7.8 DEVELOPMENT OF A JAVA-BASED METEOROLOGICAL WORKSTATION BY GERMAN METEOROLOGICAL SERVICE AND PARTNERS

Gerhard Eymann[1]
Deutscher Wetterdienst, Offenbach, Germany

1. INTRODUCTION

German Weather Service DWD and German Military Geophysical Service decided two years ago to develop a new meteorological workstation from scratch [Koppert, 2002]. During last year, Meteo-Swiss and Danish Meteorological Institue (DMI) joined the project. The aim is is to replace several older systems at the institutes that have particular tasks and have become difficult to maintain and update. New demands from meteorologists could not be integrated in the legacy systems without very large effort. Required new functionality is e.g. 3-D visualization, advanced analysis capabilities and improved geography.

The decision for Java as programminmg language and development platform was made since it meets essential criteria like independence of platform and operating system. It offers 2-D and 3-D graphics and additional APIs for GUIs, data base access and other functionality. The performance issue was evaluated before in a pilot project and is analysed during all phases of development. It turned out that this is not a problem, however many things must be considered in the design and during coding. Even numerical calculatuions are now in pure Java since we found they can outperform C++ when carefully optimized.

The design follows an object-oriented approach utilizing formal methods, in particular UML. Primary aim is a 3-tier system (client, server, access layer), a web service will be added later. The architecture of the client uses a scheme that allows the independent development and integration of individual application layers, e.g. for observation data, gridded data or geography. This facilitates every participating partner to build individual applications.

2. SYSTEM ARCHITECTURE OVERVIEW

The main principles of the architecture (Fig. 1) are:

- open and scalable multi-tier architecture,
- modularization by seperation in components,
- high degree and universal way of configurability,
- well defined interfaces and shielding of particular access to data.

In addition, we defined essential quality criteria as performance, stability and ease of maintenance. We take particular, time and cost-intensive measures to achieve and control this.

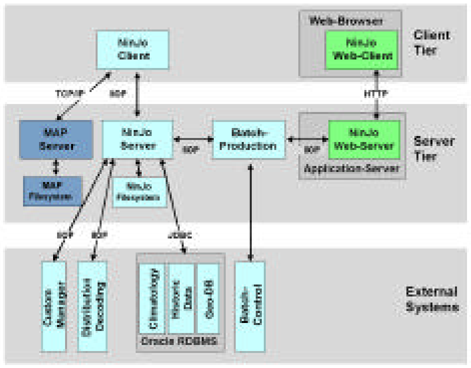

***Fig. 1:** NinJo System Architecture (left: temporarily used MAP system, right: future developments)*

Access to observation data is provided temporarily by the data base of the current MAP system. This will be replaced by the NinJo Server in the beginning of 2003. Client and Server communicate via CORBA (Iona ORBacus). This was preferred over self-developped low-level communication, because the ORB has many built-in services like load-balancing and security mechanisms. The two components at the right, Web-Server and -Client, have been postponed to later project phases.

The Batch Production Component, which is in the conceptual phase, acts virtually as a client without any interactivity and screen rendering capabilitiy. It produces graphical products like charts and diagrams off-line. For the output, a PostScript and FLASH renderer were developped, another option is SVG (Scalable Vector Graphics).

The configuration of each component is described by XML. At the back end, the system has several interfaces to external systems. They comprise of file systems with standard meteorological formats, a RDBMS and a decoding system that provides observation data. The access from the client is encapsuled through an access layer, the application layer does not have to deal with particular formats. Observation and forecast data will be handled as files in standard meteorological formats, the RDBMS is to provide

[1] *Corresponding author address*: Gerhard Eymann, Deutscher Wetterdienst, Kaiserleistr. 44, 63067 Offenbach, Germany; email: Gerhard.Eymann@dwd.de

climatological data. Time-independent bulk data, e.g. geographic vector and raster data, will by default be stored at the client.

3. FRAMEWORK COMPONENTS

In addition to the seperation in tiers, there is a distinction in framework and application-specific components. Framework components exist for the server and client as well, they comprise e.g.:

- configuration framework,
- internationalization,
- data communication,
- data servers (seperate for gridded data, point data, image data).

The functionality of the first component is covered by an own implementation, since we found some deficiencies in other API's like JAXB. The configuration module represents a complete and comprehensive description of the client and server configuration, including batch processing and diagram visualization.

Client framework components provide the necessary services for application layers. The Java built-in feature of internationalization gives an elegant way of separating language specific information (strings, units) from the code. We will support 5 languages from the beginning (english, german, danish, french, italian). Some framework components affect the client only, e.g.:

- layer container and PAC client control,
- graphics visualization library,
- graphics rendering library (GOF),
- raster imaging,
- graphical user interface (GUI).

The first acronym stands for an OO design pattern that we selected and turned out to be very useful: presentation, abstraction, control. This principle is the key to the development of individual application layers that can be attached to the system independently.

For the visualization, a self-developped graphics API called GOF (Graphics Object Factory) abstracts from the underlying Java-2d, Java-3d or the OpenGL binding GL4Java. The GOF uses internally the concept of a scenegraph, as it is known from Java-3D. Preliminary results of GL4Java are promising, concerning functionality, memory ressources and performance. Mathematical functions like isoline and -area calculations are encapsuled in a utility library, written in pure Java.

For the visualization of image or raster data, Sun's Java add-on JAI (Java Advanced Imaging) is used. It offers essential image processing and visualization techniques, including tiled, multi-resolution image handling. Geodetic transfor-mations of image data are done with the aid of a built-in warp function, where a coarse grid of points is transformed exactly and intermediate areas approximated by affine methods. Display of one of the basic geographic raster data sets (GTOPO30 from USGS, horiz. resolution ~1 km) is achieved in real-time on a state-of-the-art PC.

4. APPLICATION LAYERS

Fig. 2 illustrates the main components of the client. It comprises several application layers, which use the framework (GUI, GOF, configuration etc.) simultaneously. Important demands to the client design were scalability and independent development and operation of application layers. Each layer is able to manage and visualize its own particular data, such as:

- point observation data,
- gridded data,
- satellite data,
- radar data,
- trajectories,
- geographical vector and raster data.

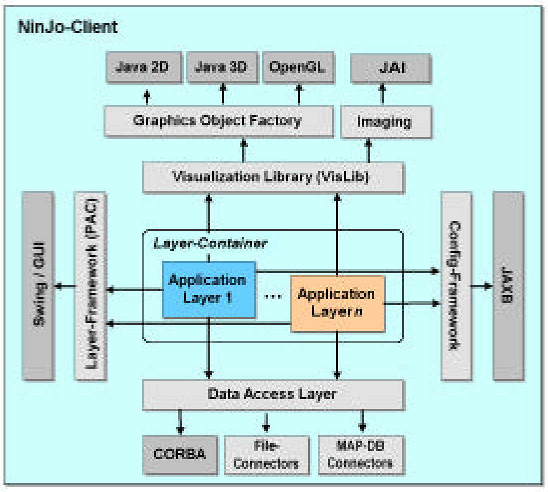

***Fig. 2:** Client Design (with some external components in dark grey)*

The PAC design pattern was chosen to solve this problem. It is an extension of the MVC (Model, View, Control) pattern and allows to have a well-defined separation of essential system components. Each application layer follows the MVC pattern, while PAC is an hierachical collection of the individual MVC's.

The main components are the *model*, that holds data and updates the view, the *view*, that gets data from the model and visualizes them, and *control*, that interacts with the GUI, the configuration, deals with events (e.g. user input, mouse), and has control over the two other components.

The client consists of a tree of MVC-triples, each one being a PAC agent. The agents are grouped in a 3-step hierachy: a single top-level one to control the whole system, a few intermediate ones to control each window, and several bottom-level agents for each application layer. Intermediate agents also control additional windows for diagrams.

5. RESULTS FROM PROTOTYPE

The GUI of the 2nd prototype from summer 2002 is illustrated in Fig. 3. A multi-window layout similar to AWIPS was preferred by the users. The number of secondary scenes can vary from none to three. Main and secondary scene can be interchanged by a mouse-click. Each scene may display 2-D or 3-D data. Diagrams will appear in a separate window.

The GUI layout is not settled yet, but shows some elements we agreed to: on top a general menu, a tool bar and an application specific menu. An interactive legend and controls for time-dependent adjustments are at the bottom. Animation control and display is inherent to the system.

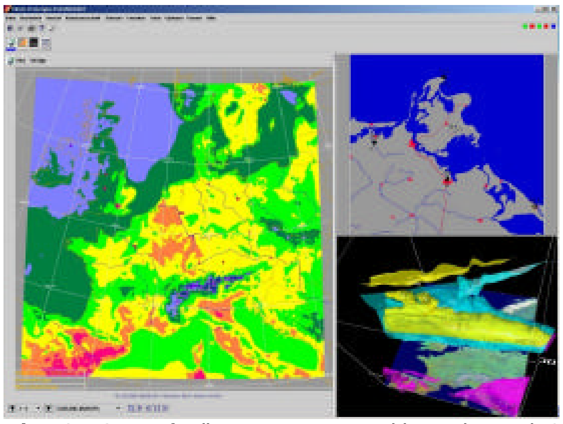

Fig. 3: *GUI of client prototype with main and 2 secondary scenes. Shown is the GeoVector and GRID layer with temperature isoareas, in the lower right 3-d data (digital elevation data with texture, above cloud humidity level)*

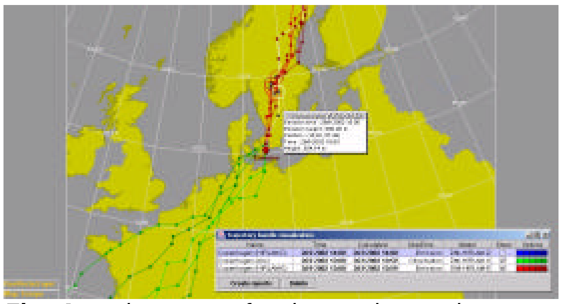

Fig. 4: *main scene of trajectory layer. shown are both forward (red) and backward (green) trajectories. A tool-tip pops up at a selected trajectory, circles indicate a particular time step.*

Fig. 4 is an example of an independent application layer by DMI. While the core layers are developped by the NinJo team, more individual layers will be developped for special purposes by others.

The implementation of the system is done step-wise with the release of two so-called developing protoypes per year. These are examined in detail by an evaluation group of experts and a quality assurance team. Most of the essential client framework components exist already. At the moment, work is going on at the server side and some framework modules like batch production and internationalization. Much work remains to de done for more application layers like satellite and radar data, as well as for 3-D visualization and diagrams.

6. REFERENCES

Koppert, H.-J.: 2002: A Java Based Meteorological Workstation, 18th International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography and Hydrography, American Meteorological Society, p. 307 – 310