



Event Processing and Temporal Reasoning

Edson Tirelli and Mark Proctor
Red Hat
February 13th, 2008

[Complex / Stream] Event Processing

“Complex Event Processing, or CEP, is primarily an event processing concept that deals with the task of **processing multiple events from an event cloud** with the goal of **identifying the meaningful** events within the event cloud. CEP employs techniques such as **detection** of complex patterns of many events, event **correlation** and **abstraction**, event hierarchies, and relationships between events such as causality, membership, and timing, and event-driven processes.”

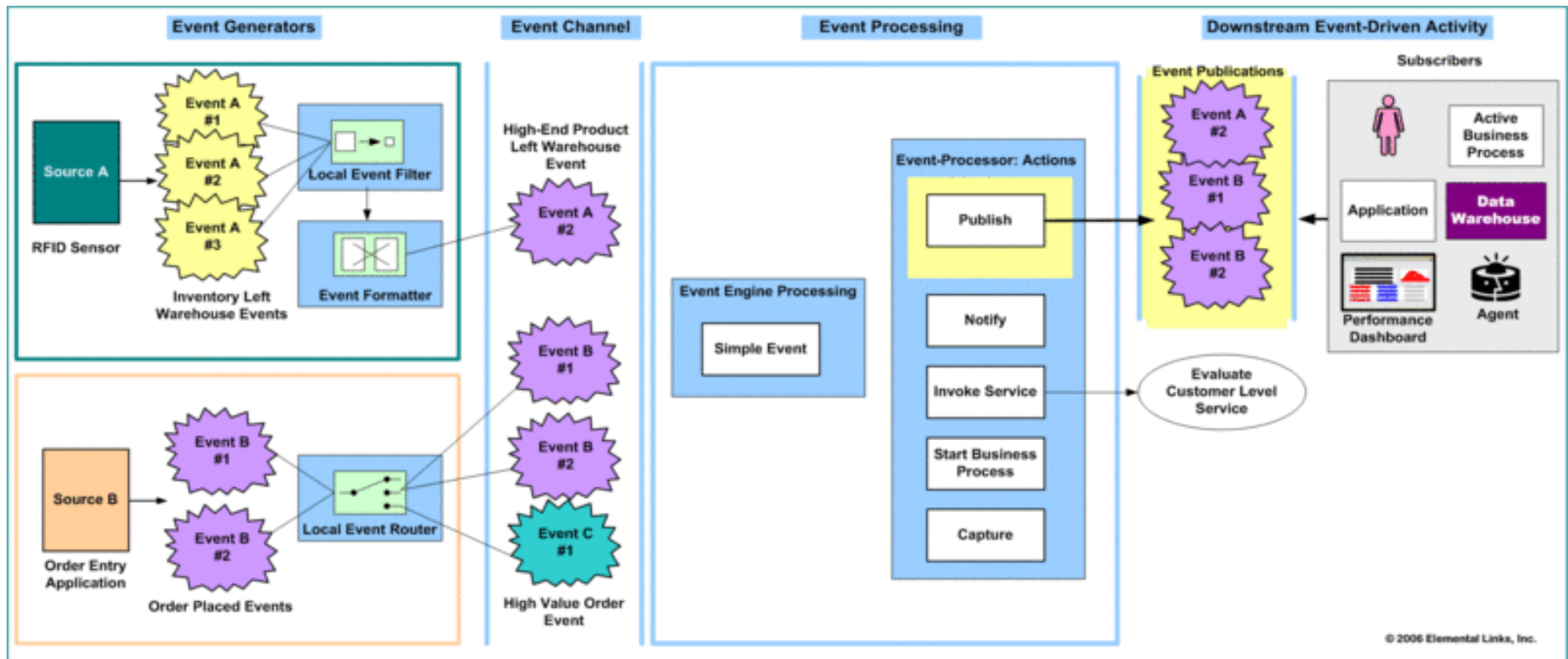
Wikipedia

[Complex / Stream] Event Processing

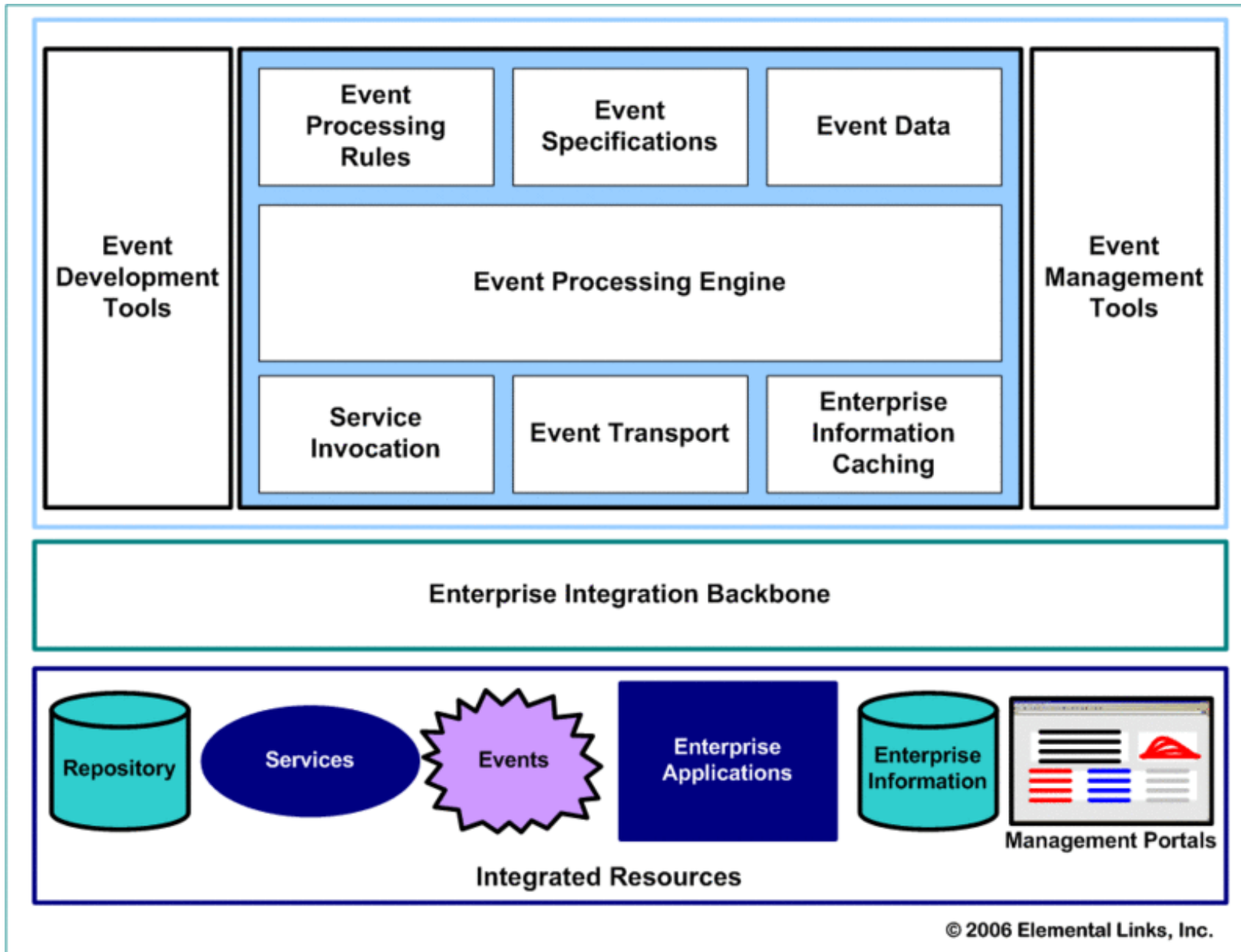
“Examples of events include **church bells ringing**, the appearance of a **man in a tuxedo** with a **woman in a flowing white gown**, and **rice flying through the air**. A complex event is what one infers from the simple events: a **wedding is happening**. ”

Wikipedia

[Complex / Stream] Event Processing



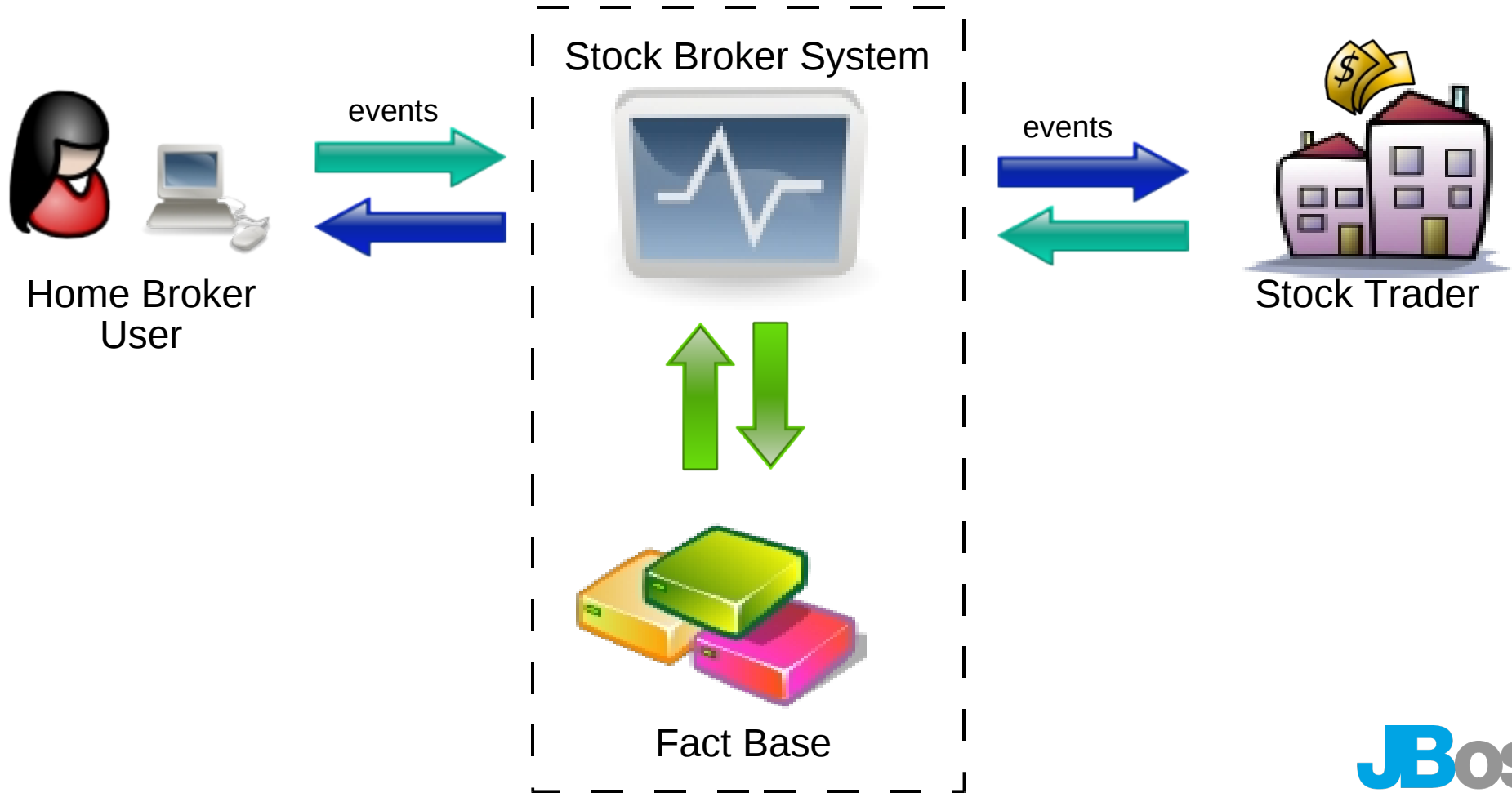
Event Driven Architecture



Requirements for Event Processing

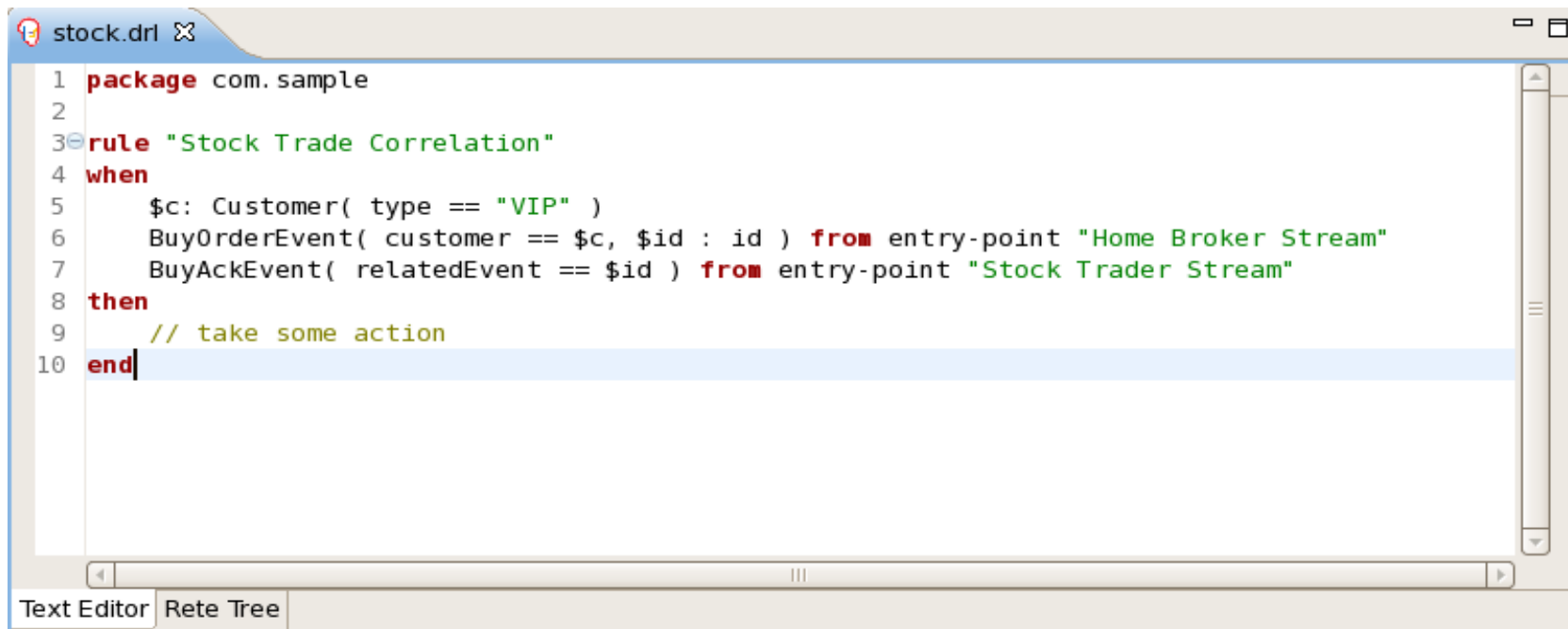
- Event Detection
 - From an event cloud, select all the meaningful events and only them.
- [Temporal] Event Correlation
 - Ability to correlate events and facts declaring temporal and non-temporal constraints between them.
 - Ability to reason over event aggregation
- Event Abstraction
 - Ability to declare composite complex events from simple atomic events

Use case: Stock Broker



Event Detection

- Business as usual for Rules Engines
 - Leverages all the power and expressiveness of the rules language
 - Extension for stream support: entry-points

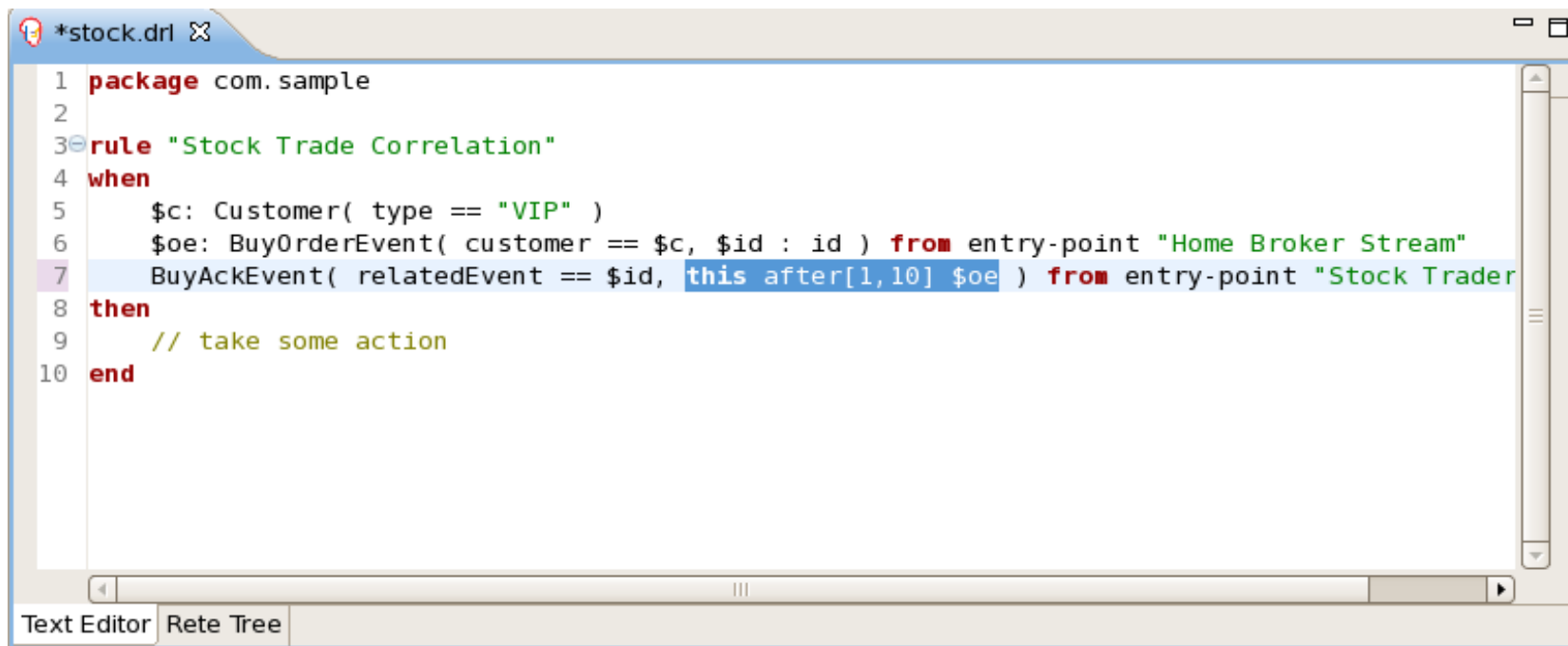


```
stock.drl x
1 package com.sample
2
3 rule "Stock Trade Correlation"
4 when
5     $c: Customer( type == "VIP" )
6     BuyOrderEvent( customer == $c, $id : id ) from entry-point "Home Broker Stream"
7     BuyAckEvent( relatedEvent == $id ) from entry-point "Stock Trader Stream"
8 then
9     // take some action
10 end
```

The screenshot shows a text editor window titled 'stock.drl'. The code is a Drools Rule Language (DRL) rule named 'Stock Trade Correlation'. It is located in the 'com.sample' package. The rule has a 'when' section with two conditions: a Customer object of type 'VIP' and two event objects, 'BuyOrderEvent' and 'BuyAckEvent', with specific entry-point constraints. The 'then' section contains a comment indicating where to take an action. The editor has a 'Text Editor' tab selected and a 'Rete Tree' tab visible at the bottom.

Temporal Event Correlation

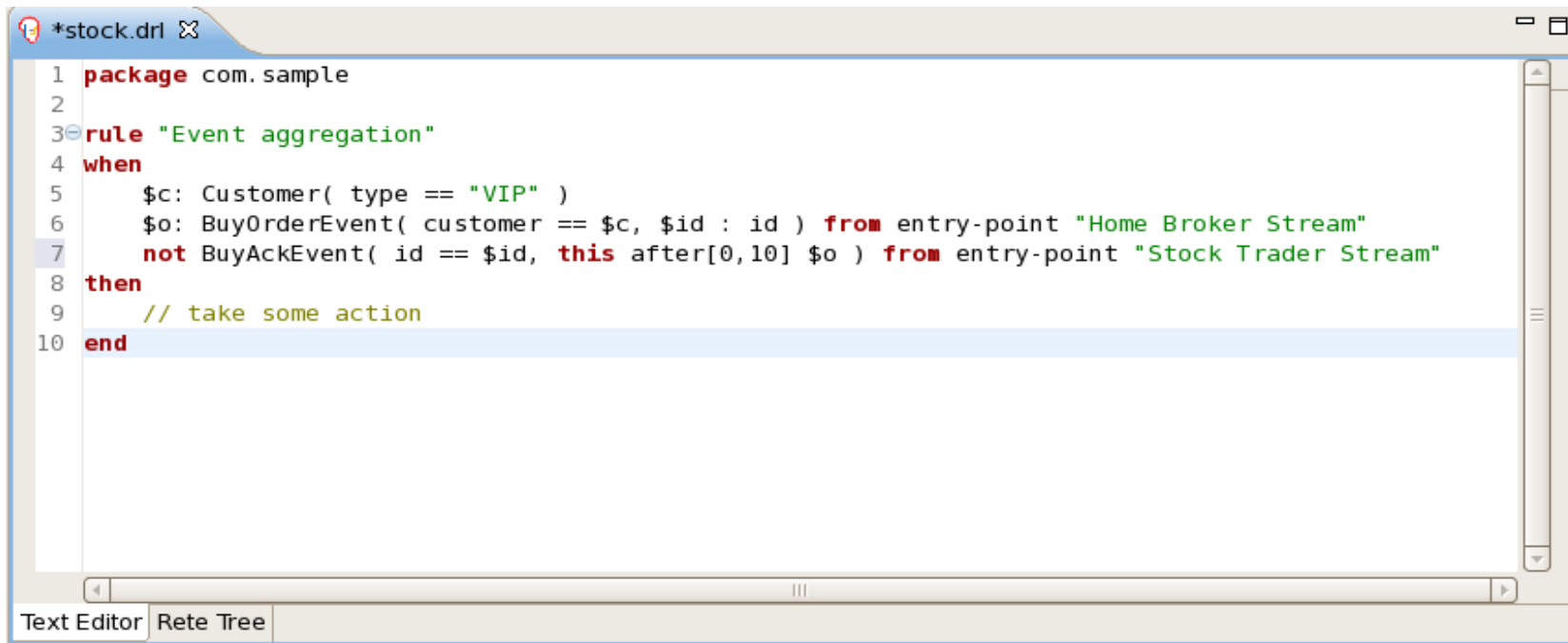
- Temporal extension to Rete Algorithm
 - Temporal operators: before, after, meets, met-by, overlaps, overlapped-by, during, contains, starts, started-by, finishes, finished-by, concurrent



```
*stock.drl
1 package com.sample
2
3 rule "Stock Trade Correlation"
4 when
5     $c: Customer( type == "VIP" )
6     $oe: BuyOrderEvent( customer == $c, $id : id ) from entry-point "Home Broker Stream"
7     BuyAckEvent( relatedEvent == $id, this after[1,10] $oe ) from entry-point "Stock Trader"
8 then
9     // take some action
10 end
```

Temporal Event Correlation

- Reasoning over absence of events
 - Implicit time-window management

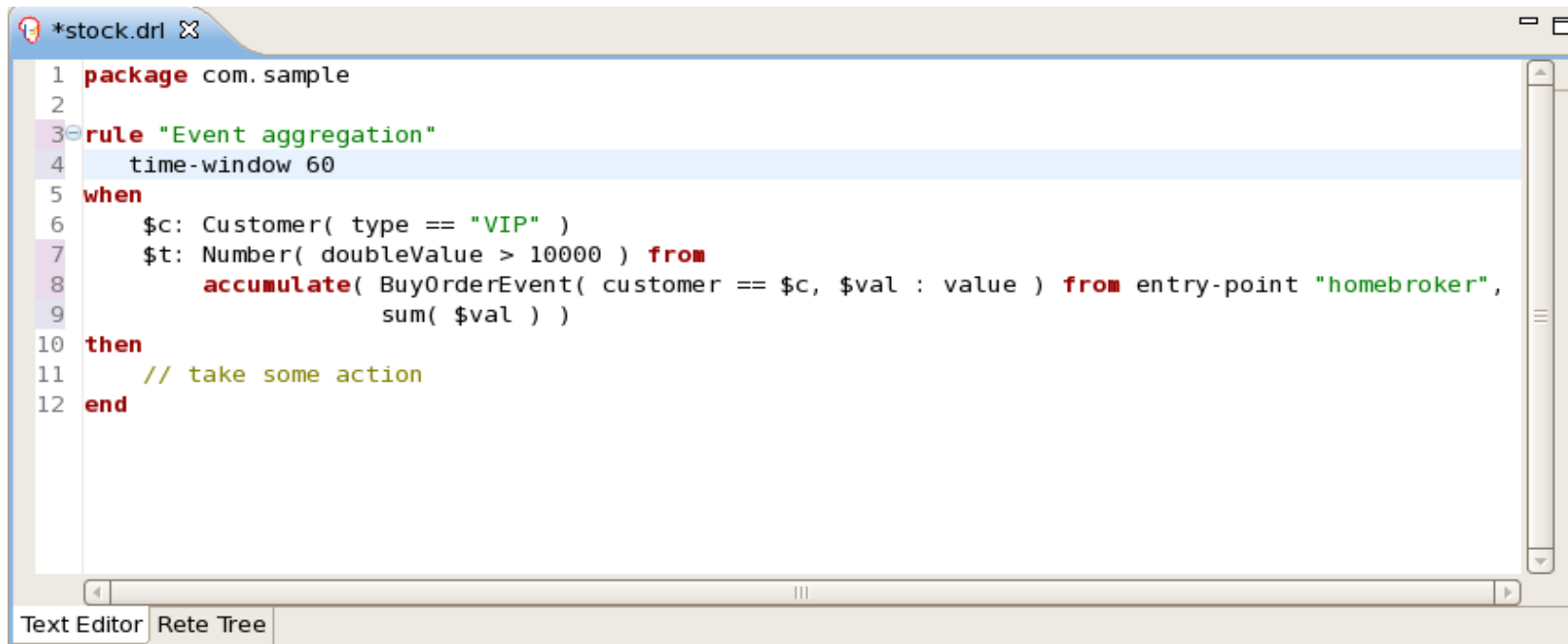


```
*stock.drl X
1 package com.sample
2
3 rule "Event aggregation"
4 when
5     $c: Customer( type == "VIP" )
6     $o: BuyOrderEvent( customer == $c, $id : id ) from entry-point "Home Broker Stream"
7     not BuyAckEvent( id == $id, this after[0,10] $o ) from entry-point "Stock Trader Stream"
8 then
9     // take some action
10 end
```

The screenshot shows a text editor window titled '*stock.drl X'. The code is a Drools rule named 'Event aggregation'. It defines a package 'com.sample' and a rule with a 'when' condition. The condition consists of three parts: a Customer object with type 'VIP', a BuyOrderEvent object from the 'Home Broker Stream' entry-point, and a 'not' condition for a BuyAckEvent object from the 'Stock Trader Stream' entry-point, which is only true if it has not occurred within a 10-unit time window after the BuyOrderEvent. The 'then' block contains a comment '// take some action'.

Event Aggregation (1)

- Sliding window support
 - Rule scoped and CE scoped windows
 - Time-based and event-based

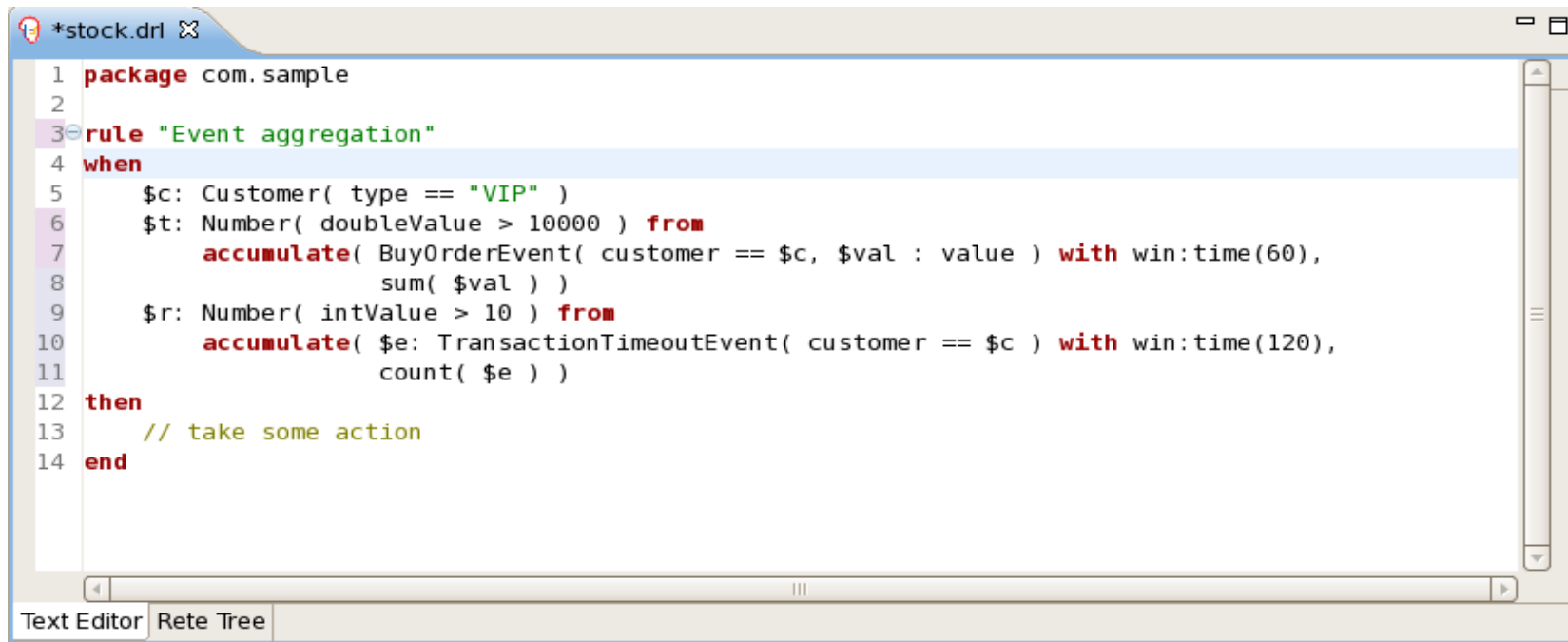


```
*stock.drl X
1 package com.sample
2
3 rule "Event aggregation"
4   time-window 60
5   when
6     $c: Customer( type == "VIP" )
7     $t: Number( doubleValue > 10000 ) from
8     accumulate( BuyOrderEvent( customer == $c, $val : value ) from entry-point "homebroker",
9               sum( $val ) )
10  then
11    // take some action
12  end
```

Text Editor | Rete Tree

Event Aggregation (2)

- Sliding window support
 - Rule scoped and CE scoped windows
 - Time-based and event-based



```
1 package com.sample
2
3 rule "Event aggregation"
4 when
5     $c: Customer( type == "VIP" )
6     $t: Number( doubleValue > 10000 ) from
7         accumulate( BuyOrderEvent( customer == $c, $val : value ) with win:time(60),
8             sum( $val ) )
9     $r: Number( intValue > 10 ) from
10         accumulate( $e: TransactionTimeoutEvent( customer == $c ) with win:time(120),
11             count( $e ) )
12 then
13     // take some action
14 end
```

The screenshot shows a text editor window titled '*stock.drl' with a scroll bar on the right. The code is a Drools rule named 'Event aggregation'. It starts with a package declaration 'com.sample'. The rule has a 'when' section with three conditions: a Customer object of type 'VIP', a Number object with a doubleValue greater than 10000, and a Number object with an intValue greater than 10. The first two conditions are connected by 'and' and the third by 'and'. The first condition is followed by a 'from' clause with an 'accumulate' function. The 'accumulate' function takes a 'BuyOrderEvent' object with 'customer == \$c' and 'value' as a parameter, and a 'with win:time(60)' clause. The function body is 'sum(\$val)'. The second condition is followed by a 'from' clause with an 'accumulate' function. The 'accumulate' function takes a 'TransactionTimeoutEvent' object with 'customer == \$c' as a parameter, and a 'with win:time(120)' clause. The function body is 'count(\$e)'. The 'then' section contains a single line of code: '// take some action'. The rule ends with 'end'. The editor has a 'Text Editor' tab and a 'Rete Tree' tab at the bottom.

Questions?

