JGRASS – present and Future

HydroloGIS
Andrea Antonello
Silvia Franceschi

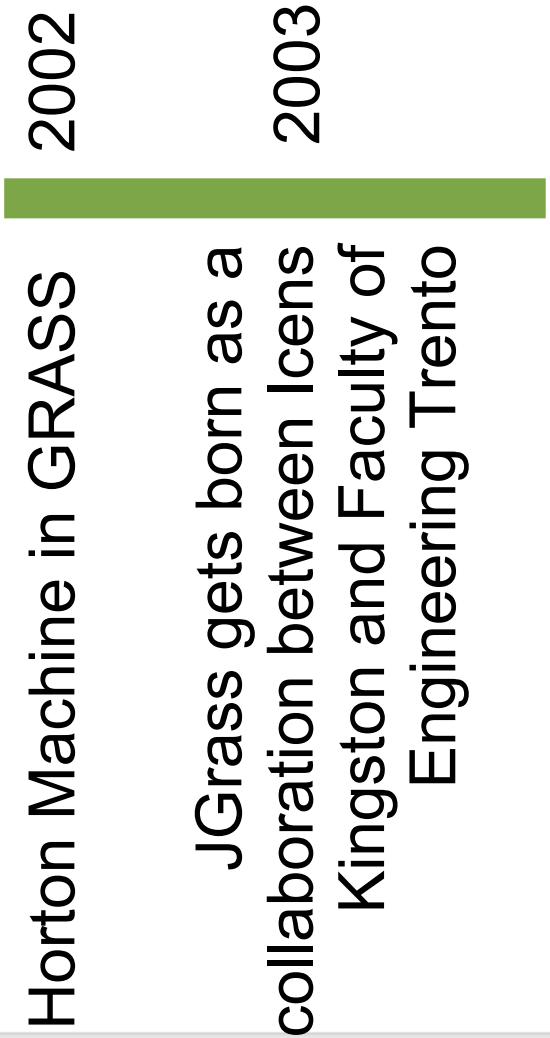
University of Trento
Prof. Riccardo Rigon

Foss4G2009 - Sydney 23 October 2009

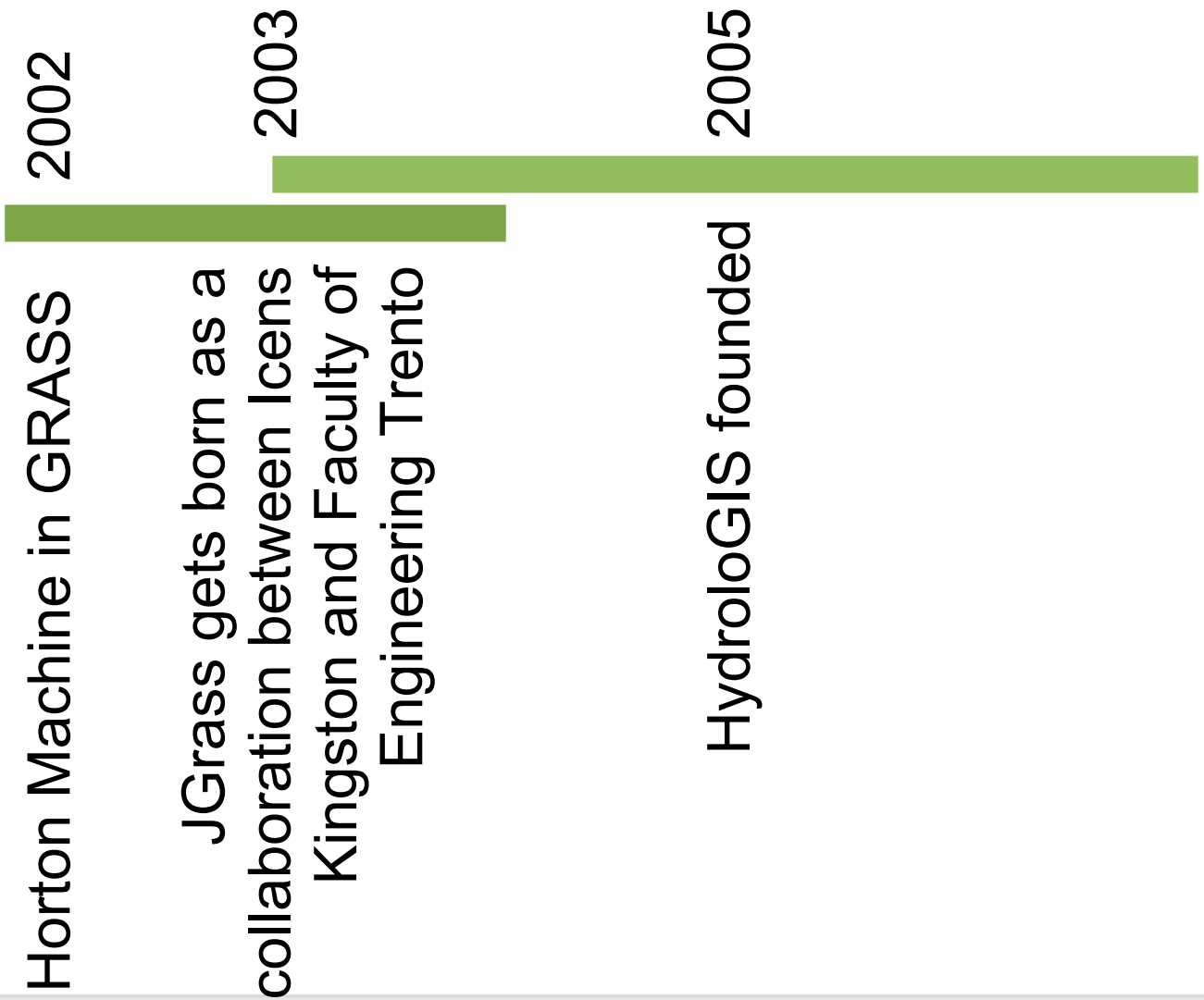
The road to JGrass

Horton Machine in GRASS 2002

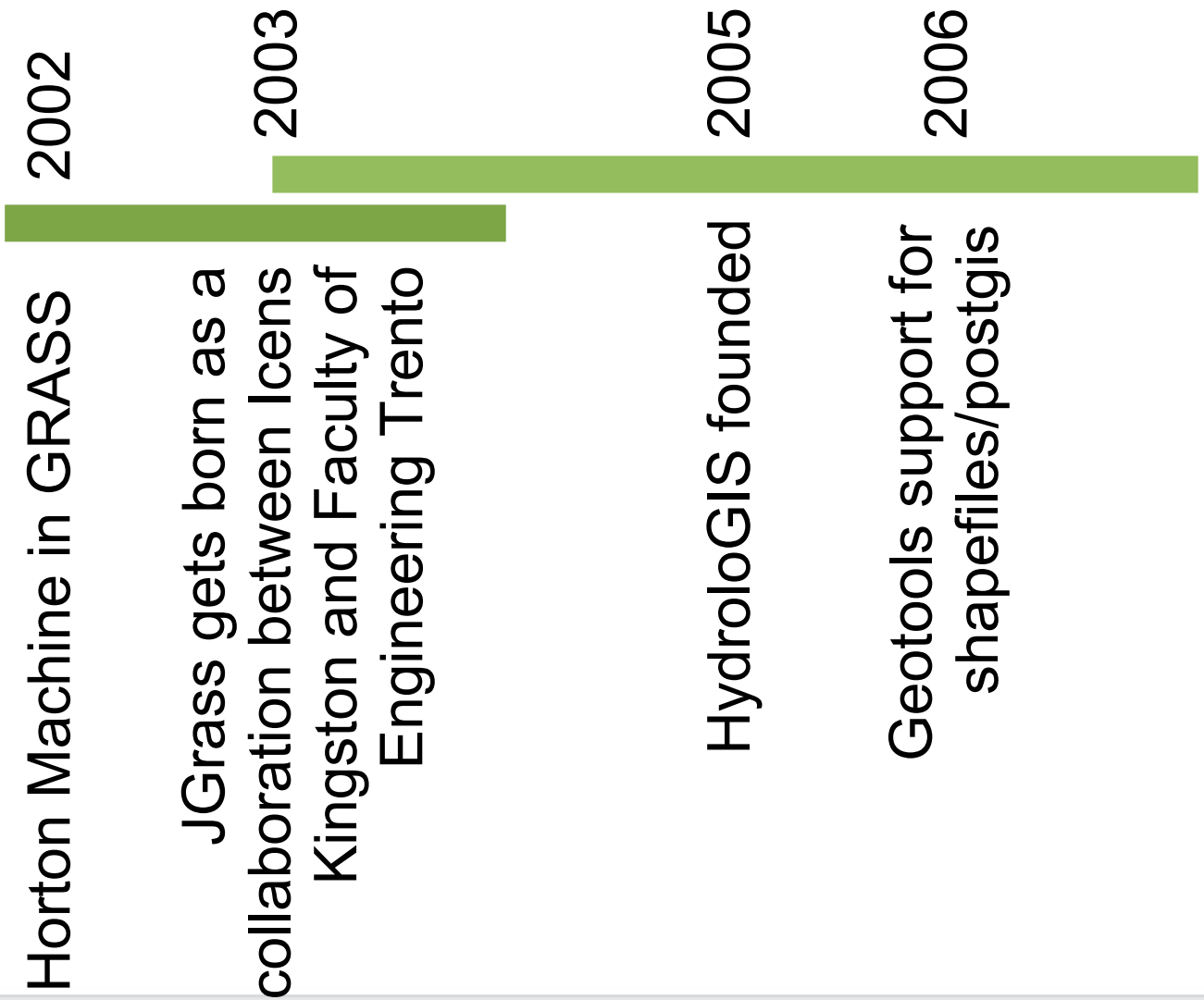
The road to JGrass



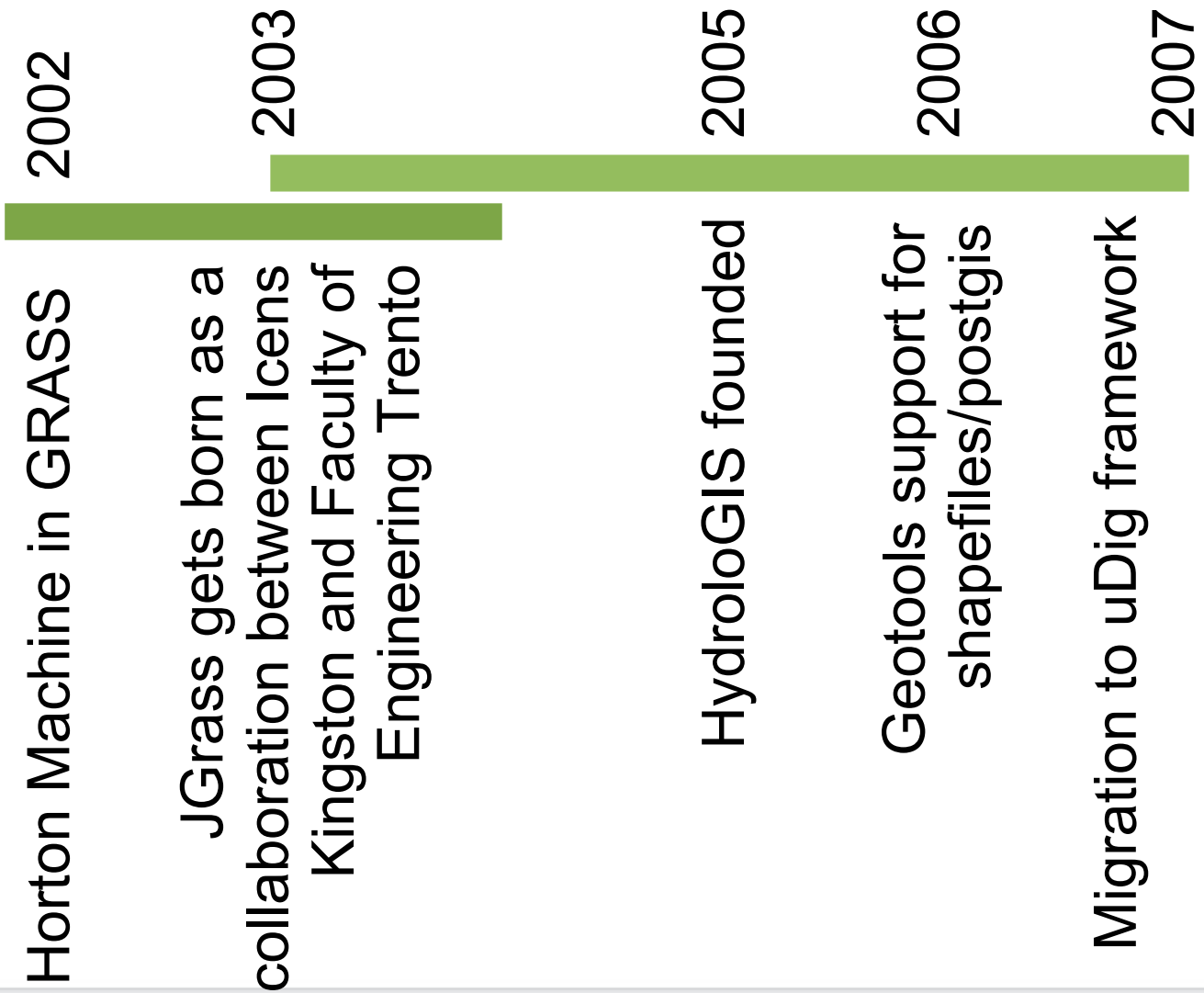
The road to JGrass



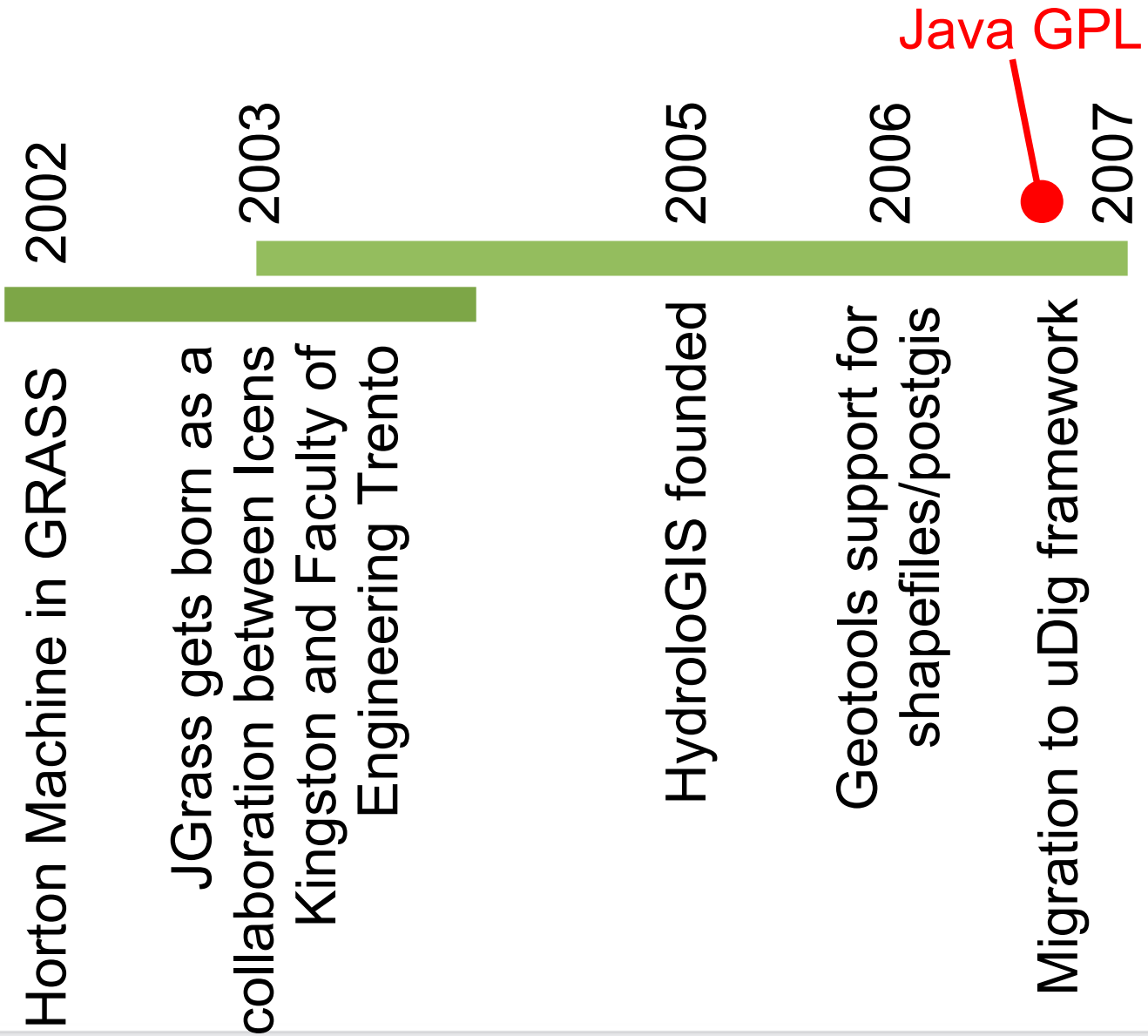
The road to JGrass



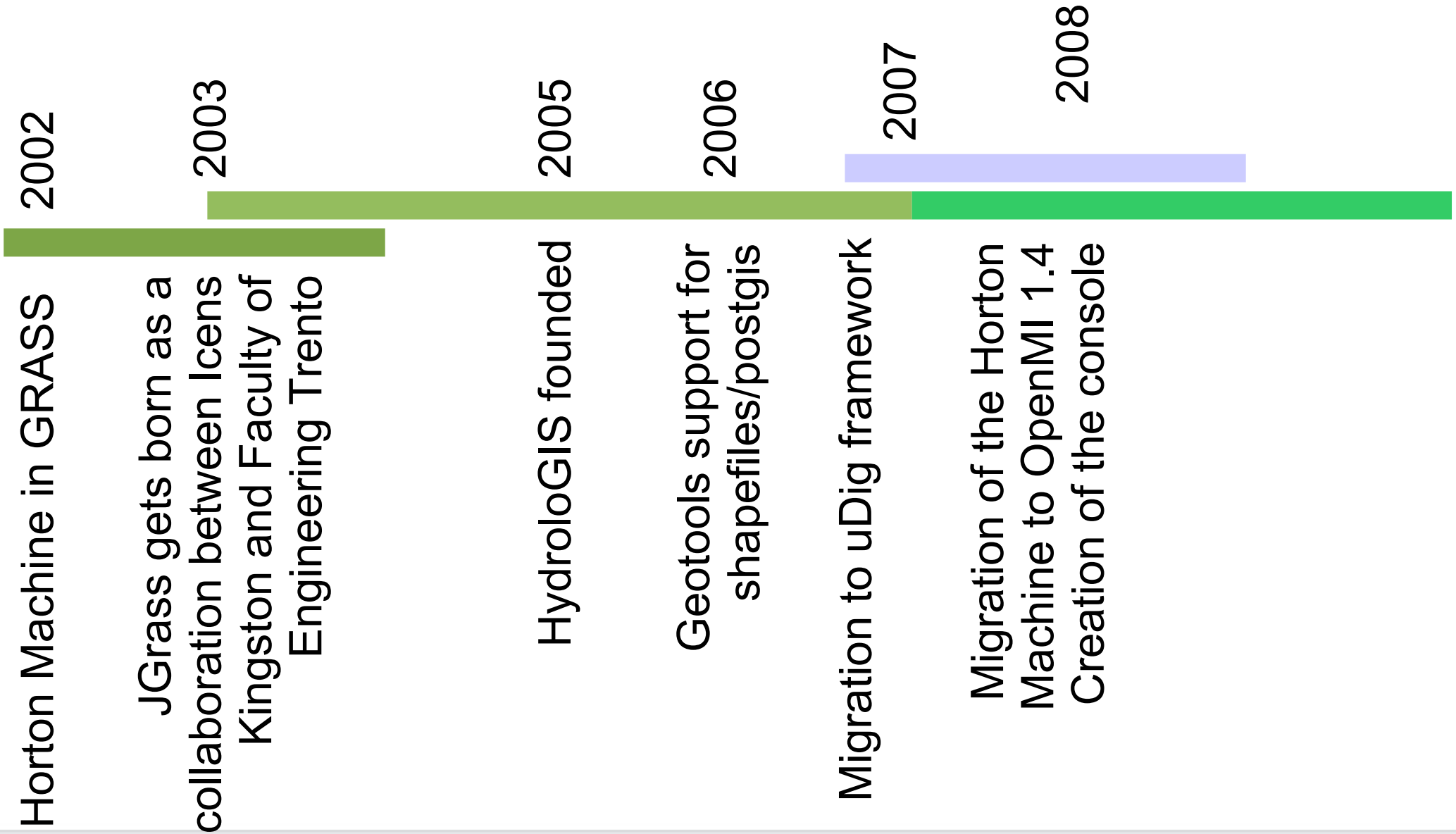
The road to JGrass



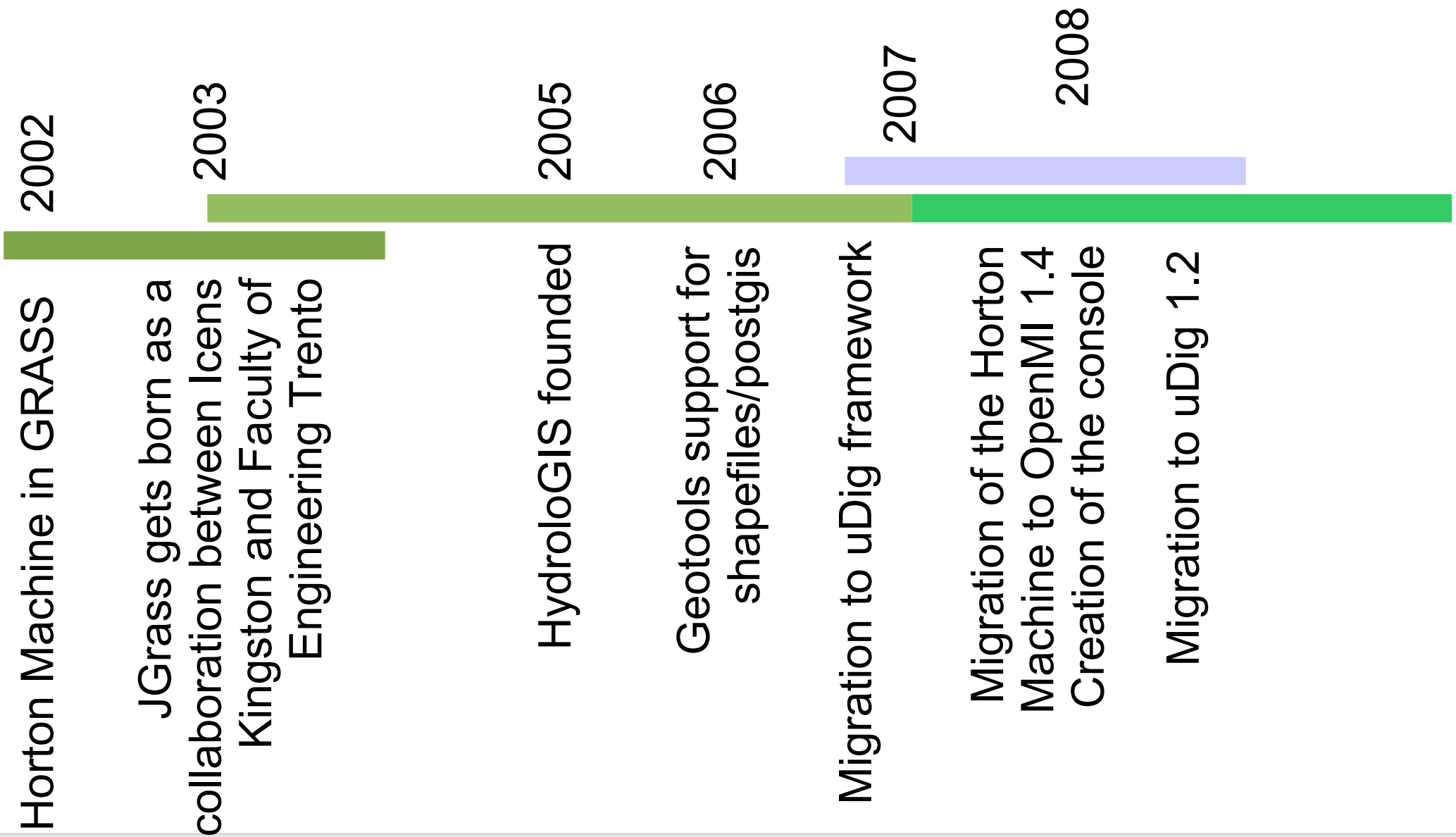
The road to JGrass



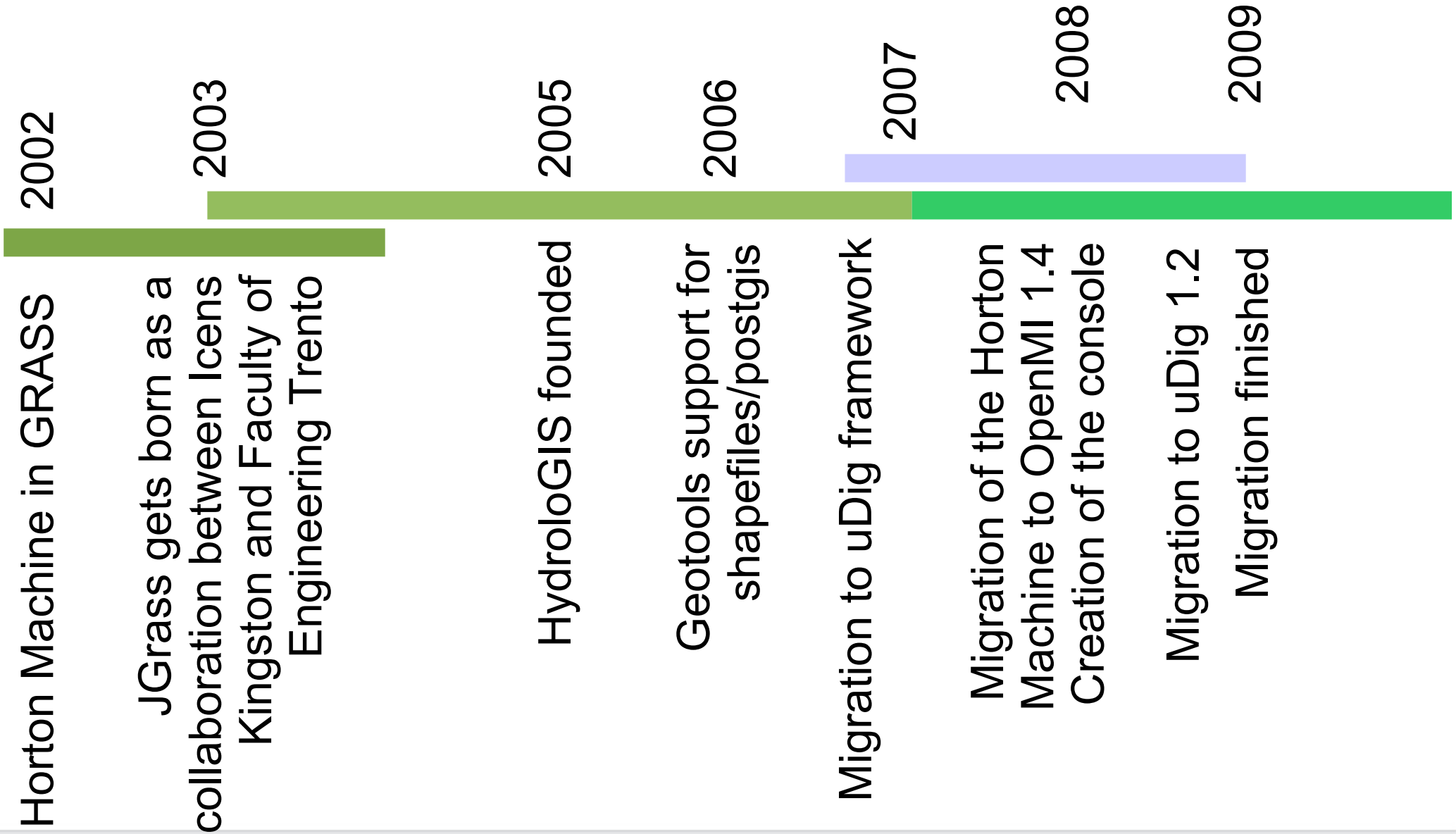
The road to JGrass



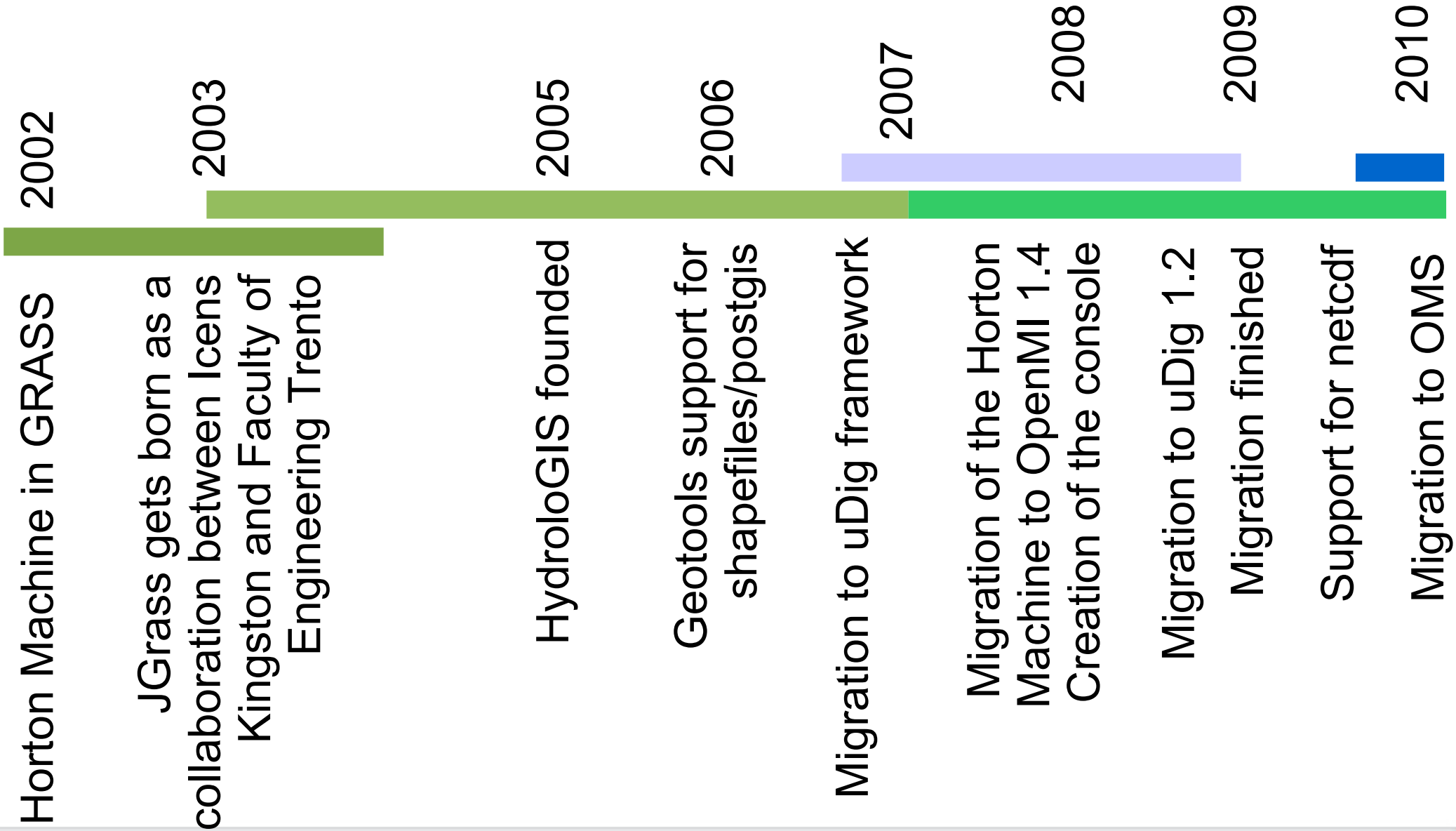
The road to JGrass



The road to JGrass



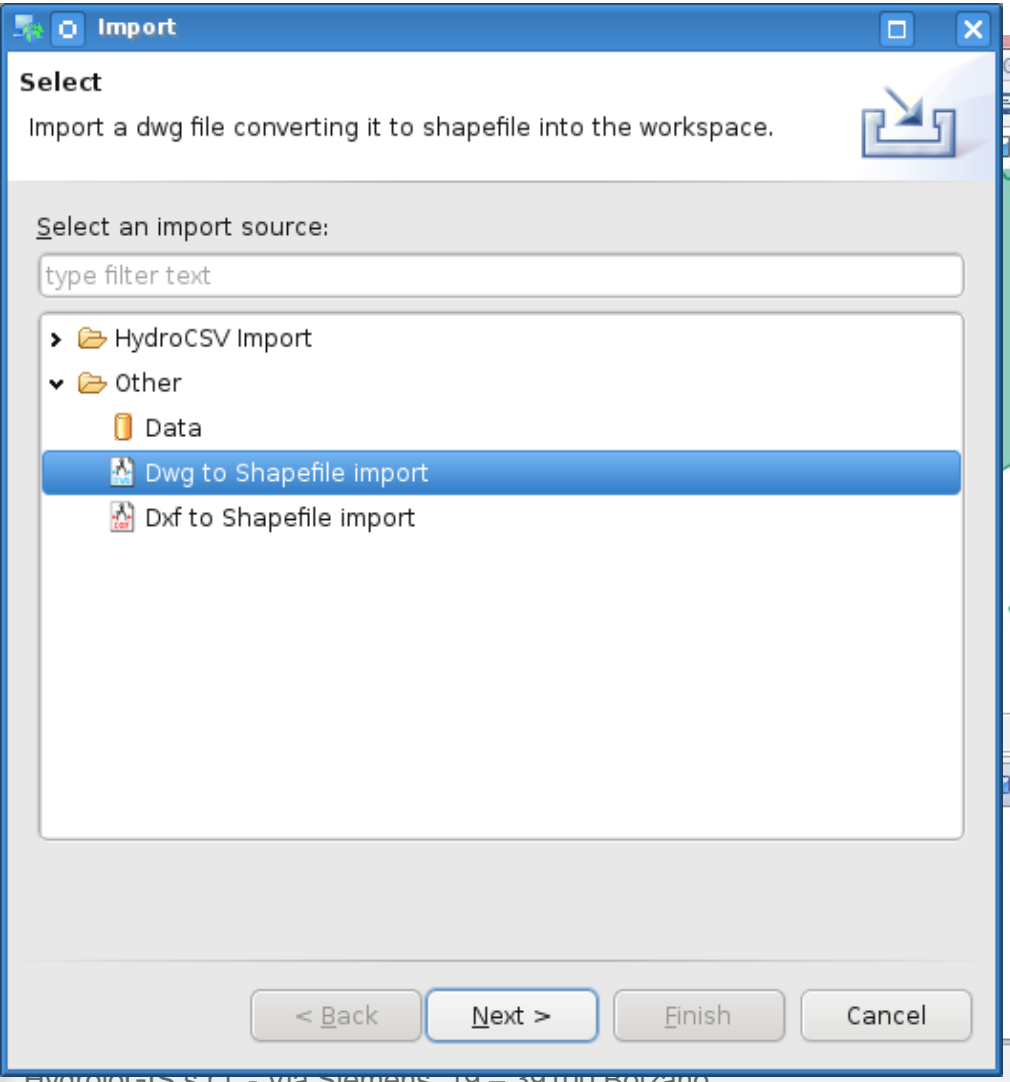
The road to JGrass

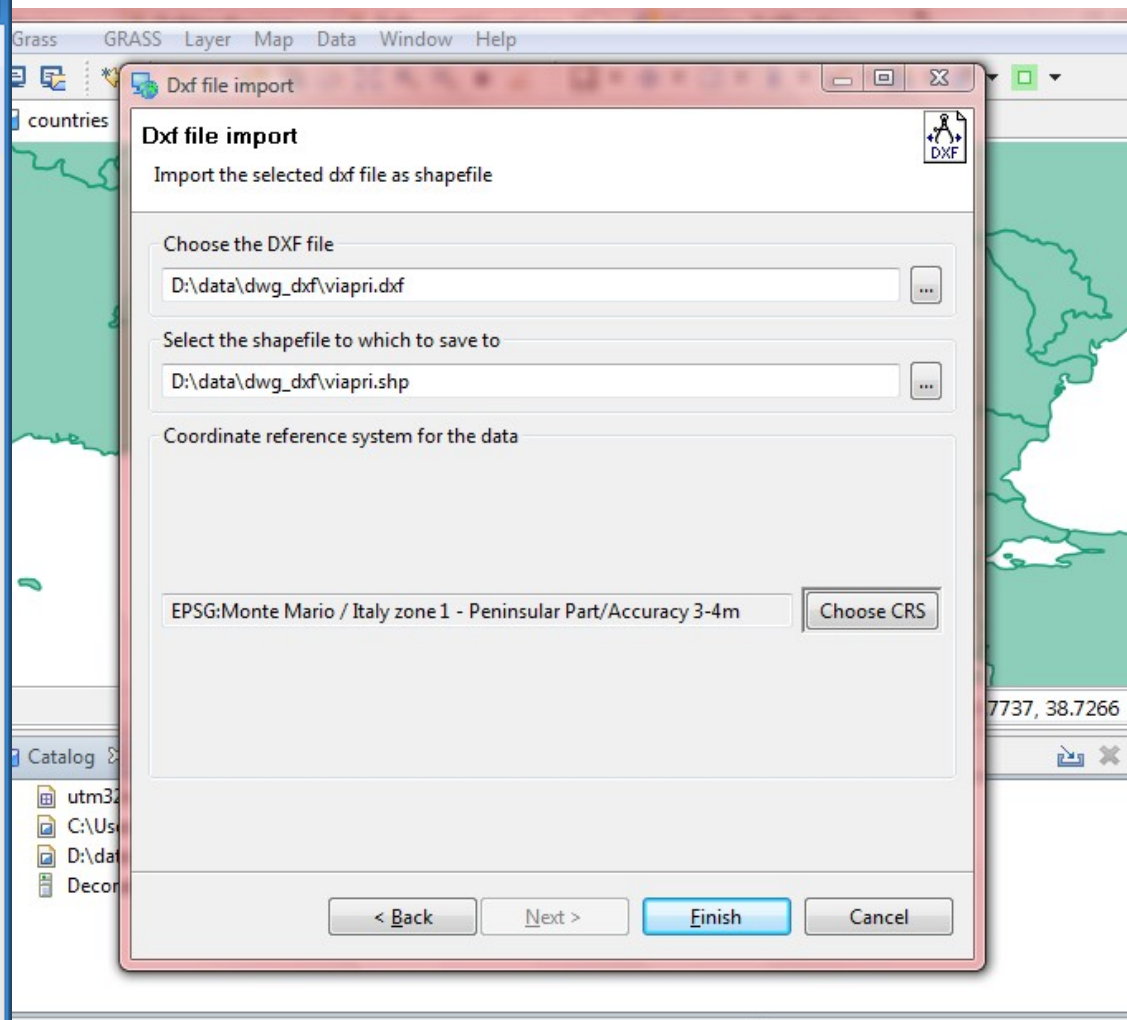


What have we been working on lately?

JGrass – handling DXF and DWG

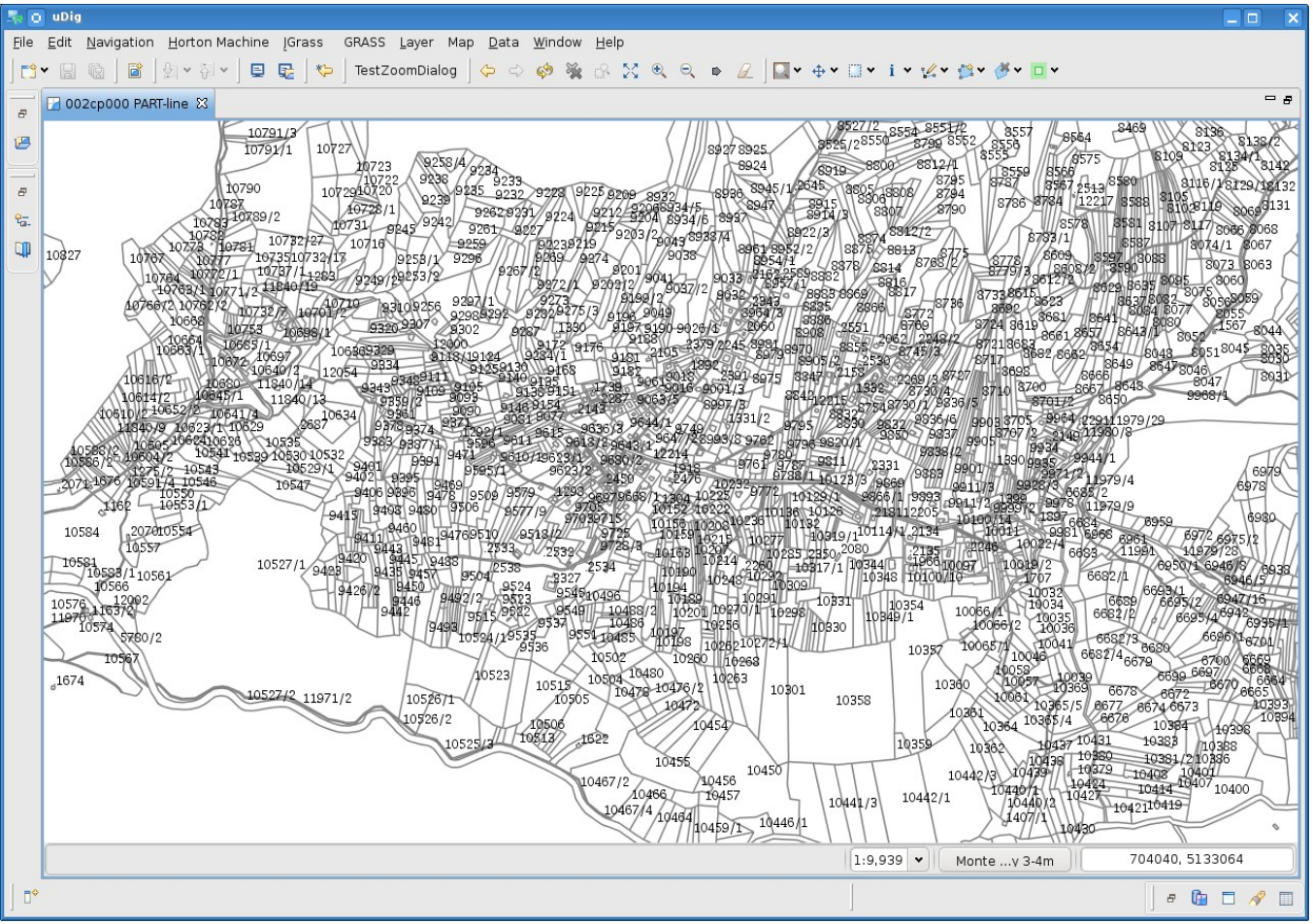
Support for DXF and DWG (up to 2000) files (are file formats used for storing design data and metadata)





JGrass – handling DXF and DWG

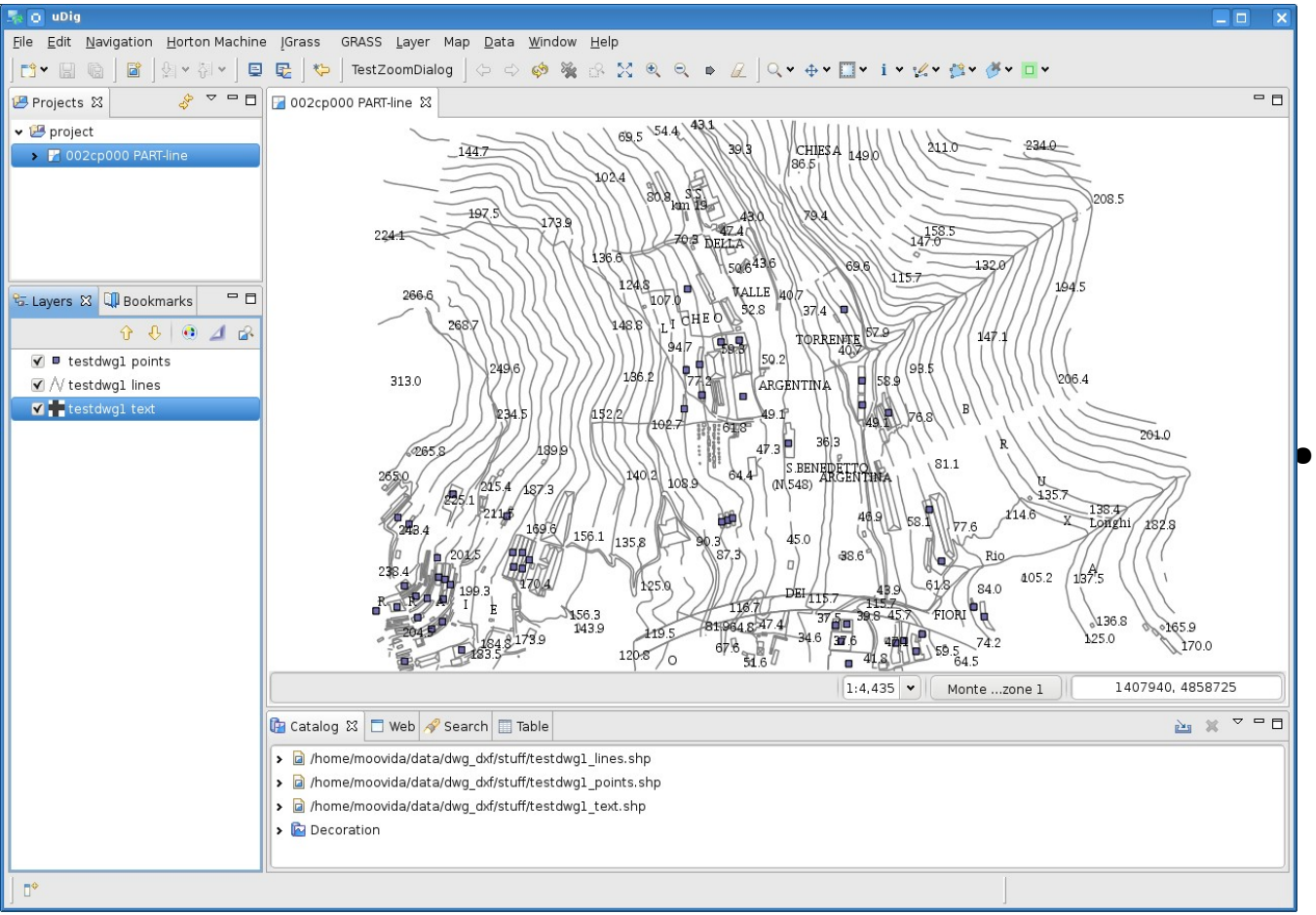
How are DXF files imported?



- every layer is imported as an own shapefile of the type contained
- the types that can be represented through points, lines and polygons, are imported, the others ignored
- text is imported as point layer with a text field to be used as label

JGrass – handling DXF and DWG

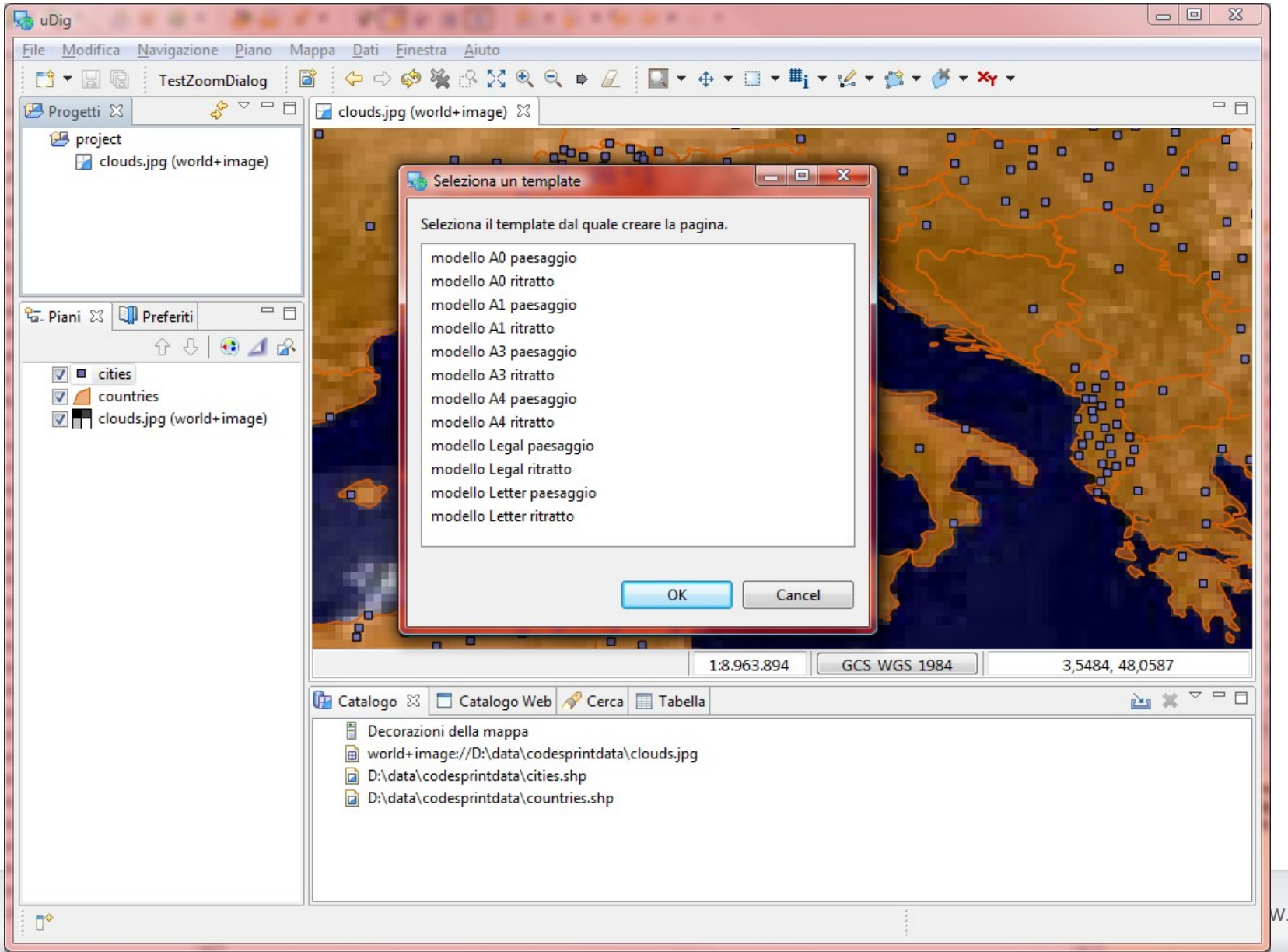
How are DWG files imported?



- given the complexity and due to the library, all the data is imported in 3 main shapefiles of the types: points, lines and polygon
- the original layer name is put as an attribute in order to be able to select data from one layer and copy/paste it to a new layer if needed

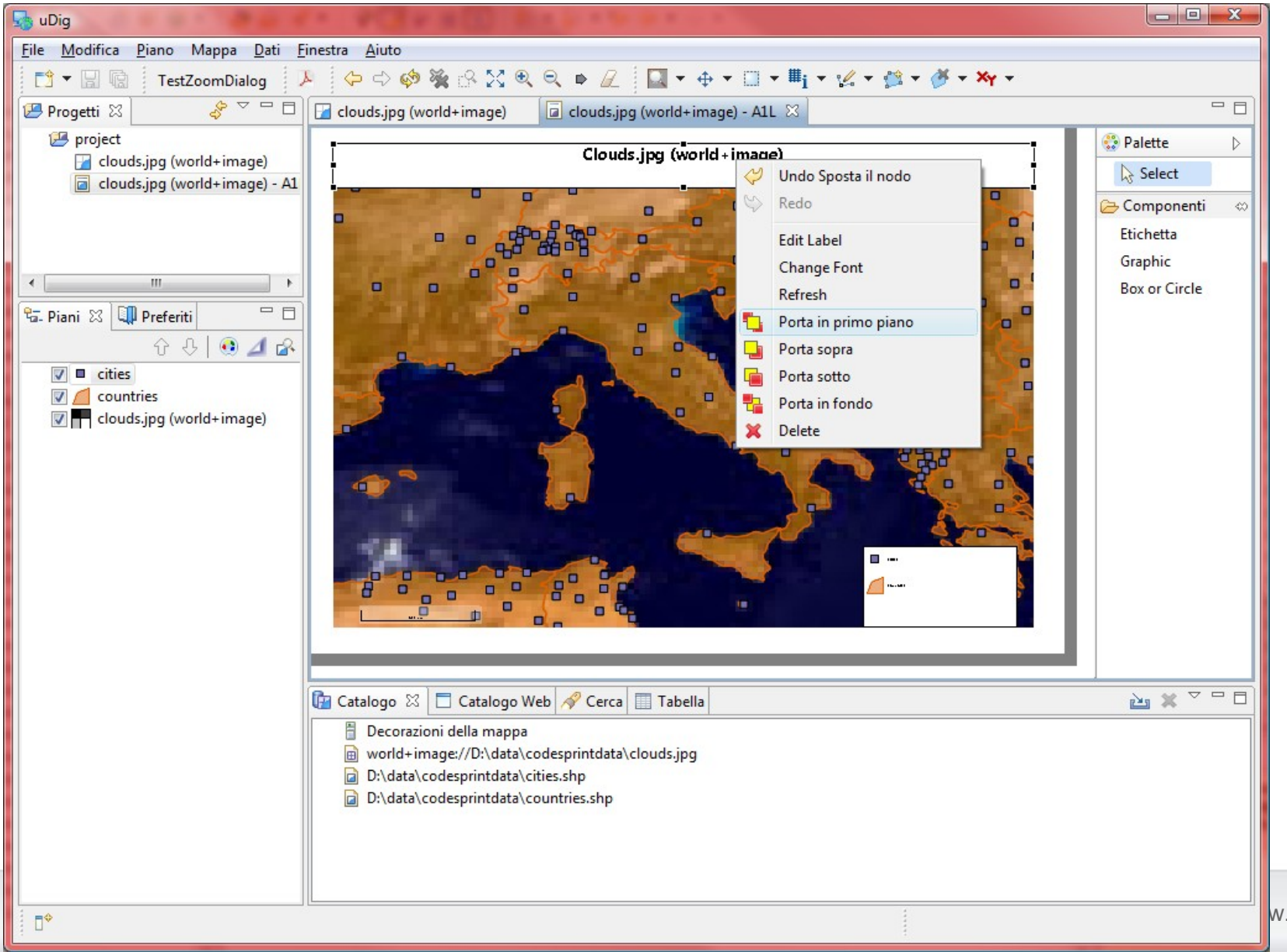
JGrass – enhancements of the printing engine

Choose your page template...



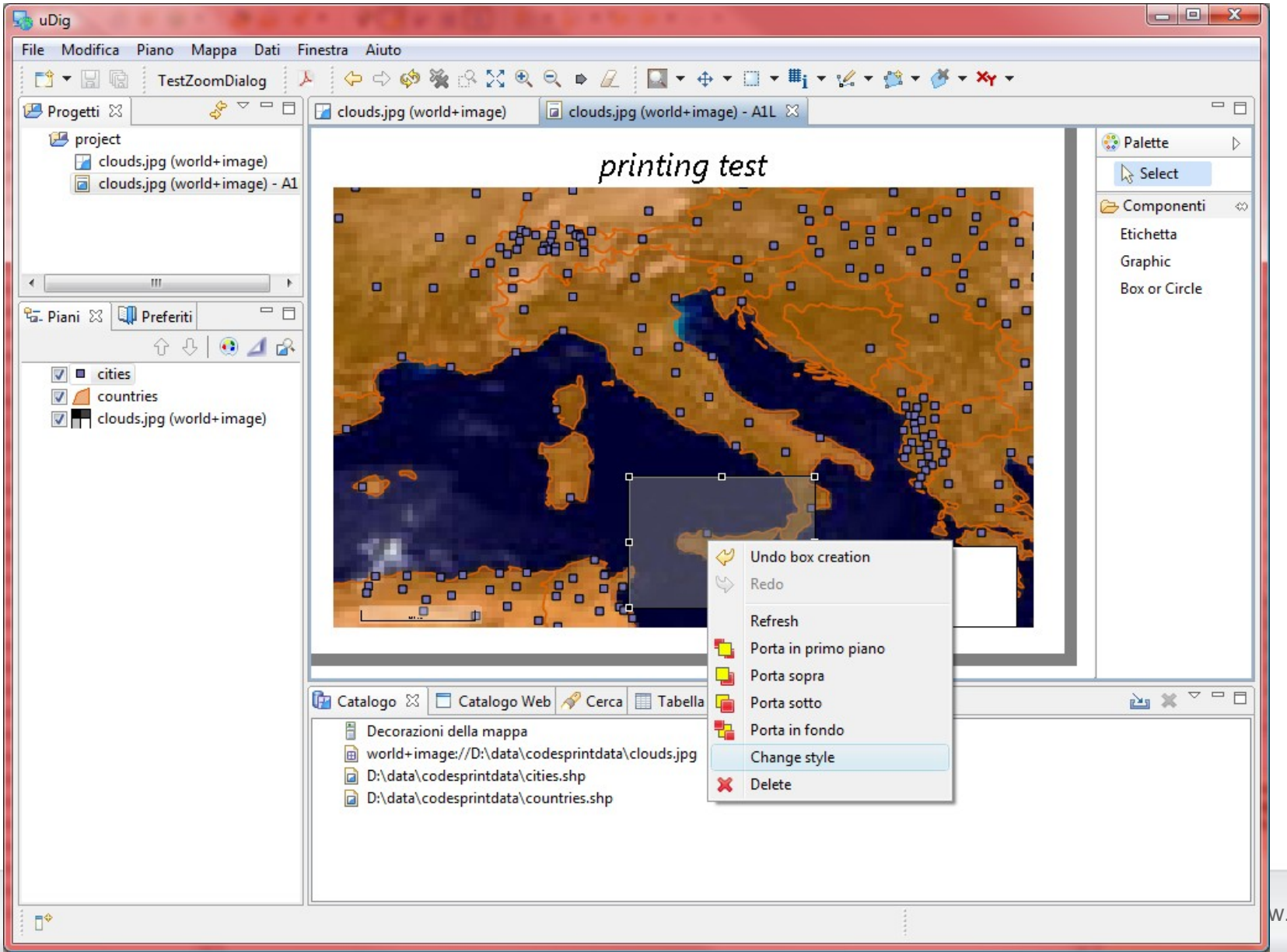
JGrass – enhancements of the printing engine

...modify the template, adapt it...



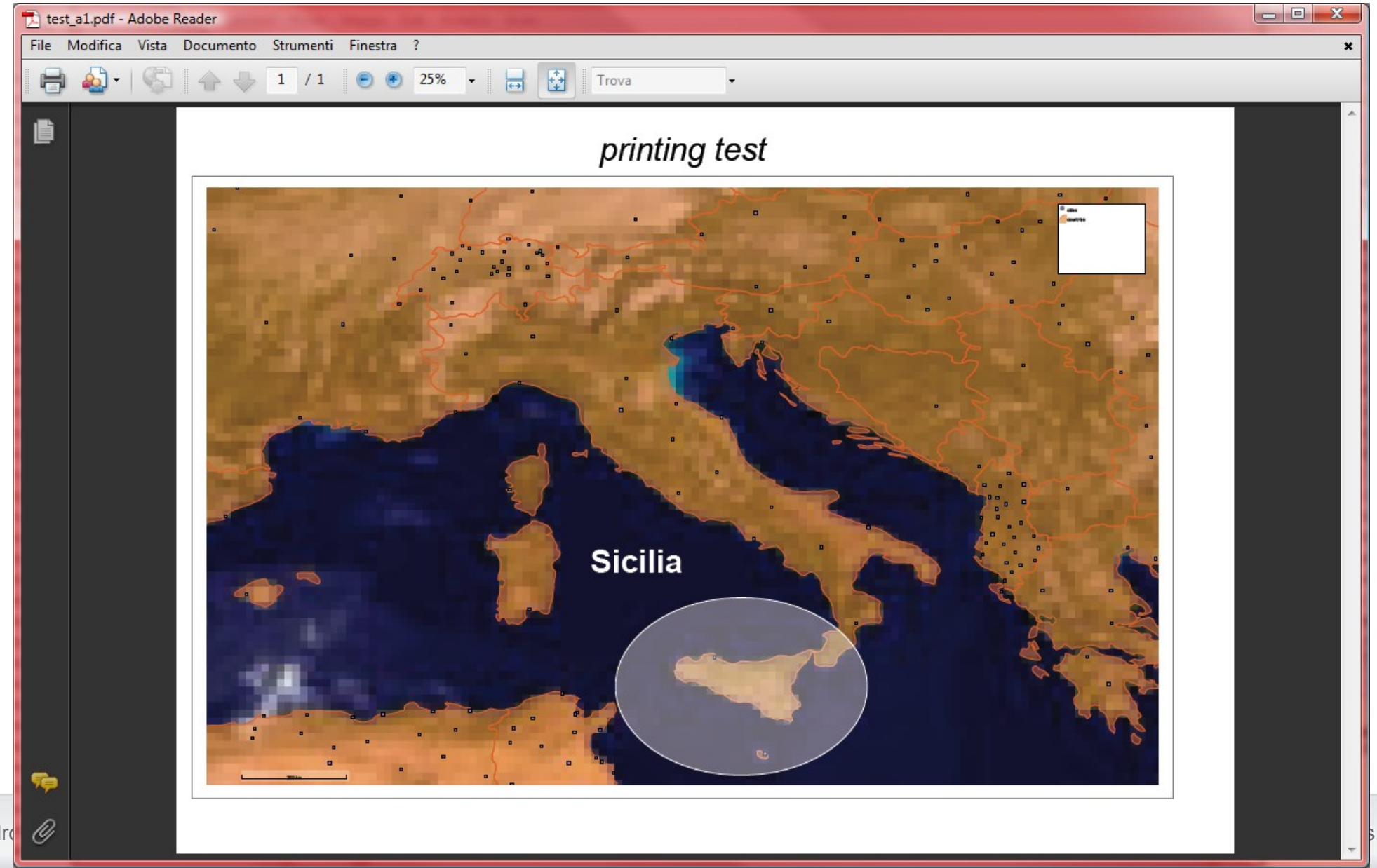
JGrass – enhancements of the printing engine

...add shapes or graphics...



JGrass – enhancements of the printing engine

...and finally print it to pdf.



JGrass – handling netcdf files

NetCDF (Network Common Data Form) is a set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data.

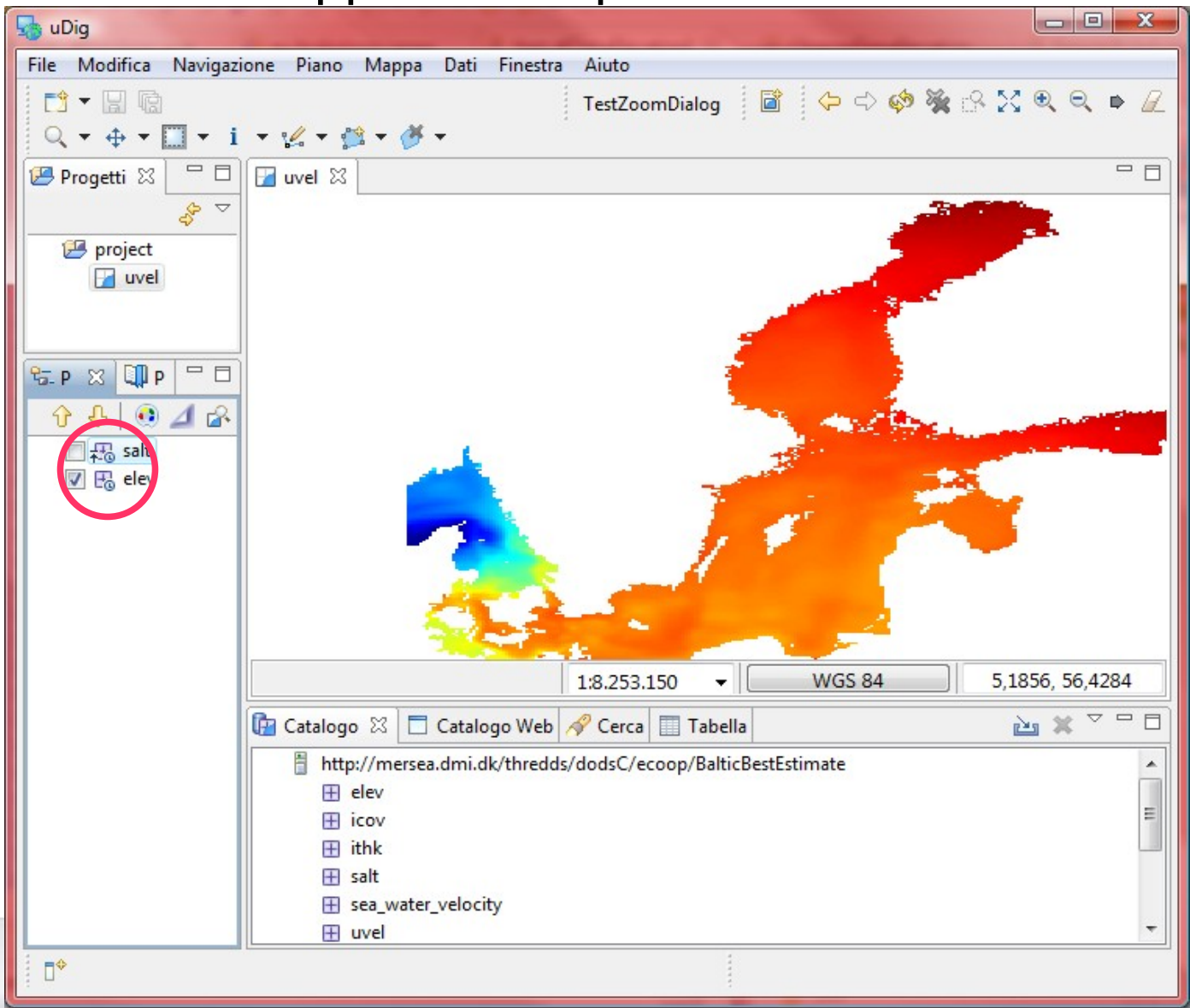
The project is primarly driven by the Unidata program at the University Corporation for Atmospheric Research (UCAR). They are also the chief source of netCDF software, standards development, updates etc. The format is an open standard.

Netcdf support was implemented with the fundings from the Google Summer of Code 2009 Program.

JGrass – reading netcdf files

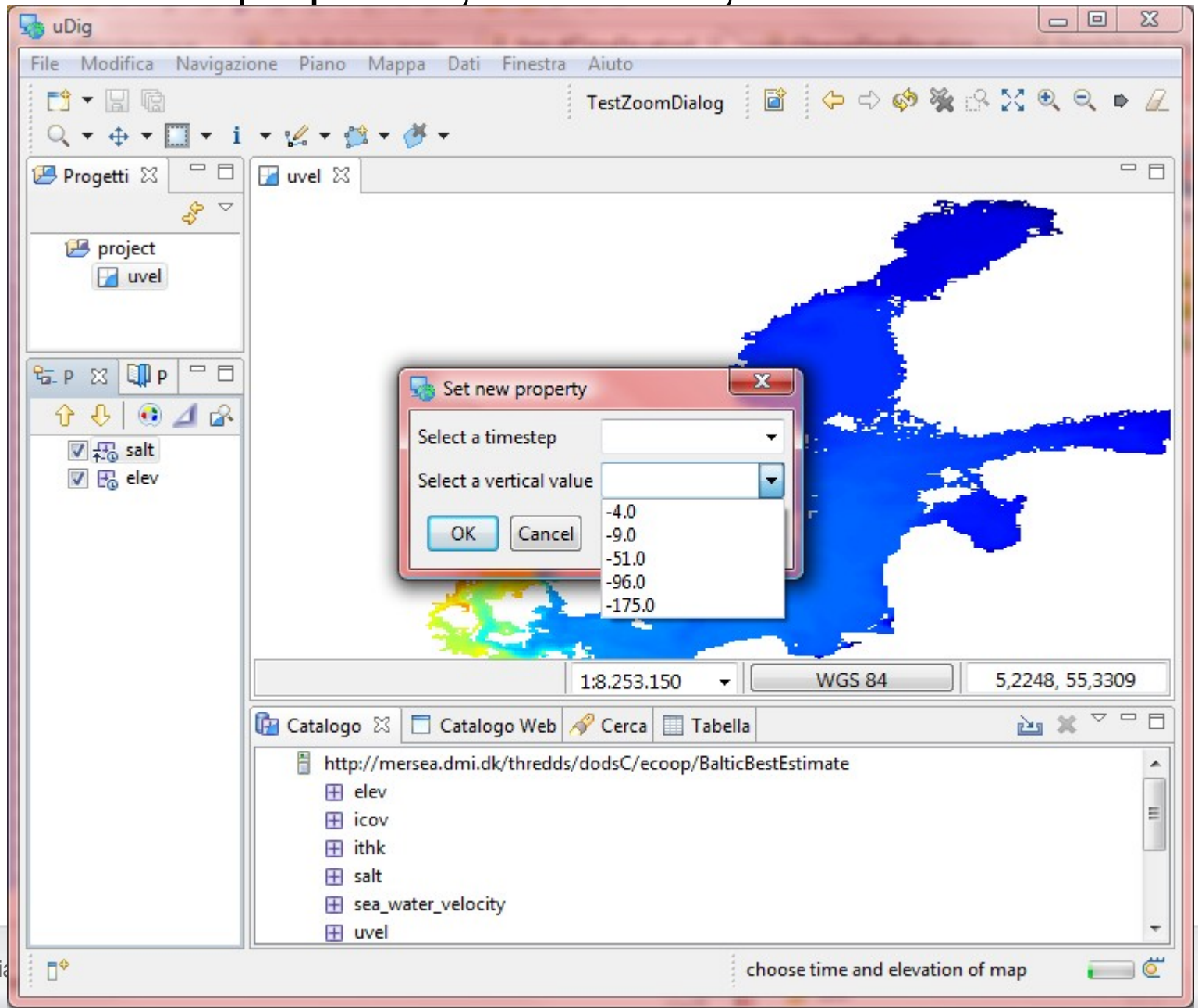
NetCDF is supported for map visualization up to 4D. 2D is handled as as normal raster map, 3D and 4D add support for depth and time levels.

- both the remote (opendap) and local dataset handling is supported
- icons of the layer show the dimension of the dataset



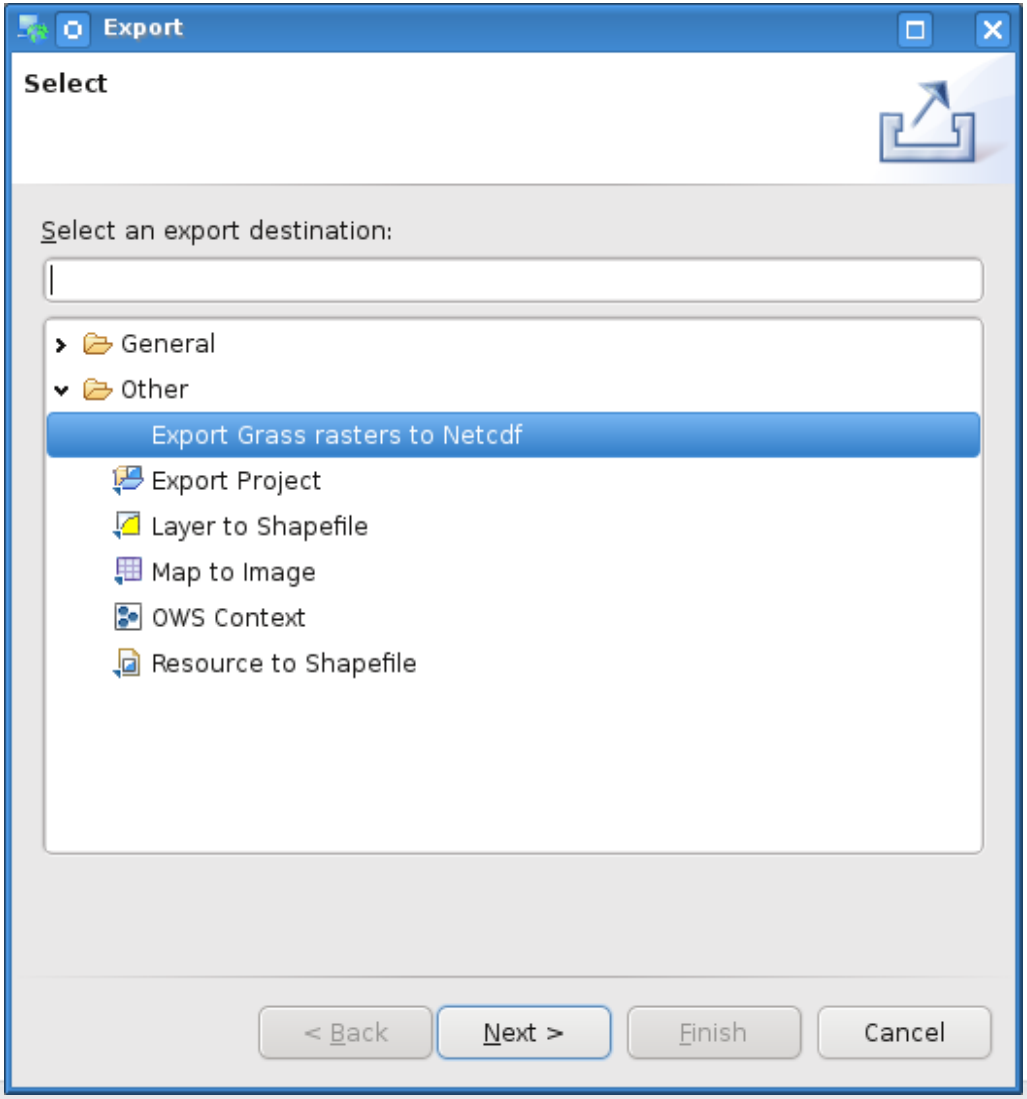
JGrass – reading netcdf files

The depth and time properties, if available, can be browsed



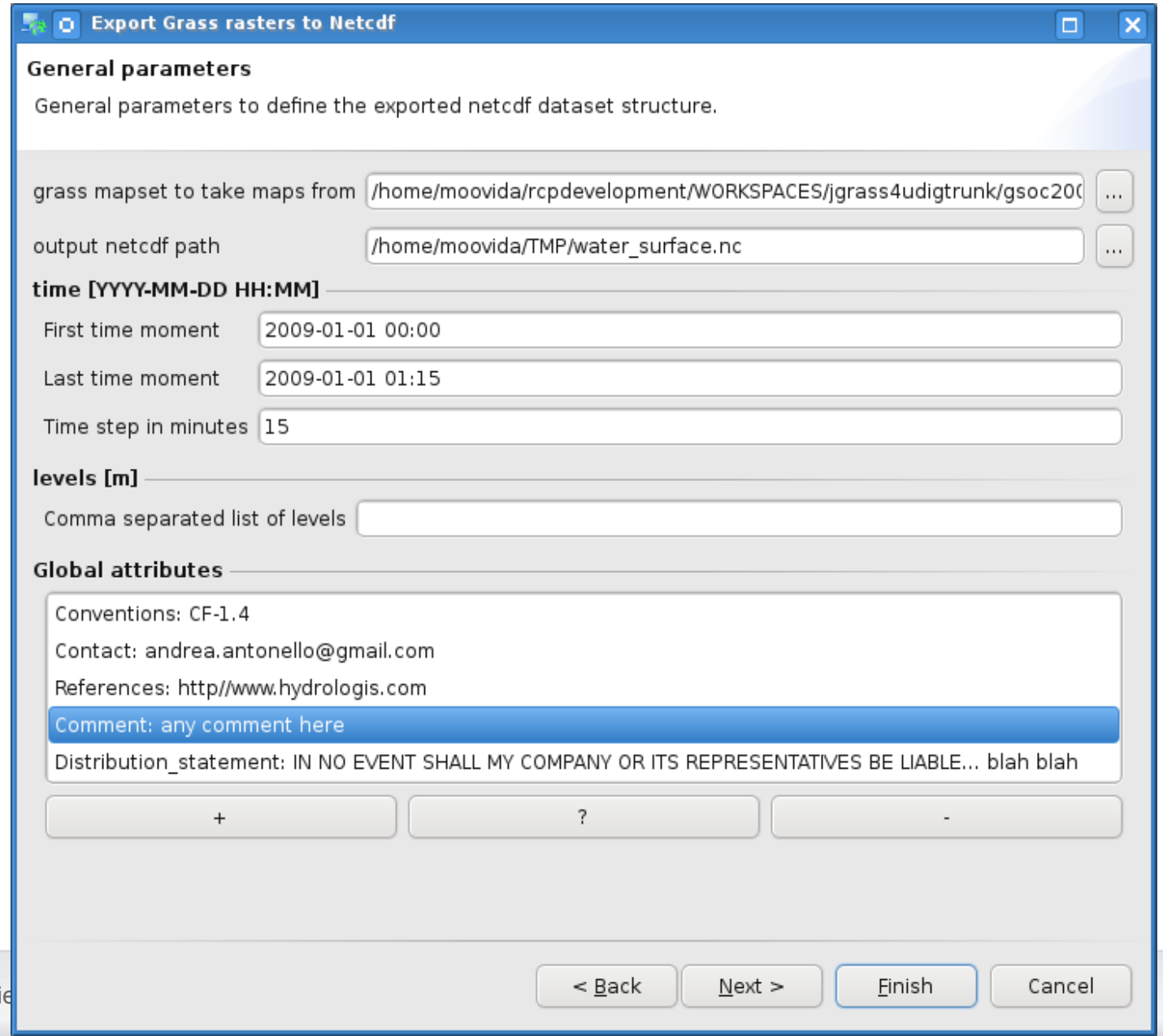
JGrass – writing netcdf files

Raster maps from the workspace can be bundled and exported as NetCDF datasets.



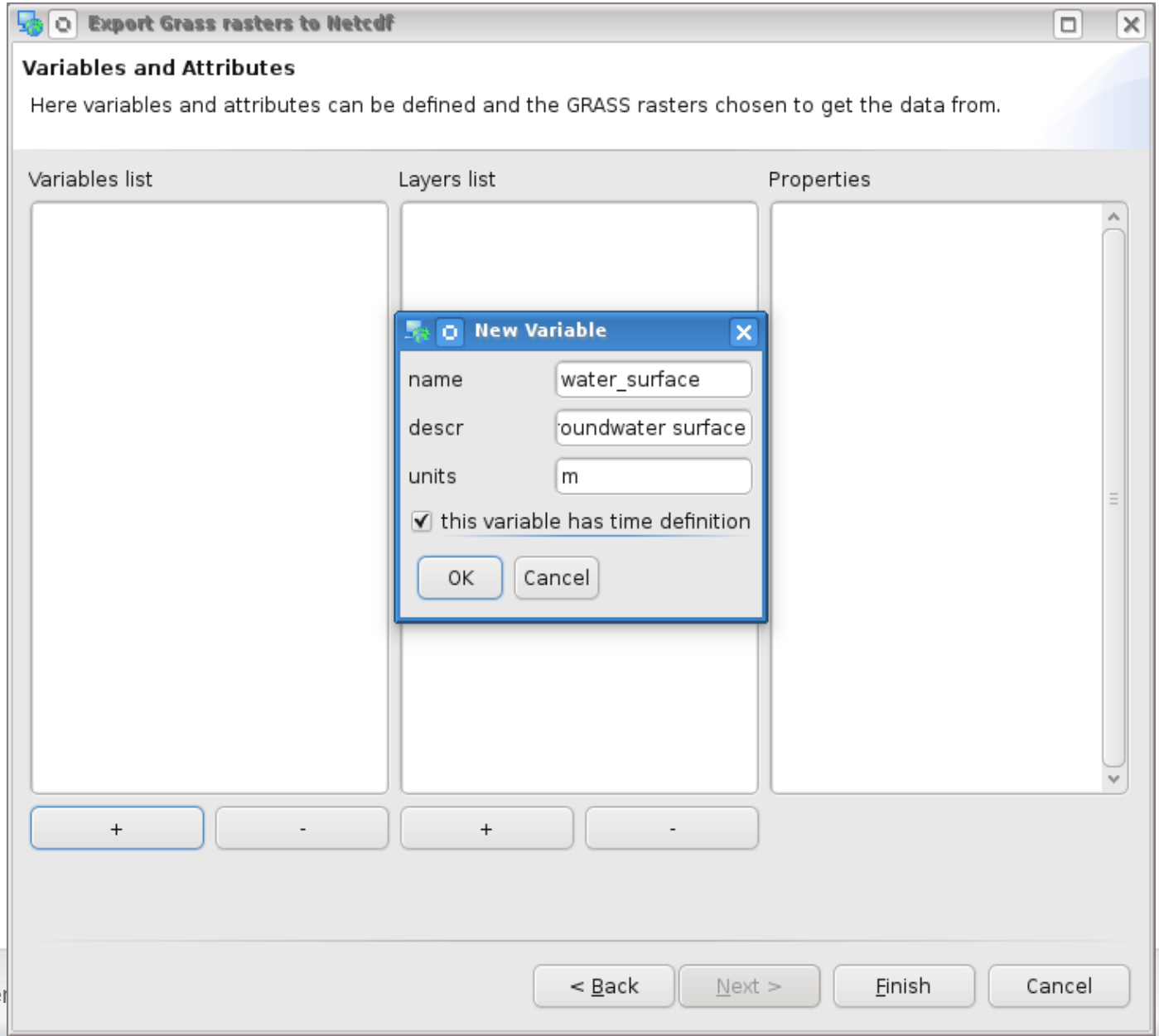
JGrass – writing netcdf files

Step 1: define general metadata



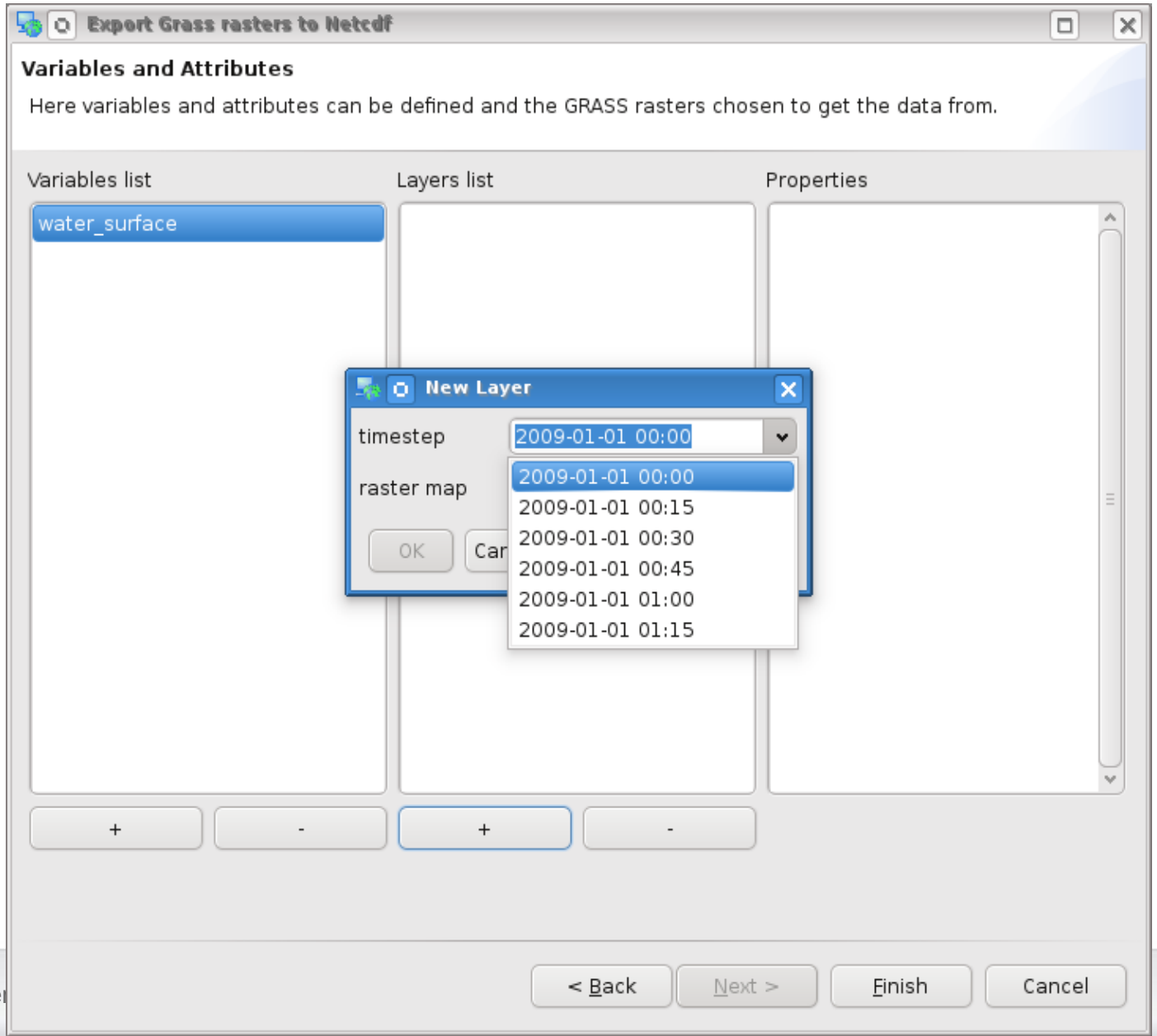
JGrass – writing netcdf files

Step 2: define variables, making sure time support is enabled if necessary



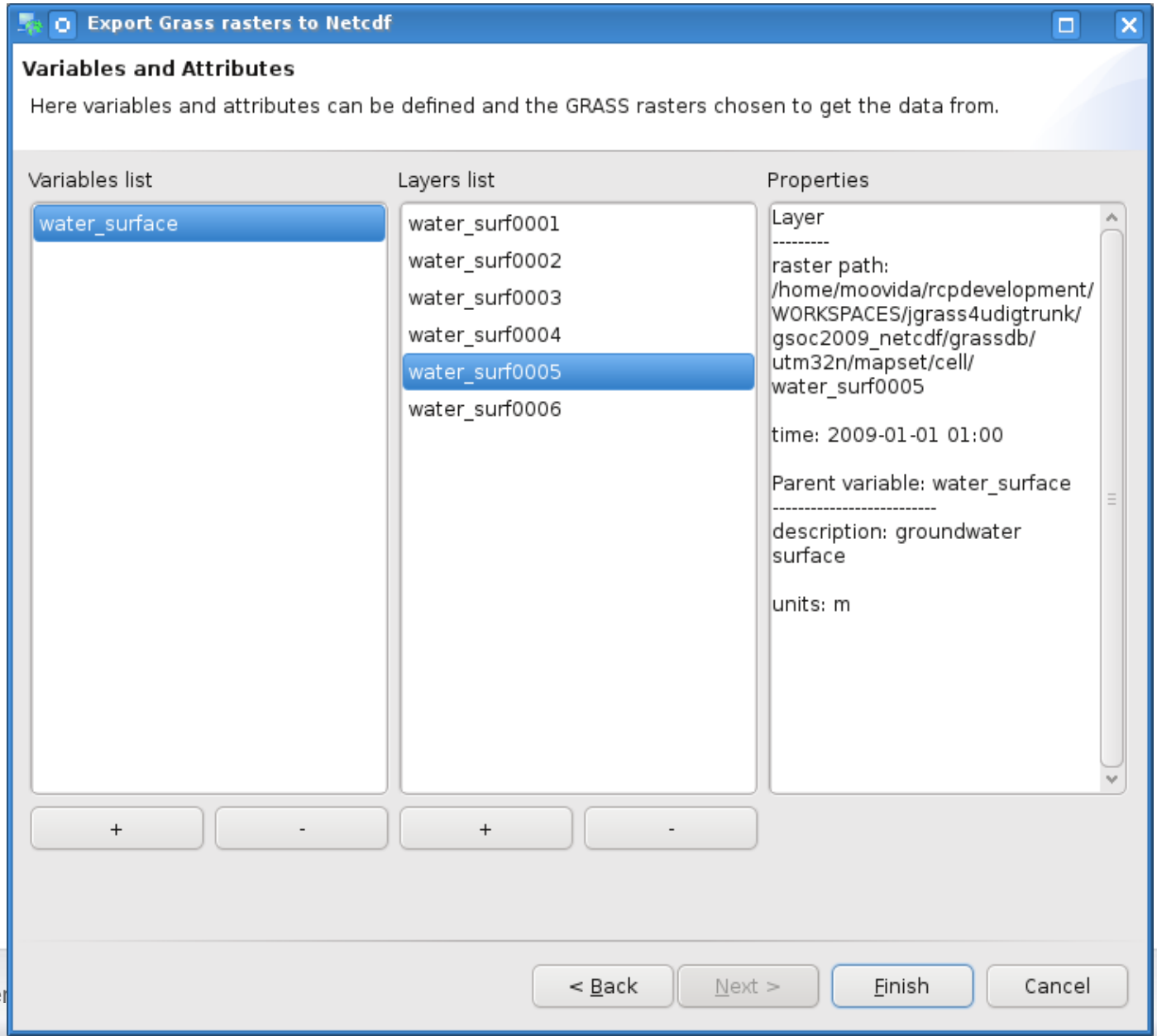
JGrass – writing netcdf files

Step 3: chose the raster maps to refer to particular timesteps, add them as layers



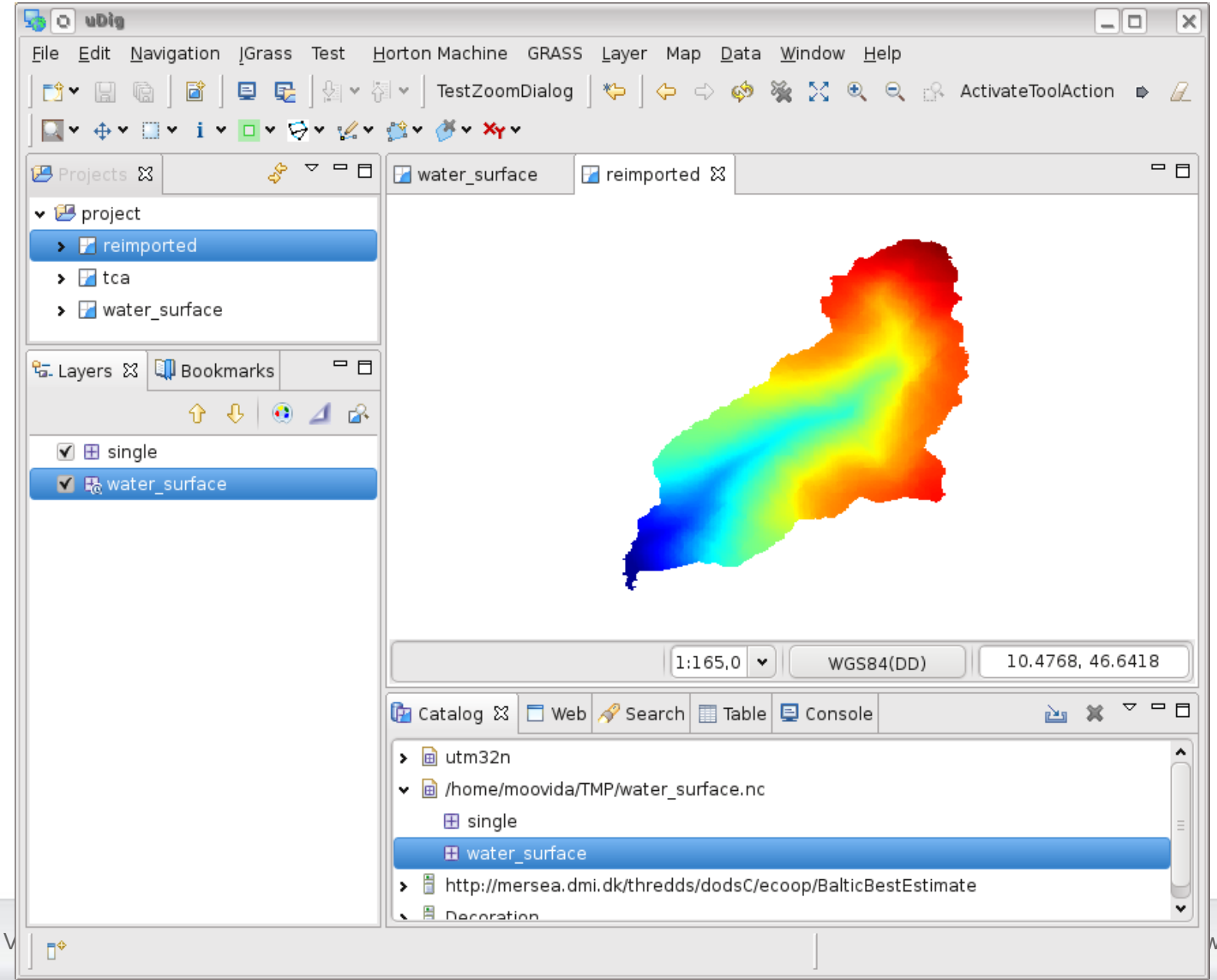
JGrass – writing netcdf files

Step 3: chose the raster maps to refer to particular timesteps, add them as layers



JGrass – writing netcdf files

Step 4: check the exported dataset



JGrass – the navigation view

current viewport boundaries

to copy/paste

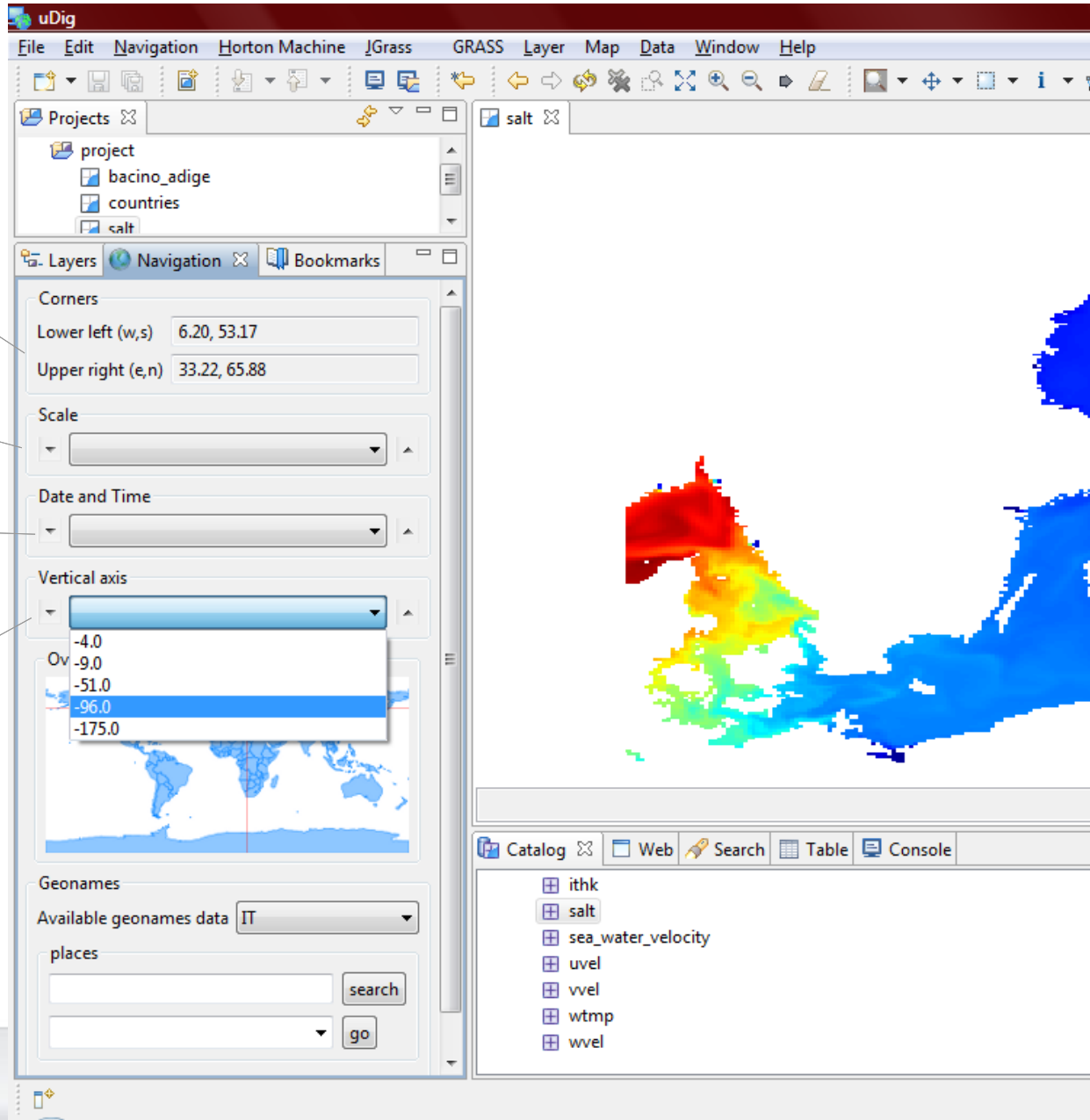
quick setting of mapscales

setting of timestamp of

the layer if supported

setting of vertical coordinate

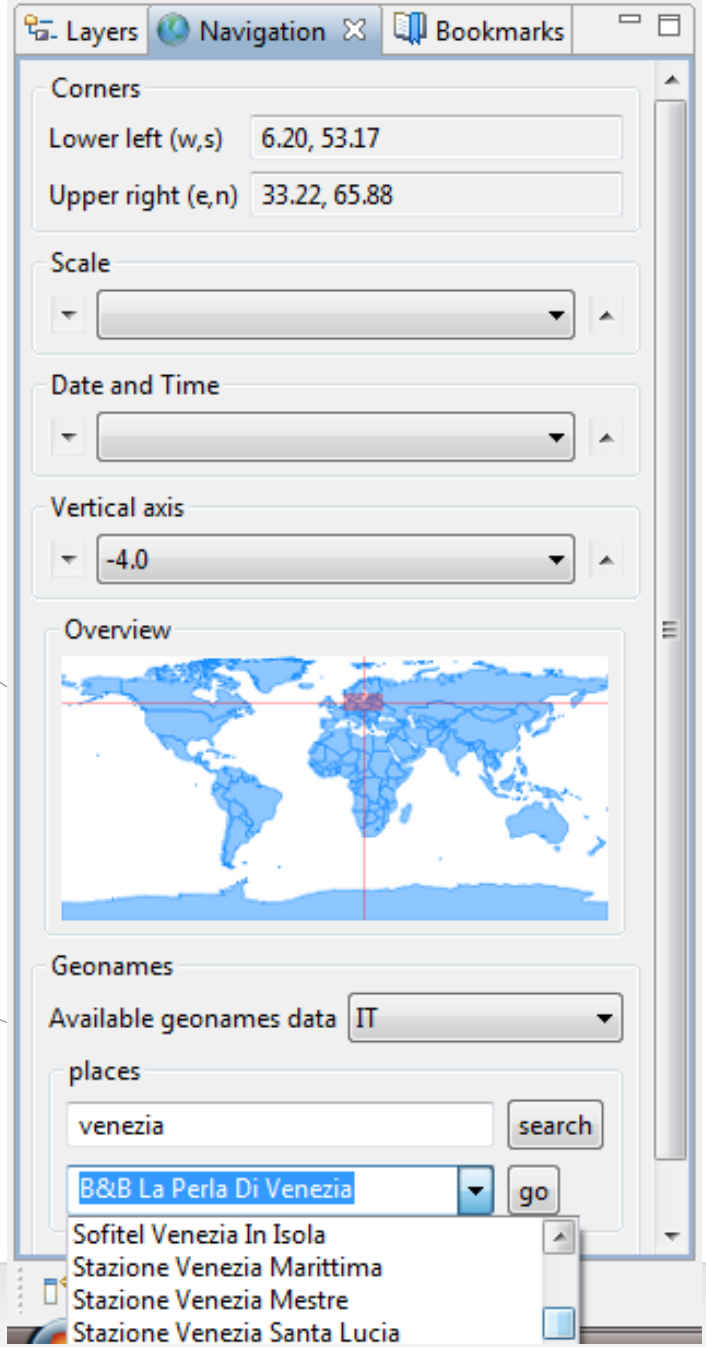
of the layer if supported



JGrass – the navigation view

world view showing the bounds of the current visualized viewport

support for browsing of/zooming to geonames (http://www.geonames.org)



JGrass – centralized repository with Ramadda

RAMADDA (**R**epository for **A**rchiving, **M**anaging and **A**ccessing **D**iverse **DA**ta) is a development effort of the Unidata Program Center. RAMADDA is freely available and provides a **publishing platform, content management system and collaboration services for Earth Science data.**

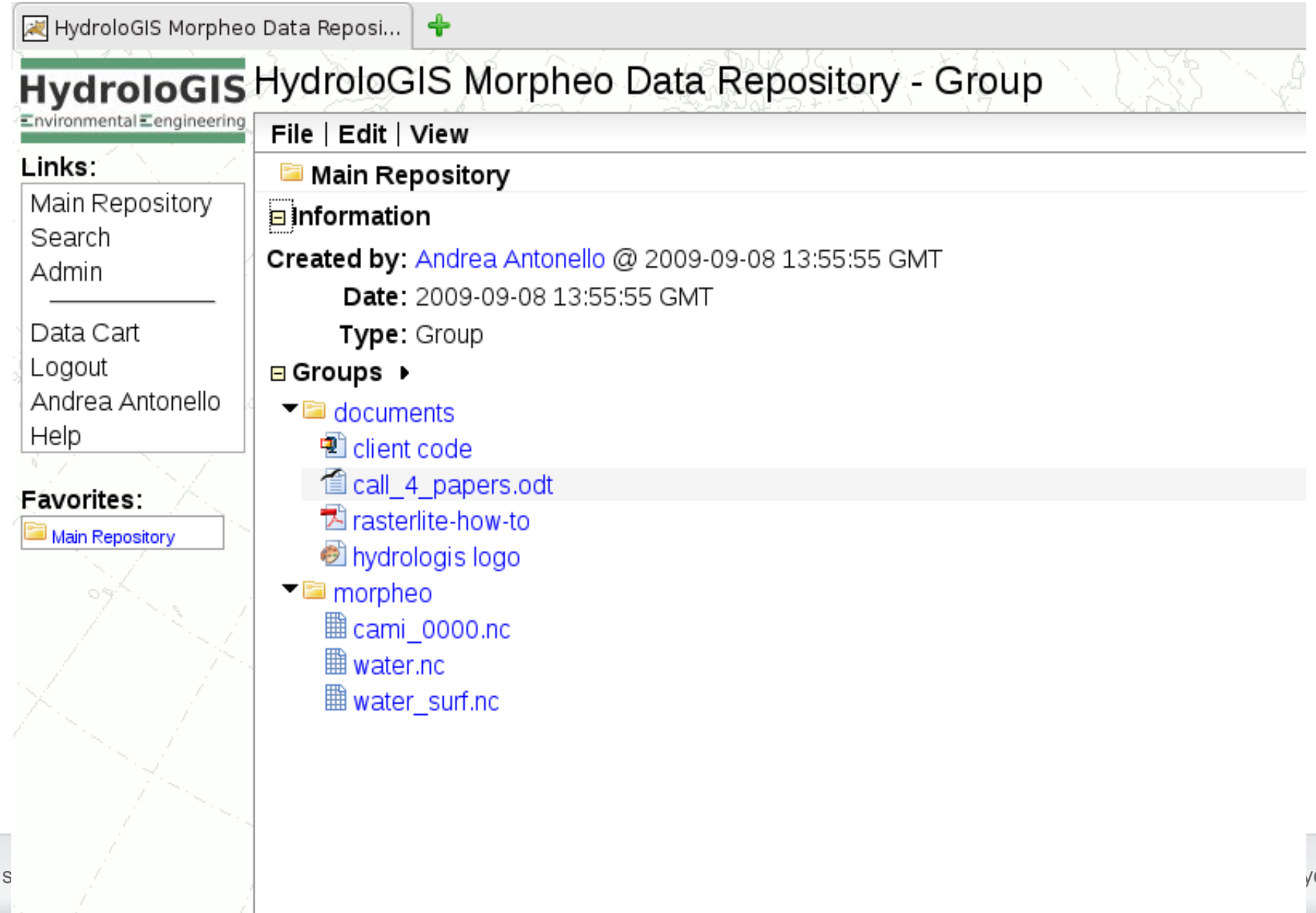
Even if Ramadda is a quite new project, it has been proven to be a good choice for centralization of results of simulations directly from within JGrass.

Ramadda can be found at:

http://www.unidata.ucar.edu/software/ramadda/index.html

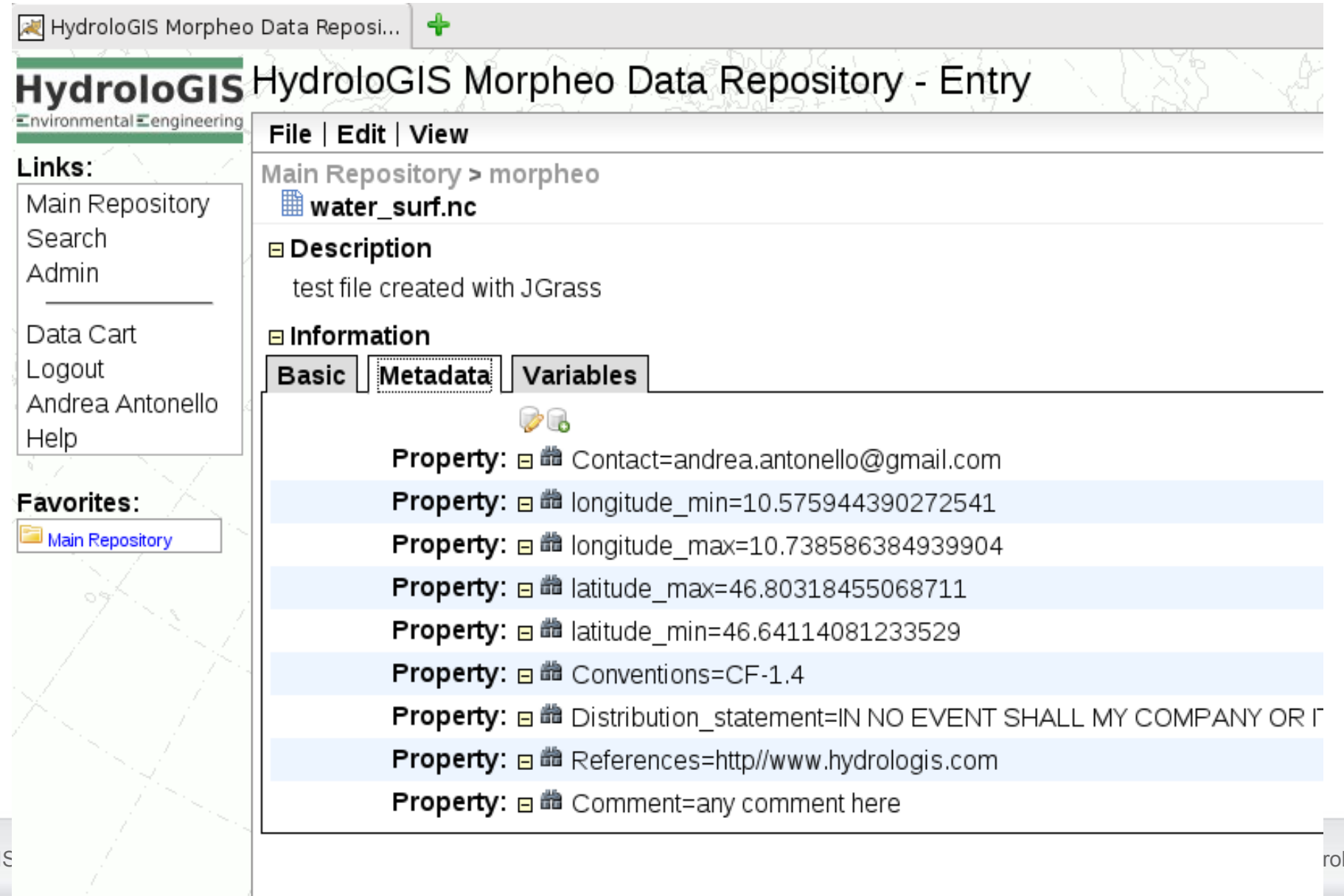
JGrass – centralized repository with Ramadda

Ramadda as seen as a j2ee web application



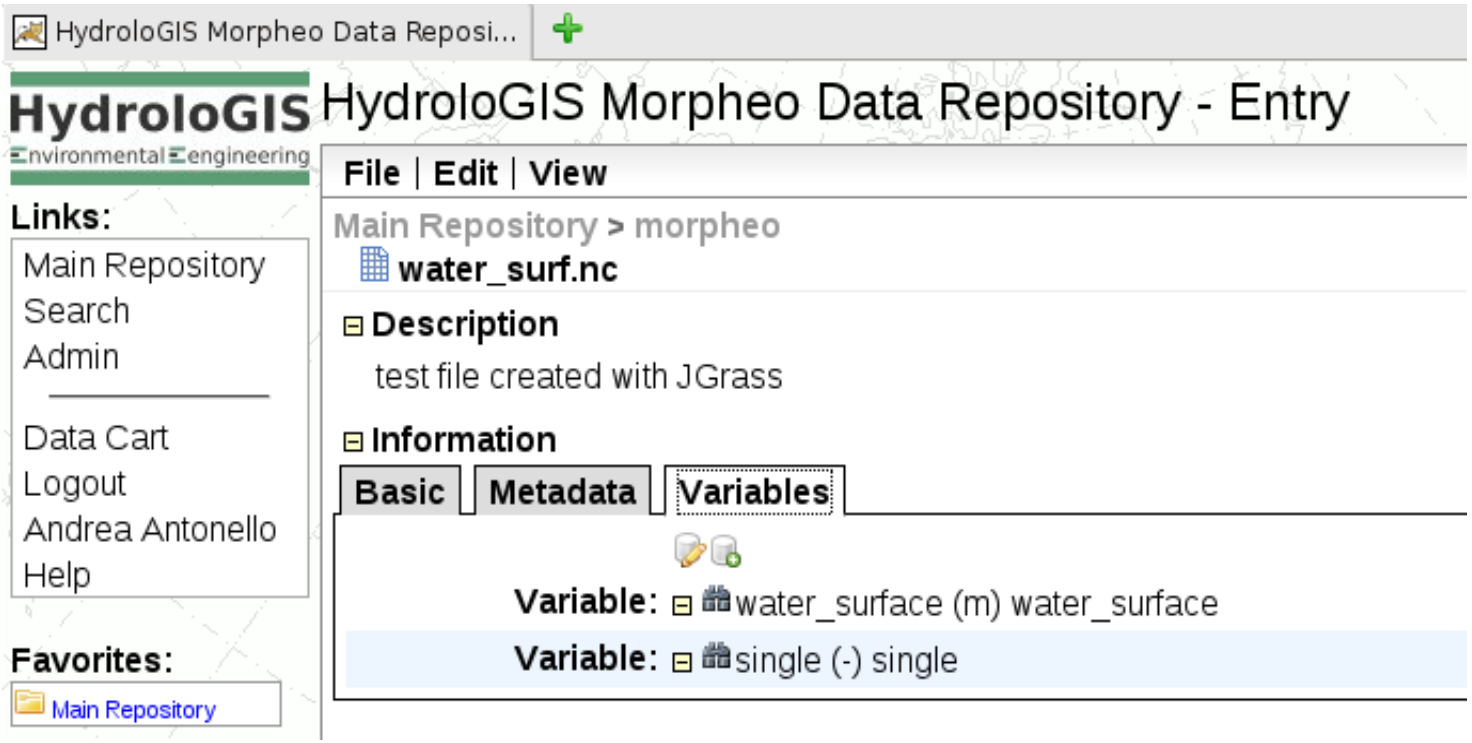
JGrass – centralized repository with Ramadda

For netcdf files great support for metadata browsing and editing



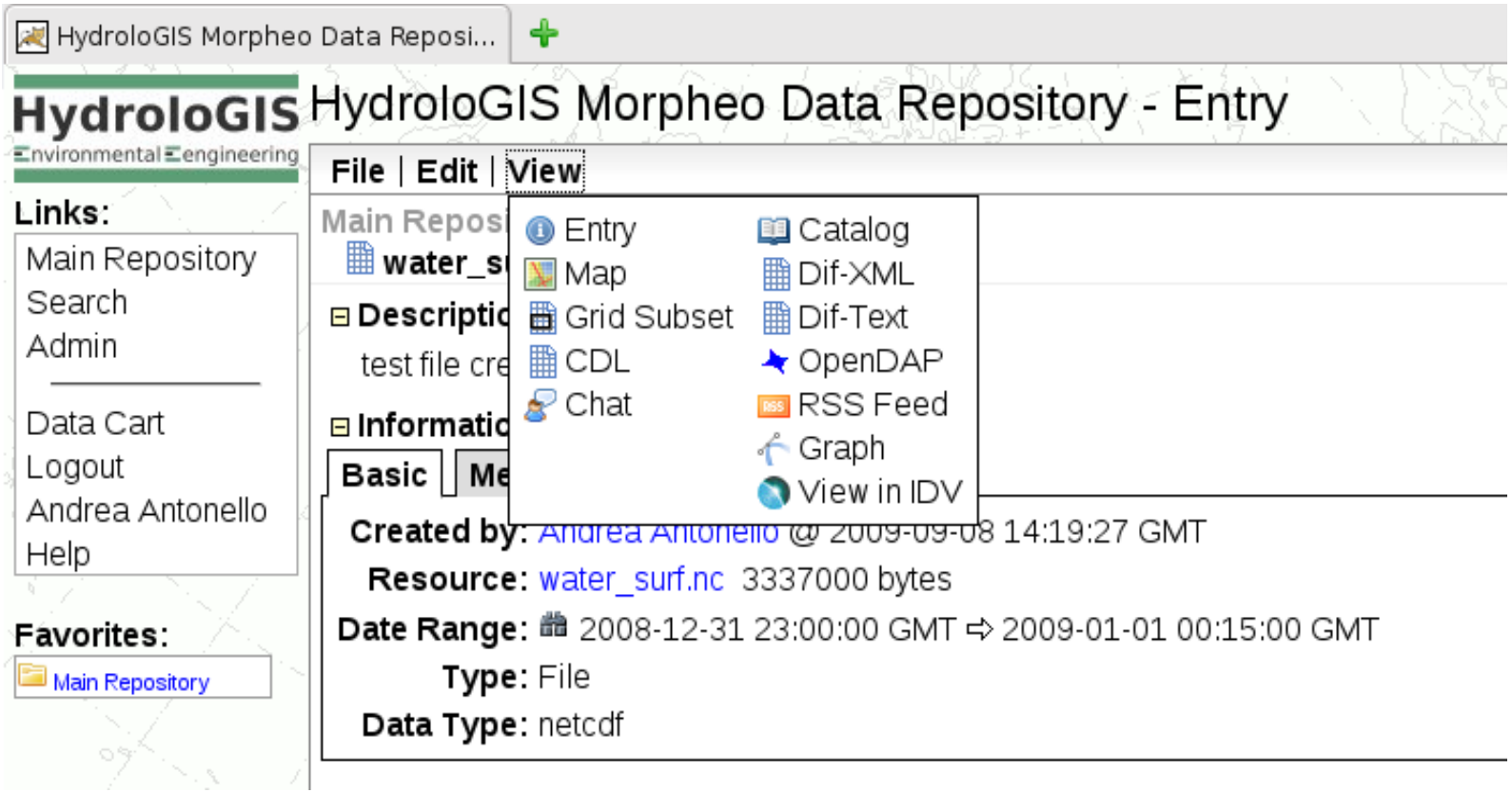
JGrass – centralized repository with Ramadda

...support for variables browsing...



JGrass – centralized repository with Ramadda

...support for remote connection via opendap (JGrass can access that), for extraction of subsets of data and more.

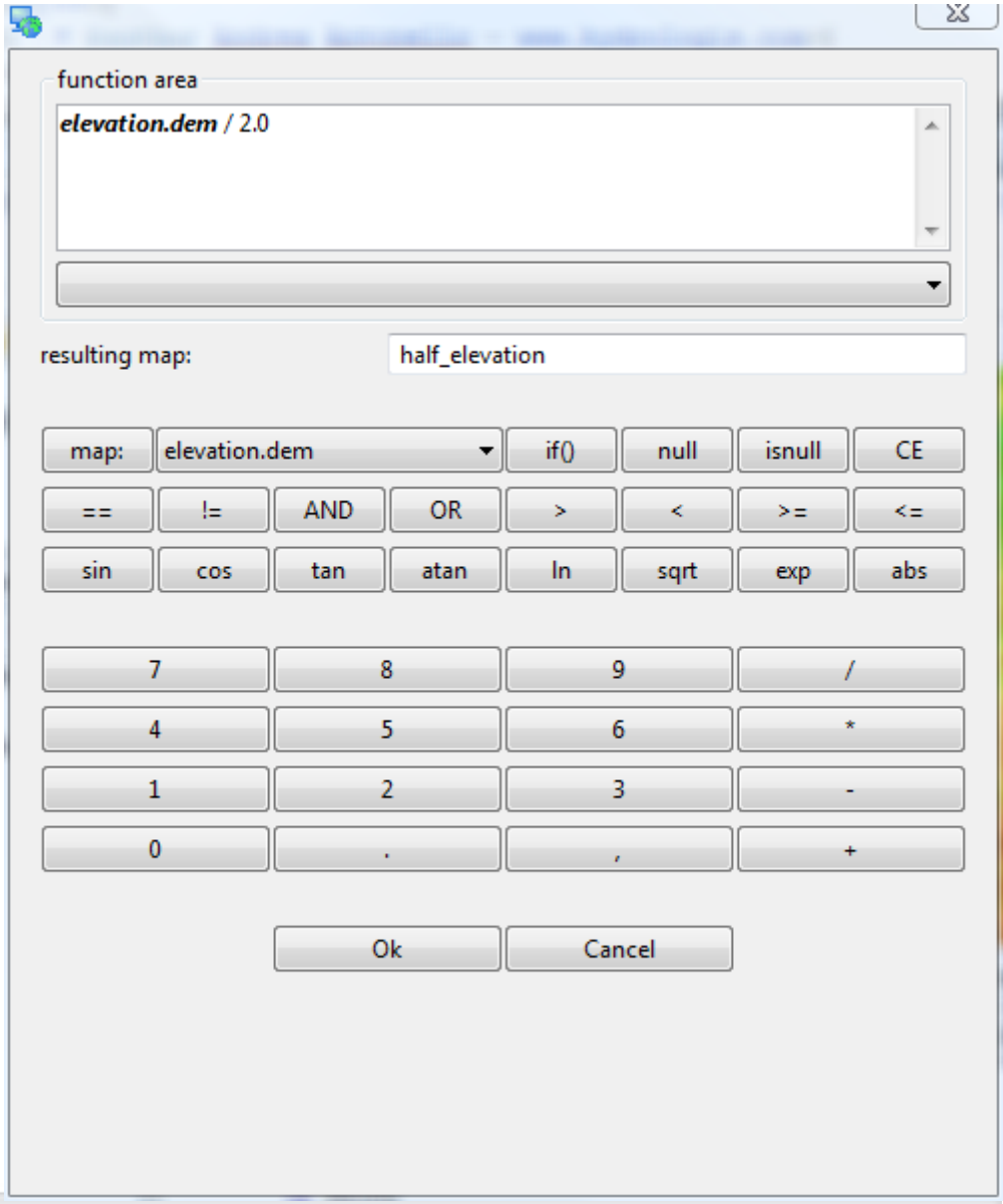


JGrass – raster mapcalculations

r.mapcalc: from GRASS over to Jiffle

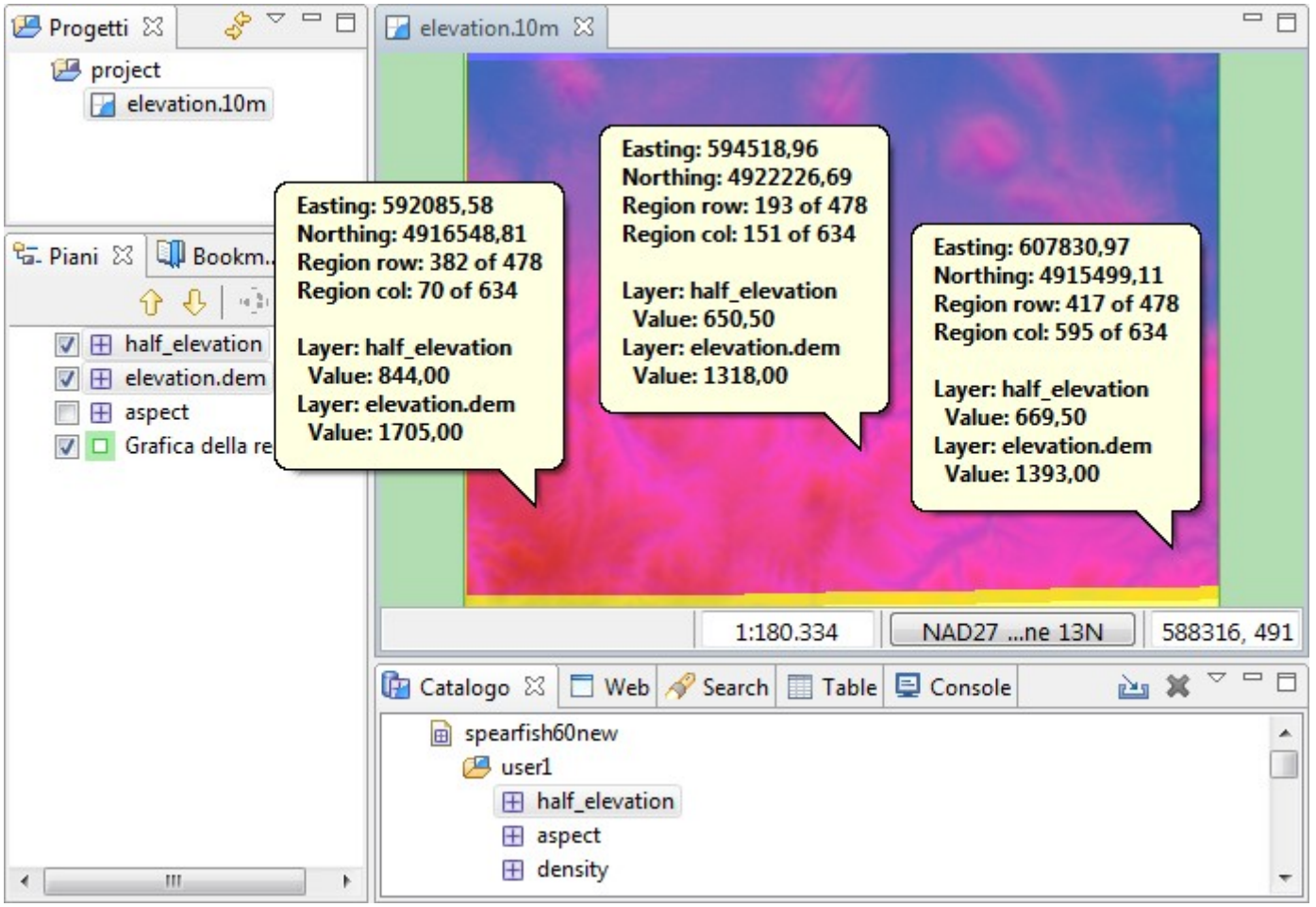
Why a substitute to the GRASS r.mapcalc?

- no control on r.mapcalc, since executed in runtime
- gave us huge random faliures
- **we want support for tiling**
- jiffle is a pure java implementation based on jai and imageio



JGrass – raster mapcalculations

jiffle mapcalculations: an example



JGrass – raster tiling

With the avenge of lidar datasets at high resolutions, it was getting difficult to do certain analyses on large datasets.

So we expanded RAM, expanded RAM and expanded RAM...

Then the JGrass raster driver to imageio in order to be able to read the data tiled.

JGrass – new models

A large project was done with JGrass to handle the water household on the second river in Italy (Adige) taking into consideration special points like dams, intakes, offtakes, artificial channels. The responsible authority agreed to leave the created code under open source inside JGrass.

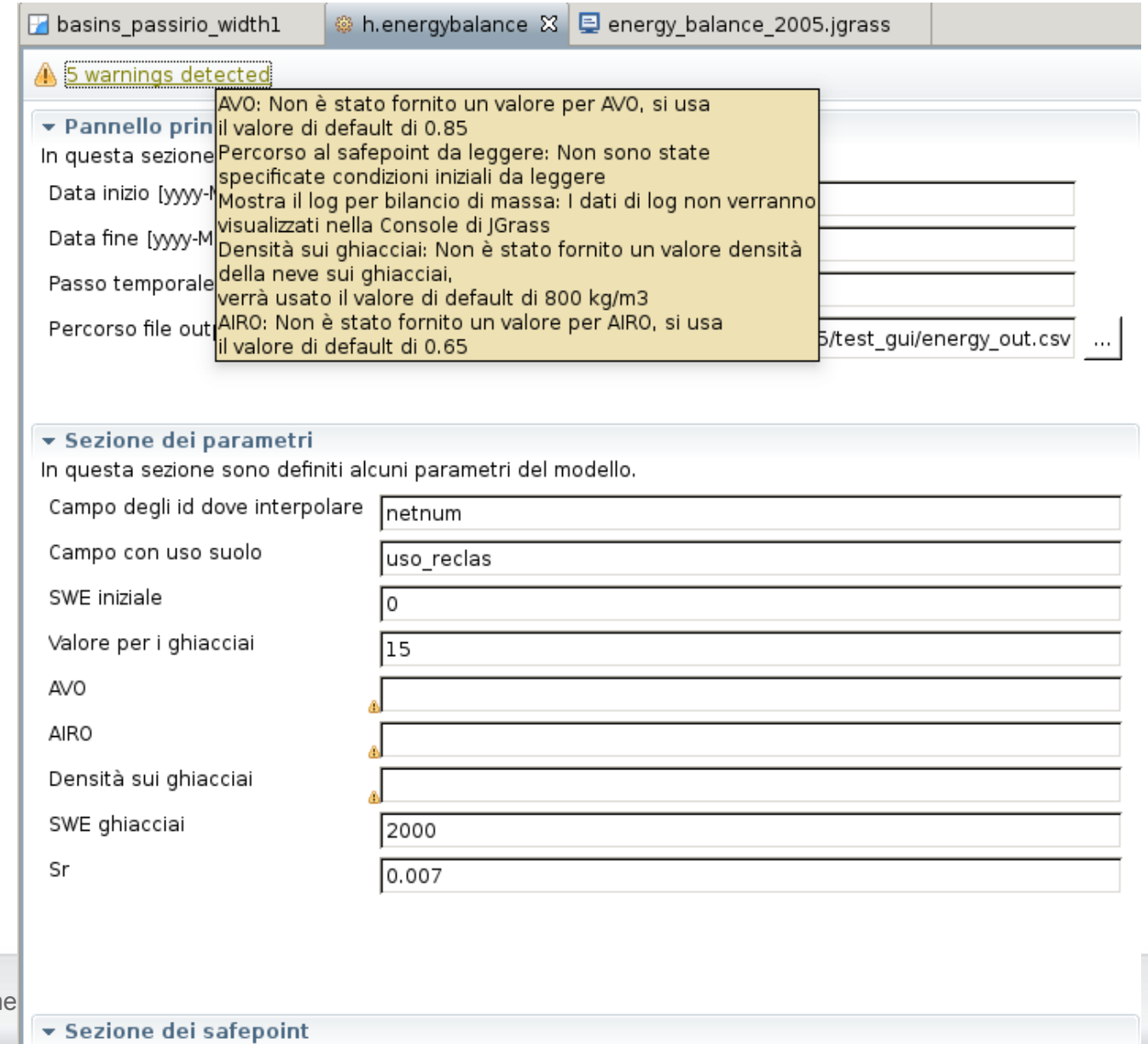
This added a bunch of new models to the JGrass modeling library as for example:

- meteorological interpolation models (h.jami, h.kriging).
- energy balance models (h.eicalculator, h.energybalance).
- discharge models (h.adige, h.santgeo).

ALL THESE MODELS IMPLEMENT THE OPENMI STANDARD

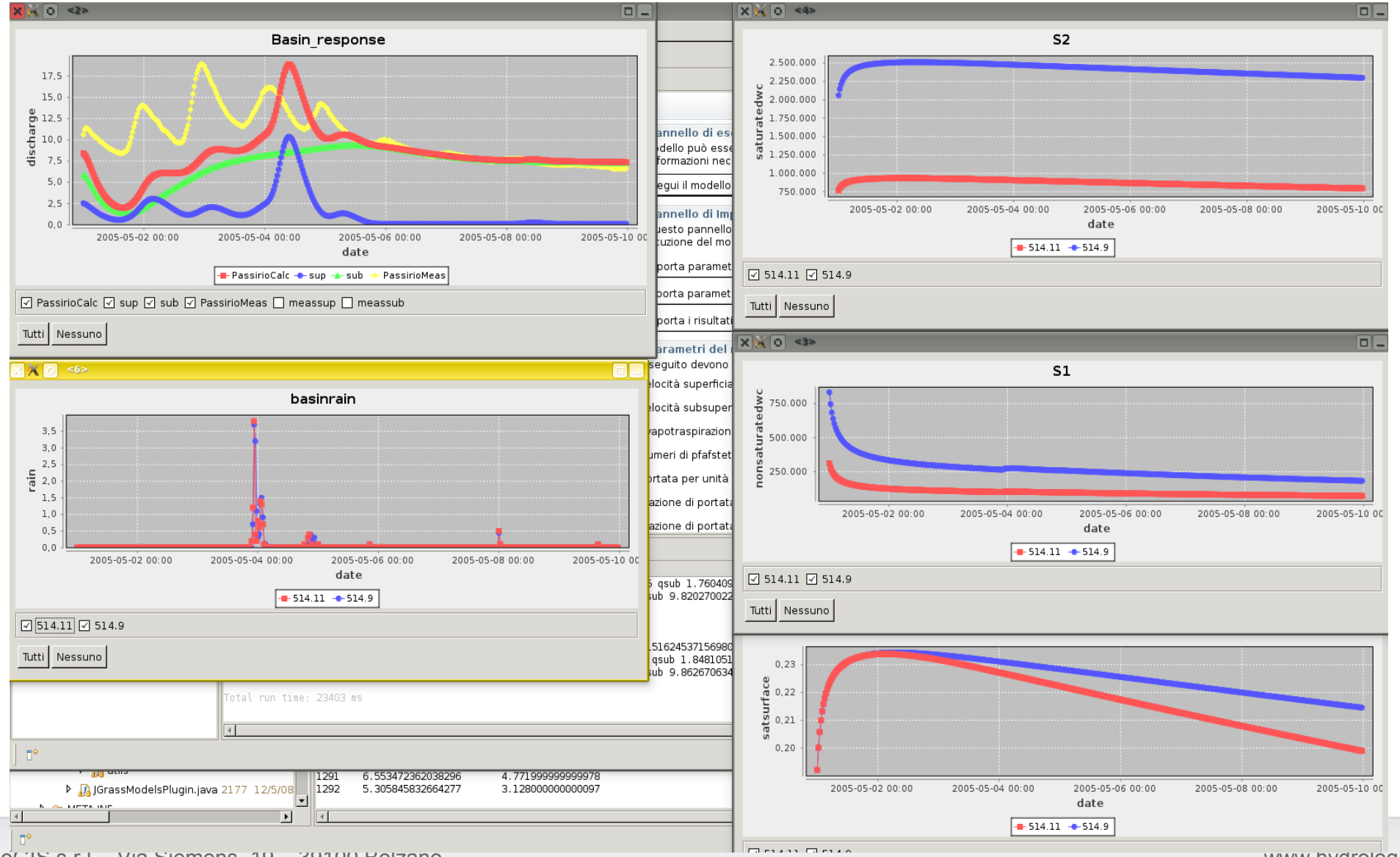
JGrass – new models

Example of eclipse forms in the h.energybalance model: inputs



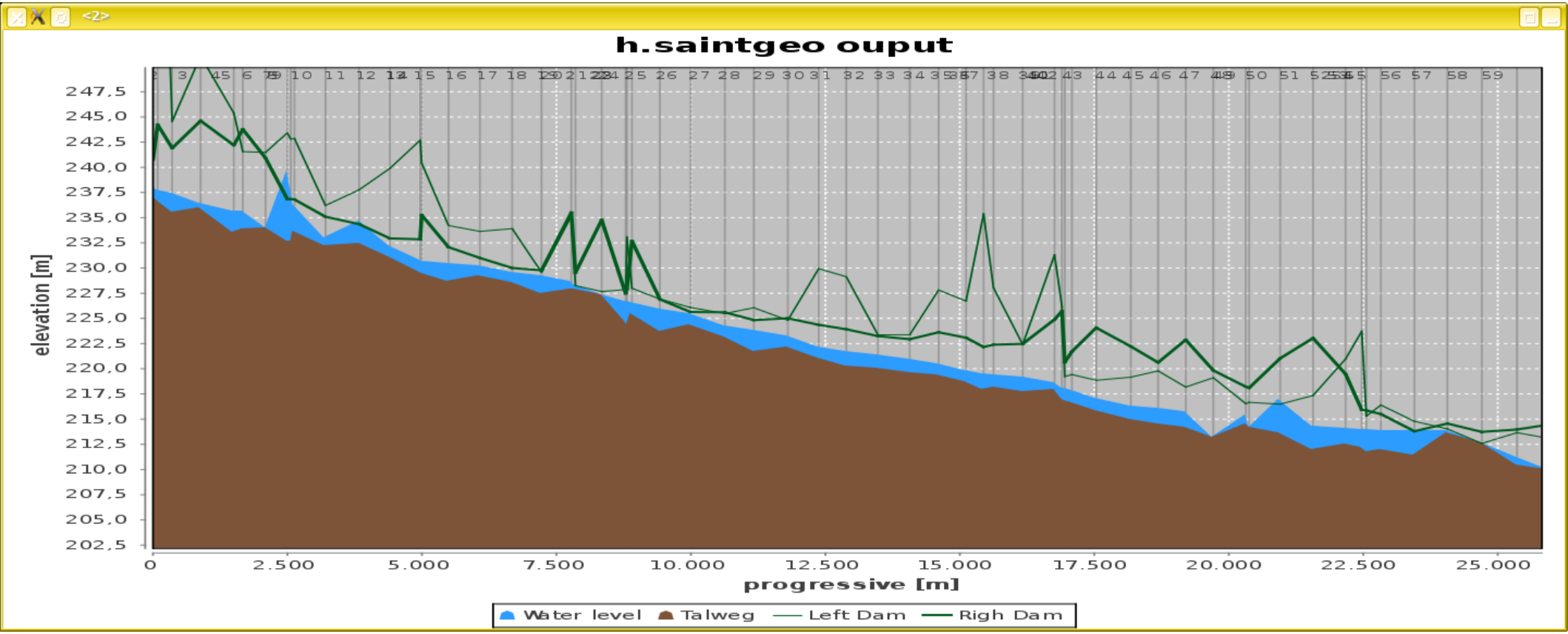
JGrass – new models

Example of the hydrologic model h.adige: results in progress



JGrass – new models

Example of the hydraulic model h.saintgeo: results in progress



Where are we heading to?

JGrass – a new strategy: towards OMS

The **Object Modeling System OMS** is a modular modeling framework that uses an open source software approach to enable all members of the scientific community to address collaboratively the many complex issues associated with the design, development, and application of distributed hydrological and environmental models.

OMS is pushed by the USDA (American Department of Agriculture).

OMS can be found at: http://www.javaforge.com/project/omslib

JGrass and OMS: what will happen to OpenMI?

A big effort has been done in the last years to bring all the models contained to OpenMI compliancy. There are several main issues that pushed the decision to migrate towards OMS:

- OpenMI forces modelers to use a quite restrictive API
- OpenMI is currently proposing its version 2, which from 1.4 introduces several changes. Migrate to that would require an enormous effort

JGrass and OMS: what will happen to OpenMI?

On the other hand:

- OMS already contains a set of components that are free and open sourced, and also already well tested at the USDA, which would come as a present to JGrass. OpenMI still doesn't have any open source components and seems to be focused on few proprietary applications
- OMS is an annotation based modern modeling framework that really focuses on adding few overhead to the modeler
- the OMS team is working on a wrapper to generate OpenMI code from OMS models

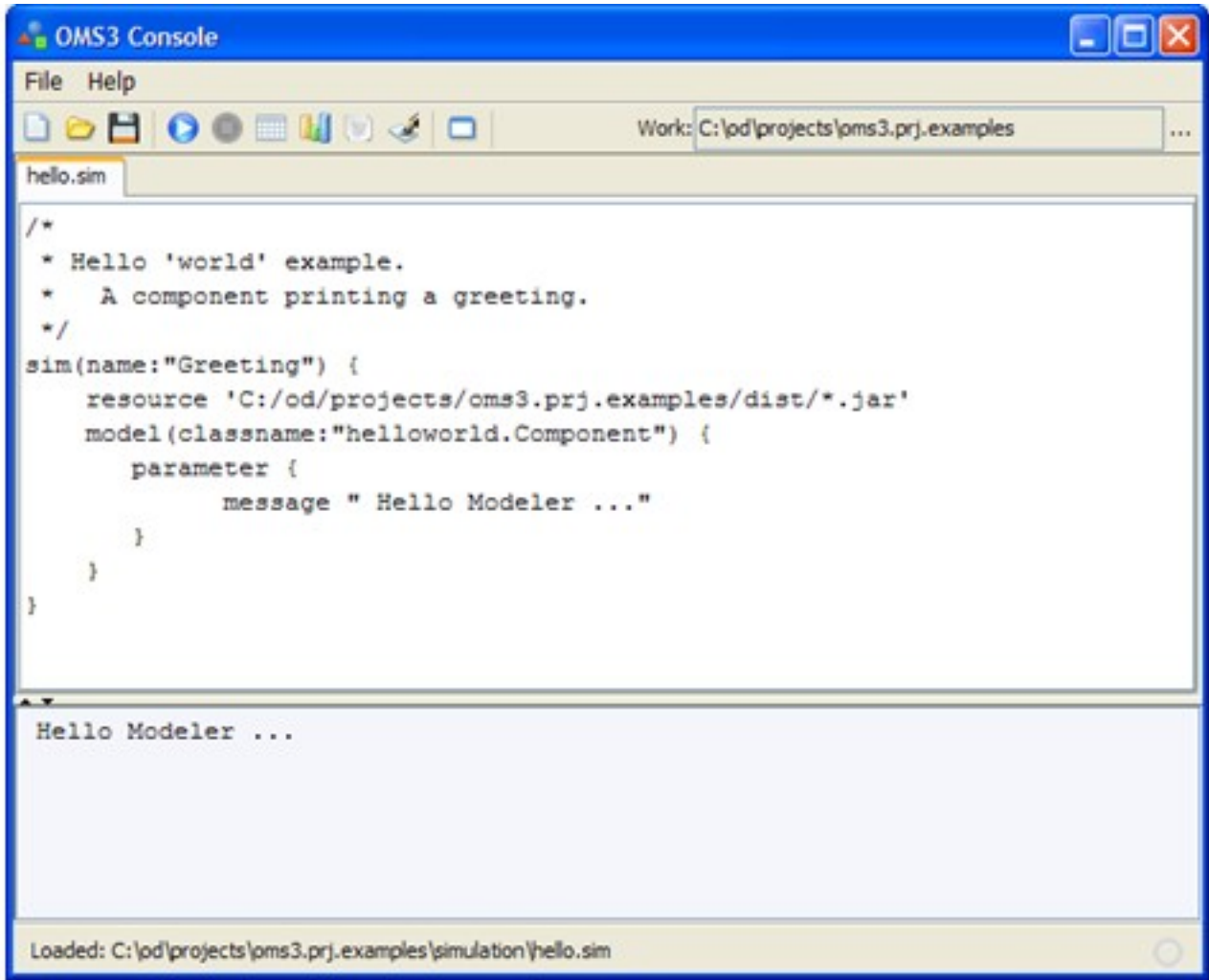
OMS: an annotations based framework

OMS minimizes the burden on a component/model developer to build code into the framework **by not imposing an API**. (I know everyone claims it, but believe me, this time it is true)

```
package helloworld;
import oms3.annotations.*;

public class Component {
        @Role(Role.PARAMETER)
        @In
        public String message;

        @Execute
        public void run() {
                System.out.println(message);
        }
}
```



OMS: other advantages

With OMS a bunch of important features come into JGrass's modeling system:

- Components always execute **multi-threaded**. If the data flow alows it, the models are executed in parallel.
- **Runtime flexibility** for simulation execution. Models can be executed in different environments that scale from a notebook to a computing cluster or even a cloud such as Amazon's Elastic Computing Cloud (EC2).

This is extremely important to JGrass, since we often need to execute models that run for hours and days, and want to exploit high performance computers or clusters.

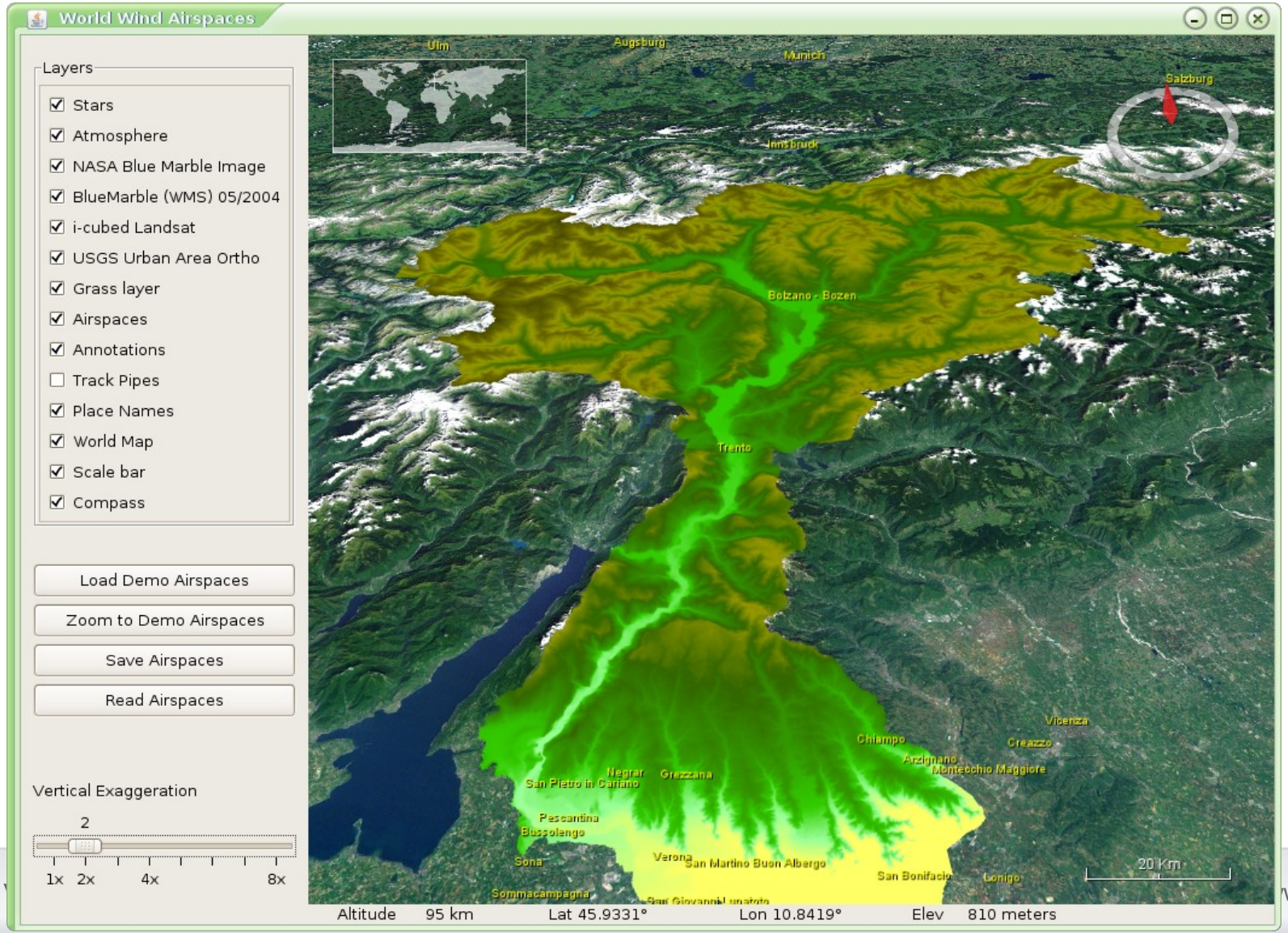
OMS: other advantages

With OMS a bunch of important features come into JGrass's modeling system:

- Components always execute **multi-threaded**. If the data flow alows it, the models are executed in parallel.
- **Runtime flexibility** for simulation execution. Models can be executed in different environments that scale from a notebook to a computing cluster or even a cloud such as Amazon's Elastic Computing Cloud (EC2).
- **Integration with JNA** (same as JGrass) for native code access. Java Native Access (JNA) integration that now supports all versions of FORTRAN, C, and C++ on all major architectures in 32 and 64 bit. FORTRAN and C/C++ programmers can continue to use their respective tools to create components
- The OMS modeler environment **bases on Groovy** scripting language, exactly as JGrass's console does
- Loading of models **libraries at runtime**

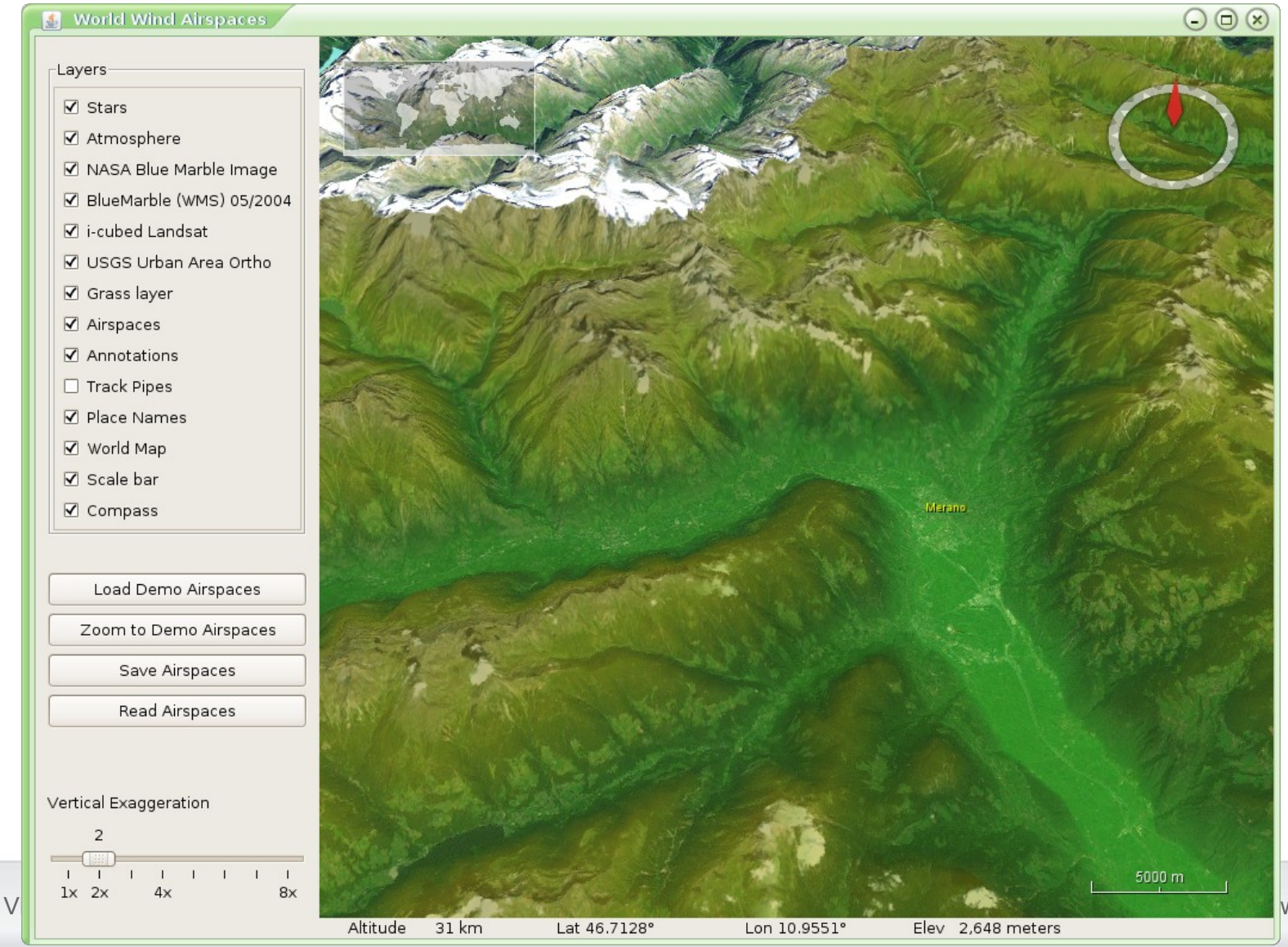
Prototyping JGrass I/O into Nasa World Wind

A small test has been done to link the JGrass I/O drivers into the NWW tiling mechanics to use it to tile and cache GRASS rasters the NWW way.



Prototyping JGrass I/O into Nasa World Wind

A small test has been done to link the JGrass I/O drivers into the NWW tiling mechanics to use it to tile and cache GRASS rasters the NWW way.





JGrass is a Free Software system which has been developed by HydroloGIS and CUDAM since the year 2003. The original community however is seeking for creating around JGrass an **ecosystem of co-developers** and **users**. In fact from the beginning JGrass was designed to serve the community, looking at a better interface for GRASS, and already made a further step in this direction **joining the uDig** community. **Beegis** is a new cooperation between HydroloGIS and the University of Urbino, that can serve as an example for other Institutions and people.