Abstract
Distributed systems development is complex; especially if system needs to specify dynamic behavior, increase scalability, maintenance, and debugging. A programming paradigm specifically designed for distributed systems could help reduce the implementation complexity of distributed systems applications. In this paper we present the implementation of a distributed system for flood alert using two programming paradigms: the event rule programming [Arroyo02] and the object-oriented programming.

1. Introduction
Floods are the most destructive and costly of all natural disasters in Puerto Rico. They are difficult to control because the nature phenomenon has unpredictable behavior. Currently, flood detection and alert system relies on two major networks: ALERT and USGS. The first network is a cooperative flood-warning system called Automated Local Evaluation in Real Time (ALERT) maintained by the Civil Defense. This system maintains 40 real time gage stations that report rainfall measurements. The other network is maintained by the U.S. Geological Survey. It consists about 123 real time gaging stations that report rainfall, discharge and stage measures.

In general, the existing alert systems lack automation and integration. The applications of the different agencies work independently in dissimilar computing platforms. The data is provided to the agencies without much processing. Most of the events and alerts generation rely on human calculations and judgment which makes difficult the real time data analysis.

In order to help different agencies in the early detection of flood events a Real Time Flood Alert System (RTFAS) is being developed at UPR-Mayagüez. This system will automatically generate flood alerts to different agencies while a weather system is in progress eliminating the need for manual calculations.

2. The Real Time Flood Alert System (RTFAS)
RTFAS receives data (precipitation, river discharge, and river and lake stage) provided by the gaging stations of the U.S. Geological Survey and the Civil Defense, and the images of the Doppler Radar of the National Weather Service. This data will be processed to detect events and report hydrological data and alerts to the Aqueduct and Sewer Authority, the Puerto Rico Electric Power Authority, the Civil Defense and the National Weather Service (see figure 1).

RTFAS is an event-driven system. The report of a gaging station constitutes the most basic event. A combination of events and the evaluation of some rules can trigger new events in order to generate a major alert stage advice event.

As an event-driven system RTFAS lends itself to an implementation with the Event Rule Framework (ERF). ERF supports the specification of behavior of distributed systems in terms of how their components react to events. The heart of ERF is RUBIES (Rule Based Intelligent Event Service) [Arroyo02]. RUBIES handles events using rules, which are used to specify, at a high level of abstraction, the behavior of a DS. In addition of handling events, this event service will have the capabilities of creating and specifying events and rule properties, posting events, scheduling rules (e.g. active or inactive), and registering and notifying interested clients. Introducing this concept, end users will have the capability to define events, rules and event services in a high-level specification language.
3. The RTFAS Prototype

An RTFAS prototype was implemented using the ERF development system. However, in order to demonstrate the advantages of ERF for developing distributed systems applications a second prototype was implemented using the well known object-oriented paradigm. These prototypes will help develop validation and verification techniques as suggested in [Lambolais99, Whitetaker00].

3.1 The RTFAS model

The general RTFAS event handler model is shown in figure 2. Three general modules compose RTFAS model: The first one is the USGS Database, which represents all gage station sensors data (SensorDBObject). The different types of sensors are represented by: RainDBObject, StageDBObject and DischargeDBObject.

The second Module is the Server, which handles all event rule management. There is an EventGenerator, dedicated to communicate with USGS Database and generate primary events which are sent to an ObjectMonitor for controlling event production and consumption. Each one of these generated events are consumed and handled byEventHandler and EventSet cooperation. The EventHandler produces new MessageGenerated (final alert advice) events that are sent to another ObjectMonitor, ready for some client consumption.

The third module is the Client side, which represents the weather agencies (WeatherAgency) interested in events consumption generated by the server.

Figure 2. RTFAS Event Handler Model.
3.2 The object-oriented implementation

An RTFAS prototype has been implemented following the model described in the previous section, using an object oriented approach; and implemented entirely in Java simulating a distributed system environment. This prototype focuses only on alerts for the Aqueduct and Sewer Authority. Major alerts for this agency are: risk watch, moderate risk, high risk, low event in progress, high event in progress, and extraordinary event in progress. The description of each major event and the rules and events that trigger them is presented in [Torres01].

This prototype uses USGS gage stations data (primary events), which are stored in a common remote database. A JDBO tool is used for interconnecting the prototype and the USGS database.

After an EventGenerator has connected with Database, it extracts the needed data and generates primary events, which are sent to an EventHandler through an ObjectMonitor object in order to ensure consumption of all events generation, because events are generated in a random time and event evaluation must not loose any event.

The heart of the system is the EventHandler class, because it implements many rules attempting to represent the real behavior of the system. Each rule has many conditions that events should satisfy. If these conditions are met, another events are triggered and stored in an EventSet object. This process is continually repeated until there are no more rules to evaluate and a major alert advice event is finally generated (MessageGenerated object). All MessageGenerated objects are sent to an ObjectMonitor for controlling Client remote consumption.

The Client is constantly invoking for getting new MessageGenerated objects through EventHandler and ObjectMonitor server classes. RMI (remote method invocation) was used for communication between the Server side and the Client side, The Client side provides alerts and messages to the agency through a user interface.

The prototype was simulated and validated using the USGS data collected during a rainy day from “La Plata Basin”. The major alerts generated and the time they were generated by the prototype were compared with major alerts and times manually calculated.

The object oriented implementation presented some disadvantages. First, it is not easy to make a scalable system. Every time a new rule or event needs to be added, a new rule implementation must be made. In addition, an analysis must be made to determine if it affects other rules in order to make the necessary changes and recompile all code again.

Another disadvantage is the difficulty to implement and specify the real behavior system, especially if it has a dynamic behavior. Also, adding new rules and events, increases the complexity of the system, making it difficult for maintaining and debugging.

3.3 The event-rule implementation

The RTFAS model used for the rule event implementation was the same one described in section 3.1. The event-rule prototype development will follow an ERF philosophy; it will specify system behavior in terms of events and rules using the tools and services that ERF provides.

DataBase will provide the entry data for the system. The EventGenerator class will connect DataBase to the prototype through the JDBO tool.

In the event-rule prototype EventSet and EventHandler are implemented with the RUBIES and EventChannel classes respectively. These classes were implemented to handle events, evaluate them, and constantly interact with each other. These are services provided by ERF and the user does not need to worry about implementing them.

In contrast to the object-oriented implementation, EventGenerator will be a Distributed Object [Arroyo2002], that will work as a client of ERF. Events will be implemented as objects. Events of the same type will be represented and defined by means of an event type specification.

The user must implement all event rules using RDL (Rule Definition Language) a service provided by ERF. The syntax for defining a rule in RDL is as follows:

```
[package <package_name>]
[import <package_name>]
rule <rule_id>
[priority <priority_no>]
on <trigger_events>
[use <usage_specifications>]
[if <condition>
then <actions>
[else <alt_actions>]
do <actions>
```
As an illustrative example let’s consider the creation of a group of events that will report the accumulation in a gage station during the last 15 minutes. This event will be triggered by another type of event that reports the precipitation in the same gage station. The following code correspond to the two the declaration of the two types of events.

```java
package aaa;
import erf.*;
public class RainfallReport extends Event {
    public double value;
    public double loc;
}

public class RainfallAccumulation_15 extends Event {
    public double accumulation;
    public double loc;
}
```

The RainfallReport events correspond to the precipitation report generated by the gage station. The fields “value” and “loc” represent the measurement and location of the sensor of the sensor respectively. The RainfallAccumulation_15 are generated every time a RainfallReport event occurs. The field “accumulation” represents the accumulation of the sensor during the past 15 minutes while the field loc represents the location of the sensor.

The code that generates the RainfallAccumulation_15 events follows:

```java
package accum;
rule rainfall_acummulation_15
on {RainfallReport [(ts()<=15) && (loc==16)]}RF1
    post RainfallAccumulation_15{accumulation = RF1.sum("value"), loc = 16}
end;
```

4. Future Work

The next step will be the development of simulation techniques to validate the event rule prototype. Then a user interface will be developed to display the flood alerts for different agencies in a way that is easy for them to understand.

In addition, we plan to conduct a programming test to compare the event rule programming paradigm with the object-oriented paradigm in terms of how programmers understand the programming paradigms, the time it takes them to code algorithms and the quality of code generated.

References


