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

Chalenges

- Myths
- Options
- Processes

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 be only 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

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

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"

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.

Company policies

- 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

- 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

- Special considerations regarding operations ..
    - Will you let your

Other strategies

- Read the white pages? (CMS matrix)
- Try out demos (CMS eksempelet?)
- Create prototypes?
- Expericence (own or others)?
- Implement more than one solution?
- Go with the flow?

Sigh ....This is hard stuff...

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
-

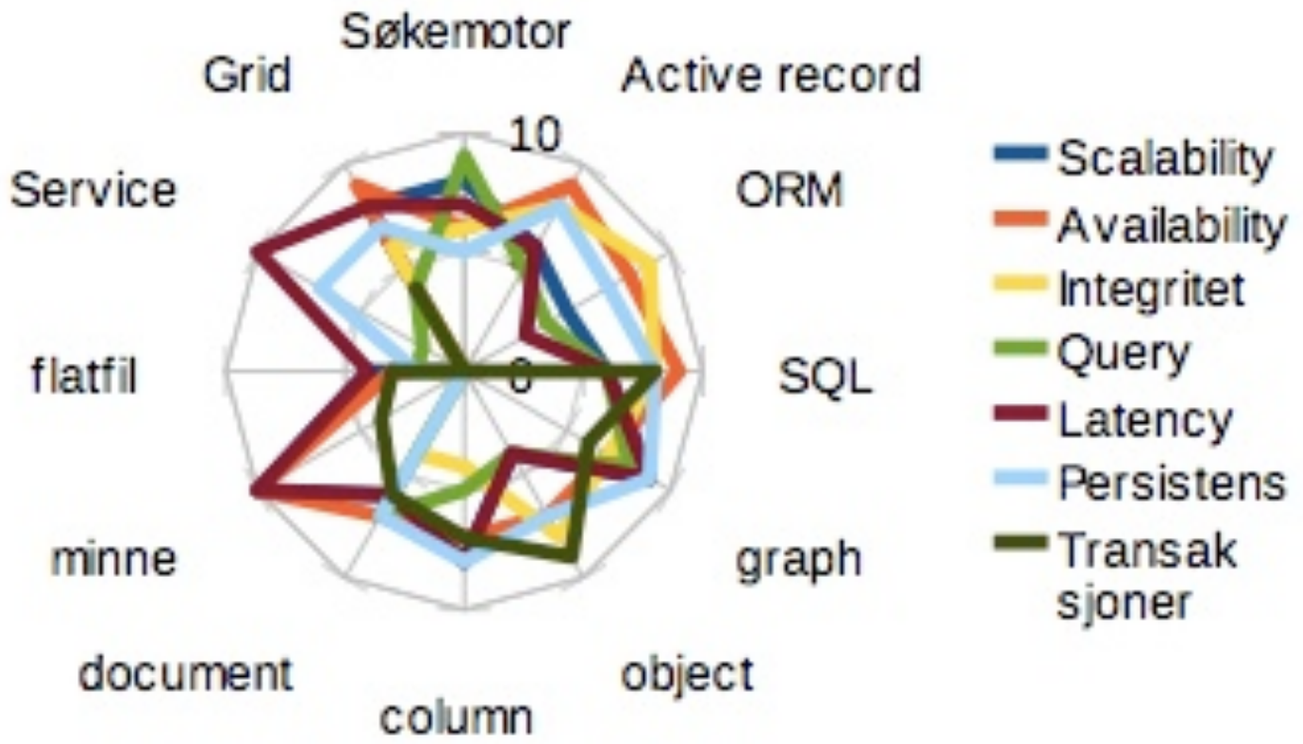
State

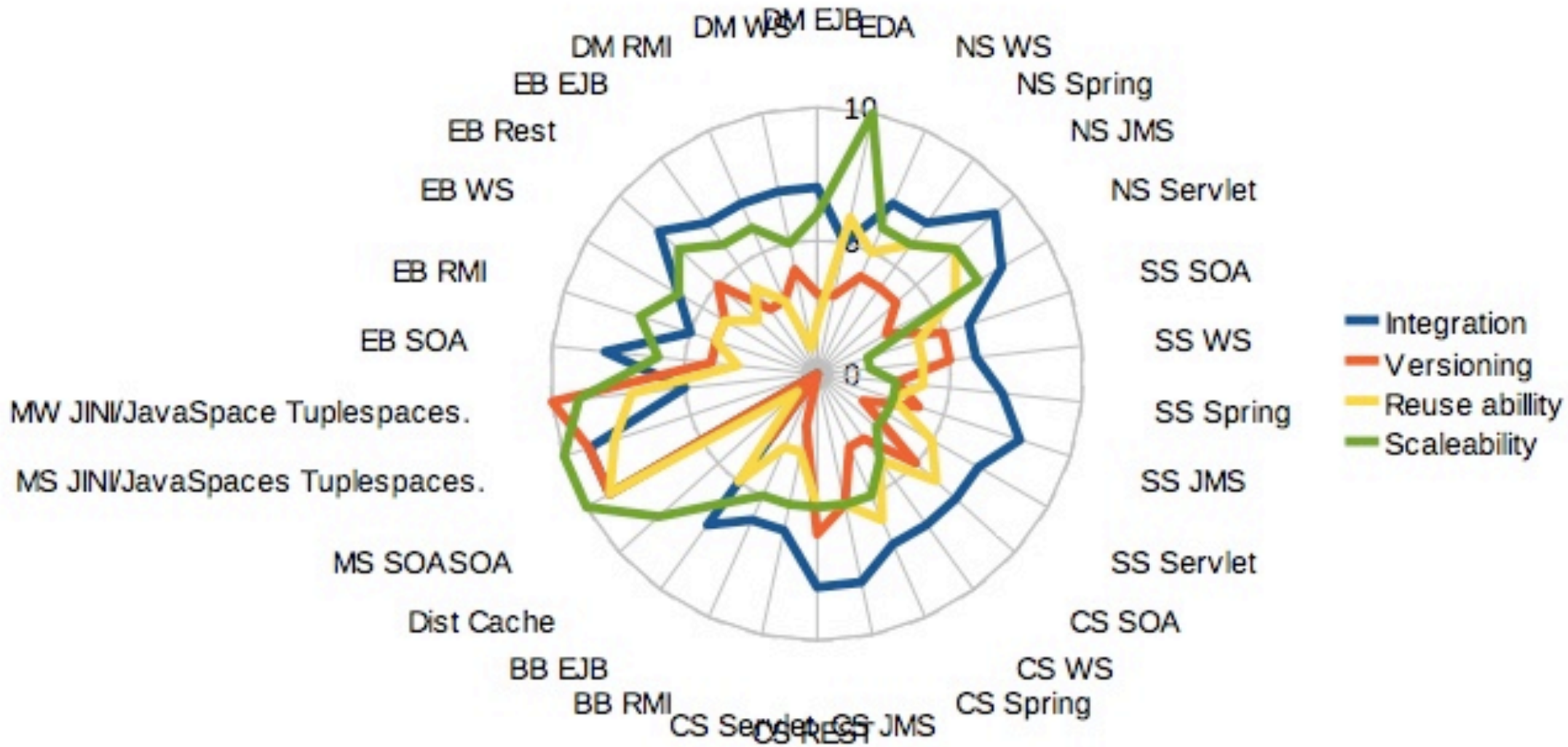
- Lets look at which strategies we have regarding handling state in server-side systems

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 driven web application.

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:
  - Java server environment.
  - All developers know Java.
  - Corporate policy: Java, Oracle og JPA.

Let´s make it fun and prductive with Java and open source!

- Standard 3 layers webapp:
  - Frontend: JSF or Struts2
  - Domain logic: Spring framework
  - Persistence: Hibernate ORM to DB
- Nobody got fired from choosing IBM.

No, no, no!

- Stop and think!
- There are much more suitable alternatives
  - much more fun
  - much more productive
  - and still open source and Java

Let´s look at 4 key Java open source alternatives

- Groovy on Grails
- jRuby on Rails
- Scala and Lift
- 4GL / model driven

jRuby on Rails

- Focus on RAD and productivity.
- Simple and productive ORM with Active Record.
- Integrates with Java.
- Built in mvc and scaffolding features.

- Productivity factor: Ten times as productive.
- Fun factor: Five times more fun.

Groovy on grails

- Focus on RAD and productivity
- Easy reuse of existing Java libraries
- ORM through GORM .

- Productivity factor: Six times as productive.
- Fun factor: Five times more fun..

Scala and Lift

- Focus on RAD and productivity.
- Supports active record.
- Good Java integration.
- MVC support

- Productivity factor: Two times as productive.
- Fun factor: Ten times more fun. On top of developers hype curve.

4GL / model driven

- Some kind of generation based on the datamodel.
- >10 open source products available.

- Productivity factor: five to twenty times as productive.
- Fun factor: Developers hate them. below zero.

Time for a decision:

- The winner is....
- jRuby on Rails.
- The best combination for fun and productivity (for this case)
- But all options are suitable alternatives.

What happens next...?

- Great sucess!!
  - 80% of the department is using the app on weekly basis.
  - Rumors of the application spreads like wild fire!
  - Developers and stakeholders are treated as heros.

... now everyone wants the application!

- The landscape changes:
  - A large number of databases and CRM systems.
  - A myriad of customer structures.
  - Off the shelf software
  - 24/7 requirements

Consequences

- Data mastering becomes a huge issue.
  - Not possible to do updates and/or deletes.
- Need some kind of data consolidation.
- Rate of change in requirements skyrocket!
- 24/7 requirements
- Our simple integration strategy is now void.

Which translates to the following features:

- Database layer
  - Though requirements in Scalability, Availability, Integrity, Latency.
- Service layer
  - Though requirements in Integration and Versioning.
- We are no longer free pick and choose freely.

Our context has changed dramaticly!

“Hate to tell, I told you so!” - The hives

- “Our Enterprise Corperate policy would have prevented this!”.
  - Wrong: No application, no problem!
  - Wrong: Poor application, same problem!

“Hate to tell, I told you so!” - The hives

- “We need to re-implement the solution on a platform that supports this new complex scenario!”
    - Wrong: There is nothing wrong with the application!
    - Wrong: The new requirements can be handled on the existing platform.
    - Wrong: The enterprise platform does not solve any problems.

.... and if we need to throw it away, it´s not a huge sunk

Stop and think (again)

- Enterprice company policy did/will not save us.
- Let´s focus on the problems at hand:
- Versoning, Scalability, Latency, Integrations, Availabilty

  What kind of technology can save us?

  None! This is a design challenge!

Landscape revisited.

- We can use the landscape to rate integration strategies for each individual system.
- Keep focus on the problems at hand.

What do we do?

- Frontend: No significant changes.
- Backend:
  - Add some new integration strategies to support the various systems.
  - Correlation of data from different sources
  - Active caching of all external data.

Keep 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.“

- Reduce the responsibility of active records from “domain objects” to an integration strategy for a subset of data sources.
- Add suitable integration strategies for the other datasources.

Architecture v2

- Reduce the responsibility of active records from “domain objects” to an integration strategy for a subset of data sources.
- Introducing domain repository
- Introducing mapers and correlators
- Adding new integration strategies
- re-wire frontend to use repository.

Conclusion