GAL Framework – Current State of the Project

Radek Bartoň, Martin Hrubý
Faculty of Information Technology
Brno University of Technology
E-mail: `xbarto33@stud.fit.vutbr.cz`, `hrubym@fit.vutbr.cz`

Keywords: design, GIS, GRASS, open source, library, dynamic language, remote procedure call

Abstract

The GAL (GIS Abstraction Layer) Framework is a component-architecture-oriented[1] remote procedure call (RPC) library with implementations of GIS-related subsystems communicating using the library and a set of demonstrational and testing tools utilizing that services. It doesn't aim to be a full-featured solution for GIS application construction but a proposal for possible incremental GRASS GIS[2] modernization. This article summarizes current state of the project, it's history, application and potential and also presents options for further advancement and areas of possible participation. Only a concern of other developers or users and the time may transform this idea into something practically usable.

History and Motivation

The project was originated as an article author's master degree diploma thesis at the Faculty of Information Technology of the Brno University of Technology in February 2007. It was intended to be a higher-level abstraction layer above GRASS GIS core libraries from the beginning allowing rapid and clear GRASS module development. It also allows sequential exchange of the current implementations with the new ones if used communication interfaces

[1] `http://trac.edgewall.org/wiki/TracDev/ComponentArchitecture`
[2] `http://grass.osgeo.org/`

would be well-designed and preserved. This could help during possible GRASS GIS innovation procedure. Support of distributed computing and dynamic language facilitation was contemplated too.

An initial stage of project realization was to design core communication mechanisms and lasted until July 2007 when the first steps to implement them was started. The library design was introduced on the last year's volume[3] of Geoinformatics FCE CTU Workshop. Further information about project creation motivation in consequence to GRASS's internal organization was discussed there also.

Main development of the framework, including the design of introductory general-purpose, raster display and raster processing interfaces, was performed during the first half of year 2008 until the end of May when the project was presented in front of a diploma thesis commission. But the development did not stop since then and it may continue further if there will be enough of interest.

Current State

The library is divided into several subsystems which are developed in parallel to allow implementation of certain features of example tools. These are mainly but not lastly a reimplementation of `d.mon` module functionality and a real-time 3D visualization tool called `d.roamer` similar to the `nviz` but with emphasis on interactivity. This paragraph will tell a few words about progress of each of the subsystems; designed interfaces and implemented modules are discussed in next paragraphs.

Generally can be said about these subsystems that GRASS's libraries has been used in their implementation everywhere it was feasible but a possibility of their replacement with different implementations has always been kept in mind.

Core Subsystem

This part of library defines basic ways of communication between the components through the interfaces, abstracts used event processing libraries to a single event loop and provides a general model for RPC based subsystems such as a D-Bus[4] subsystem is. What do the „component“ and the „interface“ terms mean in context of the GAL Framework and what is the „component architecture“ was explained the last year[5] or can be found in this document[6].

The core subsystem is naturally the most evolved part of the framework. Only things that should be done here are a proper event processing loop implementation since current one is quite naive and a user (module programmer) comfortance improvements which are not crucial in this stage of evolvement.

Exception Subsystem

[3]`http://geoinformatics.fsv.cvut.cz/gwiki/GAL_Framework`
[4]`http://www.freedesktop.org/wiki/Software/dbus`
[5]`http://geoinformatics.fsv.cvut.cz/gwiki/GAL_Framework`
[6]`http://trac.edgewall.org/wiki/TracDev/ComponentArchitecture`

It contains an exception objects' class hierarchy so far. The exceptions are generally used as the only one mechanism for an error state signalization occured during the communication between the components.

A local exception evocation and processing is provided natively by GCC but an exception passage through D-Bus message bus is not working yet.

D-Bus Subsystem

The only one RPC communication implementation present is the D-Bus subsystem. The D-Bus library was chosen because of its simplicity and desktop systems orientation, but it'll be probably replaced with an ORBit2 implementation of a CORBA architecture in the future for its robustness.

Current implementation allows only single process act as a server which provides components with interface implementations. This have to be changed so that any number of processes will be accessible to any client module soon.

General Subsystem

Together with the core, the exception and the D-Bus subsystems, general subsystem can be cut out and reused in any other project needing component architecture implementation, because it contains general purpose objects, interfaces and components. For example a command-line argument parsing and an environment variables management is located here.

The subsystem is quite solid, only a module arguments documentation strings access has to be improved. This however doesn't mean that it doesn't need other extensions. If there will occur any new requirements for general functionality, their concretization may be inserted here.

GIS Subsystem

This subsystem should include all instruments to GIS related computations. Currently it has only information about active user and default region and their control. Possible algorithms for a map projection or general GIS data transformation are waiting for their introduction.

Raster Subsystem

It comprehends everything about raster data access, manipulation and conversion. Raster architecture is designed so that data are accessed by tiles. Request for tile contains desired dimensions, position and resolution of the tile in a layer region object. Colour rules and a colour table for data presentation are associated with the returned tile similarly like in the GRASS. Actual data storage is currently kept in GRASS's competence using a `GRASSlib` library.

A present design of the raster data representation is quite initiatory and and it needs an adequate degree of revision from the outside with proper modifications. Hence any comments or suggestions would be positive and convenient contribution. If progress of the project allows practical usage of the library along with the GRASS, new implementations of the raster data storage may be added. Some examples of data analysis modules should be implemented too.

Display Subsystem

Raster data are passed to this part of the framework and displayed. A basic element of this process is a raster image object defined by its dimensions, number of channels and bit depth. First present component implementing raster data visualization emulates `d.mon`'s eight monitors but it uses Qt 4.x for window management and OpenGL for rendering, second is a `d.roamer`'s module component which displays raster data as 3D scene with terrain. Vector data display isn't currently elaborated.

Dynamic Languages Bindings

To allow easy development of modules written in scripting or dynamic languages, SWIG[7] wrapper generator was employed. Existing bindings are targeted to Python and Java.

Unfortunately, technical difficulties with dynamic and heterogeneous nature of the designed communication methodology leaded to many customizations of the wrapper and some limitations. For example a server-side module development in dynamic languages is for now impossible without using D-Bus communication. This can be translated as: „It is not possible to call Python/Java code from C++ code directly.“ Possibility to write client-side modules, which is the main reasonable dynamic languages usage, is though available.

Designed and Implemented Interfaces

Although this article shouldn't serve as the library reference, some important communication interfaces should be listed and explained here to get image about GAL Framework approaches. Interfaces are actually designed as interface objects which holds an interface configuration state (list of available functions with their signatures, a way of communication, etc.) and which are imported to a module on demand from the GAL core. INodeController – is basic interface for independent process management from outside. It's mainly used internally, for example `d.quit` module calls process termination function of this interface. Other functions will serve for communication negotiation.

- **IRasterDisplayer** – displays any raster image on a monitor. This can be tiles of raster layer or simply any raster image (legend, icon, etc.).
- **IRasterLayerDisplayer** – allows direct display of a raster layer on the monitor. This may help to reduce unnecessary computations for better performance and and lets a monitor handler to record a list of raster layer display requests.
- **IRasterLayerProvider** – gives tiled access to GIS raster data. Current implementation uses GRASS libraries for low-level data manipulation.

[7] `http://www.swig.org/`

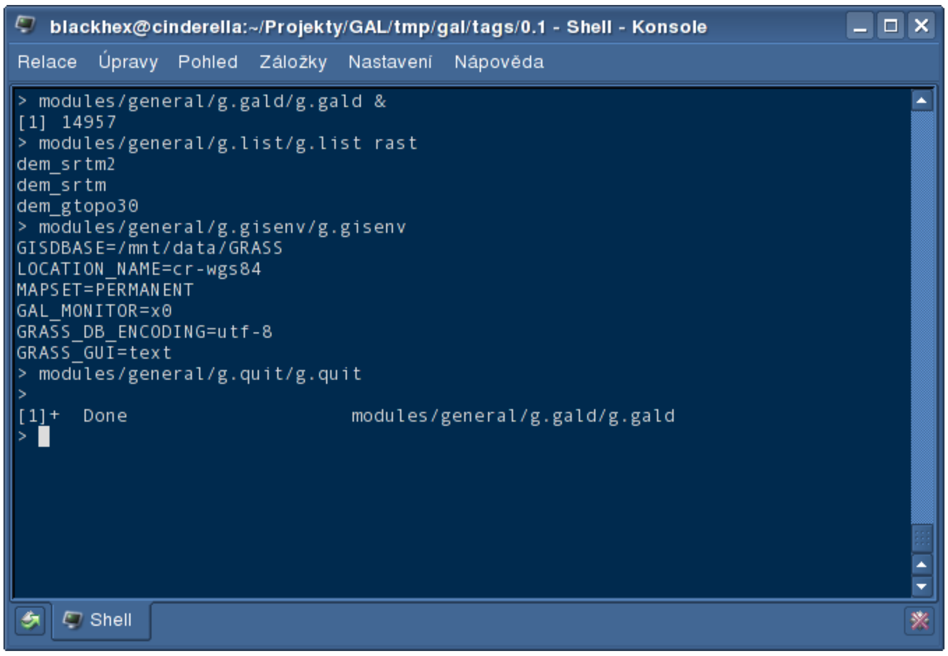
- **IEnvironmentProvider** – provides different storages for global variables. Present implementations are volatile memory, GRASS mapset configuration and GRASS global configuration storage.

Example Tools

A few modules known from the GRASS GIS was developed to test and demonstrate functionality of designed and implemented interfaces. They are described here.

`g.gald`, `g.quit`, `g.list` and `g.gisenv`

Some modules from a general category was rewritten as tests of the designed interfaces. They are a `g.list` and a `g.gisenv`. In addition, a `g.gald` and a `g.quit` modules was introduced. Figure 1. shows example of their usage. First the `g.gald` module, which provides all available functionality implementation, is executed as a daemon. Then the `g.list` is used to list raster layers of a mapset and the `g.gisenv` module displays defined environment variables. Finally, the `g.quit` module terminates the running `g.gald` module.


blackhex@cinderella:~/Projekty/GAL/tmp/gal/tags/0.1 - Shell - Konsole
Relace Úpravy Pohled Záložky Nastavení Nápověda
Shell

```
> modules/general/g.gald/g.gald &
[1] 14957
> modules/general/g.list/g.list rast
dem_srtm2
dem_srtm
dem_gtopo30
> modules/general/g.gisenv/g.gisenv
GISDBASE=/mnt/data/GRASS
LOCATION_NAME=cr-wgs84
MAPSET=PERMANENT
GAL_MONITOR=x0
GRASS_DB_ENCODING=utf-8
GRASS_GUI=text
> modules/general/g.quit/g.quit
>
[1]+  Done                    modules/general/g.gald/g.gald
> 
```

Figure 1: Some modules from general category.

`d.mon`, `d.move`, `d.resize` and `d.rast`

User interface of reimplemented `d.mon` module is shown on the Figure 2. The `d.mon` module actually only gives order to show a monitor to the waiting `g.gald` process which performs own monitor window display. It is the same with `d.rast` module that reads raster data from GRASS and sends them to `g.gald`. Other controlling modules the `d.move` and the `d.resize` tell the `g.gald` to move or resize the window.

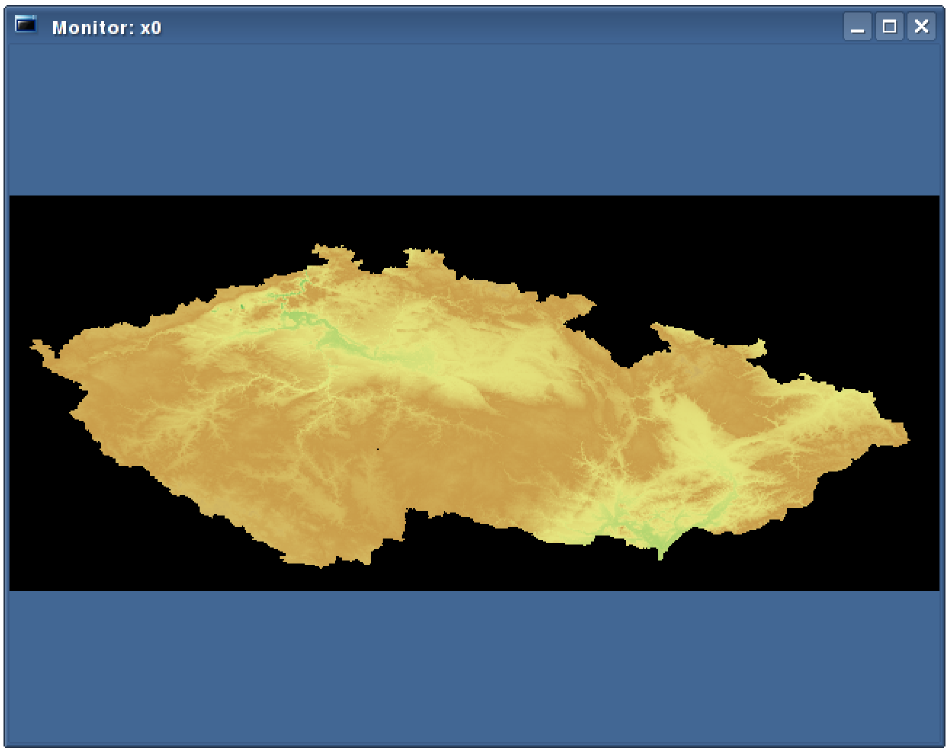


Figure 2: `d.mon` module in action.

`d.roamer`

The last presented module is called a `d.roamer` and it allows the user to fly over a visualized terrain in real-time. It's screenshots can be found on Figure 3. and 4. The first shows the terrain rendered with full faces, the second uses wireframe. This demonstrates used level of detail algorithm called geo mip-mapping.

Figure 5. contains diagram of internal communication between `d.roamer` and `d.rast` modules using the framework. Analogously as with the `g.gald`, `d.mon` and `d.rast` modules in previous paragraph, data are read form `GRASSRasterLayerProvider` component and pased through `IRasterLayerProvider` and `IRasterDisplayer` interfaces to `d.roamer`'s `RoamerComponent` component.

Areas of Future Development

As you may notice, vector subsystem is not present in the framework at all yet. The explanation is that it was not necessary to focus on so complex area as vector data architecture is for the prove of concept of proposed and designed communication strategy. Hopefully, decent vector implementation will be result of Bc. Jan Kittler's master thesis whom the article author is cooperating with. He should design new internal and external representation of vectors and some analytical tools with user interface. Core parts should be implemented in C++, analysis tools and user interface in C#. This will introduce need of C# bindings for GAL Framework.

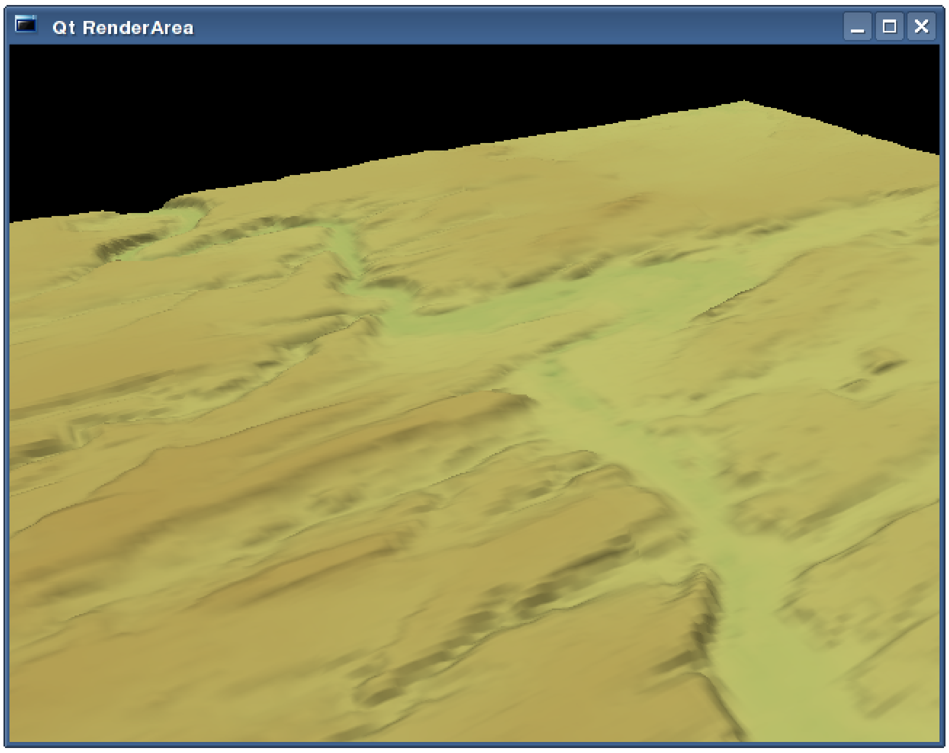


Figure 3: `d.roamer` module interface with full-faced terrain.

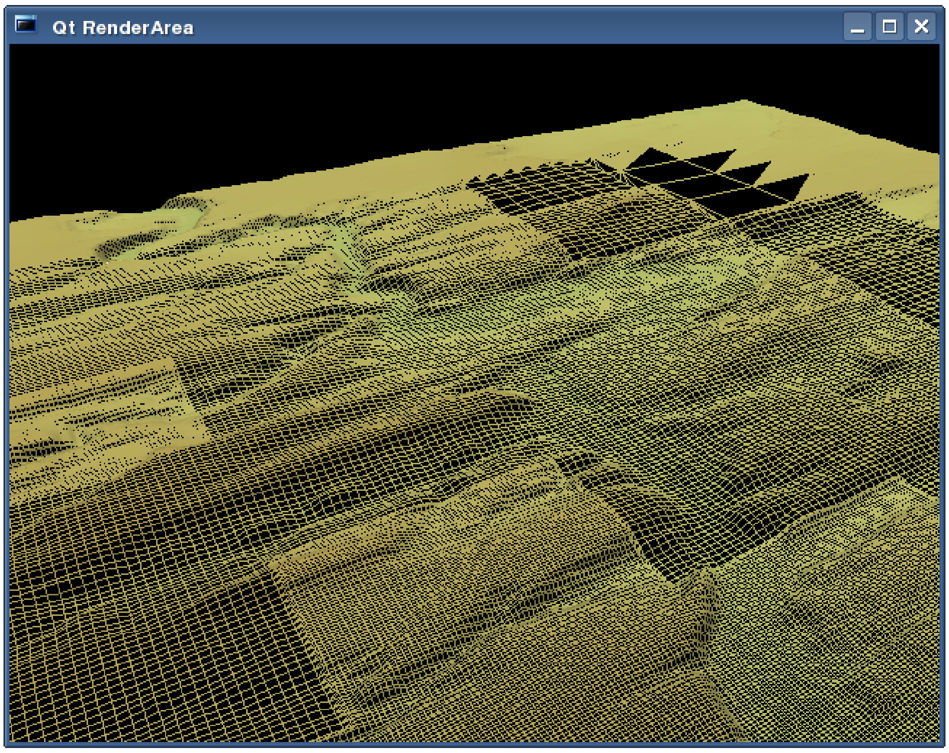


Figure 4: `d.roamer` module interface with wireframe terrain.

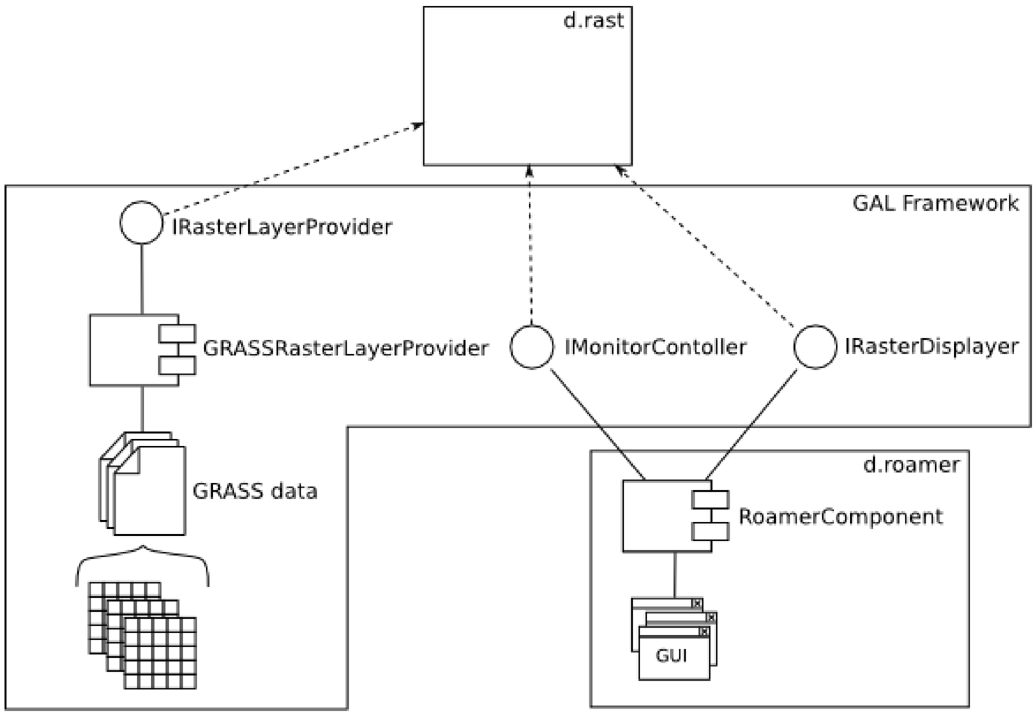


Figure 5: Architecture of `d.roamer` module.

Because of huge scale of project's extent, another outside contribution would be more than welcomed. Safe multi-thread processing of events in loop including thread-safe access to any internal data of the library may be elaborated. Better raster architecture as long as any number of raster or vector data format implementations may be added. And finally, new modules using GAL Framework may be developed. Bachelor or Diploma theses on that themes could be published.

Some Statistics

- 20 months of development of single person.
- 9000 code lines (according to http://www.ohloh.net/projects/9183/analyses/latest).
- 6500 comment lines (mainly Doxygen documentation).
- C++ as main language, Python and Java bindings.
- 41 commits to SVN repository (svn://gal-framework.no-ip.org:3691).
- Depends on D-Bus, libxml2, libgcj or libffi, Qt 4.x, SoTerrain[8] and GRASSlib libraries (some optionally).
- Homepage is Trac instance at http://gal-framework.no-ip.org.

[8] `http://blackhex.no-ip.org/wiki/SoTerrain`

References

1. *Christopher Lenz, Dave Abrahams and Christian Boos.* Trac Component Architecture http://trac.edgewall.org/wiki/TracDev/ComponentArchitecture, July 2007.

2. *Radek Bartoň and Martin Hrubý.* GAL Framework. In Proceedings of the workshop Geoinformatics FCE CTU 2007. Czech Technical University in Prague, September 2007.

3. *GRASS Development Team.* GRASS GIS. http://grass.itc.it.

4. *freedesktop.org.* D-Bus. http://www.freedesktop.org/wiki/Software/dbus.

5. *SWIG.* Simplified Wrapper and Interface Generator. http://www.swig.org.

6. *Radek Bartoň.* SoTerrain. http://blackhex.no-ip.org/wiki/SoTerrain, October 2007.