# 8A.4 An Exercise in Extending AWIPS II

Jim Fluke*, J. Edwards* and X. Jing[2]
NOAA Earth System Research Laboratory (ESRL), Boulder, CO, USA
*Collaboration with the Cooperative Institute for Research in the Atmosphere (CIRA), Fort Collins, CO, USA
[2]Collaboration with the Cooperative Institute for Research in Environmental Sciences (CIRES), Boulder, CO, USA

## 1. INTRODUCTION

A modernized version of the Advanced Weather Information Processing System (AWIPS), is being developed by Raytheon. This new version of AWIPS is officially called AWIPS Migration but will be referred to as AWIPS II in this paper. AWIPS II is based on a Services Oriented Architecture (SOA) that is modular, flexible and easily extensible. AWIPS II, like the current AWIPS, is comprised of two distinct parts, a data server and a display. On AWIPS II, these are the Environmental Data Exchange system (EDEX) and the Common AWIPS Visualization Environment (CAVE), respectively. Functionality is added to AWIPS II via plug-ins, and Raytheon has provided Plug-in Creator tools that facilitate this capability.

Over the past year or so, GSD and other development organizations have been working with the different versions of AWIPS II to become familiar with the system. As part of a training exercise, GSD wrote two types of plug-ins: one to ingest data, specifically an observation dataset from the Meteorological Assimilation Data Ingest System (MADIS); and another to display and interact with various datasets. In this paper, we discuss the steps necessary to generate the plug-ins, the problems we encountered, and some proposed solutions.

## 2. EDEX MADIS PLUG-IN

Creating an EDEX data ingest plug-in requires generating both Java source files, to decode the data, and a number of configuration files that are used by EDEX to determine how to run the Java code and what do to with the result. To help with this, Raytheon has developed an EDEX Plug-in Creator that runs under Eclipse (http://www.eclipse.org). It is intended to be used to create the framework for a specific EDEX plug-in that will ingest a specific type of data. It creates stubs for all the Java source files required for the plug-in, and creates all of the needed configuration files. As the first step in creating a MADIS ingest plug-in, we used the Plug-in Creator to create the starting framework. Then we populated the Java stubs with code that parses the XML formatted MADIS data. The first attempts to build the plug-in showed many errors. We determined that most of them were caused by problems with the configuration files. The same thing happened in the first attempts to run the plug-in. We were able to correct all of these problems, but it took a large effort to track them all down.

A report, written by James Fluke, showing the detailed steps needed to create the plug-in is on the web at: http://www-sdd.fsl.noaa.gov/~fluke/noaa/MadisPlugin.html.

Figure 1 shows a small sample of XML formatted MADIS data. We obtained it from the MADIS web page at: http://madis.noaa.gov/, using the Text/XML Viewer (account required). On the web page we specified the standard surface variables and XML format.

*Corresponding author address:* Jim Fluke, NOAA/ESRL/GSD, R/GSD4, Boulder, CO 80305; email: James.Fluke@noaa.gov

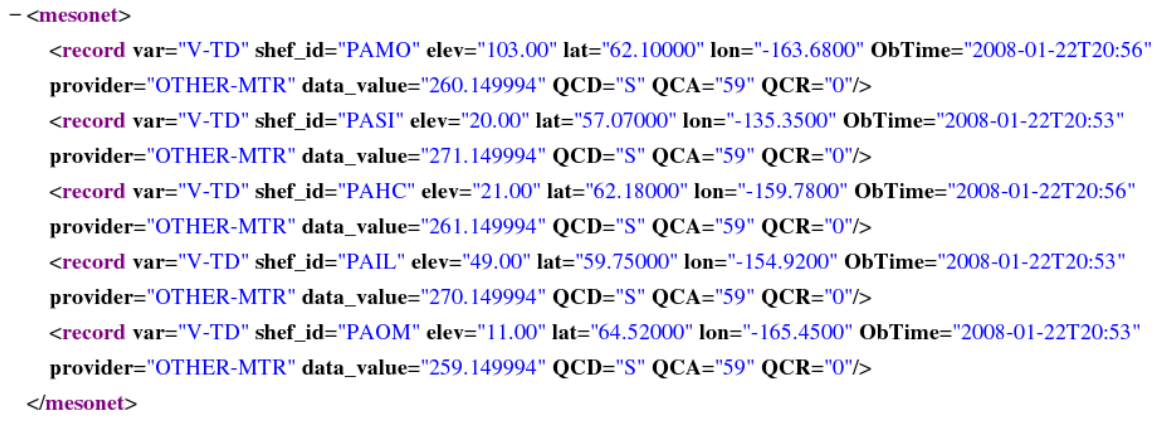


*Figure 1: Sample of XML formatted MADIS data.*

Although XML is not the format of the MADIS data read off the Satellite Broadcast Network (SBN), using it allowed us to learn about XML parsing under Java as part of the exercise.

Figure 2 shows the EDEX Plug-in Creator with three of the MADIS fields filled in. Note that the “Ipersistable” button is unchecked. This is because the MADIS data is only stored in the PostgreSQL database, not in the Hierarchical Data Format 5 (HDF5) repository, as is consistent with all the other types of point data. Also note that the "Create Separator" button is unchecked. This is because it makes more sense to send XML data to a standard XML parser in its entirety. It is pretty clear that anything else would require more code and more complicated code. Finally, the "Data URI" check box is selected for all of the fields shown so that all of them will be part of the Uniform Resource Identifier (URI) for the MADIS data.

The two Java files MadisRecord.java and MadisDecoder.java were created by the Plug-in Creator. MadisRecord.java was nearly complete and needed only minor changes. MadisDecoder.java was a stub, as expected, and we had to write the code for parsing the XML formatted MADIS data and converting it to a list of MadisRecords. We created an additional class, MadisParser.java, to help with the parsing.

One problem we encountered was determining what field types were valid. We had to examine the Plug-in Generator code to determine that only string, float, double, and integer, or int (all case insensitive except for int) are valid field types. Entering an invalid field type and then selecting "Generate Plug-in" will cause plug-in generation to fail, but there is no error dialog or log message to indicate why it failed. Replacing the "Field Type" entry field with a menu button in the generator dialog would mitigate this problem. More informative log messages would also be very useful.

Date/time types were not available, and one was needed for the MADIS “ObTime” field. To deal with this problem, we initially specified the field as a String type, and then we had to manually edit the configuration files to change it to a Java Calendar type. The Plug-in Creator should be extended to include a date/time field type.

When we attempted to build and then run the plug-in, other problems were encountered that were more minor than these, but that still took significant time and effort to resolve. We specified these in the TestTrack Report (TTR) written for the MADIS plug-in problems - TTR 405. Again, most of these were in the configuration files. We found fixes for them by examining the error messages and by comparing the MADIS configuration files to those in the existing plug-ins written by Raytheon. Changing the MADIS files to be more consistent with the Raytheon plug-ins provided most of the fixes. It is clear that the Plugin Creator could be fixed to populate the files without these problems, and will need to be if adding EDEX plug-ins is to be easy.

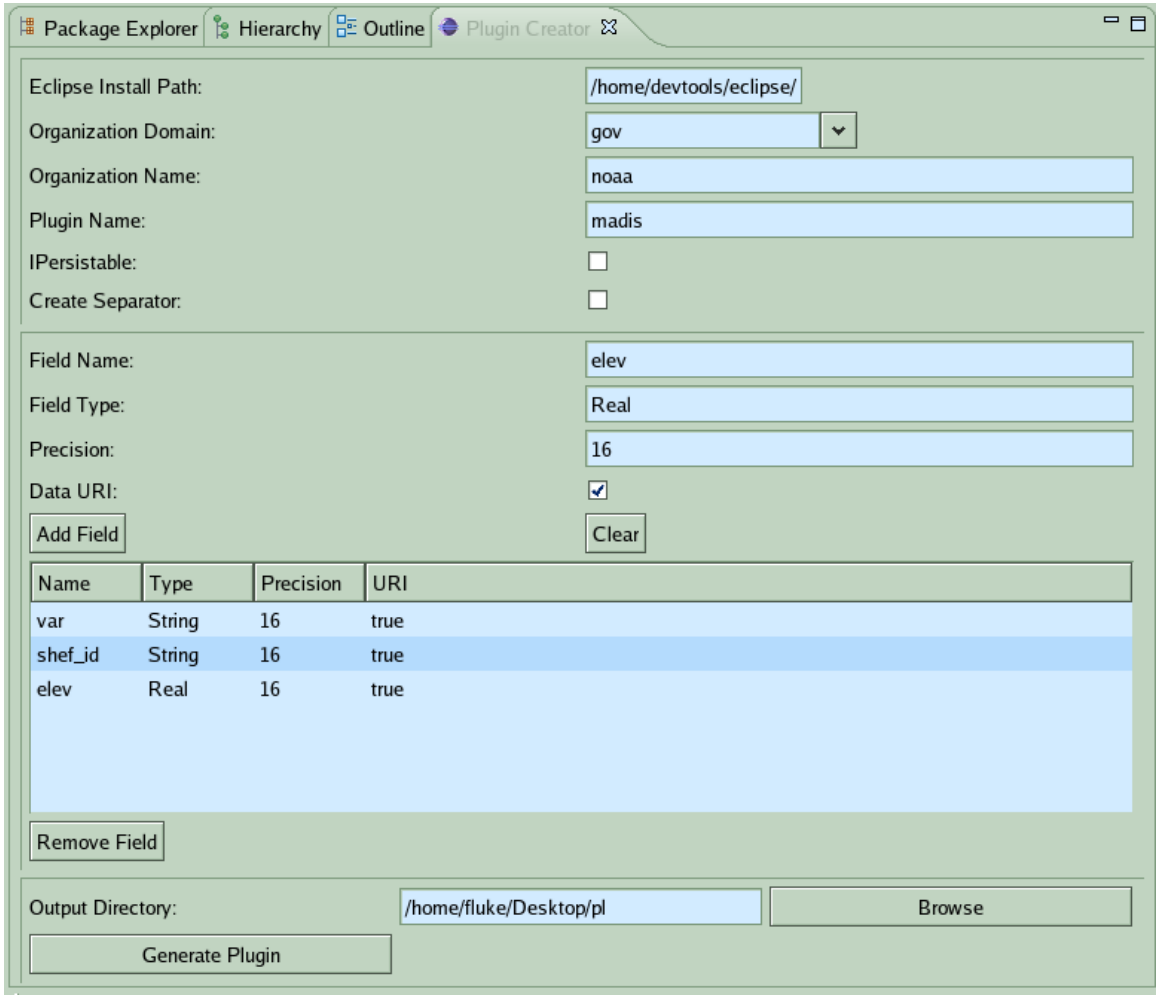


*Figure 2: The EDEX Plug-in Generator with some MADIS fields.*

Figure 3 provides some evidence that the plug-in actually works. It shows the PostgreSQL MADIS data table as displayed by the pgAdmin III tool after the sample data shown in Figure 1 was ingested.

| | oid | datauri varchar | var varchar | shef_id varchar | elev float4 | lat float4 | lon float4 | obtime timestamptz | provider varchar | ata_valu float4 | qcc rch | qca int4 | qc nt | datatime varchar | insert_time timestamptz |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 137987 | /madis/V-T | V-TD | PAMO | 103 | 62.1 | -163.68 | 2008-01-2 | OTHER-M | 260.15 | S | 59 | 0 | 2008-01-2 | 2008-10-0 |
| 2 | 137988 | /madis/V-T | V-TD | PAIL | 49 | 59.75 | -154.92 | 2008-01-2 | OTHER-M | 270.15 | S | 59 | 0 | 2008-01-2 | 2008-10-0 |
| 3 | 137989 | /madis/V-T | V-TD | PAOM | 11 | 64.52 | -165.45 | 2008-01-2 | OTHER-M | 259.15 | S | 59 | 0 | 2008-01-2 | 2008-10-0 |
| 4 | 137990 | /madis/V-T | V-TD | PAHC | 21 | 62.18 | -159.78 | 2008-01-2 | OTHER-M | 261.15 | S | 59 | 0 | 2008-01-2 | 2008-10-0 |
| 5 | 137991 | /madis/V-T | V-TD | PASI | 20 | 57.07 | -135.35 | 2008-01-2 | OTHER-M | 271.15 | S | 59 | 0 | 2008-01-2 | 2008-10-0 |
| * | | | | | | | | | | | | | | | |

*Figure 3: The contents of the MADIS database table as displayed by pgAdmin III*

Figure 4 shows that the AWIPS II Test Driver web application can also be used to display MADIS data records. We did this by entering a customized script into the ASCII Data tab's Request/Response Message box, as shown, and then clicking "Request Product"

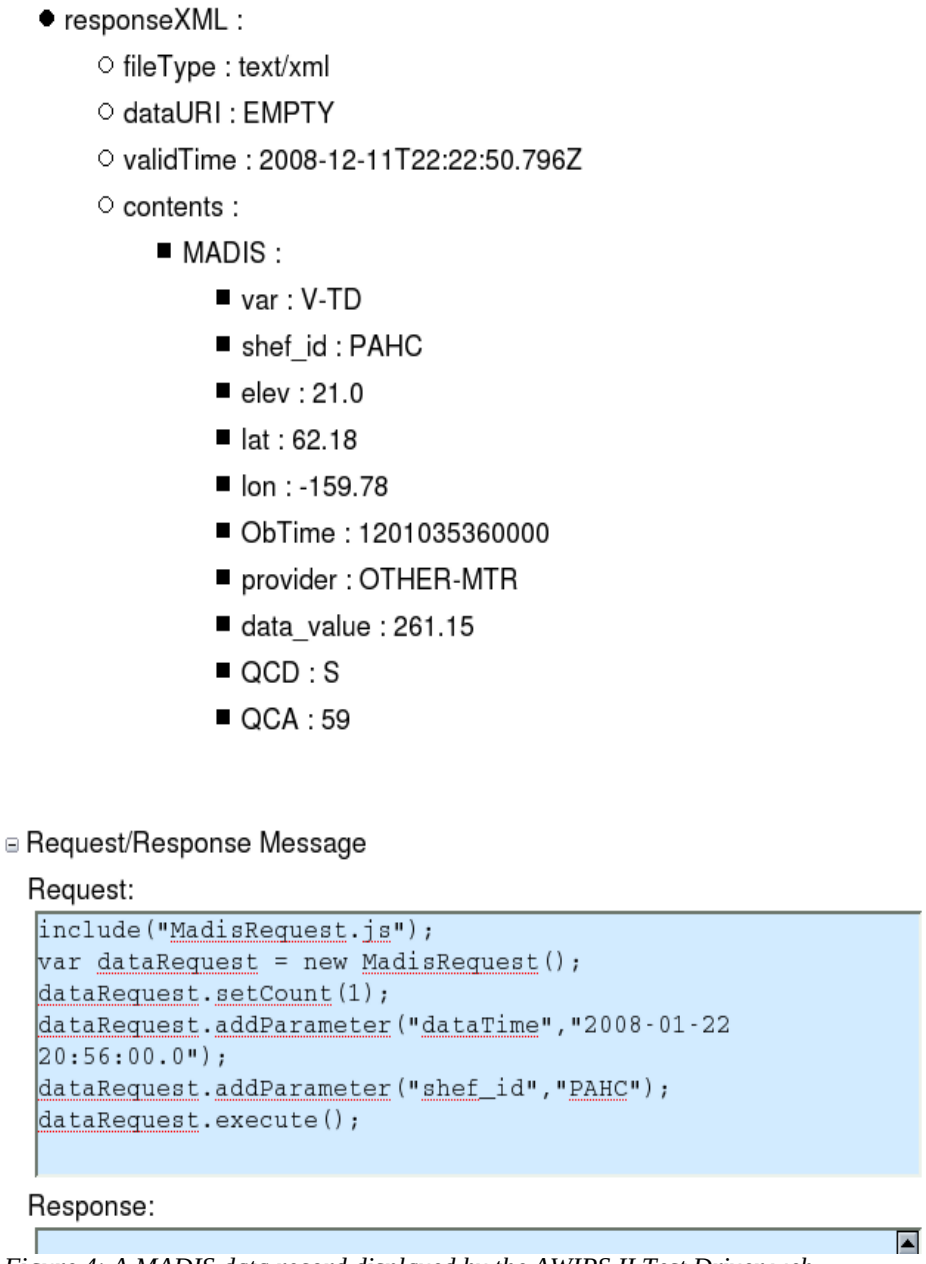


*Figure 4: A MADIS data record displayed by the AWIPS II Test Driver web application.*

## 3. CAVE PLUG-INS

Since CAVE is based on Eclipse, all CAVE plug-ins take on the form of Eclipse plug-ins. In fact, CAVE consists of a set of Eclipse plug-ins. Extending CAVE requires developing Eclipse plug-ins. The only difference is that CAVE plug-ins should usually make use of the CAVE extension points rather than just the Eclipse extensions points. All of our CAVE plug-ins were developed in this way, including the MADIS display plug-in.

The AbstractMapTool, IvisResource and IgraphicsTarget are the basic extensions points and graphics interfaces for display. The menus can be configured in the plugin.xml file of each plug-in.

Figure 5 shows the MADIS menu bar button that we added to allow invocation of the MADIS data display plug-in, and the other CAVE plug-ins developed at GSD. In this case it is being used to invoke the Interactive Draw plug-in.

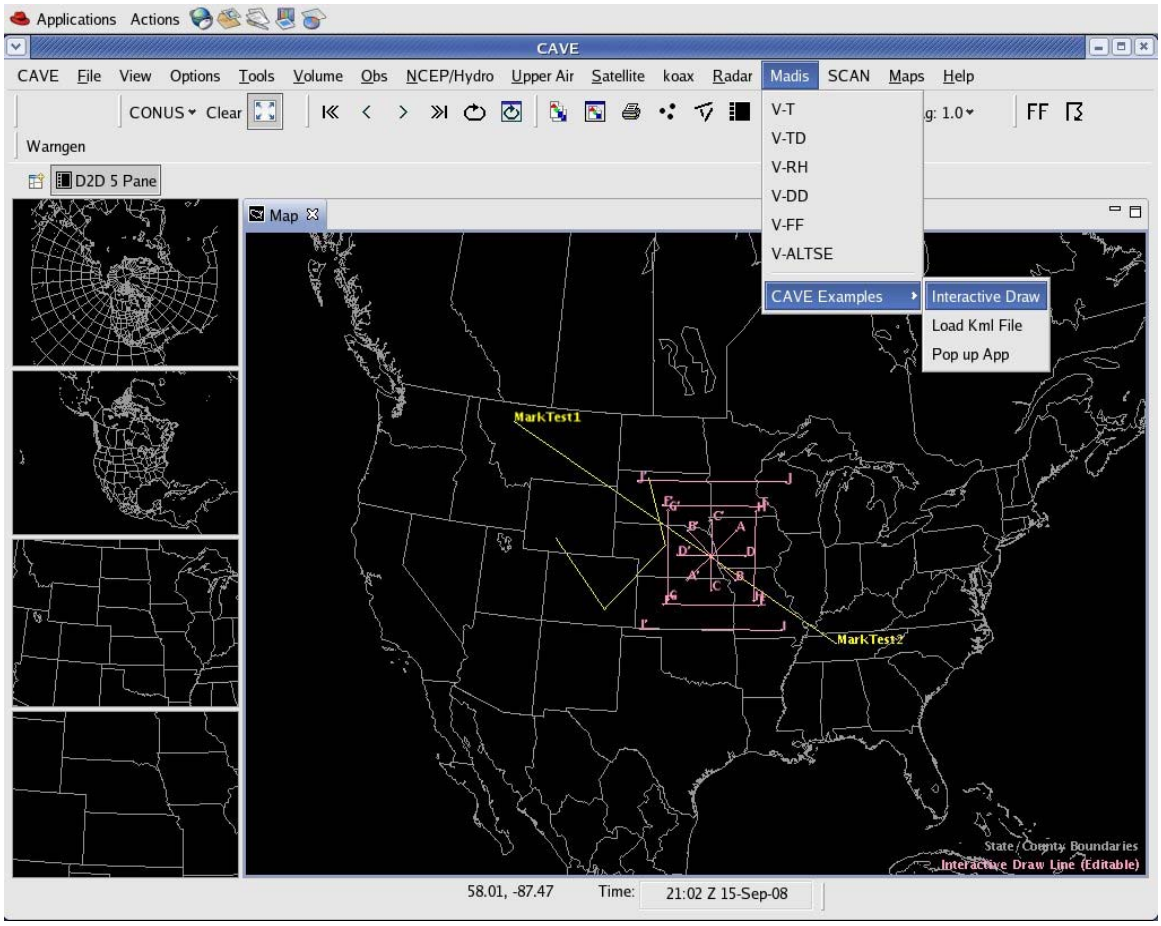


*Figure 5: Invocation of the Interactive Draw plug-in using the added MADIS menu bar button.*

Figure 6 shows a plot of MADIS relative humidity on a D-2D CAVE pane.

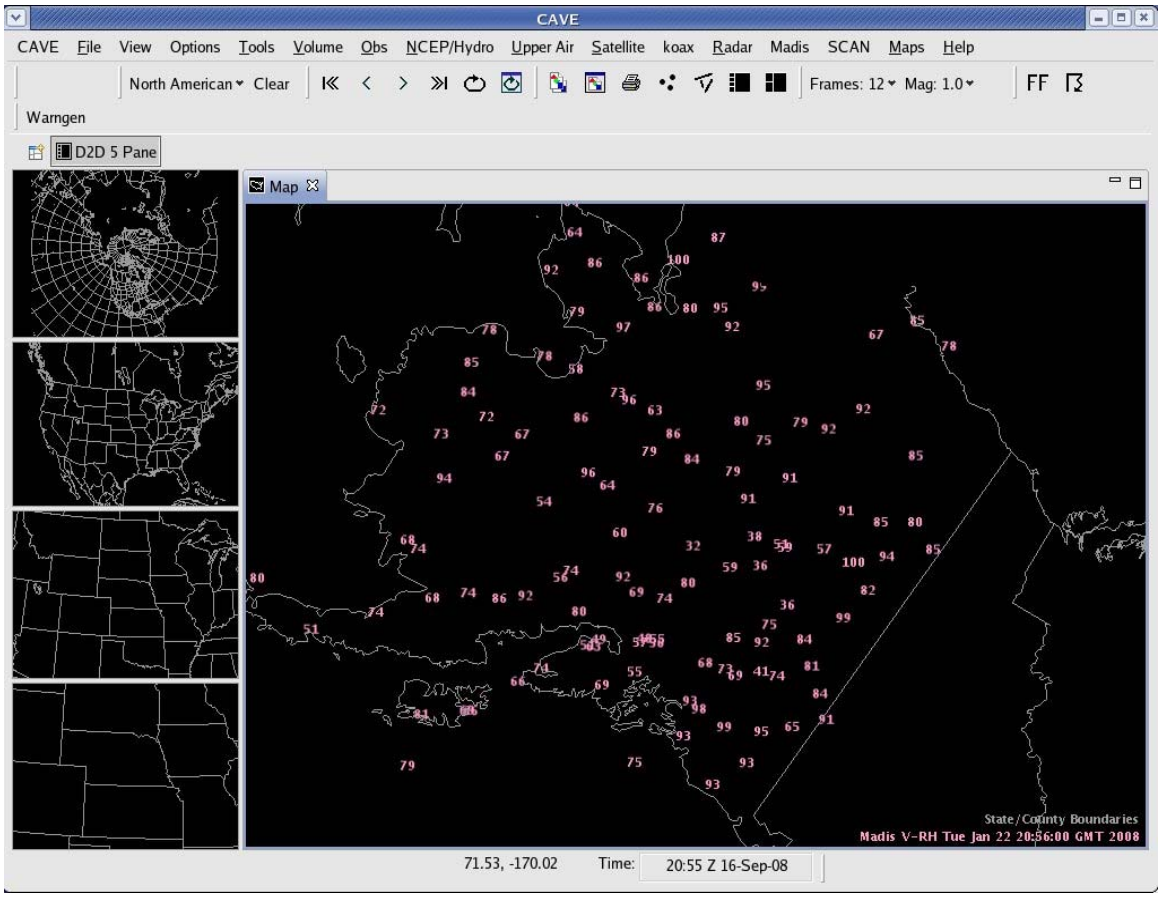


*Figure 6: Plot of MADIS relative humidity (RH) data.*

Note that for the CAVE plug-ins we manually created and populated the needed files. This took significant time and effort. There is an Eclipse plug-in creation tool, but we found that it did not make creating the plug-ins any easier.

Four concerns were discovered while learning how to manually develop CAVE plug-ins:

1) Writing plug-ins requires developing an understanding of the several layers that make up the architecture. In addition to the CAVE GUI layer, there is also the Eclipse Rich Client Platform (RCP) layer, and under that, the Java Standard Widget Toolkit layer.

2) For developers who do not understand AWIPS I, understanding CAVE will be especially difficult. Many complicated and hard-to-maintain components of the AWIPS I Interactive Graphics Capability (IGC) are also in CAVE. This includes depictables and much of the basic IGC architecture.

3) CAVE developers must know about the EDEX plug-in client interface and data formats, and how to communicate with EDEX in order to write a CAVE plug-in. Whereas, EDEX plug-in developers do not need to know anything about writing CAVE plug-ins.

4) For non-Java programmers, it will take about six to 12 months training to become efficient on AWIPS II development.

We believe that a tool to allow CAVE plug-ins to be created more automatically would significantly mitigate these concerns. A CAVE plug-in creator such as the one for EDEX could be developed, and it would make developing CAVE plug-ins much easier.

## 4. CONCLUSION

In addition to the issues we mention above, a major concern for AWIPS II developers currently is the lack of sufficiently in-depth documentation, both external and in-line. Better documentation would also make developing plug-ins much easer. Raytheon has made it clear that more documentation will be written after the basic features have been completed.

Regardless of these issues we have successfully shown that one can extend AWIPS II by writing plug-ins. Our experience has shown that with proper training in the tools, and in-depth study of the source code, one can write almost any kind of plug-in to extend the functionality of AWIPS II. We feel that as AWIPS II continues to evolve, the process of adding new functionality will be smoother.

## 5. ACKNOLGEMNTS

To Ann Reiser for her gracious help in editing the paper; Carl Bullock, Woody Roberts and Joe Wakefield for their helpful critiques of the paper.