What Would a Reusable Meteorology Component for Environmental Models Look Like

C. R. Maul
Institute of Sustainable Irrigated Agriculture, (ISIA), Tatura VIC 3616, Australia
Christian.Maul@nri.vic.gov.au

Key words: Software Components, UML, Java

Abstract: The architecture of a reusable software component is introduced that provides meteorology information. It will provide temperature, radiation, precipitation, evaporation, humidity, wind speed and hydrological events for environmental software systems. The component will be implemented as a JavaBean with XML (Extendible Markup Language) documentation. It will also be able to process forecast data and to disaggregate these data into discrete single weather events. The structure and the behaviour of the component are displayed using 'Unified Modelling Language' (UML) [2] with class and sequence diagrams. Some advantages of the approach are platform independence, robustness, small size, user friendliness and the availability of a large range of past and future meteorology data.

1. INTRODUCTION

Meteorological data are in many cases and for very obvious reasons the driving variables for environmental models. From a programming and scientific point of view producing weather data and reading meteorological files is neither creative nor scientifically challenging. Therefore many programs leave it to the user to enter the input or to produce a particular file format that the program can read. However, as meteorological data drive the programs, it is advisable to put some effort into these program parts, because on one hand the output is determined by the quality of the input, on the other hand, user friendliness determines how widely models or decision support systems are used.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35503-0_29

R. Denzer et al. (eds.), *Environmental Software Systems*

2. ANALYSIS AND DESIGN

2.1 Functional requirements

Meteorological data differ in their time horizon. They can be historical or current values or forecasts. They might originate from a data file or a measuring device. The program might need these values in a particular resolution as weekly, daily, hourly or quarter hourly data. The component must have flexible approaches to:

- input sources,
- time resolution of the data, and
- file formats.

Various organisations are concerned with the collection of meteorological data, weather analysis and forecasts. In Australia the Bureau of Meteorology is the major source of meteorological data. It seems appropriate to rely on the expertise, data and file formats of these organisations. The data that may be required are:

- air temperatures,
- terrestrial temperatures,
- soil temperatures,
- humidity,
- wind speed,
- cloud cover,
- evaporation,
- hydrological events (rain, flood, etc.),
- radiation and atmospheric transmissivity, and
- day length and latitude.

Availability of data will depend very much on the particular weather station so the component must cope with different combinations and subsets of these data. It would be useful to check the reliability of data as well because measuring devices are exposed to weather, tend not to be very well maintained and faults seem to be only detected when data become obviously wrong. This issue will be addressed in a later version.

2.2 Definition of key mechanisms

Parts of the component should be easily extensible without compromising robustness of the component. This requires object-oriented analysis and design. UML and Together as CASE-tool were chosen as design tools because they are readily available and because they have proven

to be the most versatile and advanced tools. Easy and widespread applicability requires good documentation of the Bean. HTML/XML is standard for JavaBeans. The Javadoc conventions for documenting the application program interface (API) will be used. The different requirements of software systems make it useful to choose a format that is platform-independent and that can be used in different development environments. JavaBeans provide such a format.

The component is planned as a lightweight component. So, it must not contain data that would impede or bloat the programs in which it is finally used. The maximum size of the component should be less than 1 MB. Data should not be part of the component and should reside on a remote server and be transferred by ftp.

The Internet is and will be unreliable. In the event that data connections cannot be maintained, the component must provide sufficient error handling to prevent program crashes. Because of the requirements for object-oriented design, functionality for Internet use, integrated documentation and good for error handling capabilities, Java has been chosen as the programming language.

2.3 Class Definition

The five object classes for the meteorology model are: AirProperties, Climate, SoilProperties, WeatherEvents, C2W.

AirProperties holds air temperatures and calculates the required resolution of temperatures. It reads a meteorology file and provides variables that can be used in a program. It inherits WeatherEvents that contains information about everything that is not temperature or radiation. Climate inherits AirProperties and, implicitly, WeatherEvents and calculates radiation values. Climate is associated with C2W that disaggregates forecasts from average values. SoilProperties is dependent on an instance of a Climate object to be instantiated. The packages and the note indicate which Java packages must be imported to allow a query of a remote database and reading of remote data.

The different data objects were separated to provide a flexible structure for users. If only information about rain, sunshine hours or evaporation is required, an instance of 'WeatherEvents' can be created; if in addition, temperature data are needed, an instance of 'AirProperties' can be created that automatically accesses weather events. Suppose radiation data are required: an instance of 'Climate' is created that inherits all the attributes and methods of 'AirProperties' and therefore has access to 'WeatherEvents'. Different data objects will be created depending on the requirements of the system using the component. The separation has been done to provide

necessary functions with minimum requirements for memory. 'AirProperties' is not only a data container but will also calculate degree-day sums and hourly day and night temperatures. Calculation methods will be derived from Goudriaan and van Laar (1994).

'Climate' will be able to calculate radiation if data are not provided using longitude, day length and atmospheric transmissivity. It provides the method to calculate radiation data using formulae and data of Wagenmakers (1995) and Goudriaan and van Laar (1994). However, a detailed meteorology file is required, and measured values are always better than calculated values

The 'C2W' object generates values from forecasts. To incorporate forecasts the Institute of Climate Research in Potsdam has provided support. Fortunately it is possible to use their climate-to-weather disaggregator (Bürger 1997). This software package is required because forecasts are formulated as general tendencies that have to be translated into discrete weather events. The beauty of this particular weather generator is that it is trainable for particular regions. That means it generates future weather-events according to the variation in historical data. 'C2W' inherits from 'AirProperties' and receives important parts of its functionality such as reading meteorology files, calculating degree-days and variables.

In the design of the component different ways to use the module have been incorporated which will hopefully serve various purposes while integrating the component into environmental software packages.

2.4 Behaviour

If historic weather data files are provided, the behaviour of the module is relatively straightforward. The following sequence diagram using UML shows the processing logic. The meteorology file is read, and the 'AirProperty' and 'WeatherEvents' objects are instantiated. The attributes of both objects are filled with values that are passed on to the growth module. It is desirable that the component enables different time horizons for the calculation of values, although for most purposes hourly and daily values appear to be sufficient. In both diagrams (Figure 1 and 2) the object growth represents any possible object that may require temperature, radiation, precipitation, humidity or wind speed data.

The behaviour involves more steps and additional objects if forecasts are used (Figure 2). As the Australian meteorology database METACCESS has a volume of roughly 200MB, it is planned to query the database from a remote client. For these purposes other components or enterprise JavaBeans and Java Database Connectivity (JDBC) could be used that provide functions for the database access and query.

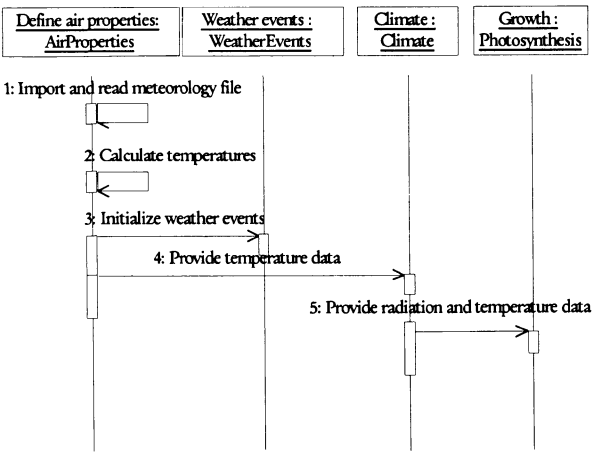

Figure 1. Sequence Diagram for Historic Weather Data

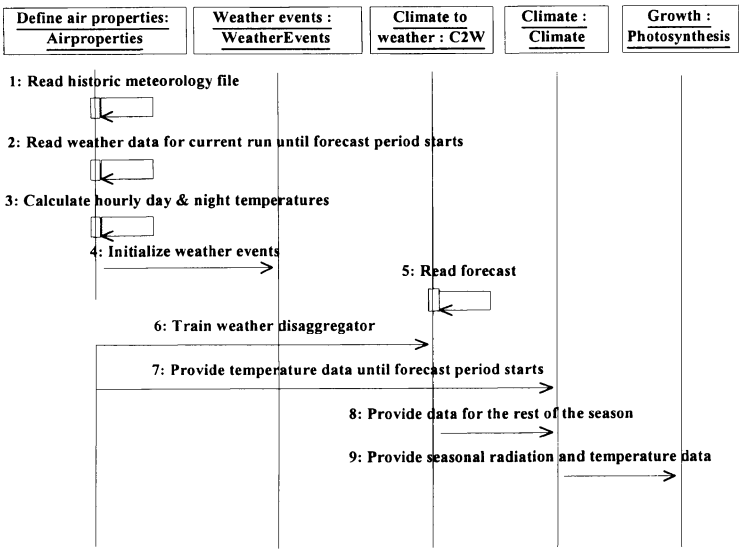

Figure 2. Sequence Diagram for providing Weather Data using forecasts

3. DISCUSSION

This paper has introduced the design of a meteorology component that could speed up software projects. Actual saving of project resources depends on the flexibility and user friendliness of the component. User friendliness means that the component is easily understood and integrated into other programs. JavaBeans currently provide the best available standard to integrate the bean into an IDE and to make documentation available to the programmer. The presentation of the data to the end user - if it is required at all - is the domain of the program in which the component finally resides. As there are many table-oriented graphical components available, a particular graphical user interface is unnecessary. The component is lean because of its partition into different objects and the use of a remote database. If a meteorological database covers historic data and all the weather stations in an area, it very quickly becomes big and awkward. To prevent size deterring the utilisation of the component, data and computing are separated. Apart from that, the memory required at runtime is minimised by only creating instances of particular objects when they are needed.

Access to a remote server will cause problems, however, because of the unreliability of the Internet. The advantages of a small component outweigh the problems of net reliability. The potential exceptions must be caught and dealt with by careful error handling. Java has in-built language constructs that can do this. However, detailed error handling must be implemented, and a good deal of the quality of the component will depend on it. The technological churn, with its newly emerging and disappearing operating systems and versions, will also cause problems. The use of Java as a programming language avoids adapting to operation system version changes and covers various platforms enabling widespread use. Java itself with Version 2 has now reached maturity and a language standard where fundamental changes are unlikely. The resource savings can only be realised if the component displays some robustness. The application of object oriented design I have used provides a clear structure that allows easier alterations and further development. I believe these targets can be met due to the novel architecture of the component.

This paper is also intended to compel other scientists to join in with knowledge, programming resources, design proposals and requirements, or simply with the desire to use such a component in future.

Processing weather data belongs to the repetitive tasks of environmental modelling and as mentioned in the introduction, it is neither challenging nor creative. The programmer leaves it to the end user to provide meteorological data, decreasing the user friendliness of the final product. A reusable

meteorology component could add significant value to models without consuming large project resources.

4. REFERENCES

BÜRGER, G (1997): On the disaggregation of climatological means and anomalies. Climate Research, 8:183-194.

GOUDRIAAN, I; van LAAR, HH (1994): Modelling Potential Crop Growth Processes. Kluwer Academic Publishers, Dordrecht, NL.

WAGENMAKERS, PS (1995): Light relations in orchard systems. PhD Thesis. Agricultural University Wageningen, NL.