omegapoint™

A Small Step towards Event Driven Architectures

Dan Bergh Johnsson
Senior Consultant and Partner
Omegapoint AB; Sweden

Rådgivare & experter - Systemarkitektur & IT-säkerhet

Overall Presentation Goal

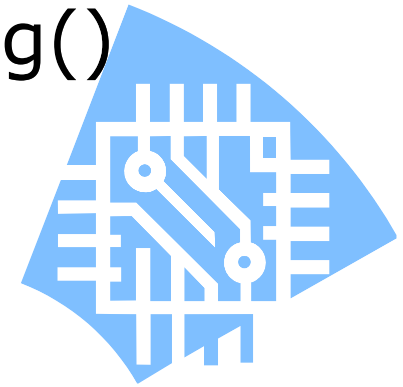
Show the mismatch of traditional call-stack architectures vs modern multicore machines

and

point in a feasible direction

Call Stack Code

```
public void clearOrderForShipping() {
   cardservice.reserveFundsOnCard();
   inventory.allocateStuff();
   custserv.notifyOrderPreparesForShip();
   storehouse.prepareForShipping()
}
```



- One-processor job!

Call Stack

Advantages

- Easy error handling
  - Cut on first fail
- “Tail” of failing job never executed
- Order seems important

Drawbacks

- Execution locked into one processor
  - Amdahl’s Law
- Limited by “headroom” inside processor
  - Moore’s “Law”

Modern computers

- Heat/power limiting factor
- New measures
  - MIPS / Watt
  - MIPS / m^3
- Multi-core

Next Generation Computers

- More processors
- Each processor not more powerful
- “Headroom inside” does not rise

The Problem to Come

Facts

- More complex computations
- Not faster processors

Effect

- Better capacity
- Same, or slower, response time

Insight

- We need to change

One Possible New Guiding Star - EDA

Event Driven Architecture

- Events trigger processing
- Processing generates events
- Watch the state whenever you want

“Order Cleared” Example

- OrderCleared -> CardPayment, Inventory …
- LowOnStock -> Replenishment

Big Mind-Shift

- Who will Make the Leap First?
  - Projects minimize risks
- Who will be Left Behind?
  - Existing systems will not be changed
- Need Low Threshold Approach

Small Step - From Verbs to Nouns

```
public void clearOrderForShipping() {
  new CardFundReservationTask().execute();
  new InventoryStuffAllocationTask().execute();
  new PreparedForShippingNotificationTask().execute();
  new StorehouseShippingPreparationTask().execute();
}
```

- reserveFundsOnCard => CardFundReservation

Small Change, Big Difference

- Separation of responsibilities
  - Defining a task
    - new CardFundReservationTask()
  - Executing a task
    - .execute();
- Opened up for parallelism

Another Step - Shifting Responsibilities

```
public void clearOrderForShipping() {
  cardserviceDest.send(
    new CardFundReservationTask());
  inventoryDest.send (
    new InventoryStuffAllocationTask());
  customerserviceDest.send (
    new PreparedForShippingNotificationTask());
  storehouseDest.send(
    new StorehouseShippingPreparationTask());
  waitUntilSynch();
}
```

- Command/Request => Inquiry/Needing Help
- Asynchronous and Parallel Computations

Order is not Always Important

Seem important

```
cardservice.
    reserveFundsOnCard();
inventory.allocateStuff();
```

Also seem important

```
inventory.allocateStuff();
cardservice.
    reserveFundsOnCard();
```

Inquiry Driven Architectures

Advantages

- Parallelism
- Faster response time

Drawback

- Execute everything even if not necessary
- Compensating action on failure
- Cumbersome error handling
- Inconsistency needs to be modelled

My Boss’s Slide

- Yes, we do consulting
  - sales@omegapoint.se
- Yes, we are hiring
  - jobs@omegapoint.se

How does this lead to EDA

Four modes of naming a channel and associated semantics [Hohpe CSS2007]

- Receiver – creditServiceDest
  - Command based
- Operation – reserveFundsDest
  - Need based
- Document – creditInfoDest
- Event – orderClearedDest

Summary

- Call stack architectures are not sustainable
- We need to change
  - and need to be able to in small steps
- Look for parallelism – sub system calls
- Question specified sequences
- Shift responsibility to callee-side
- Good luck

Concluding statement

Call-stack architectures are not sustainable. Possible to change if some assumptions are challenged. We can make a small step (on Monday).

omegapoint™

</A Small Step towards Event Driven Architectures>

Dan Bergh Johnsson
dan.bergh.johnsson@omegapoint.se
dearjunior.blogspot.com