



June 2nd - 4th 2014  
Copenhagen, Denmark



# Cut your Grails App to Pieces Build Feature Plugins

**Göran Ehrsson**

Technipelago AB

# Technipelago AB



# Custom Business Applications

Different industries

Common requirements

- Customers
- Communication (email)
- Documents
- Tasks
- ...



# The challenge

Customer want something simple but it should rocket fuel their business process

Customer have looked at standard software but found show stoppers or budget constraints

Developing from scratch each time would be too expensive or feature limited

One app with VCS branches for each customer would end up in maintenance hell

Copy code between projects is extremely bad



# Say no to copy & paste!

~~⌘-C~~

~~⌘-V~~

⌘

# Grails Plugins

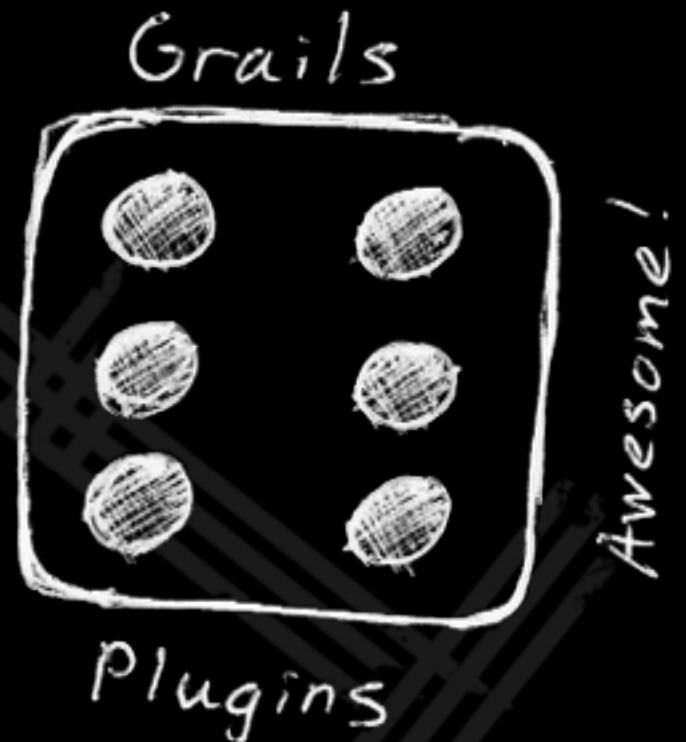
Plugins extend the platform.

A plugin can:

- extend the data model
- add services
- provide static resources
- add command line scripts

...

The Grails platform provides  
lots of extension points



# Create a plugin

```
grails create-plugin myplugin
```

```
cd myplugin
```

```
grails run-app
```

A Grails plugin is a regular Grails project with a plugin descriptor in the root of the project.

```
class MypluginGrailsPlugin {  
    def version = "0.1"  
}
```

# Installing local plugins

**grails maven-install**

```
repositories {  
    ...  
    mavenLocal()  
}  
  
plugins {  
    compile ":myplugin:0.1"  
}
```

*theapp/grails-app/conf/BuildConfig.groovy*



# -SNAPSHOT versions

Prior to Grails 2.3 local plugins with -SNAPSHOT versions was not supported due to ivy limitation

Workarounds:

- Increment version number before each release

- Delete ivy-cache after each release

- Use a remote repository manager (Artifactory)

Grails 2.3+ uses Aether as dependency resolver and local -SNAPSHOT versions are supported

# Publish plugins to your own Repository Manager

You can use a remote repository manager, like Archiva, Artifactory or Nexus. Host it yourself or use an external pay-per-use cloud service.

```
grails.project.repos.mycompany.url =  
    "http://repo.mycompany.com/plugins-releases/"  
grails.project.repos.mycompany.username = "admin"  
grails.project.repos.mycompany.password = "password"
```

*~/grails/settings.groovy*

```
grails publish-plugin --repository=mycompany
```

# Configure remote repositories

```
repositories {  
    ...  
    mavenRepo "http://repo.mycompany.com/plugins-snapshots/"  
}  
  
plugins {  
    compile ":myplugin:1.0-SNAPSHOT"  
}
```

*theapp/grails-app/conf/BuildConfig.groovy*

# Inline plugins

Inline plugins lets you develop plugins as if the code were part of the application. Auto-reloading works so you immediately see changes.

```
grails.project.dependency.resolution = {  
  repositories {  
    ...  
  }  
  plugins {  
    //compile ":myplugin:0.1"  
  }  
}  
grails.plugin.location.myplugin = "../..../plugins/myplugin"
```

*theapp/grails-app/conf/BuildConfig.groovy*

# Plugin Design

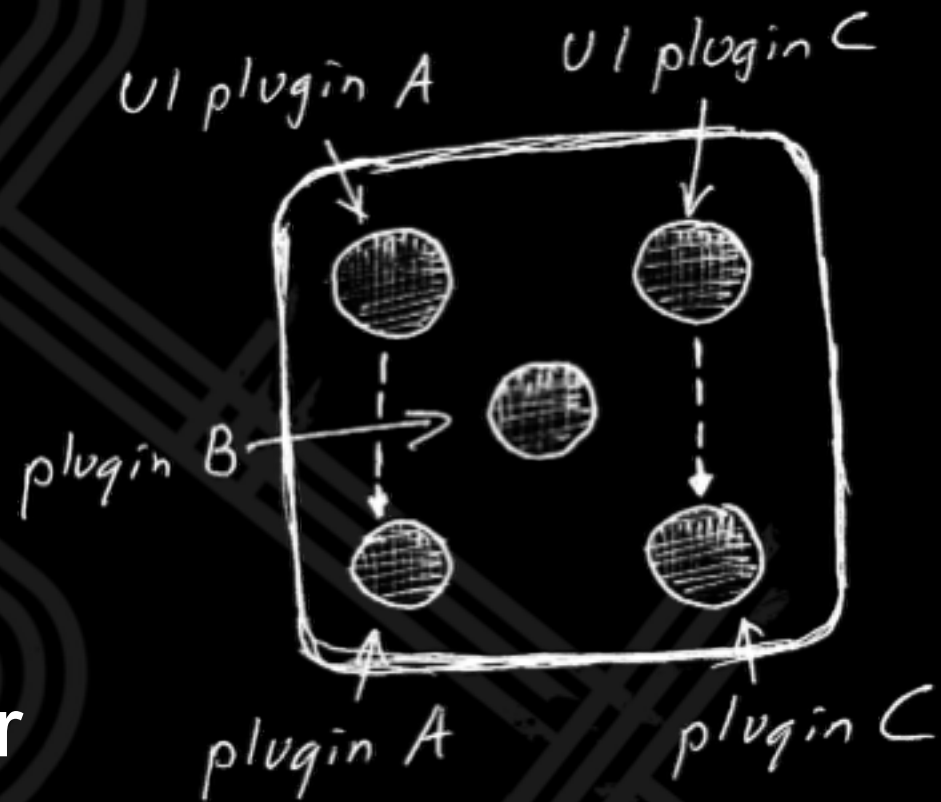
Separation of concern

Keep services and related UI  
in separate plugins

Avoid intra-plugin  
dependencies

Communicate with events

The application is the director





# Separation of Concern

Each plugin should focus on one task or domain

A plugin should be tested isolated from others

Boundaries are strong and well defined

It forces the developer to stay inside the box



# Keep services and UI in separate plugins

Most of the logic are located in the service layer

You may want to have different user interface plugins for different requirements

The same service plugin can be used in both the web-front application and in back-office without exposing admin UI to web-front

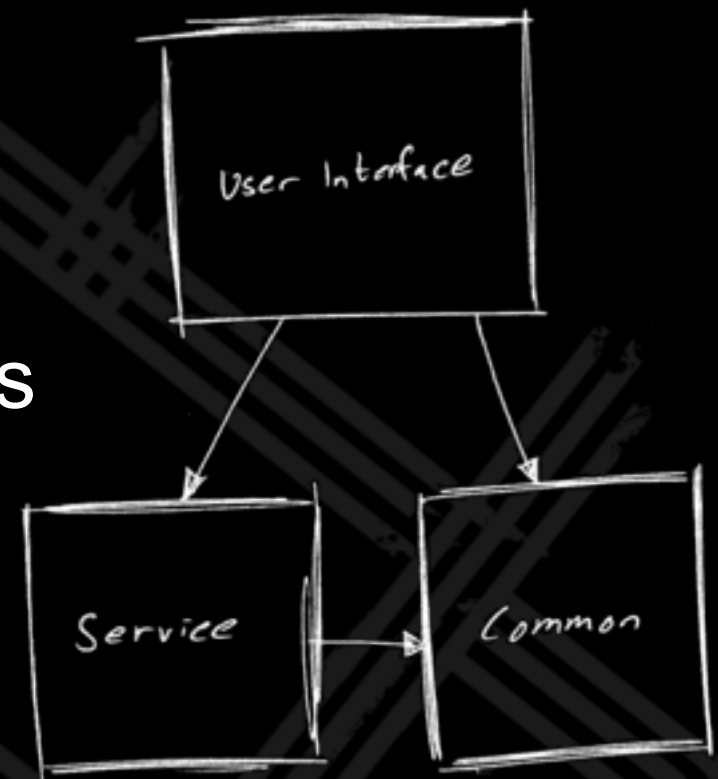
You can use the same service plugin in different micro service style applications

# Avoid intra-plugin dependencies

UI-plugins are allowed to talk directly to it's associated service plugin, but not the opposite

Place common features in one or few common plugins.

Other plugins are allowed to depend on common plugins



# Communicate with events

Spring has built-in support for both synchronous (default) and asynchronous events

Spring Integration includes advanced event support

Apache Camel supports Event Message pattern (EIP)

Grails platform-core plugin includes great event handling

Synchronous, Asynchronous, Event Reply

The Grails events plugin is a evolution of platform-core events

# Application is the director

Individual plugins should not know about other plugins

The Application route events from one plugin to another

The Application can access all plugins if needed

The Application is the director that coordinate events



# Drawbacks

Debugging events can be hard

How to deal with exceptions

Not easy to follow code paths in IDE



June 2nd - 4th 2014  
Copenhagen, Denmark



# DEMO

# Real software...



**Luke Daley** @ldaley · 3h

Most software conference talks seem to be about languages, frameworks tools and frivolity. Maybe builders of real software are too busy?



4



2



**Rob Fletcher** @rfletcherEW · 1h

@ldaley you mean you'd like to see more case studies?



1



**Luke Daley** @ldaley · 35m


@rfletcherEW yeah., pretty much.



[Hide conversation](#)

7:51 AM - 28 May 2014 · [Details](#)

# Real Plugins - Demo Application



```
// Feed your brain
gr@.technologies.each {
  yourBrain << it
}
```

★ [SPEAKERS](#) [AGENDA](#) [ADMINISTRATION -](#) [FAVORITES -](#) [GR8 CONTACTS -](#)

## Cut your Grails application to pieces - build feature plugins

Göran Ehrsson

Talk

<b>Event ID</b>	<b>Start Time</b>	<b>Type</b>
81	6/4/14 10:50 AM	Conference session
<b>Name of talk</b>	<b>Ends</b>	<b>Status</b>
Cut your Grails application to pieces - build feature plugins	6/4/14 11:40 AM	Planned
		<b>Priority</b>
		Normal
		<b>Reference</b>
		<a href="#">Göran Ehrsson</a>
		<b>User</b>
		Admin

**Description**

<p>Reuse has been taught in CS classes since the very beginning. But how should you practically do to reuse functionality from one Grails application in other applications? The plugin subsystem in Grails is an awesome machinery that makes it easy to separate functionality into distinct plugins. Plugins that can be used in many applications with similar functionality. Yet have unique features without creating a maintenance hell.</p><p>In this session you will learn how to think "feature plugins" when you're designing and developing your Grails applications. We will cover topics like inter-plugin communication using application events and how to support

Cut your Grails application to pieces - build feature plugins

Conference session


June 4, 2014

<p>Reuse has been taught in CS classes since the very beginning. But how should you practically do to reuse functionality from one Grails applicati...

**TAGS**

[grails](#) [Plugins](#)

# Real Plugins - Demo Application




```
// Feed your brain
gr#.technologies.each {
  yourBrain << it
}
```

Göran EhrssonTechnipelago AB

[SPEAKERS](#) [AGENDA](#) [ADMINISTRATION](#) [FAVORITES](#) [GR8 CONTACTS](#)

**Göran Ehrsson** Developer



TAGS

Address

Talks (1)

Files

Name	Göran Ehrsson	Email	<a href="mailto:goran@technipelago.se">goran@technipelago.se</a>	Contact Id	4
Title	Developer			Responsible	Admin
Relation	<a href="#">Technipelago AB</a>				
Postal	Djurhamn, SE				

[Q, Find Contact](#) [Edit](#) [New Contact](#) [View](#)



# GR8CRM

crm-contact & crm-contact-lite

crm-content & crm-content-ui

crm-task & crm-task-ui

crm-campaign & crm-campaign-ui

crm-product & crm-product-ui

crm-blog & crm-blog-ui

crm-ui-bootstrap

~40 plugins in total

# Dynamic Associations

If a domain class in a feature plugin need to associate with a domain instance in another plugin, use "dynamic associations"

"crmContact@42"

The framework will lookup the domain instance when needed

Lookup Spring bean "crmContact"

Call `crmContact.get(42)`

# Summary

Focus on domain model (not persistent entities)

Decouple business logic from user interface

Publish events asynchronously  
(synchronously if you must)

Let the application be just a container for plugins

Put customer unique code and message routing  
rules in the application  
(or in a separate plugin unique for each app)

# References

<http://gr8crm.github.io>

<https://github.com/goeh>

<https://github.com/technipelago>

*(all plugins open sourced under the Apache 2.0 License)*

@goeh

goran@technipelago.se

<http://www.technipelago.se>

<https://www.linkedin.com/in/gehrsson>