SERVEI DE SISTEMES
D'INFORMACIÓ GEOGRÀFICA
I TELEDETECCIÓ
Universitat de Girona

VII JORNADAS DE SIG LIBRE

Avances en la integración de GGL2 con gvSIG y Qgis

V. González [(1)], F. González [(2)]

[(1)] Freelance, geomati.co, fernando.gonzalez@geomati.co
[(2)] Freelance, victor.gonzalez@geomati.co

RESUMEN

GGL2 es un lenguaje específico para geoprocesamiento que a lo largo del último año ha sufrido diversos avances bastante notorios, tanto en lo referente a la usabilidad del propio lenguaje, como en cuanto a su integración con otras herramientas. Desde el punto de vista del lenguaje, se han producido avances fundamentalmente en cuanto al rendimiento de la instrucción join, puesto que antes era inutilizable. Si bien es cierto que todavía sería posible reducir notablemente el tiempo, el rendimiento actual ya es aceptable. En lo que respecta a la integración con gvSIG se han producido importantes mejoras. La primera y principal es que ya es posible ejecutar código GGL2 directamente desde gvSIG a través de una consola integrada dentro del propio gvSIG. Dicha consola hace uso de las herramientas de línea de comandos de GGL2, lo cual hace que la integración sea compatible con otras aplicaciones como Qgis. Además, aprovechando la ventaja comentada en el párrafo anterior, la misma integración que con gvSIG se ha realizado con Qgis de forma que también es posible utilizar GGL2 directamente desde Qgis para escribir y leer datos de forma sencilla a través de una consola. De esta manera se consigue que un usuario pueda realizar los mismos procesos con la misma consola y prácticamente la misma apariencia en ambas aplicaciones. Por último, cabe destacar que sigue siendo posible utilizar el entorno de GGL2 de forma independiente, aprovechando así todas las ventajas del editor y pudiendo interactuar con gvSIG y Qgis simultáneamente. En conclusión, en el presente artículo se analizarán todas las mejoras realizadas a GGL2, cómo afectan al usuario y cuáles son las líneas de desarrollo para el futuro.

ABSTRACT

GGL2 is a domain-specific language for geoprocessing that has evolved noticeably during the last year. This progress involve not only the language itself but also the integration of the language with other GIS tools.

Regarding the language, several developments has been made to the join instruction in order to improve its performance since it was unusable. Although it is still possible to reduce the response time, the current performance is already acceptable.

As far as the gvSIG integration is concerned, several improvements has been also performed. First and most important is that it is already possible to execute GGL2 code directly from gvSIG through a built-in console. That console uses the GGL2 command line tools, which makes it compatible with other GIS tools, such as Quantum GIS.

Moreover, taking advantage of the feature mentioned in the previous paragraph, the same integration as in gvSIG has been developed for Quantum GIS so it is possible to use GGL2 directly from Quantum GIS in order to read and write data easily through the console. Thus, it makes possible to execute the same geoprocesses using the same console and virtually the same graphic user interface from both applications.

Finally, it is important to notice that it is still possible to use the GGL2 environment independently, making the most of all the advantages of the editor and being able to interact with gvSIG and Quantum GIS at the same time.

In conclusion, in this paper all the advantages and developments in GGL2 will be analyzed in terms of the user experience and some future work will be described.

INTRODUCCIÓN

En el presente artículo se van a mostrar los principales avances que ha experimentado GGL2 a lo largo del último año y cómo estos avances afectan al usuario en lo que respecta a eficiencia y usabilidad.

De esta manera, por una parte comentaremos todos los cambios tanto internos como de sintaxis, nuevas librerías y demás desarrollos que presenta GGL2 con respecto a versiones anteriores y cuantificaremos la mejora en la eficiencia que suponen estos cambios.

Por otra parte, comentaremos los avances en una de las principales características del lenguaje: la integración con otras herramientas con el fin de geoprocesar los datos ya existentes en dichas herramientas.

Finalmente extraeremos algunas conclusiones sobre todos estos avances y enunciaremos cuáles son las líneas de trabajo para el futuro.

AVANCES DEL LENGUAJE

Instrucción *join*

Uno de los principales problemas que presentaba GGL2 era la eficiencia y usabilidad de las instrucciones de tipo ***join***. Estas instrucciones, además de ser muy comunes, implican el cruce de datos entre dos tablas o, en general, secuencias de datos de cualquier tipo. Es por esto que era imperativo para GGL2 dotar a la instrucción ***join*** tanto de una sintaxis lógica y coherente con el resto del lenguaje como de una eficiencia suficiente para que el tiempo necesario para cruzar dos secuencias de datos grandes no fuera tan alto que lo hiciera inviable.

En lo que respecta a la sintaxis de la instrucción, inicialmente contábamos con la siguiente definición:

```
<exp1> [as <alias1>] join <exp2>
       [as <alias2>] on (<condition>);
```

donde <*exp1*> y <*exp2*> representan las dos secuencias de datos y <*alias1*>, <*alias2*> y <*condition*> son autoexplicativos.

De esta manera era posible cruzar los datos de dos secuencias pero con varios problemas. El primero de ellos era que si las secuencias de datos compartían un campo con el mismo nombre, como es muy común con *the_geom* en secuencias provenientes de *shapefiles*, el resultado incluía dos campos con el mismo nombre y resultaba en un error de compilación.

Por otro lado, el resultado siempre incluía todos los campos de ambas secuencias, cuando en muchos casos únicamente se necesitan algunos de ellos o, por el contrario, se requiere algún campo nuevo resultado del cálculo de alguno de los existentes.

Además, la sintaxis no era coherente con el resto de instrucciones SQL (***select***, ***filter***, …) del lenguaje, que seguían el siguiente patrón:

```
<expression> <instruction_sql> (<alias> | <operators>);
```

Para solucionar todo esto se ha modificado la sintaxis de la siguiente manera:

```
<exp1> join <exp2> (<alias 1>, <alias2> |
                    on <condition> include <operators>);
```

donde *<exp1>, <exp2>, <alias1> y <alias2>* mantienen el mismo significado que en la sintaxis anterior. Además, se ha añadido la posibilidad de incluir campos de una y otra secuencia de datos. Incluso es posible especificar de manera sencilla todos los campos de la primera (***left***) o la segunda (***right***) secuencia o ambos (***all***). Veamos algunos ejemplos en los que se cruza una secuencia de aeropuertos con una secuencia de paises:

```
result = aeropuertos join paises (a, p |
      on a/country == p/codigo
      include p/poblacion, all left);
```

En este primer ejemplo se obtienen todos los campos de la secuencia aeropuertos, además de la población del país en el que se ubica el aeropuerto.

```
result = aeropuertos join paises (a, p |
      on a/country == p/codigo
      include a/airport_code, all right);
```

En el segundo caso el resultado incluye únicamente el código del aeropuerto acompañado de toda la información del país en el que se encuentra dicho aeropuerto.

```
result = aeropuertos join paises (a, p |
      on a/country == p/codigo
      include all);
```

Por último, el resultado del ejemplo incluye todos los campos de ambas secuencias allí donde el código del aeropuerto coincide con el código del país. Dicho de otra manera, se obtendrá una secuencia donde cada elemento tendrá todos los campos del aeropuerto junto con todos los campos del país al que pertenece.

Por otra parte, el tiempo necesario para cruzar dos secuencias de datos de un tamaño considerable era demasiado alto como para poder considerar GGL2 una alternativa viable en estos casos. Esto era debido a que es necesario recorrer todos los elementos de ambas secuencias y para ello se realizaban aperturas y cierres de ficheros totalmente innecesarios, lo cual aumentaba de manera considerable el tiempo de ejecución de estas instrucciones. La solución a este problema ha consistido en mantener abiertos los ficheros que participan en la ejecución del proceso GGL2 mientras sea necesario, disminuyendo así al mínimo el número de aperturas y cierres de fichero.

Con el fin de cuantificar la aceleración en las instrucciones ***join*** tras las mejoras mencionadas, se ha realizado un estudio en el cual se han medido los tiempos obtenidos al realizar la misma instrucción con diferente tamaño en los datos de entrada. Así, el experimento consiste en cruzar una secuencia de datos con 16 polígonos con otra secuencia que irá variando de tamaño desde 1000 hasta 520.000 puntos, aproximadamente. La instrucción en la versión de GGL2 de 2012 es la siguiente:

```
result = points prefix 'a_' as a join germany as b
            on (ST_Intersects(a/a_the_geom, b/the_geom));
```

, mientras que la misma funcionalidad con la versión de GGL2 de 2013 es:

```
result = points join germany (a, b |
            on ST_Intersects(a/the_geom, b/the_geom)
                  include all left);
```

Cabe destacar que el resultado obtenido no es exactamente el mismo, ya que la segunda instrucción (2013) incluye únicamente los campos de la primera secuencia (***include all left***), mientras que en la primera instrucción (2012) se incluyen todos los campos siempre, haciéndose necesario el uso de prefijos (***prefix***) para evitar los conflictos con los campos que comparten el mismo nombre (*the_geom*). Para obtener con la versión de 2012 el mismo resultado sería necesario aplicar después una instrucción ***select***, lo cual aumentaría todavía más el tiempo necesario.

En la Tabla 1 se pueden observar los resultados obtenidos para ambas versiones en función del número de puntos de la secuencia. En dicha tabla se puede observar claramente cómo en la nueva versión se disminuye de manera drástica el tiempo, tardando prácticamente lo mismo en cruzar una secuencia con más de medio millón de puntos que lo que tardaba la versión anterior en cruzar una secuencia con un número de puntos diez veces menor.

Además, también es destacable el hecho de que a partir de 100.000 puntos, la versión anterior era incapaz de realizar la operación, obteniéndose un error por falta de memoria. En la nueva versión se ha solucionado también dicho problema, pudiendo llegar a cruzarse capas, teniendo una de ellas más de medio millón de puntos.

Tabla 1: Comparativa de tiempos de la instrucción ***join***

Nº elementos	Versión 2012	Versión 2013
966	10.79	3.67
4928	30.75	7.85
10539	53.95	12.37
20361	106.84	14.87
30142	148.42	18.38
40307	188.71	22.80
50669	218.65	28.39
101285	*Error*	51.88
200024	*Error*	87.44
300040	*Error*	131.89
400415	*Error*	172.50
520230	*Error*	217.53

En al Figura 1 es posible apreciar de manera gráfica dichos resultados. En ella podemos observar que la curva que describe la versión de 2012 hasta ser incapaz de manejar los datos con 100.000 puntos nada tiene que ver con la que describe la versión de 2013 que hasta los 50.000 puntos es cercana a una recta con una inclinación muy pequeña.

También es posible observar que el comportamiento exponencial que aparece en la curva de 2012 hacia el final es similar al que presenta la versión de 2013 para secuencias con un número de puntos diez veces mayor.

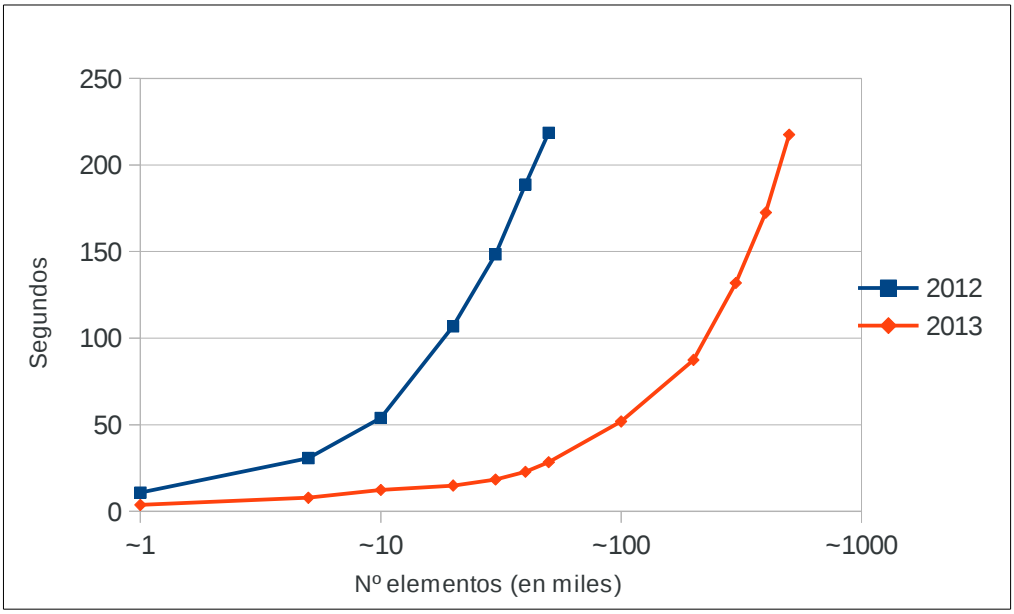

Figura 1: *Gráfica comparativa de tiempos de la instrucción* ***join***

Nuevas librerías: *vector2d* y *statistics*

Además de los avances ya mencionados en la instrucción ***join***, también se han añadido nuevas librerías al conjunto que proporciona GGL2 por defecto. En concreto, se han añadido dos librerías nuevas.

La primera de ellas, ***vector2d***, tiene por objetivo el tratamiento de vectores de dos dimensiones con operaciones como crear un nuevo vector, obtener el vector unitario o aplicar un vector unitario a una determinada coordenada. Por ejemplo, es posible aplicar un vector unitario con un ángulo de 0.2 radianes al punto (2, 0) con una distancia de 100 unidades de la siguiente manera:

```
unitVector = buildUnitVector2D(0.2);
show applyVector(unitVector, POINT(2 0), 100);
```

Un ejemplo de aplicación de esta librería es la monitorización de la migración de animales (*radio tracking*), donde los animales llevan un emisor que envía una señal a varios receptores que indican el ángulo desde el que proviene la señal. De esta manera, contando con al menos dos receptores, es posible crear dos vectores unitarios, crear dos rectas a partir de dichos vectores y hallar la intersección para obtener la ubicación del animal monitorizado.

Por otra parte, se ha añadido la librería ***statistics***, con la que es posible obtener diferentes valores estadísticos a partir de una secuencia de valores numéricos. Por ejemplo, es posible medir la desviación típica de un conjunto de valores de la siguiente manera:

```
valores = [47.2, 2.51, -73.23, 0, 185.334]
show standardDeviation(valores);
```

Un ejemplo de aplicación de esta librería es la validación de indicadores urbanísticos, donde diferentes medidas sobre edificios en un determinado distrito o barrio vienen representadas por un único valor estimado para dicho distrito. Con la librería ***statistics*** es posible obtener medidas de dispersión sobre los valores

medidos en los edificios de un barrio, comparar los resultados con el valor estimado y realizar las correcciones oportunas.

INTEGRACIÓN CON OTRAS HERRAMIENTAS

Además de los desarrollos realizados sobre el núcleo de GGL2, también se han dedicado esfuerzos a la integración con otras herramientas SIG, como *gvSIG* y *Quantum GIS*.

Desde el inicio, GGL2 está diseñado para interactuar con diferentes aplicaciones SIG con el fin de procesar sus datos y mostrar los resultados de manera automática en el mismo SIG. Sin embargo, esta interacción requería tener abiertas dos aplicaciones por separado: la aplicación SIG y el editor de GGL2, lo cual hacía algo tedioso el hecho de trabajar con GGL2.

Es por esto que uno de los principales avances de GGL2 a lo largo del último año es la integración de una consola donde poder escribir código GGL2 dentro de la propia aplicación SIG, de forma que únicamente es necesario tener abierta dicha aplicación para procesar sus datos con GGL2.

También es destacable el desarrollo realizado con el fin de agilizar la compilación y ejecución de código GGL2 desde la aplicación SIG, que es bastante más rápido que la compilación y ejecución desde la línea de comandos. Esto es debido a que el tiempo de inicialización de GGL2 es alto y es necesario cada vez que se quiere compilar o ejecutar. Sin embargo, desde la aplicación SIG se ha conseguido que el tiempo de inicialización sea necesario únicamente en la primera ejecución, manteniendo un servidor de compilación/ejecución que recibe las peticiones de compilación y/o ejecución y devuelve el resultado sin necesidad de inicializar GGL2 cada vez.

De esta manera se consigue que el tiempo de respuesta sea prácticamente idéntico al que se percibe desde el editor de GGL2. En concreto, el tiempo viene dado por la siguiente fórmula:

$$t = x + c$$

donde *x* es el tiempo de ejecución propio del proceso en sí, y *c* es el tiempo añadido por la comunicación con el servidor de compilación. Así, en la primera ejecución *c* es de ~20 s. debido a la inicialización, mientras que en las siguientes ejecuciones es únicamente de ~4s.

Es importante destacar que, a pesar de haber integrado GGL2 con varias aplicaciones SIG, sigue siendo posible utilizar el editor de GGL2, que proporciona múltiples ventajas como puede ser la autocompleción, la coloración sintáctica y multitud de herramientas para el manejo de ficheros.

Quantum GIS

En el caso de *Quantum GIS*, tanto la consola como el diálogo de configuración se pueden abrir desde el menú *Plugins*. Una vez abierta la consola, la interfaz gráfica quedará como se puede observar en la Figura 2. En dicha figura se puede apreciar que la consola se integra dentro de Quantum GIS de manera natural utilizando el sistema de *docking windows* propio de la aplicación. Además, se puede observar que la consola tiene dos pestañas distintas para diferenciar el código GGL2 de la salida del geoproceso. Por otro lado, la consola proporciona una barra de herramientas con

la que se puede ejecutar o borrar el geoproceso, recuperar un geoproceso ya ejecutado o acceder directamente a la configuración del *plugin*. Por último, es importante que en la barra de herramientas de *Quantum GIS* esté activada la conexión con GGL2 (cuarto icono empezando por la derecha).

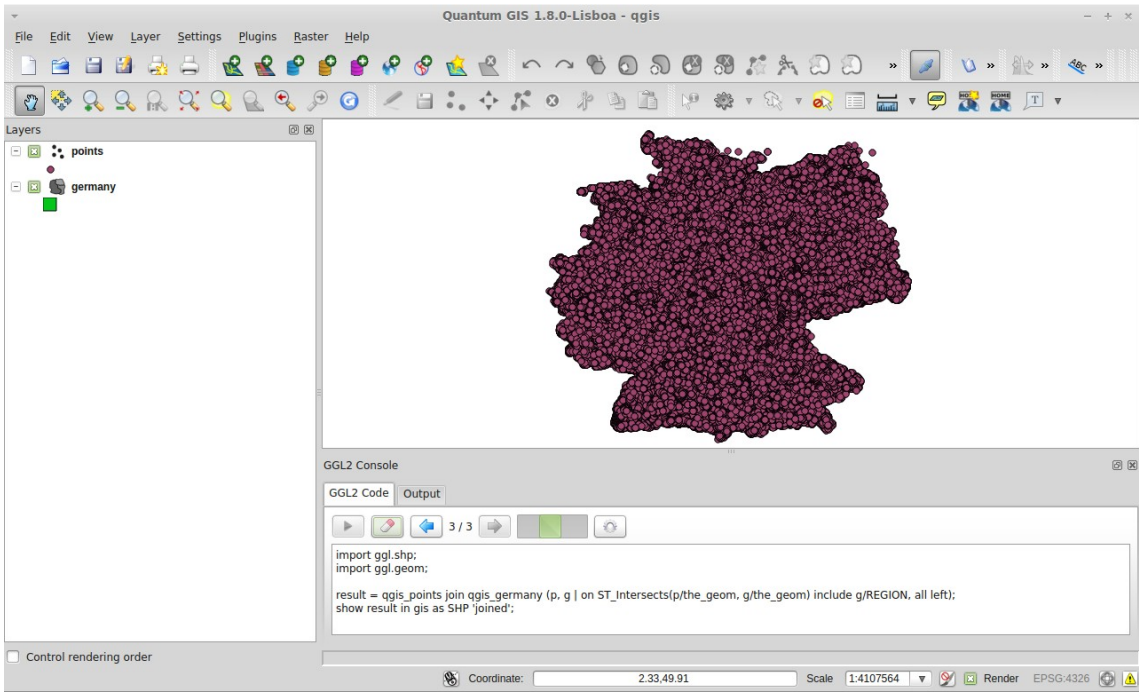

Figura 2: *Interfaz gráfica de GGL2 en Quantum GIS.*

De lo contrario, las capas de *Quantum GIS*, accesibles como variables con el prefijo '*qgis*' no estarán disponibles. Esto es debido a que existe una comunicación entre la aplicación GIS y GGL2 que requiere un tiempo que, a pesar de no ser elevado, se puede omitir en los casos en que no se vaya a interactuar con GGL2.

gvSIG

Por su parte, *gvSIG* proporciona una interfaz gráfica muy similar a la de *Quantum GIS* pero con algunas diferencias que se pueden observar en la Figura 3. En este caso, el código GGL2 y la salida del geoproceso se separan sin pestañas, mediante paneles distintos dentro de la consola.

En cuanto a la barra de herramientas de la consola es idéntica a la de *Quantum GIS*, siendo completamente independientes para el histórico los geoprocesos ejecutados en *Quantum GIS* y en *gvSIG*.

Además, de la misma manera que en *Quantum GIS*, es necesario que el botón de la barra de herramientas (segundo empezando por la derecha) esté activado para que la conexión entre el SIG y GGL2 funcione correctamente.

Por último es destacable el hecho de que *gvSIG* proporciona dos menús desplegables. Desde el primero de ellos se pueden seleccionar las capas disponibles en *gvSIG* para que se escriba en el código GGL2 la variable asociada de manera automática. Desde el segundo es posible seleccionar varias plantillas con geoprocesos de ejemplo que se mostrarán en la consola de manera automática.

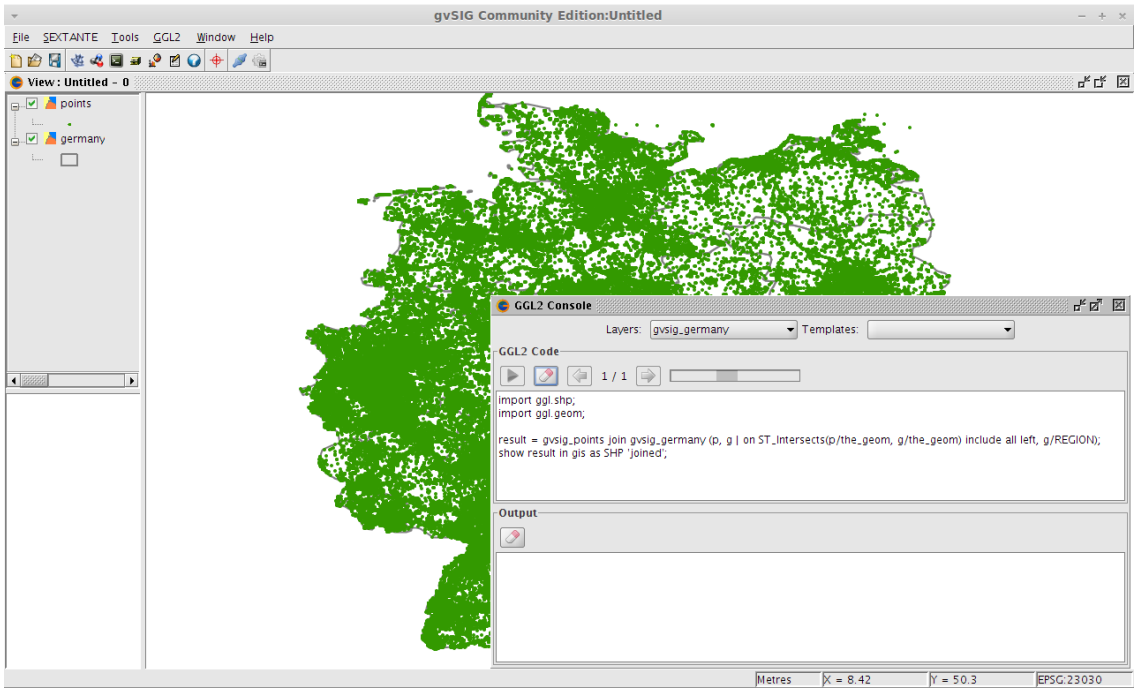

Figura 3: *Interfaz gráfica de GGL2 en gvSIG.*

CONCLUSIONES

En conclusión, todos los avances que ha experimentado GGL2 a lo largo del último año se han realizado teniendo como objetivo mejorar la experiencia del usuario en su utilización de GGL2.

Así, por una parte se han realizado cambios que afectan fundamentalmente al rendimiento de la ejecución como es el caso de la instrucción ***join*** o el servidor de compilación/ejecución de los *plugins* de *gvSIG* y *Quantum GIS*. Y, por otra parte, se han introducido cambios destinados a una mayor usabilidad de la herramienta, como son los cambios sintácticos de la instrucción ***join*** o la completa integración del lenguaje en las aplicaciones SIG mencionadas.

En lo que respecta al trabajo futuro, se contemplan tres líneas de trabajo principales. La primera de ellas es la integración del GGL2 con SEXTANTE en Quantum GIS, de forma que sea posible incluir procesos escritos en GGL2 desde el toolbox de SEXTANTE y ejecutarlos de manera sencilla con un asistente sin necesidad de conocer el código GGL2.

Por otro lado, una de las últimas capacidades adquiridas por GGL2 fue el soporte raster, por lo que es todavía insuficiente para muchos de los casos. Una de las principales limitaciones es el hecho de que únicamente soporta rasters monobanda, por lo que una de las principales líneas de trabajo consiste en dotar a GGL2 de soporte raster multibanda.

Por último, el conjunto de librerías de GGL2, a pesar de cubrir las necesidades básicas para la mayoría de geoprocesos, es bastante limitado. Por ello, una de las líneas de trabajo futuro es dotar a GGL2 de nuevas librerías por defecto para realizar tareas más específicas que sean capaces de facilitar la programación de geoprocesos más complejos.