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

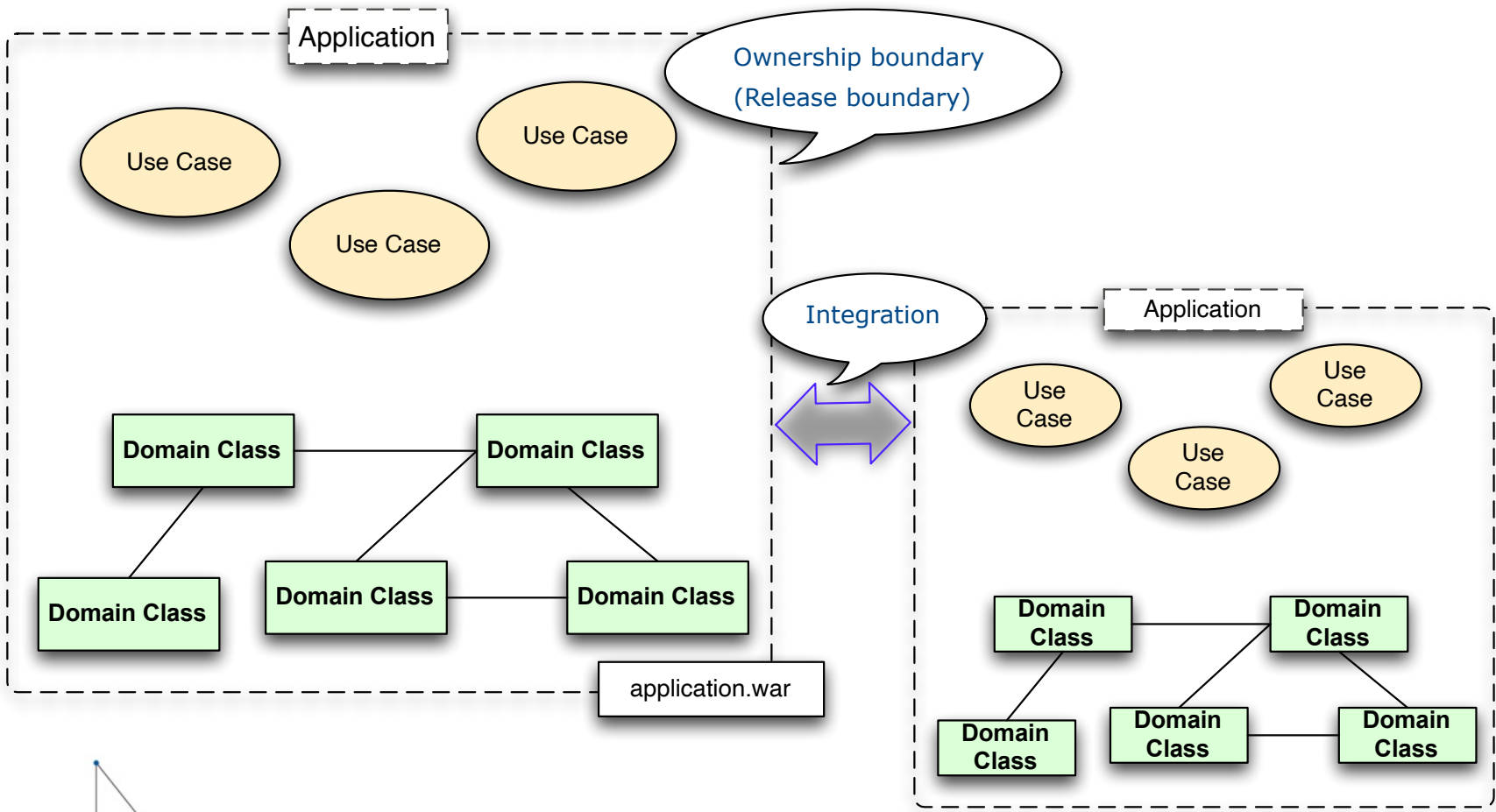- A little (very little) of language geekiness

- 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
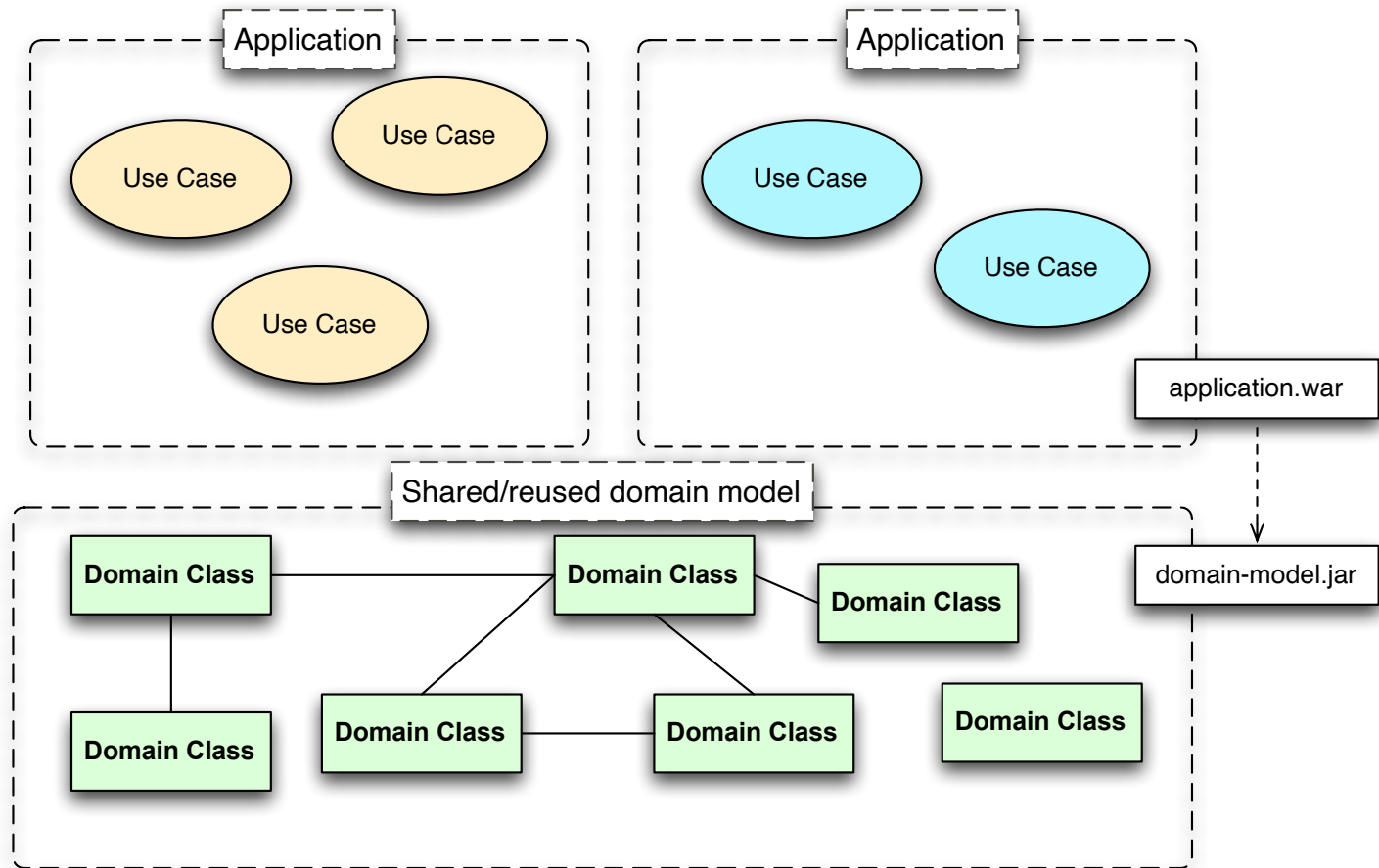   - Trygve Reenskaug (once invented MVC while at Xerox Parc)
    - Jim Coplien

CALLISTA
Enterprise

DDD works well for this...

Avoid Cluttered Domain Models with DCI and Groovy   © 2011 Callista Enterprise | www.callistaenterprise.se

But less well for this...

...which is quite common in midsize- to large systems

Avoid Cluttered Domain Models with DCI and Groovy © 2011 Callista Enterprise | www.callistaenterprise.se

Because you either end up with this...

Avoid Cluttered Domain Models with DCI and Groovy © 2011 Callista Enterprise | www.callistaenterprise.se

...or this....

Use Case   Use Case

Services
Procedural,
stateless,
zilions of calls to property setters and
getters, low level of code re-use, entity
logic scattered across services, tight
coupling to domain layer

Use Case   Use Case

Services
Procedural,
stateless,
zilions of calls to property setters and
getters, low level of code re-use, entity
logic scattered across services, tight
coupling to domain layer

Domain Class   **Domain Class**   **Domain Class**

Domain Class   **Domain Class**   **Domain Class**   **Domain Class**

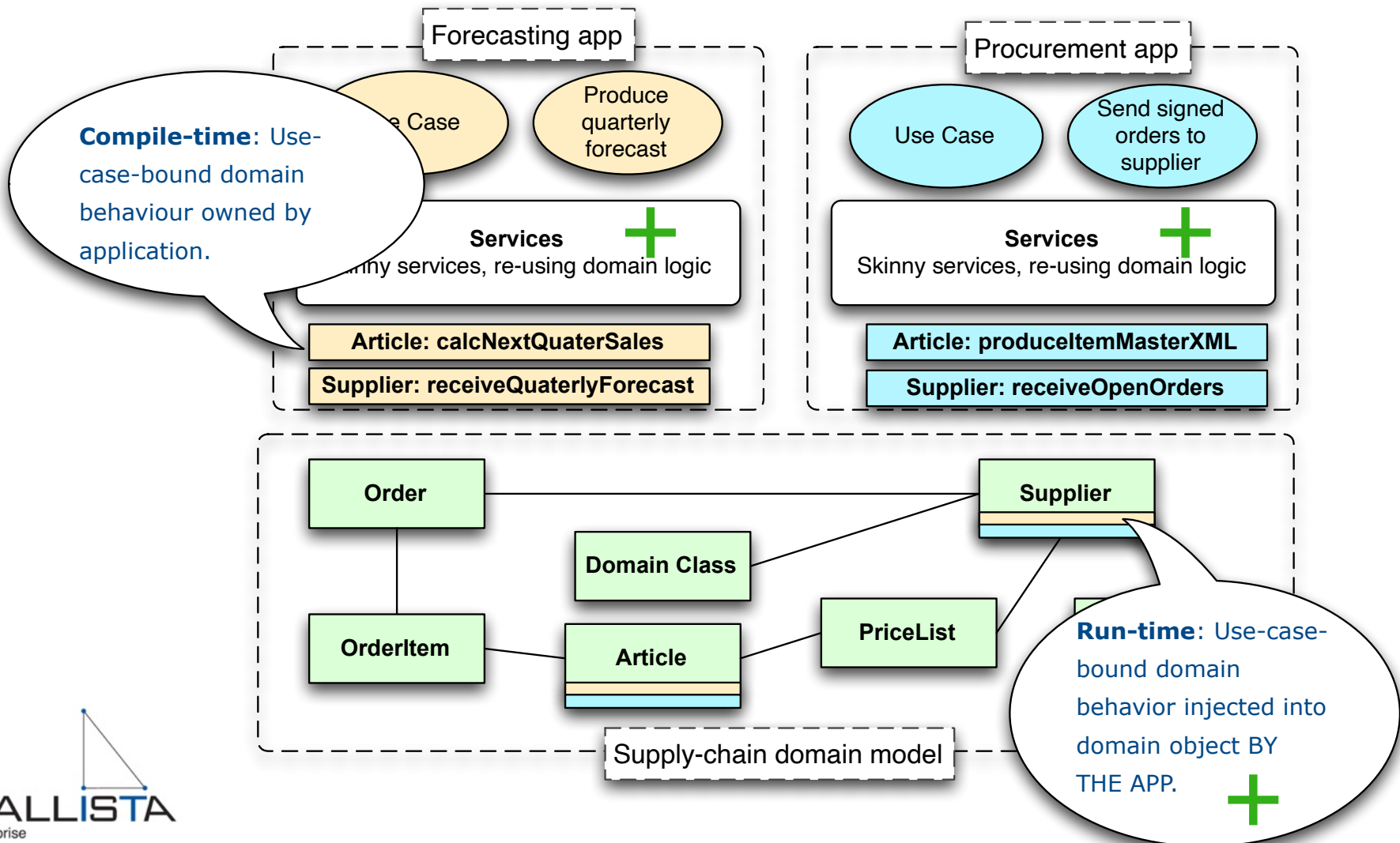Stable, "Skinny" domain model. No use-cases-specific behaviour.

CALLISTA
Enterprise

What if….

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

Like this....

Forecasting app

Use Case

Produce quarterly forecast

Compile-time: Use-case-bound domain behaviour owned by application.

Services
Skinny services, re-using domain logic

➕

Article: calcNextQuaterSales

Supplier: receiveQuaterlyForecast

Procurement app

Use Case

Send signed orders to supplier

Services
Skinny services, re-using domain logic

➕

Article: produceItemMasterXML

Supplier: receiveOpenOrders

Order

Supplier

Domain Class

OrderItem

Article

PriceList

Run-time: Use-case-bound domain behavior injected into domain object BY THE APP.

➕

Supply-chain domain model

CALLISTA
Enterprise

Compile-time...

Package com.my.forecasting-app

Classes with use-case-specific action logic

"fragments" of use-case-specific domain behavior

Package com.my.procurement-app
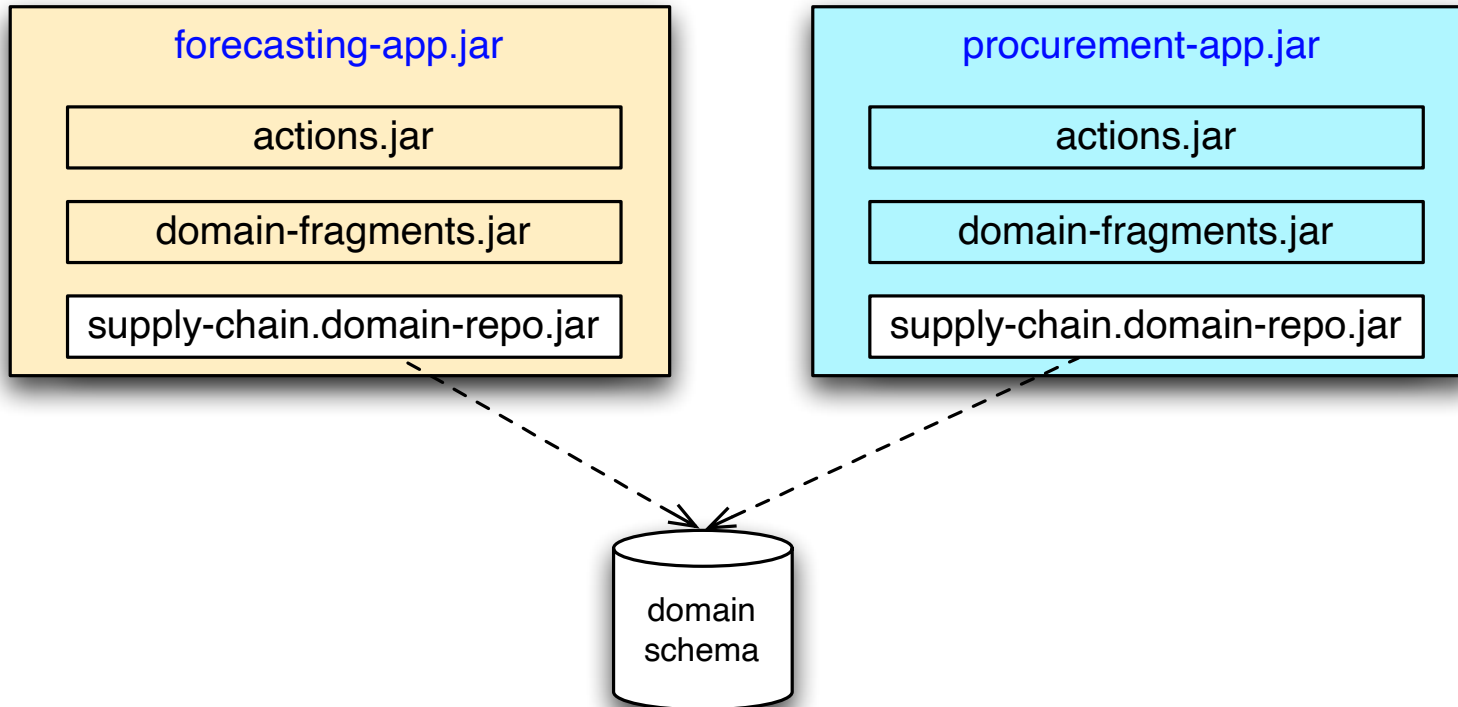
Classes with use-case-specific action logic

"fragments" of use-case-specific domain behavior

Package com.my.supply-chain-domain

Skinny domain classes with domain-bound behavior

domain schema

CALLISTA
Enterprise

Deploy-time...

forecasting-app.jar

actions.jar

domain-fragments.jar

supply-chain.domain-repo.jar

procurement-app.jar

actions.jar

domain-fragments.jar

supply-chain.domain-repo.jar
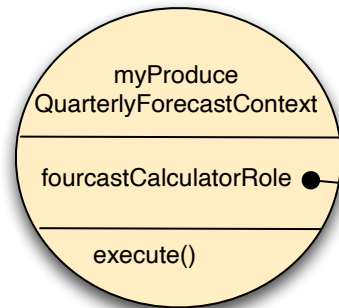
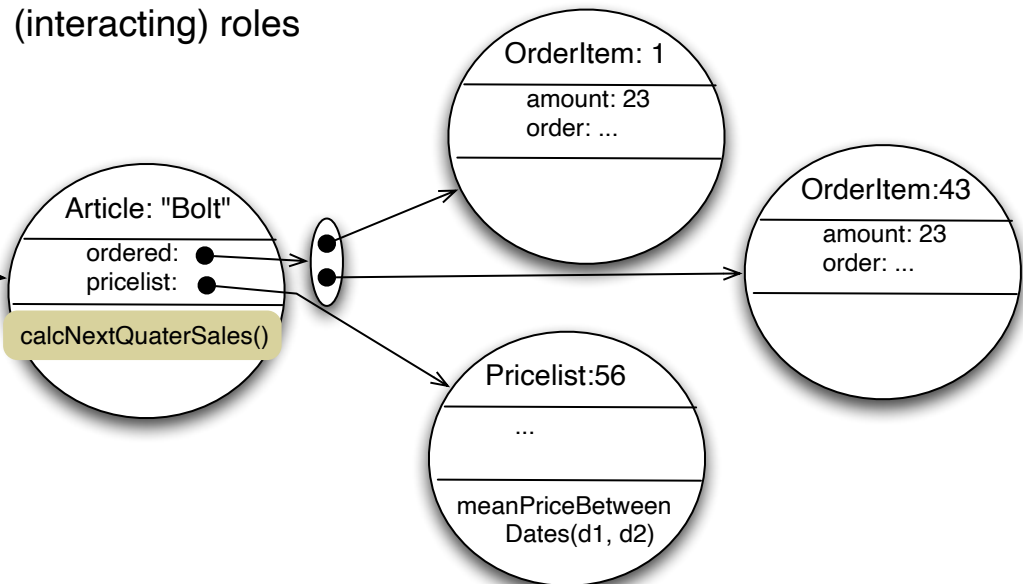domain schema

CALLISTA
Enterprise

Runtime view....

...of an interaction within the ProduceQuarterlyForecast use-case...

DCI = Data, Context and Interactions

Context objects
(use-case realization)

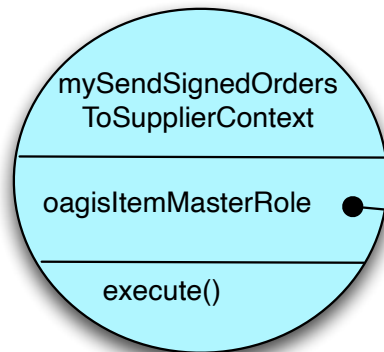Data objects and
(interacting) roles

myProduce QuarterlyForecastContext

fourcastCalculatorRole ●

execute()

Article: "Bolt"

ordered: ●
pricelist: ●

calcNextQuaterSales()

OrderItem: 1

amount: 23
order: ...

OrderItem:43

amount: 23
order: ...

Pricelist:56

...

meanPriceBetween
Dates(d1, d2)

CALLISTA
Enterprise

Runtime view....

DCI = Data, Context and Interactions

Context objects
(use-case realization)

Data objects and
(interacting) roles

mySendSignedOrders
ToSupplierContext
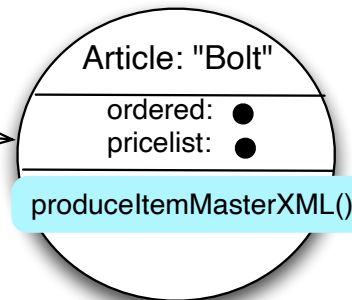___
oagisItemMasterRole ●
___
execute()

Article: "Bolt"
___
ordered: ●
pricelist: ●
produceItemMasterXML()

Avoid Cluttered Domain Models with DCI and Groovy © 2011 Callista Enterprise | www.callistaenterprise.se

CALLISTA
Enterprise

Taking a look at the role implementation….
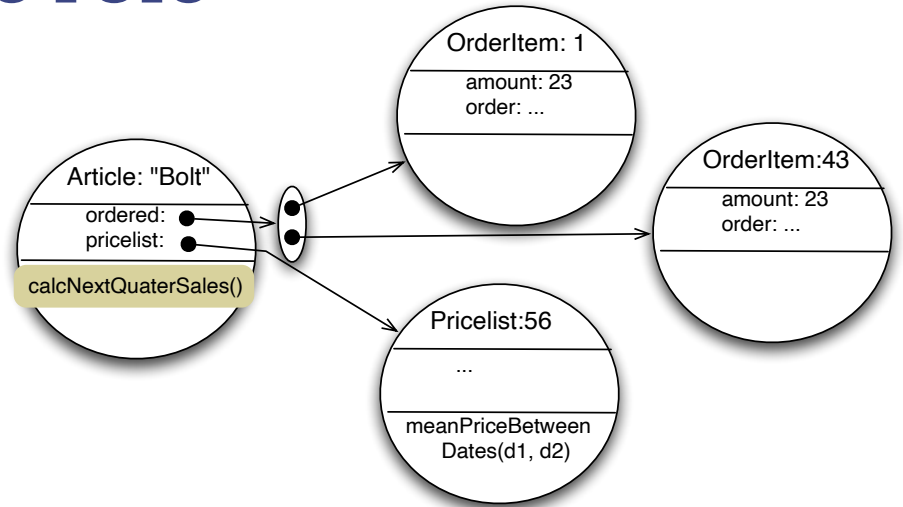

```
public class Article {
    List<OrderItem> ordered;
    PriceList pricelist;

    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….**
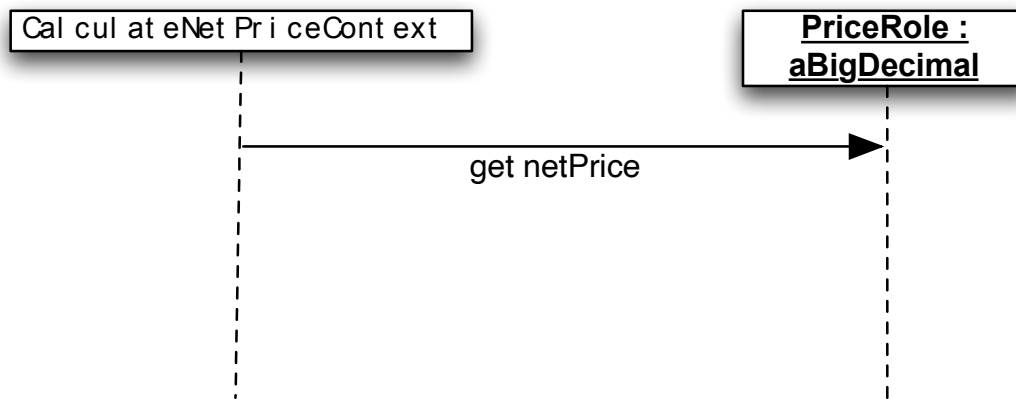
CALLISTA
Enterprise

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

```
Cal cul at eNet Pr i ceCont ext                    PriceRole :
                                                   aBigDecimal

              │                                         │
              │─────────── get netPrice ───────────────▶│
              │                                         │
              │                                         │
```

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

```groovy
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()
```

CALLISTA
Enterprise

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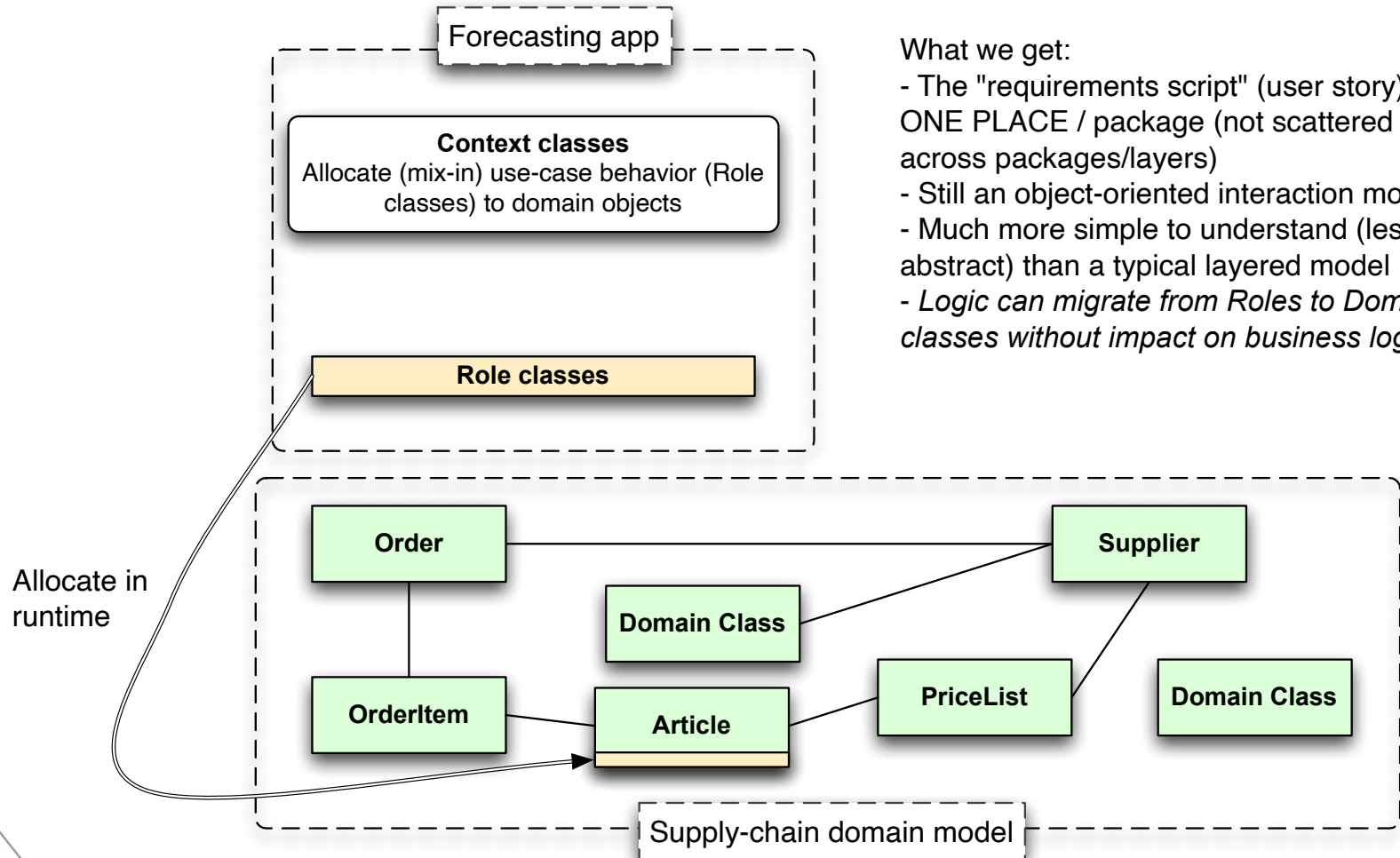
```
println new CalculateNetPriceContext(100.00).executeContext()
```

CALLISTA Enterprise

Let's revisit the BIG system….

Forecasting app

Context classes
Allocate (mix-in) use-case behavior (Role classes) to domain objects

Role classes

Allocate in runtime

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*

Order

Supplier

Domain Class

PriceList

Domain Class

OrderItem

Article

Supply-chain domain model

CALLISTA
Enterprise

Clean dependency graph!

Package com.my.forecasting-app

Actions-classes
Context-classes
Role-classes

Package com.my.procurement-app

Actions-classes
Context-classes
Role-classes

Package com.my.supply-chain-domain

Skinny domain classes with domain-bound behavior

domain
schema

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
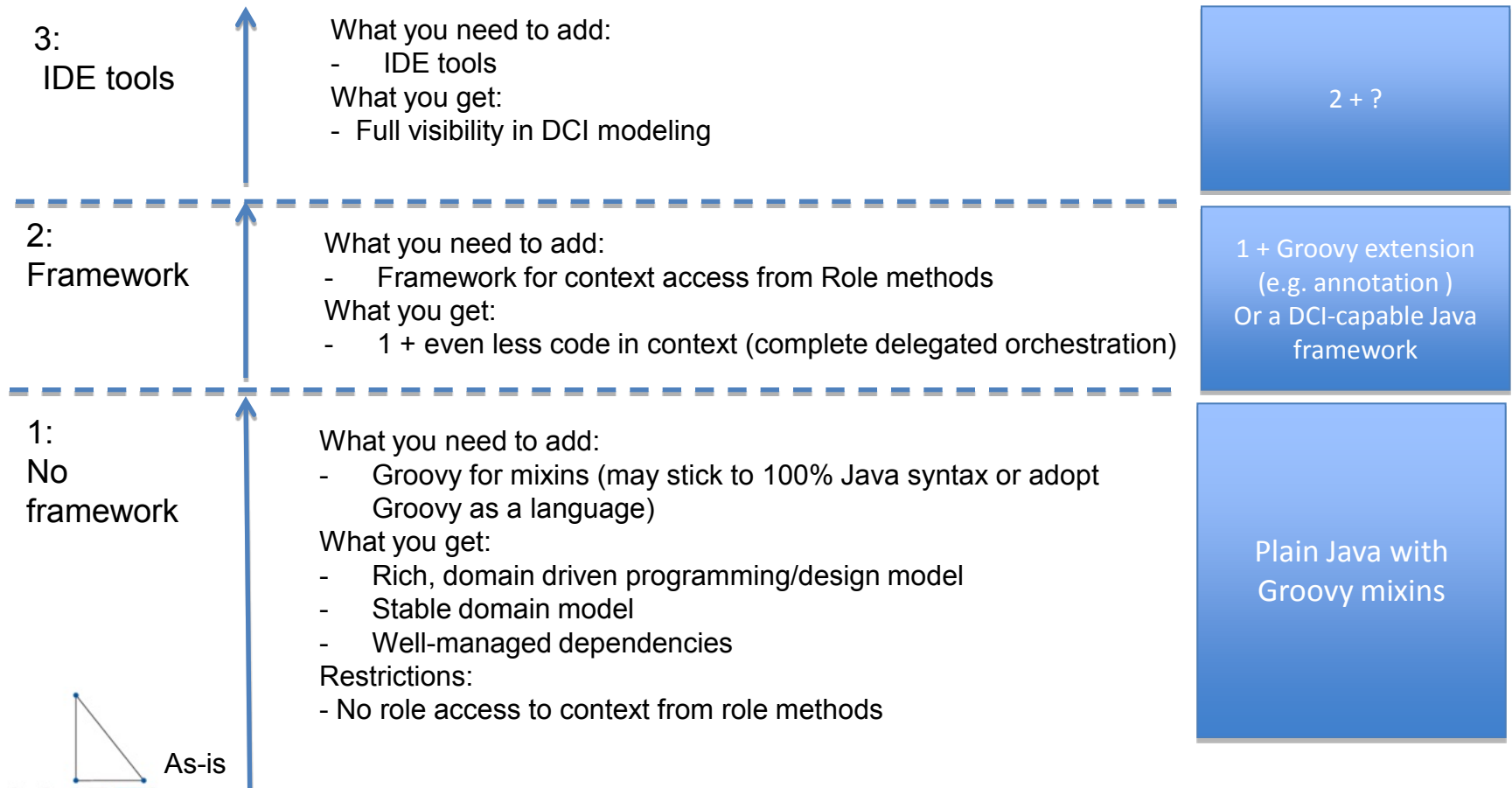Enterprise

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

3:
IDE tools

What you need to add:
- IDE tools

What you get:
- Full visibility in DCI modeling

2 + ?

2:
Framework

What you need to add:
- Framework for context access from Role methods

What you get:
- 1 + even less code in context (complete delegated orchestration)

1 + Groovy extension
(e.g. annotation)
Or a DCI-capable Java
framework

1:
No
framework

What you need to add:
- Groovy for mixins (may stick to 100% Java syntax or adopt Groovy as a language)

What you get:
- Rich, domain driven programming/design model
- Stable domain model
- Well-managed dependencies

Restrictions:
- No role access to context from role methods

Plain Java with
Groovy mixins

As-is

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/4cf0f24c0804081656l5aed67b5hf34fc73cbea37
5b0@mail.gmail.com
-Advanced meta programming: http://www.slideshare.net/zenMonkey/metaprogramming-techniques-in-groovy-and-grails

CALLISTA
Enterprise