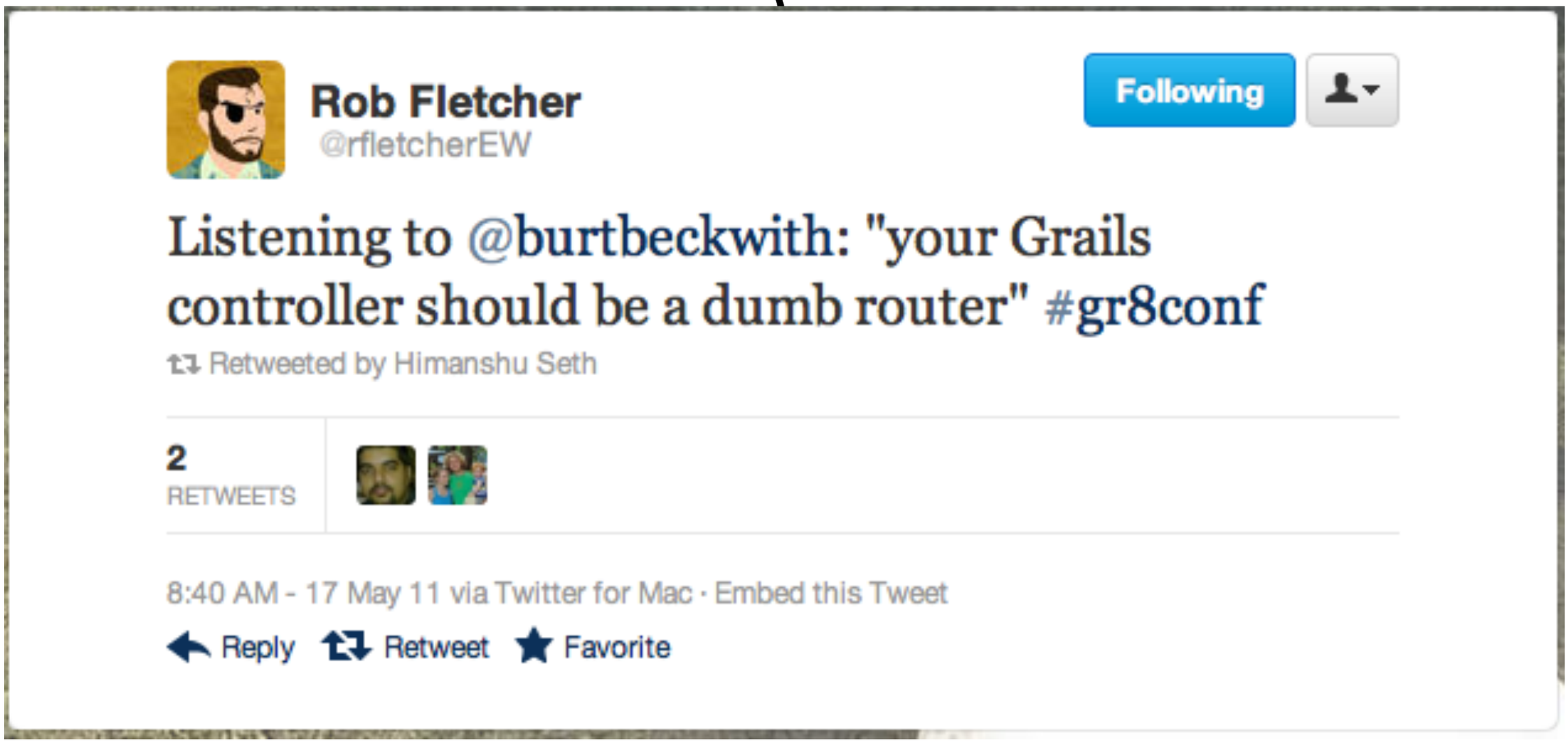
Grails Services

Services

- Architectural Layer
  - Not UI Specific
- Related set of functionality
- Transactional
- Operate on multiple domain classes
- Service Oriented Architecture



https://twitter.com/#!/rfletcherEW/status/70483875662532608

Rationale for Services

- Within a system: Complex logic
  - Logic that spans multiple data stores
  - Logic that coordinates many domain classes
  - Logic that may be useful for multiple controllers
  - Logic not specific to a UI function
- Between systems: Sharing functionality & data
  - Allowing distributed access (including inter-company)
  - Introduces concept of data "master": which system/service owns a class of data

Service Examples within a System

- Customer Service
- Address Service
- Tax Service
- Reconcile Service
- Order Service
- Licensing Service
- Inventory Service
- Price Service

Enterprise Services

- Shared between systems
- Often defined to enable central ownership of shared data
  - Customer Master
  - Product Master
- Often provided by vendor platforms
  - Order management
  - Sales tax calculation
  - Inventory management
  - ERP
- Can start as an internal service for an application and then transform into a shared service when other applications need access

Design Goals for Services

- Loose coupling
- Coarse grained
  - minimize network chattiness
  - complete common use cases in one call
- Stateless
- Language and Platform independent
  - C# application should be able to call a Grails service

Grails Services

- grails-app/services
- Naming convention (ends in “Service”)
- Creating a new service:
  - grails create-service ServiceName
  - Example:

```
grails> create-service com.foo.Tax
| Created file grails-app/services/com/foo/TaxService.groovy
| Created file test/unit/com/foo/TaxServiceTests.groovy
grails>
```

Example Service

```
package com.foo

class TaxService {

    def calculateSalesTax(State state,
      String zipCode) {
       //...
    }

}
```

Service Methods

- Can have dynamic or static return types
  - Dynamic: def calculatePayment(Order o)
  - Static: Payment calculatePayment(Order o)
- Have access to domain classes and persistence via GORM
- Can have transactions managed for them
  - transactional member of service set to true
  - Automatic commit on successful complete
  - Automatic rollback on exception

Transactions

- Transactions are all-or-nothing operations involving data and have the properties of ACID
- Atomicity
  - All operations must complete or none will
- Consistency
  - State must be valid before and after operation
- Isolation
  - Operations cannot be effected by others during the execution
- Durability
  - Changes stick around after the transaction is complete

Declarative Transactions

- All services are transactional by default
- All methods are wrapped in a transaction
- Rollback occurs if a RuntimeException is thrown

Custom Transactions

- Use the @Transactional annotation to set transactions only on specific methods (instead of the entire class)

```
import org.springframework.transaction.annotation.Transactional
class BookService {

    @Transactional(readOnly = true)
    def listBooks() {
        Book.list()
    }

    def deleteBook() {
        // ...
    }
}
```

Dependency Injection

- Decouple components of your application
- Swap out implementations as needed
- Makes your code easier to test
- Provided in Grails by the Spring Framework
- All services are automatically injected at runtime via naming convention

```
// bookService injected into controller
class BookController {
    def bookService
}
// bookService injected into another service
class AuthorService {
    def bookService
}
// bookService injected into domain class
class Book {
    def bookService
    def buyBook() {
        bookService.buyBook(this)
    }
}
```

Service Scope

- Default scope for services is singleton - one instance for the entire application
- Can also do
  - prototype
  - request
  - flash
  - flow / conversation (for webflow)
  - session
  - singleton
- http://grails.org/doc/latest/guide/services.html#scopedServices