Simplicidade, Escalabilidade, Produtividade e Testabilidade com J2EE, AOP e Rich Clients

Summa Technologies

www.summa-tech.com

Michael Nascimento Santos

- JSR Community Manager @ java.net
- Expert na JSR 207 (PD4J)
- Thinlet commiter
- Membro da organização do SouJava
- SCPJ 1.2 & 1.4, SCWCD, SCMAD
- 5 anos de experiência com Java
- Palestrante no JavaOne 2003, Abaporu, JustJava e Javali

SUMMA

Allan Jones B. de Castro

- Formando em Engenharia de Computação pela Politécnica da USP
- 3 anos de experiência com Java
- Larga experiência em sistemas corporativos, desde smart cards a desktops
- Desenvolvendo academicamente sistema de conversão texto-voz com tecnologia adaptativa

O que há de errado com a maioria das aplicações hoje?

Aplicações web

- HTML é limitado demais
- JavaScript e CSS são problemáticos
- Portabilidade é muito baixa
- Problemas com o modelo request/response
- Recursos de interação limitados
- Pode ficar bastante pesada

Aplicações web (Cont.)

- XML/XSL são atrativos no começo
  - XSL difícil demais para designer
  - Desperdiça todo o suporte JSP do container
  - Exige cuidados com a performance, como cache etc.
  - Aumenta o número de habilidades requeridas dos participantes (especialmente com o uso de Schemas)

Aplicações cliente-servidor

- Swing
  - É complexo
  - Código performático exige conhecimento do framework
  - Exige alto conhecimento de OO do desenvolvedor

Aplicações cliente-servidor

- SWT
  - Grande variação de performance entre as plataformas
  - Exige grande controle por parte do desenvolvedor
  - Não é **de verdade** mais leve

Código servidor

- EJB
  - Complexo
  - Pode adicionar overhead desnecessário
  - Ciclo de desenvolvimento maior

Código servidor (Cont)

- Beans + Servlets
  - Geralmente possuem problemas de escalabilidade
  - Não usam naturalmente serviços do servidor como transacionalidade, pooling, etc.

Principais dificuldades

- Código complexo
- Dificuldade de testes
- Lentidão/ineficiência da interface gráfica
- Ciclo de desenvolvimento lento
- Escalabilidade (!!!)

O que podemos fazer melhor?

Lado cliente

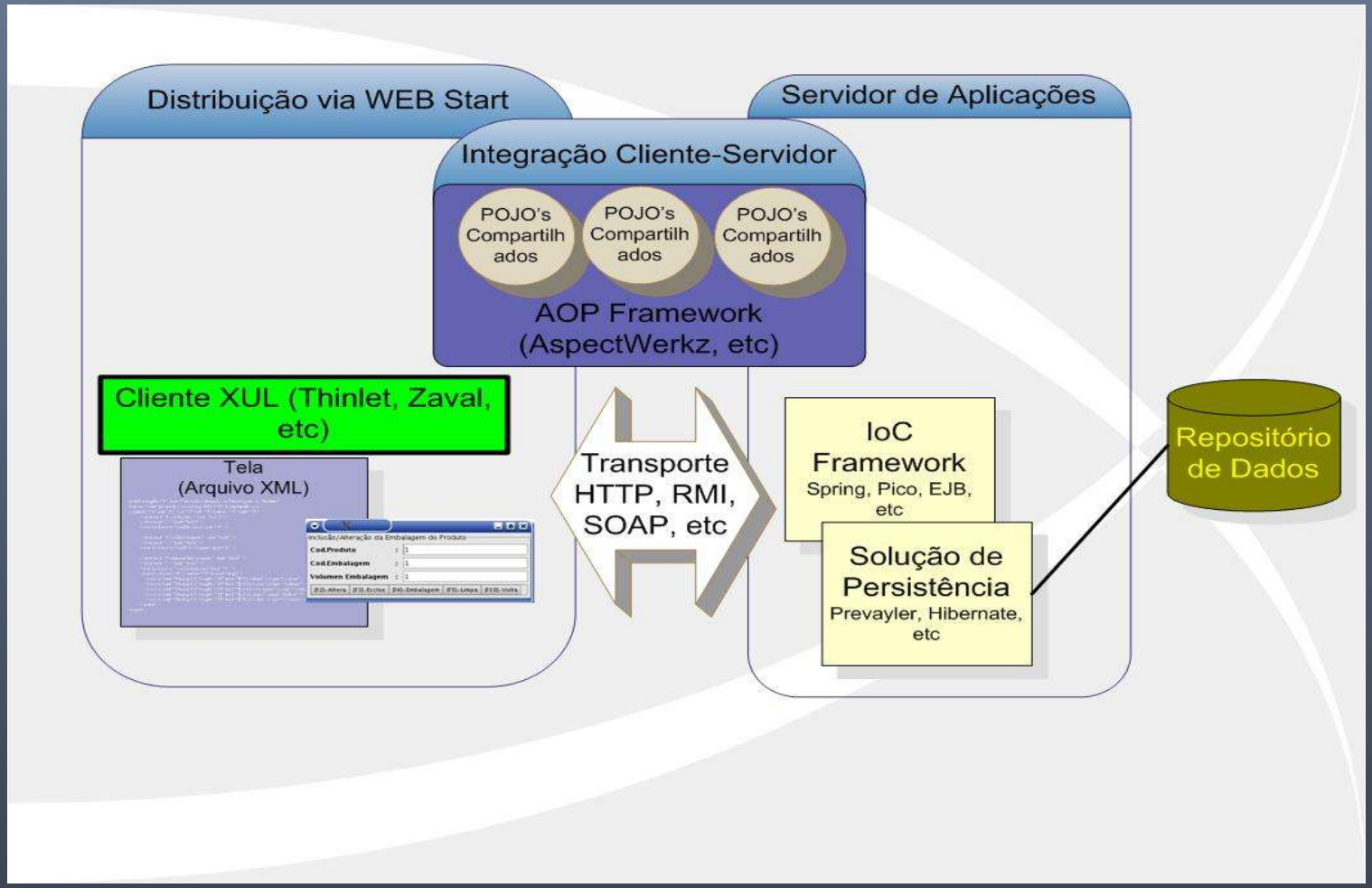
- Interface rica
- Aplicação leve
- Modelo de programação simples e produtivo
- População automática da tela
- Validação dos campos
- Facilidade de distribuição/upgrade de novas versões

Lado servidor

- Persistência simplificada
- Modelo de classes de negócio simples, produtivo e minimamente intrusivo
- Testabilidade da aplicação
- Flexibilidade de uso de tecnologia de integração com servidor
- Facilidade de mudança da topologia

Como podemos fazer isto?

Modelo geral da solução



SUMMA

Interface cliente rica e leve

- Modelo de programação XUL, baseado em XML
- Modelo dinâmico poderoso
- Modelo de programação simples
- Diversos frameworks XUL proporcionam desenvolvimento simplificado

Interface cliente rica e leve

- Thinlet – http://www.thinlet.com
  - LGPL
  - Baseado somente no AWT e XML
  - Possui jar de somente 38kb
  - Desenvolvimento extremamente rápido e simples

Interface cliente rica e leve

- Thinlet – http://www.thinlet.com
  - Alta velocidade de resposta da interface
  - Permite desenvolver aplicações inteiras que consomem somente 4-8MB de pico de memória
  - Casos reais mostram que pessoas são capazes de programar após 10-30 minutos de treinamento

Interface cliente rica e leve

- Thinlet – http://www.thinlet.com
  - Possui plugin para JEdit, ThinletHelper (http://www.beanizer.org/index.php3?page=thelper)
  - Formato XML simples permite edição manual rápida e eficaz
  - Desenvolvedores capazes de codificar manualmente em cerca de 2 dias

Suporte ao modelo de programação

- Commons BeanUtils
  - http://jakarta.apache.org/commons/beanutils/
  - Permite mostrar automaticamente beans na tela
  - Permite popular tabelas, combo-boxes e outros controles
  - Permite capturar os valores exibidos em beans

Validação de dados

- Commons Validator
  - http://jakarta.apache.org/commons/validator/
  - Permite validar de forma simplificada os dados
  - Possui conjunto padrão de validadores
  - Permite adicionar outros tipos de validação, como CGC, CPF etc.

Distribuição e upgrade da aplicação cliente

- JNLP (Java WebStart)
  - Cliente open-source: NetX http://jnlp.sourceforge.net/netx/
  - Tecnologia padrão especificada pelo JCP
  - Permite distribuição automática da aplicação
  - Permite distribuir automaticamente a versão da VM que sua aplicação suporta, se o cliente não possuir

Distribuição e upgrade da aplicação cliente

- JNLP (Java WebStart)
  - Permite fazer upgrades de versão da aplicação - tanto completos, de um jar específico ou até gerar um jar diferencial de upgrade - de forma transparente
  - Garante um ambiente de execução tão seguro quanto o das applets

Persistência simplificada

- Duas abordagens principais:
  - Prevalência de objetos
  - Mapeamento Objeto/Relacional

Prevalência de objetos

- Prevayler (http://www.prevayler.org)
  - Não é banco de dados
  - Objetos “sobrevivem” aos restarts da aplicação
  - Não possui impedância O/R
  - Objetos somente precisam ser serializáveis
  - Permite saída em formato XML

Mapeamento Objeto/Relacional

- Hibernate (http://www.hibernate.org)
  - Classes persistentes precisam somente aderir ao padrão JavaBeans
  - Altamente flexível
  - API simples e intuitiva
  - Geração automática de SQL

Mapeamento Objeto/Relacional

- Hibernate (http://www.hibernate.org)
  - Permite uso de SQL hand-coded
  - Permite mapeamento de tipos definidos pelo usuário
  - Possui arquitetura plugável para caching, controle transacional etc.
  - Referência para EJB 3.0

Classes de negócio

- Modelo simplificado
- Suporte à transacionalidade
- Suporte à segurança
- Remotabilidade transparente
- IoC
- Testabilidade
- Flexibilidade de tecnologia de implementação

J2EE + AOP

- Modelo de componentes de negócio flexível, com grau de intrusividade variável:
  - Configuração via XML
  - Anotações (metadados)
  - Marker interface
  - Marker interface + prefixos de métodos

J2EE + AOP

- J2EE e frameworks IoC, como Spring, suportam:
  - Transacionalidade
  - Segurança
  - Remotabilidade

J2EE + AOP

- Uso de AOP proporciona:
  - Flexibilidade de tecnologia de implantação
  - Lazy IoC
  - Remotabilidade transparente
  - Testabilidade
  - Facilidade de migração de tecnologia

J2EE + AOP

- Remotabilidade transparente:
  - Componentes podem ser executados remotamente ou localmente mudando somente configuração do aspecto
  - Tecnologia de transporte – HTTP, WebServices, RMI – pode ser trocada facilmente
  - Permite experimentação com outras tecnologias de forma simplificada

Uma implementação possível

- AOP via AspectWerkz (http://aspectwerkz.codehaus.org)
- Derivação do EJB Command Pattern:
  - Métodos específicos estão sujeitos a execução no servidor via anotações
  - Classes de negócio podem ser usadas livremente na aplicação cliente, sem nenhuma chamada especial

Outras tecnologias de apoio

- Ant (http://ant.apache.org)
- Commons Logging (http://jakarta.apache.org/commons/logging)
- Commons JXPath (http://jakarta.apache.org/commons/jxpath)
- XDoclet (http://xdoclet.sourceforge.net)

Objetivos da demonstração

- UI simples de escrever, leve, rápida e poderosa
- Modelo de classes de negócio simples
- Modelo de persistência simples
- Integração entre cliente e servidor simplificada
- Remotabilidade transparente
- Testabilidade

Demo

Conseqüências

- Alta Produtividade
- Simplicidade
- Facilidade de manutenção
- Testabilidade
- Remotabilidade transparente
- Escalabilidade mais fácil de atingir

Conseqüências

- Facilidade de experimentação com novas arquiteturas e novas tecnologias
- Conhecimentos de Java básico são único requisito para a maioria do time!
- Expertise necessário para “colar” as tecnologias, tomar decisões finais de arquitetura e treinar equipe

Obrigado pela atenção. Perguntas?

allan.jones@summa-tech.com
michael@summa-tech.com
www.summa-tech.com

SUMMA