Why WDSS-II is good for algorithm development

V Lakshmanan, Kurt Hondl

January 24, 2003
The Warning Decision Support System has been redesigned to be an excellent algorithm-development platform.

We considered the needs of an algorithm developer and designed the system around that.

Some of the requirements we identified follow.
The development platform must be inexpensive.

Operating System: We use Linux. Solaris versions are also available.

Tools: We use free software tools for compilers, GUI toolkits, etc.

Hardware: Requirement is a PC with a graphics card.

Users have bought algorithm development machines for as little as $1000.
Open, Extensible and Standard Data Formats

The data formats used should be open, extensible and standard.

We use XML for small chunks of data (such as tabular data) and NetCDF for array-based data.

This makes it possible to easily accommodate new algorithm outputs. WDSS-II has ingested and output disparate data sources, such as multi-sensor SCIT, supercell identification, TDWR data, and advected fields with *no source code modification* to the underlying codebase.

There are APIs already available to read and write these formats. There are tools available (besides WDSS-II) to examine the output files.
Connect to Real-time system

It should be possible for an algorithm developer to connect to a real-time system without having to be responsible for maintaining that real-time system.

Data is normally fetched for the algorithm from remote locations. Thus, it is easy for an algorithm developer to get going – they don’t need to somehow maintain a data feed.

Through LDM, we do provide the capability for the user to have a clone of the entire system on the desktop.
Archive and Real-time

It should be possible to switch between archive sets and real-time data streams using the same binary. The algorithm developer should not be required to run a data feeding algorithm to get data to her algorithm.

WDSS-II algorithms can switch via a command line option to the algorithm ("-r").

WDSS-II algorithms can be directly run on an archived case on disk (except for multisensor simulations ... )
Simulate Multi-Sensor Streams

It should be possible to easily simulate multi-sensor streams from archive.

WDSS-II provides a tool that does this simulation, so that the algorithm processes the data as if it were real-time.
It should be possible to incorporate developed algorithms into a real-time system of algorithms, or run them independently.

WDSS-II algorithms can currently be packaged into a real-time system using base data obtained using LDM – this is what we are doing at NSSL, and in the Norman and Jackson Forecast Offices.
What about the ORPG?

Until the funding for this effort ran out, WDSS-II was able to provide base data and products from an ORPG to algorithms. This capability can easily be resurrected.

There were a few problems with our effort:

1. No easy way to make an archive of ORPG products, since products in LB were aged off if tape was long.

2. The tape and the ORPG had to be run everytime the algorithm was.

3. The ORPG had to be on the user desktop. There were serious problems with running an algorithm as a user other than the orpg user.
4. The algorithm has to be on the same machine as the ORPG – there is no remote access capability.
It should support multisensor and multisource ingest for algorithms.

WDSS-II does this extremely well since it was designed with this capability in mind:

1. Dealiasing (experimental algorithm using both reflectivity and velocity data from same radar)

2. MrSCIT (“Multi-radar Storm Cell Identification”)

3. w2segmotion (Multiscale storm identification and forecast – runs on either radar reflectivity or on the infrared window channel)

4. SuperCell (Incorporates radar data, along with the outputs of other algorithms)
5. No body has integrated radar and satellite at the same time (capacity is there though) – work on integrating model inputs.
Easy Development and Testing

It should support easy development and testing of new algorithms. In the next section, we will demonstrate some of the capabilities of the system that really help an algorithm developer.

Instead of showing possibilities, we will show case studies of how people have actually used our system.
Easy Development and Testing of Algorithms: Case Studies

Noteworthy is that in all of this, there is no source code changes to anything other than the actual algorithm.
This is the architecture of a new algorithm in WDSS-II. The idea is that you use APIs for input and output and code only the scientific analysis part.
New Products

The Linear Least Square Derivative can be computed from Doppler Velocity data.

This is a completely new product. Travis Smith wrote a WDSS-II algorithm to ingest velocity data and to compute and write out NetCDF files of LLSD divergence and rotation data. He used existing WDSS-II APIs both for the ingest and output.

After he created a colormap file for his output, he could see the results...
Case Studies: LLSD

of his algorithm.
Visual comparision

He could also compare it visually to the original Velocity fields.
Quantitative Comparison

Quantitatively, he could look at the actual data values:
In fact, by writing another configuration file, he could make it so that he could filter just a single Velocity product.
He could even run his algorithm on TDWR data:

(He didn’t; I did ...)
That’s a NEXRAD algorithm!

How could an algorithm developed on NEXRAD run unchanged and unmodified on TDWR data? Because the WDSS-II system abstracts out the commonality of the two radar systems.
So, for example, both Reflectivity and Velocity data are represented by the RadialSet class, which is stored as NetCDF files.

How are Nyquist velocity and VCP represented? As “attributes” of the RadialSet, since they will not be present in every RadialSet.
Visual Programming

What happens if I smooth the image using the Gaussian filter before contouring it? This can be done simply off the one-off menus by clicking the filters consecutively.
Cascade filtering

One-off filters can be cascaded together before integration is done at the source code level.

Any WDSS-II algorithm can join the list of one-off filters and thus be combined for visual cascading with any other algorithm.
How about tabular data?

The LLSD was image-type data. What do we provide for tabular data? The supercell algorithm was developed in WDSS-II, and like LLSD uses WDSS-II APIs for ingest and output.

Inputs are base reflectivity, mesocyclone detection algorithm, tornado detection algorithm and BWER algorithm outputs. Thus, it is a multisource algorithm.

All kinds of data are ingest easily and in a consistent manner using the API.
Table Display

Like with image products, with writing one config file, tabular data can be displayed.
The configuration file allows for complex relationships, like coloring one column based on the data value of another.

Of course, the tabular data can be written out into XML using the API. There is no need to write a completely new format for every new algorithm output – there is an abstraction for tabular data just as there is an abstraction for Radial and Gridded data.
Having written out tables, trends and past tracks are automatically extracted and displayed.
The algorithm developer does not need to do anything special to generate trends. To obtain tracks, he should mark the column which provides the “name” of the row (such as the cell id).
Rank filtering

In fact, it is possible to filter the detections based on any of the parameters in the table, saving the need to keep re-running the algorithm with different parameters.
This is very convenient for algorithm development.
**Intermediate products**

Because the classes are extensible, and very generic, it is possible to output intermediate products, just for visual examination.
This is also an example of a multisensor algorithm – note that the developer marked detections based on the sensor they are obtained from.
Ground truth

Superimpose shapefiles containing ground truth (in this case tornado path) over the data and algorithm outputs.
Why not WDSS-II?

1. Not officially blessed.

2. Need to migrate your algorithm to other operational systems, e.g.: ORPG. For example, the output format would have to be an ICD. (The WDSS-II API can ingest ORPG base data and products).

3. Will the operational systems permit extensible data formats? They are typically less efficient than the proprietary ones. The extensible WDSS-II classes are less efficient than a C structure crafted for the exact product.
So Why WDSS-II?

Higher productivity for algorithm developers.

How?

1. Easy set up of an algorithm development environment.
2. Develop on archive cases and real-time from the very beginning.
3. Examine intermediate products easily.
4. Easily switch your algorithm to process data from different systems.
5. Easily implement multisource, multisensor algorithms.
6. Use extensive library from WDSS-II and the C++ STL for data manipulation.

7. Use the WDSS-II display’s one-off filters to do preprocessing and postprocessing, without having to rerun your algorithm.

8. Uses open, extensible and standard data formats – XML, NetCDF and shapefiles. Okay shapefiles are not really extensible.

9. Distribute your data.

10. Test your algorithm packaged in a real-time system.
The Future

Can the WDSS-II API be part of an operational system? One possibility is for it to be an extension to CODE.

Can it evolve with the needs of algorithm developers? With continued funding, yes. We have incorporated new data streams, new abstract types, and new algorithms and tools into WDSS-II.