2005－UBI－8（19）
2005／6／9

A Rule-based System for Sense-and-Respond Telematics Applications

SangWoo Lee[†], Jonathan Munson[‡], DaeRyung Lee[†], Gerry Thompson[‡], JungSun Park[†]

[†]IBM Ubiquitous Computing Laboratory, Seoul
[‡]IBM T. J. Watson Research Center, Hawthorne, New York
lsw@kr.ibm.com, jpmunson@us.ibm.com, drlee@kr.ibm.com,
gerryt@us.ibm.com, jspark@kr.ibm.com

ABSTRACT

We introduce technology for supporting telematics-oriented sense-and-respond applications. Those are a telematics-oriented event detection service, and programming framework supporting it, that enables application developers to more easily develop applications based on the sense-and-respond model. The system provides a rule-based programming model in which the application is partitioned in two parts: (1) a set of rules that operate on low-level position-update events and which, when triggered, produce high-level, application-defined events; and (2) logic that acts on the high-level events, which is deployed outside the event detection service, and typically within the environment of the enterprise deploying the rule. Programmers represent events of interest in the form of rules, which operate on both input received from the data acquisition systems as well as data resources provided and managed by the application programmers. Programmers declare the inputs the rules require, and the system is responsible for acquiring the inputs. A high-level programming framework assists programmers in defining the set of rules, and the actions that respond to events from the rules, and in deploying the rules to the system.

1 INTRODUCTION

In 2003 the Ministry of Information and Communication of the Republic of Korea formulated a strategy for the development of information technology, known as "8-3-9"—introducing and promoting *eight* services, building *three* infrastructures, and development of *nine* new growth engines. In 2004 the government of Korea and IBM jointly created the Ubiquitous Computing Laboratory in Seoul, where certain projects funded by the 8-3-9 program are being carried out. Telematics is both one of the eight services and one of the nine new growth engines, and is the subject of two of the UCL projects, which began in August 2004.

In this workshop we introduce one of those projects, the focus of which is to develop technology for supporting telematics-oriented sense-and-respond applications. We present an early result of the project, a prototype service we call the "Telematics Event Detection Service." The service enables application programmers to define situations of interest in the form of rules, and to deploy these rules to the Telematics Event Detection Service (TEDS). The service receives inputs from vehicles and from other data sources and evaluates the rules over them. When rules "trigger," notifications are sent to the applications that deployed them.

To test basic functions of the service, we created a simple fleet management application around the function known as "geofencing." The application expresses the geofences as TEDS rules, deploys them to TEDS, and in turn receives notifications when trucks enter and leave the geofence. The application is described in Section 3.1.

Geofencing is but the simplest of scenarios that we support with TEDS. We have identified a more complete set of target scenarios that TEDS should support, and then have used these to define the service's range of functionality.

1.1 Target Scenarios

IBM has many discussions with customers and potential customers about applications the customers are interested in. The following set of scenarios, which span a range of application domains, has been distilled from some these discussions.

Detecting changes in vehicle population density. A public-safety organization is interested in monitoring the deployment of its emergency vehicles to ensure that a vehicle can be dispatched to any location within a certain time. It would like to be alerted when the relative values between local densities change by a certain amount.

Track vehicle progress with respect to schedule. An operator of a fleet of delivery vehicles wishes to track its vehicles' progress with respect to their schedules, and be alerted when a vehicle is more than a certain amount behind schedule. It may also wish to be

alerted when a truck is returning empty to the warehouse so that it may begin preparing the truck's next load.

Location-based promotions. A marketing company handling promotions for certain establishments would like to be able to notify consumers as they approach those establishments with special promotions in effect at that time. It wants to send the promotions only to those who have accepted the service and who have been receptive to such promotions in the past, and it needs to avoid sending the promotion repeatedly.

Location-based warnings. A highways department in a fog-prone area wishes to warn motorists of dangerous fog conditions ahead so that the motorists will slow down. They need to be able to create such warnings quickly, and they need be able to distinguish between vehicles moving toward the conditions and those moving away.

Congestion detection. A highways department would like to have real-time congestion maps of area highways, and to be alerted when aggregate speeds in particular areas fall below a certain threshold.

Monitoring speed of fleet vehicles. An operator of fleet of delivery vehicles would like to ensure that its drivers obey posted speed limits Enterprise user defines the set of vehicles to monitor, and creates rule that will periodically compare their speeds with the speed limits in effect at the location where the speed was recorded.

Detection of excessive speeding. Public safety officials wish to know when aggregate speeds in a region are excessively high, either because of the accident rate in the area or because of particular road conditions.

Proximity to other mobile entity(ies). A rule is defined that when a certain number of the friends are within a given distance, causes a message to be sent to the user.

Pervasive gaming. An automobile club wishes to sponsor a combination road rally/treasure hunt, where members pair up and follow successive clues to a prize, but are penalized for exceeding speed limits.

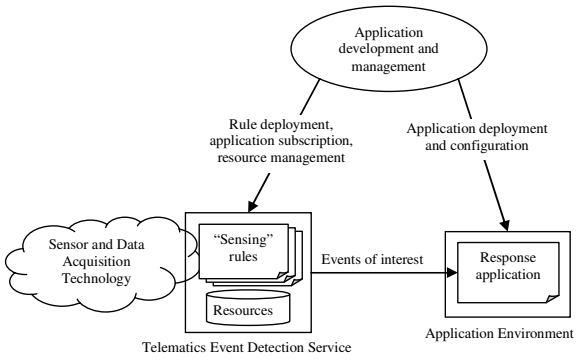

Figure 1. TEDS Logical Programming Model.

1.2 The Value of a Shared Infrastructure

We believe there may be substantial value in a general-purpose, shared, infrastructure that could support any and all of the above services, simultaneously, over a large set of vehicles. Such an infrastructure would enable service providers to reach a broad customer base, without requiring an investment in their own infrastructure. With many services using the infrastructure, no one service must bear the entire load.

2 THE TELEMATICS EVENT DETECTION SERVICE

Reflecting the structure of sense/respond applications, TEDS offers a rule-based programming model in which the application is partitioned in two parts: (1) a set of rules that operate on low-level position-update events and which, when triggered, produce high-level, application-defined events; and (2) logic that acts on the high-level events, which is deployed outside TEDS, and typically within the environment of the enterprise deploying the rule. Figure 1 illustrates.

As shown in Figure 1, programmers represent events of interest in the form of rules, which "trigger" (return true) when the events are detected. Rules operate on both input received from the data acquisition systems as well as data resources provided and managed by the application programmers. An example of an application resource is the geographical polygon describing a warehouse site's area, which would be used in a rule detecting a truck's entrance to the site. Rules and rule resources are described in Sections 2.1 and 2.5.

Applications subscribe to a rule to receive notifications of its triggering. The notification received includes any results that may have been computed in the evaluation of the rule. For example, a rule that determines if a subscriber's position lies within any zones that have a promotion associated with them will, if the position is within any such zones, include the set of zones (via identifiers) in the notification to subscribed applications. Applications may optionally choose to be notified only when the triggering status with respect to a particular subscriber changes, either from false to true, or true to false. For example, an application subscribing to a rule that detects entry to a warehouse site may wish to receive notifications only upon a truck's entry and exit of the site.

2.1 Rules

TEDS rules are essentially condition/action specifications, and may be as simple as a single Boolean expression, or a more complicated program with internal state. A rule condition is a logical expression (i.e.,

evaluates to true or false) composed of spatial and temporal logical functions joined through AND, OR, and NOT operators, and scalar functions joined with the usual relational operators. TEDS offers a number of built-in functions, which are described below.

Rule inputs include an entity's location (along with speed, heading, and accuracy estimates) received from the positioning technology, and any other input data available from the data acquisition technology.

2.2 Rule Functions

The spatial rule functions offered by TEDS operate implicitly on a subscriber position report, and have parameters associated with them that will also be input to the function. The *polygonID* and *pointID* parameters are identifiers for polygons and points. Polygons and points are specific kinds of rule resources, support for which is discussed in Section 2.5. **Table 1** below defines built-in logical functions of TEDS; **Table 2** defines built-in scalar functions.

Table 1. Built-in Logical Spatial Functions.

Name	*Parameters*
Description	
containedIn	polygonID | polygonSetID
True iff the entity position is contained in one or more of the given polygon or set of polygons.	
within DistanceOf	(pointID | pointSetID) distance
True iff the entity position is within the given distance from the given point or set of points.	
within DistanceOf	(entityID | entitySetID) distance
True iff the entity position is within the given distance from the position of the given entity or one or more of the given set of entities.	
hasIdentity	entity | entitySet
True iff the entity has the given identity or is a member of the one or more of the given entity sets.	
<, <=, ==, !=, >=, >	scalar_expr1 scalar_expr2
True iff the given relation between the two scalar expressions is true.	
elapsed	timer_identifier
True iff the timer identified has expired since the last time the expression was evaluated. Timers are declared, set, and reset explicitly, outside of condition expressions, in a language-specific manner.	
before/after	time
True iff the timestamp of the SPR is before/after the given time.	

Table 2. Built-in Scalar Functions.

Name	*Parameters*
Description	
distance From	point | susbcriber
Returns distance (in meters) from the given entity position to the specified point or entity.	
distance	point (route | path)
Traveled	
Returns the distance the entity has traveled from the given point along the given route or path. A route object is a precomputed route the susbscriber is expected to take; a path is a series of positions recorded for the entity. Use of a path object generates an implicit rule that will record the entity's positions.	
distance Traveled	time (route | path)
Returns the distance the entity has traveled since the given time along the given route or path.	

2.3 Rule Language

In the course of developing TEDS, we used multiple rule languages. Our first "language" was the construction of rules through building Java object structures of rule objects, much like an expression tree. We considered this to be not sufficiently easy to use. We then developed our own rule language and compiler. In the end, however, we realized that our primary value lay not in the rule language itself, but rather with the functions we provide outside the rule engine, such as rule optimization, application resource management, trigger reporting, and subscriber management (all discussed in later sections). Thus we adopted the approach of embedding existing rule engines within TEDS. This not only enables us to leverage the strengths of the particular rule engine in use, but gives us more flexibility in deployment as well.

The rule language we currently use is ABLE [3], an environment for building intelligent agents. ABLE supports different kinds of rule programming, through the variety of inference engines it provides. These include backward chaining, forward chaining, Rete networks, and simple sequential evaluation, among others. Below is an ABLE rule set that triggers when any one a set of trucks is in a no-standing zone.

```
ruleset FMGeofence {
  import
com.ibm.locutil.able.SubscriberPositionUpdate;
  import com.ibm.locutil.able.LUContext;
  import com.ibm.locutil.able.LURuleResults;
  import java.util.ArrayList;
  variables {
    SubscriberPositionUpdate sub;
    LUContext luCtx;
    LURuleResults results = new LURuleRe-
sults();
    ArrayList fcnResults = new ArrayList();
  }
  inputs { sub, luCtx }
  outputs { results }
  void process() using Script {
  : if (sub.isMemberOf("DeliveryShift1") &&
        sub.containedInPolygonSet(
           "FM:NoStandZones", fcnResults))
    then {
      results.setDidTrigger(true);
      results.addResult("zones", fcnResults);
     }
```

```
    }
}
```

By comparison, the same rule with our own (now abandoned) rule language is:

```
memberOf("DeliveryShift1")
&& containedInPolygonSet(("FM:NoStandZones")
```

Despite the obvious loss in succinctness by embedding a "foreign" rule engine such as ABLE, we still feel the advantages in flexibility outweigh the disadvantages. And for more complex rule sets, such as those that maintain local state, the disadvantages become less apparent.

We will continue to evaluate other rule languages and frameworks for use with TEDS, some of which are described in the section "Related Work." We are interested in rule frameworks that provide complementary value, such as strong support for temporal patterns, and that enable programmers to produce readable, easy-to-understand rule sets.

2.4 Subscriptions

Subscriptions to rules exist independently of the rules themselves. Subscriptions contain a number of parameters, including the rule being subscribed to, the address to send rule-triggering events to, and the report type, which is *full*, or *delta*. Full reports consist of the current set of subscriber position reports for which a rule was satisfied; delta reports consist only of the subscriber position reports that resulted in a change in the result of the rule evaluation with respect to a particular subscriber. Other parameters are not described here.

2.5 Rule Resources

Rules typically involve functions that relate data in the subscriber position report to other data. This data may be provided by the application, such as geometries to compare a subscriber's location to, or a set speed threshold to compare the subscriber's speed to. Or, the data may be generated by repeated evaluation of the rule. TEDS provides two forms of data store for these purposes.

Static Data. TEDS offers a simple data store to provide persistence for the data referred to in rule parameters. The global store is accessed implicitly in the *polygonSet* and *pointSet* parameters, and may be accessed explicitly as well (functions not shown). The data store may also be accessed from outside rules, through the TEDS application interface. The data store is not typed, although geometry types for points and polygons are stored differently in order to take advantage of databases with spatial functions and the spatial indexing supporting them.

Mobile Data. TEDS also provides a form of persistent data that is associated with particular subscribers. For example, a rule may wish to record a subscriber's time of entrance to a particular geographical area so that it can determine the total time a subscriber has been in the area. For purposes such as this, TEDS provides a data store associated with each subscriber.

3 TEDS FIRST PROTOTYPE

Figure 2 illustrates the architecture of our prototype Telematics Event Detection Service. The main component is the Telematics Event Detection Engine (TEDE). Through a TEDE client, an application submits rules and provisions any resources required by the rule. Rule triggering events are received from the TEDE asynchronously through an interface the application must provide.

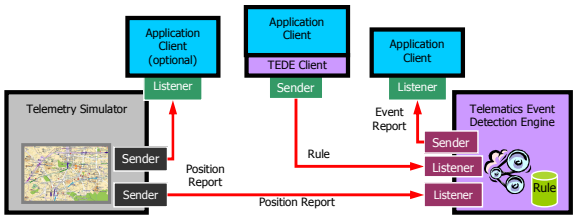

Figure 2. TEDS Demonstration Prototype.

As we as yet have no actual set of vehicles to receive input from, we developed a telemetry simulator that enables vehicle movements to be simulated. Through a map-based interface the simulator operator creates paths on the map, giving each segment its own speed.

3.1 Example: Fleet Manager Application

We developed "Fleet Manager" as an example of an application that takes advantage of the high-level event detection capabilities of TEDS. It supports various fleet management scenarios where fleet operators wish to be notified when vehicles of their fleet enter or leave certain geographical areas. It enables operators to quickly and easily define the areas and the vehicles to be monitored. It then transforms this information into a rule before deploying to TEDS. One of our objectives in developing this application is to explore high-level rule creation tools.

The Fleet Manager Application display (Figure 3) shows real-time positions of vehicles on the map and also represents locations symbolically by showing associations between vehicles and zones, as shown in the lower right window of Figure 3.

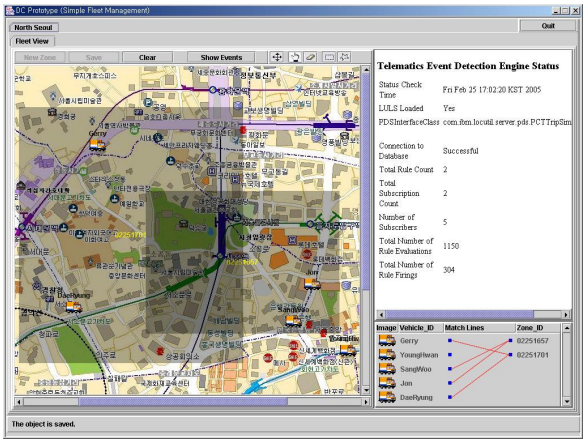

Figure 3. Fleet Manager Application.

3.2 Lessons from Prototype

In developing the Fleet Manager application for the first TEDS prototype, we realized it had a number of shortcomings. First is that application programming requires a high level of effort to create interfaces to the TEDS system. TEDS provided no support for inserting rules into the system or for receiving events from the system, thereby requiring low-level programming from the application developer. Second, TEDS provided no framework to support the sense-respond programming model. Third, the prototype did not sufficiently support certain important features, such as subscriber groups.

These shortcomings are being addressed in the second prototype. In addition, we are also addressing the fundamental way input data is acquired, we are enabling rules to operate on a wider range of inputs, we are enabling the system to acquire input data from a wider range of sources, and we are also supporting application-provided rule functions. This new functionality is briefly described in the following section.

4 TEDS VERSION 2 AND THE S&R FRAMEWORK

To make it easier to develop applications based on the sense/respond model, we have developed a framework providing both develop-time and run-time support. The framework, called the S&R Framework, relates to TEDS as shown in Figure 4 below.

4.1 S&R Framework

The S&R Framework is analogous to the Struts framework for developing Web applications based on the Model-View-Controller paradigm. With the S&R Editor, the developer defines the set of rules, and the actions for responding to events from the rules, that constitute a sense/respond application. The SNR file produced by the editor, together with the rule files and respond-action classes referenced by it, are then deployed to the Subscription Set Manager, which interfaces to TEDS to deploy rules and subscribe to them, and to the S&R Respond Controller, which receives the rule-triggered events from TEDS and invokes the related respond classes.

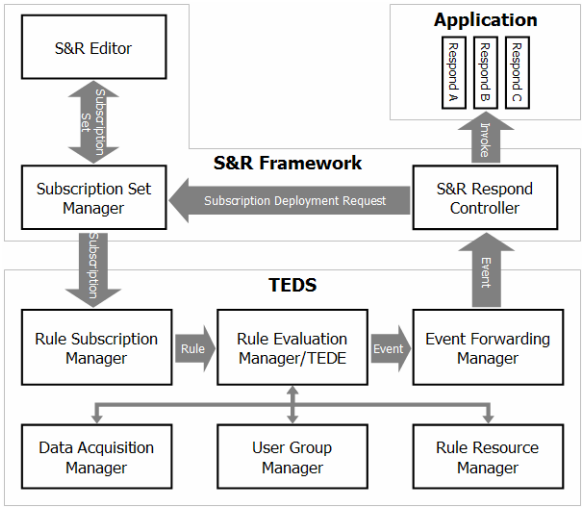

Figure 4. TEDS Version 2 and S&R Framework

4.2 Extended Rule Inputs

The new TEDS architecture enables rule programmers to define rules that operate over a range of inputs, beyond the location input supported by the first prototype. Rule inputs may include any sensor data received from mobile entities such as fleet vehicles, as well as business-related data such as “pickup needed” signals.

Acquisition of these inputs is the responsibility of the Data Acquisition Manager, shown in Figure 4, given the list of inputs a rule requires, and the rule’s specified evaluation frequency, the DAM is responsible for acquiring the inputs at the rate specified and supplying them to the Rule Evaluation Manager.

5 RELATED WORK

As a framework for programming sense-and-respond applications, TEDS is related to a broad array of work in real-time monitoring, event-driven systems, context-aware computing, and active databases. However, TEDS has a focus on those situations involved in telematics. Furthermore, it does not address applications where extended and complex patterns of events are of interest. Space does not permit a full discussion of related technologies, but we briefly note those that are most closely related.

Chandy et al [4] describe an abstract programming model for dynamic applications that corresponds closely with the programming model of TEDS. The iSpheres Halo [9] platform offers a complete system for sense-and-respond programming. It does not, however, offer specific support for spatial events.

Some work has addressed the specific application of location-based notification. Chen et al [5] describe a publish/subscribe system for spatial triggering, focusing on efficient matching algorithms. Munson and Gupta [10] describe another spatial triggering system, focusing on an architecture for scalable implementation of this system over millions of users. This work is a predecessor of the system described here.

A set of spatial predicates and means of composing them are offered by Bauer and Rothermel [2] and Nelson [11] presents a similar a set of spatial predicates, and extends this with some temporal operators.

Stronger support for event patterns is offered by iQL [6]. iQL's functions and operators apply to generic numeric and textual data, and it does not offer specific support for spatial data or location input.

Houdini [8] is a rule language and framework used for user-preferences policy management for telecommunication services. As such, it addresses a different set of applications than TEDS, and does not address spatial events.

Amit [1] is a rule-based framework for detecting situations over potentially long-running event streams. Like Houdini, it could also serve as a base rule engine for TEDS.

6 CONCLUSIONS AND FUTURE WORK

We have described the Telematics Event Detection Service, an infrastructure that we believe can enable a wide range of event-driven telematics services. The rule-based programming model of TEDS is a key feature that we plan to exploit further to address scalability. We believe it will enable the units of computation to be both replicated widely to make efficient use of resources, and pushed out close to the sensors to reduce communication needs in the wide-area network.

We have also described our initial effort at developing a high-level programming framework around the sense-and-respond application model. We believe that frameworks such as this are an important element in enabling programmers to exploit this model. We hope to extend the S&R Framework with high-level support for the development of rules themselves.

A primary goal for the future is to address the issue of scalability. We believe the flexibility of our service may make it attractive for service providers with large subscriber bases, such as large telematics service providers or wireless carriers. It would enable them to offer a wide range of services to their subscribers. For that to be feasible, however, our service must be scalable to a subscriber base numbering in the millions, and the number of services numbering in the hundreds or thousands. The resulting load on the service could be millions of rule evaluations per second. We must therefore have an architecture that can scale to this load, not only in the computational load of rule evaluation, but in the data transmission load involved in getting inputs to the rule engines. We plan to address this in a next phase of the project.

7 REFERENCES

1. Adi, A., Etzion, O. Amit – The Situation Manager. The VLDB Journal, Springer-Verlag, Heidelberg, Vol. 13, No. 2.
2. Bauer, M., Rothermel, K. Towards the Observation of Spatial Events in Distributed Location-Aware Systems. In Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops, 2002.
3. Bigus, J. P., Schlosnagle, D. A., Pilgrim, J. R., Mills, W. N. III, Diao, Y. ABLE: A Toolkit for Building Multiagent Autonomic Systems. IBM Systems Journal, Vol. 41, No. 3, 2002.
4. Chandy, K.M., Aydemir, B.E., Karpilovsky, E.M., Zimmerman, D.M. Event-Driven Architectures for Distributed Crisis Management. Presented at the 15th IASTED International Conference on Parallel and Distributed Computing and Systems, November 2003.
5. Chen, X.Y., Chen, Y., Rao, F., An Efficient Spatial Publish/Subscribe System for Intelligent Location-Based Services. Proceedings of Second International Workshop on Distributed Event-Based Systems, San Diego, 2003.
6. Cohen, N.H., Lei, H., Castro, P., Davis, J.S. III, Purakayastha, A. Composing Pervasive Data Using iQL. In Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications, June 2002, Callicoon, NY.
7. Federal Communications Commission. Enhanced 911. http://www.fcc.gov/911/enhanced/
8. Hull, R., Kumar, B., Lieuwen, D., Patel-Schneider, P.F., Sahuguet, A., Varadarajan, S., Vyas, A. "Everything Personal, Not Just Business": Improving User Experience Through Rule-based Service Customization. In International Conference on Service Oriented Computing (ICSOC 2003), Rome, December, 2003.
9. iSpheres Corporation. Halo™. http://www/ispheres.com.
10. Munson, J.P., Gupta, V.K. Location-Based Notification as a General-Purpose. In Proceedings of the Workshop on Mobile Commerce, Atlanta, 2002.
11. Nelson, G.J., Context-Aware and Location Systems. Ph.D. Thesis, University of Cambridge Computer Laboratory, 1998.