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

Document name:	D4.2.4 Android UI Designer
Document version:	**V-1.0**
Task code:	
Deliverable code:	D4.2.4
WP Leader (organization):	OrangeLabs (FT)
Deliverable Leader (organization):	OrangeLabs
Authors (organizations):	OrangeLabs
Date of first version:	12/12 2012

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. Android UI Editor Tool positioning 5
3.2. Main functionalities 7
3.2.1. Design 7
3.2.1.1. Palette 7
3.2.1.2. Diagram zone 12
3.2.1.3. Input panel 14
3.3. Application Generation 15
3.3.1.1. APK generation 15
Illustration table 17
4. References 18
5. Annexes 19

1. General description

1.1. Context

1.1.1. Objectives

The SocEDA project attempts 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 design of Applications, on top of the platform with the help of graphical tools. This document is about the *Android UI Edito*r which is supposed to facilitate the automatic creation of Android applications (format APK) in the context of the whole SocEDA platform [3].

This part complements the CEP editor which facilitates the design of CEP rules. With the help of the *Android UI Editor* both parts are covered: the CEP edition part and the Application high level part.

1.1.2. Target user

The idea is to provide a tool to help the user to generate an Android application 'CEP aware' in a simple way. He will just have to be able to understand some basic concepts such like drag and drop boxes/components and be able to link them together in order to build an Android Application, form and event oriented (pub/sub). Basically the tool will be as simple as possible in order to target 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
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)
UI	User Interface
APK	Android application Package File

3. General Architecture

3.1.1. Main principles

The Android UI editor sits near the CEP editor on top of the platform. Basically, it allows the design of an Android UI Applications from a palette, providing basic UI components plus the link to the Event pub/sub mechanism.

3.1.1.1. Android UI Editor Tool positioning

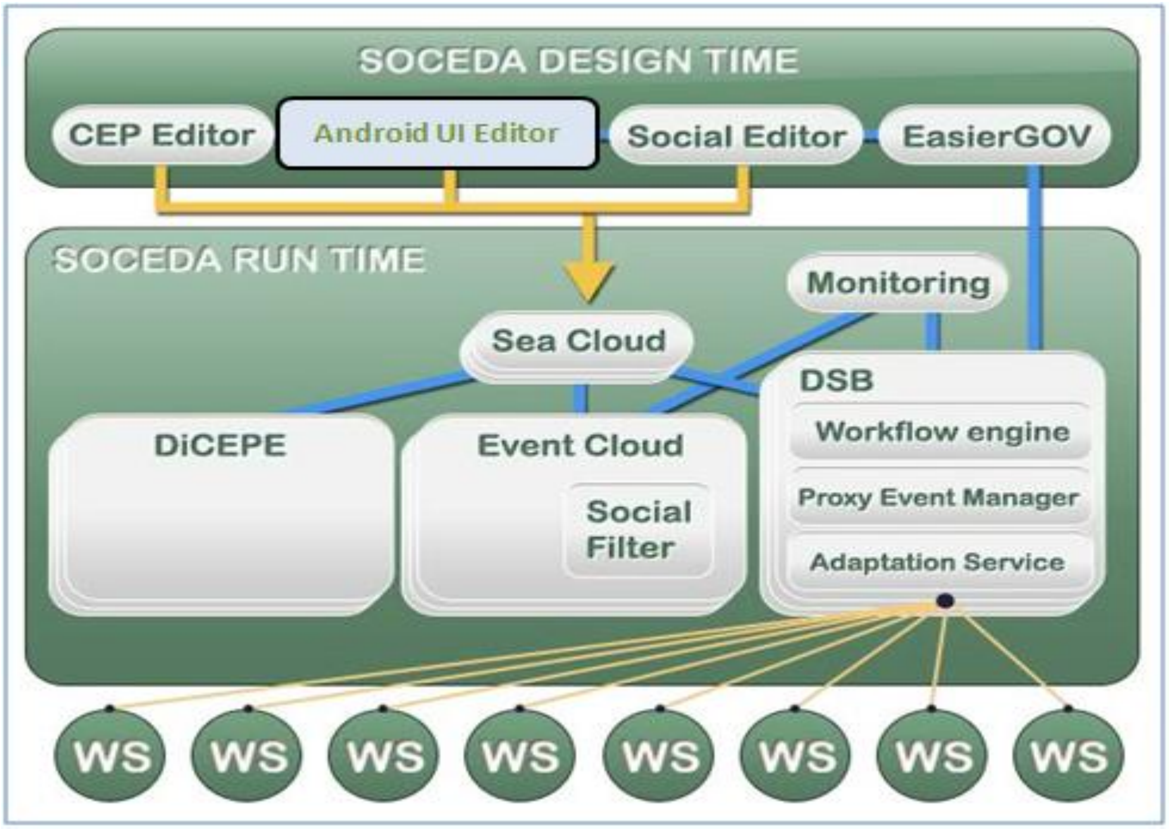


Figure 1: SOCEDA conceptual architecture (Android UI Editor)

The above figure shows the whole conceptual architecture with the platform at the first raw level, plus the different Editors on top. In this deliverable we focus on the Android UI Editor which interacts with the Seacloud component. This Seacloud component is federating the whole middleware.
This Editor is also linked to the event type repository and EPL rule repository through the Seacloud component that offers the mandatory Web Services interfaces.

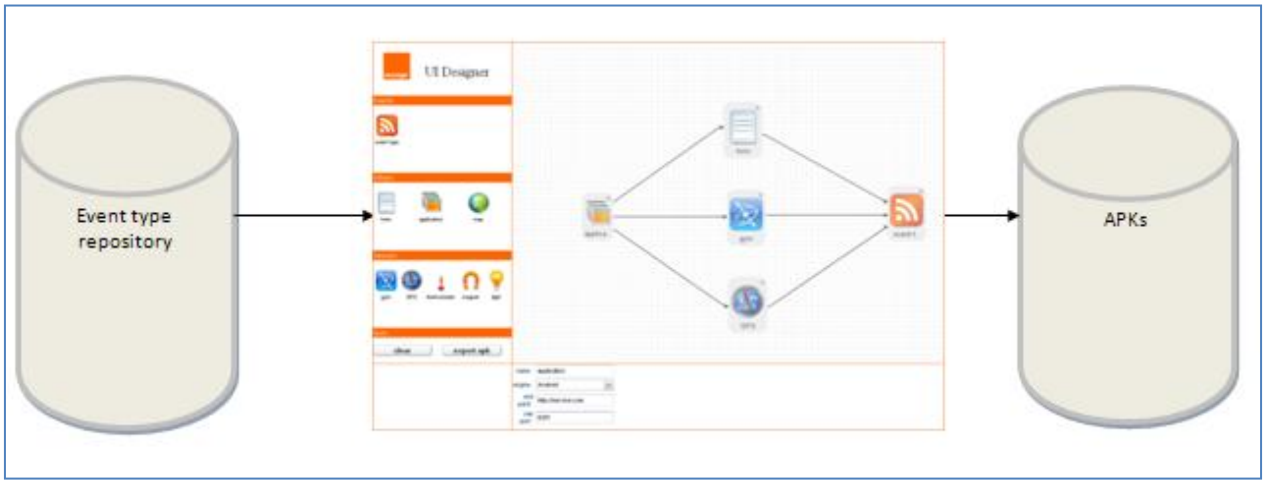
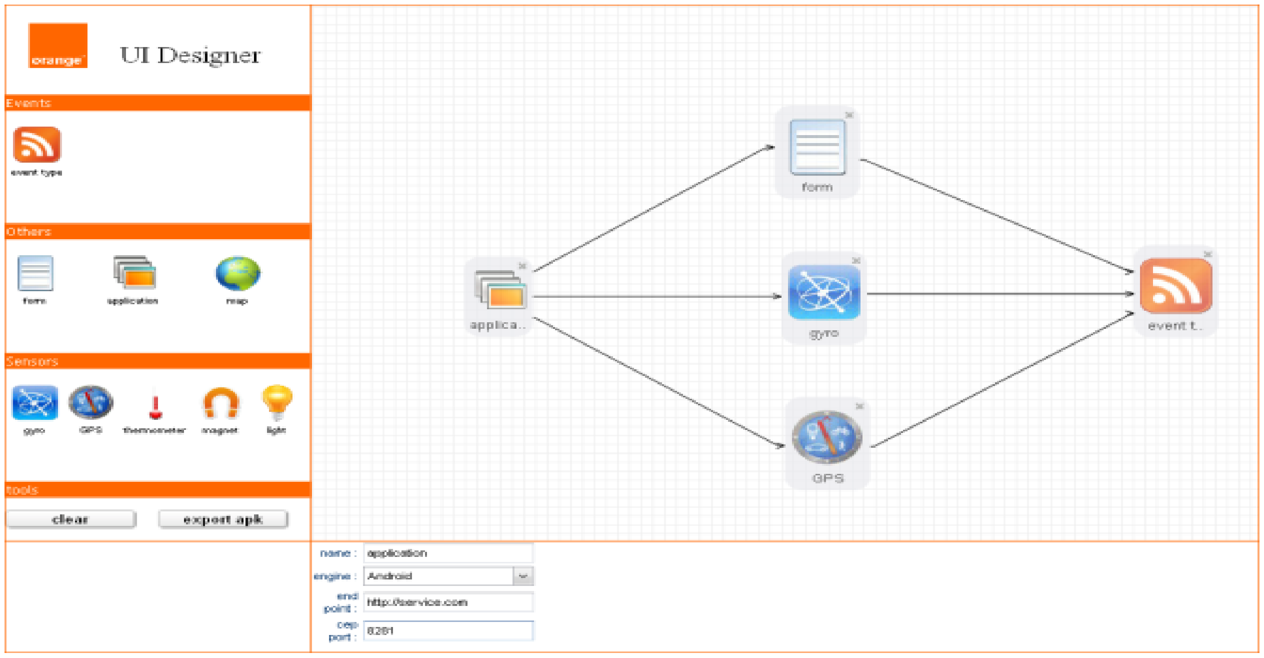


Figure 2: Input/output of the UI Android Designer

There after this is the big picture of the UI Android Designer with all the UI components. These different UI components are described in the following section 3.2.1



name : application
engine : Android
end point : http://service.com
cep port : 8281

Figure 3: UI Android Designer

3.2. Main functionalities

3.2.1. Design

This Android UI Editor Design tool is 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

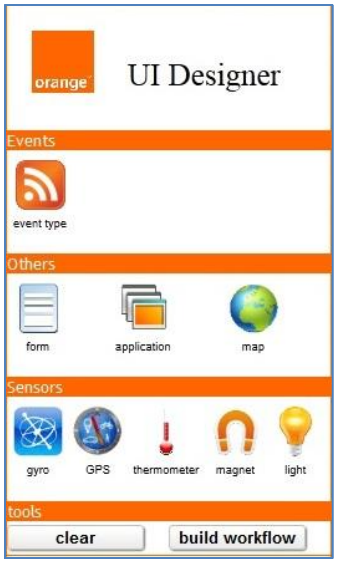


Figure 4 : Palette elements

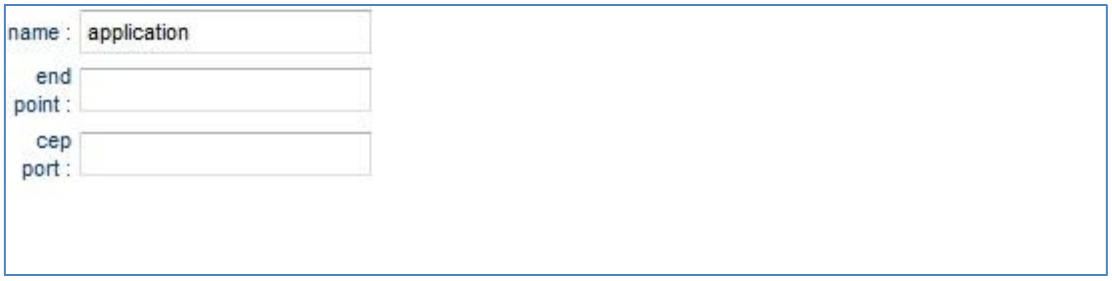
The palette contains at least the basic element to design a rule. It means:

- 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 is used to merge two input and specify which field will be used to merge them,
- aggregate is used to make some aggregate operation on some input field,
- query allows to add some condition on fields (in order to filter). it is also used to create a new field (making operations using input fields)and to add some ordering and limit.

3.2.1.1.1. Application box

When a user drops this element he has the possibility to customize it, specifying which CEP the application will be using. To configure this box he will have a bottom panel related to this input element, with three fields:

- Name of the application
- End point (CEP)
- Port (CEP)

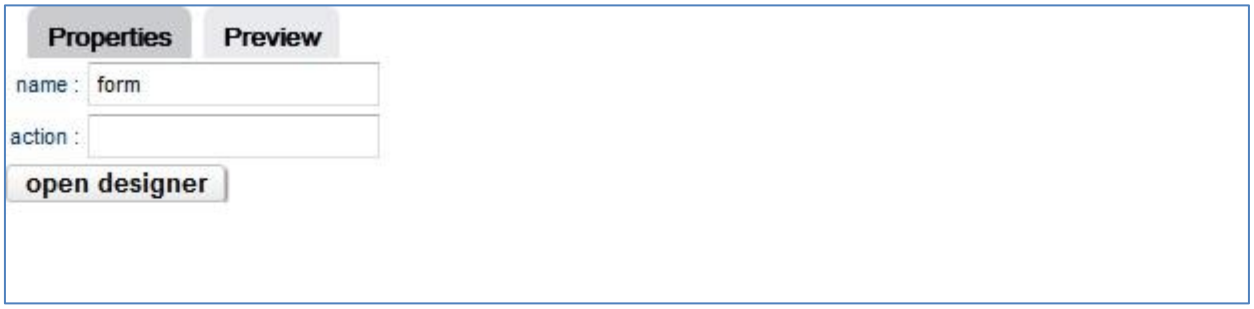


name :	application
end point :	
cep port :	

Figure 5: input bottom panel

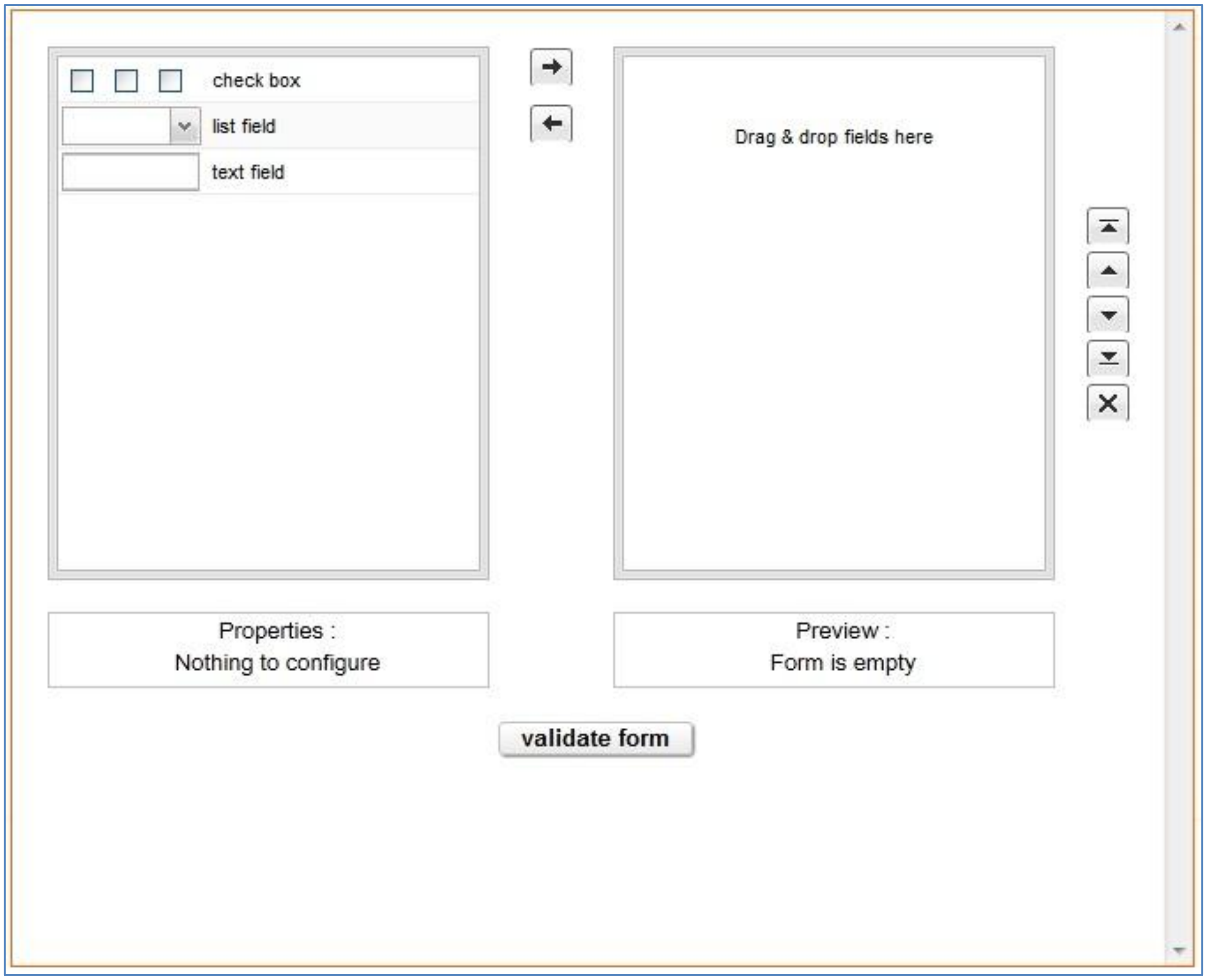
3.2.1.1.2. form box

This palette element is used to create a form through a form designer. This form can be composed with list fields, text fields or check box. This palette element does also embed a previewer in order to check the final result. This form (linked to the application creates new fields from the smartphone).



name :	form
action :	

Figure 6: form bottom panel



Properties :
Nothing to configure

Preview :
Form is empty

Figure 7. Form editor

3.2.1.1.3. Map box

We do use this palette element to add a map view within the application for contextual application where the user as to see his position. In the window below, the designer can setup the refresh time and indicate the possibility to get the users position ("follow" box).

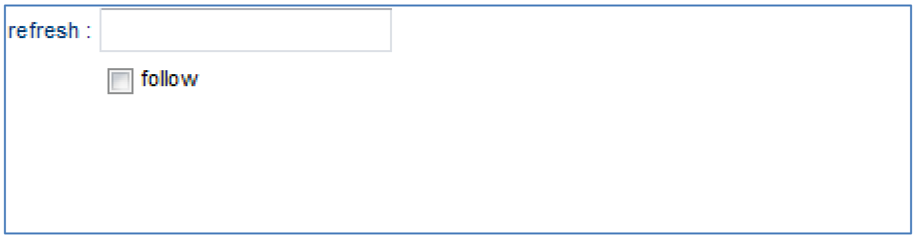


Figure 8: Map bottom panel

3.2.1.1.4. gyro box

This palette element will be used when the designer wants to retrieve and use the gyro information within the generated application. There is nothing to setup for this palette element. It will have three outputs:

- accelerationX
- accelerationY
- accelerationZ

3.2.1.1.5. GPS box

This palette element will be used when the designer wants to retrieve and use the GPS information within the generated application. There is nothing to setup for this palette element. It will have two outputs:

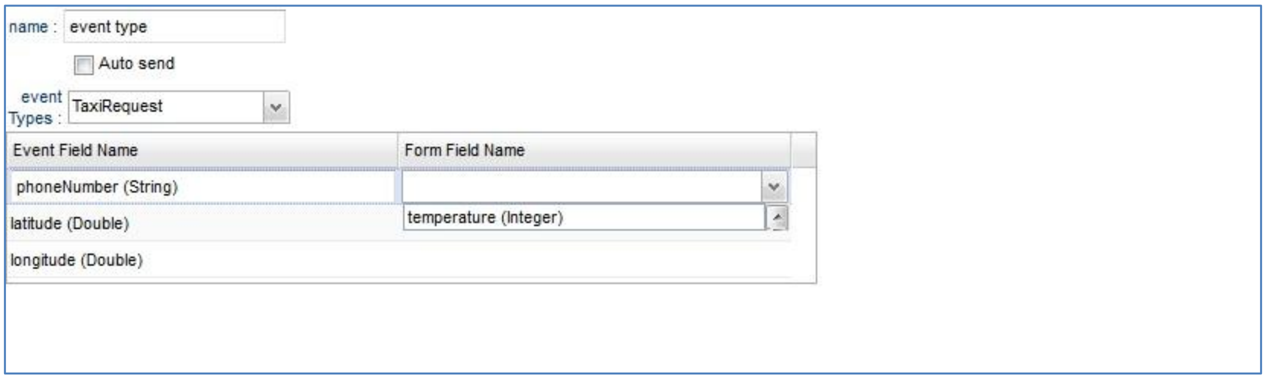
- Latitude
- longitude

3.2.1.1.6. Thermometer box

This palette element will be used when the designer wants to retrieve and use the GPS information within the generated application. There is nothing to setup for this palette element. It will have a temperature output.

3.2.1.1.7. Event type box

This palette element will be used to gather the information from the others element (gyro, gps, form …) in order to create an event type. The below configuration panel will allow the designer to specify which event type will be created and which value will be associated to the event type fields.


name : event type
☐ Auto send
event Types : TaxiRequest

Event Field Name	Form Field Name
phoneNumber (String)	
latitude (Double)	temperature (Integer)
longitude (Double)	

Figure 9 : event type bottom panel

3.2.1.1.8. The palette tools box

The palette will also contain two buttons:

- One to clear and reset the environment,
- Another one to generate the designed application.

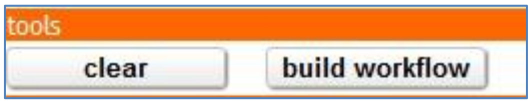


Figure 10: palette toolbox

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.

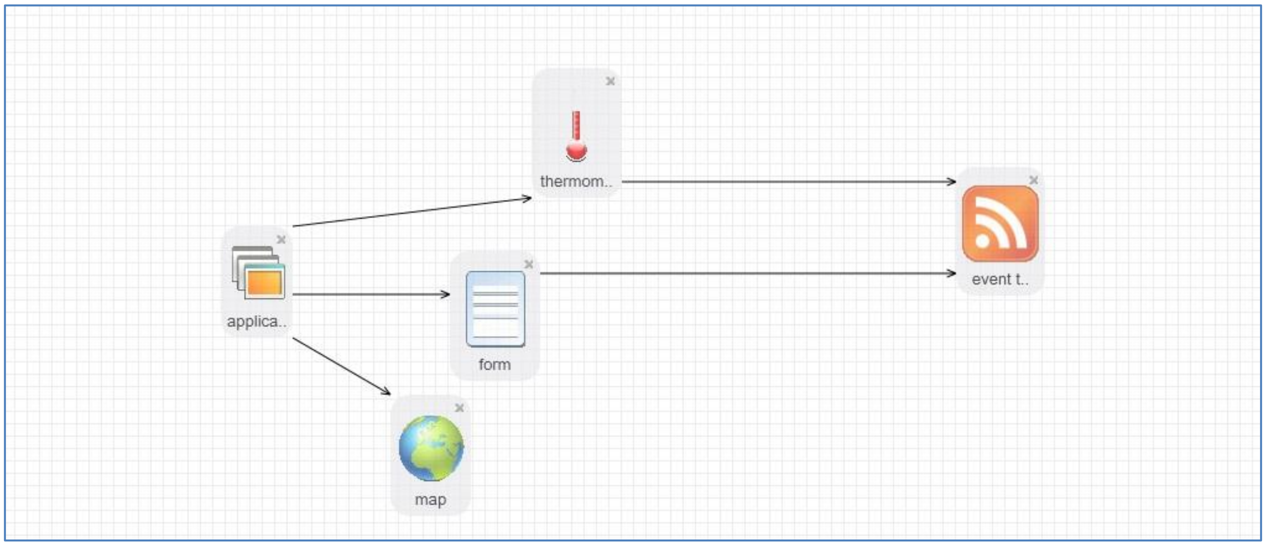


Figure 11: Diagram zone

As shown in the above picture we can link the 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.

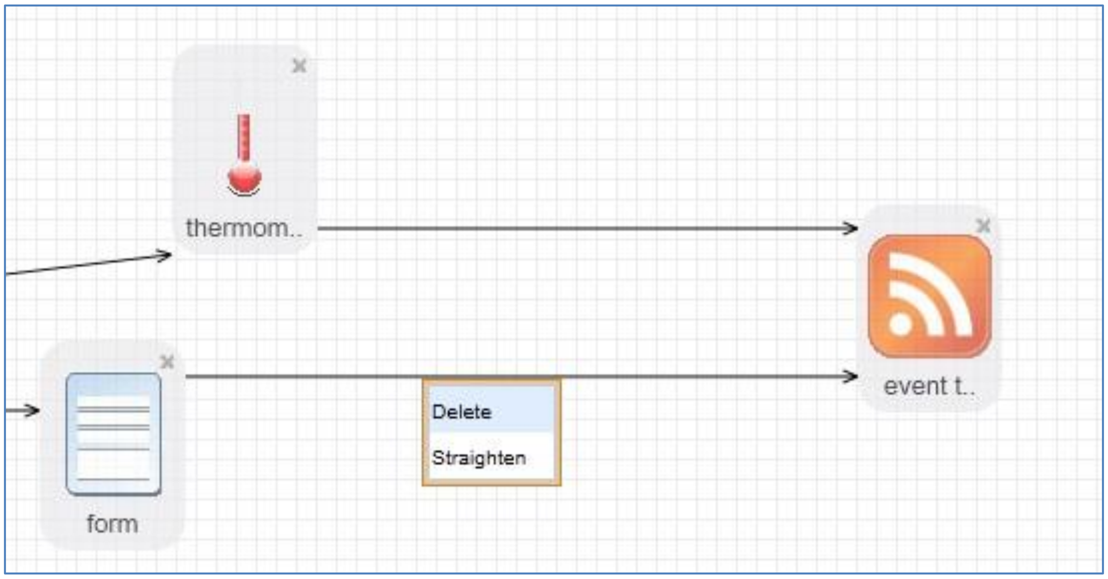
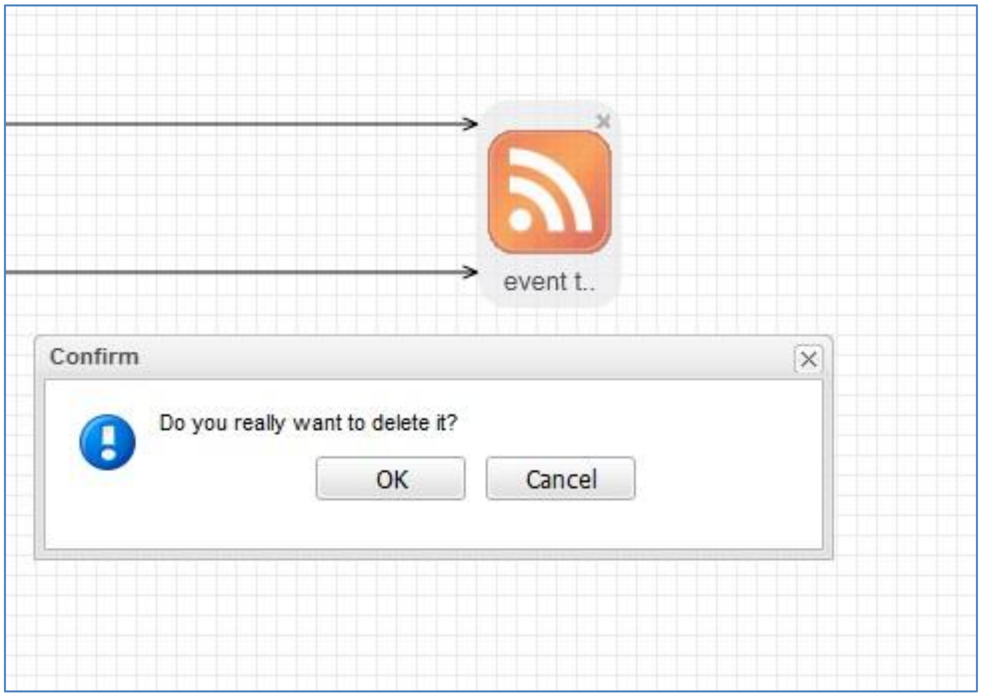


Figure 12: Connector deletion

- Delete an EPL element



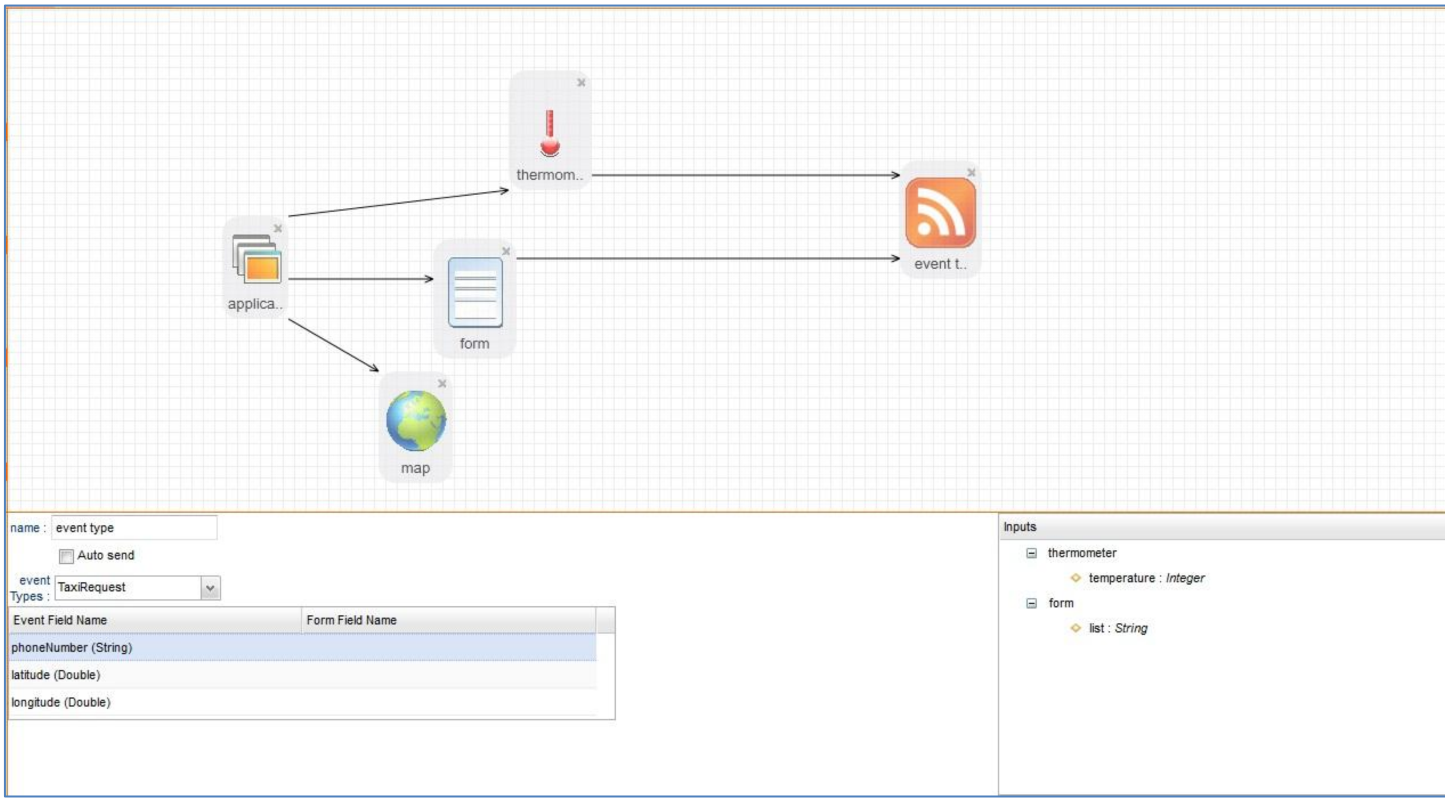
Do you really want to delete it?

Figure 13: 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 the user 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 : event type

Auto send

event Types : TaxiRequest

Event Field Name	Form Field Name
phoneNumber (String)	
latitude (Double)	
longitude (Double)	

Inputs
- thermometer
  - temperature : *Integer*
- form
  - list : *String*

Figure 14: 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. Application Generation

3.3.1.1. APK generation

The APK generation is a process starting at the client side using server side servlet and ending at client side with the generated android application. The main involved components are described thereafter:

- The UIDesigner: in charge of the edition of the Application. This is a canvas located near the CEP Editor on the GUI
- The FormDesigner : in charge of editing the form.
- The GWTServlet: in charge of displaying the Application and saving it to a format that can be transformed to an Application by phoneGap [6] tool
- The phoneGap: transforms the edited application in a concrete APK file format, runnable on Android an architecture

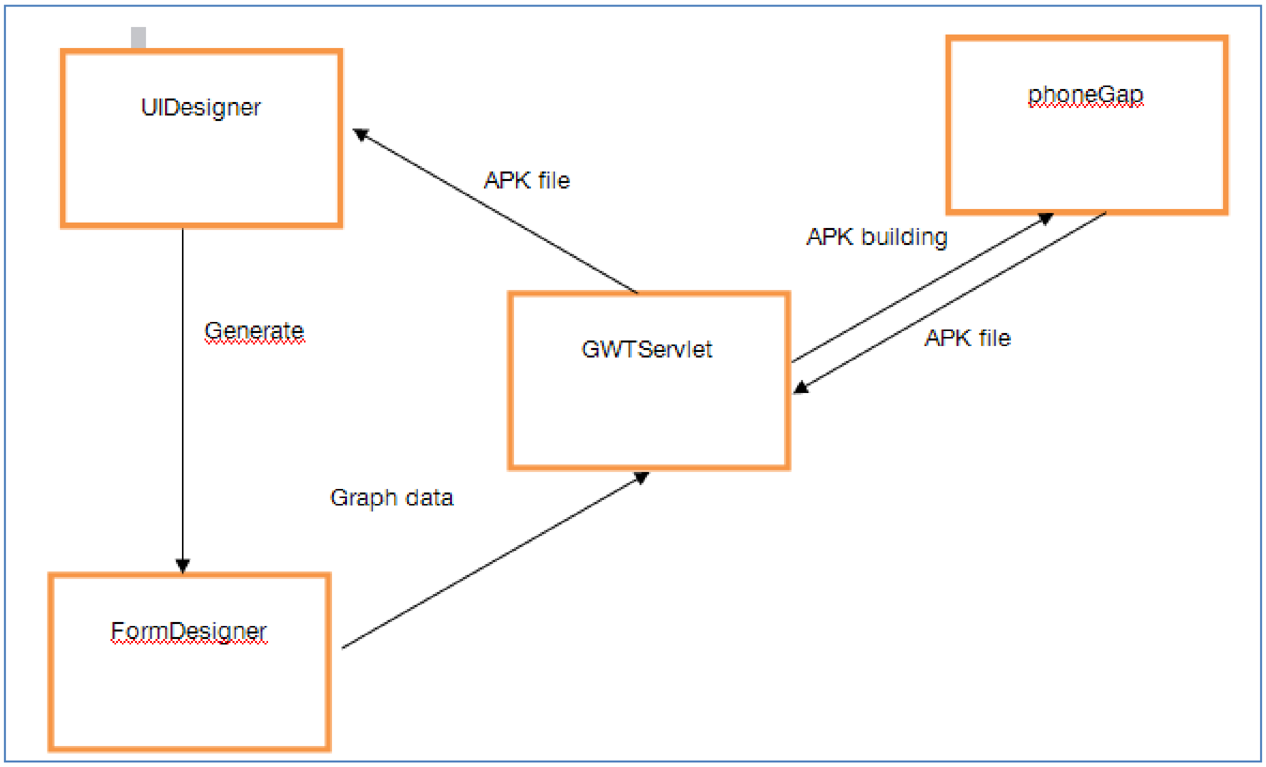


Figure 15: APK Building from the UI designer

At the client side, the user describes a workflow with the UI Designer tool. When clicking on the generate button, a graph representation is built and sent to a GWT servlet.

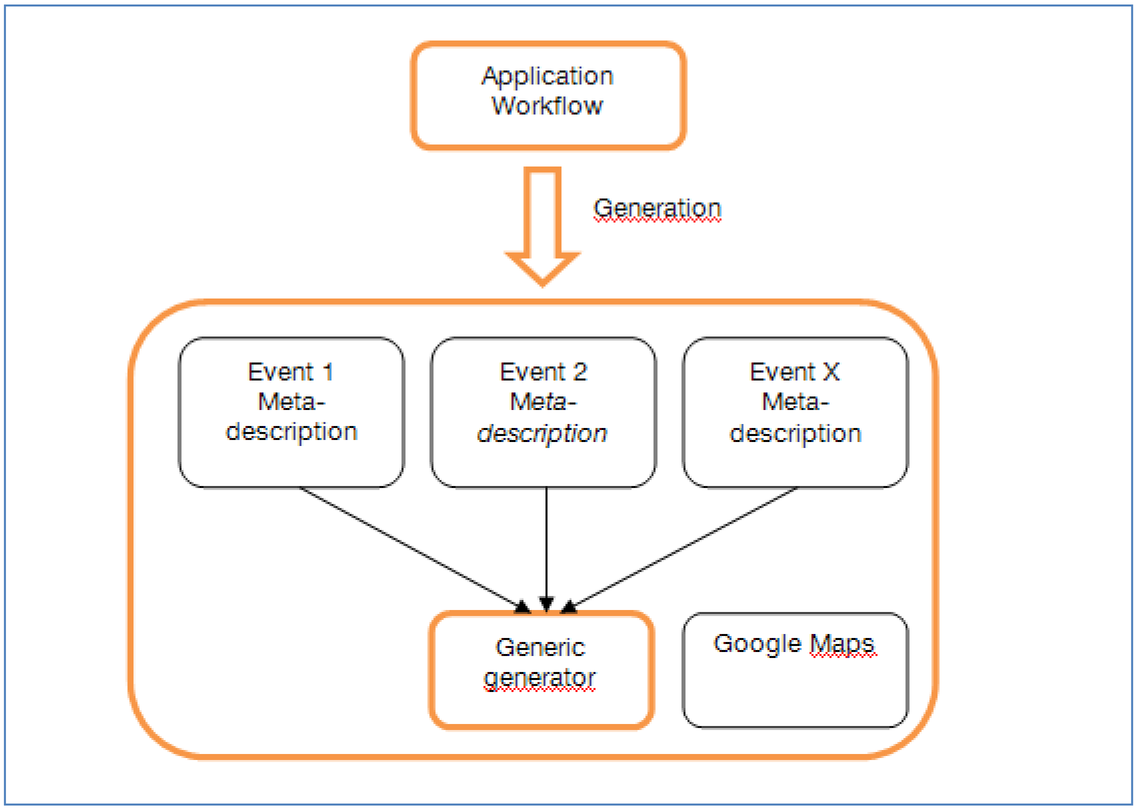


Figure 16 Application generation

This graph takes as root element an "Application", as node the events to send and as leaf some functionalities like sensors or maps. Each application have for end stream one or more events. The start point is always an application in which you define endpoint and port. You can add one and only one Google map which target the user position.

Applications are used to send event to the Soceda platform and so Events are sent with a native Java class named CEPSocket which is delegated into PhoneGap JavaScript file.

PhoneGap need an HTML file for the view of the application and a JavaScript file for the process. These two files are constructed by the GWT servlet on the server side. By going through the graph it is possible de make a user interface and all the processes associated.

Each leaf of the graph can contain JavaScript heading part, calling part and HTML display.

EventType nodes contain JavaScript head witch represent an instance of the meta-description, with the needed properties. Event can be defined as watching event and then will be sent periodically

To send event we use a generic event type generator which takes a meta-description of the events.

Illustration table

Figure 1: SOCEDA conceptual architecture (Android UI Editor) 5
Figure 2: Input/output of the UI Android Designer 6
Figure 3: UI Android Designer 6
Figure 4 : Palette elements 7
Figure 5: input bottom panel 8
Figure 6: form bottom panel 8
Figure 7. Form editor 9
Figure 8: Map bottom panel 10
Figure 9 : event type bottom panel 11
Figure 10: palette toolbox 12
Figure 11: Diagram zone 12
Figure 12: Connector deletion 13
Figure 13: EPL element deletion 13
Figure 14: diagram setup environment 14
Figure 15: APK Building from the UI designer 15
Figure 16 Application generation 16

4. 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] PhoneGap: Application generator : http://www.phonegap.com/

5. Annexes