Reactive Systems

Dave Farley
http://www.davefarley.net
@davefarley77
Let us know what you think

Click ‘engage’ to rate sessions and ask questions

Join the conversation gotocph
Reactive Systems

21st Century Architecture for 21st Century Problems

Dave Farley
http://www.davefarley.net
@davefarley77

http://www.continuous-delivery.co.uk
Our World Is Changing

**Large Applications circa 2005:**
- 10’s of Servers
- Seconds of Response Time
- Hours of Offline Maintenance
- Gigabytes of Data

**Large Applications Now:**
- Handheld Devices to 1000’s of multi-core processors
- Millisecond Response Time
- 100% Uptime
- Petabytes of Data
Our World Is Changing
The Reactive Manifesto

“21st Century Problems are not best solved with 20th Century Software Architectures”

The Evolution of modern hardware has changed many of the common assumptions of software development

Source: www.reactivemanifesto.org
Reactive Systems Are:

*Responsive:*
  - Responds in a Timely Manner
  - Cornerstone of Usability
  - Also Quick to Detect Problems

"Since we can’t afford faster computers, we’re going to hire slower workers because that will make our computers seem faster."
Reactive Systems Are:

**Resilient:**
- Remains Responsive in the Face of Failure
- Resilience Depends on - Replication, Containment, Isolation and Delegation
Reactive Systems Are:

**Elastic:**

- Remains Responsive Under Varying Workload
- Responds to Change in the Input Rate By Increasing or Decreasing Resources that Service the Input
- Decentralised Architecture, No Contention Points, No Central Bottlenecks
Reactive Systems Are:

**Message Driven:**
- Asynchronous Message Passing is the foundation for all of these properties
- Loose-Coupling, Isolation, Location Transparency
- Ability to Delegate Errors
Properties of Reactive Systems

- Flexible
- Loosely-Coupled
- Scalable
- Easier to Develop
- More Tolerant of Failure
- Respond to Failure Gracefully
- Responsive to Users
Fractal Architecture

• Large Systems Are Composed of Smaller Ones
• They Depend on the Reactive Properties of Their Constituents
• These Benefits Operate At All Scales
• Such Systems are Composable
Failure Modes in Synchronous Messaging

Component 'A'

Component 'B'

[Diagram showing failure modes between Component 'A' and Component 'B']
Synch Messaging Breeds Complexity

Synchronous Comms Increases Coupling in Location and Time
Synch Messaging Breeds Complexity
The Benefits of Asynchrony

Single Threaded! → Single Threaded!
An Example

Order("Continuous Delivery")

Cd STOP #A'

Reserve("Continuous Delivery")

Ordered("Continuous Delivery")

Complementry 'B'
An Example

- Order("Better Aerobatics")
- Ordered("Better Aerobatics")
- Reserve("Better Aerobatics")
- Ordered("Better Aerobatics")

Inventory
An Example of Idempotence

Component ‘A’

Component ‘B’

Expected(3)
Isolation

• Decoupling in Time and Space
  • Time - Sender and Receiver have independent lifecycle
  • Space - Location Transparency

• Share Nothing!

• Built on Inter-Component Communication over Well Defined Protocols
Isolation

Component ‘A’

Component ‘B’
Share Nothing

Component ‘A’

Component ‘B’
Back-Pressure

• You Can’t Isolate Stress
• The System as a Whole Needs to Respond Sensibly
• Unacceptable For a Stressed Component to Fail Catastrophically or Loose Messages
• Queues Represent An Unstable State - Load
• Components Under Stress Need to Reflect This By Applying Back-Pressure, Slowing Upstream Inputs
Queues Represent an Unstable State

Queues are always full or always empty. Anything else is transitional, on its way to full or empty.
Queues Represent an Unstable State

Queues are always full or always empty. Anything else is transitional, on its way to full or empty.

Component ‘A’  
Slightly Faster

Component ‘B’  
Slightly Slower

Always Full!
Back-Pressure

Component ‘n’

Back-Pressure!

Slightly Faster

Always Full!

Component ‘B’

Slightly Slower
Eventual Consistency

Component ‘A’

Component ‘B’
Location Transparency

- Elastic Systems Need To React To Changes In Demand
- We Are All Doing Distributed Computing
- Embracing This Means There Is No Difference Between Horizontal (Cluster) and Vertical (Multicore) Scalability
- Components Should Be Mobile
- One Pattern For Communications
  - Local Communications Is An Optimisation
Linear Scalability Through Sharding
Linear Scalability Through Sharding

Component ‘A’

Component ‘B1’

Component ‘B2’
Modern Hardware Should Change Our Assumptions

• For Efficient Software, The Biggest Cost is Shifting Data
• RAM is Not Random Access
• Disk is Not Random Access
• SSD is Not Random Access
• RAM is Slow, Network is Slow, Disk is Slower
Conway’s Law

Siloed Teams
- UI Specialists
- Middleware Specialists
- DB Specialists

Rigid Architecture
- DB

Specialists
Conway’s Law

Cross-Functional Teams
Organised by Business Function

Distributed Service Architecture
Bounded Contexts

• Each Component Is Autonomous and Isolated
• It Only Communicates Through Well Defined Protocols
• Works Best When Components Are Aligned With Bounded Contexts
• Bounded Contexts Are A Concept From Domain Drive Design (DDD) - Eric Evans

Bounded Context:
• The Context Within Which A Model Of the Problem Domain Make Sense
• There Should Be ‘Translations’ Between Bounded Contexts - No “One Model To Rule Them All”
• Bounded Contexts Tend To Align With Business Functions
• Best Way To Decompose Organisations And Systems
Example Reactive, MicroService architecture

**Gateway Services**
- Market Data Consumers
- Public API
- Market Makers

**Core Services**
- Execution Management Services
- Execution Venue Services

**General Services**
- Account Services
- Trade Report Services
- Customer Service

**Core Services**
- Clearing Gateways
- TFX Application
- Notification Service
- FIX Gateways

**Gateway Services**
- Public Message Bus
- Control Message Bus
- Gateway Services

**Core Services**
- Market Management Application
- Notification Service
- Customer Service Application
Where to start?
Please

Remember to rate this session

Thank you!