Avoid Cluttered Domain Models with DCI and Groovy

Johan Eltes| johan.eltes@callistaenterprise.se | 2011-01-19

CALLISTA
Enterprise

About this talk

- Pragmatic introduction to a new design paradigm
- Touch-points to domain-driven design
- I have SOME practical experience
- I have given the topic a LOT of thought
- A little (very little) of language geekiness
- 16 lines of code

CALLISTA
Enterprise

My goal with this talk is to…

…make your brain boil of inspiration!

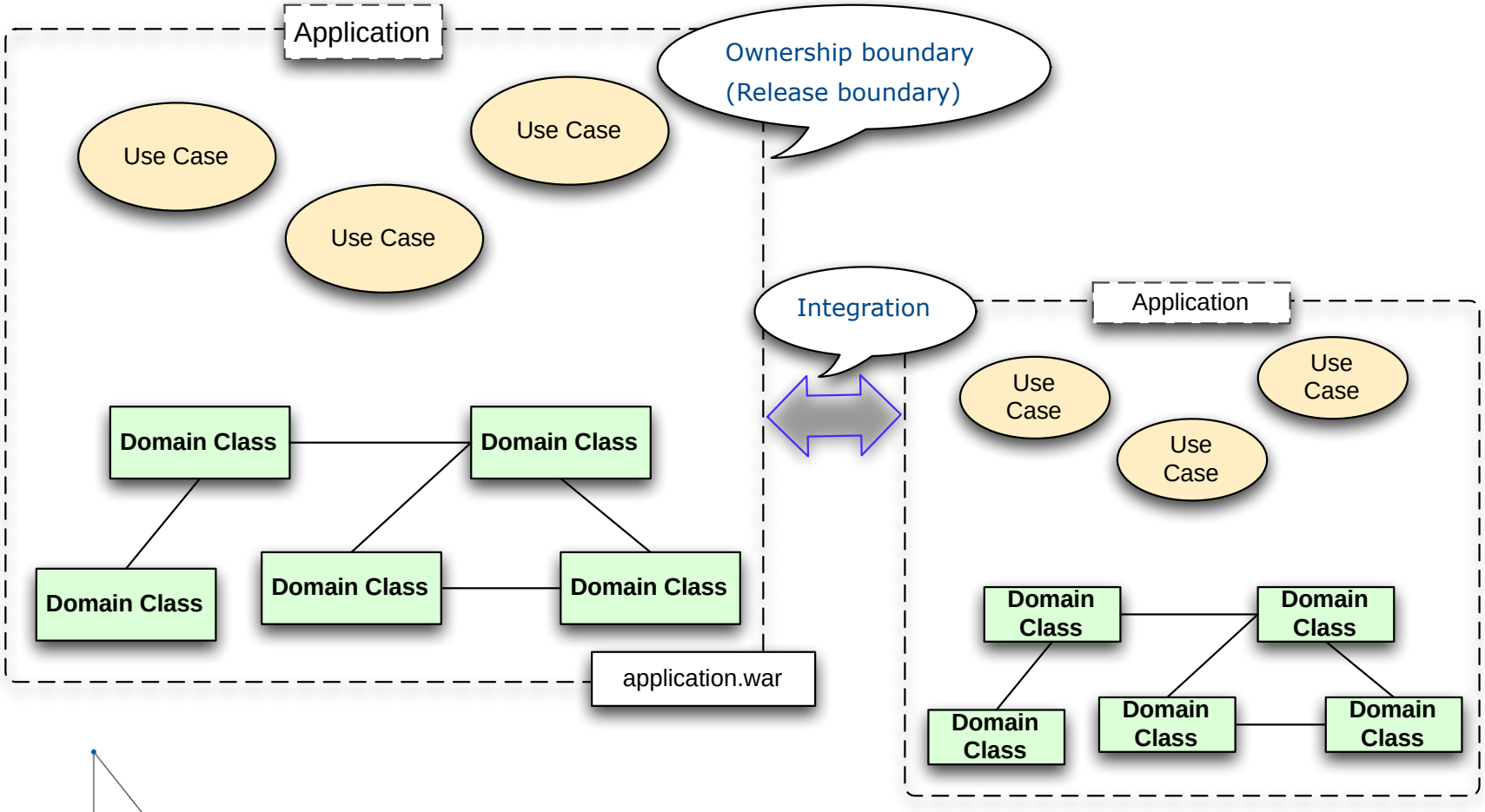
CALLISTA
Enterprise

About DCI

- A New Vision of Object-Oriented Programming
- Origin in Norway and Denmark
  - Trygve Reenskaug (once invented MVC while at Xerox Parc)
  - Jim Coplien

CALLISTA Enterprise

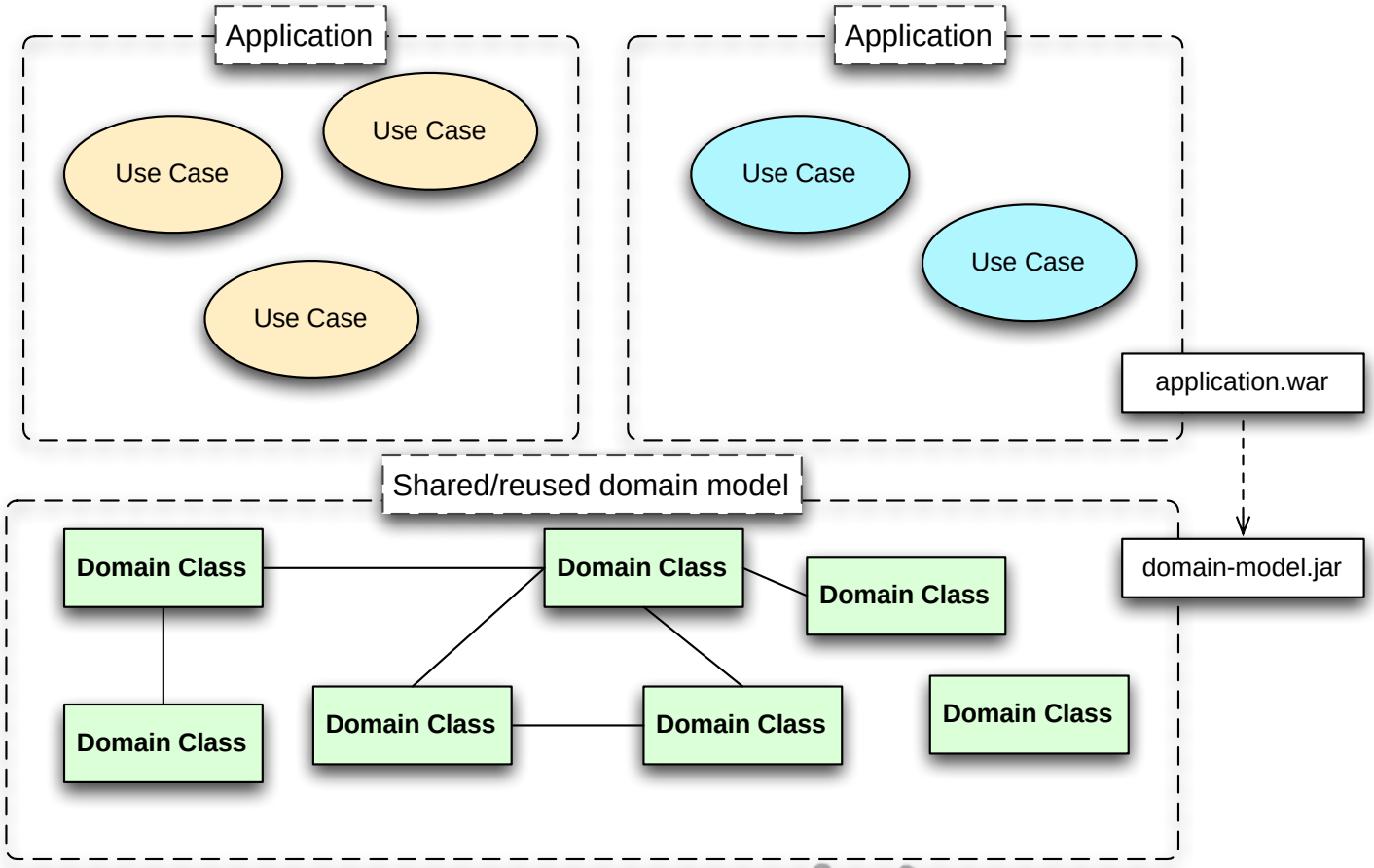
DDD works well for this…



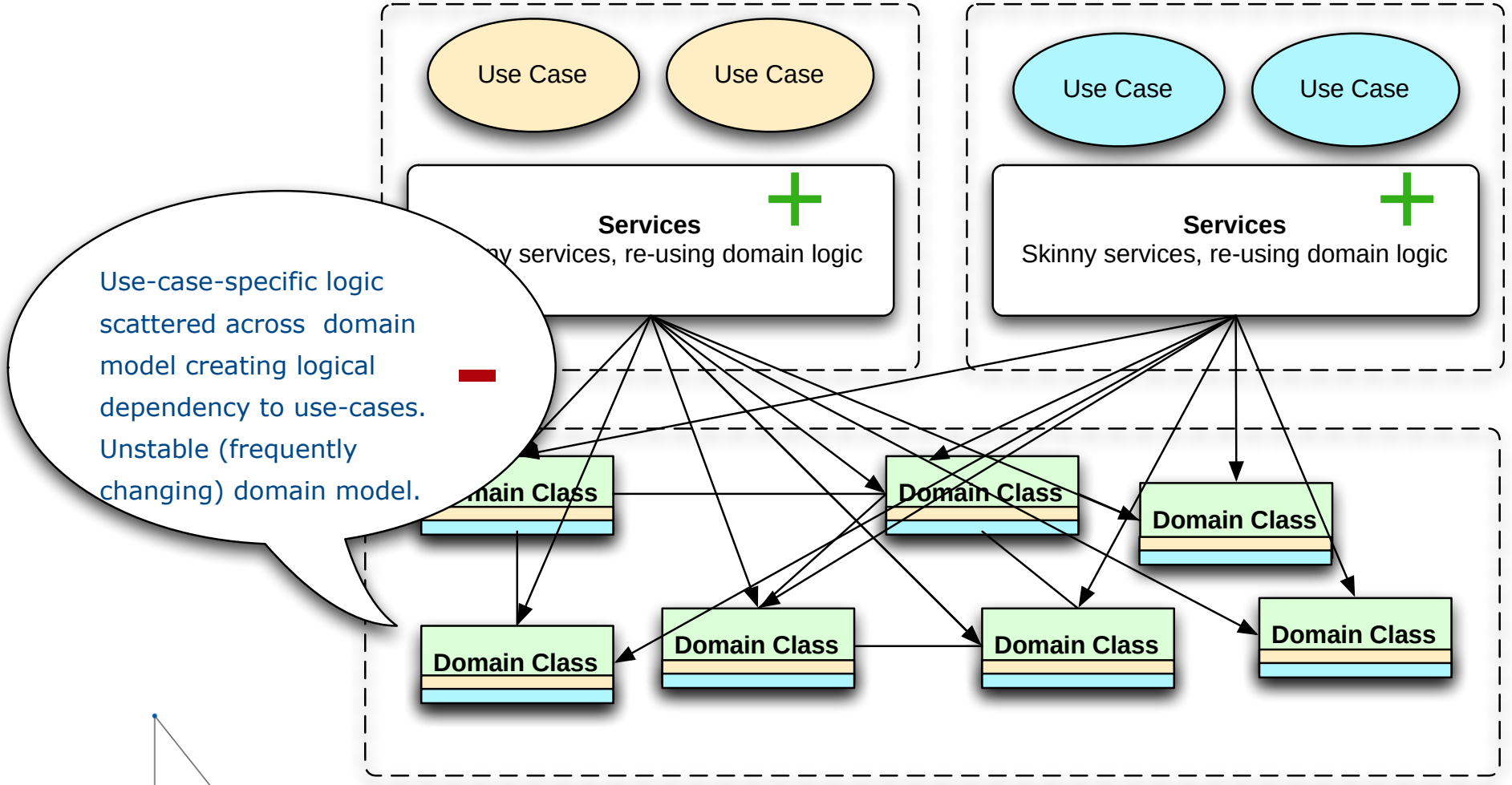
CALLISTA Enterprise

But less well for this…

…which is quite common in midsize- to large systems

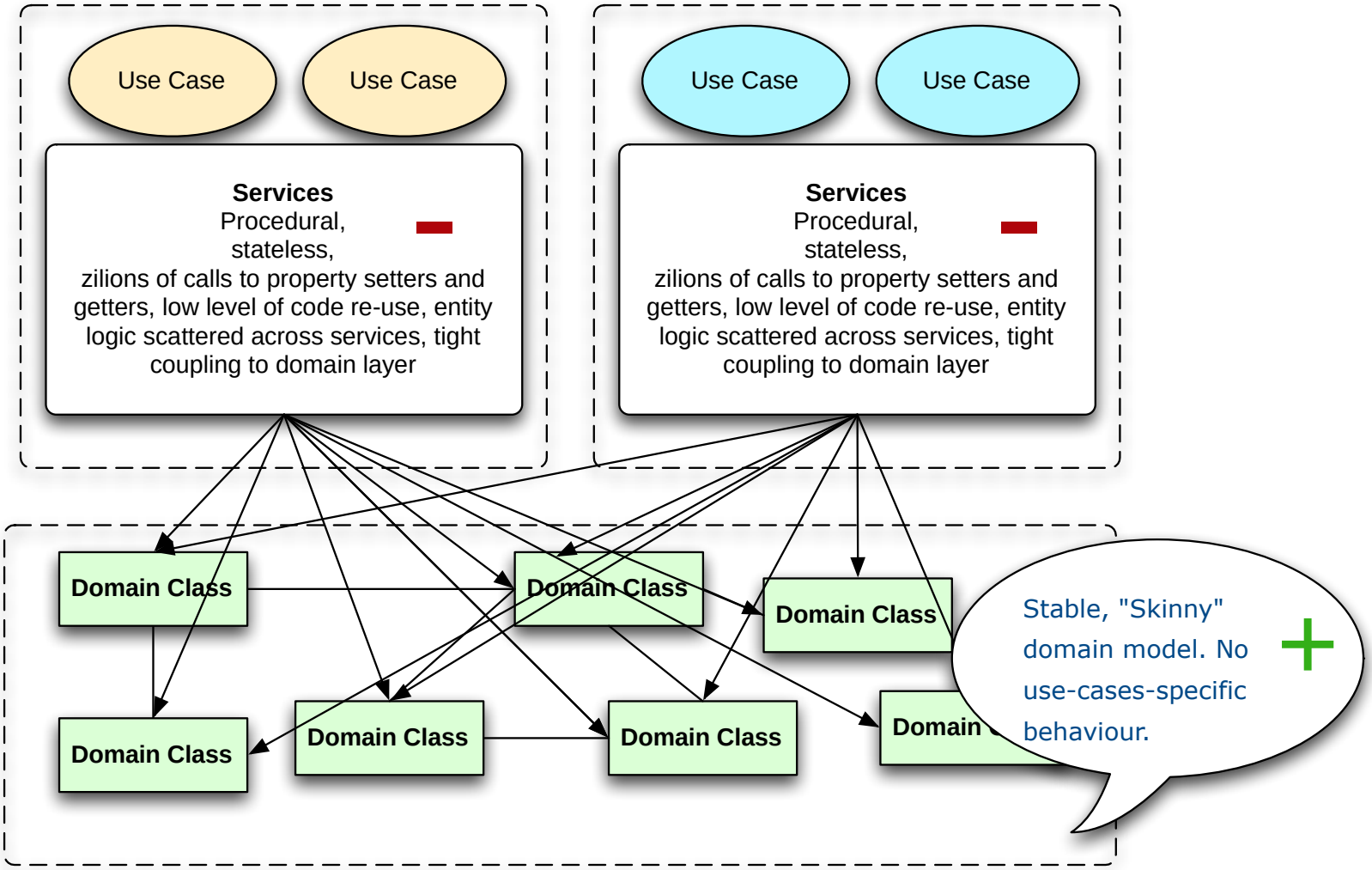


Because you either end up with this…



CALLISTA Enterprise

...or this....



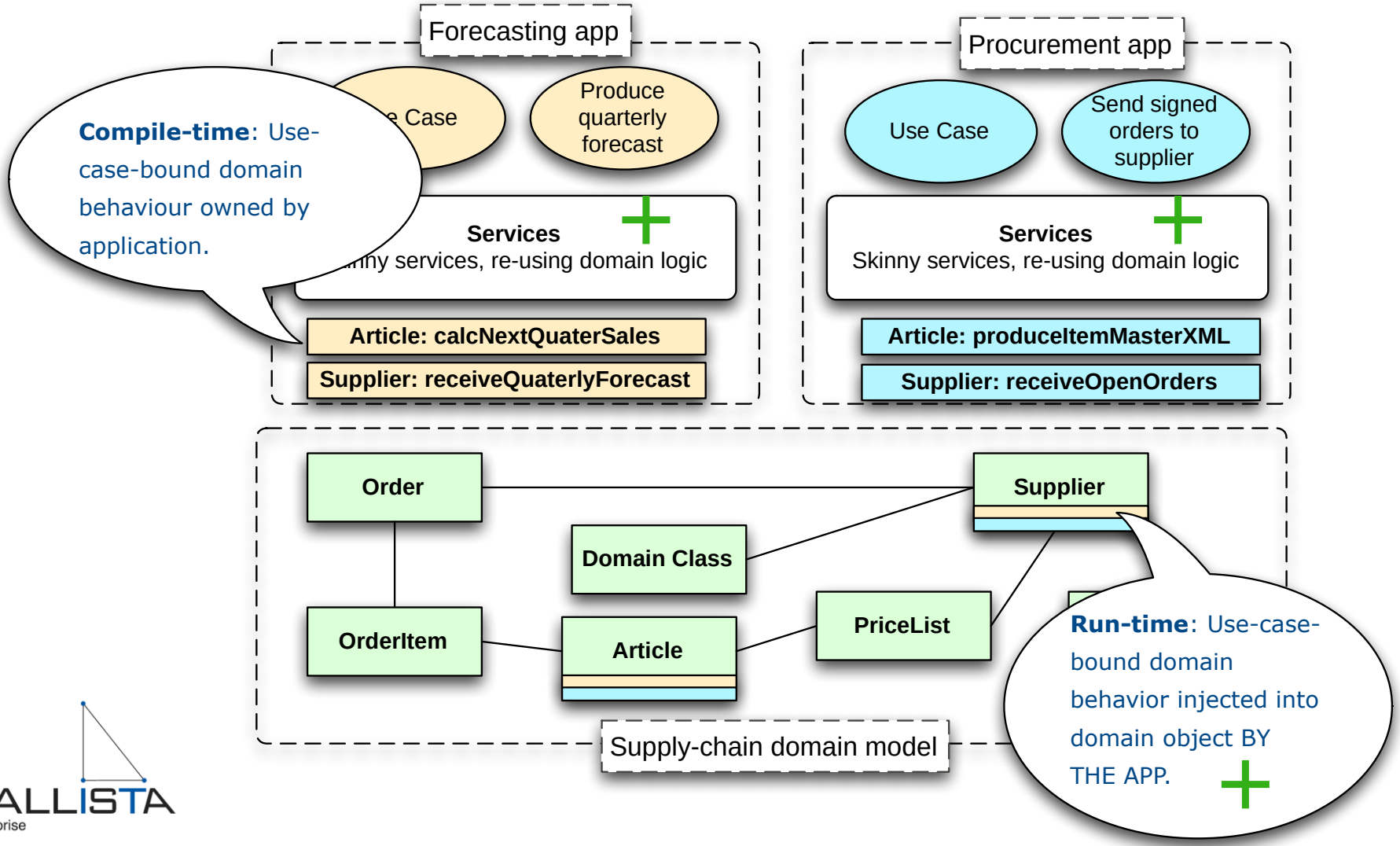
CALLISTA Enterprise

What if....

…use-case logic could be ATTACHED to domain objects when needed but still OWNED by the application modul?

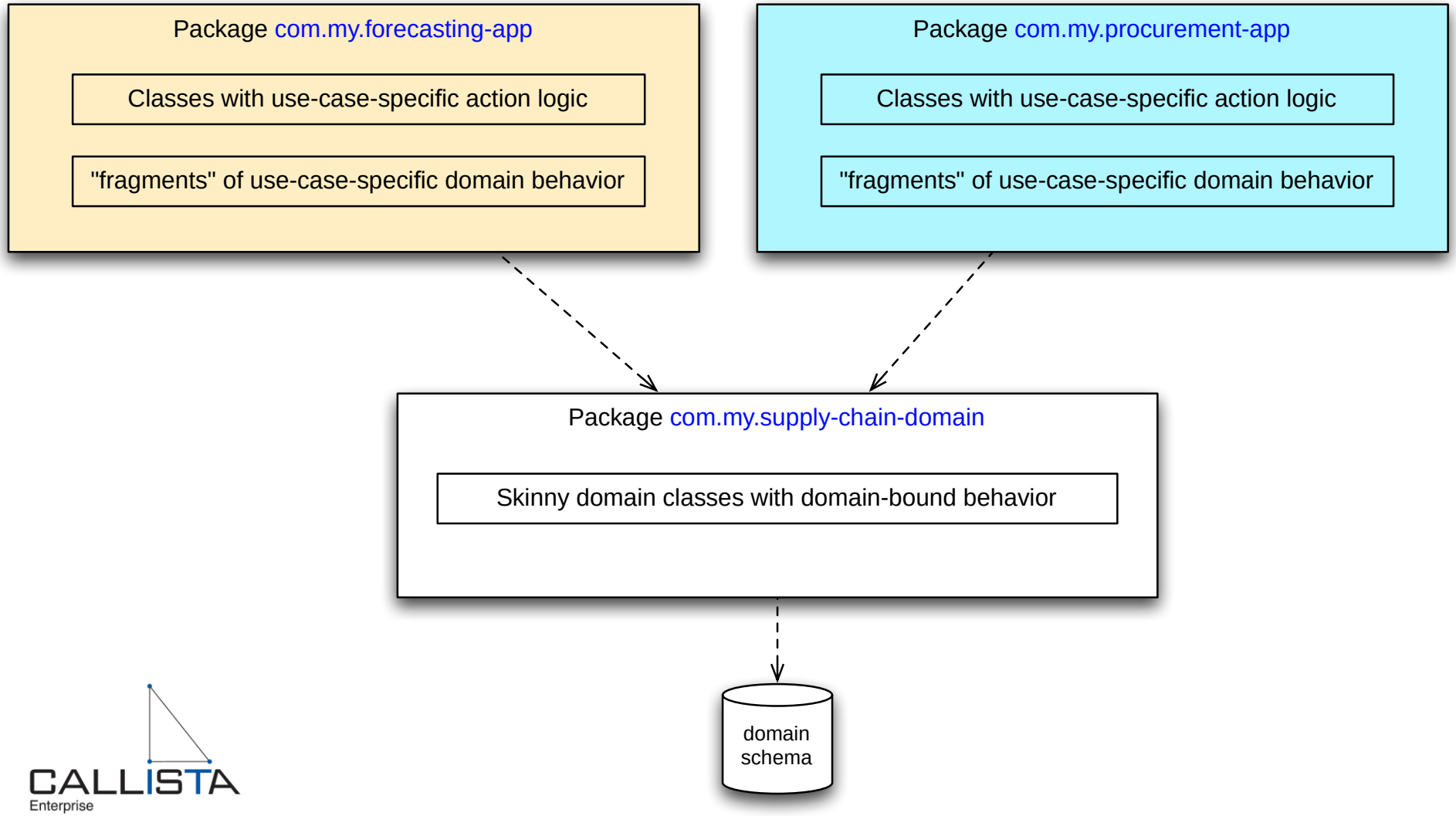
CALLISTA
Enterprise

Like this....



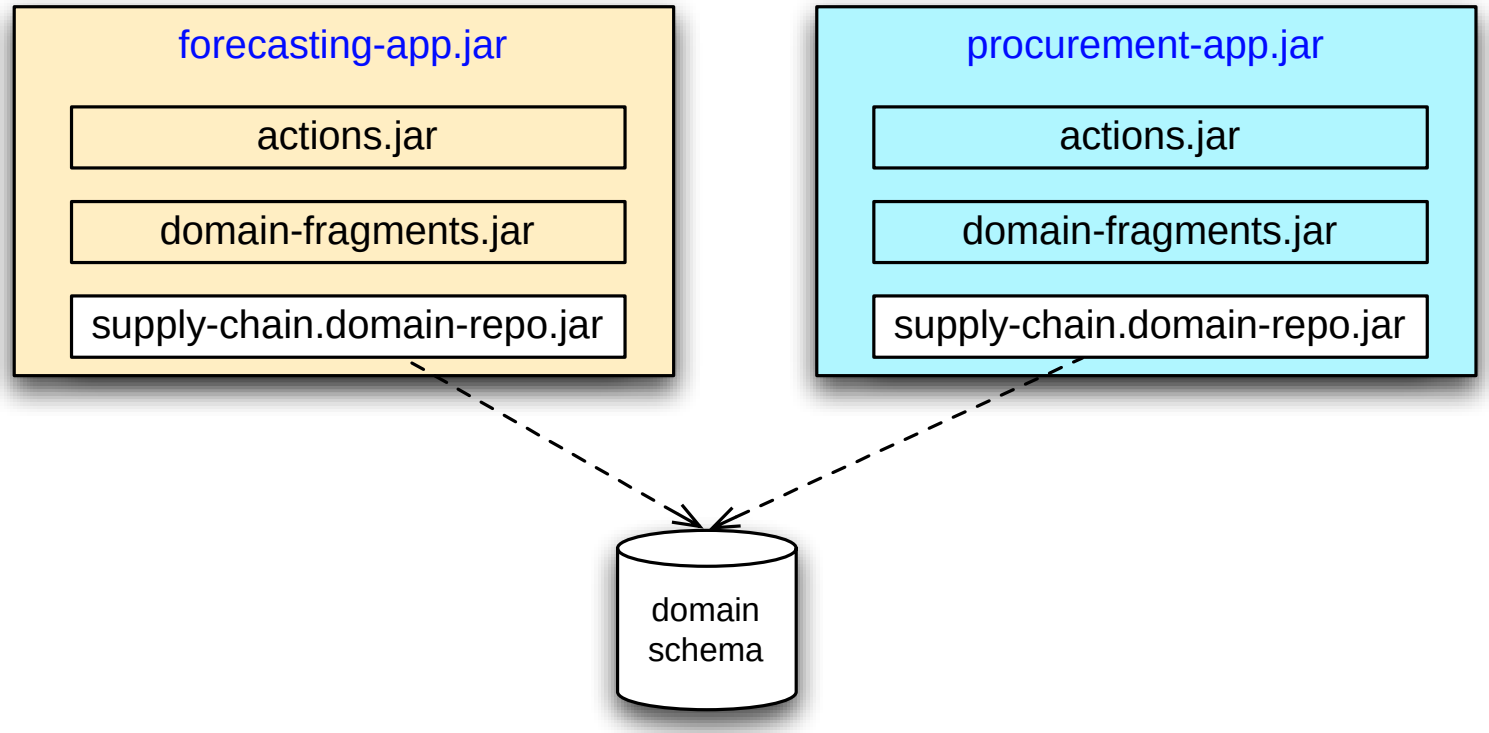
CALLISTA Enterprise

Compile-time…



CALLISTA Enterprise

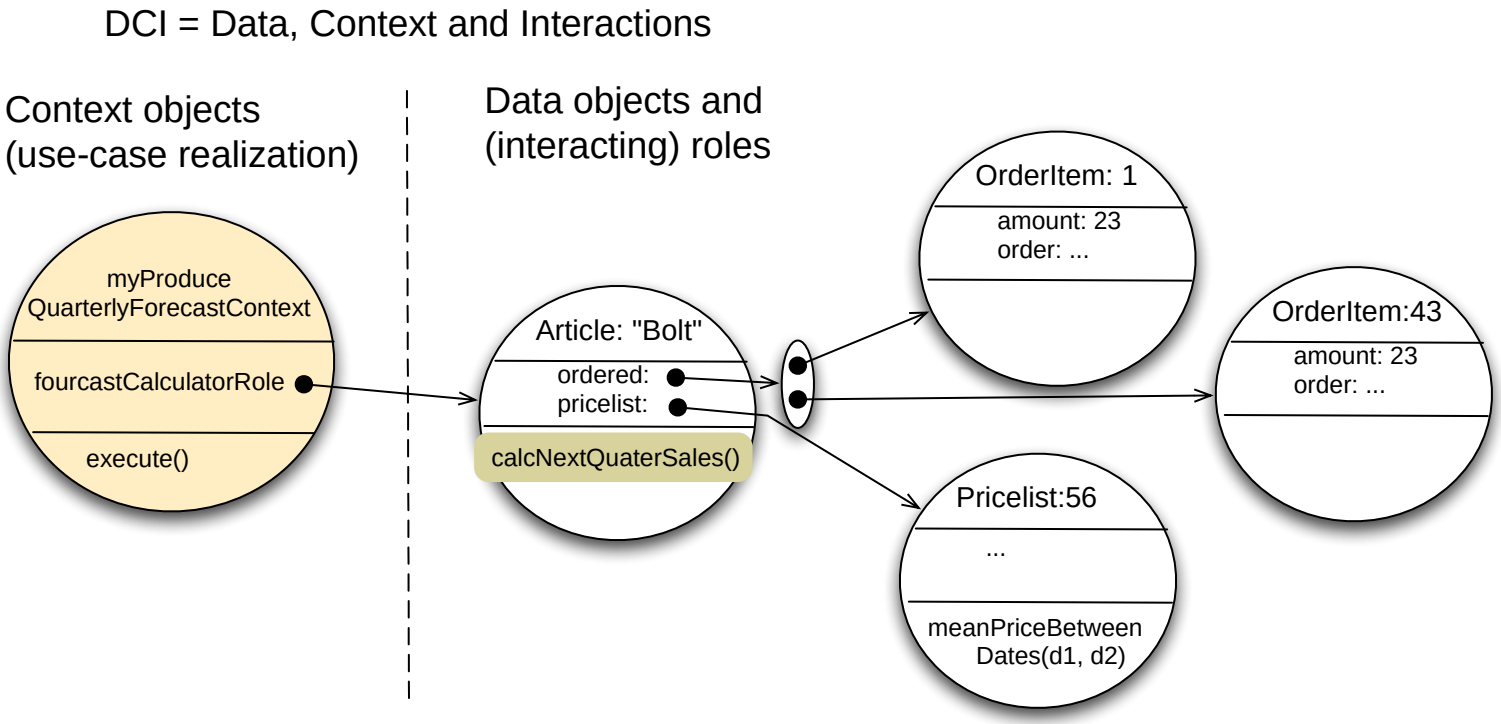
Deploy-time…



CALLISTA
Enterprise

Runtime view....

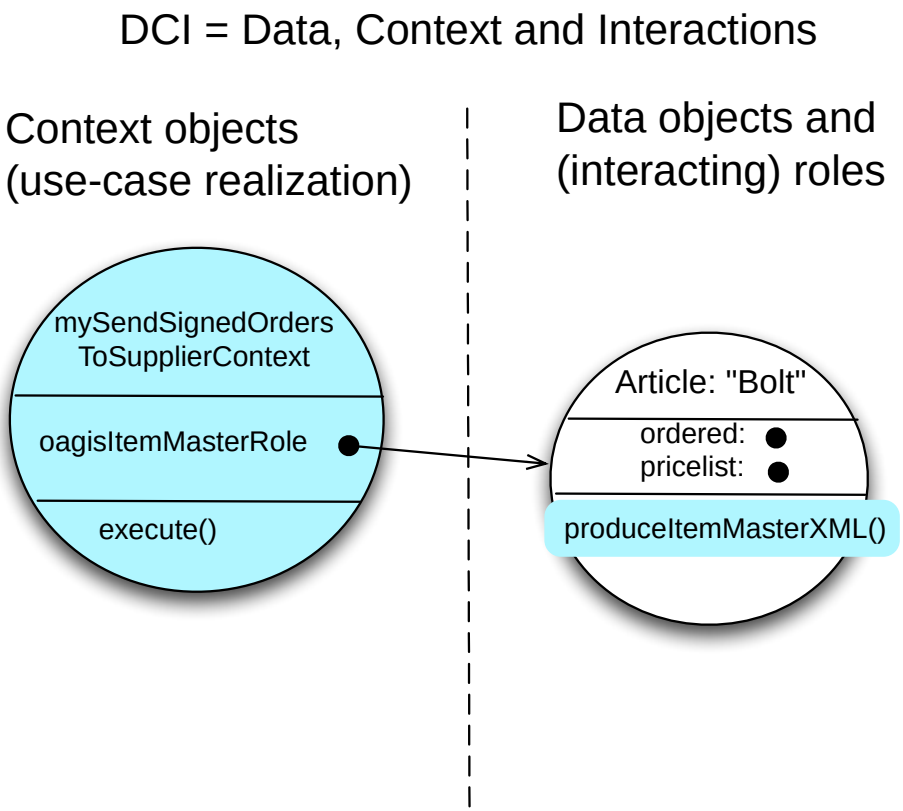
…of an interaction within the ProduceQuarterlyForecast use-case…



CALLISTA
Enterprise

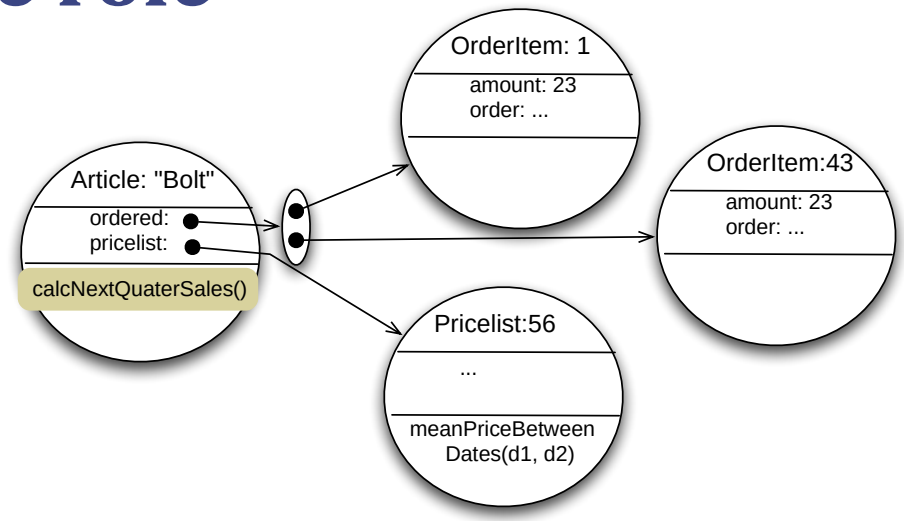
Runtime view....

…of an interaction within the SendSignedOrdersToSupplier use-case…



CALLISTA Enterprise

Taking a look at the role implementation….




```
public class Article {
     List<OrderItem> ordered;
     PriceList pricelist;

     public BigDecimal calcNextQuaterSales() {
          BigDecimal qSales = …
          ordered.findAll {OrderItem item ->
               item.deliveryDate > Calendar.nextQuaterStart
               && item.deliveryDate < Calendar.nextQuaterEnd}.each {OrderItem item ->
               qSales += item.amount
                    * pricelist.meanPriceBetweenDates
                         (Calendar.nextQuaterStart, Calendar.nextQuaterEnd)
          }
          return qSales
     }
}
```

CALLISTA Enterprise

Techically, how can we do this?

…as a Java developer…

- Using Java + an advanced framework
  - Proxies, indirections …
  - There are frameworks!
    - » Qi4J (has a much bigger scope than DCI, but supports DCI. Has an issue with dependency management when nesting contexts from different modules)
    - » “Behaviour Injection” – “DCI as simple as it gets with plain Java”
- Using a JVM-language with matching capabilities
  - Scala (Traits and implicits are good matches to DCI Roles)
  - Groovy (dynamic, which takes you fairly close to “pure” DCI)
- Using legacy languages with matching capabilities
  - C++
  - Objective-C
- **Let’s go for Groovy**
  - **An extension of Java, builds on JDK, I like it….**

CALLISTA Enterprise

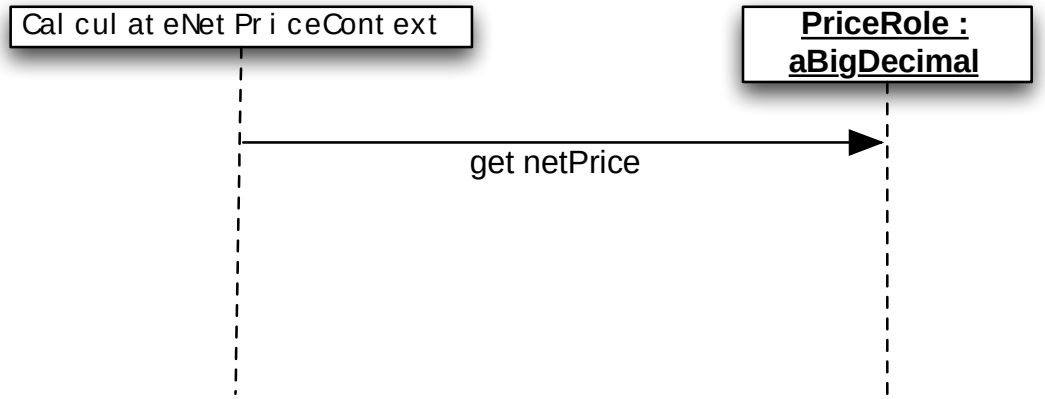
What does Groovy has to offer DCI?

Mechanisms in Groovy to add code to an existing class

- Groovy *Categories*
  - Add “Role methods” dynamically to a class *within an interaction*
  - Not instance-level
- Groovy *Mixins*
  - Add “Role Methods” to class or instance
  - Not scoped to an interaction
- My choice: *Mixins (more “DCI:ish)*
  - *In DCI, a role is acted by an instance*
  - *Categories (when using AST-transforms) have limitations*
  - *Minus: Using Mixins is a bit more “techie”*

CALLISTA Enterprise

Groovy Mixin Simple Sample



Context: CalculateNetPriceContext
Data: a BigDecimal
Interaction: netPrice on Role PriceRole

CALLISTA
Enterprise

The code – Define the Role (the mixin)

```
class PriceRole {
     BigDecimal getNetPrice() {
          return this * 0.8
     }
}
```

```
class CalculateNetPriceContext {
     def priceRole

     CalculateNetPriceContext(BigDecimal amount) {
          amount.metaClass.mixin(PriceRole)
          priceRole = amount
     }

     BigDecimal executeContext() {
          return priceRole.netPrice
     }
}

println new CalculateNetPriceContext(100.00).executeContext()
```

CALLISTA Enterprise

The code – Context assigns role to data

```
class PriceRole {
     BigDecimal getNetPrice() {
          return this * 0.8
     }
}

class CalculateNetPriceContext {
     def priceRole

     CalculateNetPriceContext(BigDecimal amount) {
          amount.metaClass.mixin(PriceRole)
          priceRole = amount
     }

     BigDecimal executeContext() {
          return priceRole.netPrice
     }
}

println new CalculateNetPriceContext(100.00).executeContext()
```

CALLISTA Enterprise

The code – method to execute interaction

```
class PriceRole {
     BigDecimal getNetPrice() {
          return this * 0.8
     }
}

class CalculateNetPriceContext {
     def priceRole

     CalculateNetPriceContext(BigDecimal amount) {
          amount.metaClass.mixin(PriceRole)
          priceRole = amount
     }

     BigDecimal executeContext() {
          return priceRole.netPrice
     }
}

println new CalculateNetPriceContext(100.00).executeContext()
```

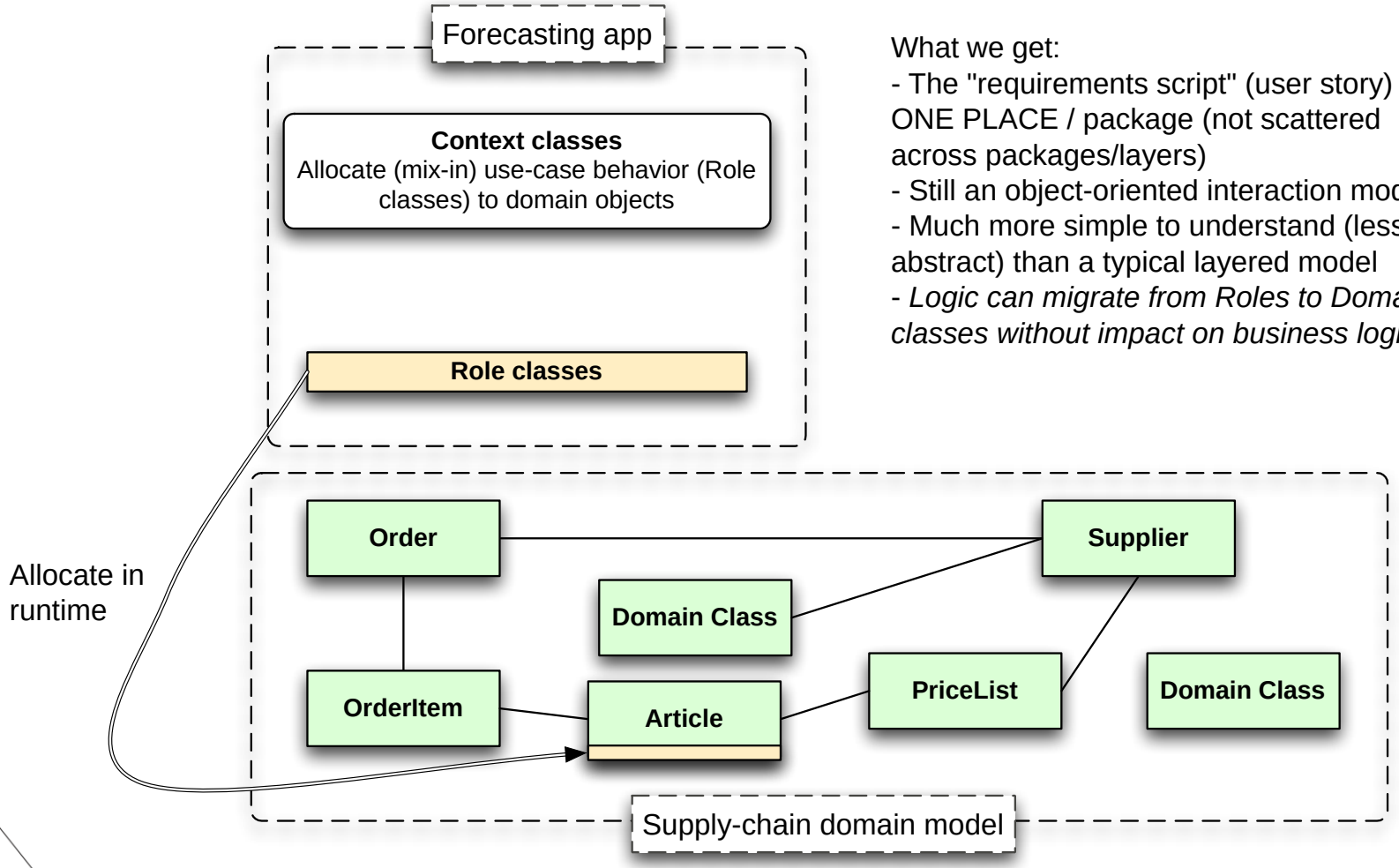
The code – ask the context to conduct the interaction

```
class PriceRole {
     BigDecimal getNetPrice() {
          return this * 0.8
     }
}

class CalculateNetPriceContext {
     def priceRole

     CalculateNetPriceContext(BigDecimal amount) {
          amount.metaClass.mixin(PriceRole)
          priceRole = amount
     }

     BigDecimal executeContext() {
          return priceRole.netPrice
     }
}
```

```
println new CalculateNetPriceContext(100.00).executeContext()
```

CALLISTA Enterprise

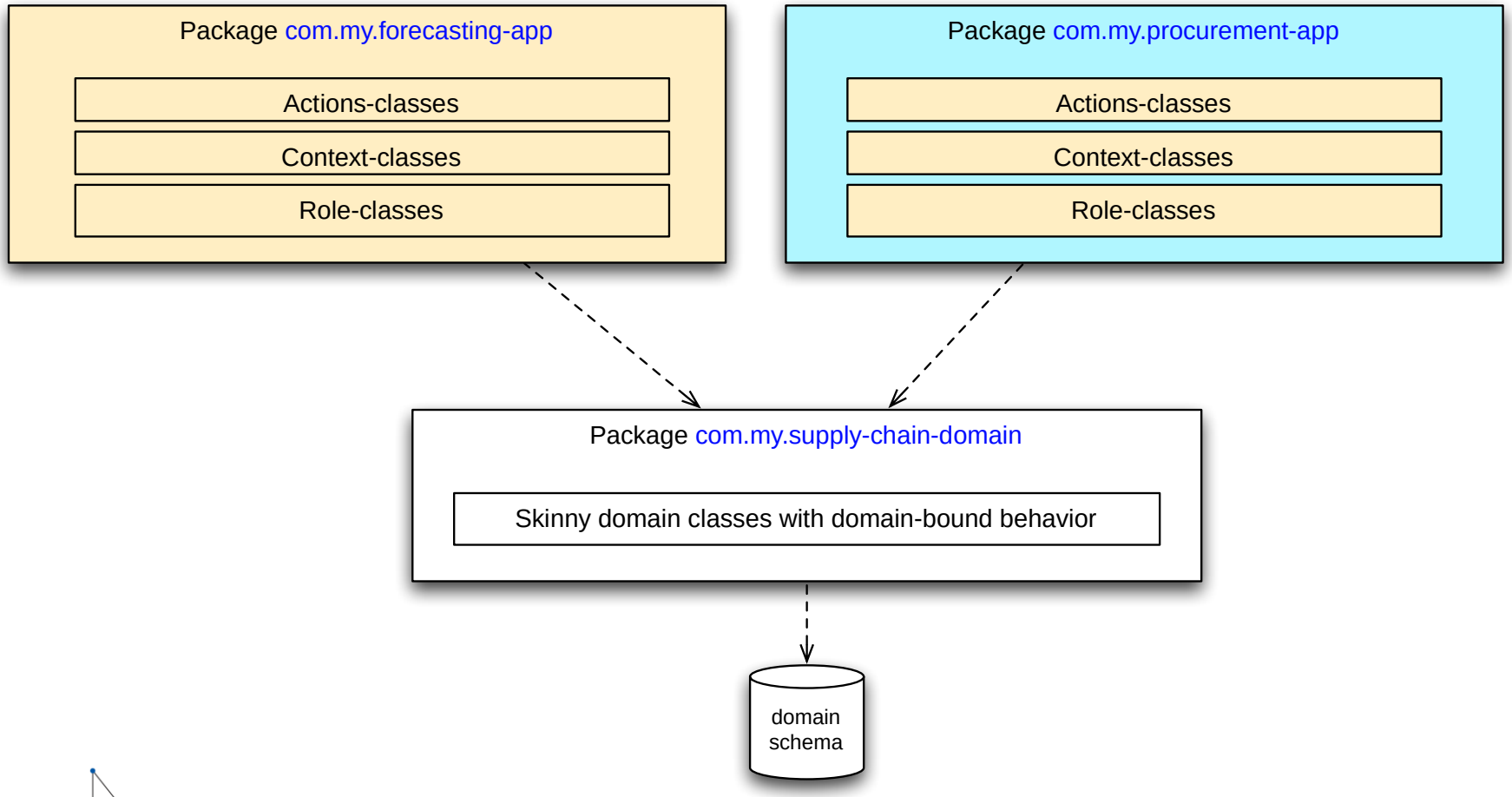
Let's revisit the BIG system....



What we get:
- The "requirements script" (user story) is in ONE PLACE / package (not scattered across packages/layers)
- Still an object-oriented interaction model
- Much more simple to understand (less abstract) than a typical layered model
- *Logic can migrate from Roles to Domain classes without impact on business logic*

CALLISTA Enterprise

Clean dependency graph!



CALLISTA Enterprise

All is good so far – but what about…

- Dependency injection
  - Less stateless objects but works as usual
- Testing
  - A Mixin needs to be bound to target data class to be tested (if logic depends on the target class)
- Debugging
  - Groovy debugging works nice in major IDE:s (e.g. Eclipse)
- Nesting / layers / hierarchies
  - Role-nesting across “to-one” relationships
  - Context-nesting for use-case-level re-use (“Habits” rather than “Use-cases”)

CALLISTA
Enterprise

What did I use it for?

- Domain-model
  - JAXB-classes generated from a metadata exchange format (service repository)
- Use-case
  - Generate Web-service metadata on the fly (WSDL) from the logical metadata model
- Architecture
  - Context class for assigning WS-metadata roles to model-classes of the logical service model
  - Role implemented in Groovy
  - An incarnation of it is here: http://wsdltools.appspot.com/

CALLISTA
Enterprise

When doesn't DCI make sense?

- It doesn't pay off when the use-case is a "mirror" of the domain/entity model
  - Plain CRUD
  - When the problem is not communicated in terms of processes, algorithms, transaction scripts, activities etc

CALLISTA
Enterprise

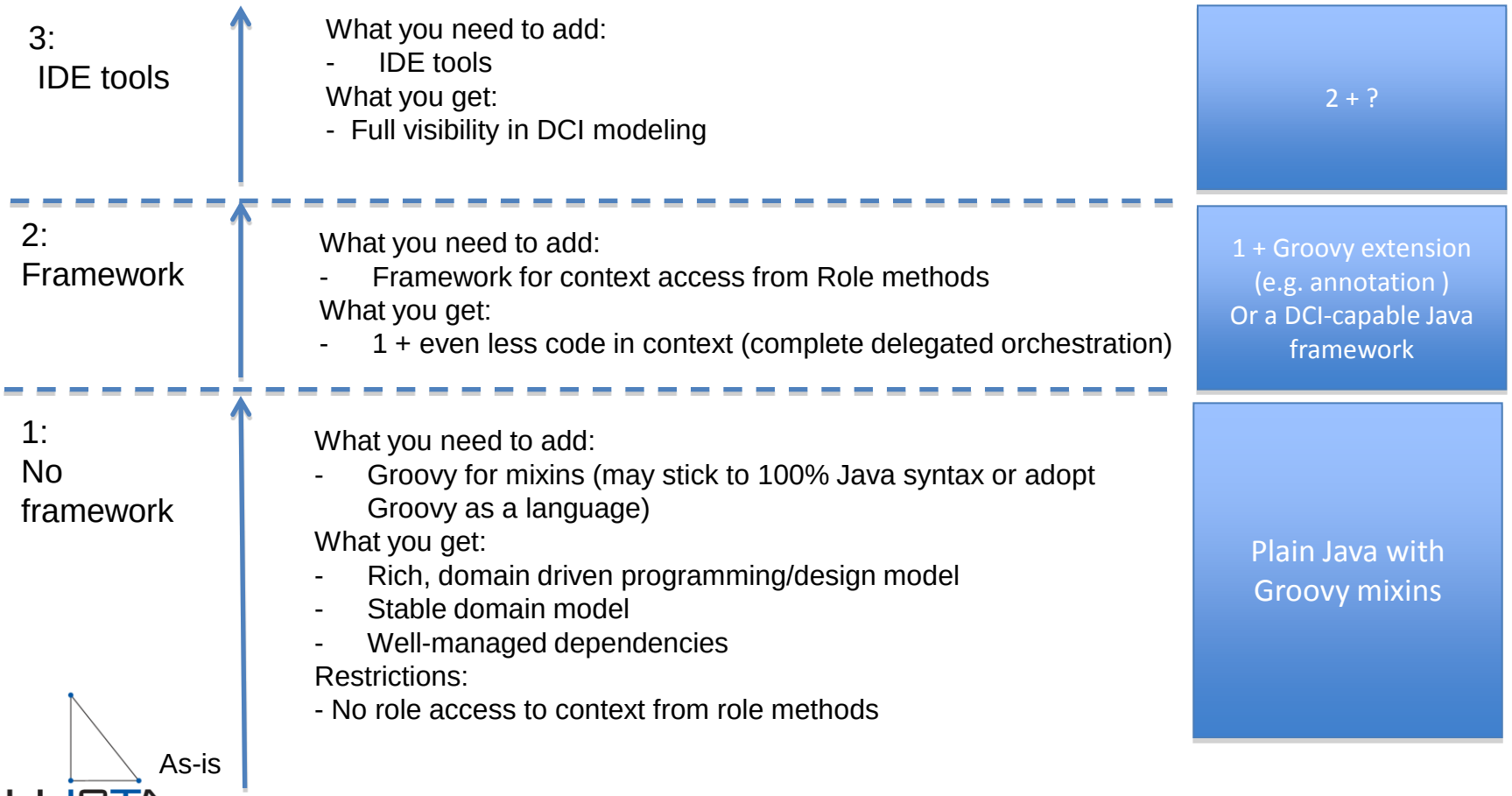
Did I teach you DCI?

Not sure, really…but I'm convinced it is useful

- The DCI vision is a composition of several concepts
  - Picking few or even most of the concepts may not result in DCI nirvana
- Nirvana DCI is still a research topic
- Pragmatic DCI with Groovy may not qualify as a DCI implementation
  - But it brings a lot of value to my design work

CALLISTA Enterprise

Possible strategies for adoption



CALLISTA Enterprise

Thanks for listening! Questions?

CALLISTA
Enterprise

References

DDD
-http://domaindrivendesign.org/resources/ddd_terms

DCI
-Vision/definition: http://www.artima.com/articles/dci_vision.html
-With Java: http://www.maxant.co.uk/tools/archive/maxant-dci-tools-1.1.0.pdf
-With Qi4J: http://www.qi4j.org/
-Öredev-talk by James Coplien: http://vimeo.com/8235574
-Discussion group (Google group): http://groups.google.com/group/object-composition/

Groovy
-Categories: http://docs.codehaus.org/display/GROOVY/Groovy+Categories
-Mixins:
http://archive.groovy.codehaus.org/lists/dev@groovy.codehaus.org/msg/4cf0f24c0804081656l5aed67b5hf34fc73cbea375b0@mail.gmail.com
-Advanced meta programming: http://www.slideshare.net/zenMonkey/metaprogramming-techniques-in-groovy-and-grails

CALLISTA
Enterprise