Developers guide to server-side productivity

A flight through the landscape of feature sets, technologies and Open Source implementations

Motivation

- Increase productivity and quality of software
- Empower developers to have more fun
- All this achieved by making better decisions

About us

- Thor Henning Hetland
- Tobias K Torrissen

Cantara

Agenda

- What challenges are we facing
- A brief look at potential aid
- Case study
- Conclusion
- Q&A

What challenges are we facing

There can only be one!

- It is a myth!! - Example: J2EE
  - 1997 - superhot! All vendors(-Microsoft), all hotshots super stoked!
  - 1998 - Pet store. Statefull FUD. Year of EJB benchmarking.
  - 2000 - Hotshots turn against EJB. No silver bullet (again)
  - 2002 - Year of Spring in early adopters markets (Norway)
  - 2005 - Hotshots turn agains Spring and Java web development. Look at RAILS!!
  - 2009 - Hotshots turn agains Rails. Look at Scala and Lift.

There can only be one!

- There are plenty of examples:
  - RDBMS
  - Web applications
  - .Net
  - Windows
  - SOA and Web Services
  - HTTP RPC (REST)
  - [...]

There can only be one!

- Conclusion:
  - There is no silver bullet.
  - Use the brain, Developer!
  - Fun to follow the hype - but it does not create value
  - Different problems require different solutions (like in the rest of the world)

Decisions, decisions, decisions

Soa EJB ORM JPA

JDO JINI

Decisions, decisions, decisions

- How to make BAD decisions in this chaotic landscape:
  - Psycological aspects
  - Cultural aspects
  - Corporate policies
- Is there a better way?

Psychological aspects

- Fear of making wrong decisions:
  - Sartre: "Med valg følger angst"
  - "Nobody evers been fired for choosing IBM"
  - "You can´t go wrong with beige"

Company policies and decisions

- Thou shal only use Java/J2EE/.NET/LAMP[...]!
  - Reduces the value created from technology (50-90%)
    - Increases the startup and training costs (30-200%)
- You create a VERY big hammer that must be used for ALL tasks: hitting nails and changing lightbowls.

Company policies and decisions

- Special considerations regarding training costs.
  - Company policies increase them!
  - Using inadequate tools create solutions that are hard to understand. Training debt!
  - When problems seems hard to solve
    - Stop and think: Am I using the right tools for the job?

Company policies and decisions

- Special considerations regarding
    - Will you let your

Culture and religion

- Polarization
  - Lightweight vs Suites
  - Linux vs Microsoft
  - Remember: There can be only one!
- A religion war!
  - Easier to get attention when crying out loud.
  - Most effective way to create a revolution.

Decisions, decisions, decisions

- Context : f.eks Artikkelbase til publisering.
- Teknologi : f.eks CMS
- Implementasjon f.eks OpenCMS

How do you make decisions

- Read the white pages? (CMS matrix)
- Try out demos (CMS eksempelet?)
- Create prototypes?
- Expericence (own or others)?
- Implementere to løsninger?
- (vurdere en slide seinere som tar for

A brief look at potential aid

A counter measure to faith, culture and religion.

- We need some kind of help in order to resist the temptation of religion!

Categorization and exemplifying solutions.

- To make good selections, we need to categorize the problems and contexts
-

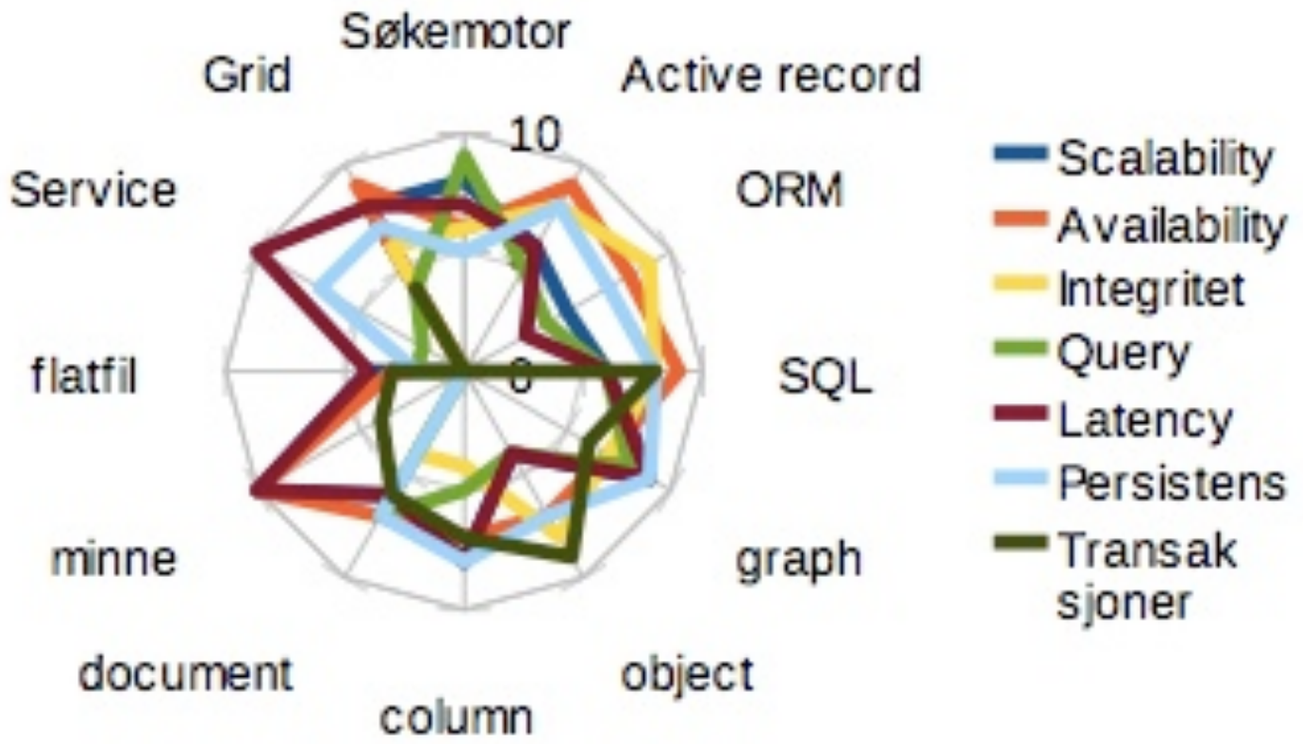
State

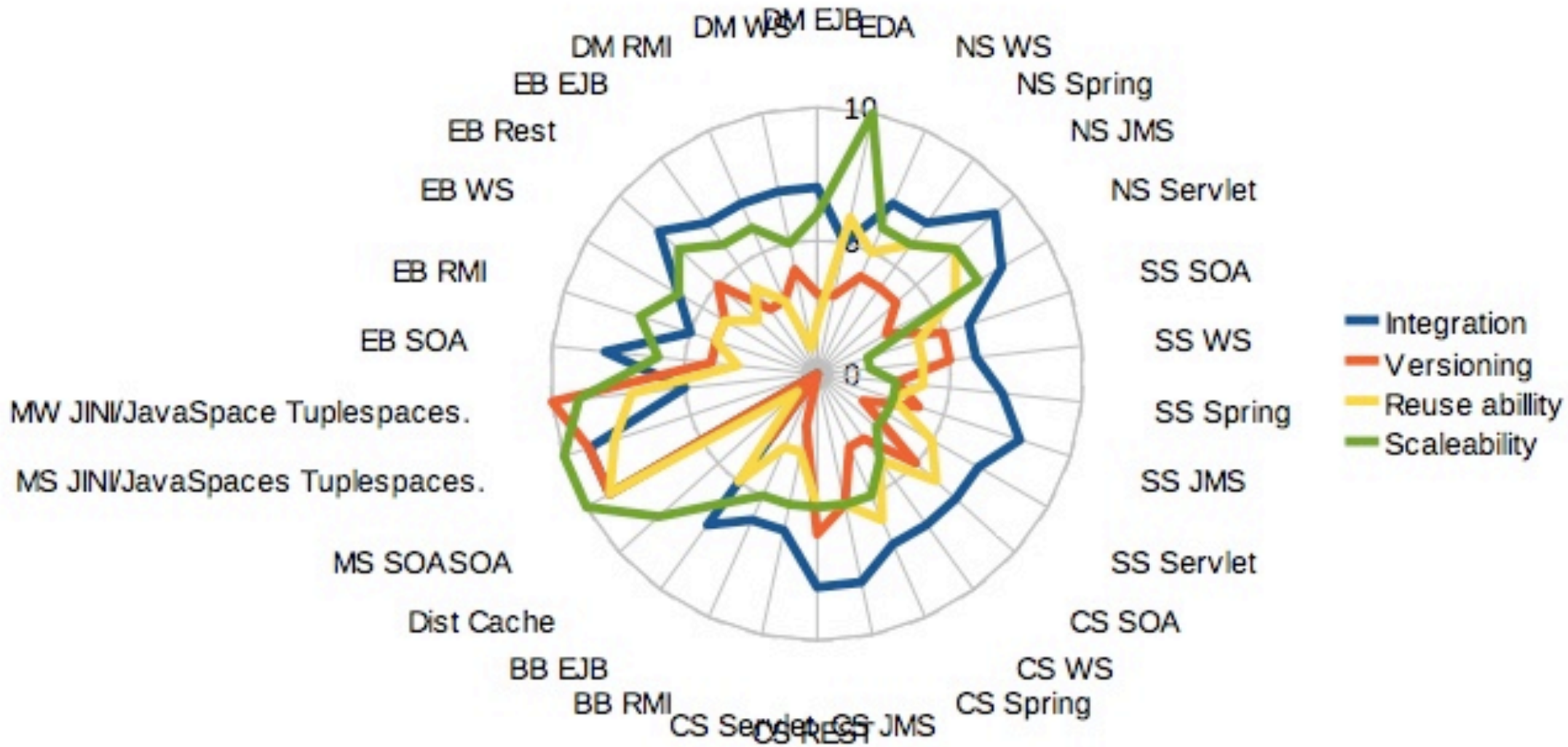
- Lets look at which strategies we have regarding handling state in server-side systems

Datastyles

- And what are our options redarding datamodels?

Landscapes


Søkemotor
Active record
ORM
SQL
graph
object
column
document
minne
flatfil
Service
Grid
10
8
Scalability
Availability
Integritet
Query
Latency
Persistens
Transak
sjoner


DM WS
DM EJB
EDA
DM RMI
NS WS
EB EJB
NS Spring
EB Rest
NS JMS
EB WS
NS Servlet
EB RMI
SS SOA
EB SOA
SS WS
MW JINI/JavaSpace Tuplespaces.
SS Spring
MS JINI/JavaSpaces Tuplespaces.
SS JMS
MS SOASOA
SS Servlet
Dist Cache
CS SOA
BB EJB
CS WS
BB RMI
CS Spring
CS Servlet
CS JMS
CS REST
10
0
Integration
Versioning
Reuse abillity
Scaleability

Case 1

- Web-based customer system for department in large enterprise (10k+ emp.)
- Existing database
- No suitable API´s to integrate with.
- Yet another database mirror through a web gui.

Which translates to the following features:

- Database layer
  - No though requirements in Scalability, Availability, Integrity, Query functionality, Latency, Persistence and transactions.
- Service layer
  - No though requirements in Integration, versioning, reusability, scalability.
- We are free to choose **fun** and **productivity**!

Decisions, decisions, descisions

- Context:
  - Driftsmiljø er i Java.
  - All developers know Java.
  - Corporate policy sier Java, Oracle og JPA.

- Valget er vel lett?

JAVA?

- Valgt et verdet?

nei og nei og nei.

- Til formålet finnes betydelig bedre verktøy.
- OG
- Husk corporate policies og opplæringsgjelden.

Andre alternativ

- Groovy and Grails
  - Fokus på høy produktivitet.
  - Gjenbruk av eksisterende Java-rammeverk.
  - ORM støtte gjennom GORM
  - Integerer sømløst med java.
  - Integerer sømløst med java
  - God XML støtte (grails)

- Ruby og Rails
  - Svært enkel ORM gjennom Active Record.
  - Integerer med java gjennom jRuby

- Scala og Lift
  - Svært god XML støtte
  - Støtte for active record.
  - Productivity i fokus.
  - Sterk typing.

Valget blir tatt:

- Ruby og Rails.
  - Har ikke noe fornuftig API å integrere mot. ActiveRecord gjør susen.
  - Enkelt og greit å få opp en liten formålstjenelig web app
  - Kan deployes på eksisterende infrastruktur.

Så hva skjer...

- Systemet ble en suksess.
  - Gav stor verdi på kort tid.
  - Lav investering.
  - Glade utviklere og brukere.

Flere vil ha

- Andre avdelinger vil ha sine kunder inn i systemet.
- Landskapet endres:
  - Flere databaser.
  - Forskjellige kundestrukturer
  - Standardsystemer med overnormalisert database.

Konsekvenser

- Ikke master for egne data:
  - Ingen oppdatering av kunder.
    - Delete skjer ikke.
- Endringshyppigheten går i taket.
- Forskjellig krav til oppetid
- Behov for konsolidering av data.

Så hva gjør vi?

- Frontend: Ikke stort.
- Backend: Mye.

Beholde Active record?

- "An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data.“
- Greit som sugrør ned i datakilden... men løser ikke noe mer en det.

Men hva med standardsystemer?

- Generiske strukturer.
- Key value pairs.
- Glem ORM.
- Og ikke gjør databaseintegrasjon.
- Se etter API-er...

Løsningen

- Bruk adekvate verktøy!
  - ORM der det passer.
  - API-er der det passer.
  - Skriv mappinglogikken selv.

Ikke generaliser før du vet hvorfor du skal generalisere

Conclusion