European Collaborative Innovation Centres for Broadband Media Service

YOUR ELECTRONIC ASSISTANT™

e-Me

Plugins

EUROPEAN UNION
European Regional
Development Fund

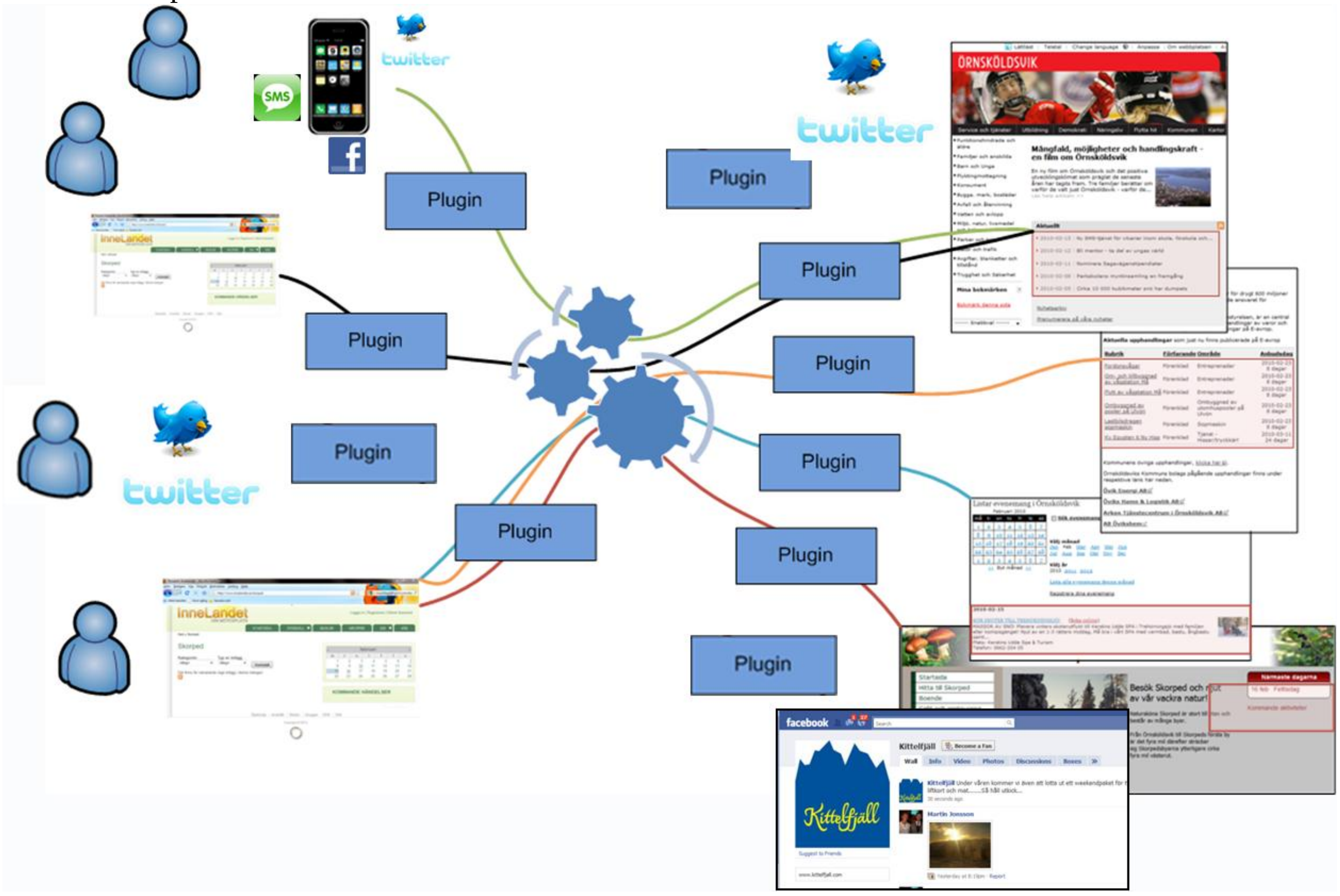
Investing in the future by working together for a sustainable and competitive region

The Interreg IVB
North Sea Region
Programme

1 Plugins

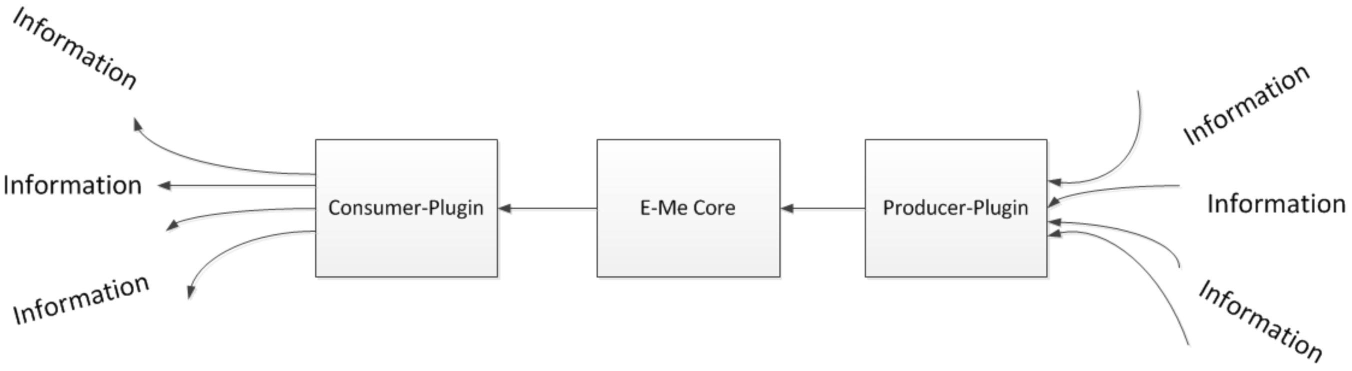
1.1 What are plugins?

Whilst e-Me Core is the engine that schedules and delegates work, plugins are the ones performing the actual work. These are the parts that bring e-Me together. Plugins are completely independent of each other and can be added and removed without affecting the rest of the system.
The image below illustrates these connections between third parties, plugins and e-Me Core. The cogs in the middle represent Core.



1.2 Architecture

There is an important part when describing the architecture and where plugins come into place in e-Me. First, there are two different types of plugins, secondly, they are built the exact same way. To explain the first statement, there are Consumer plugins and Producer plugins. The image above shows Consumers on the left hand side and Producers on the right. The second statement explains that they are built the exact same way. This is also true and it is done to simplify the way e-Me Core handles the plugins. This makes it possible for Core to take any kind of plugin without even caring about what kind it is. Of course when a plugin is registered, the developer of the plugin will fill in information about the plugin and state whether it is a consumer or producer.



1.3 Relation to Core

As mentioned previously, the plugins rely on Core. Core is the engine that starts the plugins and supplies them with their required information needs. These can be a range of things that the plugins decide. Plugins can require that users input information such are their email address, a verification to post on twitter or facebook. Pretty much anything that a user can be prompted with, it's all down to the plugin to decide what this information is.

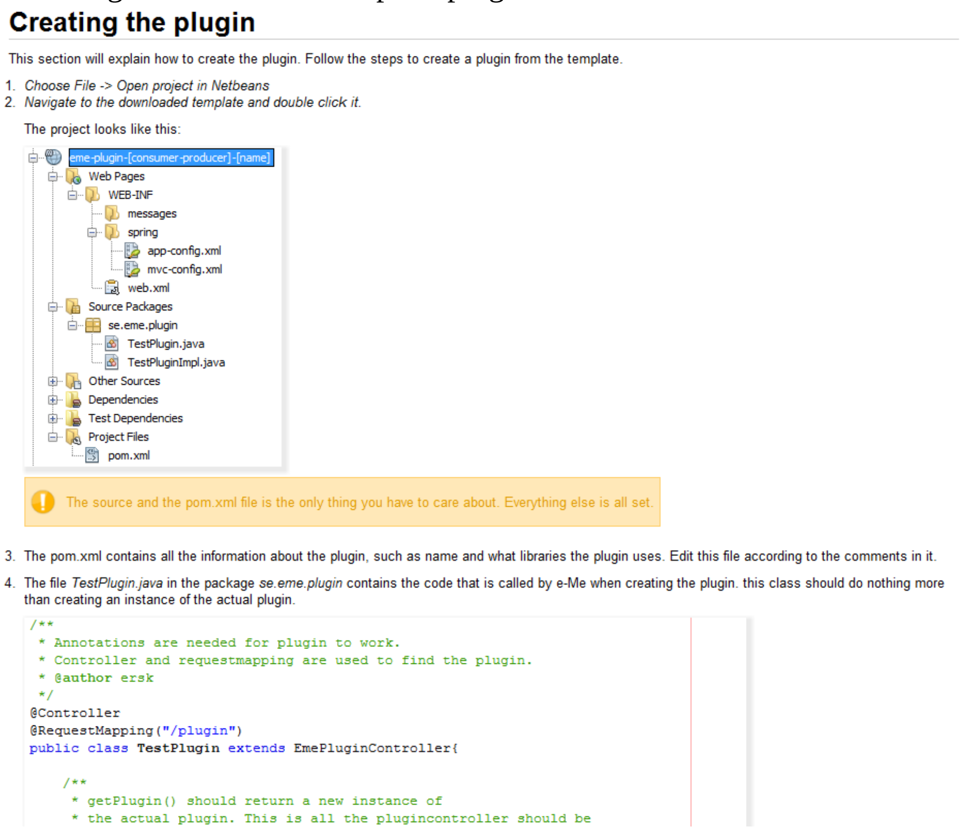
Plugins also have access to store and retrieve information from a database. This information can be used on a per-plugin and per-user level. The information is usually a date of when the plugin was last run. This is done to make sure that for example the plugin does not get the same information twice to the same user.

1.4 Software

Plugins are, just like core, java applications. They are packaged as .WAR files (web archive) and runs on the same JBoss Application Server as e-Me Core.
Plugins can use any external libraries they want to perform their tasks. Core provides a few to simplify the development of plugins. They are also maven based so a developer simply has to build using maven and all dependencies will be downloaded automatically.

1.5 Development

Developing a plugin is very straight forward. A developer goes to http://www.e-me.se/developer and follows a guide there to develop the plugin.



Creating the plugin

This section will explain how to create the plugin. Follow the steps to create a plugin from the template.

1. *Choose File -> Open project in Netbeans*
2. *Navigate to the downloaded template and double click it.*

The project looks like this:

```
eme-plugin-[consumer-producer]-[name]
  Web Pages
    WEB-INF
      messages
      spring
        app-config.xml
        mvc-config.xml
      web.xml
  Source Packages
    se.eme.plugin
      TestPlugin.java
      TestPluginImpl.java
  Other Sources
  Dependencies
  Test Dependencies
  Project Files
    pom.xml
```

The source and the pom.xml file is the only thing you have to care about. Everything else is all set.

3. The pom.xml contains all the information about the plugin, such as name and what libraries the plugin uses. Edit this file according to the comments in it.
4. The file *TestPlugin.java* in the package *se.eme.plugin* contains the code that is called by e-Me when creating the plugin. this class should do nothing more than creating an instance of the actual plugin.

```java
/**
 * Annotations are needed for plugin to work.
 * Controller and requestmapping are used to find the plugin.
 * @author ersk
 */
@Controller
@RequestMapping("/plugin")
public class TestPlugin extends EmePluginController{

    /**
     * getPlugin() should return a new instance of
     * the actual plugin. This is all the plugincontroller should be
```

A template is downloaded and the developer only has to change the content of one method in the code to create the plugin. The code is then built and uploaded to e-Me from the web site and is thereby automatically deployed on the server. This requires that a user is a registered developer.