

An Evolving Multi Agent System for Meteorological Alerts

Sandy Dance and Malcolm Gorman
Bureau of Meteorology
Melbourne, AUSTRALIA
{s.dance,m.gorman}@bom.gov.au

Lin Padgham and Michael Winikoff
RMIT University
Melbourne, AUSTRALIA
{linpa,winikoff}@cs.rmit.edu.au

ABSTRACT

The Australian Bureau of Meteorology has a requirement for complex and evolving systems to manage its weather forecasting, monitoring and alerts. This paper describes an agent-based system that monitors in real time the current terminal area forecasts (forecasts for areas around airports) and alerts forecasters to inconsistencies between these and observations obtained from automatic weather station data.

Categories and Subject Descriptors

H.4 [Information Systems Applications]: Miscellaneous; D.3.2 [Language Classifications]: Java

General Terms

Design, Algorithms, Reliability

Keywords

Agents, Meteorology, Alerts, BDI, XML

1. INTRODUCTION

The Australian Bureau of Meteorology¹ is the national weather service of Australia. The Forecast Streamlining and Enhancement Project (FSEP) within the Bureau is a major project which seeks to improve the quality, quantity, consistency and timeliness of weather products. Intelligent alerting within the forecast system has a high priority in FSEP. It must be able to evolve, be distributed and open, handle complex data sets, and involve a range of goals.

2. SYSTEM ARCHITECTURE

The architecture of the pilot system contains a number of specially developed agents, a number of existing components, including the real-time data input system, and the data representation and management layer which is crucial to the overall architecture.

These components can be run on different machines across the network. In the pilot we have successfully run the system with components running on an operational server with real-time data communications, a test system on a development machine, and agent driven graphical user interfaces (GUIs) running on forecaster workstations. See Fig 1.

¹<http://www.bom.gov.au>

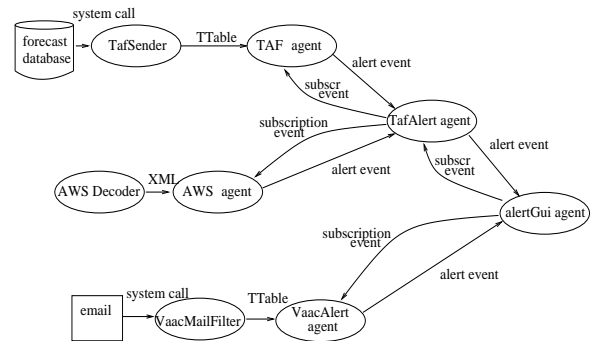


Figure 1: Diagram showing broad data-flow within the alerts system.

These components communicate using TCP/IP or JACK Intelligent Agents^{TM2} messages, and send objects encoded using tree-table-xml (see Section 2.2) contained in JACK messages.

The main component of the system is an agent system that contains a *DataStreamDispatcher* agent and some number of *TAFMonitor* agents.

The *DataStreamDispatcher* agent is responsible for managing subscriptions and for routing messages. A *TAFMonitor* agent will subscribe to TAF and AWS messages and will generate alerts that it sends to the *DataStreamDispatcher*, which are then routed to the appropriate subscribers (eg, a GUI agent).

2.1 Patterns

The agent network in the Intelligent Alerts system is connected using the publish-subscribe design pattern (see Ch5 in [2]). A server agent behaves as a publisher and advertises its service to other agents. Other agents may subscribe to the service and receive notification of events published by the server agent.

The benefits of the Publish-Subscribe pattern are:-

- Agents can be varied and reused independently without interfering with their respective subscribers or publishers.
- State changes in one agent can trigger state changes in other agents without knowing how many agents need to be changed.
- Agents can notify other agents without knowing about the other agents, and avoid tight coupling.

2.2 TTables

A generalized XML format known as TTable [3] is in use. The data is contained in a table called `data` and the corresponding

²JACK Intelligent Agents is a product of Agent Oriented Software, see <http://www.agent-software.com>.

metadata is contained in a related table, for instance:

data			
station_name	wind_speed	wind_direction	air_pressure
Melbourne	13.0	128	1001.0
Mildura	7.0	172	998.0
Avalon	20.0	117	1001.0

metadata			
element_name	unit	data_type	significant_digits
station_name	-	string	0
wind_speed	knots	double	3
wind_direction	degrees	int	3
air_pressure	hectopascals	double	4

Each column in the `data` table has a corresponding row in the `metadata` table. Event messages contain a set of `TTable` objects that are passed from agent to agent in the agent network, providing information to guide agent behaviour.

3. AGENT DEVELOPMENT TOOLKIT

The agent development toolkit which we used in this project was JACK Intelligent Agents. JACK is a third generation agent system based around the concepts of Belief, Desire, and Intention (BDI) [4]. JACK is built on top of Java and includes:

- An agent-oriented programming language that extends Java with agent concepts
- Infrastructure for running distributed agent systems and for communication between agents
- Support for *teams* of agents (not used in this project)
- An integrated development environment incorporating drag-and-drop construction of agents from capabilities and plans (however, plan bodies are textual)
- A design tool for visualizing the structure of an agent system

4. DEPLOYMENT ISSUES

The alert system has had its first exposure to aviation forecasters. This provided valuable feedback on a number of issues, mostly around GUI look and feel, which we will address in the near future.

There were a number of deployment issues. These issues were: flexibility, self-healing from system failure, and system evolvability.

4.1 Flexibility

To maximize flexibility, we have implemented the publish-subscribe pattern as noted above: when an agent subscribes to a service, it is granted a lease for a certain period. It then must resubscribe before that period has expired to continue getting the service. In this way, if a service is added or replaced by another, clients are able to seamlessly reconnect to the new service (if it has the same name). See chapter 12 of [1] for more on leasing.

4.2 Self-healing from system failure

All distributed systems are vulnerable to failures in software, machines and networks, any one of which may potentially bring down the system. Manual intervention to fix failures is unrealistic and self-healing is necessary. In the publish-subscribe pattern, the server checks whether clients still have a valid lease before providing the service, and if not discards that clients subscription. On the client side, if a server fails the client will attempt to resubscribe until the service is again available. In this way the entire system self heals without immediate human intervention.

4.3 System Evolvability

Software upgrades, updates and withdrawal of agents would also lead to system failure if this were not managed.

- the use of JACK facilitates easy implementation of new agent behaviour. For instance, a new subsystem which alerts on volcanic ash detections was implemented in less than two days.
- leasing allows developers to withdraw and replace a component safely.
- overriding the Java serialVersionID on transmitted classes (to remove dependency on particular compilations of classes at either end of a message transmission via serialization) allows components commonly transmitted between machines to be extended and replaced incrementally and safely.
- the use of the generic data object `TTable` (see Section 2.2) and its externalized text format `tree-table-xml` allows safe extension of data structures without recompilation.

The subscription model made the system very flexible, with alert GUIs running both on the forecasters desk, and several displaying the same data on the development machine. The GUI on the forecaster desk subscribed only to TAF alerts, whereas the development GUI subscribed to both TAF alerts and volcanic ash alerts. The subscriptions are controlled by a drop-down menu on the GUI.

The forecasters now have access to the alert GUI via a menu option on their workstations, so we can easily expose them to future versions of the system simply by uploading new Java library files.

5. CONCLUSION AND FUTURE WORK

This paper has described a pilot agent alerting system installed at the Australian Bureau of Meteorology. The pilot has given the authors confidence in the agent approach to system building in the Bureau context, as well as providing ideas for future work with this system. The subscription /leasing model employed to improve robustness and flexibility has also opened up a number of research issues for the future, namely: how will agents discover services that they need robustly and without human intervention, how will services advertise themselves to potential clients, and how to encode data so it describes its own semantics?

In the future we intend to expand the scope of the system so it alerts on an increasing variety of phenomena: for instance discrepancies between forecasts and observations for more data types like temperature and wind speed, serious weather events such as microbursts, storms, and hail, and system events like model availability, failed weather radar stations or communication links down.

This work has been carried out in a collaboration between the Bureau of Meteorology, the RMIT University Agents Group and Agent Oriented Software.

6. REFERENCES

- [1] W. K. Edwards. *core JINI*. Prentice Hall, New Jersey, 2001.
- [2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston MA, 1995.
- [3] M. Gorman, J. Kelly, C. Ryan, and C. Sanders. Meteorological data and XML. In *Meeting of Expert Team on Data Representation and Codes*, Prague, Czech Republic, 2002. Commission for Basic Systems, World Meteorological Organization. Available at [http://www.wmo.ch/web/www/DPS/ET-DR-C-PRAGUE-02/Doc6\(1\).doc](http://www.wmo.ch/web/www/DPS/ET-DR-C-PRAGUE-02/Doc6(1).doc).
- [4] A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 439–449, San Mateo, CA, 1992. Morgan Kaufmann Publishers.