che si otterrebbe eseguendo il Servizio con la richiesta GET. La seguente schermata mostra il form HTML con la richiesta e il documento ExecuteResponse inseriti in un iframe in fondo alla pagina:

```
<wps:Execute service="WPS" version="1.0.0" 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 ../wpsExecute_request.xsd">
        <ows:Identifier>Boundary</ows:Identifier>
        <wps:DataInputs>
                <wps:Input>
                        <ows:Identifier>InputPolygon</ows:Identifier>
                        <ows:Title>Playground area</ows:Title>
                        <wps:Reference xlink:href="http://localhost:8082/geoserver/ows/?SERVICE=WFS&
amp;REQUEST=GetFeature&amp;VERSION=1.0.0&amp;typename=topp:states&amp;SRS=EPSG:4326&
amp;FeatureID=states.15"/>
                </wps:Input>
        </wps:DataInputs>
        <wps:ResponseForm>
                <wps:ResponseDocument>
                        <wps:Output>
                                <ows:Identifier>Result</ows:Identifier>
                                <ows:Title>Area serviced by playground.</ows:Title>
                                <ows:Abstract>Area within which most users of this playground will
live.</ows:Abstract>
                        </wps:Output>
                </wps:ResponseDocument>
        </wps:ResponseForm>
</wps:Execute>
```

Use your own XML resquest :

run using XML Request

```
-<wps:ExecuteResponse 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" serviceInstance="http://localhost/zoo//">
  -<wps:Process wps:processVersion="1">
     <ows:Identifier>Boundary</ows:Identifier>
     <ows:Title>Compute boundary.</ows:Title>
   -<ows:Abstract>
       A new geometry object is created and returned containing the boundary of the geometry on which the method is invoked.
     </ows:Abstract>
```

Il valore `xlink:href` è usato nella maniera più semplice per trattare i dati di input. Si può anche usare l'intera stringa JSON della geometria, come mostrato nel seguente esempio di richiesta XML:

```
<wps:Execute service="WPS" version="1.0.0" 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 ../wpsExecute_request.xsda">
        <ows:Identifier>Boundary</ows:Identifier>
        <wps:DataInputs>
                <wps:Input>
                        <ows:Identifier>InputPolygon</ows:Identifier>
                        <wps:Data>
                         <wps:ComplexData mimeType="application/json">
```

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

```
{ "type": "MultiPolygon", "coordinates": [ [ [ [ -105.998360, 31.393818 ], [ -106.212753,
31.478128 ], [ -106.383041, 31.733763 ], [ -106.538971, 31.786198 ], [ -106.614441,
31.817728 ],  [ -105.769730, 31.170780 ], [ -105.998360, 31.393818 ] ] ], [ [ [
-94.913429, 29.257572 ], [ -94.767380, 29.342451 ], [ -94.748405, 29.319490 ], [
-95.105415, 29.096958 ], [ -94.913429, 29.257572 ] ] ] ] }
                 </wps:ComplexData>
                </wps:Data>
                </wps:Input>
        </wps:DataInputs>
        <wps:ResponseForm>
                <wps:ResponseDocument>
                        <wps:Output>
                                <ows:Identifier>Result</ows:Identifier>
                                <ows:Title>Area serviced by playground.</ows:Title>
                                <ows:Abstract>Area within which most users of this
playground will live.</ows:Abstract>
                        </wps:Output>
                </wps:ResponseDocument>
        </wps:ResponseForm>
</wps:Execute>
```

Se tutto è andato a buon fine, si dovrebbe ottenere il Boundary della geometria JSON come argomento, e avere quindi conferma che il Servizio supporta GML e JSON come dati in input. Si noti che nella richiesta precedente si è aggiunto l'attributo `mimeType` al nodo `ComplexData` in modo da specificare che i dati di input non sono quelli di default `text/xml` `mimeType` ma direttamente una stringa `application/json`. È un procedimento simile a quello di aggiungere `@mimeType=application/json` che si è trattato in precedenza.

3.3.2 Creazione di Servizi per altre funzioni (ConvexHull e Centroid)

Ora che il codice d'esempio del servizio Boundary è pronto, è facile aggiungere le funzioni ConvexHull e Centroid, in quanto contengono esattamente lo stesso numero di argomenti: cioè una singola geometria. Di seguito è spiegato come implementare e distribuire il Servizio ConvexHull. Lo stesso procedimento può essere usato per implementare un Servizio Centroid.

3.3.2.1 Versione in C

Please add first the following code to the service.c source code :

```
int ConvexHull(maps*& conf,maps*& inputs,maps*& outputs){
   OGRGeometryH geometry,res;
   map* tmp=getMapFromMaps(inputs,"InputPolygon","value");
   if(tmp==NULL)
     return SERVICE_FAILED;
   map* tmp1=getMapFromMaps(inputs,"InputPolygon","mimeType");
   if(strncmp(tmp1->value,"application/json",16)==0)
     geometry=OGR_G_CreateGeometryFromJson(tmp->value);
   else
     geometry=createGeometryFromWFS(conf,tmp->value);
   res=OGR_G_ConvexHull(geometry);
   tmp1=getMapFromMaps(outputs,"Result","mimeType");
   if(strncmp(tmp1->value,"application/json",16)==0){
       addToMap(outputs->content,"value",OGR_G_ExportToJson(res));
       addToMap(outputs->content,"mimeType","text/plain");
   }
   else{
       addToMap(outputs->content,"value",OGR_G_ExportToGML(res));
```

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

```
    }
    outputs->next=NULL;
    OGR_G_DestroyGeometry(geometry);
    OGR_G_DestroyGeometry(res);
    return SERVICE_SUCCEEDED;
}
```

Questo nuovo codice è esattamente lo stesso di quello utilizzato per il Servizio Boundary. La sola cosa che è stata modificata è la linea dove si chiamata la funzione `OGR_G_ConvexHull` (invece della funzione `OGR_G_GetBoundary` usata in precedenza). È consigliabile non copiare e incollare l'intera funzione ma piuttosto trovare un modo più generico per definire il nuovo Servizio poiché il corpo della funzione sarà lo stesso in tutti i casi. La funzione generica proposta di seguito è pensata per rendere più semplice il procedimento:

```
int applyOne(maps*& conf,maps*& inputs,maps*& outputs,OGRGeometryH (*myFunc)(OGRGeometryH
)){
    OGRGeometryH geometry,res;
    map* tmp=getMapFromMaps(inputs,"InputPolygon","value");
    if(tmp==NULL)
      return SERVICE_FAILED;
    map* tmp1=getMapFromMaps(inputs,"InputPolygon","mimeType");
    if(strncmp(tmp1->value,"application/json",16)==0)
      geometry=OGR_G_CreateGeometryFromJson(tmp->value);
    else
      geometry=createGeometryFromWFS(conf,tmp->value);
    res=(*myFunc)(geometry);
    tmp1=getMapFromMaps(outputs,"Result","mimeType");
    if(strncmp(tmp1->value,"application/json",16)==0){
        addToMap(outputs->content,"value",OGR_G_ExportToJson(res));
        addToMap(outputs->content,"mimeType","text/plain");
    }
    else{
        addToMap(outputs->content,"value",OGR_G_ExportToGML(res));
    }
    outputs->next=NULL;
    OGR_G_DestroyGeometry(geometry);
    OGR_G_DestroyGeometry(res);
    return SERVICE_SUCCEEDED;
}
```

Inoltre è possibile utilizzare un puntatore di funzione chiamato `myFunc` invece che il nome completo della funzione. È possibile re-implementare il Servizio Boundary Service nel seguente modo:

```
int Boundary(maps*& conf,maps*& inputs,maps*& outputs){
      return applyOne(conf,inputs,outputs,&OGR_G_GetBoundary);
}
```

Usando la funzione locale `applyOne` definita nel codice sorgente `service.c` è possibile definire un altro Servizio nel seguente modo:

```
int ConvexHull(maps*& conf,maps*& inputs,maps*& outputs){
      return applyOne(conf,inputs,outputs,&OGR_G_ConvexHull);
}
int Centroid(maps*& conf,maps*& inputs,maps*& outputs){
      return applyOne(conf,inputs,outputs,&MY_OGR_G_Centroid);
}
```

La funzione `applyOne` è generica e questo permette di aggiungere due nuovo Servizi al Servizio Provider di ZOO: `ConvexHull` e `Centroid`.

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

La funzione `MY_OGR_Centroid` deve essere definita prima di quella Centroid poiché `OGR_G_Centroid` non restituisce una geometria di oggetti ma setta il valore ad uno esistente e supporta come input solamente Polygon. In tal modo è garantito l'utilizzo di ConvexHull per MultiPolygon. Si usi il seguente codice:

```
OGRGeometryH MY_OGR_G_Centroid(OGRGeometryH hTarget){
   OGRGeometryH res;
 res=OGR_G_CreateGeometryFromJson("{\"type\": \"Point\", \"coordinates\": [0,0] }");
 OGRwkbGeometryType gtype=OGR_G_GetGeometryType(hTarget);
 if(gtype!=wkbPolygon){
    hTarget=OGR_G_ConvexHull(hTarget);
 }
 OGR_G_Centroid(hTarget,res);
 return res;
}
```

Per propagare i Servizi, basta copiare il file metadata `Boundary.zcfg` dalla cartella `cgi-env` come `ConvexHull.zcfg` e `Centroid.zcfg`. Poi rinominare il Servizio name nella prima linea per lanciare e testare la richiesta `Execute` nello stesso modo fatto in precedenza. È sufficiente impostare nella richiesta il valore Identifier su ConvexHull o Centroid a seconda del servizio che si desidera eseguire.

Si noti che le richieste `GetCapabilities` e `DescribeProcess` restituiscono strani risultati se non si modificano le informazione sui metadati. È possibile andare a modificare il file .zcfg per impostare i valori corretti. In alternativa può essere impiegato a scopo di test, dal momento che i dati di input e di output hanno lo stesso nome e gli stessi formati supportati o di default.

3.3.2.2 Versione in Python

Come è stato fatto per la versione in C, si parte implementando il Servizio ConvexHull. Di seguito il codice per tale Servizio:

```
def ConvexHull(conf,inputs,outputs):
      if inputs["InputPolygon"]["mimeType"]=="application/json":
            geometry=osgeo.ogr.CreateGeometryFromJson(inputs["InputPolygon"]["value"])
      else:
            geometry=createGeometryFromWFS(inputs["InputPolygon"]["value"])
      rgeom=geometry.ConvexHull()
      if outputs["Result"]["mimeType"]=="application/json":
            outputs["Result"]["value"]=rgeom.ExportToJson()
            outputs["Result"]["mimeType"]="text/plain"
      else:
            outputs["Result"]["value"]=rgeom.ExportToGML()
      geometry.Destroy()
      rgeom.Destroy()
      return 3
```

Si ricorda che è possibile copiare e incollare la funzione preparata per l'esempio Boundary e andare a modificare soltanto la linea dove viene chiamato il metodo Geometry. In ogni caso, come fatto per la spiegazione del linguaggio in C, di seguito si riporta in maniera semplice un metodo generico.

La prima fase di estrazione della geometria InputPolygon è la stessa per qualsiasi funzione di Servizio. Pertanto si può creare una funzione che faccia questo automaticamente. La stessa cosa può essere fatta per riempire i valori di output, in modo che sia un'altra funzione a farlo automaticamente. Qui di seguito il codice per creare queste due funzioni (chiamate `extractInputs` e `outputResult`) :

```
def extractInputs(obj):
```

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

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

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

Il codice della funzione può essere semplificato usando la seguente definizione:

```
def Boundary(conf,inputs,outputs):
        geometry=extractInputs(inputs["InputPolygon"])
        rgeom=geometry.GetBoundary()
        outputResult(outputs["Result"],rgeom)
        geometry.Destroy()
        rgeom.Destroy()
        return 3
```

Si usa il seguente codice per richiamare i Servizi ConvexHull e Centroid:

```
def ConvexHull(conf,inputs,outputs):
        geometry=extractInputs(inputs["InputPolygon"])
        rgeom=geometry.ConvexHull()
        outputResult(outputs["Result"],rgeom)
        geometry.Destroy()
        rgeom.Destroy()
        return 3

def Centroid(conf,inputs,outputs):
        geometry=extractInputs(inputs["InputPolygon"])
        if geometry.GetGeometryType()!=3:
                geometry=geometry.ConvexHull()
        rgeom=geometry.Centroid()
        outputResult(outputs["Result"],rgeom)
        geometry.Destroy()
        rgeom.Destroy()
        return 3
```

Si noti che in Python è possibile usare ConvexHull anche per trattare poligoni multipli (MultiPolygons).

A questo punto occorre copiare il file `Boundary.zcfg` sia in `ConvexHull.zcfg` che in `Centroid.zcfg`, come spiegato nella versione in C. Poi, usando il comando `make install` si ridistribuisce e si testa il Services Provider.

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

3.3.3 Creare il Servizio Buffer

Ora si può lavorare sul Servizio Buffer, il che richiede più tempo rispetto agli altri. Infatti ha un codice diverso rispetto a quelli usati per implementare i Servizi Boundary, ConvexHull e Centroid.

Anche il Servizio Buffer richiede una geometria in ingresso, ma utilizza un parametro in più, la BufferDistance. È possibile definire un block `LitteralData` come semplice valore intero che rappresenta la BufferDistance. L'accesso alla lettura di questo tipo di valori viene concesso tramite la stessa funzione usata in precedenza.

3.3.3.1 Versione in C

Se si torna al codice sorgente del Servizio Boundary e lo si paragona a quello seguente si nota che era molto più semplice. Qui si deve aggiungere l'accesso dell'argomento `BufferDistance` e modificare la linea che chiama `OGR_G_Buffer` (invece di `OGR_G_GetBoundary`). Di seguito l'intero codice:

```
int Buffer(maps*& conf,maps*& inputs,maps*& outputs){
   OGRGeometryH geometry,res;
   map* tmp1=getMapFromMaps(inputs,"InputPolygon","value");
   if(tmp==NULL)
     return SERVICE_FAILED;
   map* tmp1=getMapFromMaps(inputs,"InputPolygon","mimeType");
   if(strncmp(tmp->value,"application/json",16)==0)
     geometry=OGR_G_CreateGeometryFromJson(tmp->value);
   else
     geometry=createGeometryFromWFS(conf,tmp->value);
   int bufferDistance=1;
   tmp=getMapFromMaps(inputs,"BufferDistance","value");
   if(tmp!=NULL)
       bufferDistance=atoi(tmp->value);
   res=OGR_G_Buffer(geometry,bufferDistance,30);
   tmp1=getMapFromMaps(outputs,"Result","mimeType");
   if(strncmp(tmp1->value,"application/json",16)==0){
       addToMap(outputs->content,"value",OGR_G_ExportToJson(res));
       addToMap(outputs->content,"mimeType","text/plain");
   }
   else{
       addToMap(outputs->content,"value",OGR_G_ExportToGML(res));
   }
   outputs->next=NULL;
   OGR_G_DestroyGeometry(geometry);
   OGR_G_DestroyGeometry(res);
   return SERVICE_SUCCEEDED;
}
```

Il nuovo codice deve essere inserito nel file `service.c` , ricompilato e sostituito alla vecchia versione del ZOO Service Provider nella cartella `/usr/lib/cgi-bin/`. Nella stessa cartella bisogna mettere ovviamente anche il suo corrispondente file ZOO Metadata.

Come spiegato in precedenza, ZOO Kernel è permissivo, in quanto permette di tralasciare molti argomenti che sono definiti nel file zcfg. Basta quindi rinominare col nome di `Buffer.zcfg` una copia del file `Boundary.zcfg` contenente l'identificatore `Buffer`. Poi occorre testare il servizio facendo una richiesta `Execute`, come è stato spiegato in precedenza. Si otterrà il buffer risultante nel ResponseDocument.

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

Il codice riportato sopra permette di controllare il valore del BufferDistance. Se non si setta il valore di default è 1.

è possibile cambiare il valore di BufferDistance settato dal Servizio in uno calcolato sulla geometria aggiungendo il valore `DataInputs` nella richiesta. Usando la sintassi KVP, ciascun `DataInputs` è separato in semicolonne.

Ecco come appare la richiesta precedente:

```
DataInputs=InputPolygon=Reference@xlink:href=http%3A%2F%2Flocalhost%3A8082%2Fgeoserver
%2Fows%3FSERVICE%3DWFS%26REQUEST%3DGetFeature%26VERSION%3D1.0.0%26typename%3Dtopp
%3Astates%26SRS%3DEPSG%3A4326%26FeatureID%3Dstates.15
```

Essa può essere riscritta in questo modo:

```
DataInputs=InputPolygon=Reference@xlink:href=http%3A%2F%2Flocalhost%3A8082%2Fgeoserver
%2Fows%3FSERVICE%3DWFS%26REQUEST%3DGetFeature%26VERSION%3D1.0.0%26typename%3Dtopp
%3Astates%26SRS%3DEPSG%3A4326%26FeatureID%3Dstates.15;BufferDistance=2
```

Attriduendo a `BufferDistance` il valore `2` viene generato un diverso risultato, pertanto nel codice sorgente non utilizzare gli altri parametri col valore di default (che ricordiamo essere `1`).

Qui di seguito la stessa query in formato XML, da lanciare dal form HTML http://localhost/test_services.html:

```
<wps:Execute service="WPS" version="1.0.0" 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 ../wpsExecute_request.xsda">
        <ows:Identifier>Buffer</ows:Identifier>
        <wps:DataInputs>
                <wps:Input>
                        <ows:Identifier>InputPolygon</ows:Identifier>
                        <ows:Title>Playground area</ows:Title>
                        <wps:Reference xlink:href="http://localhost:8082/geoserver/ows/?
SERVICE=WFS&amp;REQUEST=GetFeature&amp;VERSION=1.0.0&amp;typename=topp:states&amp;SRS=EPS
G:4326&amp;FeatureID=states.15"/>
                </wps:Input>
                <wps:Input>
                        <ows:Identifier>BufferDistance</ows:Identifier>
                        <wps:Data>
                             <wps:LiteralData uom="degree">2</wps:LiteralData>
                        </wps:Data>
                </wps:Input>
        </wps:DataInputs>
        <wps:ResponseForm>
                <wps:ResponseDocument>
                        <wps:Output>
                                <ows:Identifier>Buffer</ows:Identifier>
                                <ows:Title>Area serviced by playground.</ows:Title>
                                <ows:Abstract>Area within which most users of this
playground will live.</ows:Abstract>
                        </wps:Output>
                </wps:ResponseDocument>
        </wps:ResponseForm>
</wps:Execute>
```

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

3.3.3.2 Versione in Python

Poiché sono già pronte le funzioni utility `createGeometryFromWFS` e `outputResult`, il codice è semplice e può essere il seguente:

```
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
```

Semplicemente si è aggiunto l'uso di `inputs["BufferDistance"]["value"]` come argomento della Geometry invece del metodo `Buffer`. Una volta aggiunto questo codice al file `ogr_ws_service_provider.py` , basta copiarlo nella cartella del Kernel di ZOO (oppure scrivere `make install` dalla cartella root di ZOO Service Provider). Il file `Buffer.zcfg` sarà utilizzato anche nella sezione successiva.

3.3.3.3 Il file MetadataFile di Buffer

Si deve aggiungere il BufferDistance al file Metadata del Servizio per permettere al clients di riconoscere che il Servizio supporta questo parametro. Per fare questo, si deve fare una copia del file originale `Boundary.zcfg` e chiamarla `Buffer.zcfg`. Poi al `DataInputs` block si devono aggiungere le seguenti linee:

```
 [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>
```

Si noti che `minOccurs` è settato a `0` , il che significa che il parametro in input è opzionale e quindi non deve essere inserito. Dovete sapere che se è settato il valore di default verrà passato al Servizio per i parametri opzionali.

Al seguente indirizzo si trova una copia del file `Buffer.zcfg` :

http://zoo-project.org/trac/browser/trunk/zoo-services/ogr/base-vect-ops/cgi-env/Buffer.zcfg
Ora è possibile fare le richieste `GetCapabilities`, `DescribeProcess` e `Execute` al kernel di ZOO per il Servizio Buffer.

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

4 Costruire un client WPS usando OpenLayers

Il prossimo passo è di collegarsi da una mappa OpenLayers al Servizio OGR che si è creato. Questo permette di applicare processi di geometrie singole o multiple sui poligoni selezionati dall'utente e di mostrare le nuove geometrie generate.

4.1 Creare una semplice mappa che mostri il set di dati come un WMS

Nel DVD OSGeoLive è inclusa anche la distribuzione di default di OpenLayers, sufficiente per le nostre esigenze. Aprire un editor di testo e scrivere il seguente frammento di codice HTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"><html
xmlns="http://www.w3.org/1999/xhtml" xml:lang="EN" lang="EN">
 <head>
  <meta content="text/html; charset=UTF-8" http-equiv="content-type"/>
  <title>ZOO WPS Client example</title>
  <style>
  #map{width:600px;height:600px;}
  </style>
  <link rel="stylesheet" href="/openlayers/theme/default/style.css" type="text/css" />
  <script type="text/javascript" src="/openlayers/lib/OpenLayers.js"></script>
 </head>
 <body onload="init()">
  <div id="map"></div>
 </body>
</html>
```

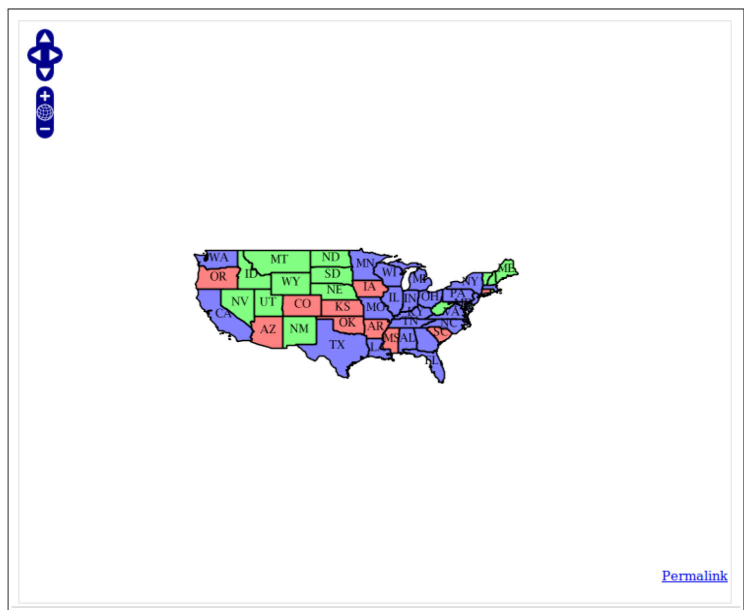
Aggiungere all'interno della sezione `<script></script>` con un `<head>` il seguente codice JavaScript. In questo modo verrà usata e visualizzata una mappa degli Stati Uniti come WMS.

```
var map, layer, select, hover, multi, control;

function init(){
  OpenLayers.ProxyHost= "../cgi-bin/proxy.cgi?url=";
  map = new OpenLayers.Map('map', {
    controls: [
      new OpenLayers.Control.PanZoom(),
      new OpenLayers.Control.Permalink(),
      new OpenLayers.Control.Navigation()
    ]
  });
  layer = new OpenLayers.Layer.WMS(
    "States WMS/WFS",
    "http://localhost:8082/geoserver/ows",
    {layers: 'topp:states', format: 'image/png'},
    {buffer:1, singleTile:true}
  );
  map.addLayers([layer]);
  map.zoomToExtent(new OpenLayers.Bounds(-140.444336,25.115234,-44.438477,50.580078));
}
```

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

Una volta fatto questo, salvare il file HTML col nome di `zoo-ogr.html` nella cartella workshop, poi copiarlo in `/var/www` e visualizzarlo con un browser Web browser usando questo link: http://localhost/zoo-ogr.html. Si dovrebbe ottenere una mappa centrata sugli USA con il layer WMS attivato.



4.2 Recuperare il layer dei dati come WFS e aggiungere il controllo di selezione

Prima di accedere alla visualizzazione dei dati tramite WFS, bisogna creare un nuovo layer vettoriale preposto a ospitare le molte interazioni che si andranno poi a creare. Aggiungere quindi le seguenti linee con la funzione `init()` e aggiungere il layer recentemente creato con il metodo map.addLayers:

```
select = new OpenLayers.Layer.Vector("Selection", {styleMap:
                new OpenLayers.Style(OpenLayers.Feature.Vector.style["select"])
            });
hover = new OpenLayers.Layer.Vector("Hover");

multi = new OpenLayers.Layer.Vector("Multi", {styleMap:
                new OpenLayers.Style({
                  fillColor:"red",
                  fillOpacity:0.4,
                  strokeColor:"red",
                  strokeOpacity:1,
                  strokeWidth:2
                })
            });

map.addLayers([layer, select, hover, multi]);
```

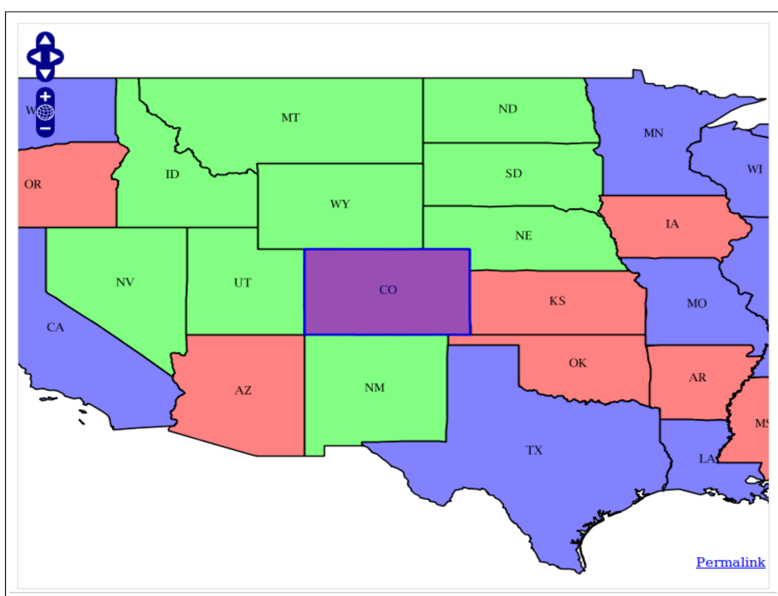
A questo punto si può accedere a tdata creando nuovi controlli per la selezionare dei poligoni, come mostrato di seguito. `OpenLayers.Protocol.WFS.fromWMSLayer(layer)` è usato per accedere alle geometrie e questi diversi stati di selezione sono dichiarati e aggiunti alla variabile `control`.

```
control = new OpenLayers.Control.GetFeature({
                protocol: OpenLayers.Protocol.WFS.fromWMSLayer(layer)
            });
```

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

```
control.events.register("featureselected", this, function(e) {
                select.addFeatures([e.feature]);
            });
control.events.register("featureunselected", this, function(e) {
                select.removeFeatures([e.feature]);
            });
control.events.register("hoverfeature", this, function(e) {
                hover.addFeatures([e.feature]);
            });
control.events.register("outfeature", this, function(e) {
                hover.removeFeatures([e.feature]);
            });
map.addControl(control);
control.activate();
```

Salvare nuovamente il file HTML. A questo punto è possibile selezionare un poligono semplicemente cliccandoci sopra. Il poligono selezionato dovrebbe essere visualizzato in blu.



4.3 Richiamare processi su geometrie singole con JavaScript

Ora che ogni cosa è settata, si può proseguire e richiamare i servizi OGR ZOO tramite JavaScript. Aggiungere le seguenti linee dopo la funzione `init()`, il che permetterà di chiamare i processi di geometria singola.

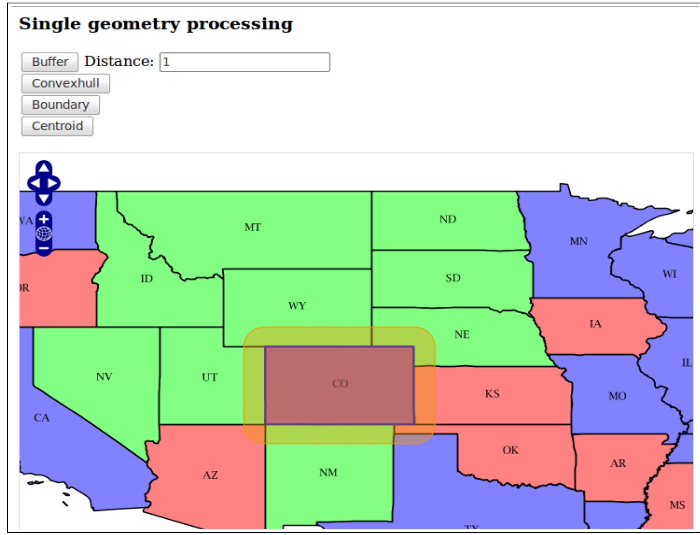
```
function simpleProcessing(aProcess) {
   if (select.features.length == 0)
    return alert("No feature selected!");
  var url = '/zoo/?request=Execute&service=WPS&version=1.0.0&';
  if (aProcess == 'Buffer') {
    var dist = document.getElementById('bufferDist').value;
    if (isNaN(dist))
      return alert("Distance is not a Number!");
    url+='Identifier=Buffer&DataInputs=BufferDistance='+dist+'@datatype=interger;InputPol
ygon=Reference@xlink:href=';
  } else
      url += 'Identifier='+aProcess+'&DataInputs=InputPolygon=Reference@xlink:href=';
  var xlink = control.protocol.url +"?SERVICE=WFS&REQUEST=GetFeature&VERSION=1.0.0";
  xlink += '&typename='+control.protocol.featurePrefix;
  xlink += ':'+control.protocol.featureType;
```

```
  xlink += '&SRS='+control.protocol.srsName;
  xlink += '&FeatureID='+select.features[0].fid;
  url += encodeURIComponent(xlink);
  url += '&RawDataOutput=Result@mimeType=application/json';
  var request = new OpenLayers.Request.XMLHttpRequest();
  request.open('GET',url,true);
  request.onreadystatechange = function() {
   if(request.readyState == OpenLayers.Request.XMLHttpRequest.DONE) {
      var GeoJSON = new OpenLayers.Format.GeoJSON();
      var features = GeoJSON.read(request.responseText);
      hover.removeFeatures(hover.features);
      hover.addFeatures(features);
    }
  }
  request.send();
}
```

A questo punto si possono aggiungere nel codice HTML pulsanti specifici, che chiamare i processi appena dichiarati. Aggiungerli in cima alla mappa scrivendo le seguenti linee prima di *<div id="map"></div>*.

```
<h3>Single geometry processing</h3>
<ul>
  <li>
    <input type="button" onclick="simpleProcessing(this.value);" value="Buffer" />
    <input id="bufferDist" value="1" />
  </li>
  <li>
    <input type="button" onclick="simpleProcessing(this.value);" value="ConvexHull" />
  </li>
  <li>
    <input type="button" onclick="simpleProcessing(this.value);" value="Boundary" />
  </li>
  <li>
    <input type="button" onclick="simpleProcessing(this.value);" value="Centroid" />
  </li>
</ul>
```

Salva nuovamente il file HTML. A questo punto è possibile selezionare un poligono e lanciarvi le funzioni Buffer, ConvexHull, Boundary o Centroid cliccando sul pulsante specifico. Il risultato del processo appare come un layer GeoJSON visualizzato col colore arancione sulla mappa.



4.4 Richiamare processi su geometrie multiple con JavaScript

Usando lo stesso procedimento, è possibile scrivere una funzione specifica per i processi di geometrie multiple. Aggiungere le seguenti linee dopo la funzione `simpleProcessing()`. Nella sezione 5 è spiegato come creare tale funzione.

```
function multiProcessing(aProcess) {
 if (select.features.length == 0 || hover.features.length == 0)
   return alert("No feature created!");
 var url = '/zoo/';
 var xlink = control.protocol.url +"?SERVICE=WFS&REQUEST=GetFeature&VERSION=1.0.0";
 xlink += '&typename='+control.protocol.featurePrefix;
 xlink += ':'+control.protocol.featureType;
 xlink += '&SRS='+control.protocol.srsName;
 xlink += '&FeatureID='+select.features[0].fid;
 var GeoJSON = new OpenLayers.Format.GeoJSON();
 try {
    var params = '<wps:Execute service="WPS" version="1.0.0"
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/../wpsExecute_request.xsd">';
    params += ' <ows:Identifier>'+aProcess+'</ows:Identifier>';
    params += ' <wps:DataInputs>';
    params += ' <wps:Input>';
    params += ' <ows:Identifier>InputEntity1</ows:Identifier>';
    params += ' <wps:Reference xlink:href="'+xlink.replace(/&/gi,'&amp;')+'"/>';
    params += ' </wps:Input>';
    params += ' <wps:Input>';
    params += ' <ows:Identifier>InputEntity2</ows:Identifier>';
    params += ' <wps:Data>';
    params += ' <wps:ComplexData mimeType="application/json">
'+GeoJSON.write(hover.features[0].geometry)+' </wps:ComplexData>';
    params += ' </wps:Data>';
    params += ' </wps:Input>';
    params += ' </wps:DataInputs>';
    params += ' <wps:ResponseForm>';
    params += ' <wps:RawDataOutput>';
    params += ' <ows:Identifier>Result</ows:Identifier>';
    params += ' </wps:RawDataOutput>';
    params += ' </wps:ResponseForm>';
    params += ' </wps:Execute>';
 } catch(e) {
    alert(e);
    return false;
 }
 var request = new OpenLayers.Request.XMLHttpRequest();
 request.open('POST',url,true);
 request.setRequestHeader('Content-Type','text/xml');
 request.onreadystatechange = function() {
   if(request.readyState == OpenLayers.Request.XMLHttpRequest.DONE) {
     var GeoJSON = new OpenLayers.Format.GeoJSON();
     var features = GeoJSON.read(request.responseText);
     multi.removeFeatures(multi.features);
     multi.addFeatures(features);
   }
 }
 request.send(params);
}
```

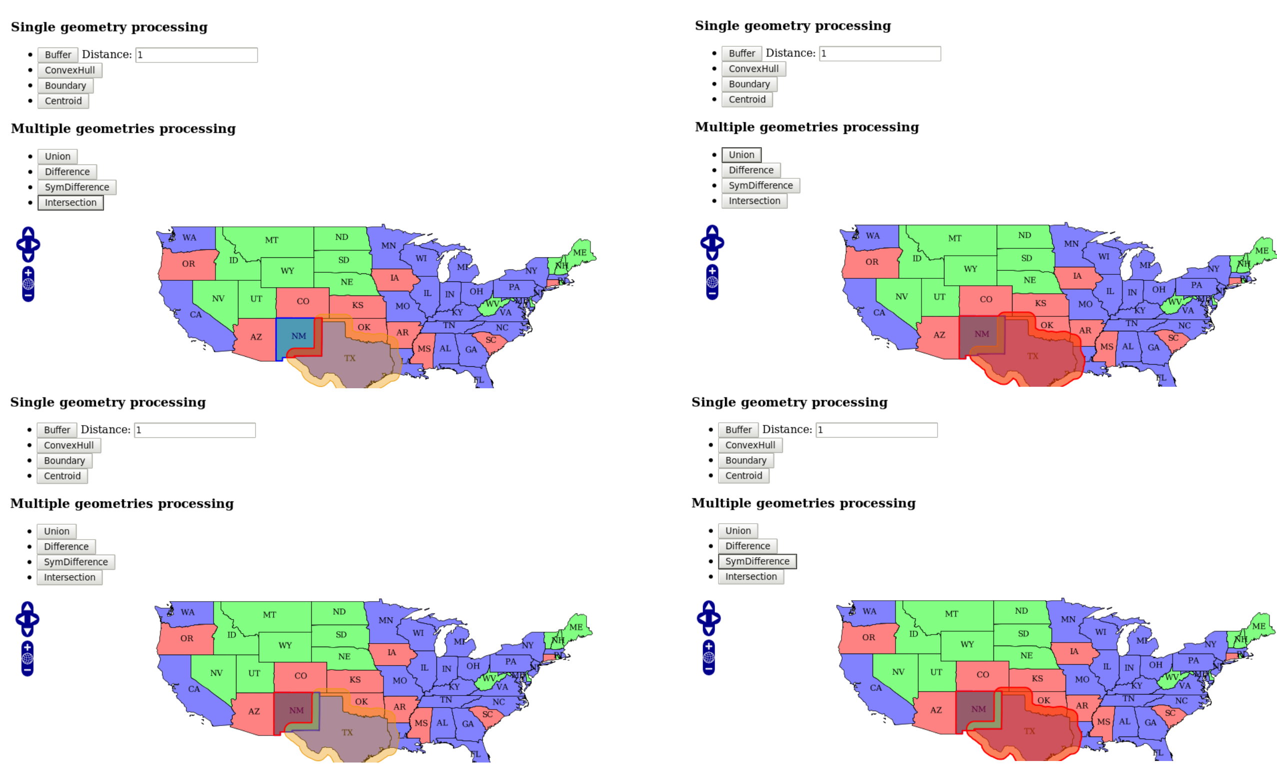
Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

In questo caso non si usa il metodo GET per fare la richiesta allo ZOO Kernel ma un XML POST. Questo perché se si usa il metodo GET si genera un errore dovuto alla limitazione dell'HTTP GET sulla lunghezza della richiesta. Se si usa una stringa JSON per rappresentare la geometria, non si hanno limitazioni di lunghezza della richiesta.

Ora che si dispone delle funzioni per richiamare i processi di geometrie multiple, bisogna aggiungere i pulsanti per eseguire tali processi. Di seguito il codice HTML da aggiungere al file `zoo-ogr.html` :

```
<h3>Multiple geometries processing</h3>
<ul>
  <li>
    <input type="button" onclick="multiProcessing(this.name);" value="Union"/>
  </li>
  <li>
    <input type="button" onclick="multiProcessing(this.name);" value="Difference"/>
  </li>
  <li>
    <input type="button" onclick="multiProcessing(this.value);" value="SymDifference"/>
  </li>
  <li>
    <input type="button" onclick="multiProcessing(this.name);" value="Intersection"/>
  </li>
</ul>
```

Ricaricare la pagina. Ora è possibile eseguire i servizi di geometria multipla e visualizzare il risultati in rosso, come mostrato nel seguente screenshots:



è evidente che manca qualcosa nel Service Provider che permetta di ottenere lo stesso risultato… Si tratta del Servizio per le geometrie multiple! Ed è proprio quello che si andrà ad analizzare nella sezione successiva.

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

5 Esercizio

Ora si dispone delle competenze per scrivere il file metadata zcfg e le parti di codice in `service.c` o `ogr_service_provider.py` , a seconda se si è scelto di usare il linguaggio di programmazione C o Python.

L'obiettivo di questo esercizio è di sviluppare i seguenti servizi di geometria multipla:

- ✔ Intersection
- ✔ Union
- ✔ Difference
- ✔ SymDifference

5.1 Versione in C

Si deve modificare il file `source.c` creato nel corso della lezione per aggiungere le geometrie multiple, usando le seguenti funzioni OGR C-API:

- ✔ `OGR_G_Intersection(OGRGeometryH, OGRGeometryH)`
- ✔ `OGR_G_Union(OGRGeometryH, OGRGeometryH)`
- ✔ `OGR_G_Difference(OGRGeometryH, OGRGeometryH)`
- ✔ `OGR_G_SymmetricDifference(OGRGeometryH, OGRGeometryH)`

Si può usare come esempio il file `Boundary.zcfg` , rinominando il dato di input `InputPolygon` in `InputEntity1` e aggiungendone un altro identico chiamandolo `IntputEntity2`. Si consiglia di inserire valori diversi nel file ZOO Metadata in modo da impostare le adeguate informazioni di metadati.

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

5.2 Versione in Python

Si vada a modificare il file `ogr_ws_service_provider.py` creato durante la lezione per aggiungere le geometrie multiple. Si può riadattare il metodo `osgeo.ogr` creato per la Geometry alle seguenti funzioni:

- ✔ `Intersection(Geometry)`
- ✔ `Union(Geometry)`
- ✔ `Difference(Geometry)`
- ✔ `SymmetricDifference(Geometry)`

Per svolgere questo compito, si può di nuovo riusare il file `Boundary.zcfg`, rinominando il dato di input `InputPolygon` in `InputEntity1` e aggiungendone uno identico chiamato `IntputEntity2`. Si consiglia di inserire valori diversi nel file ZOO Metadata in modo da impostare le adeguate informazioni di metadati.

5.3 Testare i Servizi

Una volta distribuiti i Servizi delle geometrie multiple in ambiente locale, ricaricare nel browser la pagina `zoo-ogr.html` creata nella sezione precedente e testare il nuovo Servizio ZOO.

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**

Restate sintonizzati!

ZOO Official website: **http://www.zoo-project.org**

ZOO Discuss mailing list: **zoo-discuss@gisws.media.osaka-cu.ac.jp**

ZOO IRC Chanel: **irc://irc.freenode.net/zoo-project**

ZOO Twitter: **http://www.twitter.com/ZOO_Project**

ZOO Linkedin Group: **http://www.linkedin.com/groups?home=&gid=2532284**

Pubblicato sotto CC-BY-SA da **GeoLabs** e **3LIZ**. Tradotto da **Fondazione Edmund Mach**