Lessons Learned from Developing a Prototype Meteorological Workstation with Java

Hans-Joachim Koppert [1], Helmut Haase[2], Oliver Gehrke[2], and Stefan Lehmann[2]
[1]Deutscher Wetterdienst, Kaiserleistr. 42, 63067 Offenbach, Germany
[2] Fraunhofer IGD, Rundeturmstr. 6, 64283 Darmstadt, Germany

1 Introduction

Meteorological workstation systems focus on the needs of forecasters and scientists. Most of these systems can only be implemented on UNIX platforms. In order to supply lay people with individualized on-demand weather information, software needs to be developed that can easily run on any standard PC and that uses the Web to access weather data. Therefore, DWD and Fraunhofer IGD started to evaluate a solution that ensures high portability, high interactivity, ease of maintenance and the possibility to supply data over the Internet. These efforts resulted in a prototype meteorological workstation written in Java. This prototype runs on every platform for which a Java Virtual Machine (JVM) is installed and that has access to the Internet. In addition, this software serves as a prototype to evaluate the usefulness of Java for DWD's next generation, meteorological workstation system.

2 Java

Java was chosen as the computing platform, since it promised to meet our requirements. Java is an interpreted language. The compiler produces byte code, that can be interpreted by any JVM. It fulfills the promise: "Compile once, run everywhere". The applet model allows the application to always deploy the latest software release, which helps to maintain the software on the client side. Java provides a large set of classes for web support and networking. It is the programming language of the Internet. The performance has been greatly improved by new compiler technologies and is considered to be sufficient for current, meteorological 2D-applications. Java is a robust programming environment due to its architecture, which abandons the need for pointers, multiple inheritance and explicit memory management. Java provides an extensive set of built-in libraries including defined data structures and methods supporting faster implementation time.

3 JavaMAP

The name of the project is JavaMAP. Figure 1 shows the architecture of JavaMAP. The client includes a GUI and a rendering module. It communicates with the JavaMAP-Server via the Web. The server has access to the MAP (DWD's Unix based workstation system) observational database and to databases providing satellite and radar imagery. The current version of the server applies pre-processing to the data.
JavaMAP is based on Java2 (i.e. JDK >= 1.2.1). Java2 is a stable Java version with many new features that is available for most platforms. JavaMAP makes extensive use of the Java Foundation Classes Swing and Java2D.
Java2D is output device independent. It comprises convenient handling of 2D graphical objects and offers an extensive set of features. Java2D implements additional geometric forms (such as 2D-paths), affine transformations and new context attributes (such as gradient paint).
Swing is a powerful toolkit with modifiable 'look and feel' functionality and numerous features. No restrictions are imposed on the developer, since it does not use any OS specific libraries (peerless).

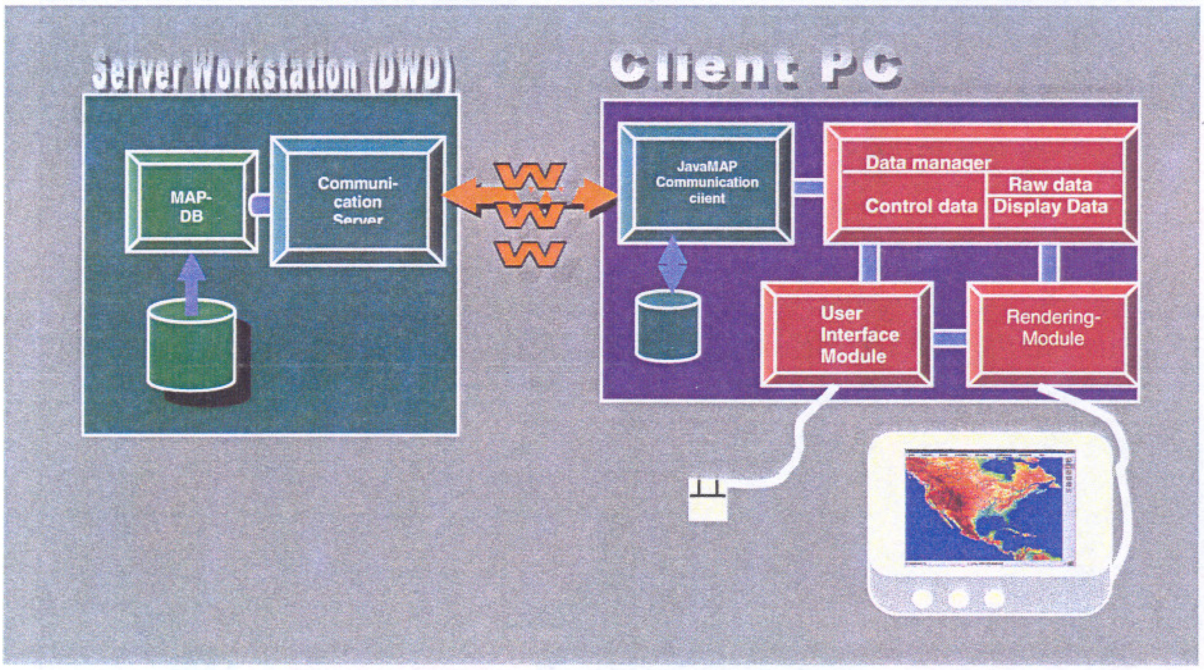

Figure 1: The architecture of JavaMAP

3.1 The Features of JavaMAP

3.1.1 Context Information

JavaMAP uses background maps based on the gtopo30 dataset from USGS. The elevation data is composed of tiles with adaptive resolution. The maximum spatial resolution is approximately 1km. The vertical resolution is 8 bit. Vector-data in 3 different resolutions is used for rivers, boundaries and coastlines.

3.1.2. Observational data

Currently, a subset of the WMO-FM12 data including station related Kalman-filtered forecasts are offered to the user. He can freely configure and display any combination of these parameters. Each parameter has its own associated colour table (colour by value). The system allows the developer to mask the background in order to ensure the readability of all values and symbols.

3.1.2 Imagery

Satellite (world (composite) and European data, infrared and vis channels) and radar imagery (German composite) are available in both polarstereographic and regular latitude/longitude projection. Currently only jpeg encoded data is supported, although GIF can be used, if full resolution is required. Any other data source can be superimposed .

3.1.3 Functionalities

The system has an intuitive user interface. It offers browser-oriented navigation features with "back" and "forward" buttons as well as the option to save settings. Intelligent zooming (additional stations appear if the necessary plotting area is available), panning, and picking (information about stations and present weather) are implemented. Combinations of the available datasets are possible. Figure 2 shows the user interface and a depiction of surface weather parameters.

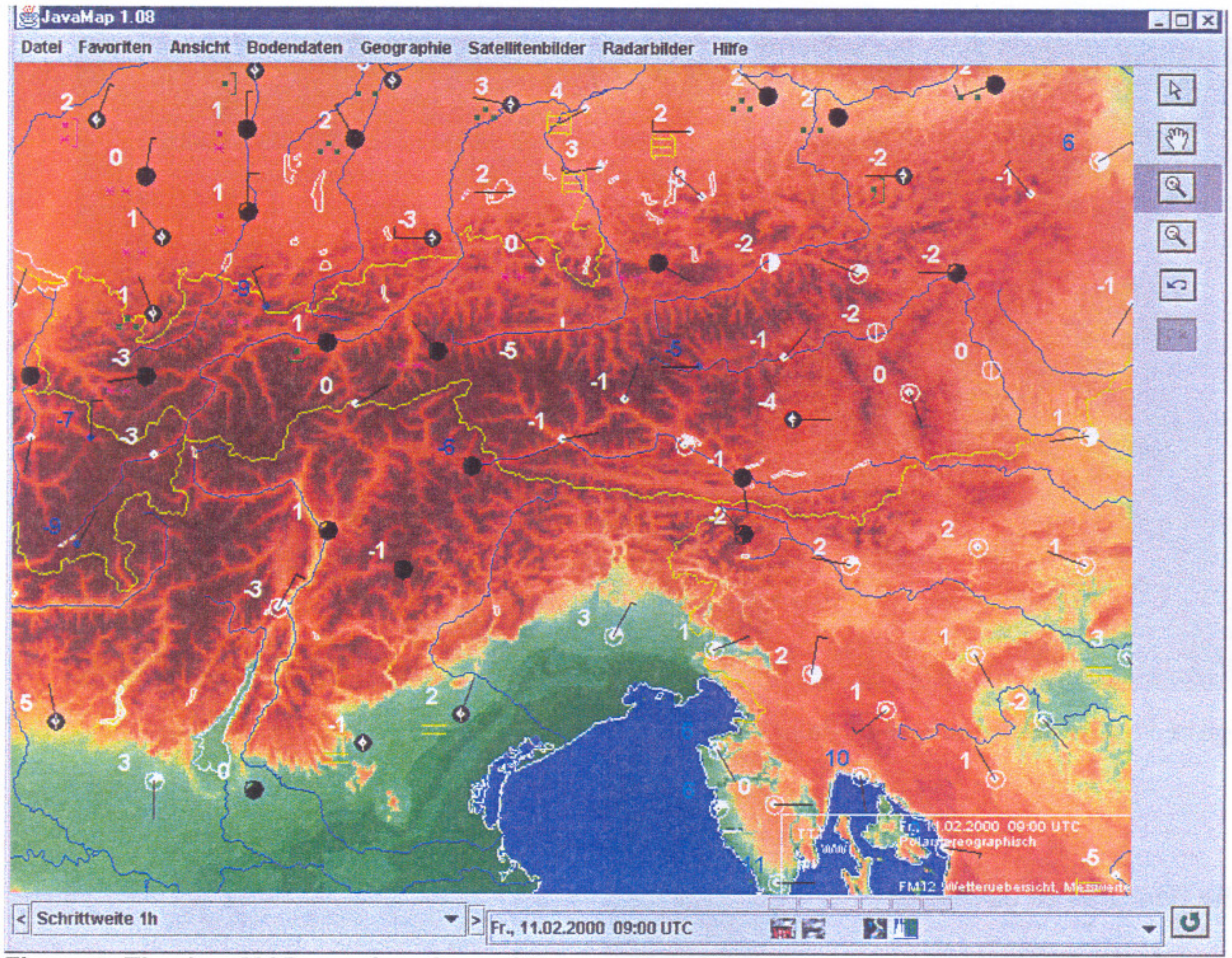

Figure 2: The JavaMAP user interface with present weather on a map of the eastern part of the alps

4. Lessons learned

4.1 Java in general

The Java programming environment allows efficient application development. Its architecture minimizes the risk of producing buggy software. But Java itself is still not mature, which is apparent from the numerous bugs found in some of the APIs. It is also getting extensive. No individual developer is able to master all the APIs and packages. Furthermore, Java is still evolving rapidly, requiring developers to keep an eye on current developments.

4.2 Swing

Swing is an easy-to-use API, providing a variety of features to build powerful GUIs. However, it requires a lot of memory for several reasons. The components of Swing are double buffered and have a more complex hierarchy then those of the AWT, the predecessor of Swing. Graphic routines are less optimized and less efficient in the run-time environment. However, the latest JDK release JDK1.3rc1 demonstrates that SUN has made much progress towards reducing Swing's footprint and increasing its performance.

4.3 Print API

Neither JDK1.1 nor JDK1.2 offer satisfying support for printing. JDK1.1 uses a fixed resolution of 72 dpi whereas JDK1.2 offers basic support for higher resolutions. The appropriate workaround is to print outside of Java, after having saved the rendering results as bitmaps, for example. However, only a few image formats are supported (jdk1.3: jpg, gif, png). An alternative is to generate Adobe PDF or postscript files. There are freeware libraries (www.retep.org.uk/pdf, currently no support for images) available for generating PDF-output.

3

4.4 Graphics

Before Java 2 had been released, the AWT used to be the only graphics API. Its functionality is too limited. Only line widths of one pixel are supported, for example. But the AWT and especially its image methods are quite fast. In contrast, Java2D has a large feature set at the expense of speed and memory (e.g. internal storage of images as RGBA). Specialized classes like affine transformations, colour spaces, and new graphical display options allow for the rendering of complex pictures of high quality. But there are also shortcomings. The security model prohibits the use of inheritance for creating colour table classes, for example.

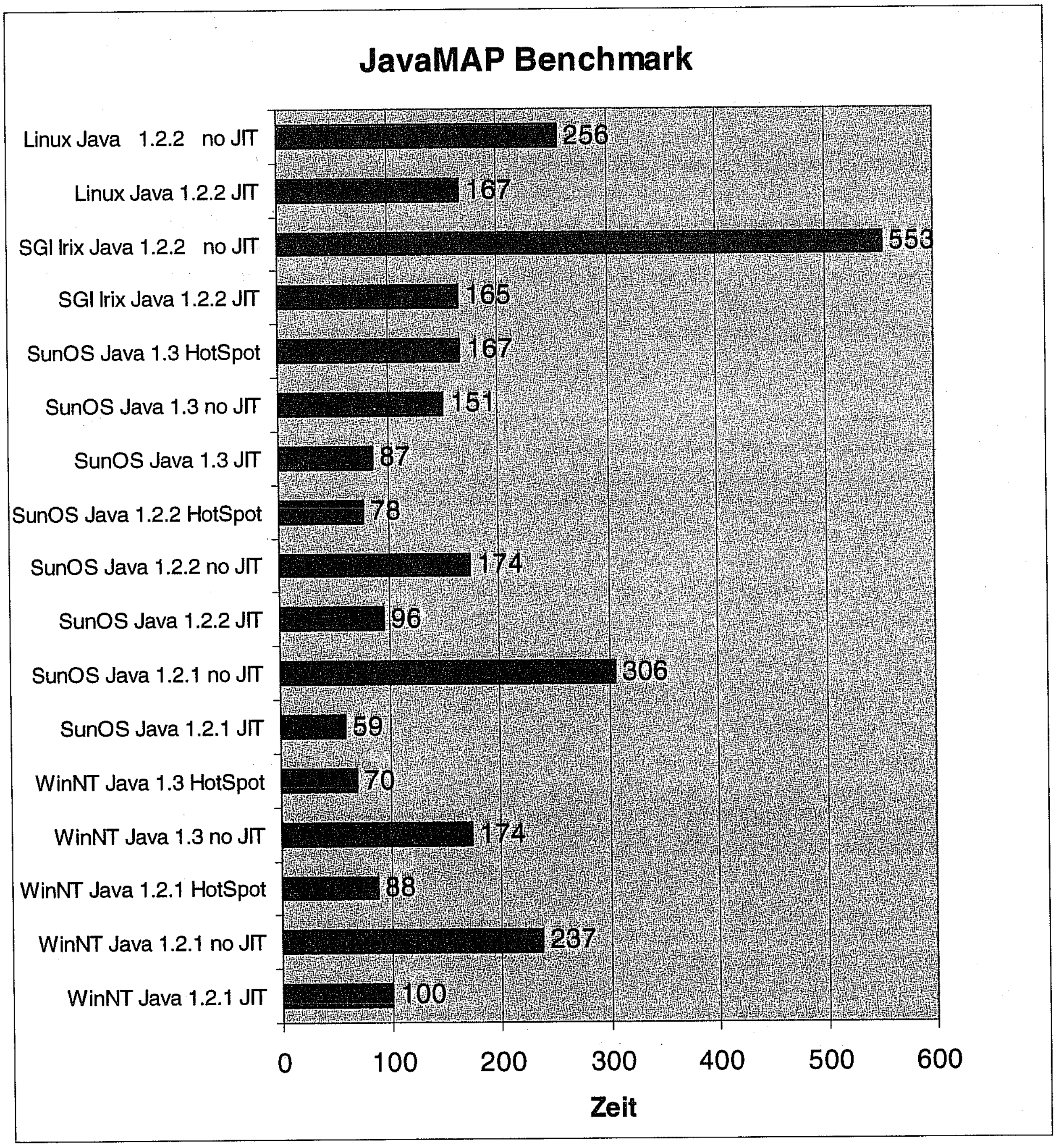

Figure 3: benchmarks of JavaMAP (Values in per cent against Windows NT with JIT)

4.5 Performance

Discussions about Java always address performance issues. In order to measure real world results, we simulated a typical forecasting session with JavaMAP. Operations like loading data, displaying data, zooming and panning were performed for testing purposes. Using JIT or HotSpot, the performance results fell in the range between 'good' and 'excellent' on all platforms. The results are shown in figure 3. The tests were performed on four different operating systems and hardware platforms:

- Windows NT 4.0;	Pentium II, 350 MHz, 128MB;	SUN JDK 1.2.1 - 1.3rc1
- Linux (RedHat 6.1);	Pentium II, 450 MHz, 256 MB;	blackdown JDK 1.2.2rc4
- SUN OS 5.7;	Ultra-5_10, 333 MHz, 128MB;	SUN JDK 1.2.1 - 1.3
- SGI IRIX 6.5;	Octane, 250 MHz R10000, 1 GB;	SGI JDK 1.2.2

Although results that were obtained on different platforms cannot be directly compared with each other, they give an estimate of the JDK performance.
The results show that SUN has achieved substantial progress by enhancing their compiler technology. In particular, the results on Windows NT show that performance has steadily improved. The test that was run using JDK 1.3 in combination with the HotSpot client VM took only 30% of the time that the JDK 1.2.1 VM needed without any compiler. Furthermore, the results show that there are significant differences between the individual JDK-implementations. While the performance on Solaris and Windows NT is about the same, the SGI IRIX VM is quite slow as compared to SUN's Solaris version.

4.6 JavaMAP

Our beta-test team is quite satisfied with the performance and the functionality of JavaMAP. The aesthetically appealing geographical background, the easy data access over the Internet from your office or from your home, the high interactivity and the intuitive user interface are well received. JavaMAP runs on most operating systems with little or no alterations. A local database on the client side contains contextual information. Only the time-dependent meteorological datasets have to be downloaded. This results in smaller amounts of data that is transmitted, as compared to models using images. JavaMAP is easily extensible and allows for the configuration of data and graphical attributes.

5. Conclusion

Java is the sound choice for a computing platform for the JavaMAP application, which, at this point, has limited functionality and performance requirements. Java is quite easy to learn, at least when coaching by experienced Java programmers is available. The main problem for the transition from Fortran to Java is to learn the object-oriented paradigm. This has to be planned very carefully and takes a considerable amount of time.
Java proved to be a platform that supports high productivity, due to large class libraries (which might be confusing for the beginner). Our experiences with this prototype encourage us to evaluate Java more closely, as a client or even server-side computing platform for DWD's next generation meteorological workstation. Java is highly portable. This portability will help in the development of software that does not depend on any special hardware or operating system. Java's component technology will make the implementation of modular software components rather easy, which can be recycled to build new applications.

6. Outlook

Since JavaMAP yielded great success at DWD, it was decided to extend its functionality. The upcoming release will include the following features:

- interactive meteogram
- animations of all parameters
- colour look-up tables for images

- forecasts of a limited set of parameters (as images)
- adjustment of the graphical display according to the needs of lay people

The communication model will be restructured from the bottom up. A well-designed communication server will supply authorized clients with data from the MAP-DB. A separate DSP interface will establish the connection between the server and the MAP-DB . The main functionality of the server includes

- user authentication
- transmission of both raster and vector data
- listing available data
- rudimentary caching
- providing information about the network load
- updating software modules, if necessary.

An optimized socket architecture guarantees high performance and reliability standards. Multithreading capabilities enable the server to handle multiple clients in parallel. The server will be able to retrieve user specific information from a separate user database via SQL. Finally, a well-designed security concept based on public/private key encryption, digital signatures and session tokens will be integrated into the system.

7. Acknowledgments

We thank Andreas Kahl (DWD) for running the JavaMAP-Benchmarks on Windows NT, Daniel Matter (SMA, Switzerland) for running the benchmarks on SUN OS, and G. Eymann and H. Kiehl (DWD) for running the benchmarks on LINUX.