Spatial Expressions and Rules for Location-based Services in Oracle

Aravind Yalamanchi, Ravi Kothuri, and Siva Ravada
Oracle Corporation, Nashua NH 03063
{aravind.yalamanchi, ravi.kothuri, siva.ravada}@oracle.com

Abstract

In this paper, we present Oracle technologies for handling spatial events and specifying arbitrary spatial rules. The framework allows construction of arbitrary expressions involving spatial predicates, indexing them using an expression filter index, and incorporating them as conditions within an event-condition-action (ECA) rule. This framework provides a seamless integration of multiple components within Oracle and provides a rich set of functionality for location-based service applications.

1 Introduction

Recent advances in mobile positioning technologies such as GPS, and A-GPS have simplified and reduced the cost of automatic location acquisition thereby enhancing the use of location in a wide variety of applications. Examples of such applications include location-based services, routing and tracking of shipments, luggage or similar items, and supply chain management using RFID. Of these, location-based services (LBS) use location explicitly. Simple examples include geocoding, mapping, and routing (driving directions) [HB04]. Advanced applications combine positioning and location-based analysis with an event alerting/management systems. [JC+04] lists several categories of LBS applications such as traffic coordination in the presence of bottlenecks, safety-related services and location-aware content delivery services. Specific examples in the last two categories are:

- Secure-Zone Monitoring: Secure regions around important places of interest could be defined using spatial buffering functions. These regions can later be used in screening flights entering those regions for security violations.
- Car Buyer Subscription Service: An interested buyer can register for a service and specify preferences for his ideal car using attributes of the car (model, price, etc.) and the distance he will travel to obtain the car. When a car enters the market, the potential list of buyers whose preferences are met by the car being sold is notified.

In order to enable the above types of location-based service applications, event management functionality capable of handling spatial events and preferences defined on such spatial events is required. In this paper, we describe a framework for managing spatial events inside an Oracle database. This framework can be easily deployed in a variety of location-based service applications including the ones described above.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

2 Spatial Event Management in Oracle 10g

To support location-based service applications, Oracle provides a generic framework as illustrated in Figure 1. The framework consists of the Rules Manager component, the Spatial and the Expression filter engines, a visualization component called Mapviewer and an alerting component.

The Rules Manager allows users to define rules inside an Oracle database. A rule is modeled using *event*, *condition*, and *action* semantics [WC96]. The spatial and expression filter components enable the condition part of a rule, and the alerting component enables the action part of a rule. Events triggering the rule can be regular database events such as triggers. Some of these events can be triggered due to the positioning aspect of a mobile user. The Oracle Application Server (Wireless) component supports GPS positioning and network-based positioning interfaces such as MLP, ParlayX provided by mobile operators. Location-based service applications can make use of these different technologies available in the Oracle Database and Application Servers. In this paper we describe the overall framework for managing spatial events using the spatial, expression filter and rule manager components of the Oracle Database Server. In the next few sections, we describe each of these components in more detail.

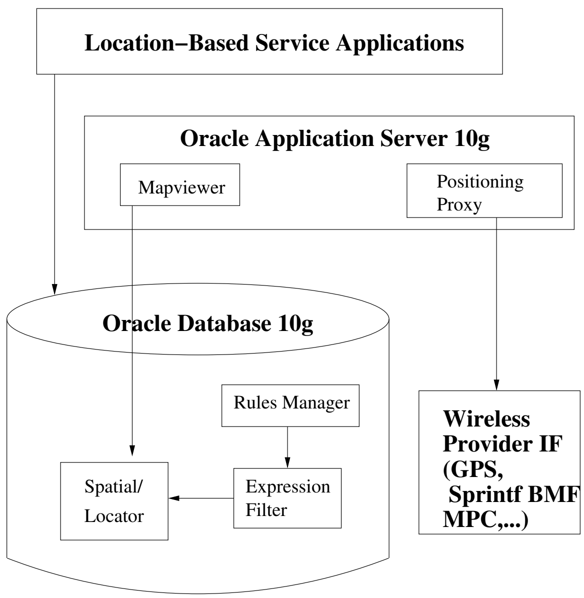

Figure 1: Framework for Spatial Events in Oracle.

3 Oracle Spatial

In most applications, location-based services heavily rely on GIS-type functionality. The Spatial engine of Oracle provides the required GIS functionality for location-based services. The functionality includes:

Data Model for Location data: Spatial defines a data type called SDO_GEOMETRY to store locations of data items. Users can create tables with columns of type SDO_GEOMETRY and index such columns using specialized indexes such as R-trees, or Quadtrees [KRA02]. Such indexes facilitate fast responses to location-based queries.

Query and Analysis: Relational operators such as $<$ are used to compare numeric or string attributes of a table. For the SDO_GEOMETRY columns, Spatial provides specialized operators to perform location-based filtering, query, and analysis. Commonly used operators include (but not limited to) the following:

- SDO_WITHIN_DISTANCE: This operator identifies all rows of a table where the corresponding location data are within a specified distance of a query location. This operator can be used to identify the restaurants in 2-mile radius of a user's current location.
- SDO_RELATE: This operator identifies all rows of a table where the corresponding location data interacts in a specified way with a query region. The types of interaction allowed with this operator include - inside, contains, covers, and anyinteract. If a mobile user has a preferred region of interest, this operator can be used to identify the restaurants in this preferred region.
- Others: OGC and SQL/MM recommend a list of other spatial functions. A majority of these functions/operators are supported by Oracle Spatial and are listed in detail in the documentation [Ora10g].

Geocoding: Oracle Spatial provides functions to convert user's addresses to geographic locations (usually specified as <longitude, latitude> coordinates). Such conversion is called geocoding in GIS parlance. Reference data used in geocoding is obtained from third-party vendors such as NAVTEQ.

Routing: Spatial provides APIs to compute routes between a starting and an ending location. The route includes driving directions from the starting location to the ending location.

4 Expression Filter

Traditional databases support storing data values in relational tables and finding interesting data values using a *conditional expression* in the WHERE clause of a SQL query. Recently, an inverse paradigm has generated considerable interest in the database community. This paradigm stores the conditional expressions in a relational table and these expressions are matched with a transient data item specified in a SQL query [YSG03]. The Expression Filter feature [OraA10g] in Oracle database provides appropriate functionality in this paradigm. For instance, in a Car-Buyer Subscription Service, Buyers' interests in cars can be captured as conditional expressions that are stored in a column of Expression data type in a relational table(Table 1).

Customer_id	Zipcode	..	Interest
1	32611	..	Model='Taurus' and Price < 15000 and Mileage < 25000
2	03060	..	Model='Mustang' and Year > 1999 and Price < 20000
..	..	..	..

Table 1: Customer table specifying conditional expressions in the *Interest* column.

Each expression column is associated with an object type that defines necessary metadata (attributes used in the expressions and their data types) for the expressions. For the previous application, a Car4Sale object type with Model, Mileage, Year, and Price attributes is configured as the metadata for the expression column.

The Car-Buyer Subscription Service can be modeled to monitor cars as they go on sale. For this purpose, a car, represented as an instance of the Car4Sale object type, can be matched with the expressions in the Interest column of the Consumer table using the following query.

```
SELECT customer_id FROM Customer
  WHERE EVALUATE(Customer.interest,
               AnyData.convertObject
                 (Car4Sale('Mustang', 19000, 25000, 2001))) = 1;
```

The EVALUATE operator, provided by the Expression Filter feature, facilitates evaluation of conditional expressions (stored in the *Interest* column) for a data item using standard SQL statements. A specialized index can be defined on the column storing expressions to speed up the above query.

```
CREATE INDEX cstidx ON Customers(interest) INDEXTYPE IS EXFSYS.EXPFILTER;
```

For the purpose of indexing expressions, the predicates in the expression set are grouped based on the commonalities in the predicates and stored in a relational table called the predicate table [YSG03]. A subset of the predicate groups are identified as *indexed predicate groups* based on the frequency and selectivity of corresponding predicates. Indexes are used for the evaluation of such predicate groups. Non-indexed predicate groups are processed in the final stage of expression evaluation.

4.1 Spatial Predicates in Expression Columns

In the Car-Buyer Subscription Service, the buyer may often have a preference on the spatial proximity of the car in addition to the traditional attributes (Model, Mileage etc.,). To support such applications, the metadata associated with the Expression column may be extended to include an attribute, VehLoc, that captures the location data for the car going on sale. Using this attribute, the customer may now restrict his search to a region, for example, a 2-mile radius of his current location (say longitude -79, latitude -38). This is possible by allowing a mix of scalar predicates using operators such as < (as in Price < 15000) and spatial predicates using SDO_WITHIN_DISTANCE and SDO_RELATE operators in the stored expressions. The Table 2 [1] shows one such example.

Customer_id	**Zipcode**	**..**	**Interest**
1	32611	..	Model='Taurus' and Price < 15000 and Mileage < 25000 and **SDO_WITHIN_DISTANCE(VehLoc, POINT(-79, -38)**, **'distance=2 unit=MILE')='TRUE'**
2	03060	..	Model='Mustang' and Year > 1999 and Price < 20000
..	..	..	..

Table 2: Customer table using spatial operator in the *Interest* column.

When an index is defined on the Interest column, all the spatial predicates form an *indexed predicate group* and the predicate table structure is extended to include an SDO_GEOMETRY column to store the *spatial regions* that are computed from these predicates. For the previous example with SDO_WITHIN_DISTANCE operator, a 2-mile buffer around the specified location is computed and stored in this column. In the case of a spatial predicate involving SDO_RELATE operator, the location specified in the predicate is stored in this column and the exact spatial interaction (inside, contains, touches etc.,) to be matched is captured as a function that is processed in the final stage of expression evaluation. A spatial index is defined on the column storing spatial regions in the predicate table. In order to evaluate the spatial predicates, the location data from the data item passed to the EVALUATE operator is matched with the *spatial regions* in the predicate table for any interaction [2]. For the expressions matching the location data (and also matching on all other indexed predicate groups), the remaining predicates associated with the expressions are evaluated for conclusive matches.

When a new car with an appropriate location enters the market, an instance of the Car4Sale object type, which now includes the VehLoc attribute for capturing the vehicle location, is passed to the EVALUATE operator as shown below.

[1] In this table, POINT is a function that takes longitude, latitude coordinates and returns an SDO_GEOMETRY.

[2] Note that SDO_WITHIN_DISTANCE operator is equivalent to an any interact operation on computed buffer

```
SELECT customer_id FROM Customer
  WHERE EVALUATE(Customer.interest,
                 AnyData.convertObject
                   (Car4Sale('Mustang', 19000, 25000, 2001, POINT(-79, -37.99)))
                 ) = 1;
```

Note that the POINT(-79, -37.9999) function captures the location of the car. This query uses the index defined on the Interest column to evaluate the expressions based on spatial and non-spatial predicates. The expression filter index in turn uses the spatial index defined on the column storing spatial buffers in the predicate table and the bitmap indexes defined for the other indexed predicate groups together to filter the expressions based on multiple predicate groups simultaneously. [3]

In order to provide alerting service for matching customers, the results from the above query can be used as input to other database components supporting notification framework. For example, if the consumer table includes the email address of each consumer, the UTL_SMTP package in Oracle database can be used to send out alerts as email messages to the matching customers. Alternately, if the matching of a car to a customer's interest should trigger a workflow, Advance Queues [Ora10g] in the Oracle database may be used.

5 Rules Manager

The latest release of Oracle RDBMS provides the Rules Manager framework to define and enforce *Event-Condition-Action(ECA)* rules [WC96] in the database. The expressions described in the previous section form the basis for the *condition* part of a rule. In this section, we will focus on the remaining aspects of managing rules such as *events* and *actions*.

An *event* is the occurrence of some state change in a component of a software system, made visible to the external world [ON03]. An event is an instance of a data structure, called the *event structure*. The event structure provides the necessary vocabulary for the rule condition by specifying the list of attributes and their data types. A rule condition defined using the attributes from an event structure evaluates to true or false for an instance of the event structure (event). The *action* associated with the rule is a program that executes in the host application when the corresponding rule condition evaluates to true.

The rules supported by the Rules Manager can be represented using standard ECA notation [Han92], where the ON clause identifies the event structure for which the rule is defined, the IF clause identifies the rule condition and the THEN clause identifies the rule action. The following code sample shows two rules defined for the Car4Sale event structure.

```
ON   Car4Sale(Id, Model, Mileage, Year, Price, VehLoc) car
IF   Model = 'Taurus' and Price < 15000 and Mileage < 25000 and
      SDO_WITHIN_DISTANCE(VehLoc, POINT(-79,-38)
                          'distance=2 unit=MILE') = 'TRUE'
THEN SendCarInfo('foo@abc.com', car);

ON   Car4Sale(Id, Model, Mileage, Year, Price, VehLoc) car
IF   Model = 'Mustang' and Year > 1999 and Price < 20000
THEN SendCarInfo('bar@abc.com', car);
```

In the database, the event structure (Car4Sale) is captured as an object type with the complete set of attributes. Each instance of this object type (event) can be explicitly added to the Rules Manager to evaluate the corresponding rules. In a rule based application, multiple rules may be defined using the same event structure. These rules together form a logical set called the rule class and they represent one unit of evaluation. That is,

[3] See Oracle documentation for details on multi-dimension filtering using index predicate groups.

for each instance of the event structure (event), the rules in a rule class are all evaluated and the actions for the rules matching the event are executed.

The rule actions are carried using a callback procedure, which at the time of action execution has access to the event instance and the rule it matched. Hence the attributes that determine the appropriate action for each rule, such as the email address of the subscriber, are associated with the rules as action preferences.

At the time of rule class creation, an association is established between the event structure, the action callback procedure and a rule class table that acts as the repository for rules. The following commands are used to create a rule class for 'CustomerRules' application involving a 'Car4Sale' event structure and a 'CustomerAlerts' action callback.

```
BEGIN
  dbms_rlmgr.create_rule_class(rule_class_nm => 'CustomerRules',
                               event_struct  => 'Car4Sale',
                               action_cbk    => 'CustomerAlerts');
END;
```

rlm$ruleid	**EmailAddress**	**..**	**rlm$rulecondition**
1	foo@abc.com	..	Model='Taurus' and Price < 15000 and Mileage < 25000 and SDO_WITHIN_DISTANCE(VehLoc, POINT(-79, -38), 'distance=2 unit=MILE')='TRUE'
2	bar@abc.com	..	Model='Mustang' and Year > 1999 and Price < 20000
..	..	..	..

Table 3: CustomerRules rule class table with rule conditions and action preferences.

In a database schema, a rule class is represented as a relational table with an Expression column that stores the rule conditions (See Table 3). The event structure for the rule class is used as metadata for the expression column. Additionally, it has one or more columns to store the action preferences for each rule. The values stored in these columns are used to determine the appropriate action for each rule as shown in the following implementation of the action callback procedure.

```
PROCEDURE CustomerAlerts(car       Car4Sale,
                         rlm$rule  CustomerRules%ROWTYPE) IS
BEGIN
  SendCarInfo(rlm$rule.EmailAddress, car);
END;
```

The events (instances of Car4Sale object type) are submitted to the rule class for evaluation using a programmatic interface. This step processes the rule condition and invokes the action callback procedure for each matching rule.

```
BEGIN
  DBMS_RLMGR.PROCESS_RULES(
      rule_class_nm => 'CustomerRules',
      event_inst => AnyData.convertObject
                      (Car4Sale('Mustang', 19000, 25000, 2001,
                                POINT(-79, -37.9999))));
END;
```

5.1 Applications involving Composite Events

Composite events [CKAK94] are formed out of two or more simple (primitive) events using an event pattern language with correlation operators such as conjunction, disjunction, negation, and sequence. The simple or primitive events that constitute a composite event may occur in different applications and at different times.

The Rules Manager framework can be used to support rule applications dealing with composite events. In this case, the rule condition can be expressed to span multiple events that occur independent of each other. The action for a rule is executed when all the necessary events are processed. In this scenario, each rule condition acts as a state machine with possible intermediate states and the rule condition is considered true when the state machine reaches an accepting state. For example, a consumer may express interest in a car based on the car attributes as well as the vehicle history report that is independently obtained for each car. One such rule can be expressed using an XML and SQL based rule condition language as shown below.

```
ON Car4Sale(Id, Model, Mileage, Year, Price, VehLoc) car
   VehHist (CarId, Rental, FireDamage, InsuranceLoss ) hist
IF
   <condition>
     <and join="car.Id = hist.CarId">
       <object name="car">
          Model = 'Taurus' and Price < 15000 ...
       </object>
       <object name="hist">
          FireDamage = 'NO' and InsuranceLoss < 5000
       </object>
     </and>
   </condition>
THEN SendCarInfo('foo@abc.com', car);
```

For the above rule condition, when a Car4Sale event occurs, the first half of the rule condition is matched and the intermediate results are stored in the database. Later, when a VehHist event for the same car occurs, it matches the second half of the rule condition and satisfies the join condition with the Car4Sale event in the event history, resulting in the execution of the rule action.

The complete XML and SQL based rule condition language is capable of handling most common event correlations such as sequence of events, conjunction of events, disjunction of events, and negation of events. This enhanced language can be used to correlate multiple spatial or non-spatial events in a uniform manner. For additional information on the rule condition language and the event management policies that dictate the behavior of the events in the system see [OraA10g].

6 Conclusions

In this paper, we described a framework for managing spatial events in an Oracle database. This framework allows combining of spatial events with non-spatial events using ECA rule semantics. The conditions in the framework are indexed using expression filter and the underlying spatial indexes thereby scaling the system to millions of rules. This integrated solution using expression filter, spatial, and alerting components in Oracle extend the salient features of commercial databases such as backup and recovery, concurrency, and replication to location-based service applications.

References

[CKAK94] S. Chakravarthy, S. V. Krishnaprasad, E. Anwar, and S. K. Kim. Composite Events for Active Databases: Semantics, Contexts and Detection. In *20th International Conference on Very Large Databases Conference*, 1994.

[Han92] E. Hanson. Rule Condition testing and action execution in Ariel. In SIGMOD Conference, pages 49-58, 1992.

[HB04] M. Horhammer and P. Biswas. A Rule Engine for Location Based Content Syndication. In *IEEE Intl. Conf. on Mobile Data Management*, 2004.

[HJ04] X. Huang, and C.S. Jensen. Towards A Streams-Based Framework for Defining Location-Based Queries. In *STDBM*, pages 73–80, 2004.

[J04] C.S. Jensen. Database Aspects of Location-Based Services. In *Location-Based Services*, pages 115–148, 2004.

[JC+04] C.S. Jensen, A.F. Christensen, T. B. Pedersen, D. Pfoser, S. Saltenis, and N. Tryfona. Location-Based Services – A Database Perspective. In *ScanGIS*, pages 59–68, 2001.

[KRA02] Ravikanth V. Kothuri, S. Ravada, and D. Abugov. Quadtree and R-trees in Oracle: A Comparison using GIS Data. In *ACM SIGMOD Intl. Conference on Management of Data*, pages 546–557, 2002.

[ON03] K. S. Ozgur and E. Nadia. RUBCES : A Rule Based Composite Event System. In *International XII. Turkish Symposium on Artificial Intelligence and Neural Networks*, 2003.

[OraA10g] Oracle 10g Application Developers's Guide - Rules Manager and Expression Filter, Release 10.2. *Oracle Corporation, Part# B14288-01*, December 2003.

[Ora10g] Oracle 10g Database User's Guide, Release 10.1. *Oracle Corporation*, December 2003.

[WC96] J. Widom, and S. Ceri. Active Database Systems. In *Morgan-Kaufmann*, ISBN 1-55860-304-2, 1996.

[YSG03] A. Yalamanchi, J. Srinivasan, and D. Gawlick. Managing Expressions as Data in Relational Database Systems. In *CIDR*, pages 303–313, 2003.