CUT YOUR GRAILS APPLICATION TO PIECES

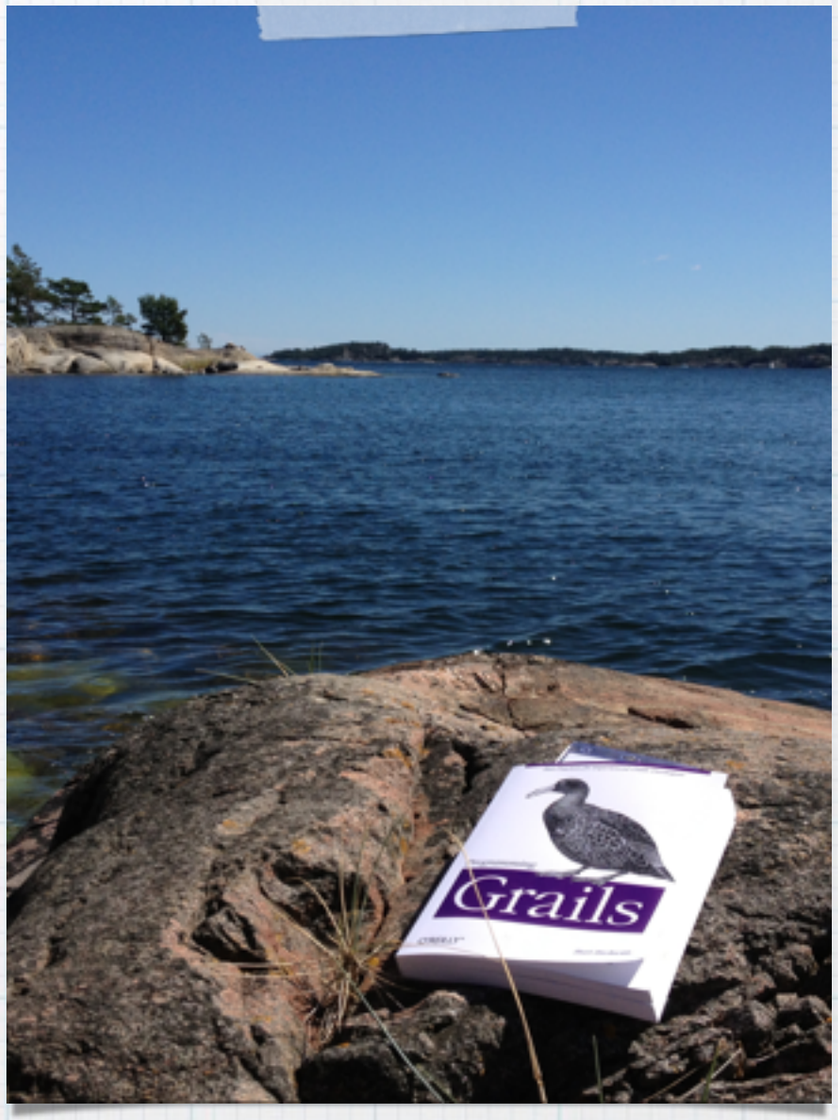
BUILD FEATURE PLUGINS

Göran Ehrsson
Technipelago AB
@goeh

Greach
the Groovy spanish conf

Göran Ehrsson, @goeh

- From Stockholm, Sweden
- 25+ years as developer
- Founded Technipelago AB
- Grails enthusiast
- Author GR8 CRM plugins

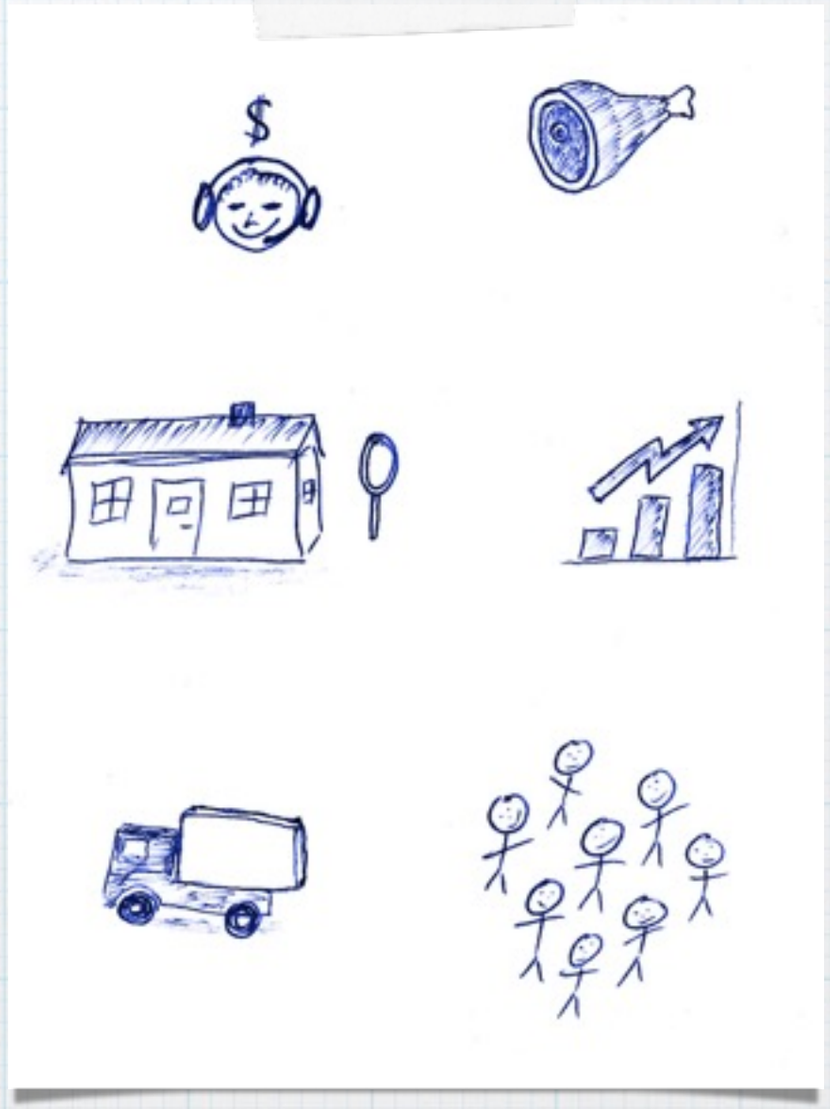


Technipelago AB

Greach
the Groovy spanish conf

Custom Business Applications

- Different industries
- Common requirements
  - Customers
  - Projects
  - Tasks
  - Documents
  - Communication



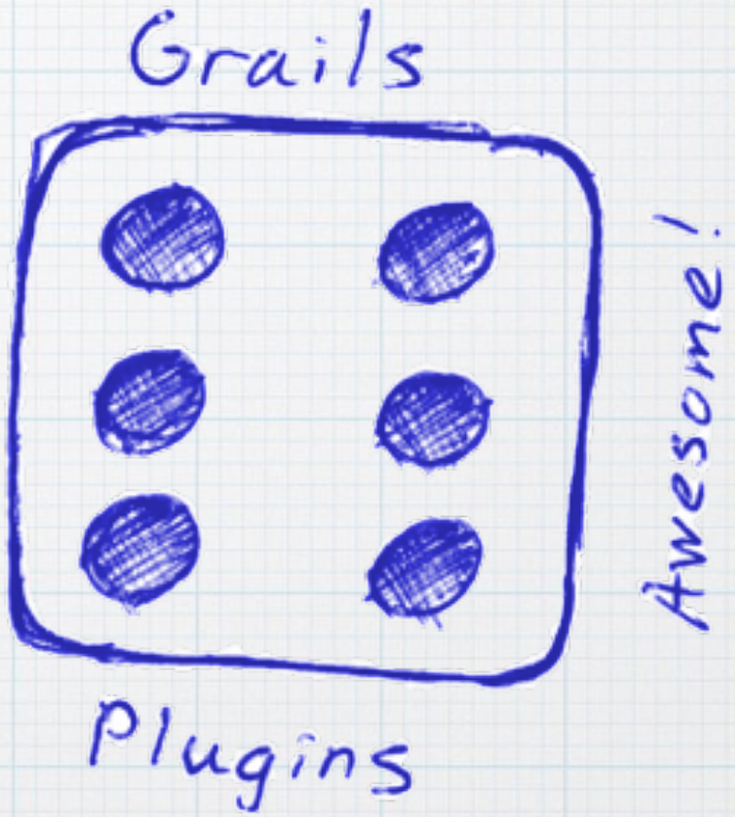
The Challenge

- Customer have looked at off-the-shelf software but faced feature limitations or budget constraints
- Customer want something simple but it should be custom made for their specific business process
- Developing from scratch would be too expensive or feature limited
- There is a gap to fill between Excel and $100 000 CRM implementations
- Develop one app with VCS branches for each customer would end up in maintenance hell
- Copy code between similar projects is also a bad idea


DID YOU SAY
COPY & PASTE?!
memegenerator.net

Grails Plugins

- Plugins extend the platform.
  A plugin can:
  - extend the data model
  - add services
  - provide static resources
  - add command line scripts
  - do a lot more...
- The plugin framework 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 efter each release
  - Use a remote repository manager (Artifactory)
- Grails 2.3+ uses Aether as dependency resolver and local -SNAPSHOT versions are supported

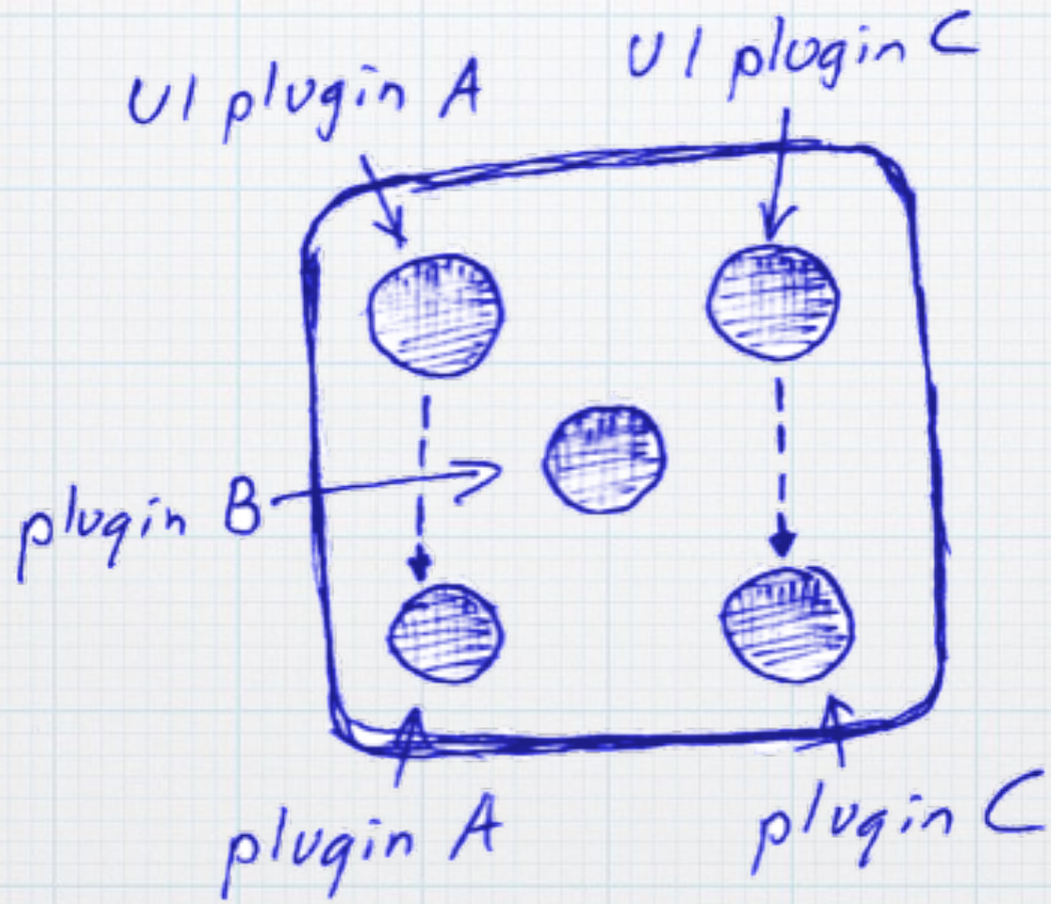
Inline plugins

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"
```

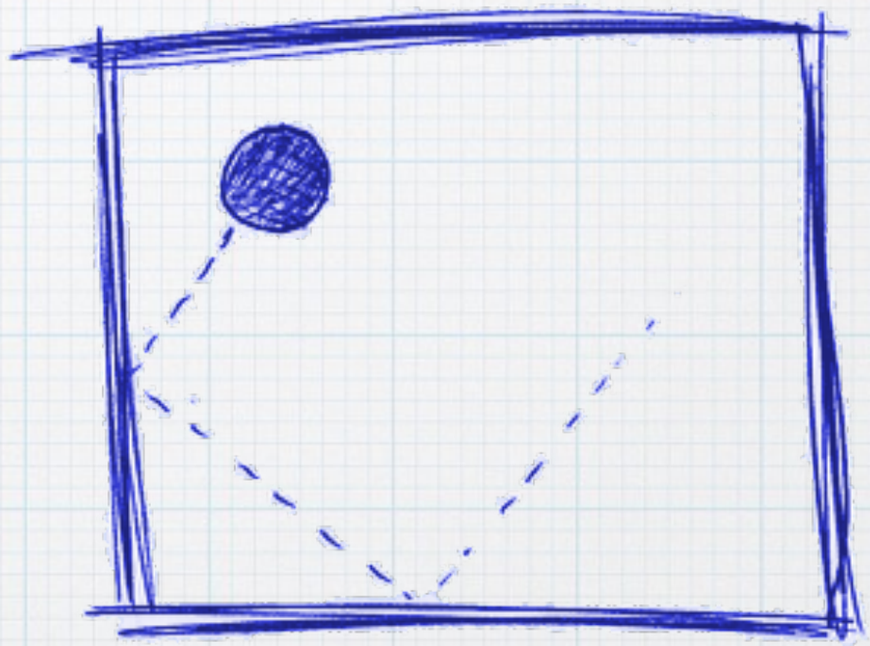
theapp/grails-app/conf/BuildConfig.groovy

Plugin Design



Separation of Concern

- Each plugin should focus on one task or domain
- A plugin should be tested isolated from others
- Plugins make the boundaries strong and well defined
- Plugins force the developer to stay inside the box

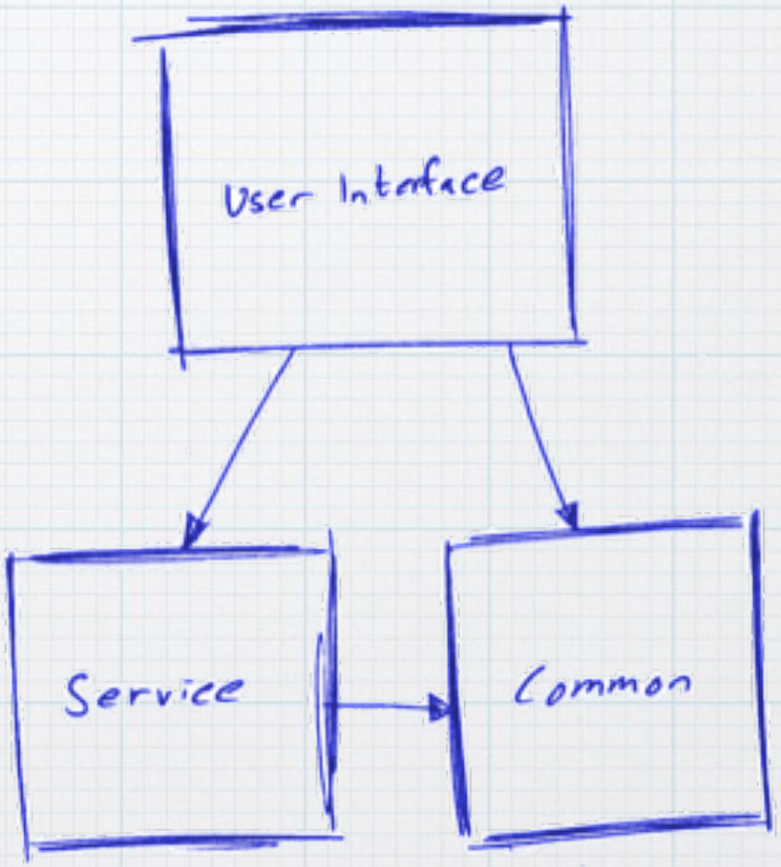


Keep services and UI in separate plugins

- Put logic in the service layer, not in view controllers
- 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 rich client or micro service style applications

Avoid intra-plugin dependencies

- UI-plugins are allowed to talk directly to its 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 messages/events

- Spring has built-in support for both synchronous 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
- Grails 3 includes event support based on the Reactor framework

The application is the director

- Individual plugins should not know about other plugins
- The application is the director that coordinate events and route events from one plugin to another
- The application can access all plugins if needed

Drawbacks

Problems you may face when going event driven:

- Error handling is harder
- Stacktraces
- Debugging events can be hard
- Not easy to follow code paths in IDE:s

What about the domain model?

How can a plugin query and fetch data from another plugin if it can't have compile time dependencies?

- DetachedCriteria
- Selection plugin

DetachedCriteria

Detached Criteria are criteria queries that are not associated with any given database session/connection. Detached Criteria queries allow you to create common reusable criteria queries, execute subqueries and execute batch updates/deletes, etc.

```
def criteria = new DetachedCriteria(Person).build {
    eq 'lastName', 'Simpson'
}
def bartCriteria = criteria.build {
    eq 'firstName', 'Bart'
}
// No Hibernate session needed above this point

def results = bartCriteria.list()
```

Selection plugin

http://grails.org/plugin/selection

- Queries are expressed as URLs
- Queries are Serializable / can be saved in database or put on a message queue

PersonService.groovy

```
@Selectable
PagedResultList<Person> list(Map query, Map params) {
    Person.createCriteria().list(params) { … }
}
```

```
def query = new URI("bean:personService/list?name=A*")
```

```
def people = selectionService.select(query, [max: 10])
```

More selection examples

```
// GORM Criteria
gorm://person/list?firstName=Sven&lastName=Anderson

// Spring Bean
bean://myService/method/arg

// External/Proxy Selection
https://dialer.mycompany.com/outbound/next?agent=liza
```

Security

```
selection.gorm = true // No restrictions, use with care
selection.gorm.com.mycompany.Person = true // Person domain class only
selection.gorm.com.mycompany = true // All domain classes in package com.mycompany
```

Soft Associations

- If a domain instance in a feature plugin need to associate itself with a domain instance in another plugin, use ”soft associations”
- Stored as a String ”person@42”
- Instantiate when needed (put generic code in service)
  - Lookup the Spring domain bean named ”person”
  - Call person.get(42)
- Find all
  - Attachment.findAllBySoft(”person@42”)

GR8 CRM

gr8crm.github.io

40+ Grails plugins for rapid development of customer relationship management applications

- crm-contact & crm-contact-ui
- crm-content & crm-content-ui
- crm-task & crm-task-ui
- crm-campaign & crm-campaign-ui
- crm-sales & crm-sales-ui
- crm-product & crm-product-ui
- crm-blog & crm-blog-ui

All GR8 CRM plugins are open source with the Apache 2.0 License

DEMO

Greach
the Groovy spanish conf

Grails 3 Plugins

- Not backwards compatible
- Plugins must be updated
  - Not as much work as expected
  - migrate2-grails3 plugin helps
- Publish plugins to bintray

Grails 3 Events API

- Based on the Reactor Framework
- Grails services and controllers implement the Events trait

```
on("myEvent1") {
    println "Hello $it!"
}
```

```
on("myEvent2") {
    return "Hello $it!"
}
```

```
notify "myEvent1", "Greach"
```

```
sendAndReceive "myEvent2", "Greach", {
  println "$it"
}
```

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 event routing rules in the application (or in a separate plugin unique for each app)

References

- gr8crm.github.io
- github.com/goeh
- github.com/technipelago
- grails.org/plugin/migrate2-grails3
- projectreactor.io
- @goeh
- goran@technipelago.se
- www.technipelago.se
- linkedin.com/in/gehrsson