Seaside

Squeak Enterprise Aubergines Server

MAC5714 – Programação Orientada a Objetos

Aplicações Web

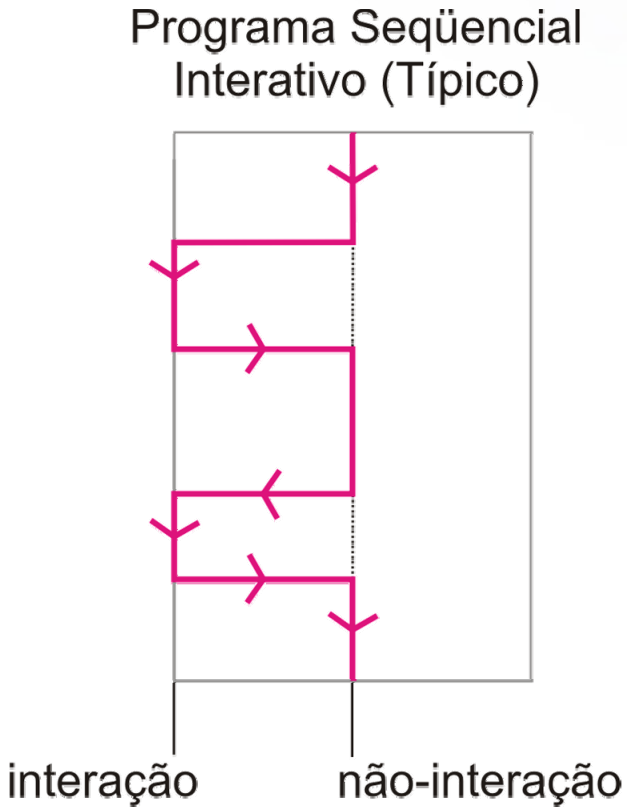
- Porque aplicações Web?
  - Aplicações Web são o futuro;
  - e o presente.
  - Navegador é tudo o que você precisa;
  - aplicações não-Web na Web;
  - sempre atuais;
  - portáteis.
- OO ajuda.
  - “[Continuation-base servers] may well prove to provide the foundation for all web application servers, in some not-too-distant future.” -- Bruce Tate, *Beyond Java*

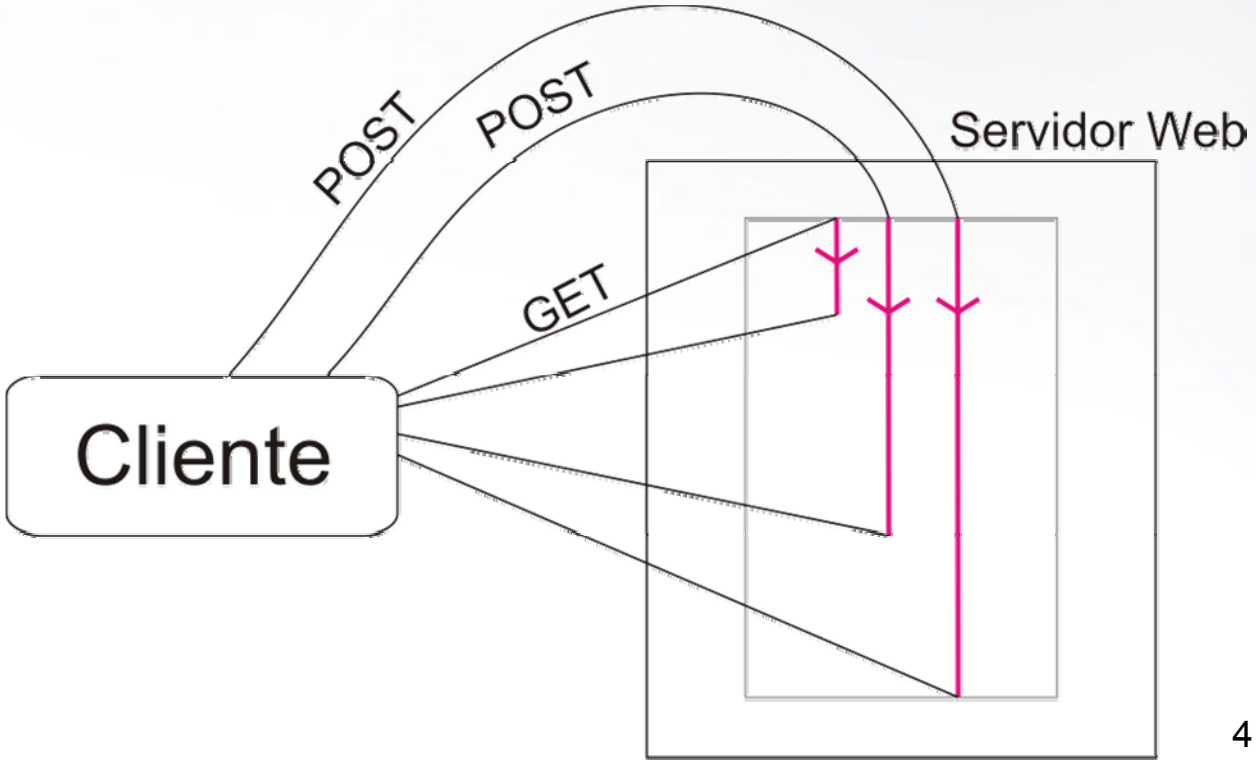
Aplicações Web

- Baseadas no HyperText Transport Protocol (HTTP)
- HTTP
  - Projetado para lidar com conteúdo estático
  - Não guarda estado
  - Relação assimétrica – servidor não atualiza cliente
- Problemas
  - Controle de fluxo
  - Sincronização de estado
  - Compartimentação

Controle de Fluxo

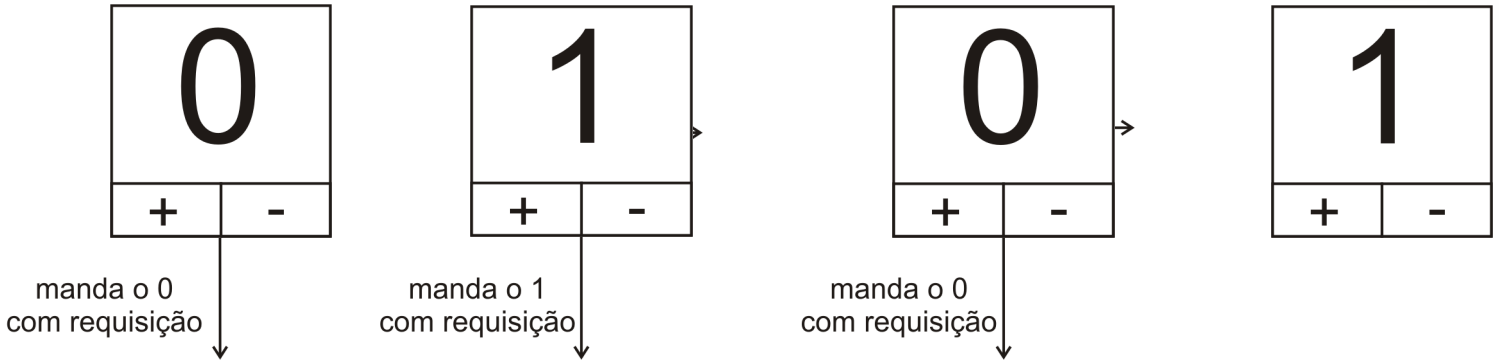
- HTML
  - Aplicações seqüenciais clássicas
  - Ciclos GET/POST





Controle de Fluxo (cont.)

- Fluxo fraturado
- Estado de continuação do servidor
  - Responsabilidade recai sobre o programador;
  - usuário pode voltar;
  - ir para frente;
  - clonar janelas.



Minha solução

- Problemas
  - Acoplamento entre domínio e apresentação;
  - difícil de reutilizar;
  - composição é quase impossível;
  - estado fica fora do servidor.

0		0		0	
+	-	+	-	+	-
0		0		0	
+	-	+	-	+	-

Sincronização de Estado

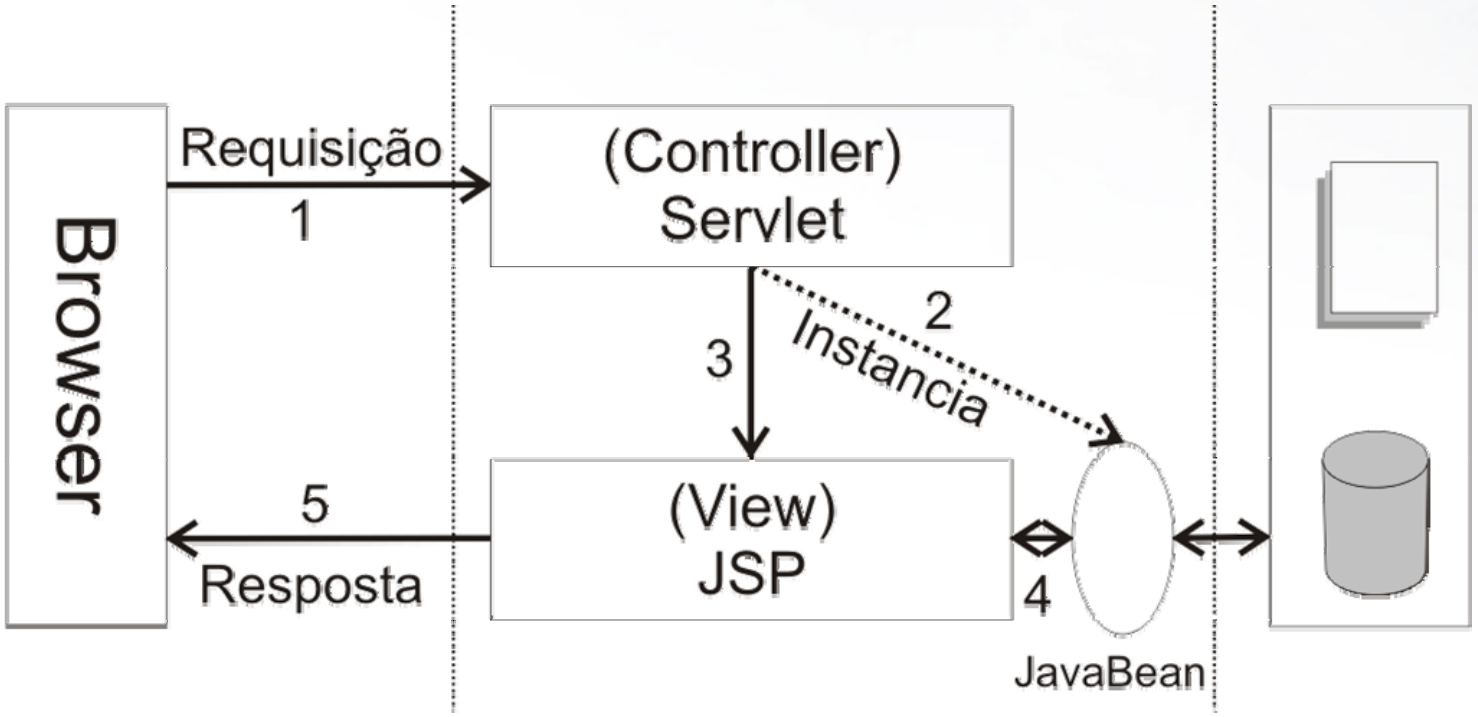
- Manter consistentes
  - Dados da sessão
    - Visão cliente-servidor
    - Página atual
    - ID do usuário
    - etc.
  - HTTP é stateless. Como eu faço isso?
    - Desenvolvedor monta uma FSM;
    - define transições legais.

Finite State Machines

- Legal, vamos usar FSMs então.
  - Contador não tem finitos estados.
  - Composição, como fica?
  - E se o usuário clonar a janela?

Controle de Fluxo

- Abordagens mais comuns
  - Struts e o MVC/Model 2

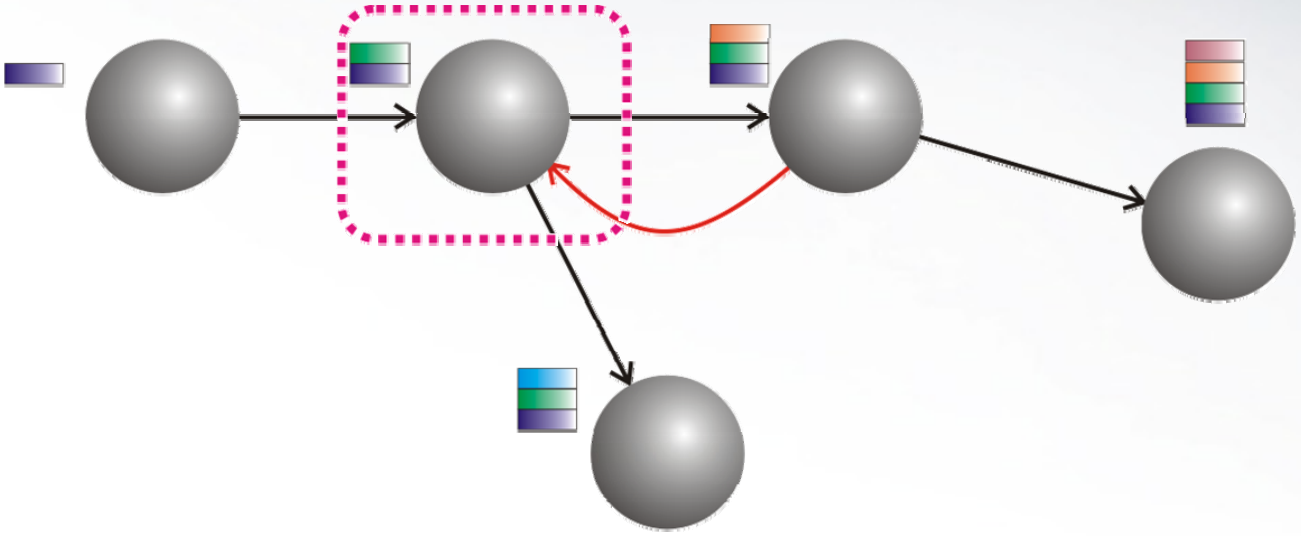


Continuações

- O que são continuações?
  - Expressão funcional do GOTO.
- Metáforas
  - Representam o restante da execução do programa (a ‘continuação’ da execução);
  - fotografia de informações de contexto;
  - ponto na execução do programa;
  - pilha de execução => árvore de execução.

Continuações (cont.)

- Árvore de continuações



```
c := Continuation new.
(someAction beenPerformed)
    ifTrue:
        [object doSomething: c].
```

```
doSomething: aContinuation
    self done: true.
    aContinuation continue.
```

Seaside

- Seaside
  - Continuações de primeira ordem;
  - estado fica do lado do servidor;
  - design baseado em componentes.
- Componente
  - Fragmento da página;
  - auto-contido;
  - pode chamar outros componentes;
  - lembra muito os Java Portlets.

Seaside (cont.)

- Aplicação Seaside: árvore de componentes
- WAComponent = view + controller (+ model?)
- Completitude comportamental
  - Idéia legada do Simula, ressucitada pelos Naked Objects
  - Objetos são auto-contidos
  - Baixíssimo acoplamento, altíssima coesão
- Não existe uma “Smalltalk Server Page”.
  - Geração de html é programática;
  - formatada com CSS.

Exemplo Preliminar

- WATask
  - Fluxos complexos, não pulverizados. Exemplo:

```
go
        | shipping billing creditCard |
        cart := WAStoreCart new.
        self isolate:
                [[self fillCart.
                self confirmContentsOfCart]
                        whileFalse].

        self isolate:
                [shipping := self getShippingAddress.
                billing := (self useAsBillingAddress: shipping)
                                        ifFalse: [self getBillingAddress]
                                        ifTrue: [shipping].
                creditCard := self getPaymentInfo.
                self shipTo: shipping billTo: billing payWith: creditCard].

        self displayConfirmation.
```

Componentes Seaside

- WATask (cont.)
  - Tipo especial de componente;
  - encapsula fluxo;
  - método **go**.
- WAComponent
  - #renderContentOn: html
    - html – “canvas”;
    - igual ao *Graphics g* de Java;
    - abstração do HTML.

Componentes Seaside (cont.)

- Modelo de callbacks
  - Essencialmente chamados em resposta ao POST
  - Os *callbacks* são de dois tipos:
    - Valor (*value*)*:*
      - executam primeiro.
      - Associados aos subcomponentes. Exemplo:

```
html textAreaWithValue: 'Texto:'
  callback: [ :texto | self texto: texto.].
```

    - Ação (*action*):
      - Executam por último.
      - Responsáveis pelas decisões de fato, muitas vezes chamam outros componentes.
      - Partem do pressuposto que os dados já foram lidos.

Componentes Seaside (cont.)

- Action *callback* (exemplo):

```
html form: [
        html submitButtonWithAction: [self answer: true]
             text: 'Proceed with checkout'.
        html break.
        html submitButtonWithAction: [self answer: false]
             text: 'Modify my order'.
]
```

Visão geral do Seaside (cont.)

- Digressão aos fluxos
  - Mecanismo call/answer – a continuação
  - #call: aComponent
    - Salva o ponto de continuação atual.
    - Componente cede seu lugar a outro componente.
  - #answer: someValue
    - Retoma a continuação que precede a atual, devolvendo o parâmetro “someValue” como resultado.
  - Vamos revisitar a WATask.

WATask revisitada

- Vamos só olhar o começo.
  - WAStore – root component.
  - Insere a task num quadro abaixo do cabeçalho.

WAStoreTask>>go

```
| shipping billing creditCard |
cart := WAStoreCart new.
self isolate:
        [[self fillCart.
        self confirmContentsOfCart]
                whileFalse].
```

WAStoreTask>>fillCart

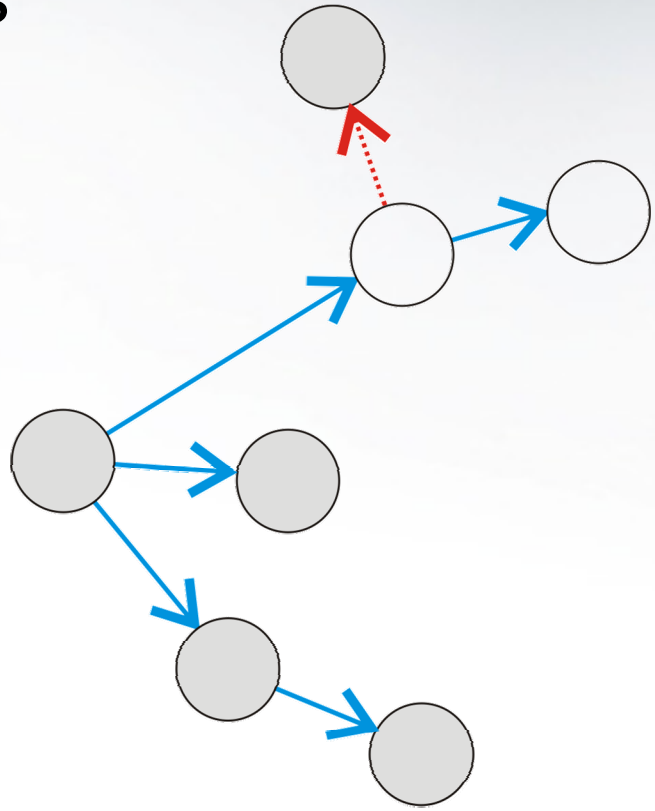
```
self call: (WAStoreFillCart new cart: cart)
```

WAStoreFillCart>>children

```
^ Array with: cartView with: main
```

WATask e WAComponent (cont.)

- WAComponent (cont.)
  - Podem conter subcomponentes
  - Declaram seus filhos
  - A cada ciclo GET/POST:
    - Árvore determina quais componentes mostrar;
    - HTML é gerado;
    - usuário preenche campos e envia dados;
    - *Callbacks* são disparados.



WATask e WAComponent

- WATask e WAComponent
  - Cada componente define seu próprio fluxo;
  - composição de componentes = criação/composição de fluxos;
  - smalltalk para definição de fluxos = fluxos complexos ficam simples de definir.
- Você não precisa se preocupar com:
  - A lógica que propaga informações entre páginas;
  - o estado do servidor;
  - a lógica que codifica quais componentes exibir.

Backtracking e transações

- Backtracking
  - Guardar o estado completo de cada objeto é muito custoso.
  - Solução:
    - Guardar somente parte do estado.
    - Cada componente registra pedaços para o backtracking via Session >> registerObjectForBacktracking: anObject.
- Transações:
  - Backtracking só entre o início e o fim da transação.
  - Component#isolate: transactionalBlock.

Desenvolvimento no Seaside

- Smalltalk é reflexiva!
- A mescla compilação-execução nos permite:
  - Mexer no servidor enquanto ele roda (take that, JBoss!)
  - Abrir o depurador.
  - Inspecionar quaisquer aspectos da aplicação e interagir com ela ao mesmo tempo.

Sumário

- Seaside
  - Baseado em continuações;
  - modular;
  - modela fluxos e componentes em alto nível;
  - conta com poderoso ambiente de desenvolvimento;
  - permite combinar múltiplos fluxos;
  - tudo em uma única linguagem.

Dúvidas?