7.6 EXPANDING THE POWER OF AWIPS WITH PLUGINS

U. Herbert Grote*
NOAA Research - Earth System Research Laboratory, Boulder, CO

Mark McInerney
NWS, Silver Spring, MD

1. INTRODUCTION

The National Weather Service AWIPS (Advanced Weather Information Processing System) has been modified many times to accommodate changes in user and data requirements. Often, these changes were implemented with little more than modifications to tables, but others required detailed knowledge of the software design to properly modify the code. Over the years these modifications, including the migration to the Linux operating system, have contributed to increasing the complexity of the code and the size of some software modules.

The objective of the work described herein is to develop an approach that simplifies development and integration of more powerful new applications with AWIPS code. These applications would be able to generate manual graphics, create charts, and display image data, and overlay them appropriately on other data displayed on AWIPS.

2. PLUGIN CONCEPT

Web browsers have used plugins for years to add specific capability to the browser that are not native to the browser application. These plugins can be integrated easily by downloading the desired software, executing an install script, and usually restarting the browser. The browser does not require the plugin unless the user desires the specific capability it provides. Another important characteristic is that these plugins are typically developed by various vendors and usually not the developer of the browser.

* *Corresponding author address:* U. Herb Grote, Earth Systems Research Laboratory, 325 Broadway, Boulder, CO 80305; e-mail: U.Herb.Grote@noaa.gov

Extending this concept to AWIPS means that developers outside of GSD (ESRL Global Systems Division, formerly Forecast Systems Laboratory) would be able to add significant capabilities to AWIPS/D2D without knowing the detailed internal design of D2D. Furthermore, the integration of the new feature would be as easy as placing the code in a particular directory and restarting AWIPS; otherwise known as plug-and-play. This is a major departure from the current approach where the developer is either on the AWIPS development team or works closely with key individuals to develop and integrate the new capability.

3. AWIPS ENVIRONMENT

The D2D software has three fundamental ways of adding display functionality to the workstation: depictables, extensions, and applications. Depictables are software modules (objects) that generate the various types of displays and are probably the most complex modules to develop. They are part of the core software and require links to most of the AWIPS libraries. Examples of depictables include radar, satellite, model data, and station plot displays. Extensions are interactive depictables that run as separate system processes but are closely tied to the D2D core software (Kelly, 1997). Frequently used extensions are WarnGen, Baselines, and Points. The most loosely integrated software is D2D applications. Applications can instruct D2D to load a known display or create their own display in a separate window, but they cannot modify or edit information on the screen. An example of a D2D application is the volume browser that allows an AWIPS user to select model and other data for display.

One of the reasons for this particular architecture is that when D2D was developed the goal was to make the software heavily object-oriented. This meant

that classes could be derived from a base class and objects could be exchanged between processes. Thus, extensions would be derived from a parent class that included inter-process communications and other features. It was hoped that this would simplify the writing of extensions. In practice, it required some degree of familiarity with the extensions base class and C++ programming skills. It also meant having to link with many of the D2D software libraries to compile the extension, increasing the integration complexity.

4. ARCHITECTURE MODIFICATION

Implementing the plugin concept required the development of a new interface to D2D to ensure the necessary separation of the plugin code from the display process. This meant that the interface could not use any of the existing D2D libraries and had to be independent of the programming language. This would allow applications to be developed without having to link to the many interdependent D2D libraries, and "wrappers" could be written for the interface code to support different programming languages.

The first API (applications programming interface) that we implemented was the graphical interface to the display process. Future developments will include an interface library for accessing data on both local disks and remote servers. In order to support a comprehensive drawing capability, the graphical interface needed to include the ability to draw multi-color graphics. This required significant modifications to D2D since all D2D graphics were historically treated as single-color overlays. The original display code was designed for a display card that supported an 8-bit image buffer and several single-color graphic overlays. When the D2D code was ported to Linux, the 8-bit images were converted to 24-bit true color images, but graphics continued to be tracked as single-bit overlays. Multi-color graphics were emulated by linking and concurrently displaying several single-color overlays.

The challenge of displaying multi-color graphics was solved by treating them as transparent images. Each graphic image includes the alpha channel information which specifies the amount of transparency to be applied to each pixel. To render the graphics to the screen, each graphic image is blended with the other data already being displayed. The resulting image is a single true color image, with its image components retained to allow toggling and graduated blending. Although this resulted in increased memory requirements, some memory was reclaimed by removing the obsolete code for 8-bit pseudo color images.

The API also includes code that handles socket communications between the plugin and the display process. The design allows multiple plugins to run concurrently, each appearing on the screen as a separate overlay. Only one interactive plugin can be active at any time.

The initial API provides basic graphic drawing capabilities for circles, lines, rectangles, etc. and is approximately at the level of the Java Abstract Window Toolkit, but at a higher level than the X11 library. Figure 1 illustrates a simple graphic annotation using a drawing plugin. Capabilities for drawing fronts and other meteorological symbols will be implemented as layers on top of the current API.

The goal is to convert all D2D extensions into plugins and eventually eliminate the extensions interface. To date, most of the extensions have been converted collectively by creating a new base class. Some consideration is also being given to replacing D2D depictables, but that is not within the scope of the current project.

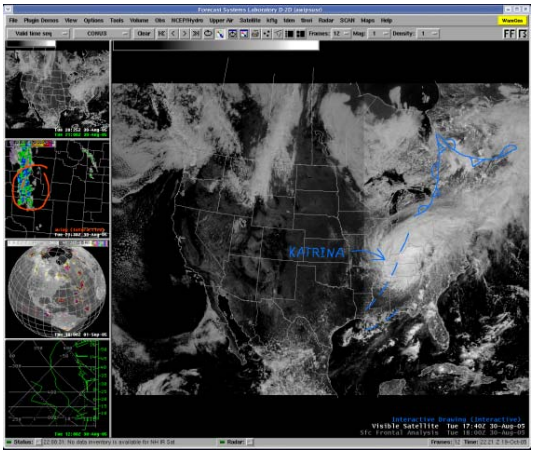

Figure 1. Simple Graphic Annotation on AWIPS

5. DEVELOPMENT ENVIRONMENT

Plugins can easily be developed outside the AWIPS development environment. A plugin developer's package has been created that consists of the API library, interface documentation, a stand-alone plugin driver to emulate the D2D display, and some sample plugins. Developers can use this package to create plugins on a variety of different machines. The plugin driver maintains a display window that renders and responds to the plugin instructions. Although the plugin driver implements a complete API, it does not perform every function of the D2D display. It is expected to be sufficient for initial testing of newly-developed plugins, but final testing must be done with D2D. Software integration entails no more than placing the compiled plugin code in a specified directory and restarting D2D. The plugin manager will detect the code and start it, and also terminate it when instructed. The menu will automatically be updated with the specified name of the plugin. Unless a specific menu location is identified, the plugin will be added to a default menu.

6. FUTURE DIRECTION

The development of the plugin interface is part of the ALPS (Advanced Linux Prototype System) project (Grote et al., 2005), which, among other things, includes prototyping remote data access. A goal is to include the data access interface in the plugin developer's package. This would allow developers to use a standard library instead of developing their own data interfaces.

The utility of many existing meteorological applications could be enhanced by converting them to plugins. This would allow them to be overlaid on other D2D meteorological data and leverage existing D2D interactive capabilities.

Some possible candidates for integration as plugins are:

- Hydroview, which displays point hydro-meteorological data such as river and stream gage readings, precipitation amounts, and similar environmental information that is typically observed and forecast for a specific location.

- ATCF (Automated Tropical Cyclone Forecasting System) which provides graphical displays of tropical cyclone track, fix, and forecast information, as well as synoptic fields and rawinsonde observations.

- SLOSH (Sea, Lake, and Overland Surges from Hurricanes), which estimates storm surge heights and winds resulting from historical, hypothetical, or predicted hurricanes.

- FFMP (Flash Flood Monitoring and Prediction) displays such information as precipitation and flash flood index.

A Plugin Application Repository and Automated Delivery System could provide a secure online service to manage NWS field- and laboratory-developed plugin applications. Developers would be provided a method for submitting plugins, which after formal review and testing would be available for download to operational sites. The installation download process could be automated, more in line with today's software updating process where sites can check for version updates of desired plugin applications, select, and install. Such a process has large potential to track and manage plugins installed (or not installed) at field sites, manage version control, and more.

At the moment, D2D plugins can replace only D2D extensions, but it may be possible in the future also to replace depictables. This could lead to a Common AWIPS Visualization Environment for D2D where plugins are the main source for generating data displays.

7. SUMMARY

Plugins provide a unique capability for creating and integrating new data displays without requiring deep knowledge of the AWIPS D2D software. Many applications that currently run outside of D2D could be integrated using the new interface; as more developers begin to integrate existing and new plugin applications into AWIPS, we plan to continue to enhance the current API. The features described herein were developed as part of the ALPS project, using AWIPS Operational Build 6 as its baseline. No dates

have been defined when this capability may be made part of the operational AWIPS.

8. ACKNOWLEDGEMENTS

The author wishes to acknowledge the efforts of others, especially James Ramer from the Cooperative Institute for Research in the Atmosphere, Gerry Murray with Systems Research Group, and Michael Romberg from Global Systems Division.

9. REFERENCES

Grote, H., D. Davis, C. Bullock, and J. Tuell, 2005: An advanced Linux prototype of AWIPS. *21st Int. Conf. on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology*, San Diego, CA, Amer. Meteor. Soc., CD-ROM, 8.4.

Kelly, S., 1997: An object-oriented framework for local extensions to the WFO-Advanced forecaster display workstation, D2D. *13th Int. Conf. on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, Long Beach, CA, Amer. Meteor. Soc., 324-327.