Filtering Features for a Composite Event Definition Language

Susan D. Urban, Ingrid Biswas, Suzanne W. Dietrich
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-8809
susan.urban@asu.edu, ibiswas@asu.edu

Abstract

This research has enhanced a distributed, rule-based application integration environment with a composite event definition language (CEDL) and detection system. CEDL builds on existing composite event operators and selection modes, adding features to support the filtering of primitive and composite events. The filtering features include basic parameter filtering on primitive and composite events, aggregate and quantifier filters on cumulative event parameters, and time filters for defining the lifetime of the composite event detection process. This paper presents examples of CEDL, illustrating the expression of application-oriented events through the aggregation and correlation of distributed events.

1. Introduction

Composite events are increasingly being used as a communication mechanism to achieve enterprise application and business-to-business integration. As an example, a credit card company may need to keep track of the customers who don’t make payments on monthly statements. The company can monitor the total amount of consecutive non-payments of a customer on continuous late payments and take appropriate action against the customers’ credit rating. A banking application could restrict the number of automatic teller machine (ATM) withdrawals and total amount on the withdrawals that can be done over the period of a working day. In an online shopping application, a company may want to be notified if the collective purchases of a customer are over a certain amount within a given period of time so that it can offer free shipping for further purchases. A common theme among all of these examples is that the generation of composite events often involves monitoring related events, such as late payments by the same customer or aggregate ATM withdrawals with an accumulated total from the same account.

The primary objective of this research has been to investigate the development of a composite event specification language and processing environment with filtering capabilities for the Integration Rules (IRules) [3, 8, 9] project. The IRules project is a distributed integration environment that integrates software black-box components using active rules known as integration rules. IRules events provide the means for independent, distributed components to communicate with each other through the triggering of integration rules that invoke services provided by distributed components or global application transactions.

The event processing capabilities of the IRules environment were originally developed in [6, 10], which included a language for the specification of primitive events, event generating capabilities for distributed components, event synchronization capabilities for synchronizing the execution of events, rules, and transactions, and an event handler for communicating the occurrence of primitive events to the integration rule processor. This research has enhanced the event processing capabilities of the IRules environment with the Composite Event Definition Language (CEDL) and corresponding composite event detection environment [1]. CEDL was developed by adopting existing event algebra operators and selection modes from past research [2, 4, 5] on composite events. The unique aspect of CEDL is the support it provides for filtering of primitive events as well as filtering of composite events and their associated aggregate values and timelines. The filtering capabilities are based on features initially explored in [11]. Filtering of events reduces the rule-processing load on the IRules rule manager by checking conditions on event parameters before rules are triggered. The primary advantage, however, is that filtered composite events enhance integration activities with a more meaningful approach to the expression of the types of complex, application-oriented events that are needed in the construction of distributed, event-driven applications.

2. IRules Primitive Events

The IRules environment supports the definition of several different type of events. The event type that is the most relevant to this research is the method event, which represents a call to a method of a component. Event modifiers can be used with method events to specify if the event is raised before or after the execution of the method. An example of the specification of a primitive
The event handler will wait for 1 day after the occurrence of the complete event before it discards the event instance. OR events are specified in a manner similar to AND events except that 1) parameter filters are not supported with an OR event since the system is not aware of the event that will trigger the composite event until runtime, and 2) time filters are also not used with OR events, since the first event that occurs triggers the OR event after satisfying the conditions.

4. CEDL Filtering Features

This section describes the filtering features of CEDL in the context of composite events. For simplicity, all examples assume the use of the latest selection mode.

4.1. Filters for AND, OR, and SEQ

Figure 2 illustrates the most basic form of the AND operator. The composite keyword is used to specify the start of the composite event specification. The name freeShipping is the name of the event, while loginName is the event parameter value that will be returned with the instance of the composite event. The event freeShipping is triggered to offer free shipping to customers on their next order for those who have placed two orders, indicated by a conjunction of different completeOrder events for the same customer within a one day period, with order amounts more than $99 each.

In Figure 2, there is an implicit parameter filter on the loginName of the events, indicated by the use of the same name (i.e., loginName) for the first parameter of each completeOrder event. The implicit parameter filter implies that the orders are associated with the same customer. The time filter, within 1 day, adds a restriction on the amount of time the composite event will remain active in the system. The event handler will wait for 1 day after the occurrence of the first event, before it discards the event instance. OR events are specified in a manner similar to AND events except that 1) parameter filters are not supported with an OR event since the system is not aware of the event that will trigger the composite event until runtime, and 2) time filters are also not used with OR events, since the first event that occurs triggers the OR event after satisfying the conditions.

```java
{completeOrder(String loginName, String orderId1, float amount1) AND
completeOrder(String loginName, String orderId2, float amount2)
where orderId1 != orderId2 and
    amount1 >= 99 and amount2 >= 99
within 1 day;
```

**Figure 2. An AND Event with Parameter and Time Filters**
conditions in the parameter filter.

Figure 3 is an example of a SEQ event. This event is defined to complete an order for a customer. When the afterCheckOut event occurs for the same loginName and orderId as the afterCheckout event, the composite event handler will test the parameter filter and trigger the completeOrder event, as long as the afterCheckCredit event occurs within 3 hours of the occurrence of the afterCheckOut event.

composite completeOrder(loginName, orderId, amount) 
(afterCheckout(String loginName, String orderId, float amount, String status) 
where amount > 0 and status == "OK"
within 3 hours;)

Figure 3. A SEQ Event with Parameter and Time Filters

4.2. Basic Use of TIMES

In Figure 4, a TIMES event updateCustomerHistory is defined that listens for the occurrence of afterCancelOrder events two times within the time period of eight weeks. This event can be used to monitor the shopping habits of customers. A '*' can be specified instead of the constant 2 to listen for an unlimited number of occurrences of a particular event in the given time span. The composite event that is listening for '*' event occurrences is triggered at the end of the time period defined in the time filter. The loginName in the for clause is the key for the TIMES event. The key implies that for all occurrences of the events that are being consumed, the key values are the same for all the event occurrences. In the case of a TIMES composite event, the output parameters that can be sent with the composite event are a projection of the key parameter values specified on the event.

composite updateCustomerHistory(loginName) 
(TIMES(afterCancelOrder(String loginName, String orderId, float amount), 2) 
for loginName 
within 8 weeks ;)

Figure 4. A TIMES Event with a Time Filter

4.3. Parameter Filters for the TIMES Event

An example of the use of the TIMES indexed parameter filter is given in Figure 5. As seen in the example, amount(1) is an event parameter that represents the amount value in the first event instance of the composite event. Similarly, amount(2) represents the amount value in the second event instance. The example shows that the individual event parameters can be compared to other event parameters as well as to constants. In the example given in Figure 5, the parameter filter checks that orderId(1) is not equal to orderId(2) and that the value of amount(1) and amount(2) is more than $100.

composite updateCustomerHistory(loginName) 
(TIMES(afterCancelOrder(String loginName, String orderId, float amount), 2) 
for loginName 
where orderId(1) != orderId(2) 
and amount(1) > 100 and amount(2) > 100 
within 8 weeks ;)

Figure 5. A TIMES event with Parameter Filter

The second type of filter provided to the TIMES event is an aggregate filter that can be applied on the event parameters. An example of an aggregate filter is given in Figure 6, where the sum function is used to determine if the sum of the cancelled orders is greater than $3000. The various aggregate functions supported by CEDL are sum, count, min, max, and avg, where each function provides the obvious meaning. All of the aggregate functions are applied to arithmetic values, except for the count function, which can be applied to any parameter.

composite updateCustomerHistory(loginName) 
(TIMES(afterCancelOrder(String loginName, String orderId, float amount), 2) 
for loginName 
where sum(amount) > 3000 and orderId(1) != orderId(2) 
within 8 weeks ;)

Figure 6. A TIMES Event with Aggregate Filter

The third type of TIMES filter is the quantifier filter that is applied to the cumulative parameter values. The system provides both universal and existential quantification. An example of the universal quantifier filter is given in Figure 7. In this example the filter determines if the amount for each order is greater than $100.

composite updateCustomerHistory(loginName) 
(TIMES(afterCancelOrder(String loginName, String orderId, float amount), 2) 
for loginName 
where amount(1) != amount(2) 
and for all a in amount: a >= 100 
within 8 weeks ;)

Figure 7. Universal Quantifier Filter in a TIMES Event

4.4. Use of Nested Composite Events

Composite events can be composed in a nested fashion to create more complex composite events. Figure 8 illustrates a complex composite event that is created as a result of nesting other composite and/or primitive events. The nested composite event is monitoring possible nuisance shoppers who place orders and then either return items, register complaints on the items purchased, or
cancel the order within a period of twelve weeks. The possibleNuisanceShopper composite event is defined as a SEQ event, with completeOrder followed by a disjunction of three different events (i.e., a TIMES event on returnItems, a TIMES event on registerComplaint, or a cancelOrder event). In this example, there is an implicit filter on the customer’s loginName and orderId to ensure that all events are associated with the same customer and order.

Figure 9 shows an example of detecting multiple instances of a complex composite event, where nuisanceShopper is defined on multiple occurrences of the composite event possibleNuisanceShopper over a period of 24 weeks. The complex events in Figures 8 and 9 demonstrate the strength of the CEDL composite event operators together with the filtering capabilities for defining meaningful, application-oriented events.

6. References


