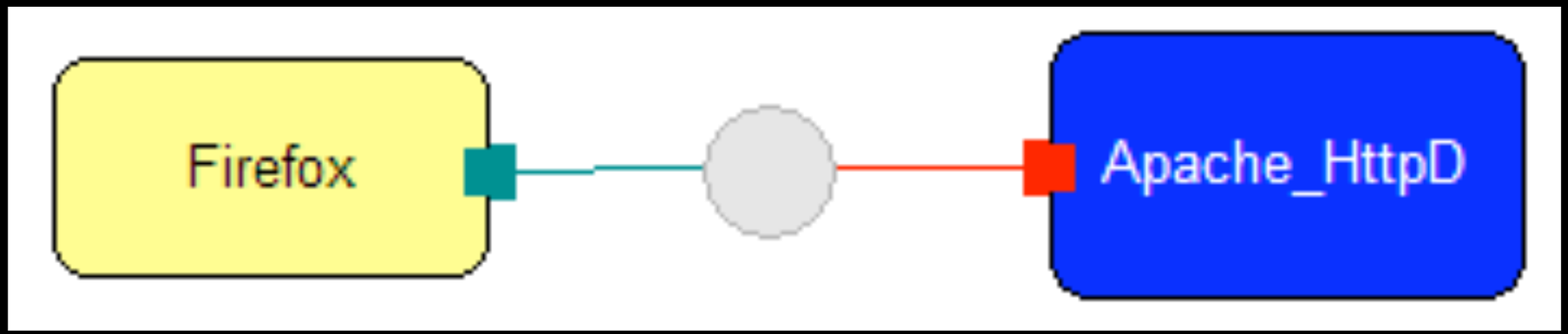
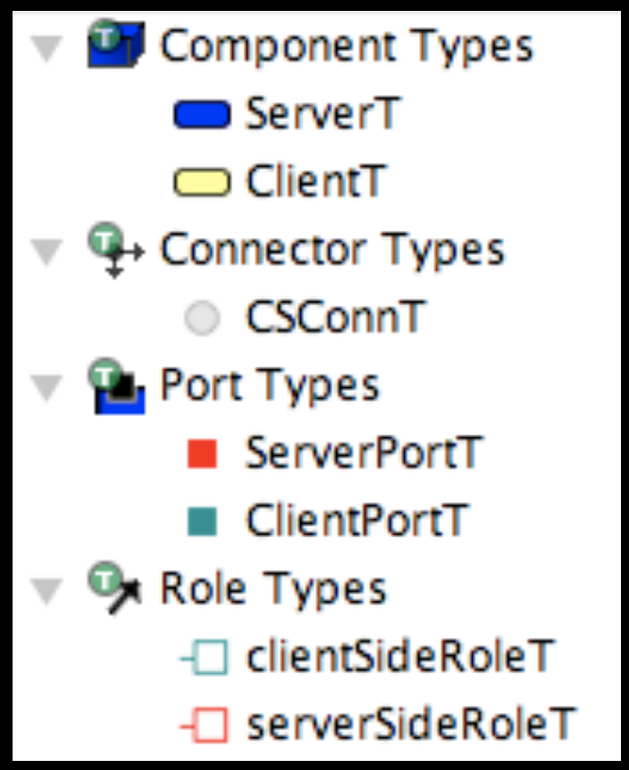
Comportamento de componentes e conectores

Paulo Borba e Fernando Castor
Centro de Informática
Universidade Federal de Pernambucio

Componentes e conectores...





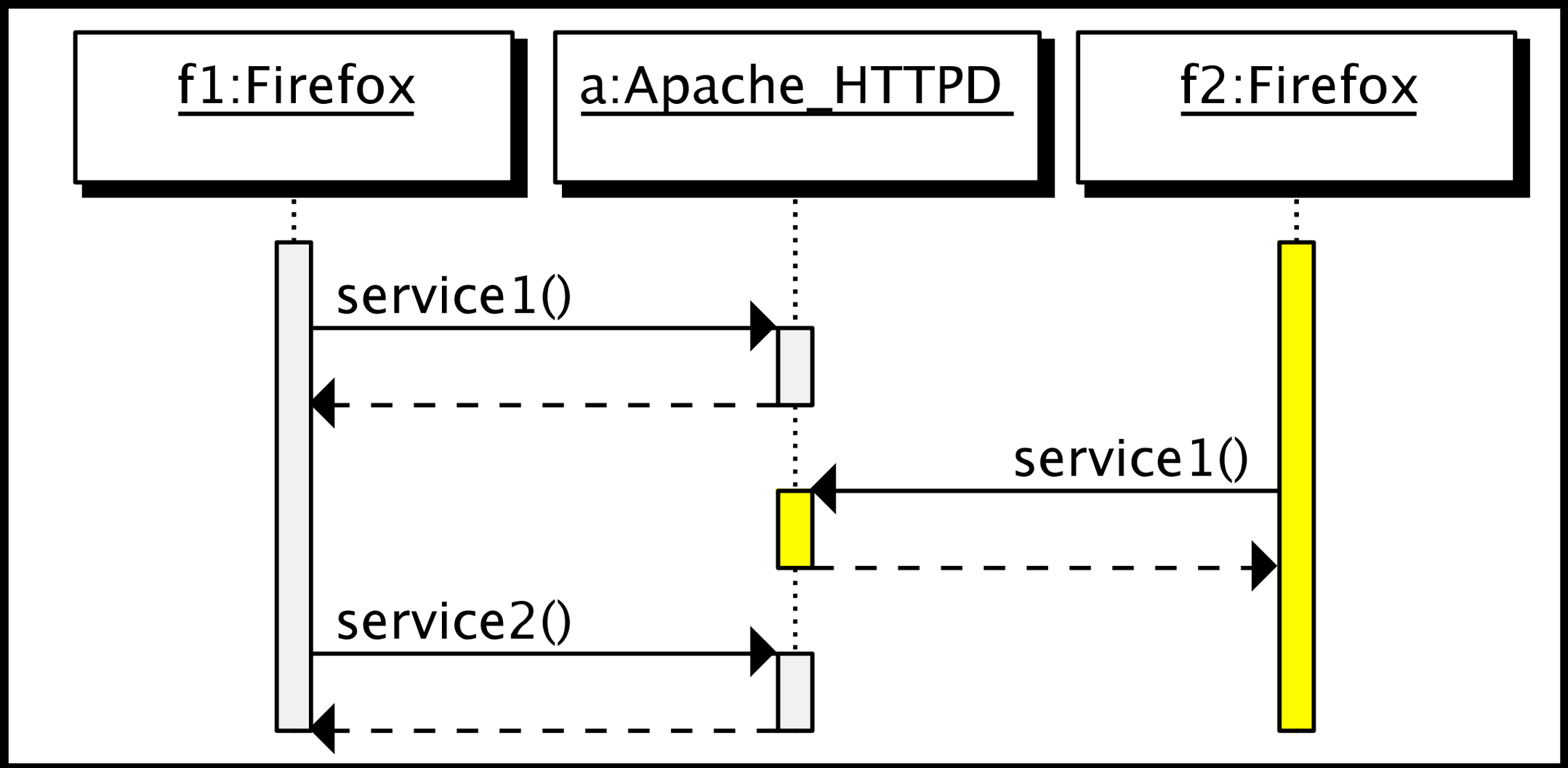
http://www.cs.cmu.edu/~acme/AcmeStudio/index.html

sintaxe bem definida,
mas bem abstrata...

interfaces poderiam
detalhar melhor
serviços associados às
portas!

mas semântica ainda seria apenas parcial e imprecisa!

Diagramas de sequência, comportamento menos parcial



comportamento
completo
=
todos os possíveis
diagramas de
sequência?

Trabalhoso!

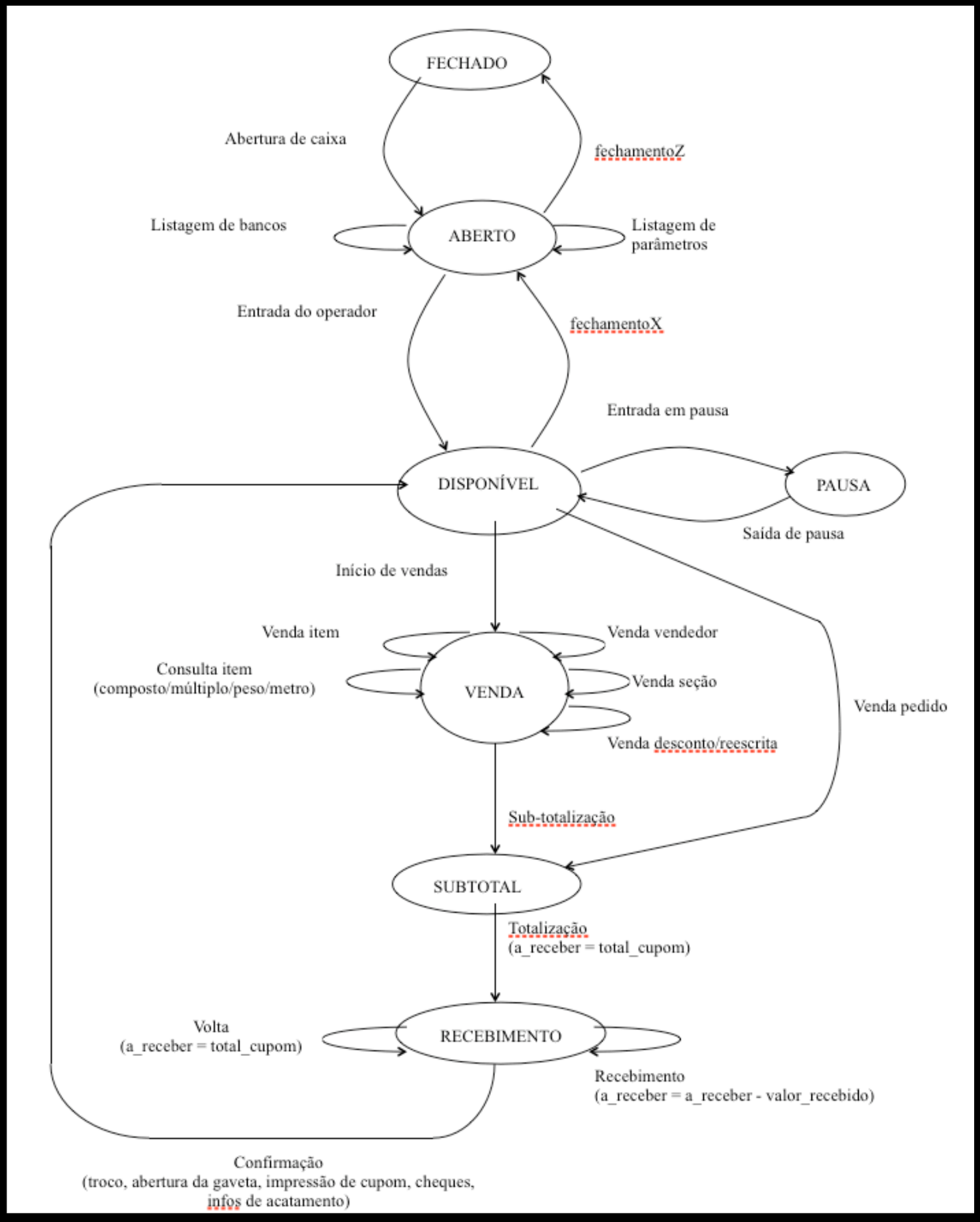
comportamento
completo
=
todos os possíveis
fluxos de execução

Também
trabalhoso...
e infinito!

Mas emerge das possíveis sequências de eventos de...

- Componentes (portas)
- Conectores (papéis e ligações entre eles)
- e da composição e interação dos mesmos

a ordem de eventos pode ser codificada em termos de pré e pós condições, mas melhor não!



máquina de estados: ordem de eventos

Faltam recursos para composição!

Falta CSP

(álgebra de processos)

Eventos

- A chamada de um método atômico
- O início ou fim da chamada de um método não atômico
- A notificação de um evento da GUI ou de algum periférico
- Algo atômico, instantâneo

ordem de execução
de eventos
=
processo

Processos em CSP

Declarando os eventos ou canais de comunicação:

```
channel up, down
```

Definindo (nomeando) um processo cujo alfabeto é determinado pelos eventos acima:

```
P0 = up -> down -> up ->

     down -> STOP
```

Escolha externa e parâmetros (estado)

```
COUNT(n)=
   if n==0 then
      (up -> COUNT(1))
   else
      (   up -> COUNT(n+1)
       []
          down -> COUNT(n-1))
```

Escolha Prefixada

Operador para construir processos:

`?x:A -> P(x)`

dado uma variável `x`, um conjunto de eventos `A`, e um processo `P(x)`

`STOP` equivale a `?x:{} -> P(x)`

Eventos Compostos

```
DATA = {0,1}

channel left, right:DATA

COPY = left?x -> right!x -> COPY
```

```
Binf(s) =
 if s==<> then
     left?x -> Binf(<x>)
 else  (left?x -> Binf(s^<x>)
       [] right!head(s) -> Binf(tail(s)))
```

Escolha Interna *versus* Externa

```
(a -> STOP)[](b -> STOP)
```

sempre oferece os eventos `a` e `b`

```
(a -> STOP) |~| (b -> STOP)
```

pode rejeitar `a` ou `b`

Composição Paralela

Operador para construir processos:

`P [|X|] Q`

dado processos `P` e `Q` e um conjunto de eventos `X`

`P` e `Q` são executados em paralelo mas só sincronizando os eventos em `X`:

os outros eventos são realizados de forma independente

Outros operadores de composição paralela

```
P || Q = P [|Events|] Q

P ||| Q = P [|{}|] Q
```

Comportamento de client-server

```
Connector CSconnector
    Role Client = (request!x -> result?y -> Client) |~| Tick
    Role Server = (invoke?x -> return!y -> Server) [] Tick

    Glue = (Client.request?x -> Server.invoke!x ->
              Server.return?y -> Client.result!y -> Glue)
            [] Tick
```

Roles são
especificações para
as portas

Comportamento de shared data, roles

```
Connector SharedData3
    Role Initializer = set -> A
      where {
        A =       set -> A
                |~| get -> A
                |~| Tick
      }
    Role User =       set -> User
                    |~| get -> User
                    |~| Tick
```

Comportamento de shared data, glue

```
Glue =     Initializer.set -> Continue
        []  User.set -> Continue
        []  Tick
    where {
      Continue = Initializer.set -> Continue
          [] User.set -> Continue
          [] Initializer.get -> Continue
          [] User.get -> Continue
          [] Tick
    }
```

Verificando habilidades...

- Defina o comportamento da instância de publisher-subscriber da sua arquitetura do Google Wave

Comportamento de componentes e conectores

Paulo Borba e Fernando Castor
Centro de Informática
Universidade Federal de Pernambucio