SocEDA

Cloud based platform for large scale social aware EDA

ANR-10-SEGI-013

Document name:	State of the Art in CEP
Document version:	**V-1**
Task code:	
Deliverable code:	D4.1.1
WP Leader (organization):	OrangeLabs (FT)
Deliverable Leader (organization):	OrangeLabs
Authors (organizations):	OrangeLabs, ADAM, THALES
Date of first version:	May 2011

Table of Contents

1. Introduction

1.1. Purpose

The goal of SocEDA is to develop and validate an elastic and reliable federated SOA architecture for dynamic and complex event-driven interaction in large highly distributed and heterogeneous service systems. Such architecture will enable exchange of contextual information between heterogeneous services, providing the possibilities to optimize/personalize the execution of them, according to social network information. The main outcome will be a platform for event-driven interaction between services, that scales at the Internet level based on the proposed architecture and that addresses Quality of Service (QoS) requirements. The platform consists of:

1. Federated middleware layer: a peer-to-peer overlay network combined with a publish/subscribe mechanism, that has the task to collect events coming from the heterogeneous and distributed services,
2. Distributed complex event processor: an elastic, distributed computing cloud based engine for complex processing of events coming from different services in order to detect interesting situations that a service should react on,
3. Social aware event modeling and matching plus an event based workflow engine for filtering events and proposing adaptation and changes in running business processes and services,
4. A monitoring and governance framework as well as a Mashup frontend to describe and manage services as well as business events.

This document focuses on the Event Processing aspect as we will need a complex event processing engine to manipulate and understand incoming events. The purpose of this document is to give an overview of existing solutions that could fit our needs, based on criteria described in the section Technical criteria.
This State Of The Art will allow the selection of the best CEP solution according to the CEP requirements for the SocEDA platform.

1.1. Structure

After an introduction describing mainly the context and the objectives in Section 1 and 2, Section 3 contains the description of the methodology and methods used to select the tools as well as the criteria to evaluate them. Section 4 contains the list of tools and their detailed study while Section 5 lists the requirements from the SocEDA Platform point of view and Section 6 is the conclusion.

2. Context

2.1. Objectives

SocEDA project aim is to provide an open distributed platform for event-driven interaction between services that scales at the Internet level. To achieve this goal, a federated architecture will be deployed to address the multiplicity and the heterogeneity of service networks. On top of this platform, a tool will be offered to CEP advanced users but also to business users in order to:

- design, deploy and run complex and distributed EPL Statements in a DCEP2.5] ,
- define complex event patterns,
- enrich existing events.

The idea would be to simplify the CEP application definition. We may provide a graphical tool which will facilitate the definition. In order to make the event manipulation easier the graphical aspect will be an important requirement.
In this perspective, this document will list available software achieving CEP programming and authoring, including design time, deploy time and runtime. This document reports a selective evaluation about a subset of those CEP software, in order to:

- Picture the overall level of development of graphical CEP composition solutions, the functionalities offered, their maturity and the tools available for non developer users,
- Frame and structure, through selective criteria, the evaluation of the listed software,
- Refine requirements for the CEP design tool that is expected in the SocEDA platform,
- Determine if one of evaluated software could be embedded in the platform,
- Determine the work to enhance it and fulfill the whole platform needs, or if the expected solution should be implemented.

2.2. Context and definitions

References about CEP usually mention *rule-oriented language* as well as *stream-oriented language*. They both concern the event processing software environment and nothing related to Business Rule Management solutions. In general, a stream-based CEP will be based on a SQL like language used to query the incoming stream of events while a rule-based CEP engine will be based on a language used to trigger state transition or rules according to the incoming event type. As far as we have seen the stream-oriented languages are the most used to express complex event processing logic. Those languages are often compared to SQL which is very handy for the developers. So, we will mainly focus on those stream-oriented solutions. Before starting any analysis it is mandatory to establish a common language in order to avoid any confusion. This section is all about a quick introduction to the vocabulary used in this document.

Events: An Event is an action or message, coming from an external event source. An Event has an event type and it is composed of data properties and their associated values. An Event is equivalent to an event instance of an event type.

Event Types: Event type could be presented as the equivalent of an event class or event definition or event schema/model. Event types are usually described through computer language such like XML Schema or Java POJO. They are composed of attributes such like a unique identifier, timestamps and other attributes specific to the event type. Those attributes also mentioned as properties can be simple but also complex data type.

Event Abstraction: An event can be composed by a set of events which is basically an abstraction of those events. This simple event composition turns the event on a complex event which is an abstraction of those simple events.

Event Relationship: Various relations exist between events, one obvious is the time but it can also be space causality and of course abstraction as it has been explained in the previous definition. If you have a look on complex event environment you may find the “poset” term, which is a reference to partial ordered set of events (time relation).

Event Channel: The event channel is the communication layer transporting the information from event sources (emitters) to event sinks (consumers). It could be a TCP socket, HTTP or any other communication protocol (JMS). A channel can carry events of multiple types. A single event channel may be consumed by multiple event consumers.

Event stream: Event stream also called event feed is an ordered sequence of events composed of events instance from various event types, which can come from multiple sources. This stream can be bounded, by criteria such like the time, thanks to the time window aspect of complex event processing engine.

Complex Event Processing (CEP): The complex event processing term is used to talk about an engine that can read, create, transform, abstract, and process many events bringing intelligence and take real time decision or action.

Event Processing Language (EPL): Different languages styles are available in the CEP world. One can broadly classify them in the following four slots:

- Logic programming: example Etalis (ELE/EP-SPARQL),
- Rule oriented: examples Drools Fusion, Tibco BusinessEvents,
- Stream Oriented: examples Esper (EPL oriented), Aleri , Streambase,
- Programming language oriented: example Progress Software (Apama MonitorScript [1]).

Event Processing Networks (EPN): The Event Processing Network describes the network architecture around the CEP engine. It is composed of many components:

- Event sources, also called event emitter or producer. Anything that can send an event such like sensors,
- Event sinks, are on the other side the event receiver or consumer such like a database or an actuator,
- Event channels are the channel connection or bus used to transmit events from events sources to event sinks. Of course, the same channel can conduct multiple events from various sources.
- Event Processing Agents described below.

Event Processing Agents (EPA) (Event processing component, Event processing mediator): Event Processing Agents are basic software blocks implementing functionality like filtering, transformation, composition, enrichment. The combination of these different basic blocks leads to the architecture of a complex EPN which may be centralized or distributed.

Event processing: Event processing is the name for the broad set of technologies that perform operations on events, including modifying, creating and destroying events, and complex event processing.

Event Template: An event template matches single events by replacing the variables with values.

Event pattern: An event pattern matches sets of related events by replacing variables with values.

Event Sink (Event consumer): An event sink is an entity that receives events.

Event Source (Event emitter or Event producer): An event source is an entity that sends events.

Event-Driven Architecture (EDA): EDA is a type of software architecture in which some of the components are event driven and communicate by means of events.

Event Stream Processing (ESP): Term originally used in academia that refers to database-oriented techniques of processing streams that assumed events arrived in order into the stream processing computing engine. As such, ESP is considered a subset of CEP, which does not assume that events arrive in order.

Complex Event: An event that is an abstraction of other events called its members.

Date Stream Management System (DSMS): Data Stream management is a relatively new form of database that allows streams to be stored and replayed in real time, and in the order in which they are detected.

Stream: The word stream is typically used (e.g stream processing, event stream processing, stream processing engine) because events that are processed in real-time tend to arrive in stream.

Constraint (Event pattern constraint): A Boolean condition that must be satisfied by the events observed in a system.

Rule (in event processing): A rule is a prescribed method for processing events.

Window: A window is a bounded portion of an event stream.

2.3. Technical principles

The figure below shows the components involved in basic Event Driven Architecture.

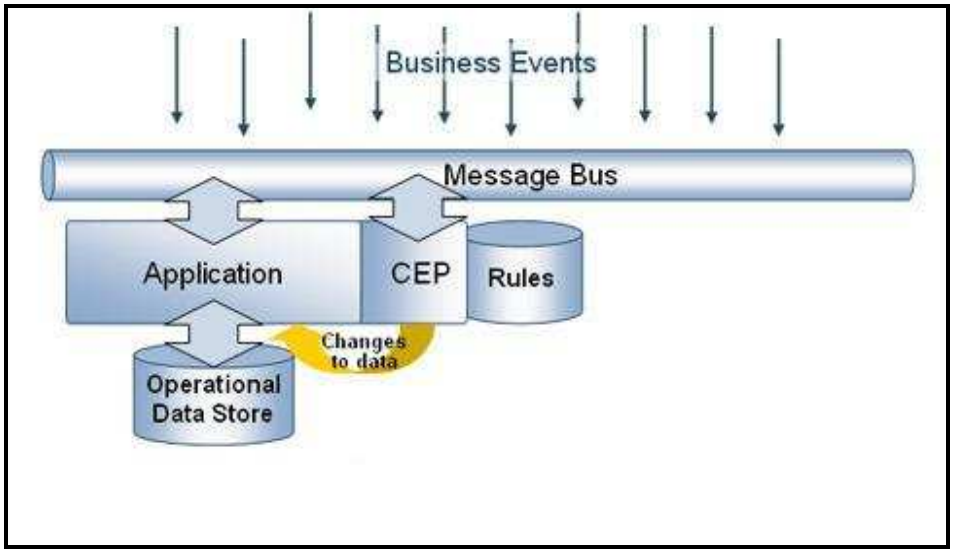


Figure 1 Event driven architecture components

The business events are data taking many forms. It may be anything from sensor information to stock options values. The CEP relies on defined rules to treat many incoming events. Those events are sent to the CEP through a message bus. The CEP engine is set with a matching rule-action, meaning the CEP will trigger an action as soon as a rule matches the simple or complex incoming events. The CEP engine interprets all incoming events and can infer complex events from a combination of simple events. The CEP engine relies on a number of techniques, such as:

- Event-pattern detection,
- Event abstraction,
- Detecting relationships (such as causality, membership or timing) between events.

2.4. CEP positioning

The Event Driven Architecture (EDA) seems being the answer to the increasing volume of event flows coming from external sources. The principle of event driven processing is turning as the model to answer this fast moving data. That way organizations can understand events but also event patterns identifying critical threats, new conditions, opportunities that could have an impact on their activities.
CEP solutions enable to consolidate information from various data sources, gathering intelligence and alerting appropriate parties when events of interest occur. Basically, it gives sense of data to which it might concern when it might be concerned.

2.4.1. Perimeter

The main characteristic of a CEP engine is the capability to consume a huge volume of data flow bringing intelligence and meaning from defined rules. The rule definition aspect could be CEP queries manually inserted within the code or it could be done through an intuitive interface. The question of how easy the rule insertion is will be addressed through the tools review in the following chapters.

2.4.2. Standardization

At the moment there is no real standard in the CEP world, but most of the CEP engines are query-based using a derivative of SQL as their query language. Those EPL languages, similar to SQL, make it understandable for most of the developers. Even if they almost all look like SQL there is no standard on that aspect. In terms of event type or event model there is definitely a need for a cross enterprise understanding of events or complex events. Basically a common event specification would help.

2.5. Acronyms

Acronym	Definition
CEP	Complex Event Processing
DCEP	Distributed Complex Event Processing
EDA	Event-Driven Architecture
ESB	Enterprise Service Bus
POJO	Plain Old Java Object
EPL	Event Processing Language
QoS	Quality of Service
ESP	Event Stream Processing
DSMS	Data Stream Management System
EPA	Event Processing Agent
EPN	Event Processing Network

3. Methodology

3.1. Approach

We place our context within the framework of an event based service oriented platform for companies. The platform offers a CEP editor tool, allowing a non developer user to enrich event or enrich the CEP engine with new rules. Basically, the CEP editor should rely on an event repository offering the possibility to enrich event and create complex event. This same CEP editor should also make the authoring easier. In order to fulfill our requirements, we are looking for a tool with a user-friendly interface, capable of editing simple Events to create complex Events and enrich the event repository. The environment should allow to design, configure or edit simple and more complex rules for the triggering of new event interaction with services components.

The CEP editor tool is part of the platform, so it should offer interfaces with the workflow engine. As we can't evaluate the complete set of tool in the market, and as we're looking as well for graphical capabilities than complex processing abilities, we chose to first list a large number of tools and sorted them according to their interesting features. Then, for each type, we'll choose one or more of the listed tools, and proceed to evaluate them, following the criteria described in the chapter below[3.3]. Our selection of CEP tools has been based on our knowledge of the current tools available, and when we had no practical experience, on literature or editors descriptions and show cases. We took into account different criteria, including software maturity and authoring feature's availability.

Below is the list of tools we considered containing closed source as well as open source products. Of course, the open sources ones will be favored, though we don't reduce our scope to those, as we're looking for powerful functionalities, on a market that is still not mature:

- *Esper* from EsperTech [3],
- *StreamBase* from Streambase Systems[2],
- *Drools* from Jboss community[4],
- *Aleri* from Sybase[5].

Finally, to make this report as effective as possible, the following simple SMS arrival Scenario extracted from a whole Taxi use-case will be taken as the recurrent theme for the evaluation of the different tools. It will be implemented (or part of it) as a CEP application running in the CEP component.

3.2. Use Case Scenario

Abstract

The case study describes the Smart Taxi Application system which is designed to manage normal situations (i.e. SMS arrival and SMS proximity scenario) as well as the handling of special cases (i.e. arrival of unexpected events or Traffic Info Events). Such special cases include several different actions, such as the adaptation of the process flow as well as the management of Operator intervention. The actors involved in this case study are Customers, Drivers and Taxi Operator.

SMS Arrival Scenario description (short)

A Call or a SMS is sent to a Taxi company number (which is supplied by Orange - Taxi Application), by a customer expecting a taxi. Based on the phone number, a location request is sent. As a result, a location event (X, Y) is received, associated with the customer request. In parallel, Taxi Availability Events, IMS presence Events, and Taxi Location are sent to Taxi application, producing a list of available taxis and their Location. From those information, using the GIS partner services to compute and compare the Customer-Taxis Path, the proper Taxi can be chosen (the available one, whose path is shortest to the Customer) and the proper Path (the shortest, where no TAXI Event has occurred) the taxi should use. Then, an SMS is sent to the Customer to inform him about the Taxi, a MMS is sent to the Taxi with the map & Path to pick up the Customer, and the references of this treatment are stored.

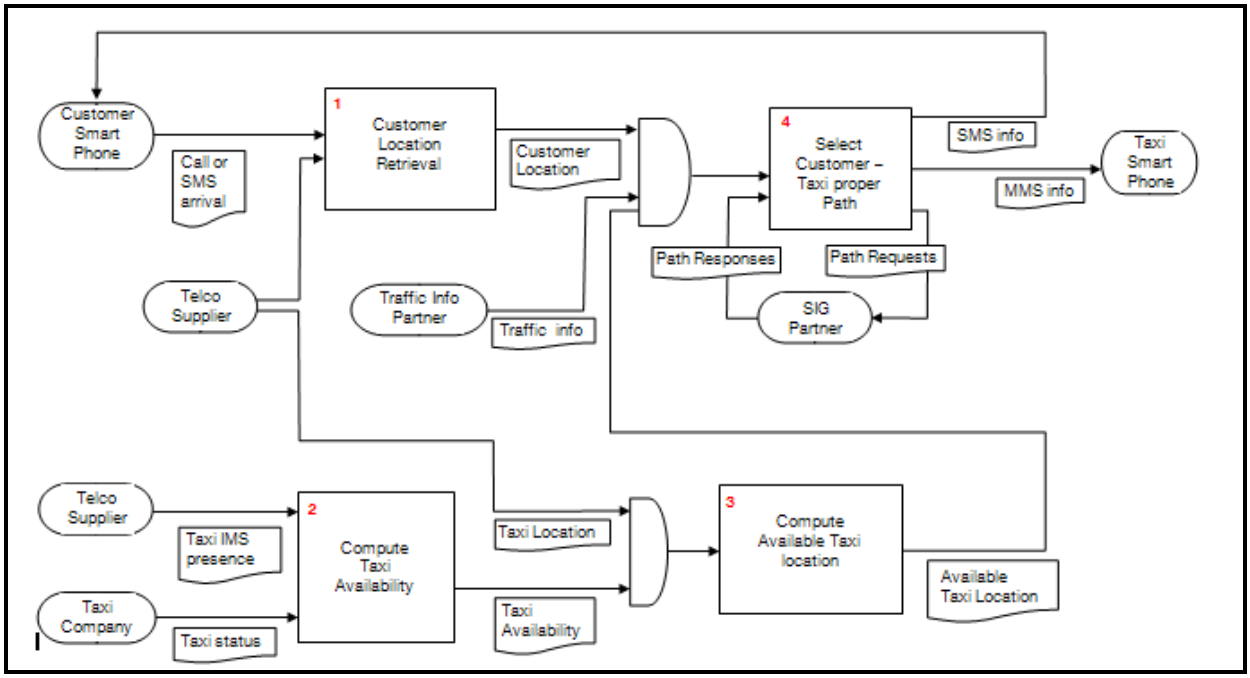


Figure 2 Scenario SMS arrival Logical view

3.3. Relevant criteria

This detailed comparison will be based on general and specific criteria to assess technical and non technical aspects of our preselected solutions. The most important points regarding CEP tools are the following ones:

3.3.1. Runtime Architecture, Deployment

Concerning the runtime architecture the issues are:

- The security and scalability aspects are pretty important. The security features within the different tools have to be tested but if security is a major concern scalability regarding event suppliers and consumers for example is mandatory,
- The deployment aspect will also be part of the evaluation because we want to keep it as simple as possible. The idea would be to deploy an application in few steps,
- Any application will be relying on the CEP engine capability of processing incoming events, so the solution has to be very reliable. We will be looking for a very stable solution in order to ensure a quality of service (QoS),
- Finally, we will be looking for a manageable solution through an administration tool allowing the control of deployed application. The monitoring aspect is also important as we need to know the status of each deployed application. Performance and logging aspect would also be very useful.

3.3.2. Event processing features

The event processing engines are reachable through different protocols and format depending on the provided adaptors. This key point is important because it will determine which protocol and which format can be used to send events to the CEP engine. So, we will check the supported events format in our study. We are also interested by a handy solution with easy and fast communication protocol integration. Basically, the engine will have to embed some implemented standard adaptor such as socket or http transport protocols.
An interesting aspect is the ability of filtering events. The CEP engine would have to support pattern matching but also time-based conditional logic with events retention. Basically, all concepts associated to CEP should be considered. This aspect will define the ability of setting intelligence within the application relying on this CEP engine. Another important issue is that the CEP engine will have to be able to manage multiple event streams in order to allow event cross correlation. This feature will allow the creation of new complex events from multiple simple or complex events.
We will also check the query language aspect to evaluate the power or limits of the query language. This query language will border the ability of adding intelligence to the applications.
Another aspect related to the intelligence ability is the possibility of adding new functionality from external function to the CEP. The existing language already offers some intelligence but it would be useful if we can extend those features with external implemented functions.

The event format is an important issue because it will be strongly related with the interoperability. We will have to check this feature within the tested solution to define the event format used to communicate with the CEP engine. Finally, a combination of events and stored data would be an interesting aspect, so we will check if we can mix database data with incoming events.
Regarding adaptors, even if the tested tool doesn't bring every adaptor already implemented, we have to check if it offers the ability of implementing them. An important issue is to have the possibility to implement custom adaptors to customize the CEP engine for our needs. If the community is wide enough, the adaptor could also have been already implemented by some community group, or at least a piece of it.
And last point, it would be very handy to get a dashboard or at least any visualization tool, for a debugging aspect. So, we will check if the whole platform or engine comes with a visualization tool, such like dashboard or web portal.

3.4. Technical criteria

3.4.1. Event Processing features

Event processing features mean features related to the associated language. Basically we are looking for a CEP engine with:

- time notions (time, length, time-ordering),
- event retention possibilities,
- grouping, aggregation, sorting, filtering and merging event stream,
- subqueries possibility,
- update, insert and delete operations,
- event pattern matching,
- input and output adapters.

3.4.2. CEP Tooling around the CEP (SDK, Editors)

The CEP engine could be interfaced through a complete IDE embedding testing and debugging tool or it could be a simple library offering an API. Of course the best solution would have a user interface to manage the engine and debug the applications. In order to feed the CEP engine with new event type and rules it would be easier to do it through an embedded editor.

Another convenient tool that we would like to have is a testing/simulation tool or environment where we can run rules and check how the engine reacts on incoming events. We will select a tool offering a rich and extendable SDK, as we need to offer many services. In the best case, the final solution would also embed an editor allowing to setup and create CEP applications. A simulation, test and debugging environment would be a must because it will really makes easier CEP application authoring.

3.4.3. Runtime Architecture, Deployment, Admin tools

The previous criteria are oriented towards *authoring* but we would also be interested by deployment aspects through a handy user interface. The deployment aspect is quite important as we want to allow a business user to deploy his application on a distributed CEP environment but keeping it transparent for the final user.

3.5. Product Environment criteria

3.5.1. Standards and interoperability

The interoperability aspect is a critical one as the platform will be interacting with event suppliers. We have to check the API richness. As we want to get a platform for CEP programming mixed with business process management, we will check the BPM features of the different tested tools. It would be a good point if the tested solutions can be based on standards for any future interoperability.

3.5.2. Product strategy

We are talking about quite new technologies, so the strategic aspects are definitely important to ensure the enhancement of the product. We will be checking roadmap but also support and community aspects. Of course, we would like to have an open source solution in order to allow any customization or extension. So, the license point is important as well. That doesn't mean we will be rejecting any closed source solution but between two equivalent solutions we will tend towards the open source solution.

3.5.3. Roadmap

This criterion evaluates the maturity and evolution planned for the next version. This is also a sort of warranty of a certain activity around the solution for the next years.

3.5.4. Licensing

This important criterion influences the choice. The tool's license, from open source to closed source will be an important criterion. We have clearly a preference for free open source solution but we will stay open to solutions under commercial licenses.

3.5.5. Community

This criterion evaluates the community support. This is relevant in various aspects, such as:

- Taking advantage of a fulfilled component library to leverage the ability of business application creation,
- Wealth of documentation (forums, blog, wiki) and community support,
- Somehow, a warranty of durability,
- A media for future dissemination.

4. CEP Tools

We will apply the previous criteria and the authoring-testing-running cycle for the SMS arrival scenario described just before to the following list of tools starting with Esper from the EsperTech Company, followed by Streambase from StreamBase Systems, ALERI from Sybase and the Drools Fusion from the Jboss Community. A complete list of products incremented with others interesting tools is available in Appendix[1]

4.1. Esper

4.1.1. Main features

The EsperTech company does present the core Esper features on the official website. The Esper engine provides the features mentioned in the technical criteria section. Two solutions are provided for two different kind of implementation, Esper for Java and *Nesper* for .Net. We will now test this whole free open-source solution (only the Java implementation) for the purpose of our State Of The Art.

Esper delivers CEP components and products from a basic engine (GPL license v2) to complex commercial offers named respectively *EsperEE* and *Esper HA*.

- Esper: this is the basic CEP Esper engine (GPL),
- EsperEE: this is the Enterprise Edition,
- EsperHA: this is Esper High Availability.

4.1.2. Big Picture

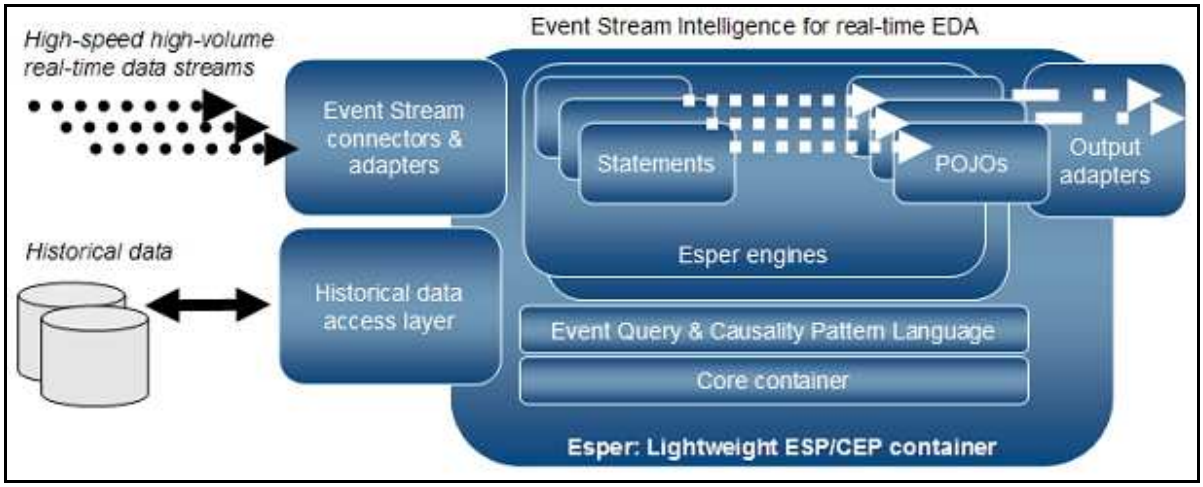


Figure 3 Esper CEP middleware component

As seen in the figure above the Esper engine can enrich any existing java application with Event Stream intelligence. As long as the java developer describes the future statement through POJO and build the event queries to bring to intelligence.

The Esper engine comes with a full set of input connectors to allow a maximum of interoperability as well as a full set of output connectors.

This lightweight container does also embed an historical data layer to eventually store events in order to mix passed events with incoming events.

4.1.3. Technical criteria

4.1.3.1. Event Processing features

The Core engine presented on the official website has almost all the features mentioned in the technical criteria 3.3 section. Basically Esper offers:

- An Event Processing Language (EPL) which is an optimized language for dealing with high frequency time-based event data
- Continuous queries, time and length windows, pattern matching, aggregation plus many other features

EsperEE offers some more features such like:

- Event type manager, Design statement tool, Module/statement manager
- UI application builder, Simulation/Debugging tool.

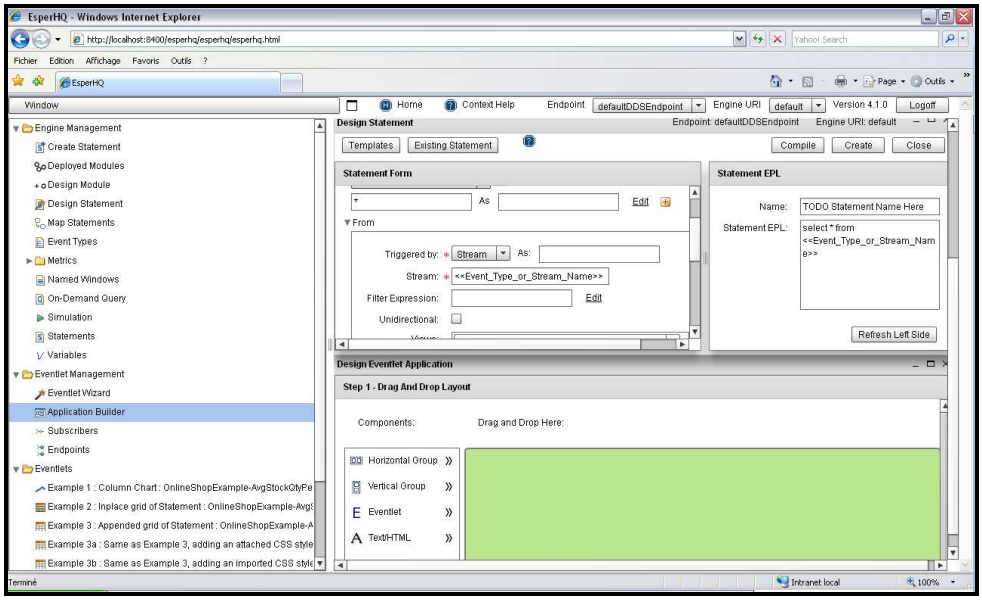


Figure 4 Rich web-based creation tools of the enterprise edition

4.1.3.2. CEP Tooling around the CEP (SDK, Editors)

Esper is proposed as a simple platform but it is also packaged within an *Enterprise Edition*. The standalone version is quite poor in terms of tooling. The standalone version is an access to an *Esper API without* any console, debugging tool or EPL testing environment. In the Enterprise version the *Esper* engine comes with a rich web-based user interface allowing the management of the engine aspects. This enterprise environment embeds a debugging tool to display the distributed real-time event streams plus a runtime manager to monitor over a standard JMX connector.

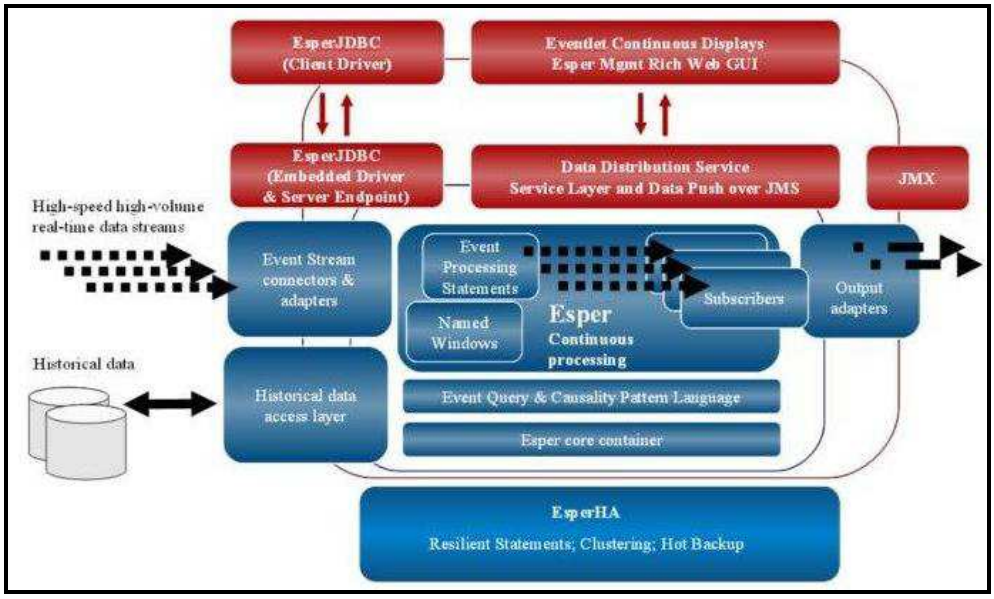


Figure 5 Technical perspective of Esper Enterprise edition

4.1.3.3. Use case implementation

The aim is to implement the scenario described before in the use-case Scenario part. This test will allow to properly evaluate the limitations of the solution. To implement this use-case we have used the basic Espec engine as it is the provided open source solution. The basic Esper engine environment only provides a jar file as an API. So we used Eclipse for the IDE and we have created a java project using the Esper 4.2 version. Then, we have to create the rules according to our scenario.

In this scenario we need to use the current customer position, the taxis position and presence in order to find the closest taxi. We calculate the flying distance between the customer and all available taxis. We need two windows to store the current taxis position and presence. We will also create a window to store the customer query including the selected taxi. The EPL rules written to cover the scenario are listed below:

Create the taxis presence window

CREATE WINDOW StoredTaxiPresenceEvents.std:groupwin(phoneNumber).win:length(1) (phoneNumber string,status boolean

Create the taxis last position window

CREATE WINDOW StoredTaxiPositionEvents.std:groupwin(phoneNumber).win:length(1) (phoneNumber string,latitude double,longitude double)

Create the customer request window

CREATE WINDOW StoredCustomersRequests.std:groupwin(taxiId).win:length(1) (customerNumber string,latitude double,longitude double,taxiId string,distance double)

Store the taxis presence

We have to update those windows in order to store the data.

INSERT INTO StoredTaxiPresenceEvents SELECT phoneNumber,status FROM PresenceEvent where userType='Driver'

Store the taxis last position

INSERT INTO StoredTaxiPositionEvents SELECT phoneNumber, latitude ,longitude FROM GeolocationEvent where userType='Driver'

Handle incoming customer queries

Using the previous stored data (taxi position and presence) and the incoming customer query (including his current location), we can calculate the distance between the customer and all taxis. We use the closest one (ordered by distance and we select the first one).

```
INSERT INTO StoredCustomersRequests (customerNumber,latitude,longitude,taxiId,distance)
SELECT Customer.customerNumber, Customer.latitude, Customer.longitude,
TaxiPresence.phoneNumber, (6378137 *
Math.sqrt(Math.pow((Math.toRadians(StoredTaxiPositionEvents.latitude) -
Math.toRadians(Customer.latitude)), 2) +
Math.pow((Math.toRadians(StoredTaxiPositionEvents.longitude) -
Math.toRadians(Customer.longitude)), 2))) as distance
FROM StoredTaxiPresenceEvents as TaxiPresence,StoredTaxiPositionEvents,
SMSCustomerAlert.win:length(1) as Customer
WHERE StoredTaxiPositionEvents.phoneNumber=TaxiPresence.phoneNumber and
TaxiPresence.status=true and TaxiPresence.phoneNumber not in (select taxiId from
```

StoredCustomersRequests) and Customer.customerNumber not in (select customerNumber from StoredCustomersRequests) order by distance asc limit 1

Details

INSERT INTO StoredCustomersRequests (customerNumber,latitude,longitude,taxiId,distance)

we do insert the customerNumber, latitude, longitude, taxiId and distance from the result of the following select query

```
SELECT Customer.customerNumber, Customer.latitude, Customer.longitude,
TaxiPresence.phoneNumber, (6378137 *
Math.sqrt(Math.pow((Math.toRadians(StoredTaxiPositionEvents.latitude) -
Math.toRadians(Customer.latitude)), 2) +
Math.pow((Math.toRadians(StoredTaxiPositionEvents.longitude) -
Math.toRadians(Customer.longitude)), 2))) as distance
FROM StoredTaxiPresenceEvents as TaxiPresence,StoredTaxiPositionEvents,
SMSCustomerAlert.win:length(1) as Customer
WHERE StoredTaxiPositionEvents.phoneNumber=TaxiPresence.phoneNumber and
TaxiPresence.status=true
```

We do select the customer Number, latitude, longitude, taxi phone Number and the compute distance from the customer request (SMSCustomerAlert.win:length(1)) and all the available taxis (StoredTaxiPositionEvents.phoneNumber=TaxiPresence.phoneNumber and TaxiPresence.status=true)

TaxiPresence.phoneNumber not in (select taxiId from StoredCustomersRequests) and Customer.customerNumber not in (select customerNumber from StoredCustomersRequests) order by distance asc limit 1

We are only interested by taxi that are not vurrently on a fare (TaxiPresence.phoneNumber not in (select taxiId from StoredCustomersRequests))
and customer that do not have already a processing request stored (Customer.customerNumber not in (select customerNumber from StoredCustomersRequests))
Finally we are only interested by the closest one, so we do order by distance and return the first one (order by distance asc limit 1)

Those are the described rules needed to implement the scenario, in order to inject those rules within the Esper CEP engine we just have to call a method and assign an action listener that will be triggered when some rule matches the incoming events (as described below).

```
stmt = "INSERT INTO StoredCustomersRequests (customerNumber,latitude,longitude,taxiId,distance SELECT Customer.customerNumber,Customer.latitude,Customer.longitude,TaxiPresence.phoneNumber, (6378137 * Math.sqrt(Math.pow((Math.toRadians(StoredTaxiPositionEvents.latitude) - Math.toRadians(Customer.latitude)), 2) + Math.pow((Math.toRadians(StoredTaxiPositionEvents.longitude) - Math.toRadians(Customer.longitude)), 2))) as distance
FROM StoredTaxiPresenceEvents as TaxiPresence,StoredTaxiPositionEvents, SMSCustomerAlert.win:length(1) as Customer
where StoredTaxiPositionEvents.phoneNumber=TaxiPresence.phoneNumber and TaxiPresence.status=true
and TaxiPresence.phoneNumber not in (select taxiId from StoredCustomersRequests) and Customer.customerNumber not in (select customerNumber from StoredCustomersRequests) order by distance asc limit 1";

statement = esperEngine.getEPAdministrator().createEPL(stmt);
statement.addListener(new TaxiSelectionListener());
```

4.1.3.4. Runtime Architecture, Deployment, Admin tools

EsperEE is a target deployment platform. Esper alone is not shipped with a server as it is designed as a core CEP engine. To support packaging and deploying event-driven applications, Esper offers an infrastructure with:

- EPL modules to build a cohesive, easily-externalizable deployment unit,
- A deployment administrative interface,
- Instructions and code for use when the deployment target is a J2EE web application server or servlet runtime.

This EPL module bundles EPL statements with deployment procedure. The environment keeps track of the deployed EPL modules and makes it easier to add, remove, update or deploy and undeploy EPL modules. The administrative interface allows managing the packaging and deployment.

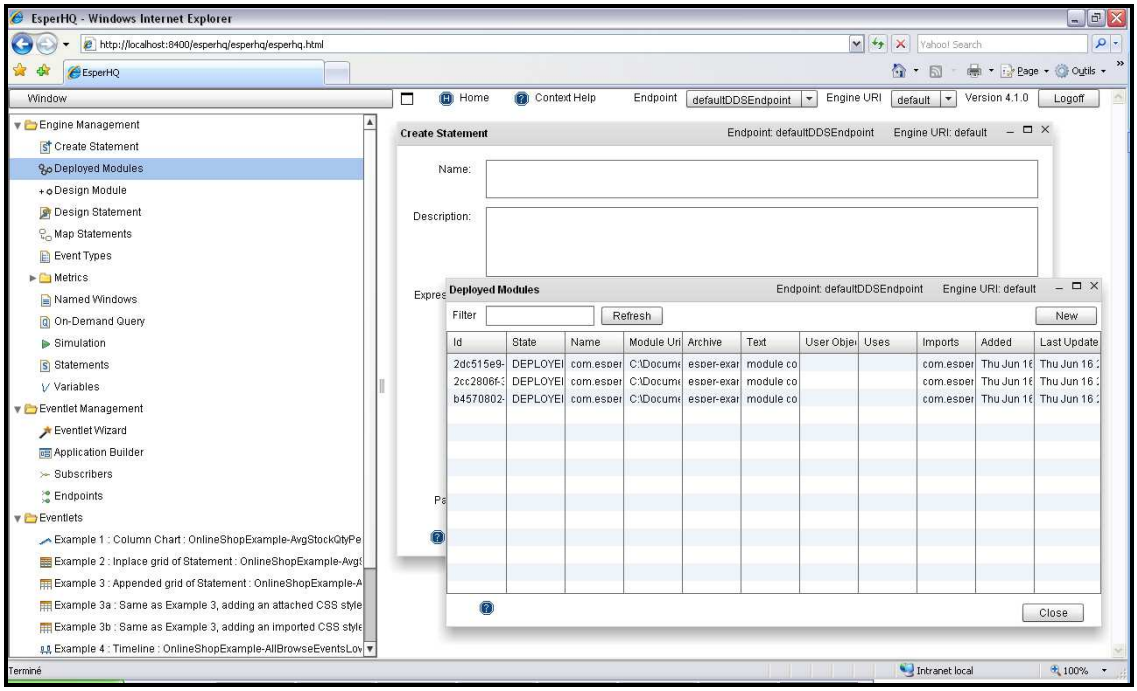


Figure 6 Rich web-based administration tool of the enterprise edition

4.1.3.5. Standards and interoperability

The Esper Enterprise edition comes with a JDBC compliancy for interoperability. The Esper JDBC module turns Esper into a JDBC server that can be queried through the Esper JDBC driver on the client side. This module provides interoperability with any JDBC compliant tool.

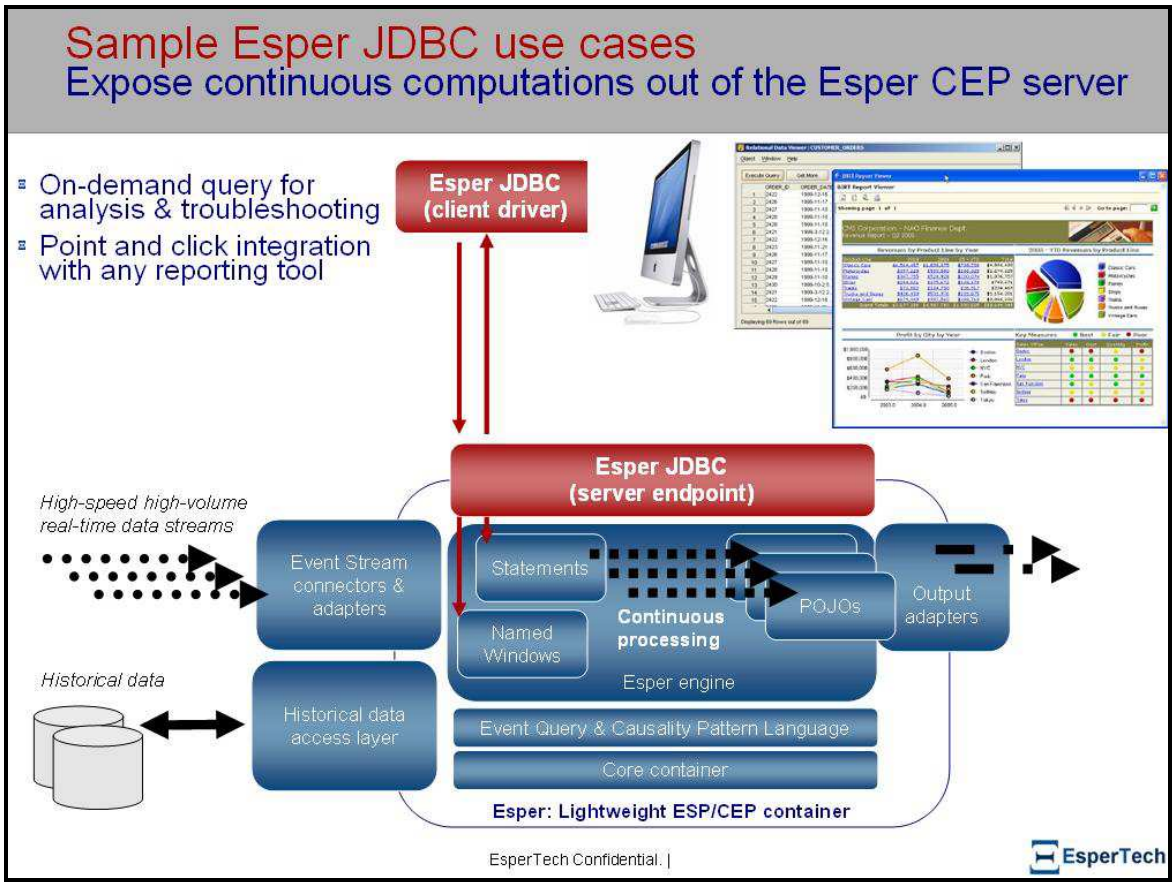
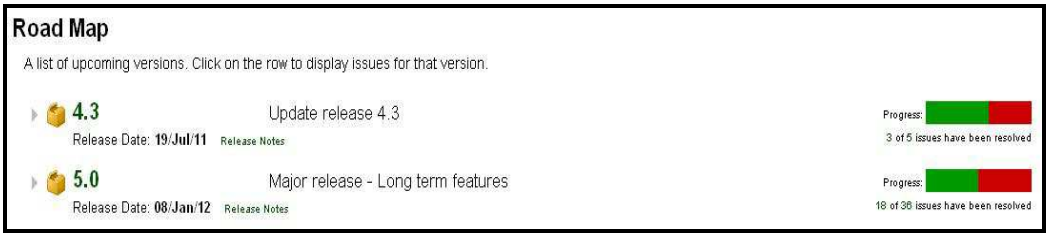


Figure 7 Esper Enterprise JDBC architecture sample

4.1.4. Product Environment criteria

4.1.4.1. Roadmap

Since the beginning of the year 2011 EsperTech delivers the 4.1 and 4.2 versions, the 4.3 version is supposed to be released on July 2011 and the version 5.0 released beginning of 2012.
Below is an extract of the website showing the current implementation status:



4.1.4.2. Licensing

As described before the Esper product is proposed under two different licenses, one for the standalone Esper engine and another one for the EsperEE. Below is an extract from the two licenses description:

- Open Source GPL License – Esper and NEsper are developed under an open source model under the GPL v2 license – the General Public License. The GPL requires that any linked code must also be made GPL when distribution occurs (section 2.b). The software is also provided as-is, unsupported. This has serious limitations for any enterprise grade deployments or any ISVs building products atop EsperTech technology:
  - your intellectual property is exposed under the terms of the GPL license
  - no service level agreements available on support response time
  - risk of disclosure of sensitive information as help is only provided as-is through public forum or mailing lists
  - no liability and indemnification

- Enterprise Edition – Esper and NEsper are available under an Enterprise Edition version:
  - supported on 24x7 or 8x5 for production and development, in due confidentiality by phone, email, or web portal
  - offered with liability protection and indemnification
  - not licensed under GPL viral copyleft license
  - offered as a single package with add-ons for enterprise grade deployments:
    - Data Distribution Svc: Service layer and managed data push
    - Rich GUI and Eventlet: Management UI and time-and-event-driven displays
    - *EsperJMX*: runtime management and monitoring over standard JMX
    - *EsperJDBC*: JDBC client and server endpoints for interoperability

4.1.4.3. Community

The *EsperTech* website does contain a forum but unfortunately it seems not active.

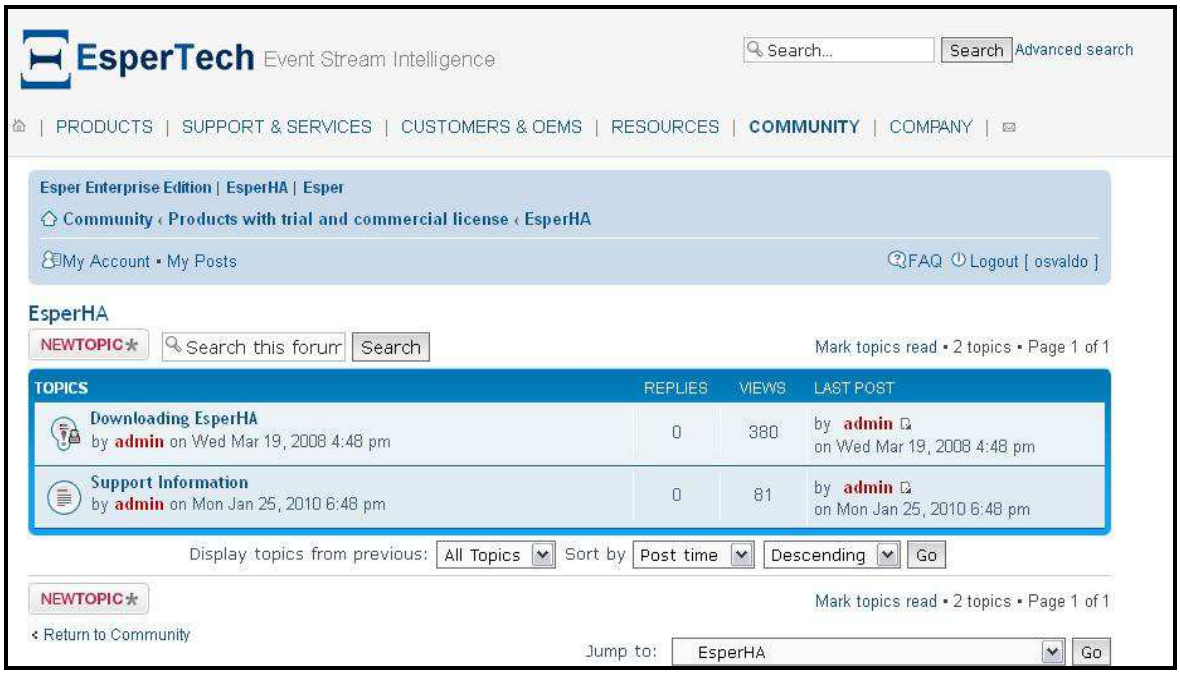


Figure 8 EsperTech forum view

Fortunately, the active community on Esper is located here http://old.nabble.com/Esper-f24500.html with around 50 questions per month in the forum.

4.2. Streambase

4.2.1. Main features

The Streambase Company is focused on complex event processing systems. It claims being the leader in high performance Complex Event Processing, bringing together three capabilities in one integrated platform:

- Rapid development via the industry's first and only graphical event-flow language,
- Extreme performance with a low-latency high-throughput event server,
- The broadest connectivity to real-time and historical data.

From their description it seems the whole platform is quite complete with rapid, on-the-fly processing complex data streams and the fastest time to prototype, test, and deploy real-time CEP applications.

4.2.2. Big Picture

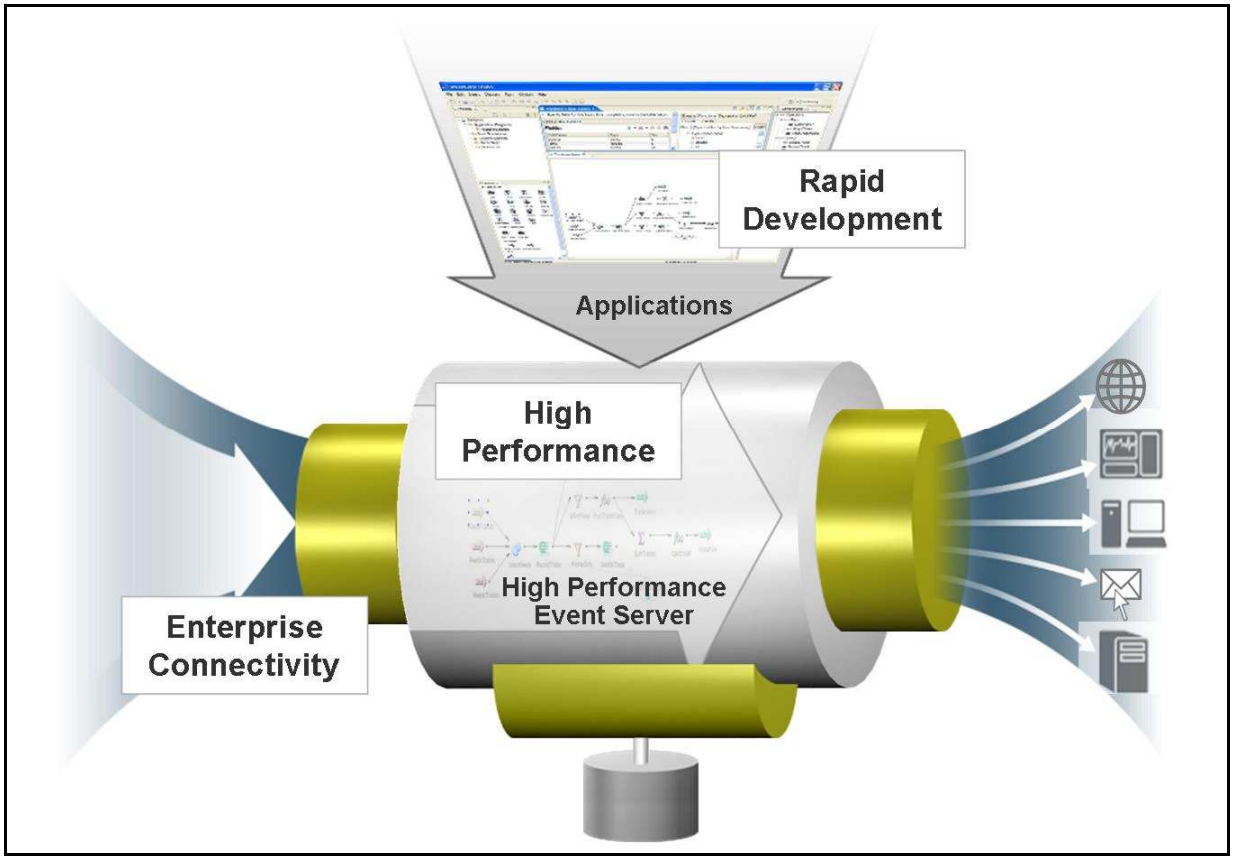


Figure 9 StreamBase highlevel view

As it is presented in the high level view Streambase figure, the key component of the solution is the IDE which facilitates the CEP edition but also the monitoring and debugging aspect. The IDE is a real interface with the event server allowing:

- Shorter development cycles, an easy maintenance, flexibility and adaptation and High level or business view programming.

4.2.3. Technical criteria

4.2.3.1. Event Processing features

The Streambase CEP engine is based on the StreamSQL language which is an equivalent to the SQL language used to query on databases. In this case it is possible to mix access to the non-persistent stream with access to the stored database. This StreamSQL language extends the semantics of the standard SQL by adding rich windowing and stream-specific operators. Of course, the language operators allow to filter streams, merge, combine, correlate multiple streams and run time-window-based aggregations on real-time stream or stored tables. The system allows extending the language capabilities with new implemented processing functionalities.

4.2.3.2. CEP Tooling around the CEP (SDK, Editors)

The Streambase studio solution is an Eclipse-based IDE to design application through a graphical event-flow language. Basically, the drag and drop environment allows the creation of CEP applications. Of course, the tool contains a developer environment to manually write CEP applications using the Stream-SQL language.

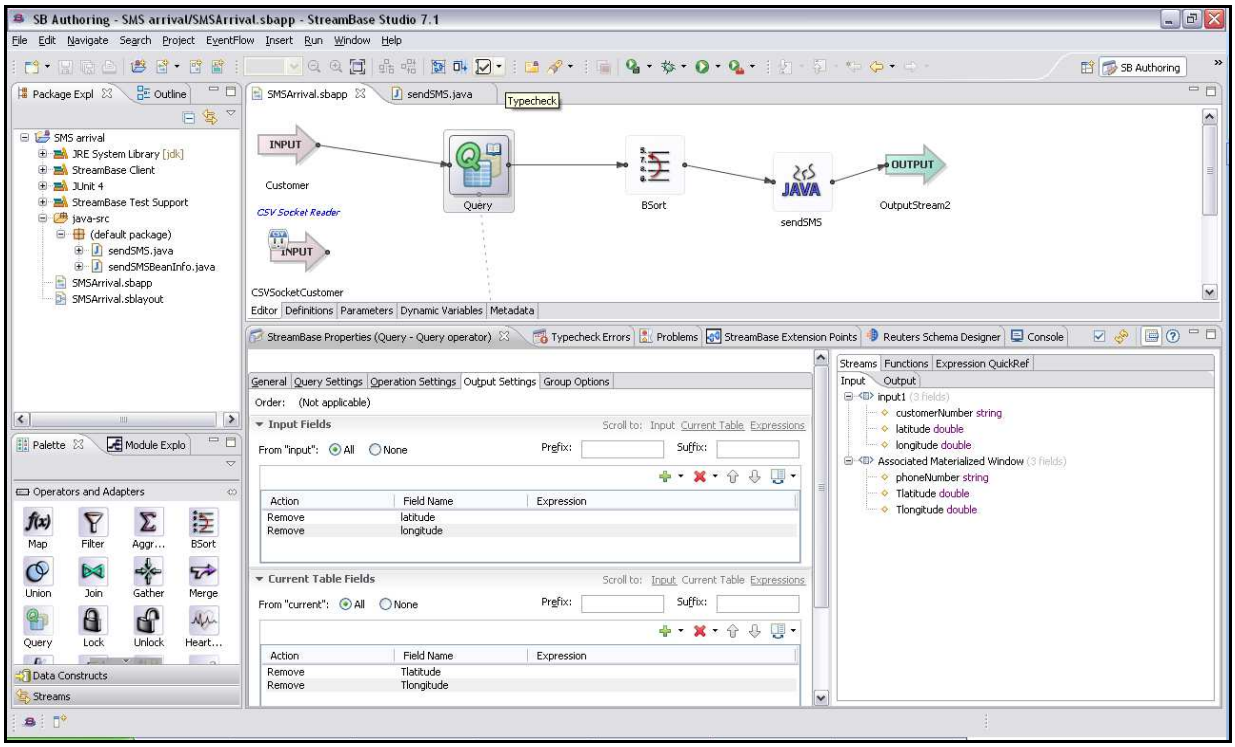


Figure 10 Design environment within Streambase studio

The studio environment is composed of tools such like:

- Application modeling,
- Stream recording/playback,
- Testing/debugging tool,
- Dashboard,
- The possibility of adding external Eclipse plug-in functionalities.

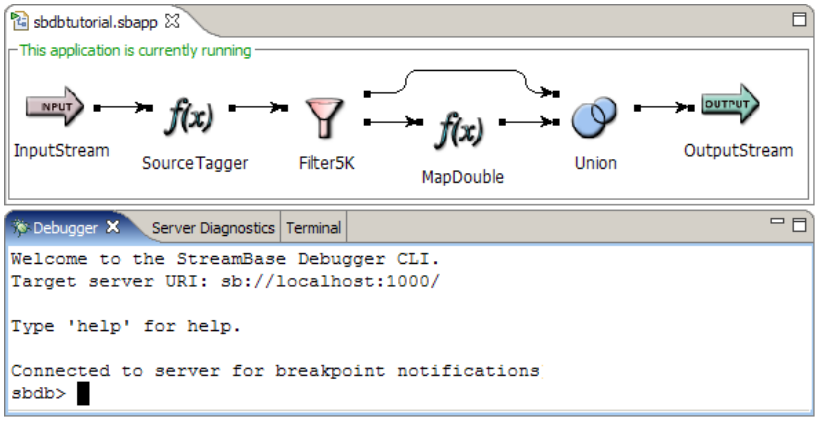


Figure 11 Streambase debugger

4.2.3.3. Use case implementation

In order to properly test the Streambase studio environment we have implemented the light use case called the Taxi Use Case Scenario. Basically a customer sends a taxi query event (the source isn't really important, it could be anything implementing the socket API), the CEP is also listening to all incoming taxiPositionEvent. Those positions are stored and use to assign the closest taxi for the new customer. Finally the application is supposed to notify the customer about this information.

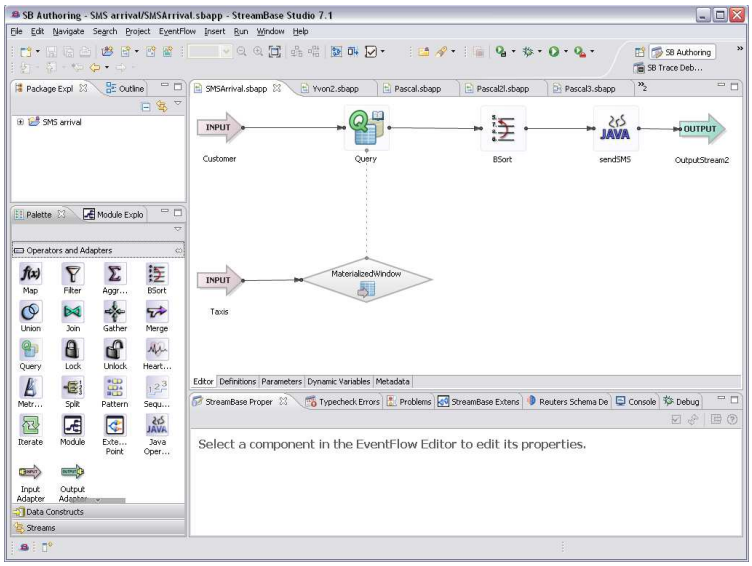


Figure 12 Use Case designed within Streambase

Thanks to the drag&drop interface we have been able to design our application:

- Incoming Taxis positions are stored in a window,
- Incoming Customer query is processed and uses the taxi position to find the closest one,
- The distance between the customer and all stored taxi positions is calculated. Then we do sort the whole result by distance.

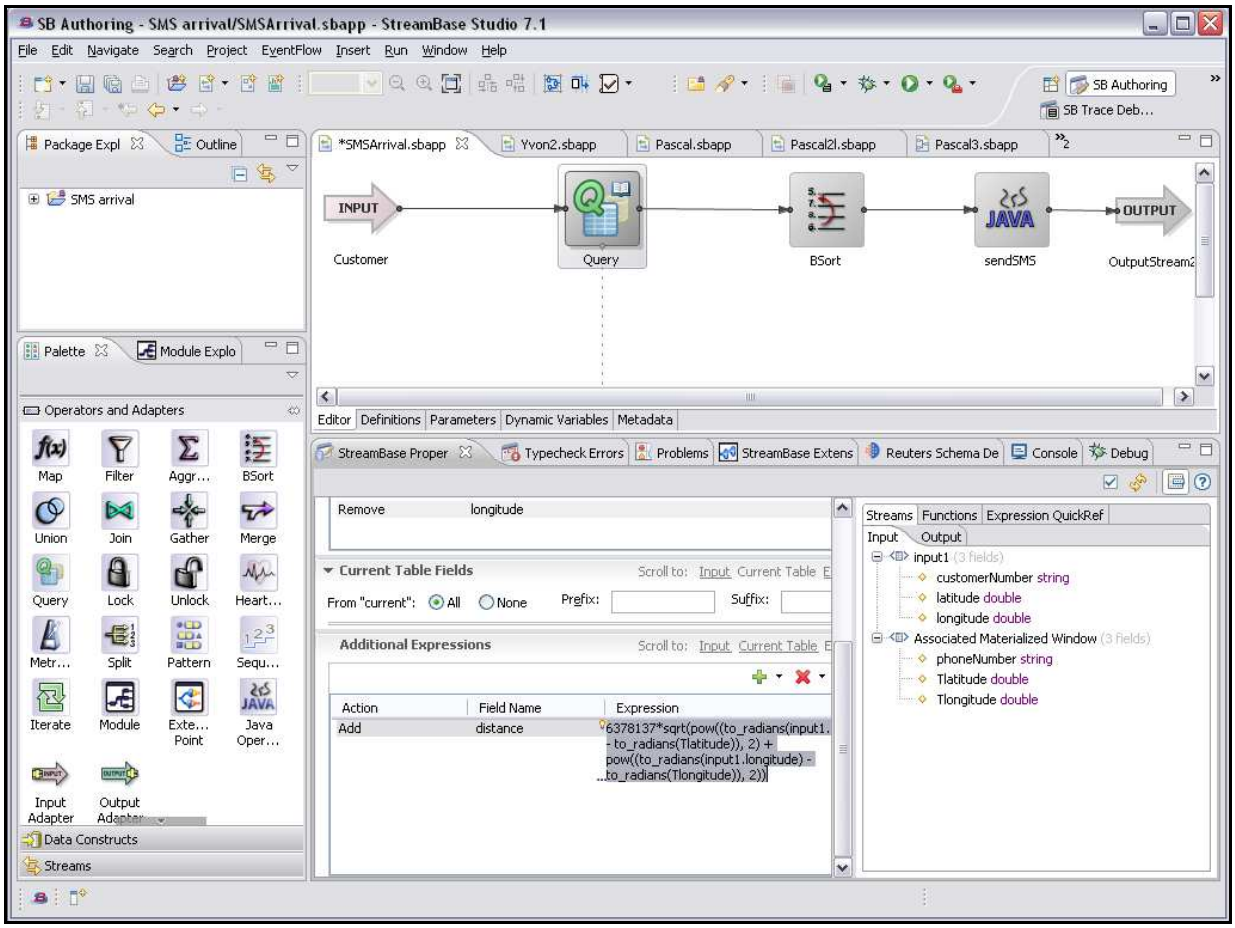


Figure 13 Query definition

- Finally we get the first one (the closest one).
- We do use customer phoneNumber information to send a sms.

This sendSMS has been implemented using an orange rest API. So we have implemented it in java and retrieved it in our palette. Finally we do display a result in a console thanks to an outputStream element from the palette.

This is the business view but we could have implemented such a thing using the StreamSQL language. The generated result is:

```
CREATE PUBLIC INPUT STREAM Customer (
    customerNumber string,
    latitude double,
    longitude double
);
CREATE INPUT STREAM Taxis (
    phoneNumber string,
    Tlatitude double,
    Tlongitude double
);
CREATE OUTPUT STREAM OutputStream2 ;

 -- <window name="MaterializedWindow" type="materializedwindow">
 --             <param name="storage-method" value="memory"/>
 --             <param name="shared-table" value="none"/>
 --             <param name="type" value="tuples"/>
 --             <param name="window-size" value="1000"/>
 --                                                 <param    name="partition-by"
value="phoneNumber,Tlatitude,Tlongitude"/>
 --             <input port="1" stream="Taxis"/>
 --         </window>
CREATE MEMORY MATERIALIZED WINDOW MaterializedWindow AS Taxis[SIZE 1000 TUPLES
PARTITION BY phoneNumber,Tlatitude,Tlongitude];

CREATE STREAM out__Query_1 ;
SELECT  Customer.customerNumber,  MaterializedWindow.phoneNumber,  6378137  *
sqrt(pow(to_radians(Customer.latitude)    -    to_radians(Tlatitude),    2)    +
pow(to_radians(Customer.longitude) - to_radians(Tlongitude), 2)) AS distance
 FROM Customer, MaterializedWindow
WHERE true
INTO out__Query_1;

CREATE STREAM out__BSort_1 ;
BSORT out__Query_1 ON distance SIZE 1.0 INTO out__BSort_1;

APPLY JAVA "sendSMS" AS sendSMS (
    content = "the taxi is coming",
    number = out__BSort_1.customerNumber
)
FROM out__BSort_1
INTO OutputStream2;
```

4.2.3.4. Test & simulation Environment

On click is needed to run your application on a server and use the simulation mode. This simulation perspective is very useful as you can follow the events along their paths and check what is stored and where events are eventually stuck.

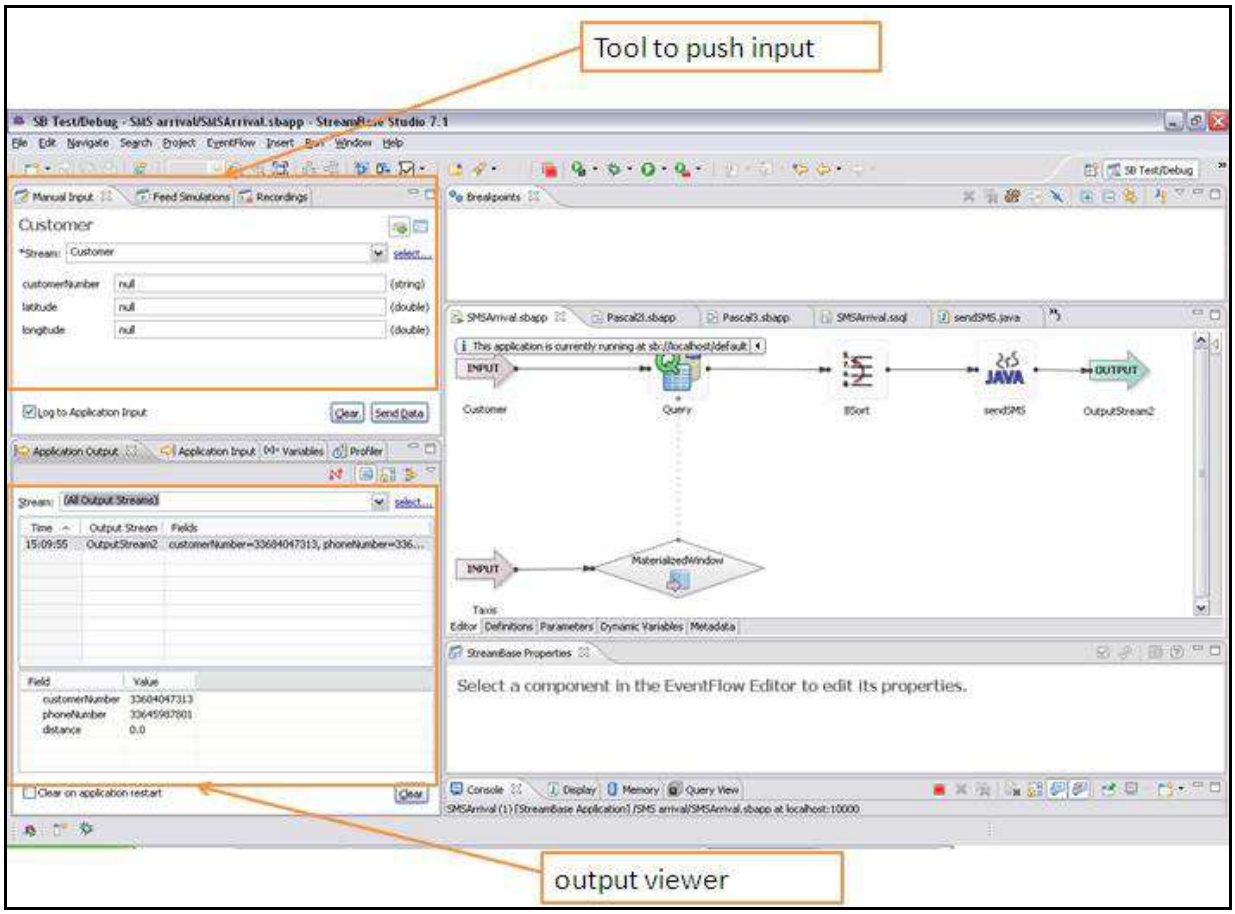


Figure 14 Streambase simulator

4.2.3.5. Runtime Architecture, Deployment, Admin tools

The Streambase studio solution offers more than an IDE for development of application through a user friendly interface. It is also an environment for building, testing and deploying applications. It offers many tools to cover the whole life cycle:

- StreamBase Feed Simulator, to push simulation data,
- StreamBase Debugger, debugging tool,
- StreamBase Monitor, mainly to monitor performance,
- StreamBase Record and Playback, capture incoming data and play them back,

- StreamBase Java API, customize or create new components such like operators, adapters or functions.

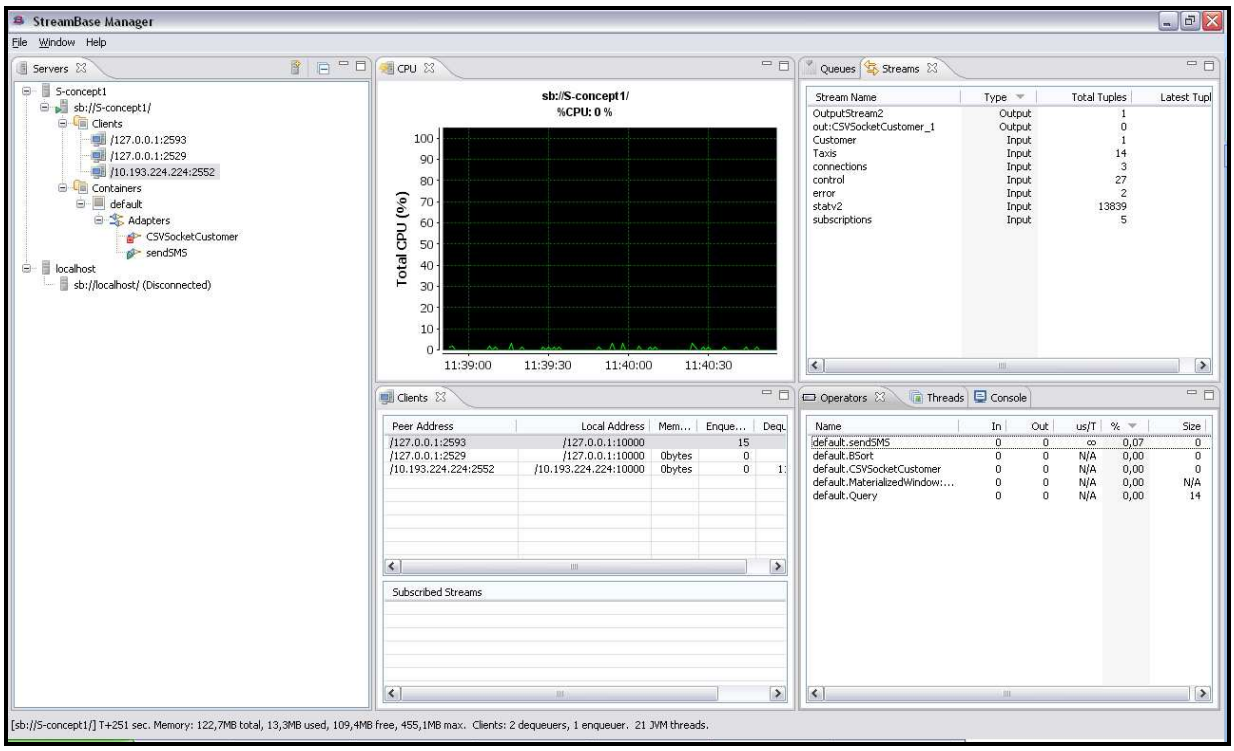


Figure 15 Stream base monitoring tool

4.2.3.6. Standards and interoperability

The solution is based on the well known standard SQL which is a good point in term of user experience; it means there is no need to learn a new query language to start working with it.

4.2.4. Product Environment criteria

4.2.4.1. Roadmap

Currently at version 7 stage, below is the prevision plan for the releases:

Release	Satus	Initial Ship	Sundown Begins	Last Maintenance Release (no further patches issued	Support Ends
7.0	Fully Supported	Nov 10, 2010	9.0 GA	9.0 GA + 1 year	9.0 + 2 years
6.6	Fully Supported	Apr 15, 2010	8.0 GA	8.0 GA + 1 year	8.0 + 2 years
6.5	Prior Version	Nov 15, 2009	November 8, 2010	November 8, 2011	November 8,2012
6.4	Prior Version	Jun 12, 2009	Apr 14, 2010	Apr 14, 2011	Apr 14, 2012
6.3	Prior Version	Mar 12, 2009	Dec 16, 2009	Dec 16, 2010	Dec 16, 2011
6.2	Prior Version	Nov 17, 2008	Jun 12, 2009	Jun 12, 2010	Jun 12, 2011
5.1	Fully Supported	Dec 27, 2007	November 8, 2010	November 8, 2011	November 8,2012

4.2.4.2. Licensing

There is no open source license and as described below in the table (extract from the official website), only a trial edition is free:

Feature	Developer Edition	Enterprise Edition
Supported Platforms	• Microsoft® Windows™ 7 Professional, 32- and 64-bit • Windows Vista Business, 32- and 64-bit • Windows XP Professional, 32-bit • Windows Server 2008, 64-bit • Windows Server 2003, 32- and 64-bit • Red Hat™ Enterprise Linux 4 AS, 32-bit Developer Edition kits are 32-bit versions. The Linux kit requires a 32-bit version of Linux. The Windows kit installs and runs on both 32-bit and 64-bit Windows.	• Red Hat Enterprise Linux 4 AS (32-bit or 64-bit) • Red Hat Enterprise Linux 5 AS (64-bit) • Novell SuSE Linux Enterprise Server 10 (64-bit) • Microsoft® Windows™ 7 Professional, 32- and 64-bit • Windows Vista Business (32-bit or 64-bit) • Windows XP Professional (32-bit) • Windows Server 2008 (64-bit) • Windows Server 2003 (32- or 64-bit) • Sun Solaris 10 for SPARC or Intel (64-bit; StreamBase Server and command-line environment only)
License	Trial	Subscription or Perpetual
Price	Free	Varies according to the scale of your Enterprise deployment.
StreamBase Studio development environment to create and test applications	Yes	Yes (not available on Solaris)
Disk-based Query Tables to persist application state	No	Yes
All other features	Yes	Yes

4.2.4.3. Community

The website does not offer any forum or chat, otherwise the documentation and sample/demo matter seems being quite full. The website does also contain a training section as well as a component exchange platform. This final aspect is quite interesting because it is an exchange platform to share implemented component between Streambase customers.

4.3. ALERI

4.3.1. Main features

The Sybase Aleri Streaming Platform is a next generation Complex Event Processing (CEP) platform that delivers Continuous Intelligence™ across enterprise networks for faster decision making and business execution. The Sybase CEP technology is deployed:

- in financial institutions and markets, where real-time insight and rapid response is critical,
- in the front office, where market data represents events, such as executed trades and quotes that have been published.

Industries domains where CEP is being deployed include telecommunications, network security, e-commerce and retail. The Sybase Aleri Streaming Platform (SASP) provides high-performance complex event processing (CEP) for rapid development. The key features provided by the SASP are:

- Providing of immediate response and better decisions in front of changing conditions.
- Building and deploying applications that have the power of analyzing and acting on streams of event data, in real-time,
- Building and deploying applications that have the power of detecting patterns of interest, and deliver continuous intelligence for better decision making.

The SASP is a complex event processing (CEP) software that performs real-time processing and analysis of incoming data. It helps to build applications for getting timely insight into current conditions or an immediate response to opportunities and threats.
On top of the SASP, Sybase allows users to implement their event processing logic using a visual dataflow editor (Aleri Studio). This Aleri Studio environment eliminates the need to learn an event processing language, making authoring accessible to non programmers.

The Aleri Studio is the authoring environment most users initially choose to create, edit and test data models in the SASP. It is an Integrated Development Environment (IDE) based on the Eclipse™ framework. With the help of the Studio, a non programmer can create data models, add processing elements called "streams" and connect the streams to define the data flow. To summarize, Sybase proposes not only a CEP component but a complete middleware platform from the ground up, composed of the following components (see Fig 11):

- *Sybase CEP Engine* : The CEP middleware component,
- *Sybase RAP*: The Trading Edition (Sybase RAP). It provides a real-time cache and a historical repository with the ability to load, store and analyze market data,
- *Sybase Studio*: A complete IDE environment helping to build CEP applications.

4.3.2. Big Picture

The stack of the different components delivered by Sybase is gathered on the following figure:

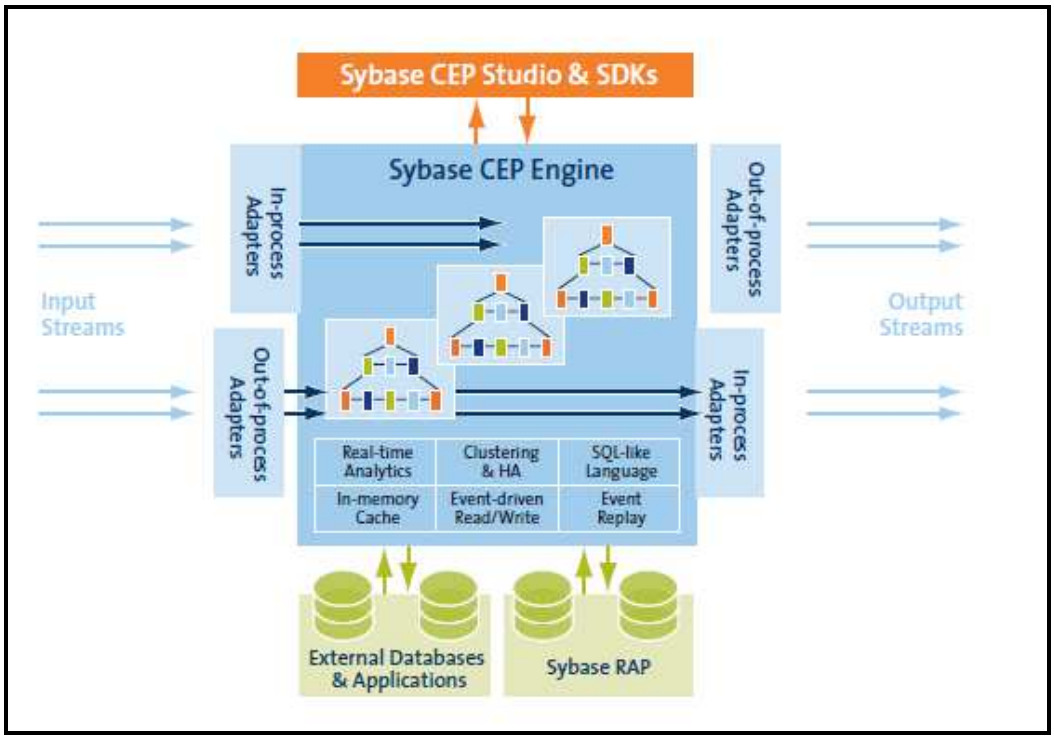


Figure 11 Sybase CEP Middleware components

The key component of the SASP is the CEP engine which runs as a server process. The event processor loads a data model that has been defined using the Aleri authoring options (from Aleri Studio), and then waits for message(s) to occur on the source streams. As each event arrives, the processing logic contained in the data model is applied to update one or more derived streams. Output messages are continuously generated for the derived streams that have registered subscribers.

Main features

This is the list of the key features proposed by the platform:

- Streaming data sources providing continuous event data and static data sources extracting information from databases or files,
- Message Oriented Middleware supplying messages from data sources to the Aleri Streaming Platform and delivering streaming output to applications that will use the results (Tibco Rendezvous, MQ and Java Message Service),
- Reporting tool and dashboard used to display output from the Aleri Streaming Platform.
- Export Interfaces towards Excel and others tools used to query the Aleri Streaming Platform by using the Open Database Connectivity (ODBC), C++ ConnectivityAPI, or Java Database Connectivity (JDBC),
- Storing events in an Event Database. Aleri LiveOLAP accumulates historical event data, which can be queried,

- Connectors and Adapters converting incoming messages into the Aleri data format and converting outgoing messages from the Aleri data format into the format expected by the receiving application. The Aleri Streaming Platform has a range of built-in connectors that can be used to link the most common general-purpose data sources and destinations through JDBC, XML formatted messages from/to a file or socket, CSV formatted messages from/to a file or socket, Aleri Streaming Platform to/from Aleri Streaming Platform, JMS, mail Server,
- High Availability: Multiple high availability options including instant hot-failover and high speed data persistence for rapid recovery without data loss,
- Security features include authentication, a stream-level access control, and encryption.

4.3.3. Technical criteria

4.3.3.1. Event Processing features

Input and adaptors
Aleri provides predefined input events such as: ACTIV Financial, Atlas, FIX 4.2/4.4, NYSE/Wombat, Reuters Market feed, Reuters OMM, and SWIFT.
Numerous adaptors are available such as: JMS, JDBC, CSV files, SMTP, Oracle, SQL Server, and TIBCO Rendezvous (RV). Advanced customers have the possibility to develop themselves custom adapters in C++, Java, and .NET.
Events
Aleri provides customers with the ability to retain events based on time and number of events. In addition, customers can use Aleri's SPLASH scripting language to develop a custom retention window. External applications can query live events within the platform through Aleri's JDBC or ODBC interface. The Sybase Aleri Platform is the only CEP engine to combine standard relational operators with a procedural scripting language for custom operators and functions. This provides increased flexibility and control in the types of business logic that can be easily implemented
Output
Microsoft Excel plug-in helps to visualize output within Excel.

4.3.3.2. CEP Tooling around the CEP (SDK, Editors)

Aleri Studio IDE
Aleri's IDE named *Aleri Studio* is based on Eclipse. It provides authoring, testing and debugging environment on top of the Aleri Platform SASP. The testing tool record and play back data, generates manual or simulated input data, and runs manual queries against retained windows. For simulation, developers can use the Aleri IDE to manually inject events into the platform streams. Finally, the Aleri Studio provides a visual data-flow editor that can be used to build complete applications. The user connects to the SASP and then authors a new CEP application. The complete cycle (visual authoring, testing and deployment of the CEP Application) is covered. A short tutorial (Introduction to Data Modeling and the Aleri Studio) is available for download on the Aleri Web site (see Appendix).
We have implemented parts of the Taxi Use Case (Scenario SMS Arrival see 3.2]) with the Aleri Studio [4.4.3.3] .

Programming
The Java and C++ programming languages are available for developers and Aleri delivers programming interfaces:

- A Publish/Subscribe (Pub/Sub) Application Programming Interface (API) for Java,
- A Pub/Sub API for C++,
- A Pub/Sub API for .NET,
- Command & Control interface,
- An SQL query interface.

4.3.3.3. Use case implementation

We have tested the studio with the Taxi Use Case Scenario, described before. The studio delivers a palette of graphical objects classifying objects as **Streams** and **Stores** components. Components may be gathered to constitute **Modules** and **Clusters**.

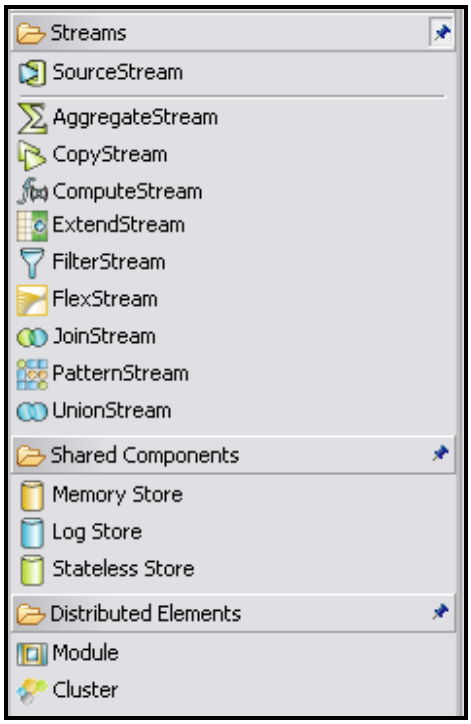


Figure 16 Palette

The following screen shows a first view at design time. We have visually designed the UC scenario with components available form the Palette.

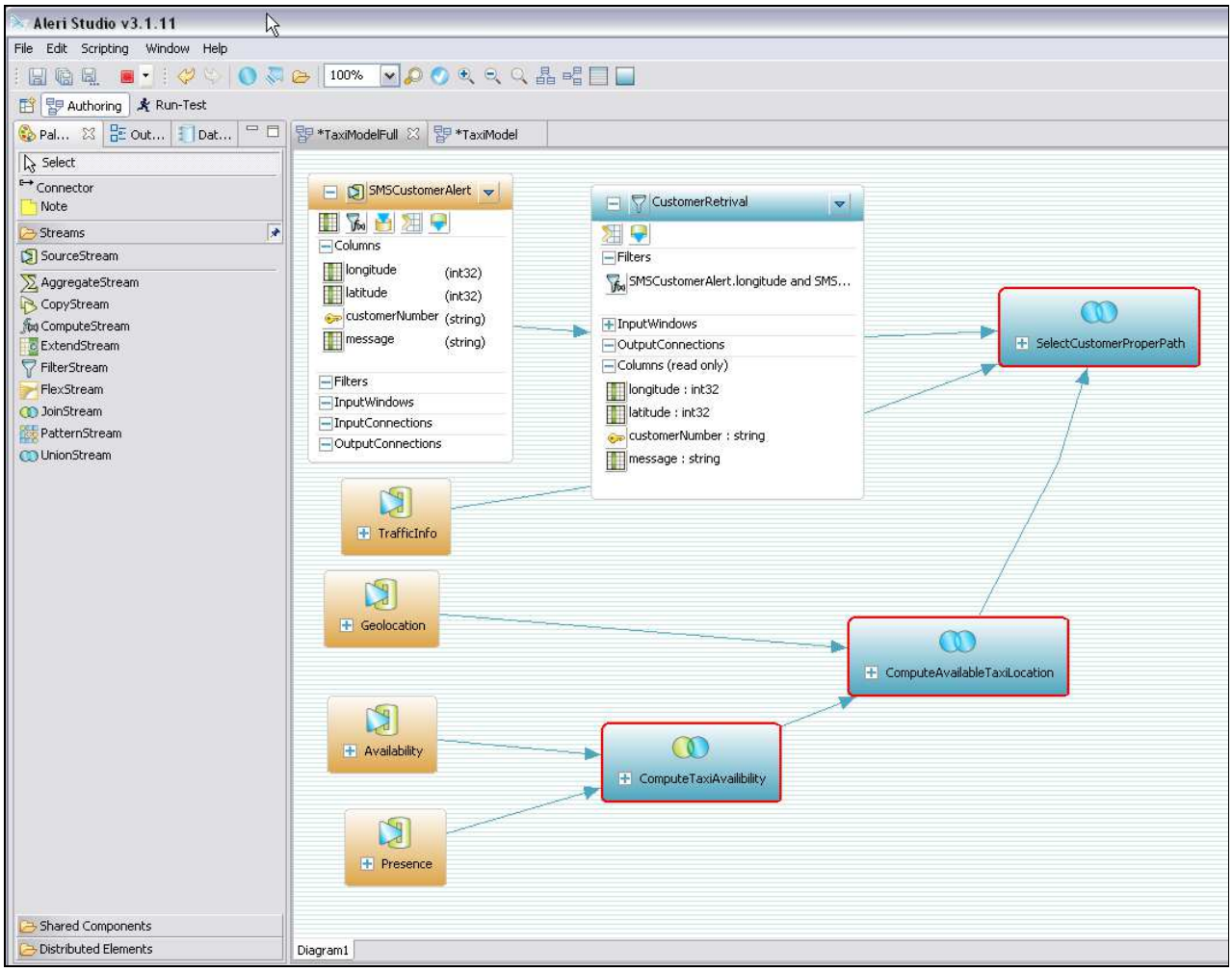


Figure 12 Aleri Studio - Design step SMS Arrival

Then testing can be done immediately by connecting the Streams to XML files or databases:

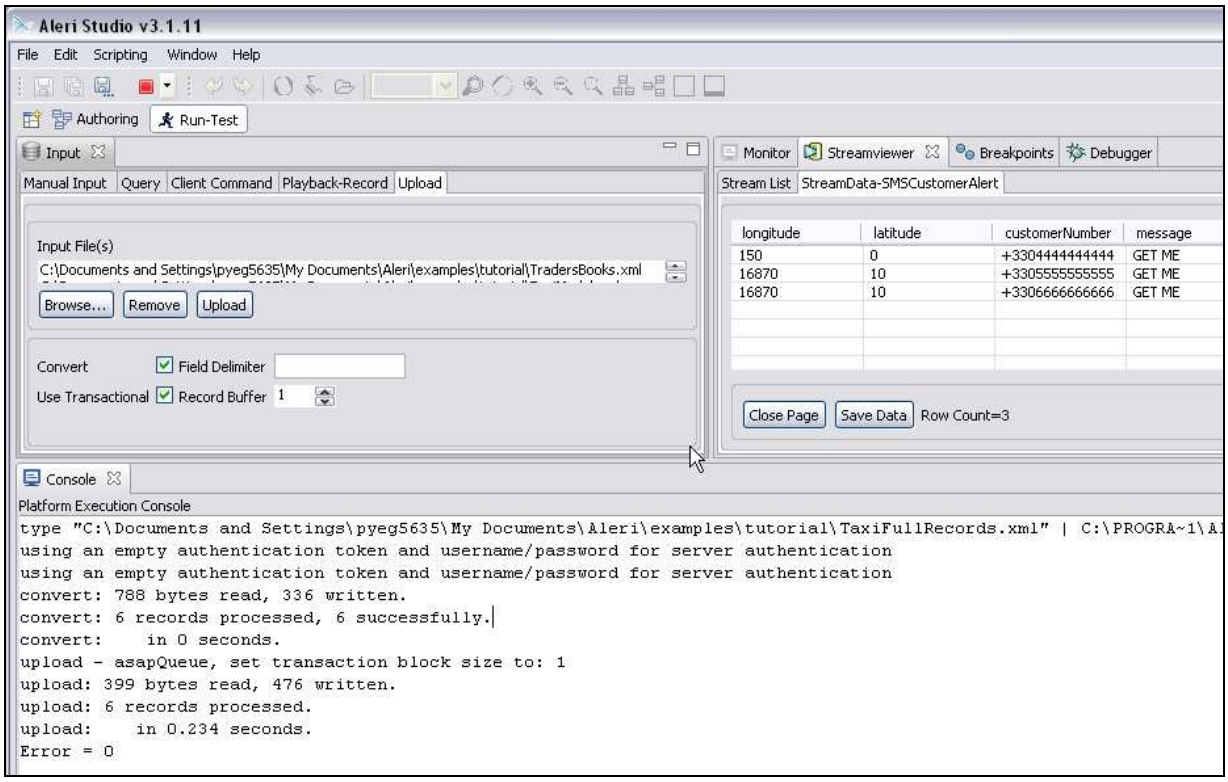


Figure 13 Aleri Studio – Testing Streams with data from XML files

4.3.3.4. Runtime Architecture, Deployment, Admin tools

Runtime

Aleri applications can be deployed on Linux, Solaris, and Windows. The Aleri platform provides user- and role-based authentication and authorization for access to tools.

Authentication

The product supports Pluggable Authentication Modules (PAM) on Unix, RSA, and/or Kerberos as additional authentication methods.

Threading

Aleri is designed to use all available memory and CPU cores on a single server. Event streams are automatically multithreaded, allowing them to be automatically scaled across multiple CPU cores. Customers can break the application into smaller units of work and choose to deploy those smaller work units on named servers.

Aleri's cluster manager helps to scale out by using to automatically run a single application across multiple servers. The cluster manager automatically assigns units of work to each server in the cluster. A running application state can be paused and then restarted with a new deployment of the applications.

4.3.3.5. Standards and interoperability

The platform exposes a documented API to access the functionality. Java, and C++ programming languages are available for developers through a documented API. Developers can also build BPM integrations with the Aleri platform.

4.3.4. Product Environment criteria

4.3.4.1. Roadmap

Roadmap

Aleri has a strong and comprehensive product roadmap for the Aleri platform and gives customers three license choices as well as a free download option. The company has a complete corporate strategy.

Documentation

All the mandatory documentation is available : Installation Guide, Authoring Authoring Reference, Guide to Programming Interfaces Provides instructions and reference information for developers, Administrator's Guide, Introduction to Data Modeling, Aleri Studio, SPLASH Tutorial.

4.3.4.2. Licensing

The Aleri Streaming Platform is available under three licenses: perpetual license, 180-day trial license, annual subscription.

4.3.4.3. Community

Support is provided from the company by the help of *sybase.public.eventprocessing*.

4.4. Drools Fusion

4.4.1. Main features

Drools Fusion is one of the main five integrated modules of the Drools platform. It is responsible of event processing capabilities into the platform, and adds a set of feature to enable it:

- detection, correlation, aggregation and composition of events,
- events streams processing,
- temporal reasoning (13 operators to model temporal relationships between events),
- event garbage support (resources freeing),
- reasoning over absence of events,
- adapters for event input into the engine,
- events relevant relationships detection (pattern matching),
- sliding windows support (temporal or length-based),
- performing actions based on detection.

4.4.2. Big Picture

Drools is a Java Based, open source platform providing a integrated framework for Rules, Workflow and Event Processing.

Drools framework splits into four production modules:

- Guvnor: the Web based governance system,
- Expert: the Rules Engine,
- Fusion (CEP): this module extends the Rules Engine in order to handle events,
- jBPM 5: the Workflow module.

and four incubation modules:

- Planner: dedicated to allocation and scheduling problems,
- Grid: provides distributed *knowledge sessions* execution across distributed grid of machines/nodes,
- Semantics[1]: provides a base language for building a *RuleSet* in XML, allowing to plug domain specific rule languages into Drools,
- Chance[2]: a module for reasoning with “Imperfect Knowledge”.

[1] http://docs.codehaus.org/display/DROOLS/Base+Semantic+Module
[2] Activity towards this module is inactive since 2009

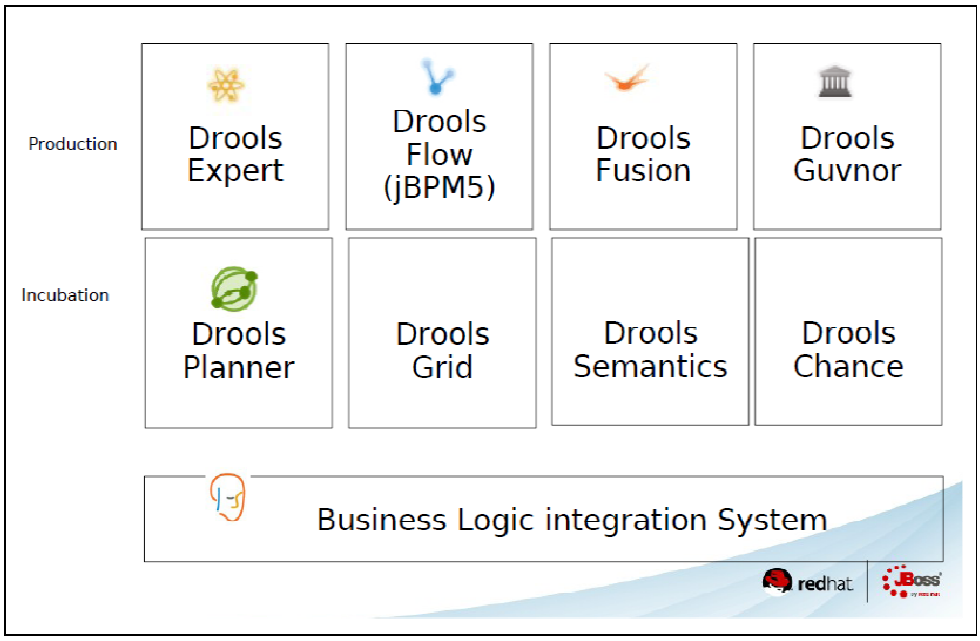


Figure 17 : Drools platform

Drools implements and extends the *Rete*[3] algorithm in an implementation called *ReteOO*.

The Rules are stored in the Production Memory and the facts (/events) that the Inference Engine matches against are kept in the Working Memory. Facts (/events) are asserted into the Working Memory where they may then be modified or retracted. A system with a large number of rules and facts (/events) may result in many rules being true for the same fact assertion; these rules are said to be in conflict. The Agenda manages the execution order of these conflicting rules using a Conflict Resolution strategy.

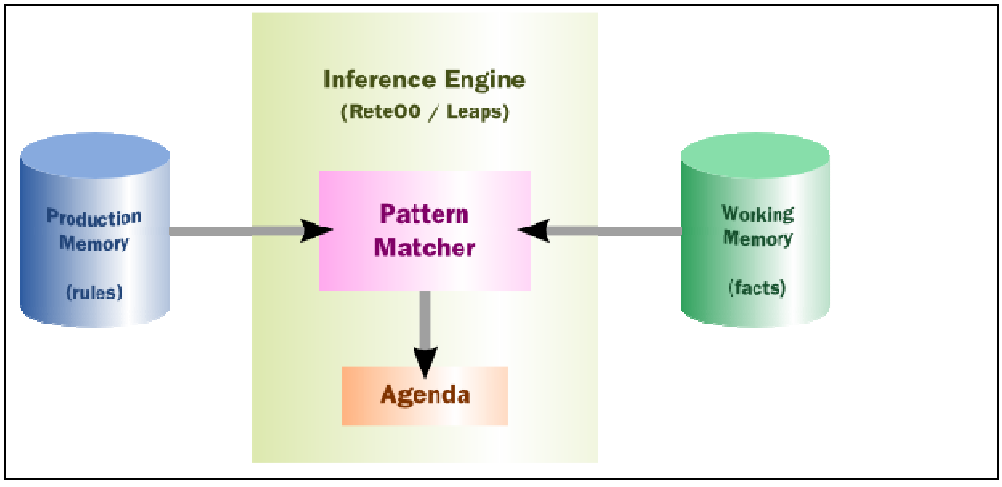


Figure 18 : High-level View of a Rule Engine

[3] Pattern matching algorithm for implementing production rule systems

Drools Fusion enhanced its Business Rule Management System by introducing events principles (in particular, temporal constraints). Thus, the rule engine deals with a special type of *facts* that are events. In many cases in this chapter when we will mention events, it may apply to facts in general, though, as our focus is on Event Processing, we will never generalize.

4.4.3. Technical criteria

4.4.3.1. Event Processing features

As we mentioned previously, events are particular type of facts. Drools facts are piece of information, which structure is implemented by Java bean instances, and that are managed (as rules) in the *Working Memory*, or in database. Events are facts including temporal information, and on which applies temporal rules.
The following set of 13 operators, that are key words in the rule files, allow temporal reasoning:

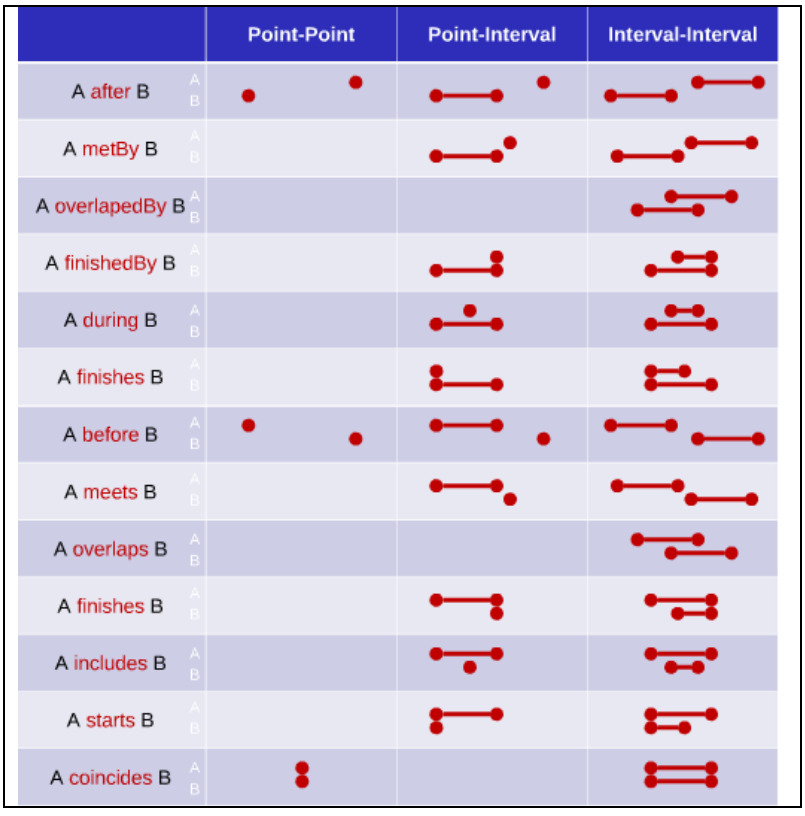


Figure 19: Drool Fusion temporal operators

Event retention is also possible, through setting an expiration time (that can be infinite) to events.

Adapters, allowing to easily receiving events from the outside world, are still in the experimental stage and has to be completed. They're implemented in the Drools Pipeline API (from the Drools core API). This API allows inserting events into the rule engine from JMS queues and text files. Transformers are provided for Smooks, JAXB, Xstream and Jxls.

4.4.3.2. CEP Tooling around the CEP (SDK, Editors)

An Eclipse plug-in for drools allowing edit and test rules, in addition to classical Java code in which CEP part is integrated. It provides debugging facilities, with views adapted to Drools runtime. It allows editing graphically Domain Specific languages (DSL) rules, managing the mapping with DSL assertions.
Eclipse plug-in used to provide a Guided Rule Editor (Wizard), allowing to edit rules graphically. It has been removed in Drools 5.

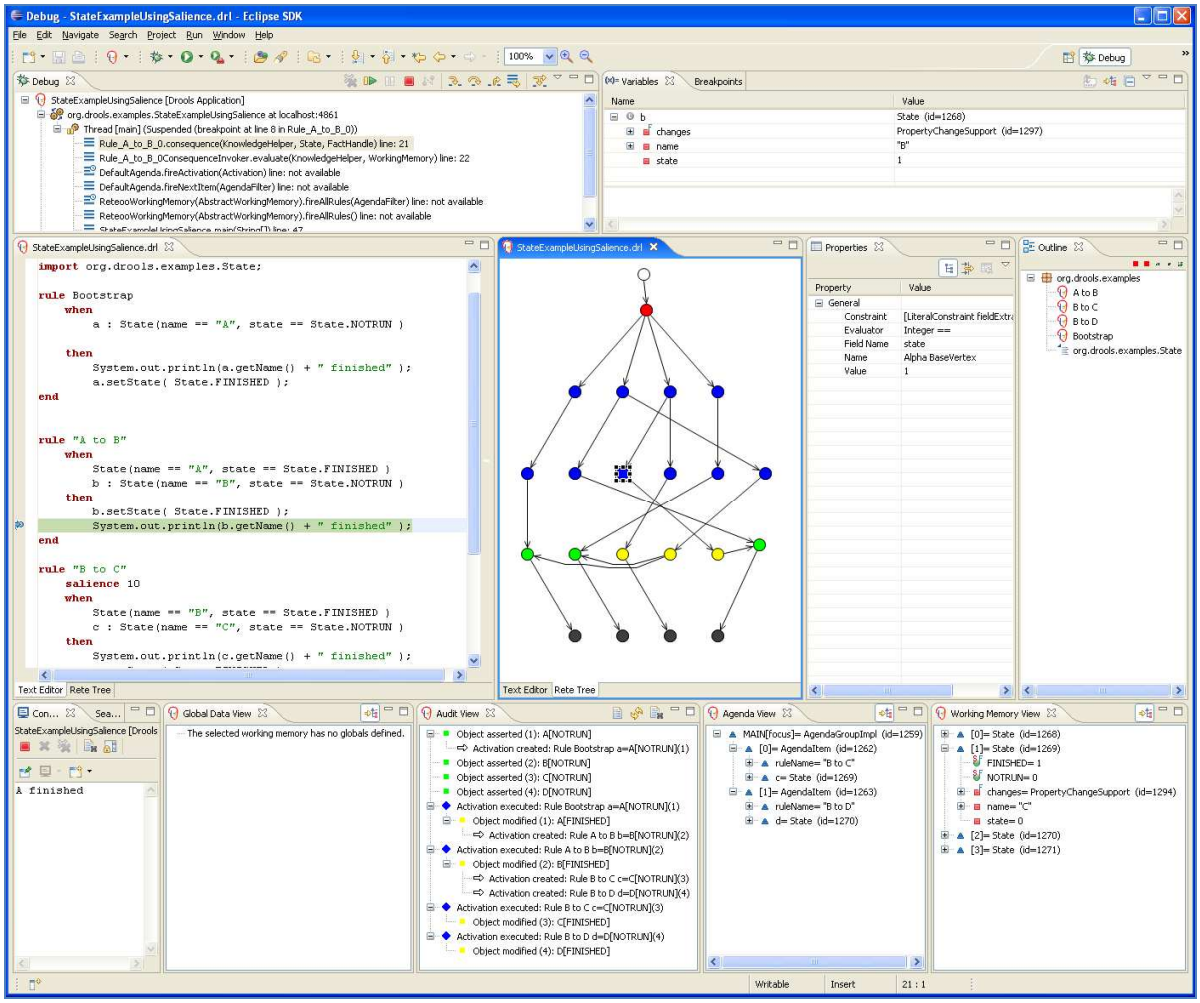


Figure 20 : Drools views in Eclipse

Programming language for Drools is Java 1.5.
The Java APIs available are:

- A core API, to manage and package resources (processes, rules, functions and types), to interact with the Drools engine (inserting, updating events), etc.
- An experimental API based framework, available for runtime session inspection and reporting.
- A Rule API allowing fully programmatic creation of rules.

The Semantics Module Framework (SMF) allows domain specific rule languages (DSL) to be plugged in to Drools. Drools provides SMF implementations for Java, Groovy and Python. We used the Java one, just by declaring it as the default *dialect*. Though, as documentation describes additional configuration to carry languages expansion in rules, we may expect to have to do so for other languages or in particular cases.

Coming from jBPM5 module, workflows can be designed with Drools eclipse plug-in. It makes convenient sequencing packages of rule, to design a higher level rule flow. Beside rules, it includes a few basic items from BPMN standard.

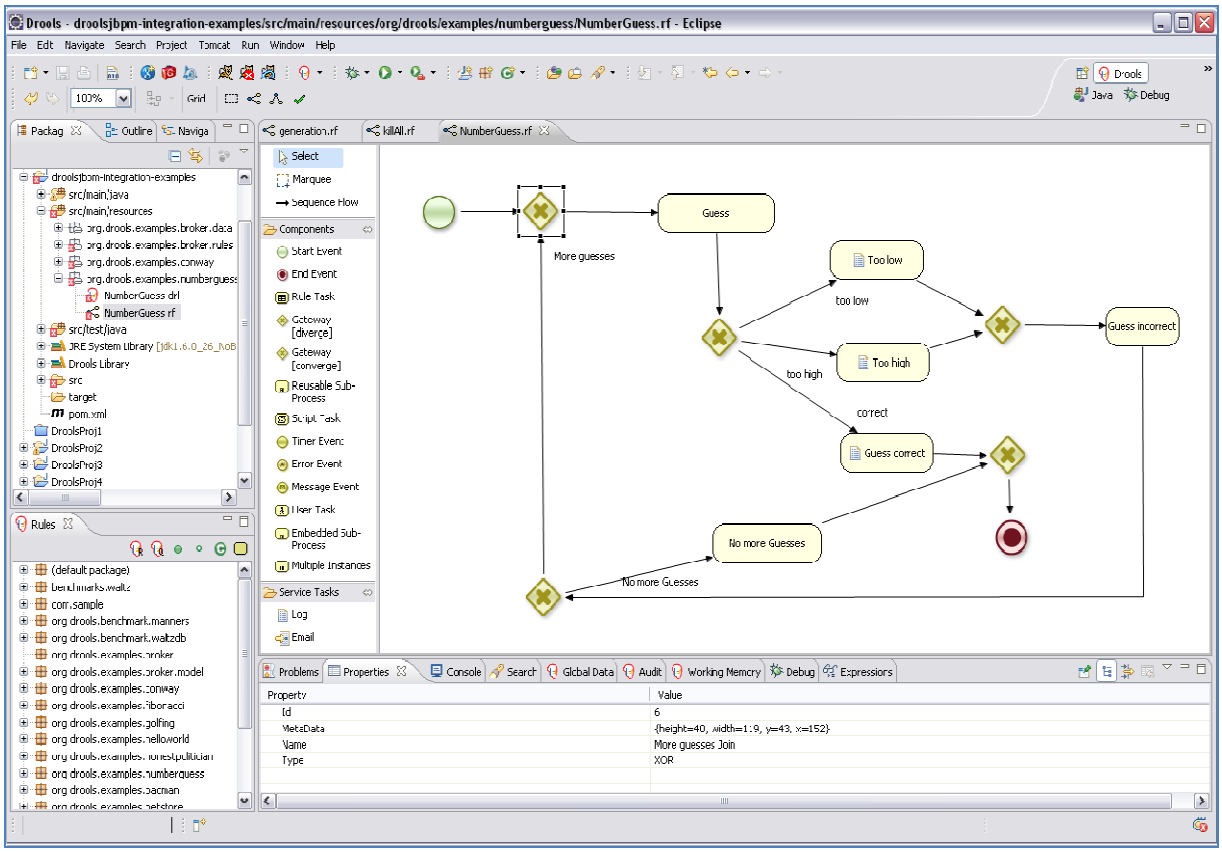


Figure 21 : jBPM Flow

4.4.3.3. Taxi Use Case (SMS arrival)

Through the Drools Fusion development of the Taxi use case, we could confirm that there is still a gap between a rule engine, even integrating temporal reasoning, and a native event engine.

To design the Taxi use case, the graphic modeler has not been used for the following reasons:

- Rule tasks[4], script tasks, and other components available in the rule flow, are sequenced as

[4] Include one or more rules

a typical process flow, where the resulting service is a sequence of ordered operations. The taxi use case, as a typical event flow, is not of that kind. It is designed using tasks and links as a process flow (see **Erreur ! Source du renvoi introuvable.**), but most of operations are parallelized.

- The rule flows does start on an event, but events occurring during the process can not be just modeled. Even using the few events available in the palette[5], the incoming events as the rule that will listen to it, should be fully coded in the rules files. Thus, except for some actions isolated (as logging), the rule flows is more descriptive than executable.

Consequently we implemented the taxi use case without jBPM, using Java code and rule packages. Above the corresponding resources we had to develop:

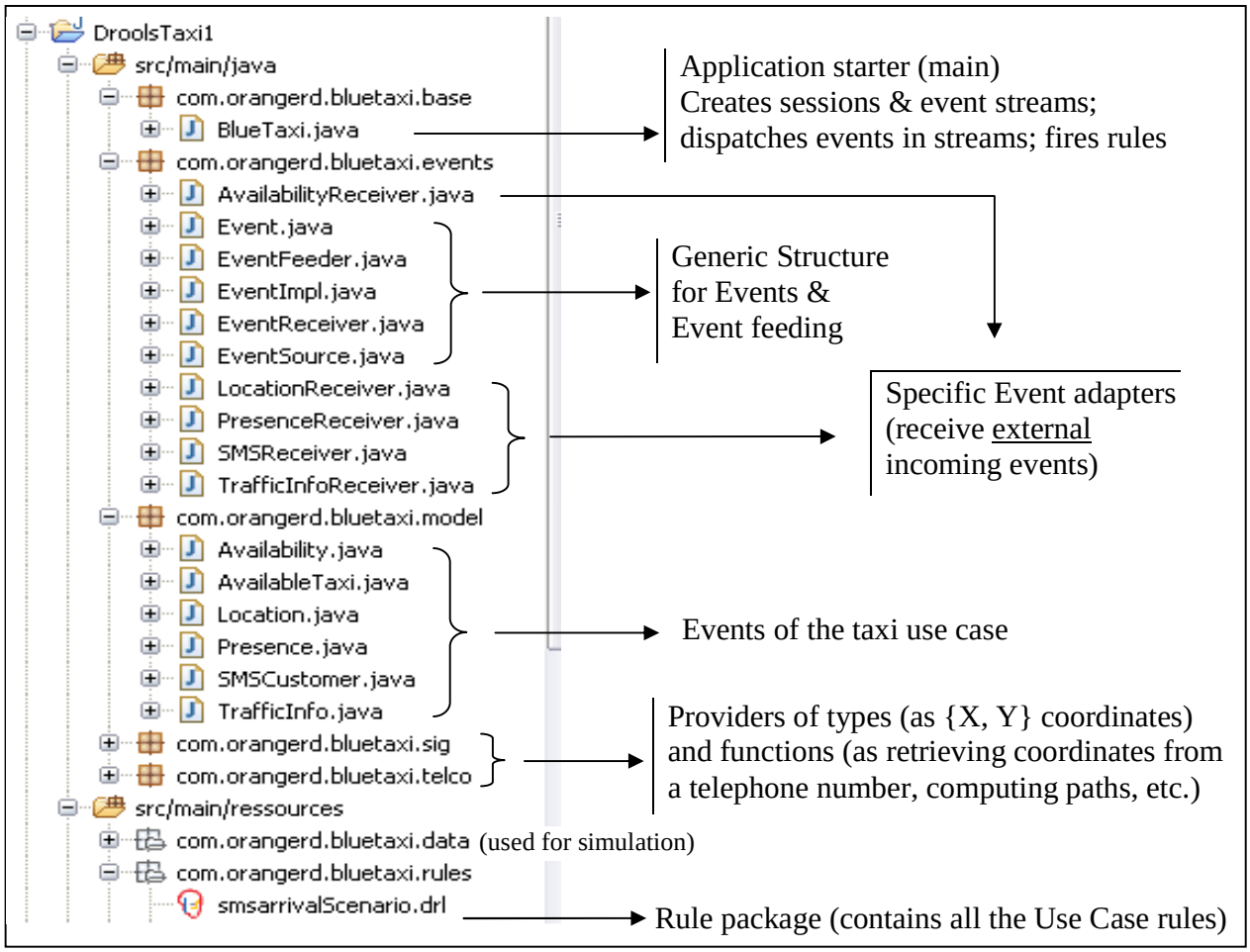


Figure 22: Taxi Use Case structure with Drools

[5] Signal Event, Timer Event, Error Event

The *main* Class (in BlueTaxi), instantiates Blue Taxi and

- Instantiates the event sinks (for external incoming events):

```
LocationReceiver locationSource = new LocationReceiver();
PresenceReceiver presenceSource = new PresenceReceiver();
        ⋮                ⋮                      ⋮
```

- Instantiates the corresponding feeding structures and start feeding:

```
EventFeeder locationFeeder = new EventFeeder(clock, locationSource, blueTaxi, 3000 );
locationFeeder.feed();

EventFeeder presenceFeeder = new EventFeeder(clock, presenceSource, blueTaxi, 3000 );
presenceFeeder.feed();
      ⋮
```

BlueTaxi (implements *EventReceiver*, the interface for internal incoming events)

- Declare the resources files:

```
    private static final String[] ASSET_FILES = {
//          "/com/orangerd/bluetaxi/rules/smsarrivalScenario.rf",
        "/com/orangerd/bluetaxi/rules/smsarrivalScenario.drl"};
```

- Load and packages resources files, prepare and instantiates the session, fires rules, and define Stream Entry Point for the current session:

```
KnowledgeBuilder builder = KnowledgeBuilderFactory.newKnowledgeBuilder();
for( int i = 0; i < ASSET_FILES.length; i++ ) {
    builder.add( ResourceFactory.newInputStreamResource(
                        BlueTaxi.class.getResourceAsStream( ASSET_FILES[i] ) ),
                 ResourceType.determineResourceType( ASSET_FILES[i] )
                );
}
KnowledgeBaseConfiguration conf = KnowledgeBaseFactory.newKnowledgeBaseConfiguration();
conf.setOption( EventProcessingOption.STREAM );
KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase( "BlueTaxi", conf );
kbase.addKnowledgePackages( builder.getKnowledgePackages() );
StatefulKnowledgeSession session = kbase.newStatefulKnowledgeSession();

session.fireAllRules();

this.locationStream = this.session.getWorkingMemoryEntryPoint( "Location stream" );
this.presenceStream = this.session.getWorkingMemoryEntryPoint( "Presence stream" );
```

- Receive and dispatch incoming events, using java reflection through a generic Event type:

```
public void receive(Event<?> event) {
    try {
        if (event.getObject() instanceof Location) {
            Location location = ((Event<Location>) event).getObject();
            this.locationStream.insert( location );
        } else if (event.getObject() instanceof Presence) {
            ...
```

The *generic event structure*, are Interfaces and Classes to support engine event feeding.

- Interface *Event* and its concrete class *EventImpl* provide a default implementation for events:

```
public class EventImpl<T> implements Event<T> {
    private final long timestamp;
    private final T object;

    public EventImpl(long timestamp, T object) {
        super();
        this.timestamp = timestamp;
        this.object = object;
    }
```

- Events defined in package model of the project will “inherit” the event template above, or overwrite its method, without using inheritance mechanisms.

- Dedicated *Event receivers* instantiate the typed event (here the *Presence* one)

```
public class PresenceReceiver implements EventSource {
    private long baseTimestamp;
    private Event<Presence> next;
    private PresenceGenerator generator;
                :
                :
                :

    @Override
    public Event<?> getNext() {
        return next;
    }

    @Override
    public boolean hasNext() {
        try {
            Presence pres = generator.generateNewPresence(); // presence simulator
            next = new EventImpl<Presence>( pres.getTimestamp(), pres);
            return true;
        } catch ( Exception e ) { // nothing to do, return false
            e.printStackTrace();
        }
        return false;
    }
```

- As we saw in the main class, the *EventFeeder*.feed() method is the first to be called. It feeds the initial context with the first event from each type:

```
public void feed() {
    if ( source.hasNext() ) {
        Event< ? > event = source.getNext();
        FeedContext context = new FeedContext( event );
        FeedTrigger trigger = new FeedTrigger();
        trigger.setNextFireTime( event.getDate() );
        FeedJob job = new FeedJob( source, sink, trigger, clock );
        clock.scheduleJob( job,
                           context,
                           trigger );
    }
}
```

- Then the execute() method (from *FeedJob*, an *EventFeeder* inner class) will bed called automatically, reproducing the scheduling mechanism repetitively.
  - The *BlueTaxi* receive() method is invoked for dispatching events to the proper stream
  - Then, from the next incoming event, a new scheduling for feeding is set.

```
public void execute(JobContext context) {
    this.sink.receive( ((FeedContext) context).event );
    if ( this.source.hasNext() ) {
        ((FeedContext) context).setEvent( this.source.getNext() );
        this.trigger.setNextFireTime( ((FeedContext) context).getEvent().getDate() );
        clock.scheduleJob( this,
                           context,
                           trigger );
```

The *Rules* (all stored in smsarrivalScenario.drl)

- Import the objects (events, types, etc) that will be manipulated in the rule package

```
import com.orangerd.bluetaxi.model.Location;
import com.orangerd.bluetaxi.model.Presence;
                                        ⋮
import com.orangerd.bluetaxi.sig.Point
                                        ⋮
```

- Define global classes in order to make their methods available in the rule package

```
global com.orangerd.bluetaxi.telco.TelcoSupplier telcoServices
global com.orangerd.bluetaxi.sig.SIGPartner sigServices
```

- Declare events (special type of facts), and the dialect used (mvel by default)

```
declare Location
    @role( event )
end
```

```
dialect "java"
```

- Declare the rules (here the one retrieving customer location when an SMS event occurs, and the one producing an new available taxi event)

```
rule "Customer Location retrival"
    when
        $cus: SMSCustomer($tel : telnb, $name : name)
    then
        telcoServices.setNewLocationFromTel($tel, $name);
end

rule "Taxi Presence and Availability"
    when
        $pres: Presence($name : name, $ts : timestamp) from entry-point "Presence stream"
        $avail: Availability(name == $name, available == true) from entry-point "Availability stream"
    then
        with( avtaxi  = new AvailableTaxi() ) {
            name = $name,
            timestamp = $ts
        }
        entryPoints["Available Taxi stream"].insert( avtaxi );
end
```

4.4.3.4. Runtime Architecture, Deployment, Admin tools

Drools runtime consists in a collection of Java libraries. Different runtimes can be configured depends on the needs, embedding needed Jars for the application.

Drools applications can be deploy on Linux, Mac or Windows Systems.

Drools implementation of the ReteOO algorithm supports coarse grained parallelization. It does by partitioning runtime components containing packages of rules, and having a pool of worker threads to propagate events through the partitions. All working memory actions (insert/retract/modify) are executed asynchronously.

A JMX interface is available for monitoring sessions. Monitoring information can be accessed through the dedicated console above:

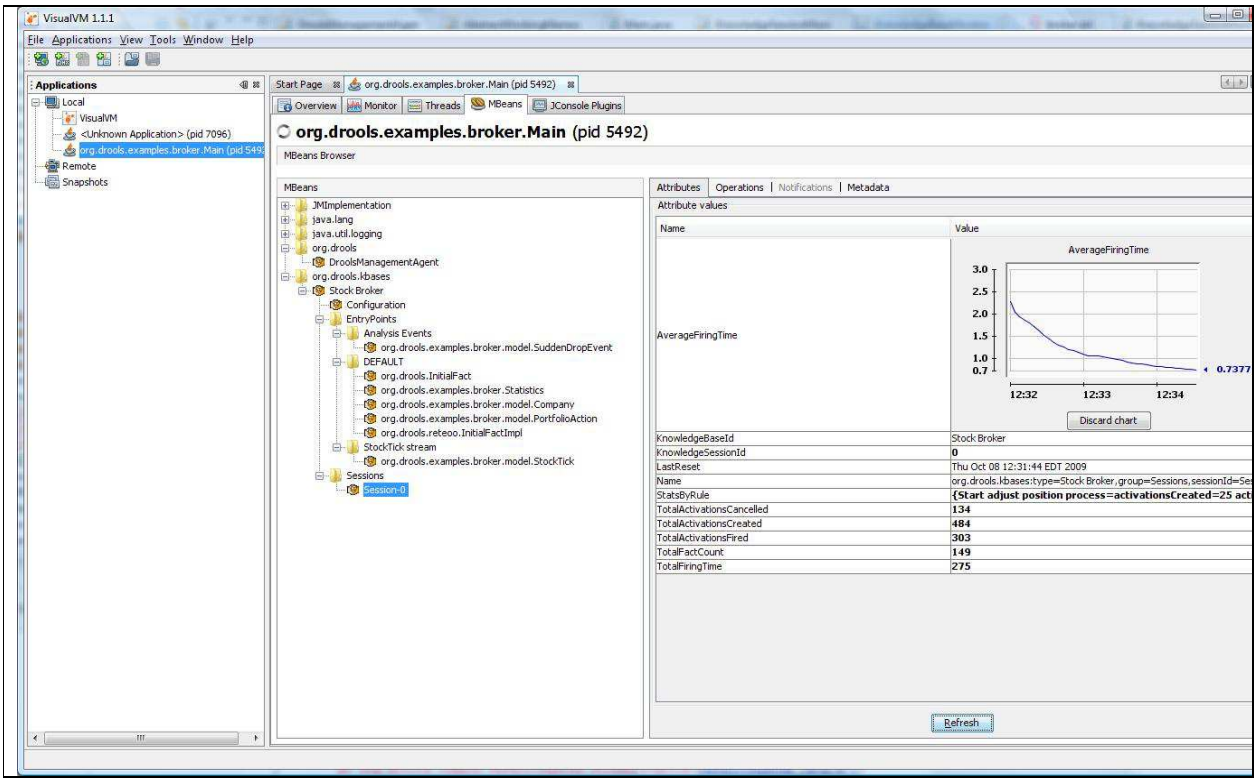


Figure 23: Knowledge Session stats

4.4.3.5. Standards and interoperability

The Java based software Drools has its API in Java, which is obviously a facility for integration.

Drools supports Spring which allow XML configuration of sessions and other features:

- *JPA Persistence*, which allow relational/object mapping toward databases,
- Execution nodes based on *Camel* framework provide session routing through multiple protocols,
- Batch scripts startup that can also be set up though Spring files.

4.4.4. Product Environment criteria

4.4.4.1. Roadmap

Drools software does not display a roadmap for the future releases contents. Only some selective information is available on Drools blog. From it, we can report an enhancement in the process: the integration for jBPM with the *SwitchYard* component, a *service delivery framework providing full lifecycle support for developing, deploying, and managing service-oriented applications*. As a result, the process modeling capabilities should enhance with “true services” inclusion, in the graphical design up at least to deployment.

4.4.4.2. Licensing

Drools is an open source software, released under the Apache License V2.0, which means liberal conditions, including redistribution and modifications.

4.4.4.3. Community

JBoss Drools has a large community of members and developers. A wiki and a blog are available, with a reasonable activity. Forums (and Mailing lists) are also offered, where Drool’s team, as well as members, provides valuable support. Documentation covers all tools modules, though more details or deeper explanation would sometimes be welcome. A Javadoc is also provided, but the API description is not complete.
A set of code samples is available with downloaded software, allowing better understanding. Though, several classes and primitives used in those samples are part of the lacking API description, thus, they have to be understood through the samples themselves.

5. CEP Tool requirements (SocEDA)

This section describes the expected requirements from the Soceda platform point of view. We are looking for a tool to graphically design event driven application and to allow designing rules in order to feed distributed CEP engines. Options would be a full IDE with monitoring, debugging, deployment and running aspects. The following list gathers informally these requirements.

5.1. Functional requirements

5.1.1. Main features

As described in section 3 we know the expected CEP tool features for the purpose of the final SocEDA platform. Basically we are looking for a CEP engine with:

- time notions (time-length, time-ordering)
- event retention possibilities
- grouping, aggregation, sorting, filtering and merging event stream
- subqueries possibility
- update, insert and delete operations
- event pattern matching
- input/output adapters

The CEP editor will easily feed the engine with rules and will deliver the following features:

- Drag and drop user friendly interface to draw rules,
- Create, edit and delete deployed rules,
- Deploy/undeploy rules,
- Debug CEP rules and traces the reasoning.

5.1.2. Technical criteria

5.1.2.1. Event Processing features

In terms of event processing feature we already have mentioned the expected one in the previous section 5.1. Those features are quite common to all CEP engines. When we look at the products on the market, they almost all offer the same capabilities. The editor and IDE are the main difference between the tools.

5.1.2.2. CEP Tooling around the CEP (SDK, Editors)

The design tool should contain a palette to easily create rules and complex event. We keep in mind that the final SocEDA solution does also contain a monitoring tool as well as a governance tool, so we will check for that as well even if this state of the art mainly focus on the CEP edition. The editor has to contain a business user view as well as a developer view. The rules could then be produced from the edition tool but also from scratch in text mode.

5.1.2.3. Runtime Architecture, Deployment, Admin tools

The SocEDA platform is based on a distributed CEP engine, so the CEP editor will have to manage this aspect at the deploy time and runtime. Of course a monitoring tool to manage this distributed aspect would be appreciated, as well as a simulation tool, testing tool and deployment tool.

5.1.2.4. Standards and interoperability

For the moment there is no real standard approved by the CEP community. There are some tentative such like StreamSQL from StreamBase, Oracle and the Stanford university tried to create an EPL from the SQL bases.

5.1.2.5. Support community

We expect a tool supported by a large and effective community. It would be a certain guarantee for product progressiveness, for the requests to be addressed and for the component library enrichment.

5.1.2.6. Technical documentation

A technical documentation structured, complete and offering plenty of samples would be appreciated. Though, a lack on this domain can be compensated, if the tool is intuitive and if the community is responsive and productive.

5.1.2.7. License

Permissive open source licenses are preferred, though, other types may be considered until a free access to the software is possible.

6. Conclusion

This document illustrates a current state of the art in terms of Complex Event Processing solutions. At least we gather an evaluation of the most popular and used event processing engine. We still have to define the project requirements from the need of our use cases which will be define a bit later in the project. The definition of the use cases and the requirements plus this state of the art document will make it easier to select the best product that will be embedded in the final platform.

Complex Event processing is taking more and more attention in the IT community. The current CEP solutions have been based on technical and formal languages targeting the developer's communities. We would like to turn this aspect as easy as possible to be handled by business experts. The idea would be to add a CEP design layer on top of this complexity to leverage the power of events combined with the knowledge of business experts, making it easier to create event driven applications.

Through our tentative to implement our use case with the different solutions we can say that there are some commercial solutions that seem answering a part of our needs (such like Streambase). Unfortunately, we would still need to extend one of those solutions in order to completely fit our needs which is not possible as those products are under commercial licenses (without any possibility of source code exploitation).

This last assessment means that we will have to implement this design aspect. In order to do so, we should go in the open source solutions direction. The best product would be an open solution with a large community and strong documentation. Those aspects would help and make it easier to extend the product.

7. Relevant standards

7.1. StreamSQL

StreamSQL isn't recognized as a standard by any consortium or any community, but it has been though to become the event processing language standard. StreamSQL is a query language extending the well known SQL language. It brings the ability to process real-time data streams. SQL has been though to manipulate relations between stored data when StreamSQL adds the ability to manipulate streams. The main arguments to turn this language as a standard are:

- A familiar, standard paradigm,
- Querying over time windows,
- Powerful operators,
- Customization and extensibility.

7.1.1. Technical details

StreamSQL extends the SQL language to support streams in addition to tables. Several new operations have been added to manipulate streams. As described in the Wikipedia definition, below are the StreamSQL technical features:

- ***Selecting from a stream*** - A standard SELECT statement can be issued against a stream to calculate functions (using the target list) or filter out unwanted tuples (using a WHERE clause). The result will be a new stream.
- ***Stream-Relation*** Join - A stream can be joined with a relation to produce a new stream. Each tuple on the stream is joined with the current value of the relation based on a predicate to produce 0 or more tuples.
- ***Union and Merge*** - Two or more streams can be combined by unioning or merging them. Unioning combines tuples in strict FIFO order. Merging is more deterministic, combining streams according to a sort key.
- ***Windowing and Aggregation*** - A stream can be windowed to create finite sets of tuples. For example, a window of size 5 minutes would contain all the tuples in a given 5 minute period. Window definitions can allow complex selections of messages, based on tuple field values. Once a finite batch of tuples is created, analytics such as count, average, max, etc, can be applied.
- ***Windowing and Joining*** - A pair of streams can also be windowed and then joined together. Tuples within the join windows will combine to create resulting tuples if they fulfill the predicate.

7.2. ESPER EPL Language

EPL that stands for Event Processing Language isn't a standard but this SQL like language brings the ability to define complex event rules. It is very close to SQL with the SELECT, FROM, WHERE, GROUP BY, HAVING and ORDER BY clauses where stream could be compared to SQL tables and events compared to SQL rows. The language brings CEP aspects such like:

- Event window, pattern matching, temporal notions and stream aggregation.

7.2.1. Sample

In this scenario we need to use a current customer position, taxis positions and presence in order to find the closest taxi. We calculate the flying distance between the customer and all available taxis. We need two windows to store the current taxis position and presence. We also create a window to store the customer query including the selected taxi.

7.2.1.1. Create the taxis presence window

CREATE WINDOW StoredTaxiPresenceEvents.std:groupwin(phoneNumber).win:length(1) (phoneNumber string,status boolean

7.2.1.2. Create the taxis last position window

CREATE WINDOW StoredTaxiPositionEvents.std:groupwin(phoneNumber).win:length(1) (phoneNumber string,latitude double,longitude double)

7.2.1.3. Create the customer request window

CREATE WINDOW StoredCustomersRequests.std:groupwin(taxiId).win:length(1) (customerNumber string,latitude double,longitude double,taxiId string,distance double)

7.2.1.4. Store the taxis presence

We have to update those windows in order to store the data.

INSERT INTO StoredTaxiPresenceEvents SELECT phoneNumber,status FROM PresenceEvent where userType='Driver'

7.2.1.5. Store the taxis last position

INSERT INTO StoredTaxiPositionEvents SELECT phoneNumber,latitude ,longitude FROM GeolocationEvent where userType='Driver'

7.2.1.6. Handle incoming customer queries

Using the previous stored data (taxi position and presence) and the incoming customer query (including his current location), we can calculate the distance between the customer and all taxis. We use the closest one (ordered by distance and we select the first one).

```
INSERT INTO StoredCustomersRequests (customerNumber,latitude,longitude,taxiId,distance)
SELECT Customer.customerNumber, Customer.latitude, Customer.longitude, TaxiPresence.phoneNumber,
(6378137 * Math.sqrt(Math.pow((Math.toRadians(StoredTaxiPositionEvents.latitude) -
Math.toRadians(Customer.latitude)), 2) + Math.pow((Math.toRadians(StoredTaxiPositionEvents.longitude) -
Math.toRadians(Customer.longitude)), 2))) as distance
FROM StoredTaxiPresenceEvents as TaxiPresence,StoredTaxiPositionEvents,
SMSCustomerAlert.win:length(1) as Customer
WHERE StoredTaxiPositionEvents.phoneNumber=TaxiPresence.phoneNumber and TaxiPresence.status=true
and TaxiPresence.phoneNumber not in (select taxiId from StoredCustomersRequests) and
Customer.customerNumber not in (select customerNumber from StoredCustomersRequests) order by distance
asc limit 1
```

7.2.1.6.1. Details

```
INSERT INTO StoredCustomersRequests (customerNumber,latitude,longitude,taxiId,distance)
```

we do insert the customerNumber, latitude, longitude, taxiId and distance from the result of the following select query

```
SELECT Customer.customerNumber, Customer.latitude, Customer.longitude, TaxiPresence.phoneNumber,
(6378137 * Math.sqrt(Math.pow((Math.toRadians(StoredTaxiPositionEvents.latitude) -
Math.toRadians(Customer.latitude)), 2) + Math.pow((Math.toRadians(StoredTaxiPositionEvents.longitude) -
Math.toRadians(Customer.longitude)), 2))) as distance
FROM StoredTaxiPresenceEvents as TaxiPresence,StoredTaxiPositionEvents,
SMSCustomerAlert.win:length(1) as Customer
WHERE StoredTaxiPositionEvents.phoneNumber=TaxiPresence.phoneNumber and TaxiPresence.status=true
```

we do select the customer Number, latitude, longitude, taxi phone Number and the compute distance from the customer request (SMSCustomerAlert.win:length(1)) and all the available taxis (StoredTaxiPositionEvents.phoneNumber=TaxiPresence.phoneNumber and TaxiPresence.status=true)

```
TaxiPresence.phoneNumber not in (select taxiId from StoredCustomersRequests) and
Customer.customerNumber not in (select customerNumber from StoredCustomersRequests) order by distance
asc limit 1
```

we are only interested by taxi that are not vurrently on a fare (TaxiPresence.phoneNumber not in (select taxiId from StoredCustomersRequests))

and customer that do not have already a processing request stored (Customer.customerNumber not in (select customerNumber from StoredCustomersRequests))
Finally we are only interested by the closest one, so we do order by distance and return the first one (order by distance asc limit 1)

8. References

8.1. Bibliography

[1] Progress software http://web.progress.com/en/apama/event-processing-platform.html
[2] StreamBase Systems, StreamSQL online documentation - http://streambase.com/developers/docs/latest/streamsql/index.html - 2007
[3] Codehaus.org, Esper online documentation set - http://esper.codehaus.org/tutorials/tutorials.html - 2007
[4] Drools http://www.jboss.org/drools/drools-fusion.html
[5] Aleri http://www.sybase.com/products/financialservicessolutions/complex-event-processing
[6] Towards a Streaming SQL Standard – by Namit Jain, Shailendra Mishra, Anand Srinivasan, Johannes Gehrke, Jennifer Widom, Hari Balakrishnan, Uğur Çetintemel, Mitch Cherniack, Richard Tibbetts, Stan Zdonik - Proceedings of the VLDB Endowment VLDB Volume 1 Issue 2, August 2008

[7] Complex Event Processing for Operational Intelligence - A Vitria Technical White Paper

[8] Complex Event Processing - Lokhi Prasad Deori - Integration Practice – August 2010

[9] A Graphical Editor For Complex Event Pattern Generation - Sinan Sen, Nenad Stojanovic, Ruofeng Lin – 2009

[10] streamSQL technical details – Wikipedia

[11] Event Processing Glossary 1.1 - David Luckham, Roy Schulte – July 2008

[12] Event Processing In Action,Maning Publications co. – David Luckham – August 2010

9. Illustration table

10. Appendix

10.1. CEP tools list

Name	Description	URL
Esper	Esper is an entirely free open-source component available under the GNU GPL license (GPL also known as GPL v2). Esper can hook into any Java based system as a message consumer	http://esper.codehaus.org/
Apama	Proven in the most demanding of environments, the Progress® Apama® Event Processing Platform monitors rapidly moving event streams, detects and analyzes important patterns, and acts - in sub-millisecond time. With Apama, business events can be correlated and analyzed across multiple data streams in real-time, providing new levels of decision making that dramatically improves business precision.	http://web.progress.com/en/apama/
Drools Fusion	Fusion is a module of the open source Drools platform, which extends the capabilities of the rule engine to integrate temporal features. Thus, Drools enhanced by Fusion become a Complex Event Processing platform.	http://www.jboss.org/drools/drools-fusion.html
Aleri	Sybase Aleri Streaming Platform enables rapid application development and deployment of robust applications that derive insight from streaming event data, empowering instant responses to changing conditions. Sybase Aleri Streaming Platform developers report that using Sybase to build event-driven analytic applications, cut development time and effort by anywhere from 67% to 85%.	http://www.sybase.com/products/financialservicessolutions/complex-event-processing
Oracle CEP	Oracle launched its event-driven architecture suite in 2006 and added BEA's WebLogic Event Server to it in 2008, building what is now called "Oracle CEP", a system that provides real time information flow processing. Oracle CEP uses CQL as its rule definition language, but, similarly to Coral8 and StreamBase, it adds a set of relation-to-relation operators designed to provide pattern detection, including conjunctions, disjunctions, and sequences. An interesting aspect of this pattern language is the possibility for users to program the selection and consumption policies of rules.	http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/index.html

StreamBase	StreamBase [Streambase 2010a] is a software platform that includes a data stream processing system, a set of adapters to gather information from heterogeneous sources, and a developer tool based on Eclipse. It shares many similarities with the Coral8 CEP Engine: in particular, it uses a declarative, SQL-like language for rule specification, called StreamSQL.	http://www.streambase.com/
Tibco Business Events	Tibco Business Events [Tibco 2010] is another widespread complex event processing system. It is mainly designed to support enterprise processes and to integrate existing Tibco products for business process management. To do so, Tibco Business Events exploits the pattern-based language of Rapide, which enables the specification of complex patterns to detect occurrences of events and the definition of actions to automatically react after detection. In-terestingly, the architecture of Tibco Business Events is capable of decentralized processing, by defining a network of event processing agents: each agent is responsible for processing and filtering events coming from its own local scope	http://www.tibco.com/products/business-optimization/complex-event-processing/businessevents/
Event Zero	A similar architecture, based on connected event processing agents, is used inside Event Zero , a suite of products for capturing and processing information items as they come. A key feature of Event Zero is its ability of supporting real-time presentations and analysis for its users.	http://www.eventzero.com/
Etalis	is an open source system for Complex Event Processing with two accompanied languages called: ETALIS Language for Events (ELE) and Event Processing SPARQL (EP-SPARQL)	http://code.google.com/p/etalis/
Padres	Padres [Li and Jacobsen 2005] is an expressive content-based publish-subscribe middleware providing a form of complex event processing. As in traditional publish-subscribe it offers primitives to publish messages and to subscribe to specified information items, using detecting rules expressed in a pattern-based language. Unlike traditional publish-subscribe, expressed rules can involve more than one information item, allowing logic operators, sequences, and iterations. Padres uses a time model with an absolute time, which can be addressed in rules to write complex timing constraints.	http://padres.msrg.utoronto.ca