Effective Radar Algorithm Software Development at the DWD

Nils Rathmann, Michael Mott
Deutscher Wetterdienst, Frankfurt Straße 135, 63067 Offenbach (Main), Germany
nils.rathmann@dwd.de, michael.mott@dwd.de
(Dated: 31 May 2012)

1. Introduction

In 2010, with the radar network update to dual polarization radars, the Deutscher Wetterdienst (DWD) has launched the project 'Radarmaßnahmen' to implement and verify algorithms exploiting the new radar features. The project goals include the optimization of the development process. Hence, we have started to create a development-, verification- and runtime environment called POLARA (**POLA**rimetric **R**adar **A**lgorithms) to support the implementation and operationalization of radar algorithms. Thereby each new developer benefits from the experience of preceding algorithm developments. The purpose of POLARA is to avoid redundant work, standardize the look and context of new algorithms and make continuous data processing as simple and stable as possible.

So far, POLARA has been developed for two years and includes, among others, algorithms for quality assurance (Werner et al. 2012; this conference), hydrometeor detection and composite generation (see Chapter 5). The algorithm section of POLARA is furthermore going to be extended by quantitative precipitation estimation and mesocyclone detection algorithms. At this stage, the software can basically be divided into two parts: the development environment, containing a data management core, a framework for new algorithms, some helper modules and a lot of utility methods; and the runtime environment, a defined structure to process algorithms continuously with live data. How these two sections are used and how they are connected to optimize the work on new algorithms is described in this extended abstract. The next chapter introduces the concept of POLARA, followed by details about the two faces – development and runtime environment – of POLARA in Chapter 3 and 4. With the composite generation, the fifth chapter presents an exemplary algorithm implementation in POLARA. The final chapter summarizes this abstract and gives a short outlook on future enhancements of POLARA.

2. The POLARA Concept

The work on POLARA started in 2010 with four developers which were confronted with the same issue: A huge amount of real-time or verification data in different formats needs to be read, organized and processed and the results need to be written or displayed. Elementary purpose of POLARA is to avoid redundancy, make therein developed algorithms both comparable and connectable and the developers experience useful for the whole group. Hence, we decided to spend time for definitions of a common software development environment.

With POLARA, the basic programming efforts follow a defined and prepared way, from the used language to the real-time data processing. To answer some of the basic questions every new developer faces, we started with some definitions about the *how to develop*. Beginning from bottom to top, we defined Linux as development OS, C++ with GNU Compiler Collection as programming language and CMake as build system. The prerequisites for POLARA are split in libraries that every Linux system can have installed from public repositories (e.g., Boost – extended C++ libraries –) and more special libraries that are part of a POLARA reference platform (e.g., ROOT – a data analysis framework developed at CERN –) which needs to be installed manually on every used system.

Working in the POLARA development environment is regulated by some defaults like coding guidelines for C++, documentation standards in doxygen, and a confluence wiki as well as the decision for Eclipse as our common used integrated development environment. Furthermore, defined locations user specific source code and guidelines how to integrate it to the POLARA build system are provided. With these predefinition we provide a basic development environment for any new radar process developer in the DWD. In addition, it creates experience intercommunication and solves a basic problem:

When you have a question about your development environment or language, there is always someone to ask!

To have several developers working together on a synchronized source tree, we use a software revision control system. Hence, any kind of file and source code versioning is done in a Perforce depot, which can be accessed from any server in the DWD network (see Fig. 1). To process the real-time data from the German radar network, a runtime environment can be installed from a POLARA installation package to any DWD Linux system. The runtime environment consists of one process to control the file handling and another process for the scheduling of executables, implemented

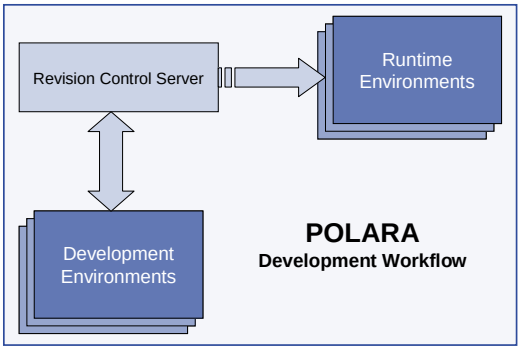

Fig. 1 POLARA development workflow

using the POLARA development environment.

Developers can use the POLARA development environment on their own system or any other Linux computer in the DWD, as long as it fulfills the prerequisites mentioned above. All developers synchronize with the Perforce depot to work on the same code basis. A runtime environment can be installed on any DWD Linux system which fulfills the same conditions as a development computer. The runtime synchronizes with the depot and builds the requested executables right on the local machine.

3. The Development Environment

The first step to make the development more effective is to provide a software environment, wherein all users can develop together and benefit from each others work. In order to achieve this, all developers use one Perforce depot to synchronize their sources. The depot is a folder tree with source code, scripts, images, configurations, documentation and the build system files. The contained CMake build system files can be used to generate Makefiles and compile executables, using the POLARA sources.

Beyond that, with POLARA we created a growing system that handles the basic functionalities of reading, organizing and writing radar data as well as generated data products or 'external' data from other measuring instruments or models (e.g., COSMO-DE). POLARA provides modules with easily accessible interfaces for logging, configuration file access and plotting. Furthermore, a growing amount of utility functions for various needs is available.

3.1 The POLARA Core

Core of the POLARA system is the data management with a data catalog that keeps metadata and data in linked, redundancy free objects. Metadata is uniquely identified by a radar site WMO number, a scan time, a scan elevation and a moment or product name.

To read data to the data catalog, a reader controller identifies the file format by file header and creates a specific reader (for BUFR, HDF5, GRIB, etc.). The reader takes the metadata from file to create a metadata tree with unique leafs. To access the data, an algorithm developer creates search queries with specific information about the required data and the data matrix is read from file and returned, if available. No matter which file type is read by the POLARA reader, it is always organized in the same data management structure and hence other modules or algorithms can handle the data in the same defined way.

The file writing functionality follows the same principle. The developer needs to know the data that should be written and picks the specific writer for the desired output format. There is no need to know more about the way data is organized internally to write it to different filetypes – as long as there is a writer class for it.

To support the use of new filetypes, the writer- and reader-controller are easy to extend.

3.2 Jobs

In POLARA, binaries with one or more algorithm, prepared to get executed in the runtime environment are called jobs. These jobs consist of an algorithm session object, which repeatedly executes and supervises a sequence of algorithms, represented by algorithm manager objects. The concept of a POLARA job is depicted in Fig. 2. The session updates the data catalog and sends requests to all registered algorithm managers. Depending on the current timestamp, each manager has to check for his algorithm, if the required data to start processing is available, if there are results that should be written to disc or if the algorithm finished work for the current run.

This loop continues until all algorithm manager return, that they are finished or the job gets an external signal to end soon. When algorithms are organized by a session, they are prepared to process live data in a POLARA runtime environment.

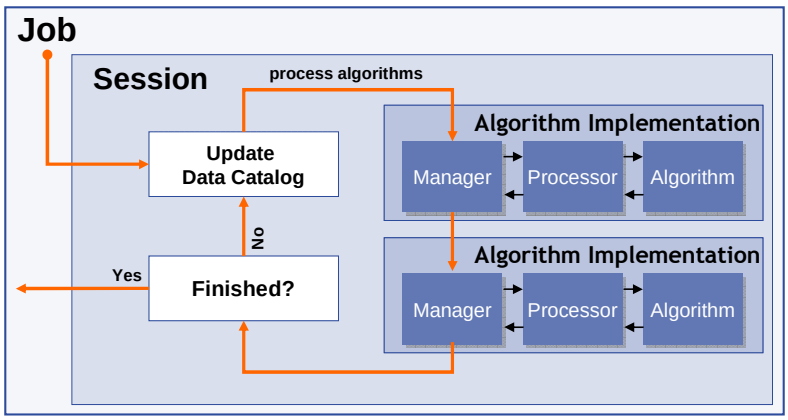

Fig. 2 POLARA job concept

3.3 Algorithms

The fundamental sense of POLARA is, beside the core and all useful helper classes, the algorithm development. To ease the implementation and use of algorithms in POLARA, a defined set of base classes is provided. These classes contain virtual methods, which have to be implemented and can be considered as a step by step guide to integrate a new algorithm to the POLARA software. Each new algorithm has to be created with a set of derived classes and implementations for thereby required methods.

The algorithm class is the actual implementation of the algorithm related calculation, based on a dataset which has been prepared by the algorithm processor. In the POLARA system, an algorithm is represented by an algorithm manager class, capable of answering questions about the algorithms state. The manager is tightly linked to the job concept, described in 3.4.

Following these guidelines has several advantages:

- Easier access to the required data
- Improved readability for other developers
- Connectivity with other algorithms in the system

3.4 Helpers

With POLARA, we tried to offer easy to use interfaces for common parts of software development, with a focus on radar data algorithm development. Part of the basic helpers is a logging module to have log files with notifications in different severities, microsecond timestamps and in a consistent look from any executable compiled in POLARA. Furthermore, it can be used to start and stop timers to measure the speed of certain code snippets.

A configuration module can be used to create configuration files in XML format, access their nodes and attributes or set default values to have default configurations created at first startup.

Several utility classes offer functionality to get information about the environment (where are log-, configuration-, image- or testfiles on disc), do some extended mathematic calculations, string cast operations, file handling and statistic functionality and many more.

A plotter module takes metadata and data from the data catalog and generates different types of plots. Depending on the data, PPI, RHI, CAPPI or B-Plots can be generated, as well as vertical profiles or scatter plots. All plots are based on read or generated data and are stored as portable network graphics.

4. The Runtime Environment

All functionality offered to the developers by the POLARA development environment can be used in tool, test or verification executables or in the runtime environment with real-time data from the DWD radar network. The POLARA runtime environment is a real-time data processing structure and can be installed to any Linux system in the DWD that fulfills the requirements of a POLARA development environment. Perforce access to update executables from POLARA source code is optional, but is an enormous improvement of the developer's workflow.

The runtime environment is organized in a folder structure with defined import and export directories and controlled by two processes. Runtime jobs process the real-time data to create products like hydrometeor detection or composites. Imported as well as created data is archived and the work done is logged. With configurations, the runtime environment or job specific behavior can be adapted to the users needs.

A filehandler organizes the files within the runtime directories, rejects invalid files, imports valid files to reference time folders, archives outdated files and removes outdated archive files. Basically, every file operation within the runtime is done by the filehandler.

A scheduler starts jobs by a certain schedule in defined intervals to process the currently valid data. Each job gets a defined time to run, will be observed during runtime and receives a signal to end processing soon to finish the latest calculations or do whatever is necessary to avoid data loss.

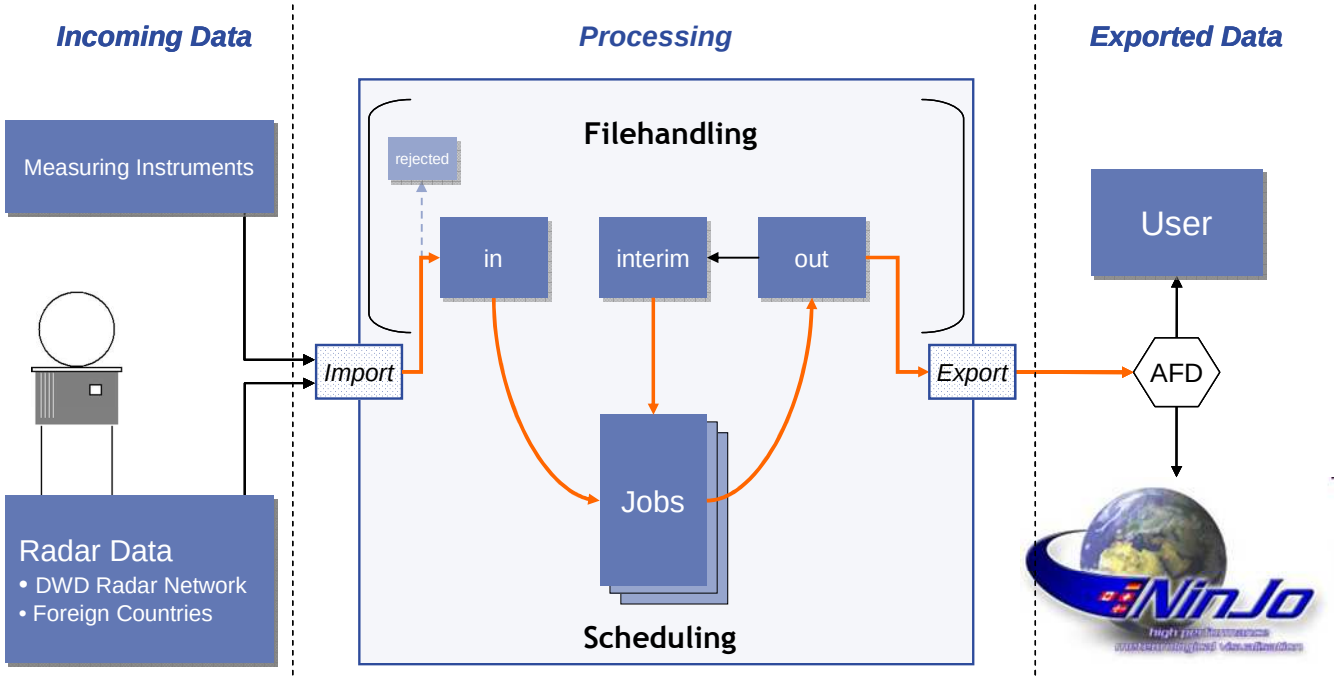

Fig. 3 POLARA runtime environment

Scripts are used to control the runtime in any required way. For example, a runtime environment contains the version of any job that is part of POLARA from the build time of the installation package. To get newer version from a certain job to the runtime environment, a script extends the runtime environment functionality with the option to update the runtime from depot, which means to get the latest sources, build the POLARA executables on the runtime environment system and copy the job to the binary folder of the runtime environment. This allows a developer to create a new job on any developer system, synchronize it with the depot and update it to any runtime environment.

Everything a developer needs to do to get a new algorithm processed continuously with real-time data, is to implement a job, synchronize it with the depot, update the job in the runtime and schedule the new job in the runtime environment configuration.

5. Exemplary Functionality: Composite Generation

Algorithms which are implemented within POLARA can either be combined in one executable to allow direct usage of one algorithm's output as input to the next one, or split to several executables. The following example describes the composite generation, implemented as a POLARA job, using the algorithm base class structure.

The composite creator is a tool to easily create products with a projected geographical context. For this purpose, methods are made available to the developer with which the transfer of the input data into the target product is enabled.

For each composite product, an entry exists in a configuration file, which defines the used basic conditions (such as projection, grid size, etc.). Another entry determines the time context and the list of input data to process. For each input data an attribute is defined which determines its validity period.

The process of creating a composite in POLARA follows the same principles like all other algorithm implementations. An algorithm manager checks at fixed time steps, if new data has arrived for processing. The algorithm processor controls the data flow to and from the algorithm (see Fig. 4).

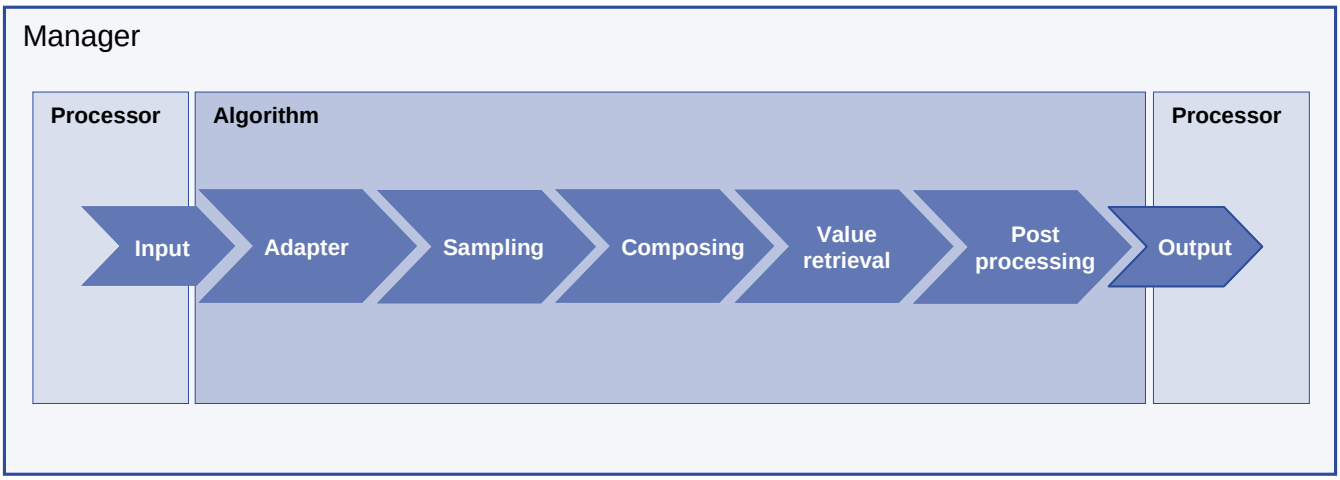

Fig. 4 Process of composite creation

The core components of the algorithm are the adapters, the sampling and composing, the value retrieval and the post processing.

5.1 Adapter

Each input data is assigned to a data transmitter. For each data transmitter, an adapter is implemented, which adapts different data structures or projections to the target product. After deployment of the input data by the processor for each date a data adapter is generated.

Moreover, it is here possible to take influence on the incoming values. For the European data transmitters, data range adjustments are made to prevent strong gradients in the overlap region.

5.2 Sampling

For every input data, the position and size is determined in the target grid. Afterwards the accompanying value is taken for the area in the target grid for every lattice cell from the adaptor.

Such a procedure is usually called a 'pull-method' (Lang et al. 2009) and has been used since 2006 in the DWD. An extension that is known as 'extended-pull-method' additionally uses quality information.

For each type of input data, it is necessary to use a special sampling method. It is divided into polar, mesh, and point-based data. The collected data values are stored in a container. The sampling can, if there is more than one input data, be executed in parallel on multiple processor cores.

5.3 Composing

The collected sample values are supplied in the composing to the target grid. Here it is also recorded which input data has been processed. This makes it possible to make a statement which data is missing.

The three steps described in 5.1–5.3 are repeated until either all the required input data have been processed or the time frame of the composite has expired.

5.4 Value retrieval

At the value retrieval, the developer controls how the values of the grid cells are interpreted. This is particularly necessary when there are multiple data values for one cell. Figure 5 shows the overlap of DWD weather radars. The dark green areas indicate that there are values from two radars, in light green regions three, in orange four, and in red regions there are values from five radars.

Several possibilities for selection exist. For example, one may use the maximum value, the value with the smallest distance from the radar, or the value with the best quality information.

The three pictures in Fig. 6 below shows the overlap of the radars Berlin (bln), Dresden (drs), Neuhaus (neu) and Eisberg (eis). In the first picture the maximum value was used, so the spokes from Dresden are completely visible. In the second picture the smallest distance from the radar is used, so the spokes from Dresden only reach the scan border. The third picture uses the value with the best quality value. So the spokes from Dresden are almost completely removed.

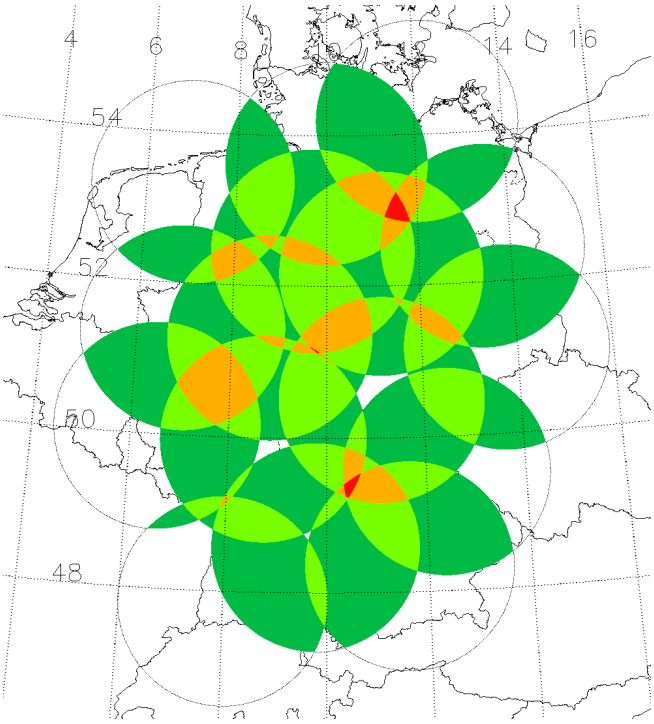

Fig. 5 Overlap of German radar network

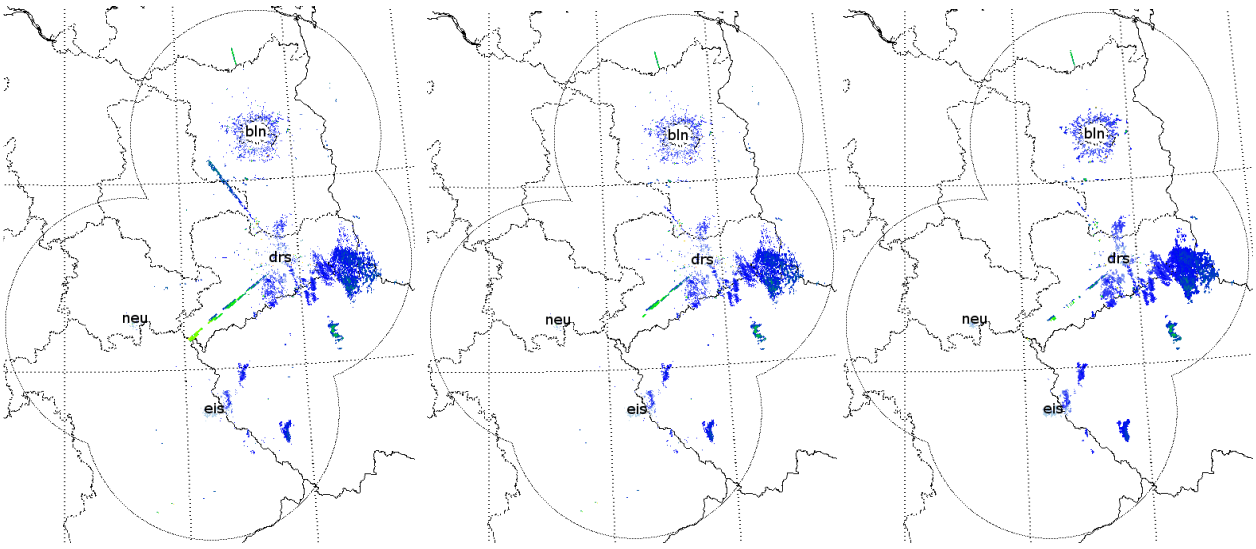

Fig. 6 ***Left:*** *Maximum method;* ***Middle:*** *Smallest distance to radar;* ***Right:*** *Quality information*

5.5 Post Processing

At the final step of the algorithm, the post processing offers more opportunities to influence the data. For example, it is possible to remove singular values or map values to classes. Here, also a border can be drawn around empty lattice cells.

Subsequent to the post-processing, the algorithm processor steers the data stream to the algorithm manager. Here the results are passed to the POLARA data catalog (cf. 3.1) and can be further processed by other algorithms.

6. Conclusion and Outlook

In the last two years, POLARA grew to a complete suite for developing, testing and running radar algorithms. The defined environment for new developers keeps the code organized and accessible for others. Implementing new algorithms and use them with real-time data from the DWD radar network is reduced to a minimum of redundant work and supported by a large amount of utility classes. The data catalog structure with related reader classes makes it easy to support new data formats without the need to change any existing code. By now, POLARA contains, among others, algorithms for quality assurance, hydrometeor detection, or composite generation and is going to be extended by quantitative precipitation estimation and mesocyclone detection.

The composite creation, described as a POLARA job example in Chapter 5 will be the first algorithm in POLARA that becomes operational and replaces legacy qualitative composite products from the DWD. To satisfy the needs of an operational job – being stable, reliable and according to the DWD IT requirements – with the same code base as jobs that are under development, the concept of POLARA needs to be extended. Hence we are working on a labelling and branching concept for POLARA and will have tested nightly builds.

References

Lang, J., Siegmund, M., Sacher, D., 2009: RADOLAN Softwarehandbuch Version 2.3.1.1