	SocEDA ***Cloud based platform for large scale social aware EDA*** ANR-10-SEGI-013	

SocEDA

Document name:	CEP Editor Tool Functional Specification
Document version:	**V-1.0**
Task code:	
Deliverable code:	D4.1.2
WP Leader (organization):	OrangeLabs (FT)
Deliverable Leader (organization):	OrangeLabs
Authors (organizations):	OrangeLabs
Date of first version:	September 2011

Table of Contents

1. General description 3
1.1. Context 3
1.1.1. Objectives 3
1.1.2. Target user 3
2. Terms and Concepts definition 4
2.1.1. Acronyms 4
3. General Architecture 5
3.1.1. Main principles 5
3.1.1.1. CEP Editor tool positioning 5
3.1.1.2. Event type repository 7
3.2. Main functionalities 8
3.2.1. Design 8
3.2.1.1. Palette 9
3.2.1.2. Diagram zone 13
3.2.1.3. Input panel 14
3.3. EPL Generation & Validation 15
3.3.1.1. EPL generation 15
3.3.1.2. EPL validation 15
3.4. Export/Import 16
4. Roadmap 17
Illustration table 18
5. References 19
6. Annexes 20
6.1. Sample diagram 20
6.1.1. EPL description 20
6.1.2. Diagram representation 21

1. General description

1.1. Context

1.1.1. Objectives

The SocEDA project attempt to provide an open distributed platform for event-driven interaction between services that scales at the internet level. The platform environment should cover the whole event-driven application life cycle:

- design, deploy and run complex and distributed EPL Statements,
- define complex event patterns,
- enrich existing events.

As described in the Complex Event Processing state of the art document [2] a key issue is to simplify the CEP design aspect with graphical tools. This document is about the CEP Editor which is supposed to facilitate the CEP rules creations in the context of the whole SocEDA platform [3].

In the SocEDA project one of the targeted CEP is the ESPER engine [1], so we will be talking about EPL for event processing language. The main goal is to translate a designed model and a high level definition of conditions into EPL Rules (ESPER-EPL as a first step).

1.1.2. Target user

The idea is to provide a tool to generate EPL rules in a simple way. The user will just need some basic EPL knowledge. He will just have to be able to understand some basic concept such like drag and drop boxes/components and link them together in order to build a diagram.

Then the user will have to fill in some forms in order to specify some details of queries, in the case he wants to create complex queries. Basically the tool will be as simple as possible in order to target lambda users that are at least able to drag and drop and fill in forms.

2. Terms and Concepts definition

2.1.1. Acronyms

EPL	Event Processing Language
CEP	Complex Event Processing
Event type	Event type (event class, event definition, or event schema) represents a class of event objects
DCEP	Distributed Complex Event Processing
IDE	Integrated Development Environment
DiCEP API	Distributed Complex Event Processing API
REST	Representational State Transfer
Servlet	A servlet is a small program that runs on a web server See Java Servlet Application Programming Interface (Sun API)

3. General Architecture

3.1.1. Main principles

In order to make this EPL rule design aspect easier, we would like to provide a drag & drop tool where you can use event types coming from an event type repository.
Basically, it will allow the designing of a small diagram representation of rules in order to push EPL queries within an ESPER instance without having to interact with any IDE.

3.1.1.1. CEP Editor Tool positioning

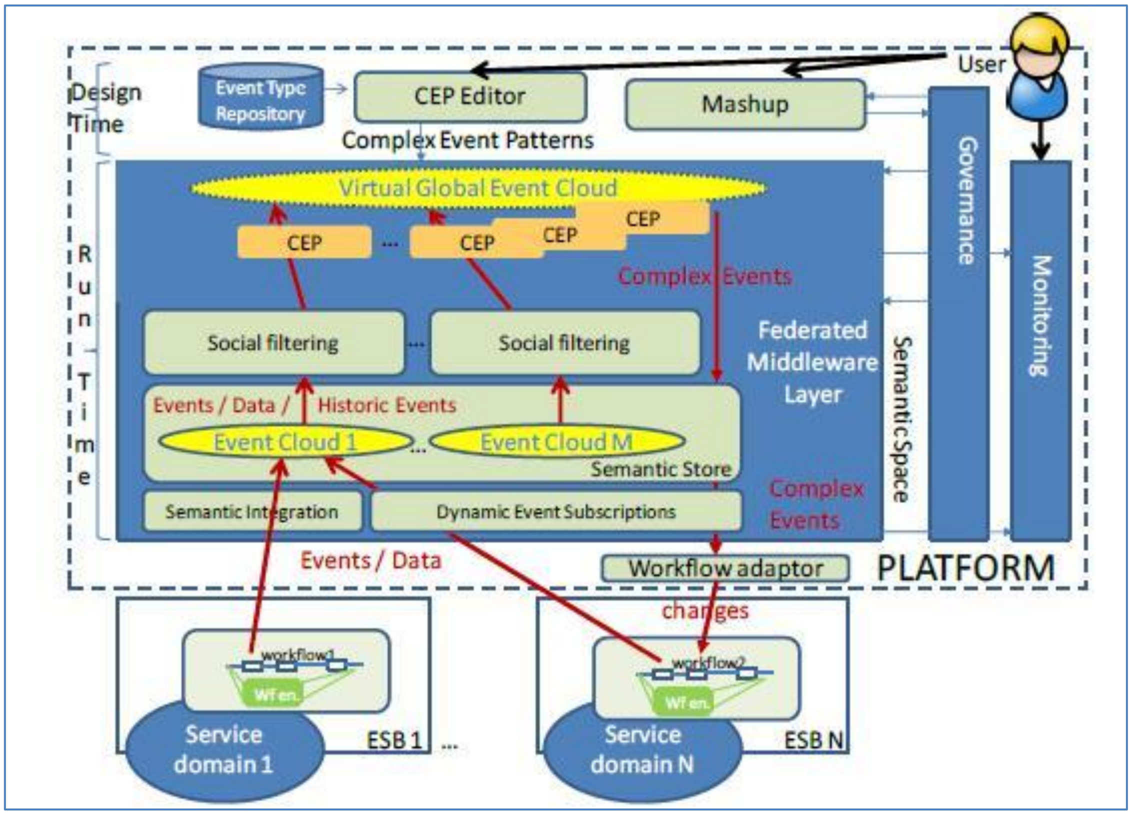
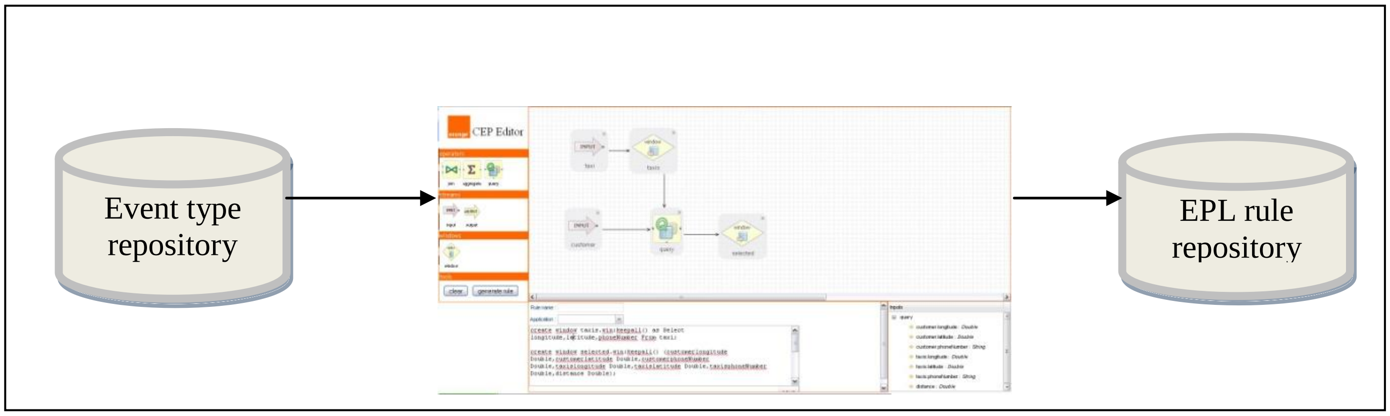


Figure 1: SOCEDA conceptual architecture

The above figure shows the whole conceptual architecture of our project. In this deliverable we mainly focus on the CEP Editor which is between the event type repository and a Virtual Global Event Cloud (VGEC). This VGEC is federating the DCEP environment.

In order to go further on the CEP Editor that we could also name as EPL Editor we will simulate both with a database where we will retrieve the event types and push our generated queries[1].



```
create window taxis.win:keepall() as Select
longitude,latitude,phoneNumber from taxi;

create window selected.win:keepall() (customerlongitude
Double,customerlatitude Double,customerphoneNumber
Double,taxislongitude Double,taxislatitude Double,taxisphoneNumber
Double,distance Double);
```

Figure 2: Input/Output of the EPL editor

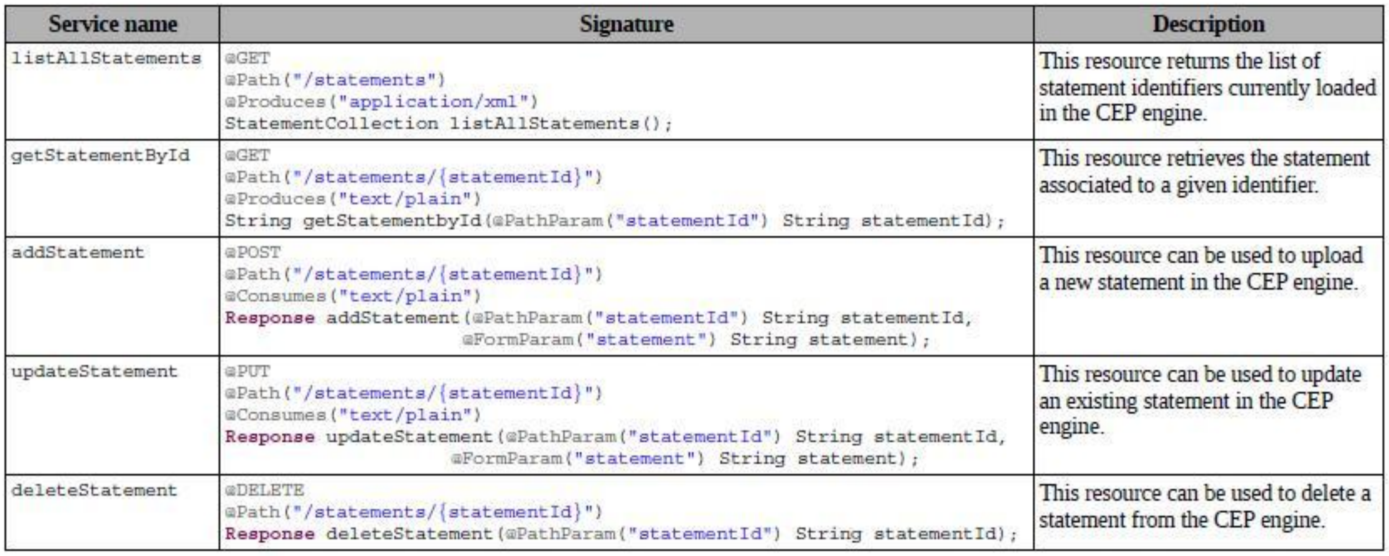
As shown in the above figure, we do simulate an event type repository and an ESPER instance with databases. In the following section we will describe the event type repository database structure.

In order to use those event types in the EPL editor we need to retrieve an event type description. The EPL editor will contain an input stream component where the user will define or select an existing event type. We will then have to retrieve the properties of this event type in order to allow the user to manipulate those properties within the next components of the diagram that are connected to this input stream component.

This first architecture will stand for the first implementation but the idea behind is to get a first stable version that we could bind with the rest of the SocEDA architecture.

For the moment we do not have any DiCEP Administration API or any Event Type Repository API to bind with the EPL Editor, but the next step will be to use them. We have been informed about the future interface on the DiCEP API. It will be a REST API. Those details will not change anything on the client side but the backend will be changed to be plugged to those web services instead of the current databases.

[1] At that stage we do not have any event type repository to retrieve the event type description neither any VGEC



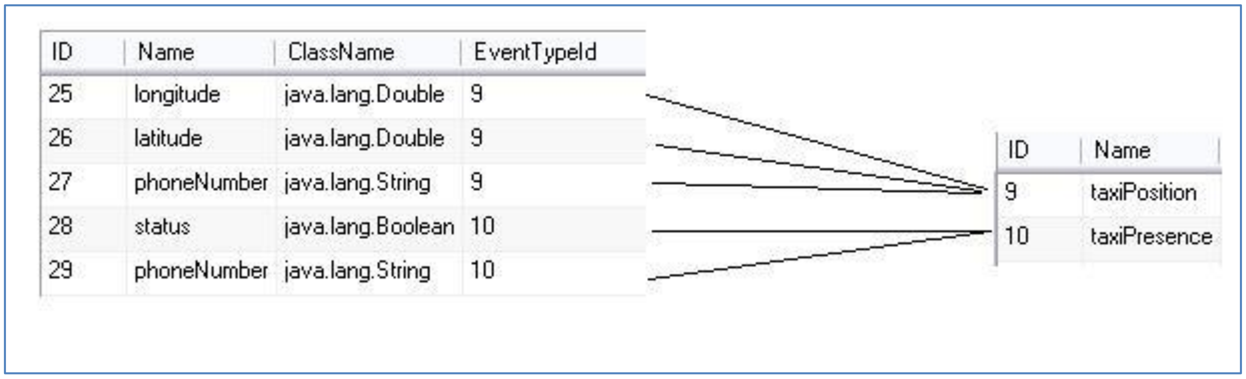
Service name	Signature	Description
`listAllStatements`	`@GET @Path("/statements") @Produces("application/xml") StatementCollection listAllStatements();`	This resource returns the list of statement identifiers currently loaded in the CEP engine.
`getStatementById`	`@GET @Path("/statements/{statementId}") @Produces("text/plain") String getStatementbyId(@PathParam("statementId") String statementId);`	This resource retrieves the statement associated to a given identifier.
`addStatement`	`@POST @Path("/statements/{statementId}") @Consumes("text/plain") Response addStatement(@PathParam("statementId") String statementId, @FormParam("statement") String statement);`	This resource can be used to upload a new statement in the CEP engine.
`updateStatement`	`@PUT @Path("/statements/{statementId}") @Consumes("text/plain") Response updateStatement(@PathParam("statementId") String statementId, @FormParam("statement") String statement);`	This resource can be used to update an existing statement in the CEP engine.
`deleteStatement`	`@DELETE @Path("/statements/{statementId}") Response deleteStatement(@PathParam("statementId") String statementId);`	This resource can be used to delete a statement from the CEP engine.

Figure 3: DiCEP Administration API (REST)

As described in the figure above the DiCEP interface will allow listing all the current statements running on the DCEP side. The API will also be used to push new statement from the EPL Editor to the DCEP through the "addStatement()" method. Those are the basic functions used from the perspective of the CEP Editor but the API is also supposed to provide all the CRUD method in order to update and delete statements.

3.1.1.2. Event type repository

The event type repository will return event type descriptions. An event type does have a name and an ID, but also some properties.
Those event properties do have a name and a className or classType.



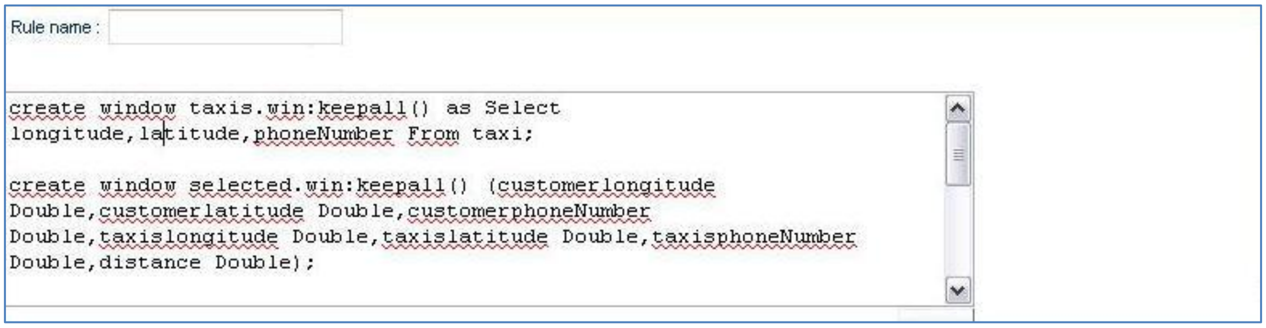
ID	Name	ClassName	EventTypeId
25	longitude	java.lang.Double	9
26	latitude	java.lang.Double	9
27	phoneNumber	java.lang.String	9
28	status	java.lang.Boolean	10
29	phoneNumber	java.lang.String	10

ID	Name
9	taxiPosition
10	taxiPresence

Figure 4 : event type repository representation

In order to simulate the running CEP instance we do use a database to push the generated EPL queries. From a single EPL diagram the CEP Editor could generate more than one query. If the diagram is complex and does contain some window the generated EPL will contain the window creation queries as well as the linked window insertion.

When the user wants to push its generated queries, it does give a name to its queries. Then the EPL editor pushes the queries one by one and does check the existing name ("rule Name" field in the figure below), if there is more than one query generated the EPL editor will generate a name for each rule.



```
create window taxis.win:keepall() as Select
longitude,latitude,phoneNumber From taxi;

create window selected.win:keepall() (customerlongitude
Double,customerlatitude Double,customerphoneNumber
Double,taxislongitude Double,taxislatitude Double,taxisphoneNumber
Double,distance Double);
```

Figure 5: Generated rules scenario

For example if the EPL editor generates 3 queries:

- One window creation
- One window insertion
- One select query

The given name (the one inserted in the "Rule name" field) will be used to store the "select query", then the EPL editor will push the window creation queries with generated names such like "rule name"+_window and the last one will be named "Rule name"+_window1.
This is for insertion purpose but it will be transparent for the user.

3.2. Main functionalities

3.2.1. Design

This design tool will be based on well known concepts:

- Drag and drop box from a palette to a diagram zone,
- Configuration panel to customize and setup the different boxes,
- Arrow connectors to link the boxes,
- Input panel to see the input field coming from the previous linked box,

In the next section we will describe each aspect mentioned above starting with the palette and the diagram zone.

3.2.1.1. Palette

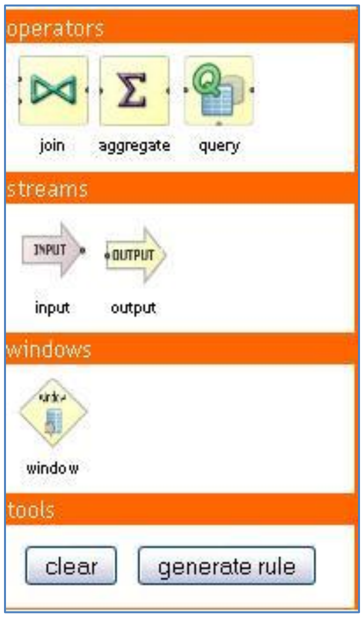
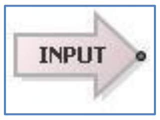


Figure 6 : Palette elements

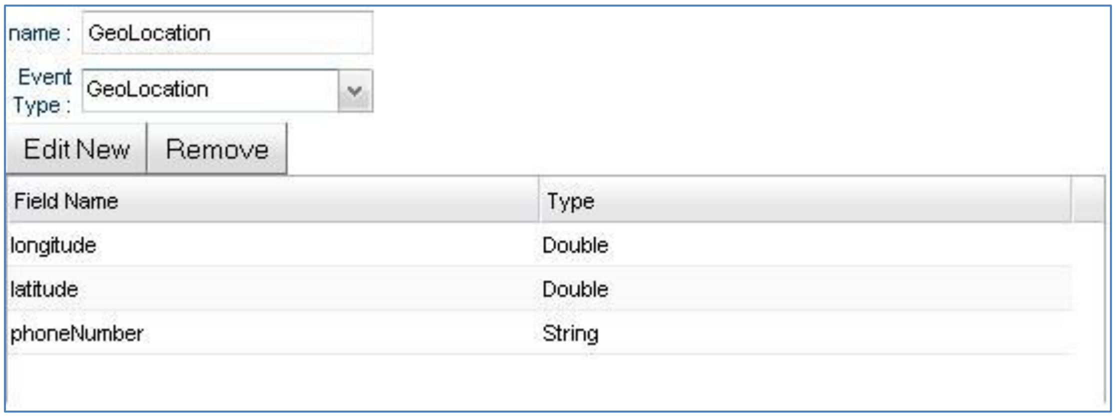
The palette will contain at least the basic element to design a rule. It means:

- input box to specify the type of incoming event associated the design rule,
- output box to specify the field we want to filter in the rule,
- window to specify which field we want to store in a window (epl element used to store data),
- join will be used to merge two input and specify which field will be used to merge them,
- aggregate will be used to make some aggregate operation on some input field,
- query will allow to add some condition on fields (in order to filter), it will be also used to create a new field (making operations using input fields). It will also be used to add some ordering and limit.

3.2.1.1.1. Input box



When we will drop this element we will have the possibility to customize it, specifying which type of element we want to use in this EPL query. To configure this box we will have a bottom panel related to this input element.



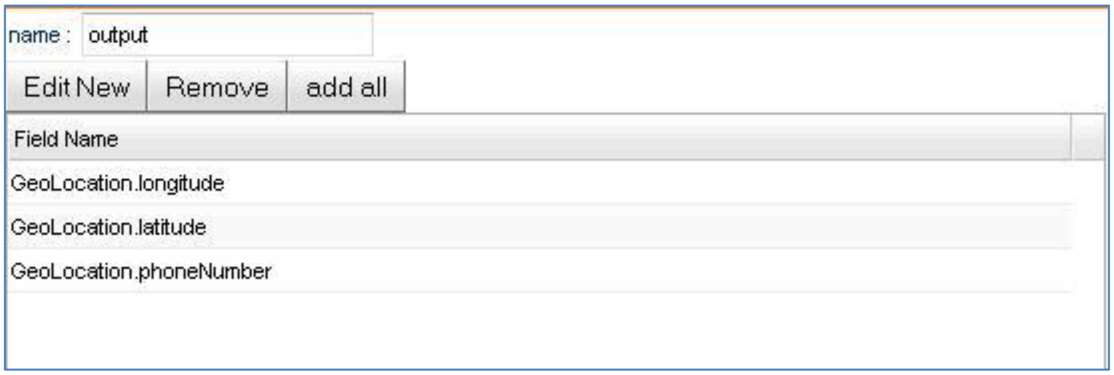
name : GeoLocation
Event Type : GeoLocation

Field Name	Type
longitude	Double
latitude	Double
phoneNumber	String

Figure 7: input bottom panel

3.2.1.1.2. Output box

This palette element is used to specify which field we want to filter in our query. Basically through the associated configuration panel, we will select the fields we want to use, retrieve or select.



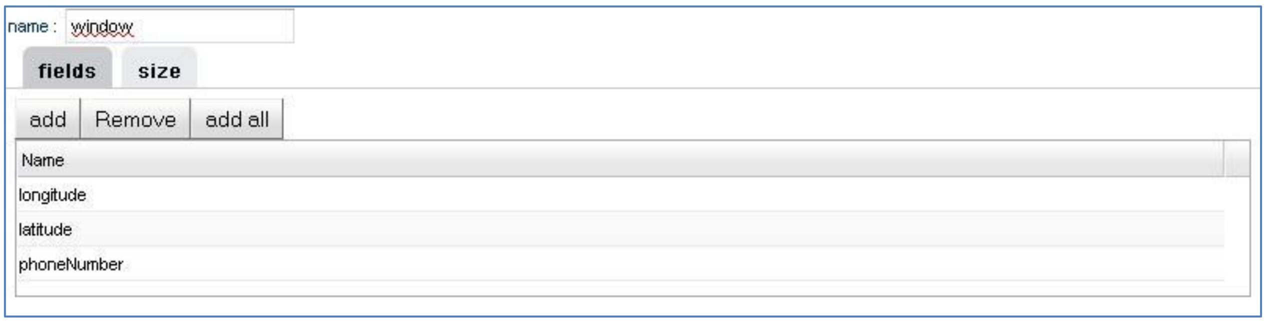
name : output

Field Name
GeoLocation.longitude
GeoLocation.latitude
GeoLocation.phoneNumber

Figure 8: output bottom panel

3.2.1.1.3. Window box

We do use this palette element to setup the window, which means to select the field we want to store in this window but also to setup the size of this window. Of course the previous palette element linked to this box has to be an input box or query outputs.



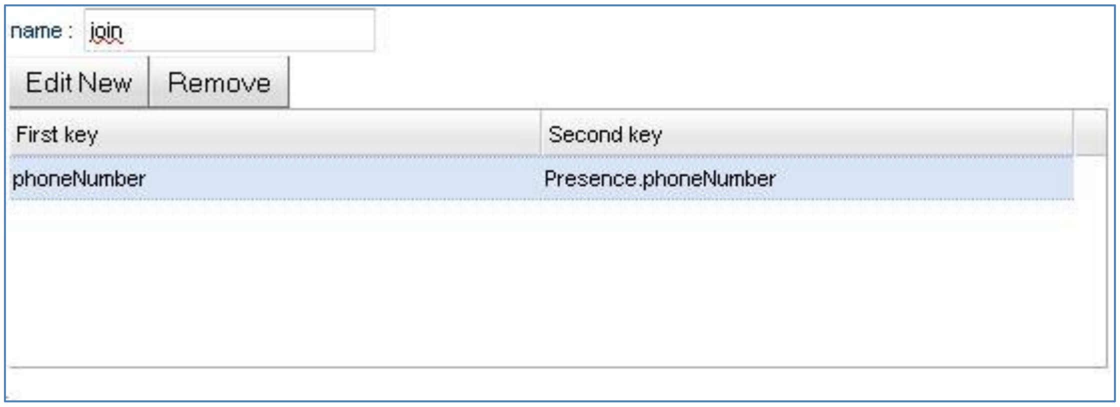
name : window

Name
longitude
latitude
phoneNumber

Figure 9: window bottom panel

3.2.1.1.4. Join box

This palette element will be used when the user wants to merge two different inputs according to two input field values.



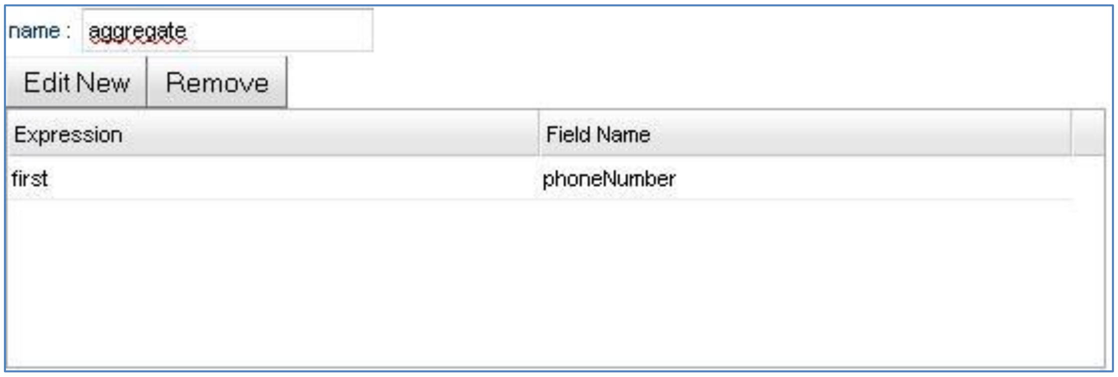
name : join

First key	Second key
phoneNumber	Presence.phoneNumber

Figure 10: join bottom panel

3.2.1.1.5. Aggregate box

This palette element is used to create an aggregation on an input field (e.g. avg, last, first, min , max …). The associated bottom panel will offer the possibility to specify which field we want to aggregate and which aggregate function we want to apply.



name : aggregate

Expression	Field Name
first	phoneNumber

Figure 11: aggregate bottom panel

3.2.1.1.6. Query box

This palette element will be the more complex because it will offer the possibility to add a condition in a rule and also to create a new field from a mix of other fields. It will also offer the possibility to order the result and limit the number of outputs.

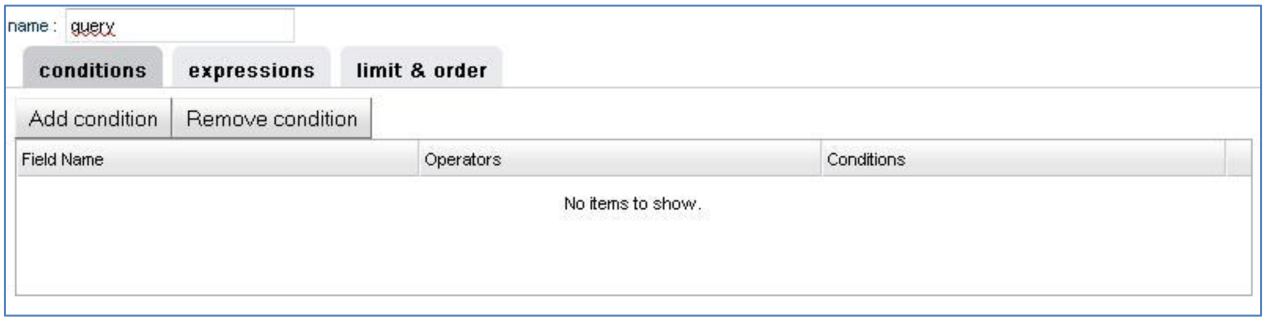


Figure 12: query bottom panel

3.2.1.1.7. The palette tools box

The palette will also contain two buttons:

- One to clear and reset the environment,
- Another one to trigger the EPL generation from the EPL diagram,

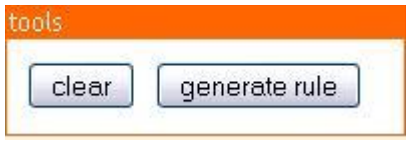


Figure 13: palette toolbox

3.2.1.2. Diagram zone

This part of the tool is the zone where we drop the dragged palette element, in order to design the EPL diagram.

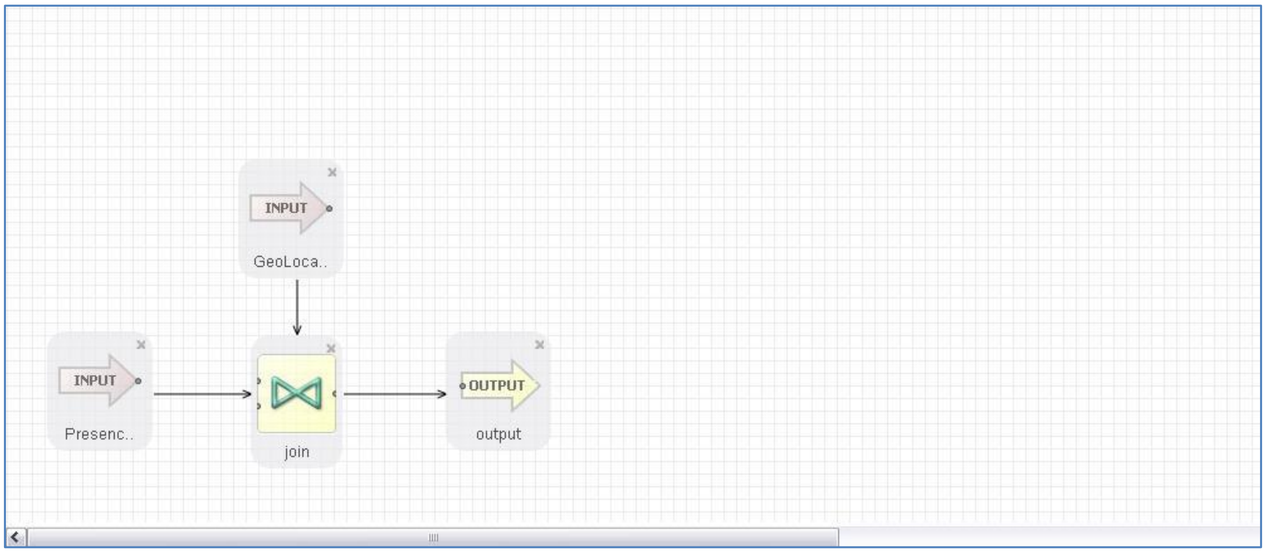


Figure 14: Diagram zone

As shown in the above picture we can also link the EPL elements between them. Using connectors (arrows). The label appears in each bow in order to recognize them. The basic diagram environment should allow:

- Drop elements from the palette
- Link two elements using connectors (light connection validation)
- Delete a connector

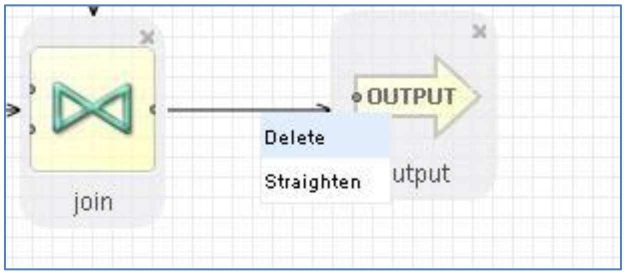


Figure 15: Connector deletion

- Delete an EPL element

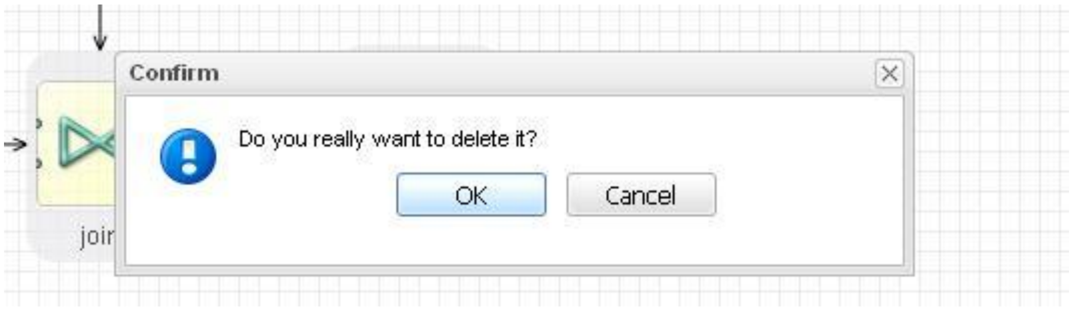
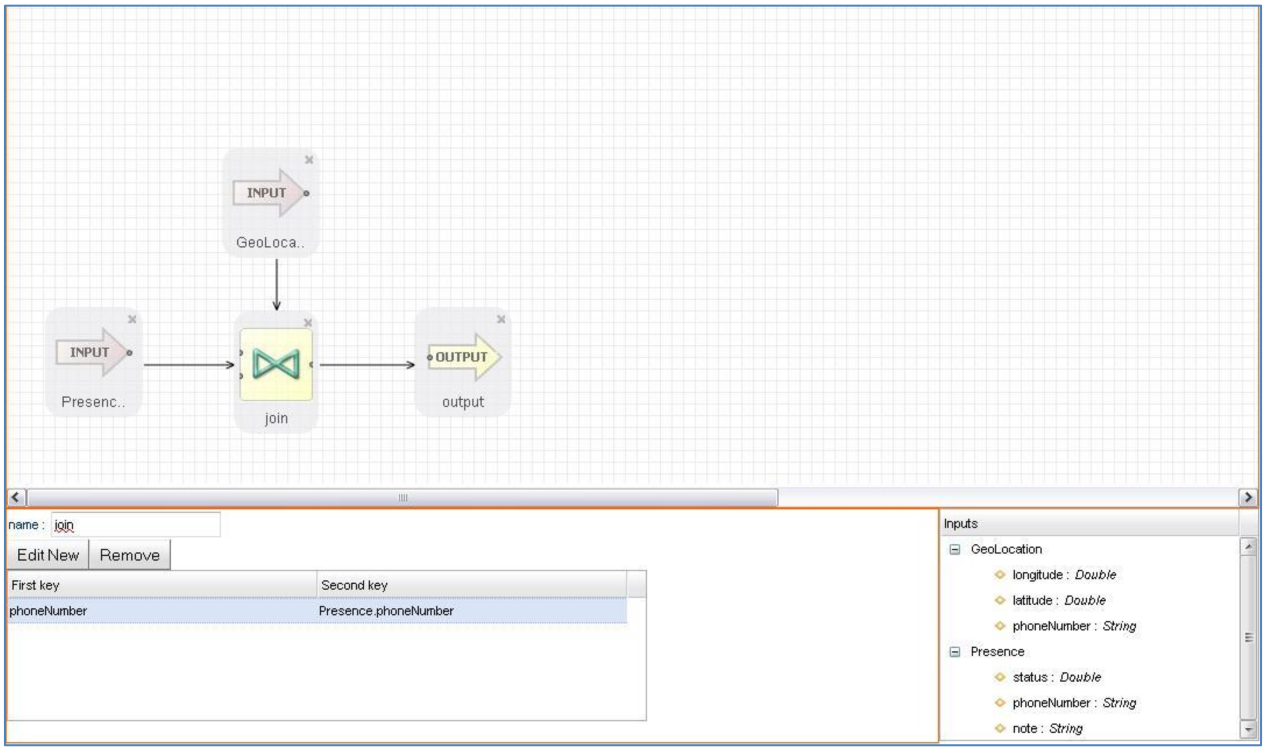


Figure 16: EPL element deletion

- Drag and drop within this zone

3.2.1.3. Input panel

The idea of this panel is to see which fields or accessible in this element. It means when you link an element with another the next element will have access to the first element outputs. In order to make it easier to setup the boxes we will have a panel specifying those inputs



name : join

First key	Second key
phoneNumber	Presence.phoneNumber

Inputs
- GeoLocation
  - longitude : *Double*
  - latitude : *Double*
  - phoneNumber : *String*
- Presence
  - status : *Double*
  - phoneNumber : *String*
  - note : *String*

Figure 17: diagram setup environment

As shown in the above picture you can see on the bottom right corner a small panel that contains the input information of the join element which has access to the outputs of both input elements linked with it.

3.3. EPL Generation & Validation

3.3.1.1. EPL generation

From an EPL diagram the tool is supposed to generate all the needed queries to run the whole diagram scenario. It means when the user click on the “generate rule” button the tool parses the whole diagram and generates the window creations in case it is necessary as well as it generates the insertion to update those windows. It also has to generate the join, aggregation and queries details previously setup in the configuration panels.

3.3.1.2. EPL validation

It would also be interesting if the tool could offer an EPL validation. In order to do so, the EPL editor should communicate with a java Servlet where an ESPER engine is plugged. This ESPER engine will have access to the event type repository in order to be aware of available event types and validate this query aspect. Then when the user clicks on the “validate EPL” button the queries should be sent to this Servlet, with an ESPER instance running behind. Then the error output or the validation message should be sent back to the tool which will advise the user.

The architecture below would be a potential architecture:

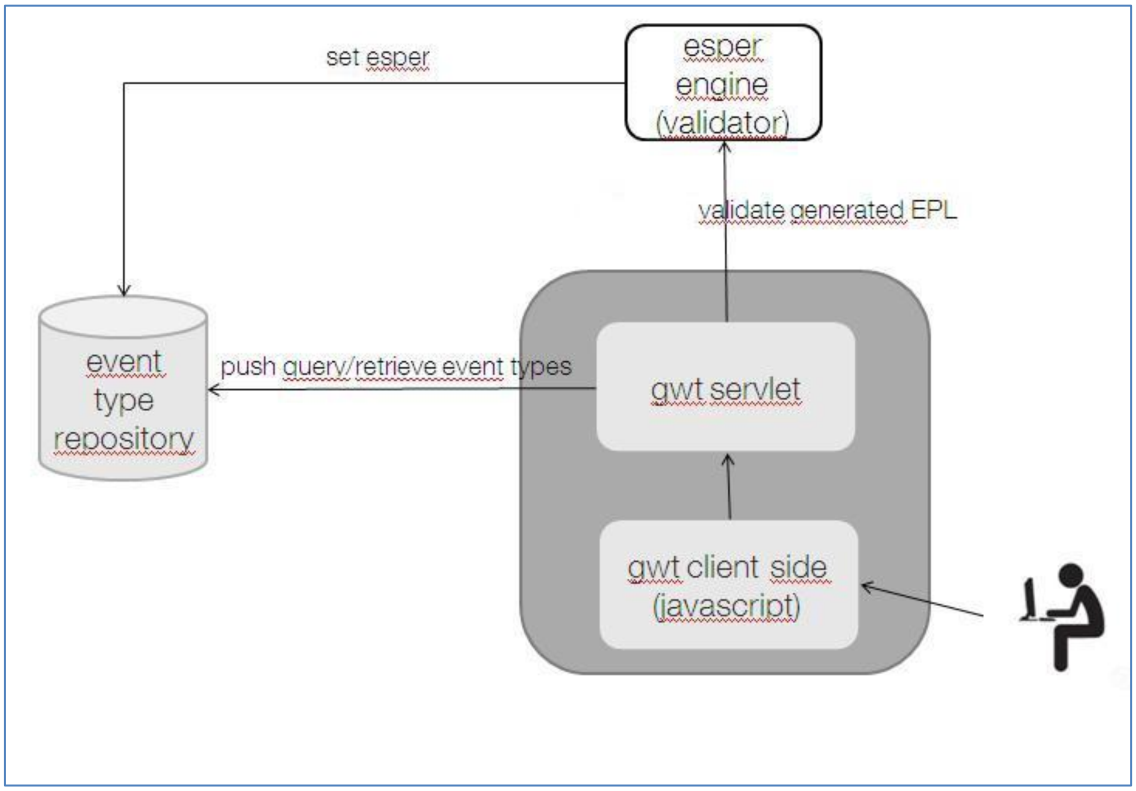


Figure 18: Potential architecture

3.4. Export/Import

This functionality will be used to save EPL diagrams on the local machine. This saving aspect will be also used to store the EPL diagram in the browser database in order to retrieve the last EPL diagram created by the user.

It does not need to use any specific format to serialize the diagram, it will be an xml representation of the EPL diagram but it will also embed the different component setup done by the user.

The different components are linked with a configuration panel used to fill in some properties related to the EPL rule. When we export the diagram, the xml representation has to contain the different component and the link between them but it also has to contain the attach configuration panel. So, the different configuration panel will also be serialized in xml format.

As the tool does not have to be compliant with any existing format, it offers some more flexibility to manage those exports and interpret it in order to manage the import functionality.

4. Roadmap

This tab lists the functionalities that have been considered in this document, and clarifies if there are meant to be implemented or not in the very first release of the CEP Editor tool (V.0).

Functionality	V. 0	Reference / Comment
Base environment and operations: Palette & Flow, drag & drop, components connection, opening properties, etc.	X	3.1.1
EPL generation (basics)	X	3.3.1.1
EPL generation (advanced)		3.3.1.2
Binding with an event type repository	X	3.1.1.2
EPL validation	X	3.3.1.2
Export/import		3.4
Binding with the easierGov[5]		3.1.1.1
Interaction with the Mashup tool [4]		

Illustration table

Figure 1: SOCEDA conceptual architecture 5
Figure 2: Input/Output of the EPL editor 6
Figure 3: DiCEP Administration API (REST) 7
Figure 4 : event type repository representation 7
Figure 5: Generated rules scenario 8
Figure 7: input bottom panel 9
Figure 6 : Palette elements 9
Figure 8: output bottom panel 10
Figure 9: window bottom panel 10
Figure 10: join bottom panel 11
Figure 11: aggregate bottom panel 11
Figure 12: query bottom panel 12
Figure 13: palette toolbox 12
Figure 14: Diagram zone 13
Figure 15: Connector deletion 13
Figure 16: EPL element deletion 14
Figure 17: diagram setup environment 14
Figure 18: Potential architecture 15
Figure 19: EPL diagram 21
Figure 20: Creation of the "distance" field 21
Figure 21: Ordering and limit setup 22
Figure 22: Output form 22

5. References

[1] ESPER http://esper.codehaus.org/

[2] D4.1.1 State of the Art in CEP (Deliverable SocEDA)

[3] D1.2.1 Overall Framework Model (Deliverable SocEDA)

[4] D4.2.1 State of the Art Mashups.docx (Deliverable SocEDA)

[5] Easygov see SocEDA WP3: Monitoring and Governance (Deliverable SocEDA)

6. Annexes

6.1. Sample diagram

6.1.1. EPL description

In this section we will describe a diagram representation for a given rule. We took a rule from a use case where we need to find the closest taxi according to the current customer position.

We will need to store the last taxis positions, The query is:

create window taxis.win:keepall() as Select longitude,latitude,phoneNumber From taxiPosition;

with those positions we can try to find the closest taxi when a customer sends his position.

We will calculate the distance between the customer and all the taxis, then we will sort the result and take the first one (the one with the lowest distance). The query is the one below:

Select 6378137 * Math.sqrt(Math.pow((Math.toRadians(taxis.latitude) - Math.toRadians(customer.latitude)), 2) + Math.pow((Math.toRadians(taxis.longitude) - Math.toRadians(customer.longitude)), 2)) as distance,customer.phoneNumber,customer.latitude,customer.longitude,taxis.longitude,taxis.latitude,taxis.phoneNumber From customer.win:length(1), taxis order by distance asc limit 1

6.1.2. Diagram representation

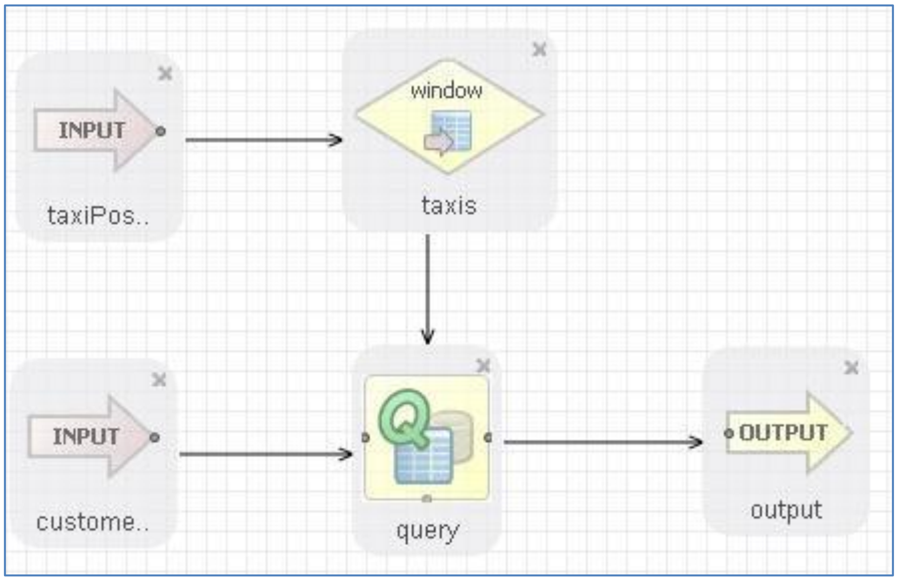
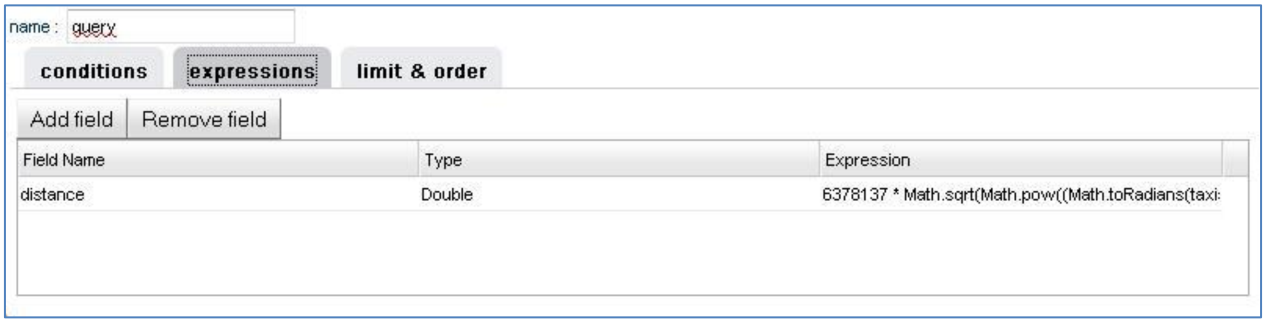


Figure 19: EPL diagram

The input components represent the two different events we are interested in: a customer query and all the last taxis positions. The taxi positions event is stored in a window component called taxis.

Then we do setup the complex part of the query within the query component.

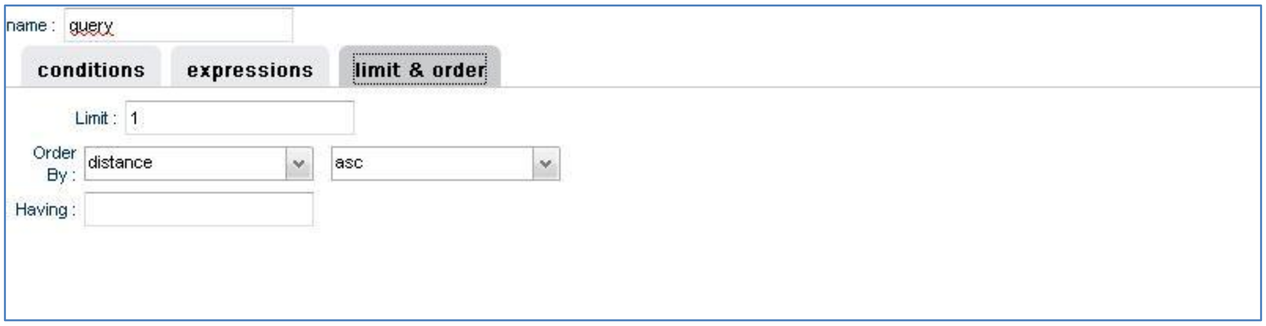


name : query

Field Name	Type	Expression
distance	Double	6378137 * Math.sqrt(Math.pow((Math.toRadians(taxi:

Figure 20: Creation of the "distance" field

In this form we do create the distance field using the fields coming from the inputs customer latitude and longitude, plus the taxis latitude and longitude.

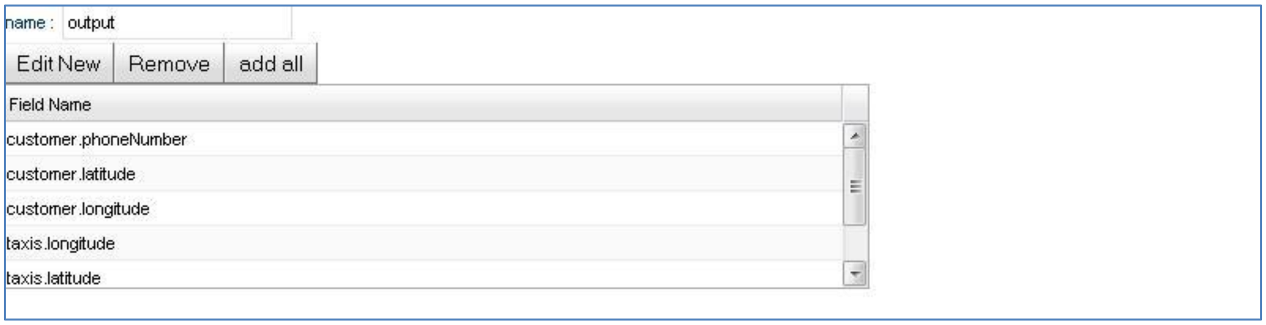


name :	query

Limit :	1	
Order By :	distance	asc
Having :		

Figure 21: Ordering and limit setup

Through this form we setup the ordering and limit aspect of the query. Now, we can link this bow with an output component has shown in the diagram representation figure.



name :	output

Field Name
customer.phoneNumber
customer.latitude
customer.longitude
taxis.longitude
taxis.latitude

Figure 22: Output form

Finally we specify which field we want to retrieve through the output forms.