Linguagens Interpretadas na Adaptação Dinâmica de Aplicações

Simpósio Brasileiro de
Linguagens de Programação 2004

Renato Cerqueira
Departamento de Informática - PUC-Rio

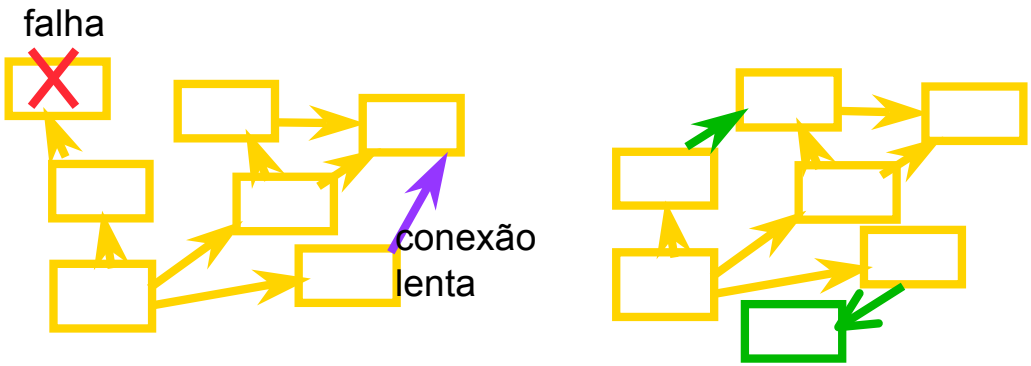
Por que adaptação dinâmica?

- Manutenção sem interrupção dos sistemas
  - Correções de bugs
  - Novos requisitos do usuário
- Tratamento de falhas
- Variações no ambiente de execução do sistema
- Flexibilidade para acomodar alterações não previstas durante o desenvolvimento do sistema

O que é adaptação dinâmica?

- Definição
  - Alteração de um sistema em execução

- Programa deve lidar com ambiente que muda dinamicamente
  - Alocação de recursos deve "reagir" ao estado de execução



Exemplos

Sistema	Características	Exemplo
auto adaptativo	reage a mudanças em seu ambiente, ajustando automaticamente seu comportamento	agentes inteligentes
longa duração	serviço oferecido continuamente	caixa eletrônico
autônomo a distância	intervenção humana improvável	robô caça-minas
missão crítica	interrupção proibitiva ou de alto custo	cobrança telefônica
Time to market	1a. versão com recursos mínimos p/ chegar logo ao mercado, mas com atualizações freqüentes posteriormente	reserva de passagens aéreas

Tipos de alterações

- Esperadas
  - Aplicações inteligentes: computação móvel, agentes de software
- Inesperadas
  - Sistemas duradouros: essenciais ou computacionalmente pesados

Adaptação em ponto grande

- Alterações na estrutura macro do sistema
  - Modular
    - Substituição de módulos
  - Orientação a objetos
    - Troca de objetos e relacionamentos
  - Componentes
    - Troca de conexões entre componentes

Adaptação em ponto pequeno

- Alterações internas aos componentes que formam o sistema
  - Estruturas mais fundamentais
  - Nível de detalhe maior
- Níveis de detalhe
  - Ambiente de execução (*middleware*)
  - Linguagem de programação
  - Componentes/objetos
    - Consulta ao estado e estrutura interna de um componente
    - Manipulação da implementação do componente

Ponto grande vs. Ponto pequeno

- Ponto grande
  - Mais simples e eficiente
  - Exige planejamento da arquitetura para acomodar alterações
  - Adequada para alterações esperadas
  - Aplicações auto-adaptáveis (alterações no contexto)
- Ponto pequeno
  - Mais flexível, porém mais complexa
  - Adequada para alterações não esperadas
  - Manutenção dinâmica de sistemas

Mecanismos de adaptação

- Reflexão computacional
  - Introspecção
    - Interfaces de módulos e objetos (ex.: API de reflexão de Java, repositório de interfaces de CORBA)
    - Estado do sistema
  - Intercessão
    - Pontos em que podem ser introduzidas alterações no sistema
    - Interceptadores, ponteiros para função, meta-classes, etc.

Linguagens de Programação

- Smalltalk
  - metaclasses
- Common Lisp Object System (CLOS)
  - protocolos de metaobjetos
- Java
  - Pacote `Reflection`: classes `Class` e `Proxy`
  - Classe `ClassLoader`
- C#
  - Classe `System.Type`
  - Pacotes `System.Reflection` e `System.Reflection.Emit`
- Linguagens interpretadas
  - *eval*

Plataformas para sistemas distribuídos

- As plataformas para sistemas distribuídos têm evoluído para oferecer um melhor suporte à adaptação dinâmica
  - Serviços para localização de componentes (nomes, diretórios e de *trading*)
  - Descrição de interfaces de serviços (Repositório de Interfaces)
  - Construção de chamadas dinamicamente (DII - Dynamic Invocation Interface)
  - Interceptação de chamadas (Portable Interceptors)
  - Implementação dinâmica de componentes (DSI - Dynamic Skeleton Interface)
  - Representação explícita das dependências entre componentes (CCM - CORBA Component Model)
  - Serviços com informações sobre o ambiente de execução

Mas ainda temos outros problemas...

- Prever todas as possibilidades de adaptação ainda em tempo de projeto
- Identificar o momento em que um sistema precisa se adaptar
- Identificar o momento em que um sistema pode se adaptar
- Definir os procedimentos de adaptação de forma modular
- Gerenciar os procedimentos de adaptação

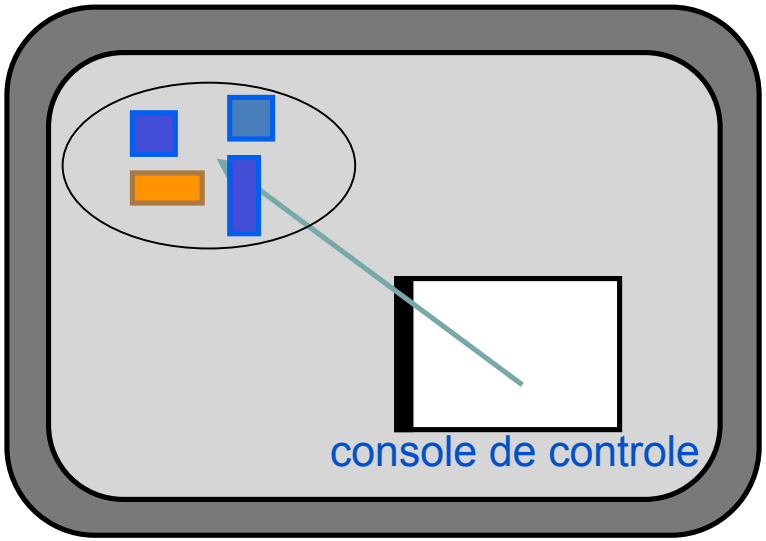
Precisamos experimentar mais para entendermos melhor o problema...

Obstáculo para a experimentação

- Dificuldade de uso!!!
  - Dependência excessiva de mecanismos reflexivos?
  - Ferramentas de programação inadequadas?
  - Incompatibilidade entre linguagens estaticamente tipadas e reflexão computacional?

Projeto do grupo de linguagens da PUC-Rio

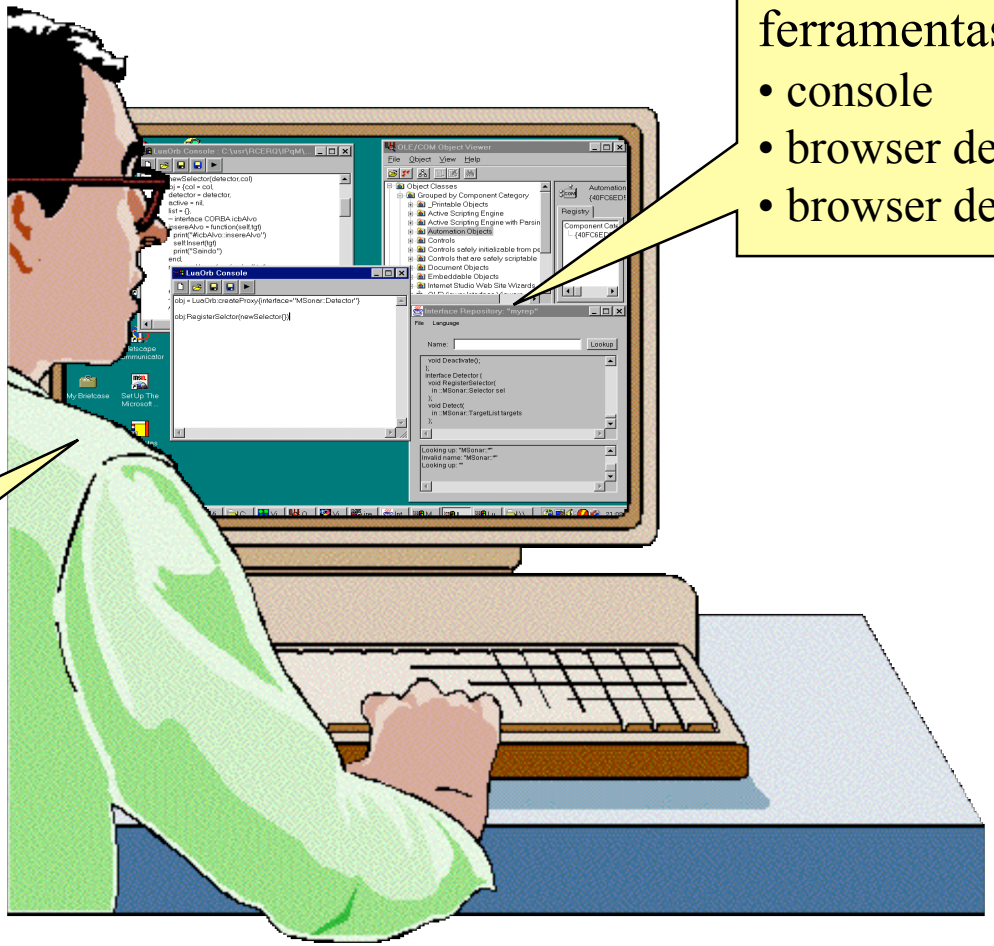
- uso de uma linguagem interpretada para flexibilizar a programação distribuída
  - adaptação e interoperabilidade
  - ambiente programmer-friendly!
  - simplicidade + poder expressivo
  - Noemi Rodriguez
  - Renato Cerqueira
  - Roberto Ierusalimschy



Vantagens

- Prototipagem
  - Flexibilidade para programação exploratória
- Programação distribuída
  - Suporte para reconfiguração dinâmica
- Interoperabilidade
  - Integração transparente entre diferentes sistemas de componentes

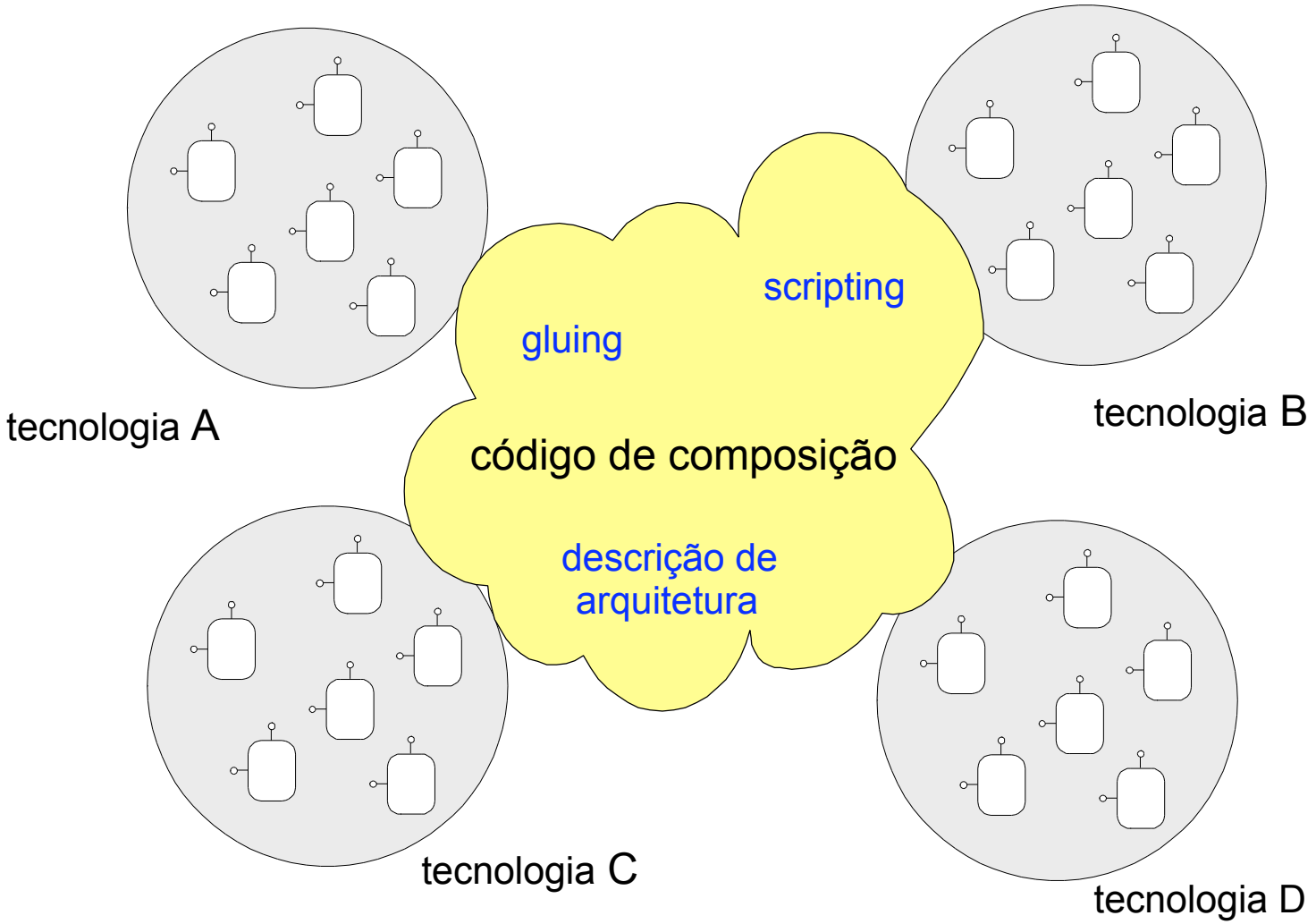
Ambiente típico



ferramentas:
- console
- browser de interfaces
- browser de componentes

usuário avançado: escreve código que combina componentes

Uso da linguagem Lua



Lua

- Mais uma linguagem de script
  - similar a Python, Tcl, Perl, Ruby, …
- Feita para ser embutida em aplicações
  - é uma biblioteca C (ANSI C **puro**)
- Uma linguagem para descrição de dados
  - Capacidade de expressão similar a XML
- Simples e flexível
  - "coisas simples são simples, coisas complexas são possíveis"

Por que Lua?

- Leve
  - uma linguagem simples e pequena, com poucos conceitos
  - núcleo com aprox. 60K, executável completo com 140K
- Portável
  - escrita em ANSI C *clean*
  - executa em PalmOS, EPOC (Symbian), Brew (Qualcomm), Playstation II, XBox, sistemas embutidos, mainframes, etc.
- Eficiente (ver *benchmarks*)
- Fácil de ser embutida
  - C/C++, Java, Fortran, Ruby, OPL (EPOC), C#, ...

Lua é pequena

- Tamanhos de algumas linguagens (Solaris)
  - Lua 4.0: 120K
  - Tcl (? < 8.0): 655K (~ 5x)
  - Python 2.0: 860K (~ 6.5x)
  - Perl 5.6: 1.1M (~ 8.5x)
- Seu tamanho pequeno tem outros benefícios:
  - Fácil de instalar
  - Fácil de adaptar
  - Portável

Lua é eficiente

- Mais rápida que Perl, Python; muito mais rápida que Tcl
  - Ver **`www.bagley.org/~doug/shootout`**
- Eficiente para descrição de dados
  - Compilador rápido
- Fácil de ser embutida em C/C++
  - Fácil recorrer a C quando eficiência é crítica

Alguns resultados de desempenho

- Computer Language Shootout Scorecard
  - http://dada.perl.it/shootout/

- Comparações
  - Tempo de CPU
  - Uso de memória
  - Linhas de código

Tempo de CPU

SCORES					
Language	Implementation	Score	Missing	Failing	Avg.Score
Delphi	delphi	553	7	0	30.72
C	vc	665	2	1	30.23
C	bcc	665	3	0	30.23
Ocaml	ocaml	665	0	3	30.23
C	lcc	663	3	0	30.14
C	mingw32	663	1	2	30.14
Mercury	mercury	475	7	2	29.69
C	gcc	674	0	2	29.30
Lua	lua5	638	2	1	29.00
Ocaml	ocamlb	630	0	3	28.64
Java	java	710	0	0	28.40
Forth	gforth	595	1	3	28.33
SML	smlnj	479	0	8	28.18
Lua	lua	648	2	0	28.17
Pike	pike	675	0	1	28.13
Ada	gnat	447	8	1	27.94
Eiffel	se	502	5	2	27.89
Nice	nice	556	5	0	27.80
Common Lisp	poplisp	389	0	11	27.79
Pascal	fpascal	305	8	6	27.73
C++	vc++	442	5	4	27.63
Pascal	vpascal	465	8	0	27.35
C#	csharp	574	4	0	27.33

Uso de memória

SCORES					
Language	Implementation	Score	Missing	Failing	Avg.Score
Ocaml	ocamlb	643	0	3	29.23
C	vc	617	2	1	28.05
C	lcc	617	3	0	28.05
Delphi	delphi	504	7	0	28.00
C	mingw32	616	1	2	28.00
C	bcc	615	3	0	27.95
Ocaml	ocaml	598	0	3	27.18
Lua	lua5	588	2	1	26.73
Icon	icon	424	9	0	26.50
Oz	oz	447	6	2	26.29
Pascal	fpascal	286	8	6	26.00
C++	vc++	407	5	4	25.44
Eiffel	se	457	5	2	25.39
Ada	gnat	403	8	1	25.19
SML	smlnj	427	0	8	25.12
Mercury	mercury	399	7	2	24.94
Lua	lua	560	2	0	24.35
ICI	ici	560	0	2	24.35
C	gcc	557	0	2	24.22
Forth	gforth	497	1	3	23.67

Linhas de código

SCORES					
Language	Implementation	Score	Missing	Failing	Avg.Score
Common Lisp	poplisp	464	0	11	33.14
Modula-2	***modula2***	250	15	2	31.25
Ada	***gnat***	470	8	1	29.38
Nice	***nice***	580	5	0	29.00
VBScript	vbscript	365	5	7	28.08
Haskell	***ghc***	402	4	6	26.80
Erlang	erlang	373	3	8	26.64
Ruby	ruby	532	0	4	25.33
Scheme	guile	522	0	4	24.86
Awk	gawk	431	6	1	23.94
Forth	***bigforth***	236	9	6	23.60
ICI	ici	535	0	2	23.26
Rexx	rexx	320	7	4	22.86
PHP	php	318	9	2	22.71
Awk	***awka***	431	6	0	22.68
Awk	mawk	431	6	0	22.68
Ocaml	ocamlb	489	0	3	22.23
Ocaml	***ocaml***	489	0	3	22.23
Python	python	518	0	1	21.58
Perl	perl	496	0	2	21.57
Perl	cygperl	493	0	2	21.43
Icon	icon	338	9	0	21.13
Pike	pike	504	0	1	21.00
Lua	lua5	459	2	1	20.86

CPU + memória

SCORES					
Language	Implementation	Score	Missing	Failing	Avg.Score
Delphi	***delphi***	1058	7	0	58.78
C	***vc***	1283	2	1	58.32
C	***lcc***	1281	3	0	58.23
C	***bcc***	1280	3	0	58.18
C	***mingw32***	1279	1	2	58.14
Ocaml	ocamlb	1273	0	3	57.86
Ocaml	***ocaml***	1263	0	3	57.41
Lua	lua5	1227	2	1	55.77
Mercury	***mercury***	874	7	2	54.63
Pascal	***fpascal***	591	8	6	53.73
C	***gcc***	1232	0	2	53.57
SML	***smlnj***	907	0	8	53.35
Eiffel	***se***	960	5	2	53.33
Ada	***gnat***	850	8	1	53.13
C++	***vc++***	849	5	4	53.06
Icon	icon	846	9	0	52.88
Lua	lua	1209	2	0	52.57
Forth	gforth	1092	1	3	52.00
Oz	***oz***	870	6	2	51.18
ICI	ici	1170	0	2	50.87
S-Lang	slang	953	6	0	50.16
Pascal	***vpascal***	852	8	0	50.12
Modula-2	***modula2***	395	15	2	49.38
Awk	mawk	922	6	0	48.53
Python	python	1141	0	1	47.54

CPU + memória + LDC

SCORES					
Language	Implementation	Score	Missing	Failing	Avg.Score
Ada	gnat	1320	8	1	82.50
Modula-2	modula2	645	15	2	80.63
Ocaml	ocamlb	1762	0	3	80.09
Ocaml	ocaml	1752	0	3	79.64
Lua	lua5	1686	2	1	76.64
Common Lisp	poplisp	1040	0	11	74.29
ICI	ici	1705	0	2	74.13
Icon	icon	1184	9	0	74.00
Delphi	delphi	1306	7	0	72.56
C++	vc++	1153	5	4	72.06
C	vc	1581	2	1	71.86
Lua	lua	1652	2	0	71.83
Awk	mawk	1353	6	0	71.21
C	mingw32	1554	1	2	70.64
C	lcc	1547	3	0	70.32
C	bcc	1546	3	0	70.27
Pascal	fpascal	765	8	6	69.55
Forth	gforth	1459	1	3	69.48
Python	python	1659	0	1	69.13
S-Lang	slang	1299	6	0	68.37
Forth	bigforth	679	9	6	67.90
Mercury	mercury	1087	7	2	67.94
Awk	awka	1290	6	0	67.89
Perl	perl	1549	0	2	67.35

Algumas aplicações

- Jogos
  - LucasArts, BioWare, Microsoft, Relic Entertainment, Absolute Studios, Monkeystone Games, etc.
- Outros usos
  - tomsrtbt: ”The most Linux on one floppy disk”
  - Crazy Ivan Robot (campeão das RoboCup 2000 e 2001 na Dinamarca)
  - Sistema de monitoramento de pacientes com PDAs (InCor)
  - Layout de chips (Intel)
  - APT-RPM (Conectiva e United Linux)
  - Space Shuttle Hazardous Gas Detection System (ASRC Aerospace)

Evolução da linguagem Lua

- Diretrizes
  - Simplicidade
  - Eficiência
  - Portabilidade
  - Extensibilidade
- Versão 1.0
  - Descrição de dados
  - Recursos básicos de processamento
- Versão 2.0
  - Mecanismos de extensibilidade
  - Suporte a OO
- Versão 3.0
  - Melhor e mais extensibilidade
- Versão 4.0
  - Múltiplos estados e preocupação com concorrência
- Versão 5.0
  - Programação em ponto grande (programação modular)
  - Programação concorrente
- Novas Tendências
  - Sistemas de tempo real

Como é Lua?

- Sintaxe similar a Pascal
- Tipagem dinâmica
- Gerenciamento de memória automático
  - Estruturas de dados dinâmicas
  - Coleta de lixo
- Sete tipos básicos
  - números, booleanos, tabelas, funções, strings, userdata, nil

```
function fat (n)
  if n == 0 then
    return 1
  else
    return n*fat(n-1)
  end
end
```

Tabelas

- Implementam arrays associativos
  - qualquer valor (incluindo funções e outras tabelas) pode ser usado como índice e valor
- Crescem e diminuem dinamicamente
  - sem gerar lixo
- Semântica referencial
- Tabelas implementam as estruturas de dados mais comuns de forma fácil e eficiente
  - arrays, *records*, conjuntos, arrays esparsos, tabelas(!), listas, etc.

Tabelas x Estruturas de Dados

- arrays: números como índices
  - muito eficiente
  - algoritmo para armazenar arrays como arrays!

```
local a = {}
for i=1,N do
  a[i] = 0
end
```

- records: strings literais como índices
  - açúcar sintático: `a.x` ≡ `a["x"]`

```
a.x = "hello"
print(a.x)
```

- conjuntos: valores como índices

```
a[x] = 1          -- insere x
a[y] = nil        -- remove y
if a[z] ...       -- pertence?
```

Construtores

- Expressões para criar e inicializar tabelas
- Estilo *record*

```
point = {x=10, y=20}
print(point.y)   --> 20
```

- Estilo lista

```
days = {"Sun","Mon","Tue","Wed","Thu","Fri","Sat"}
print(days[3])  --> Tue
```

- Estilo misto

```
points = {{x=0,y=0}, point; n=2}
```

Construtores e Descrições de Dados - exemplo 1

açúcar sintático `set({...})`

```
function Set(tb)
   local set = {}
   for i,v in tb do
      set[v] = true
   end
   return set
end

s = Set{"joão", "maria", "josé"}

print(s["joão"])    -- true
print(s["pedro"])   -- nil
```

Construtores e Descrições de Dados - exemplo 2

açúcar sintático `book({...})`

```
book{
   author="F.P.Brooks",
   title="The Mythical Man-Month",
   year=1975,
}

book{
   author="Jon Bentley",
   title = "Programming Pearls",
   year=1986,
}
```

Funções

- Valores de primeira classe

```
function inc (x)
  return x+1
end
```


```
inc = function (x)
        return x+1
      end
```

- Escopo léxico

```
function count (x)
  return function () x = x+1; return x; end
end
a = count(10)
print(a())     --> 11
print(a())     --> 12
```

Escopo léxico

- Exemplo: objetos

```
function newObject ()
  local x
  local obj = {}
  obj.get_x = function () return x end
  obj.set_x = function (new_x) x = new_x end
  return obj
end
```

Objetos

- Funções de primeira classe + tabelas = *quase* OO.
  - tabelas podem ter funções como valores de campos (≅ métodos)
- Açúcar sintático para a definição e chamada de métodos
  - parâmetro implícito *self*

```
function a:foo(x)
  ...
end
```

açúcar

```
a.foo = function(self,x)
  ...
end
```

```
a:foo(x)
```

açúcar

```
a.foo(a,x)
```

Metatables

- *Metatables* redefinem o comportamento das tabelas (e *userdata*)

- Exemplo típico: herança (delegação)
  - o campo "index" em sua *metatable* define como uma tabela deve tratar o acesso a índices ausentes
  - o valor do campo "index" pode ser uma tabela (delegação direta) ou uma função (comportamento genérico)

Herança (delegação)

```
Class = {x = 0}
Class.__index = Class

function Class:add (d)
  self.x = self.x + d
end

function Class:new (o)
  setmetatable(o, self)
  return o
end
```

```
b = Class:new{}
print(b.x)    -> 0
b:add(20)
print(b.x)    -> 20
```

Herança (delegação)

```
b:add(20) →  b.add(b, 20) →  b["add"](b, 20)
b.["add"] →  nil ?
  metatable(b).__index["add"] →
  Class.__index["add"] →
  Class["add"] → <method add>

b.["add"](b, 20) →  <method add>(b, 20) →

          b.x = b.x + 20
```

procura por "index" na *metatable*

Interceptação

```
Obj = {
    add = function (self,x,y)
        return x+y
    end
}

print(Obj:add(4,3))

Obj = Proxy(Obj)

print(Obj:add(4,3))
```


```
7
Chamando o método add
Parâmetros:
1       4
2       3
Resultados:
1       7
7
```

Interceptação - Proxy

```
function Proxy(obj)
   local mt = {
      __index = function (proxy, index)
         if type(obj[index]) == "function" then
            return function (proxy, ...)
               print("Chamando o método "..index)
               print("Parâmetros:")
               table.foreachi(arg,print)
               local ret = {obj[index](obj,unpack(arg))}
               print("Resultados:")
               table.foreachi(ret,print)
               return unpack(ret)
            end
         else
            return obj[index]
         end
      end,
   }
   local p = {}
   setmetatable(p,mt)
   return p
end
```

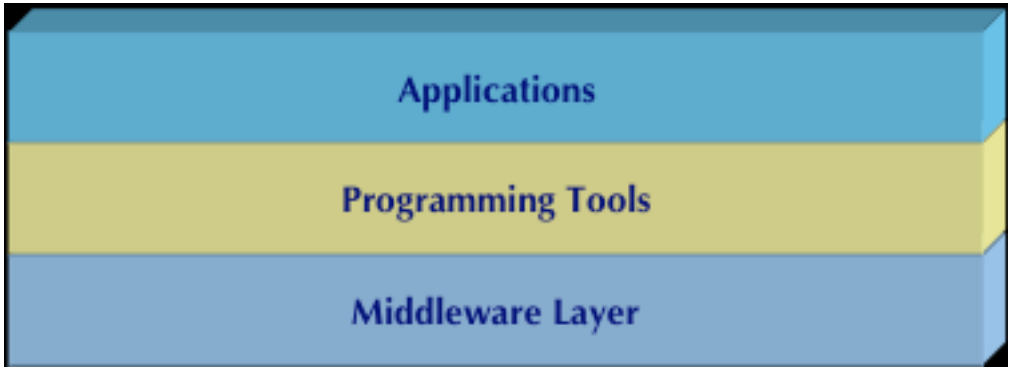
O Projeto LuaOrb

02/??	investigar modelos e ferramentas que facilitem o desenvolvimento de uma nova geração de aplicações distribuídas
00/01	investigar técnicas e ferramentas de programação para sistemas distribuídos baseados em componentes
94/99	investigar o uso de linguagens interpretadas no desenvolvimento de aplicações distribuídas baseadas em componentes (prototipação e reconfiguração dinâmica)

Principais ingredientes

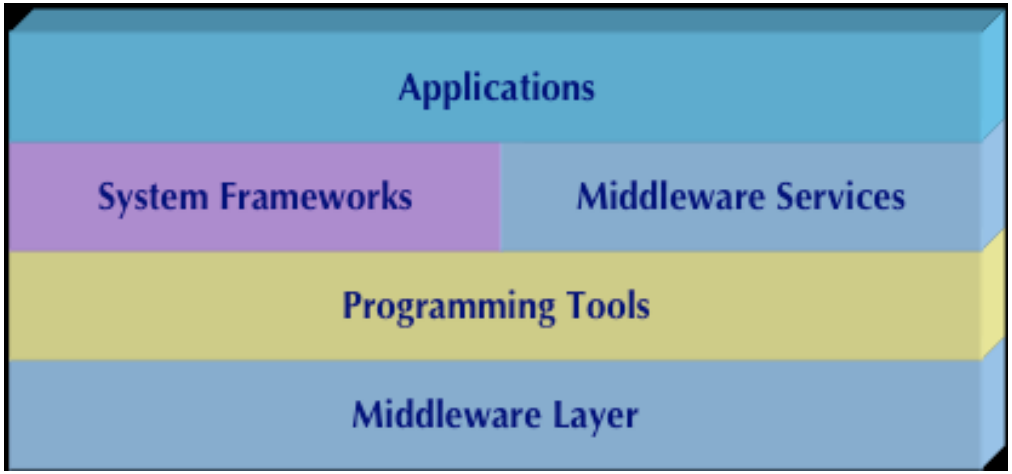
- Linguagens interpretadas
- Componentes de software
- Reflexão computacional
- Adaptações antecipadas e não-antecipadas

Fase 1 - Uso de linguagens interpretadas



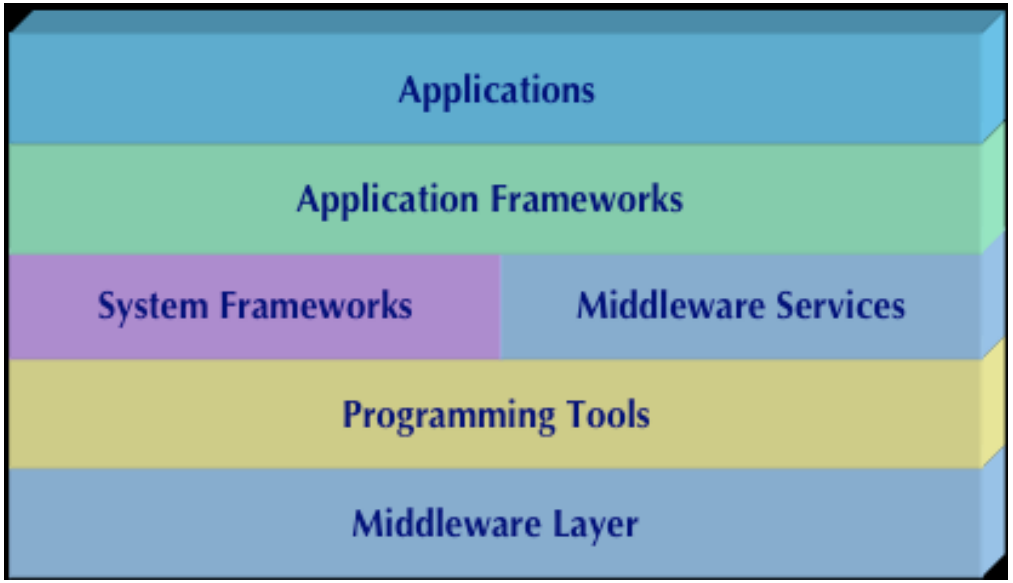
- Bindings de Lua com diferentes middlewares (CORBA, COM, Java)
- Aplicações
  - Testes de Sist. Distr.
  - Simuladores distribuídos
  - DVRL
- Interoperabilidade
- Ferramentas RAD p/ CORBA

Fase 2 - Suporte a adaptação dinâmica



- LuaOrb Adaptation Framework (LOAF)
- Interfaces para serviços CORBA
- Gerenciador de grupos
- Coordenação de espaços ativos
- Experimentação com aplicações mais dinâmicas

Fase 3 - Frameworks de aplicações



- LuaCCM
- LOAF 2.0
- Agents2CCM
- Outros bindings (.NET, Brew)
- O^2
- Apl. De CSCW
- CSBase (eScience)
- Comando e controle
- Avaliação!!

LuaOrb

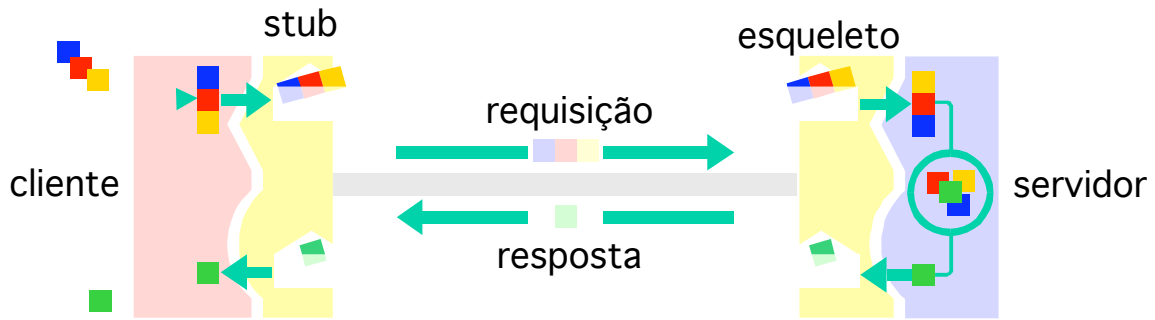
- Uma ferramenta de programação para sistemas de componentes de software (*middlewares*)
  - Atualmente trabalha com CORBA, COM, Java e .NET
  - "CORBA pode ser fácil!"
- Implementa *bindings* dinâmicos entre Lua e esses sistemas de componentes
  - Totalmente baseado em mecanismos reflexivos
  - **não usa *stubs* estáticos!**
- Oferece gratuitamente interoperabilidade entre os sistemas de middleware
  - Através de pontes dinâmicas

Principais Ferramentas

- LuaCorba - programação distribuída baseada em CORBA (objetos distribuídos)
- LuaCom - integração com a tecnologia COM da Microsoft
- LuaJava - integração com Java
- Lua.NET - integração com a tecnologia .NET da Microsoft
- LuaCCM - Modelo de componentes dinâmicos para o CCM
- O^2 - implementação CORBA totalmente em Lua para dispositivos móveis e embarcados
- LuaBrew - ferramenta de programação para celulares CDMA

Exemplo CORBA - LuaCorba

- Modelo tradicional:
  - criação de programa: geração de *stubs* e esqueletos, desenvolvimento de código, compilação, ...
  - como fica a adaptação dinâmica...?



LuaCorba

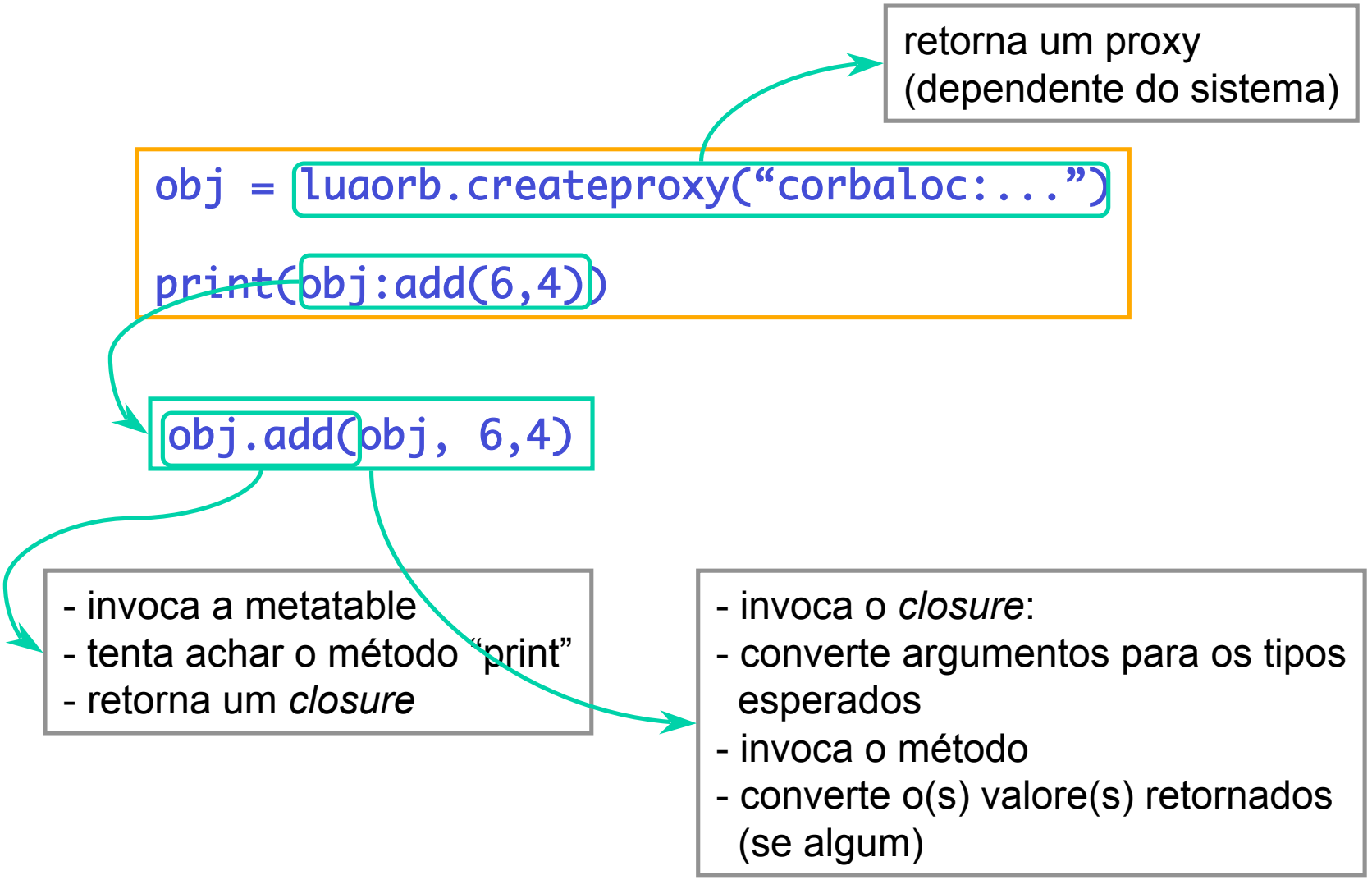
- *Binding* entre a linguagem e Lua e CORBA
- Características dinâmicas de Lua
  - fica natural o uso das interfaces dinâmicas de CORBA
- Composição e criação dinâmica
  - novas interfaces e componentes podem ser definidos em tempo de execução
- Objetos CORBA usados como objetos locais

Exemplo Cliente

```
interface Calc {
   long add (in long x, in long y);
};
```

```
obj = luaorb.createproxy("corbaloc::host:7777/CalcServer",
                         "Calc")
print(obj:add(6,4))
```

Mecanismo de chamada dinâmica



Exemplo Servidor

criação explícita:

```
calc_obj = {add = function (self, x, y)
                            return x+y
                          end
           }
Server = luaorb.createservant(”Calc",calc_obj)
```

criação implícita:

```
collection:Register(calc_obj)
```

Exemplo LuaJava

```
public class Calc {
    public:
    float add (float x, float y) {
        return x+y;
    }
}
```

```
calc = LuaOrb:javaNew("Calc")
print(calc:add(6,4))
```

LuaOrb e Reflexão Computacional

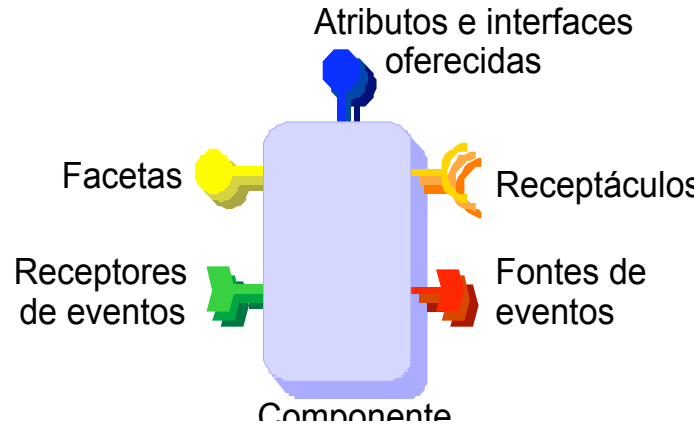
- Mecanismos introspectivos dos sistemas de componentes permitem consultas ao estado do ambiente durante execução
  - repositório de interfaces
  - Trader e propriedades dinâmicas
- Mecanismos reflexivos de Lua permitem redefinir o comportamento da linguagem em situações de exceção
  - chamadas a métodos não existentes!!
  - possibilidade de abstração de alto nível

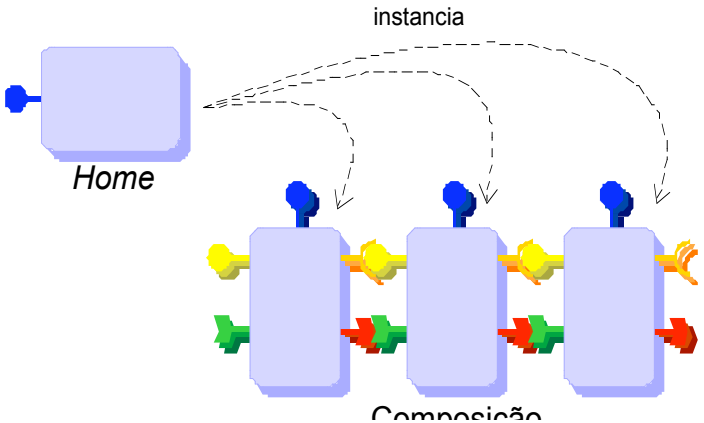
LuaCCM

- A versão 3.0 de CORBA definiu o Modelo de Componentes de CORBA (CORBA Componente Model - CCM)

- LuaCCM define um modelo de componentes dinâmicos para o CCM

O Modelo CCM

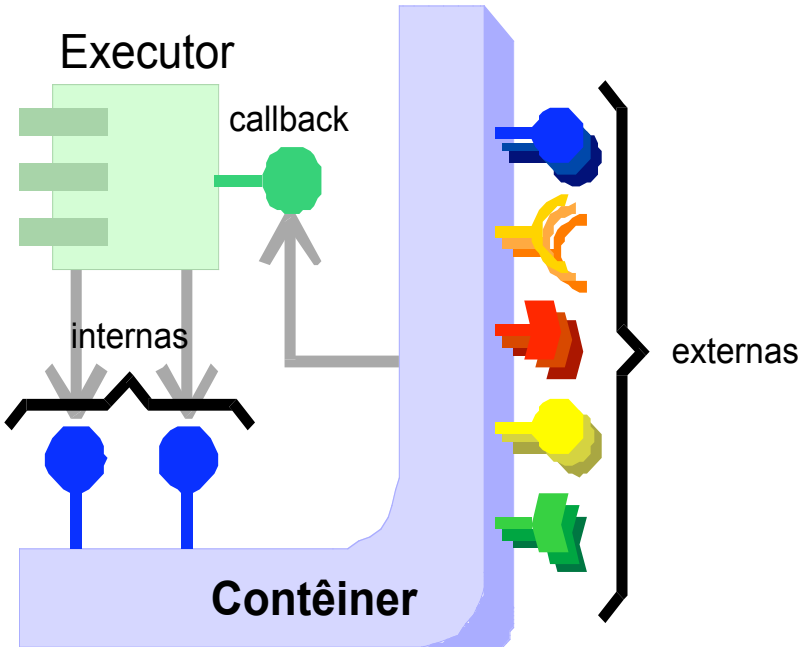




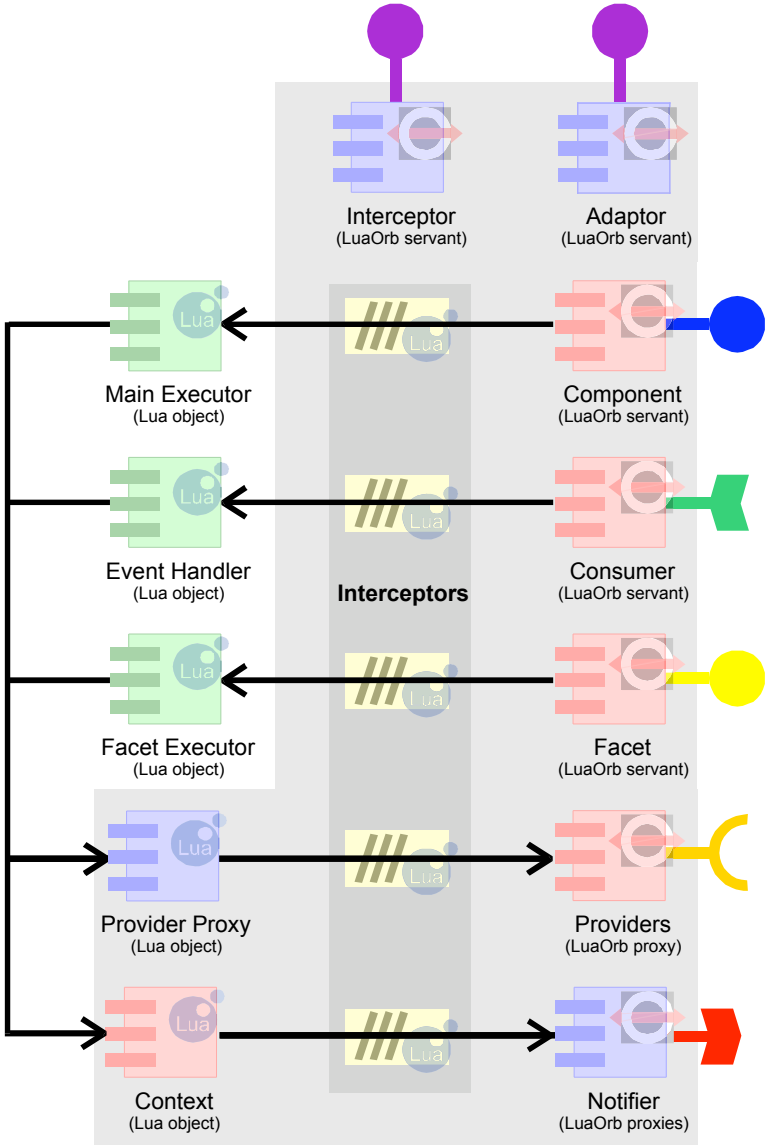

```
component MeuComponente
    supports InterfaceOferecida :
    SuperComponente
{
    attribute string atributo;
    provides Interface faceta;
    uses Interface receptaculo;
    publishes Evento publicador;
    emits Evento emissor;
    consumes Evento consumidor;
};

home MeuHome
    manages MeuComponente {};
```

Contêiner CCM

- Ambiente de execução protegido
  - Diferentes tipos de componentes.
  - Intermedia todas as interações.
- Simplifica a implementação
  - Gerencia instâncias
    - Criação e ativação.
  - Oferece serviços
    - Transação, persistência.
  - Gerencia conexões.



Contêiner Lua



Componentes LuaCCM

- Recursos reflexivos
  - Introspecção (interfaces do CCM)
    - Obter informações sobre portas.
    - Estabelecer e desfazer conexões em portas.
  - Intercessão (LuaCCM::Adaptable)
    - Adicionar e remover portas.
    - Substituir a implementação de portas.
    - Associar interceptadores a portas.
- Alterações em níveis de abrangência (mesmo servidor)
  - Por Definição (através do contêiner)
  - Por Home (através do home do componente)
  - Por Instância (através da instância do componente)

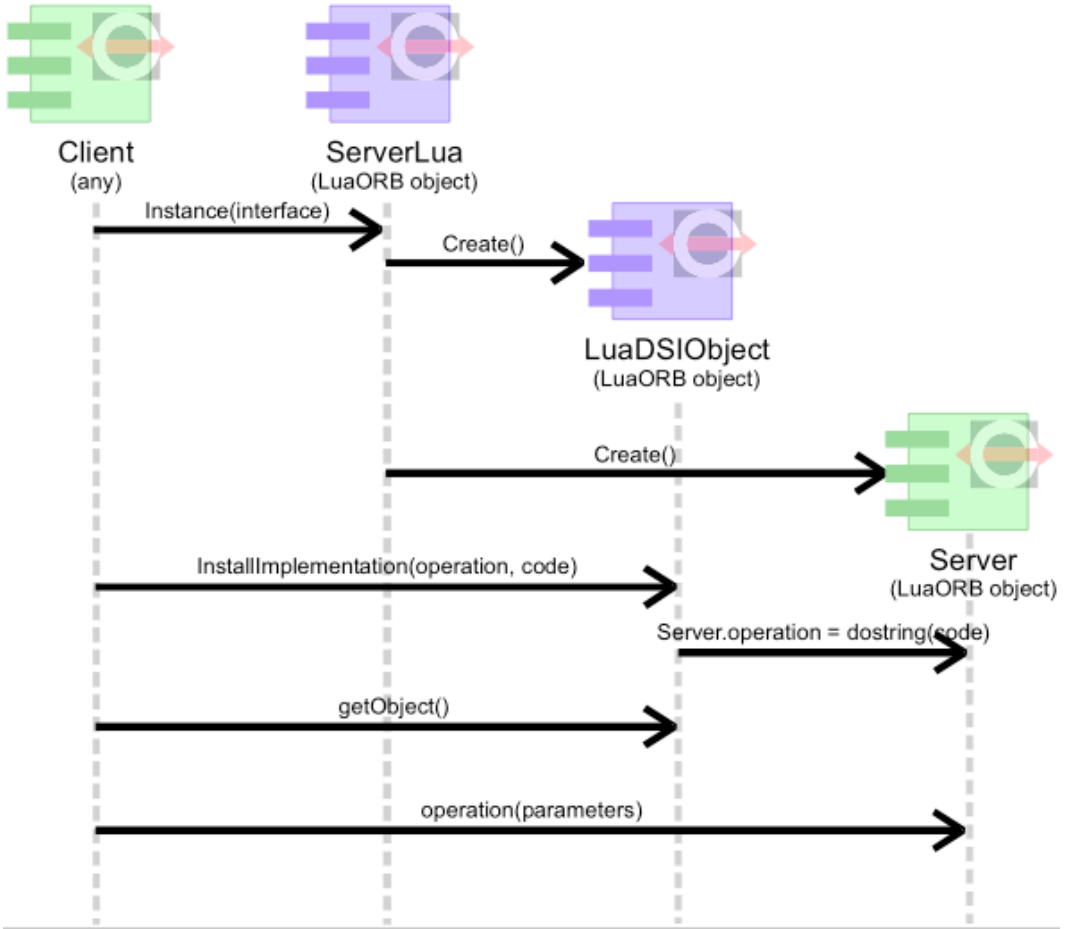
LuaOrb Adaptation Framework (LOAF)

- Conjunto de mecanismos voltados para adaptação dinâmica de aplicações
- Adaptação em ponto pequeno
  - Manipulação da estrutura de objetos/componentes
- Adaptação em ponto grande
  - Manipulação da estrutura da aplicação (reconfiguração dos relacionamentos entre objetos/componentes)

Interface ServerLua

```
interface LuaDSIObject {
   readonly attribute Object obj;
   void InstallImplementation(
       in string opname,
       in string luacode
   );
};
```

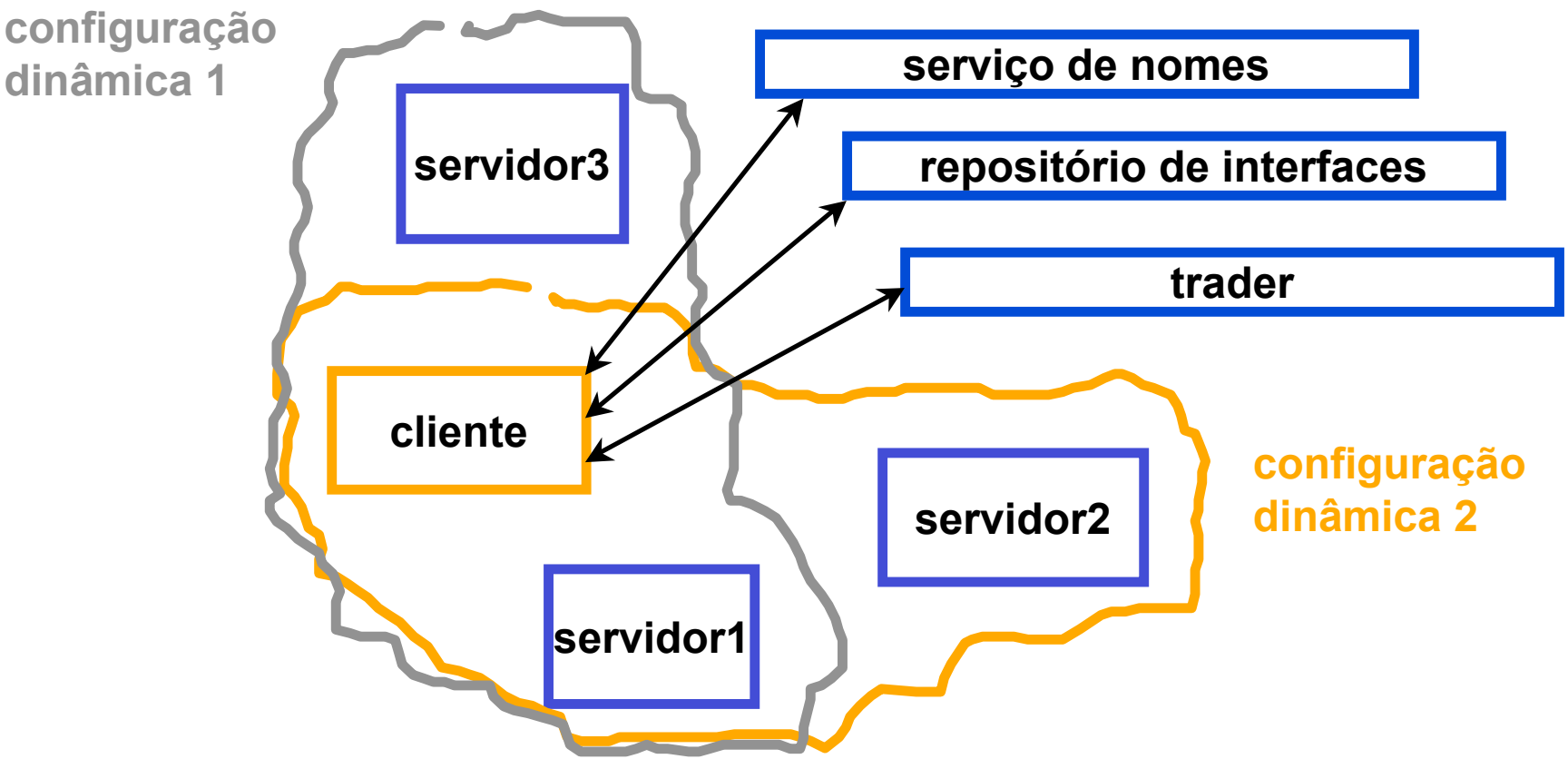
```
interface ServerLua {
   LuaDSIObject Instance(
       in CORBA::InterfaceDef intf
   );
};
```



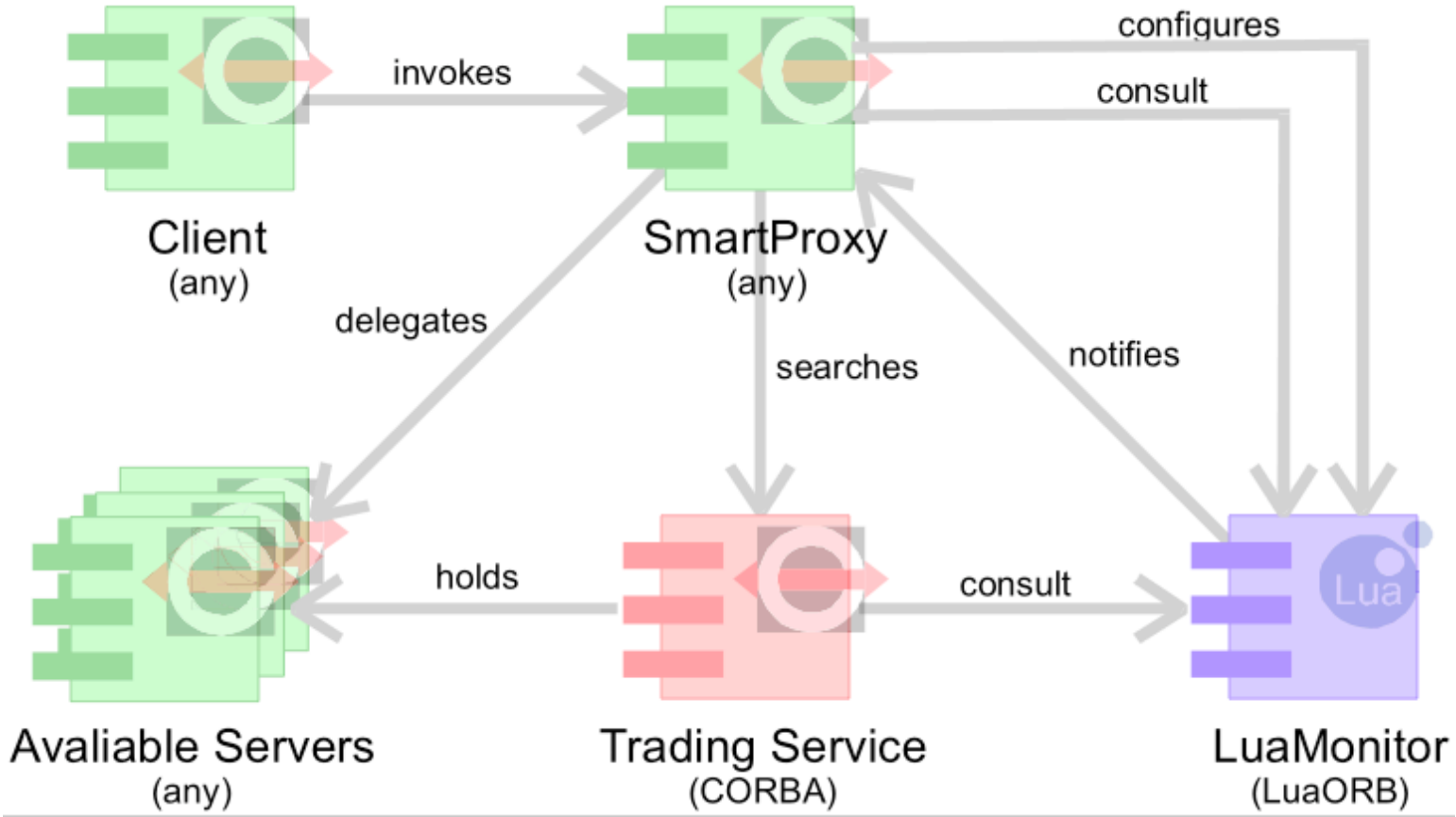
Interfaces de Adaptação do LuaCCM

```
interface Adaptable {
  void add_facet(in string name, in string iface, in LuaCode code);
  void add_receptacle(in string name, in string iface, in boolean
    ismultiple);
  ...
  void set_implementation(in string port, in LuaCode code);
  ...
  void intercept(in string point, in LuaCode code);
};

interface AdaptableContainer  :  Components::Container {
  Adaptable get_component_adaptor(in string absolute_name);
};
```

Reconfiguração



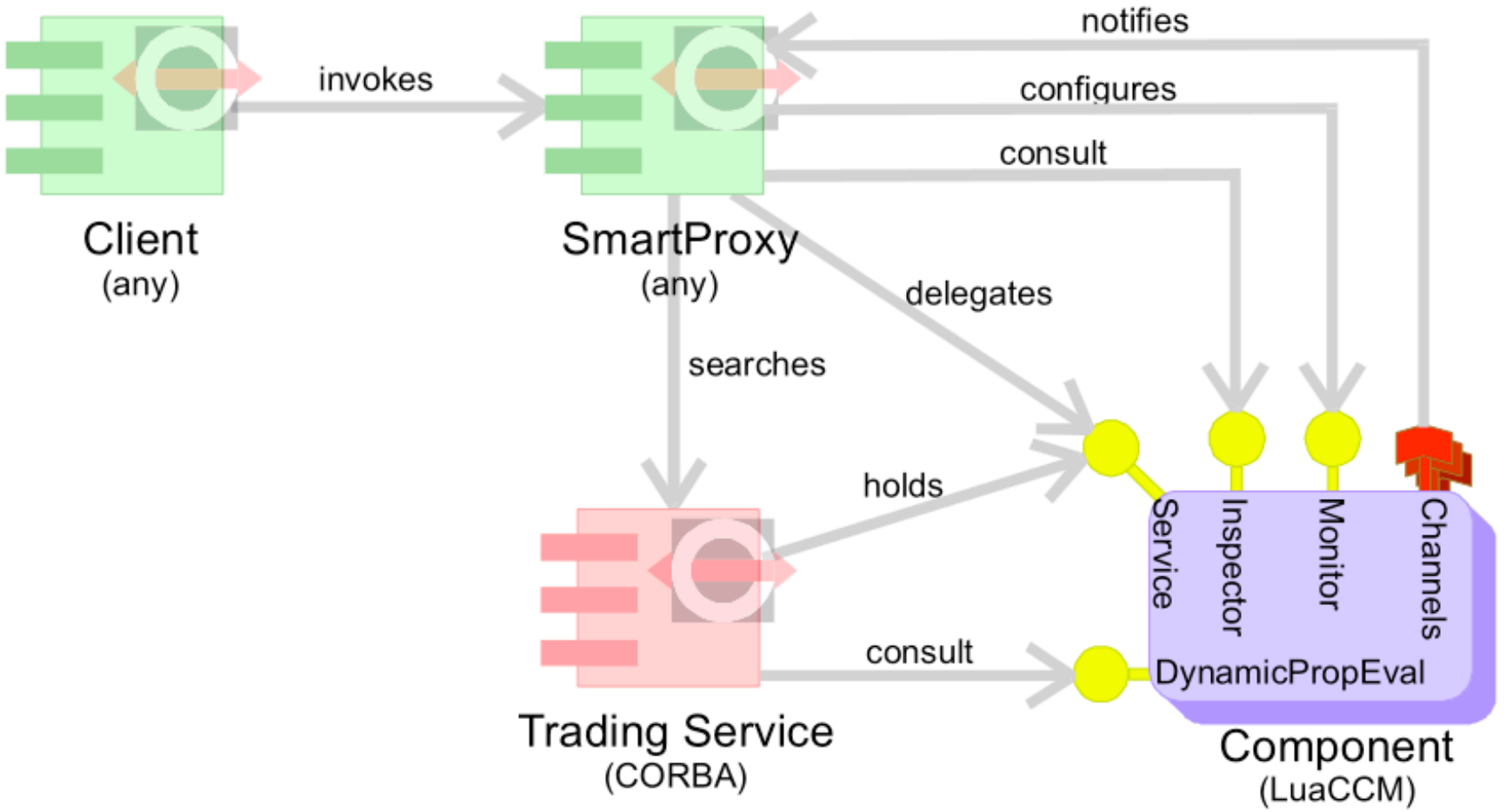
Smart Proxies



LuaMonitor

```
interface BasicMonitor {
   PropertyValue getValue(in string name);
   void setValue(in string name, in PropertyValue property);
};

interface EventObserver {
   oneway void notifyEvent(in EventID evID);
};

interface EventMonitor : BasicMonitor {
   EventObserverID attachEventObserver (in EventObserver obj,
                           in EventID evID, in LuaCode notifyf);
   void detachEventObserver(in EventObserverID id);
};

interface AspectsManager {
   PropertyValue getAspectValue(in Aspectname name);
   AspectList definedAspects();
   void defineAspect(in AspectName name, in LuaCode updatef);
};
```

Smart Proxies (versão LuaCCM)

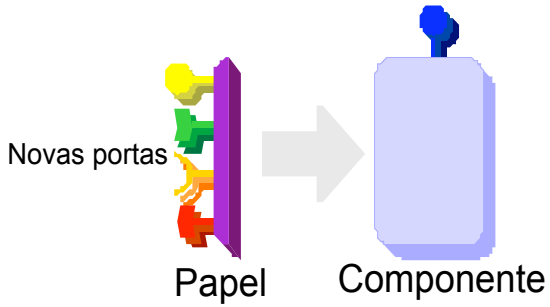


Reconfiguração dinâmica

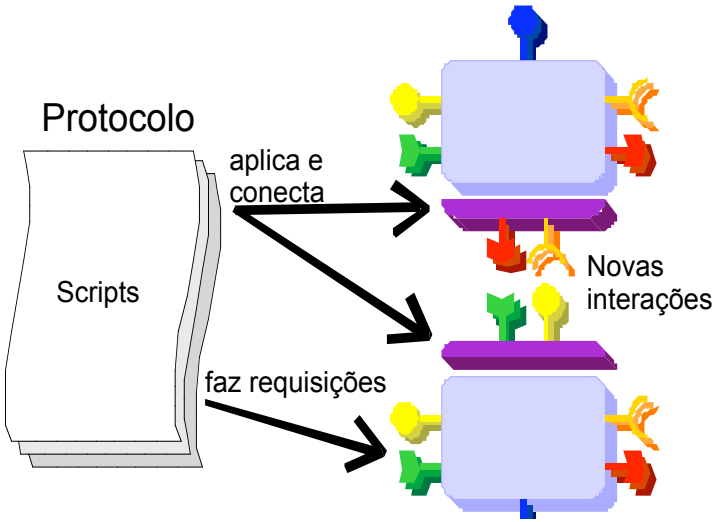
- Sobre a base que temos, que outras abstrações podem ser úteis?
  - tolerância a falhas
  - balanceamento de carga
  - qualidade de serviço
- Como especificar reconfiguração?
  - no nível do proxy
    - programador avançado
  - no nível do programa
    - descrições de condições e ações de reconfiguração
    - programador da aplicação

Exemplo de Abstrações: Papéis e Protocolos

- Papel
  - Conjunto de alterações no componente para realizar uma nova função (papel)

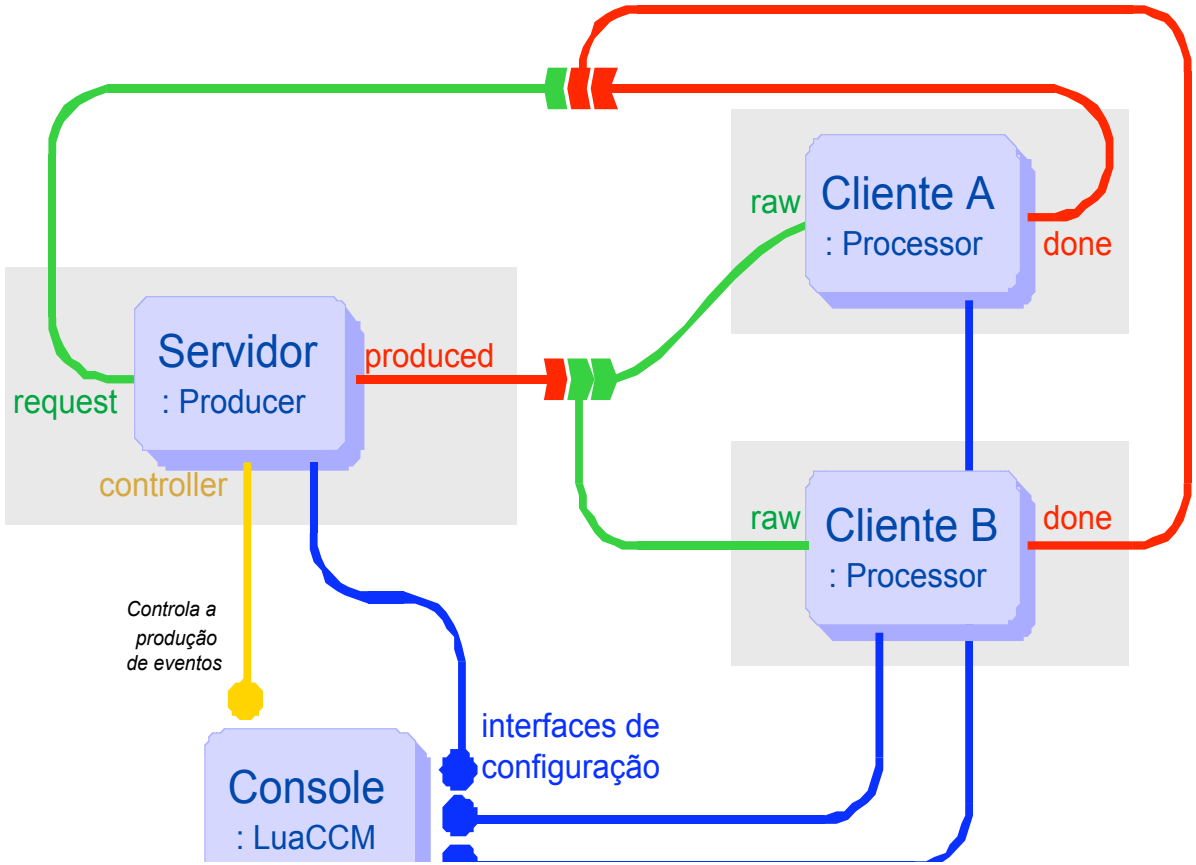


- Protocolo
  - Regras de como os papéis são aplicados e os componentes são reconectados para fornecer a nova funcionalidade.



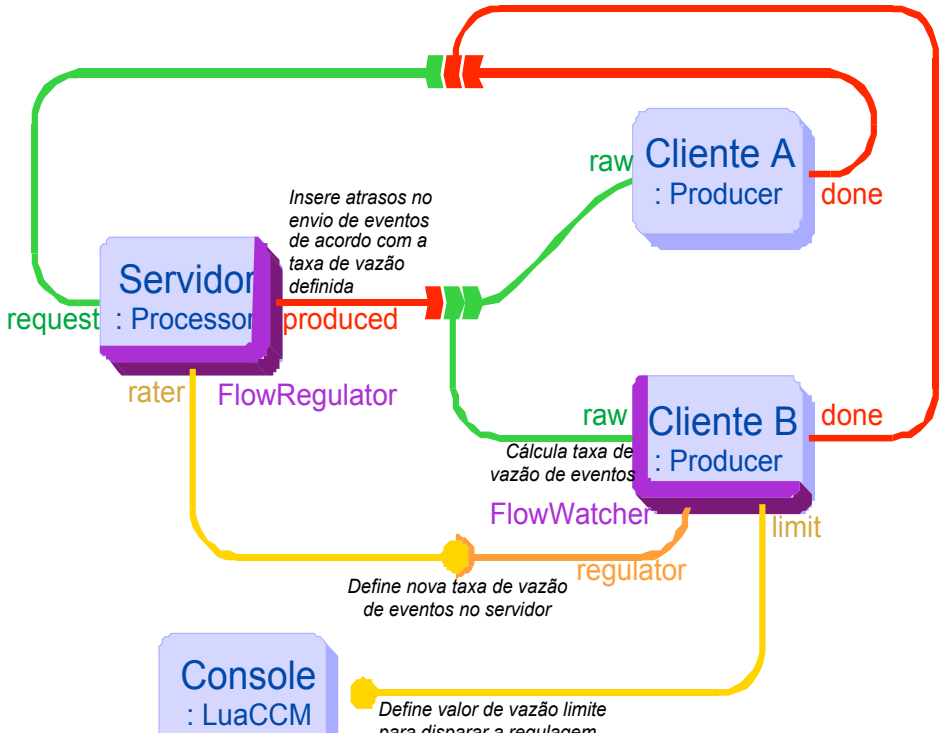
Aplicação de Exemplo

```
server = producers:create()
client1 = processorsA:create()
client2 = processorsB:create()

server.name  = "Servidor"
client1.name = "Cliente A"
client2.name = "Cliente B"

client1.raw = server.produced
client2.raw = server.produced
server.request = client1.done
server.request = client2.done

server.controller:start()
```



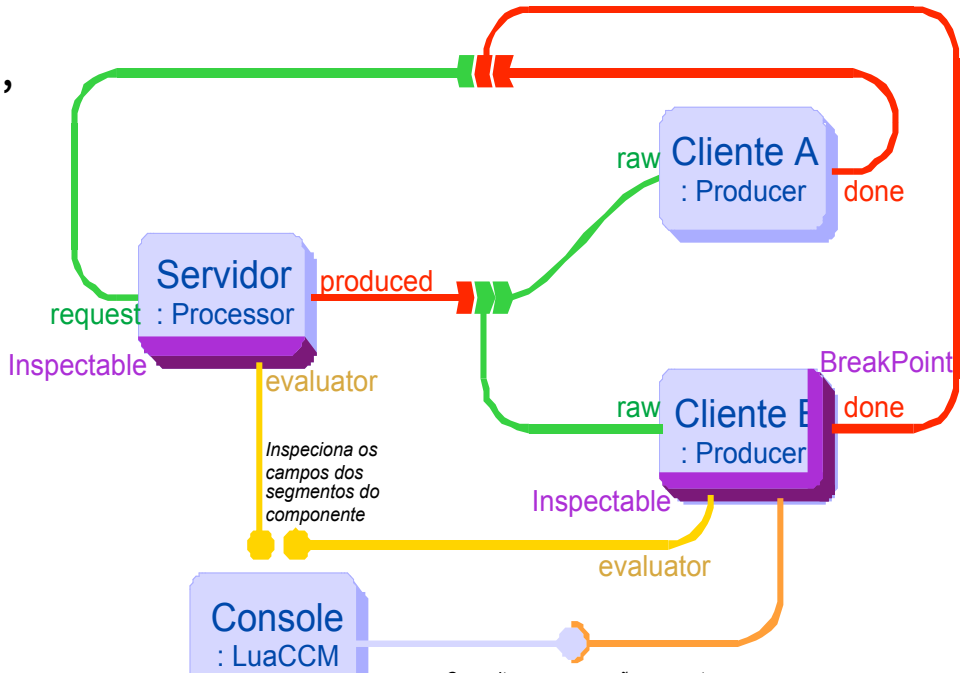
Sincronização de Fluxo

```
FlowWatcher = loaf.Role {
  before = {
    raw = { code = [[...]] }},
  after = {
    raw = { code = [[...]] }},
  provides = {
    limit = {
      interface = "Limited",
      code = [[...]]}},
  uses = {
    regulator = { interface =
    "Rateable"}}}

FlowRegulator = loaf.Role {
  before = {
    produced = { code = [[...]]
    }},
  provides = {
    rater = {
      interface = "Rateable",
      code = [[...]]}}}
```



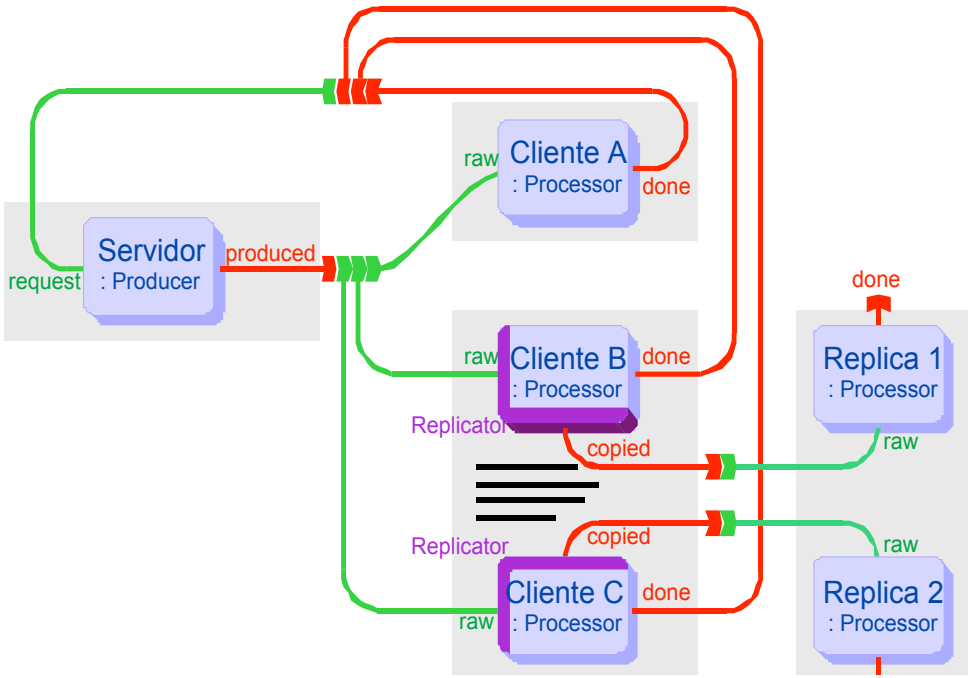
Depuração Distribuída

```
BreakPoint = loaf.Role {
  uses = {
    pauser = { interface = "Pauser" }},
  before = {
    done = {
      code = [[
function(self, request)
  local c = request.context
  local p = c:get_connection_pauser()
  if p then
    while p:locked() do
      sleep(1)
    end
  end
end
      ]]}}}
```



Replicação Passiva

```
adaptor =
  container:get_component_adaptor(
    "Processor“
  )

Replicator:assign(adaptor)

replica1 =
  loaf.handler(replicas:create())
replica2 =
  loaf.handler(replicas:create())

replica1.name = "Replica 1"
replica2.name = "Replica 2"
replica1.raw = client2.copied
replica2.raw = client3.copied
```



Trabalhos em andamento

- O^2
  - linguagens interpretadas na implementação de ORBs
  - adaptação dinâmica do próprio middleware
  - uso em dispositivos móveis
  - modelo de concorrência cooperativa (corrotinas)

- LuaCCM
  - abstrações para desenvolvimento de sistemas baseados em componentes
  - LuaOrb Adaptation Framework
  - suporte a sistemas baseados em agentes

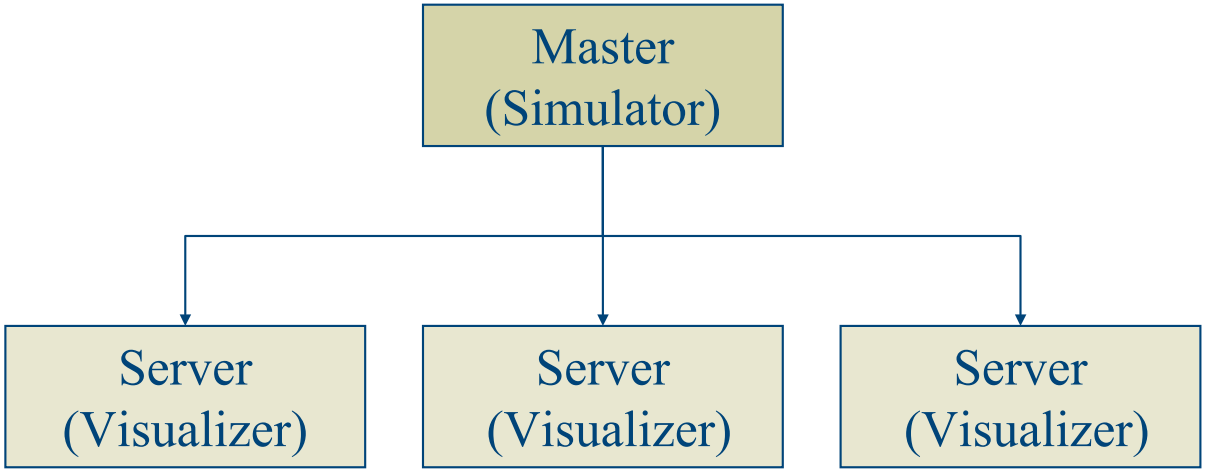
Trabalhos em andamento (cont.)

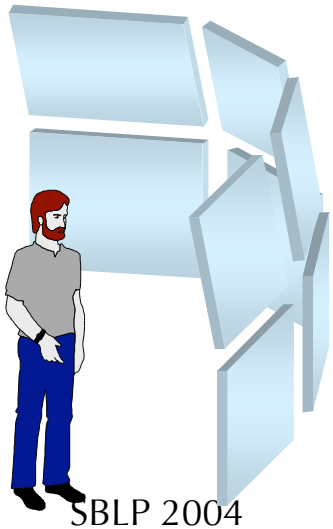
- Middleware para computação em grade e computação ubíqua
  - definição de uma arquitetura que atenda a esses domínios de aplicações
  - suporte a colaboração síncrona e assíncrona de usuários
  - sistema de arquivos distribuído
  - mecanismos de segurança
- Adaptação e mobilidade
  - geração "automática" de proxies inteligentes
  - integração de LuaCCM com monitoramento: políticas de adaptação
  - abstrações adequadas

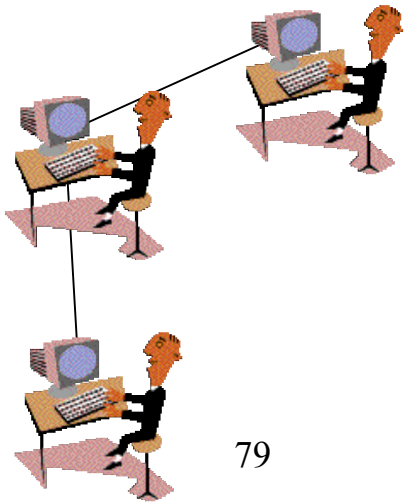
Algumas aplicações desenvolvidas

- Visualização distribuída
- Apresentações distribuídas
- Computação móvel
- Módulo de Procedimentos Automatizados
- Gaia

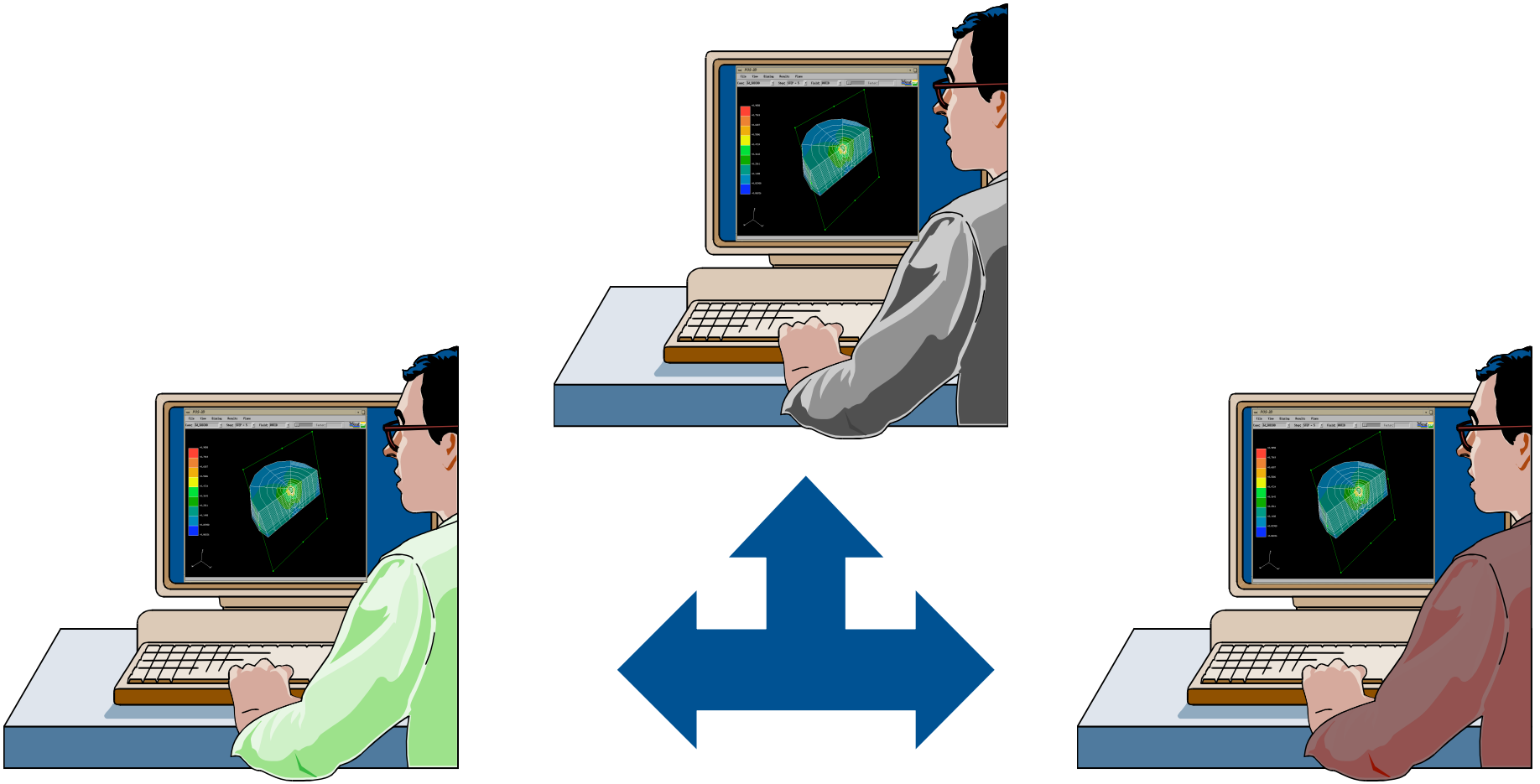
Vis - Visualização Distribuída



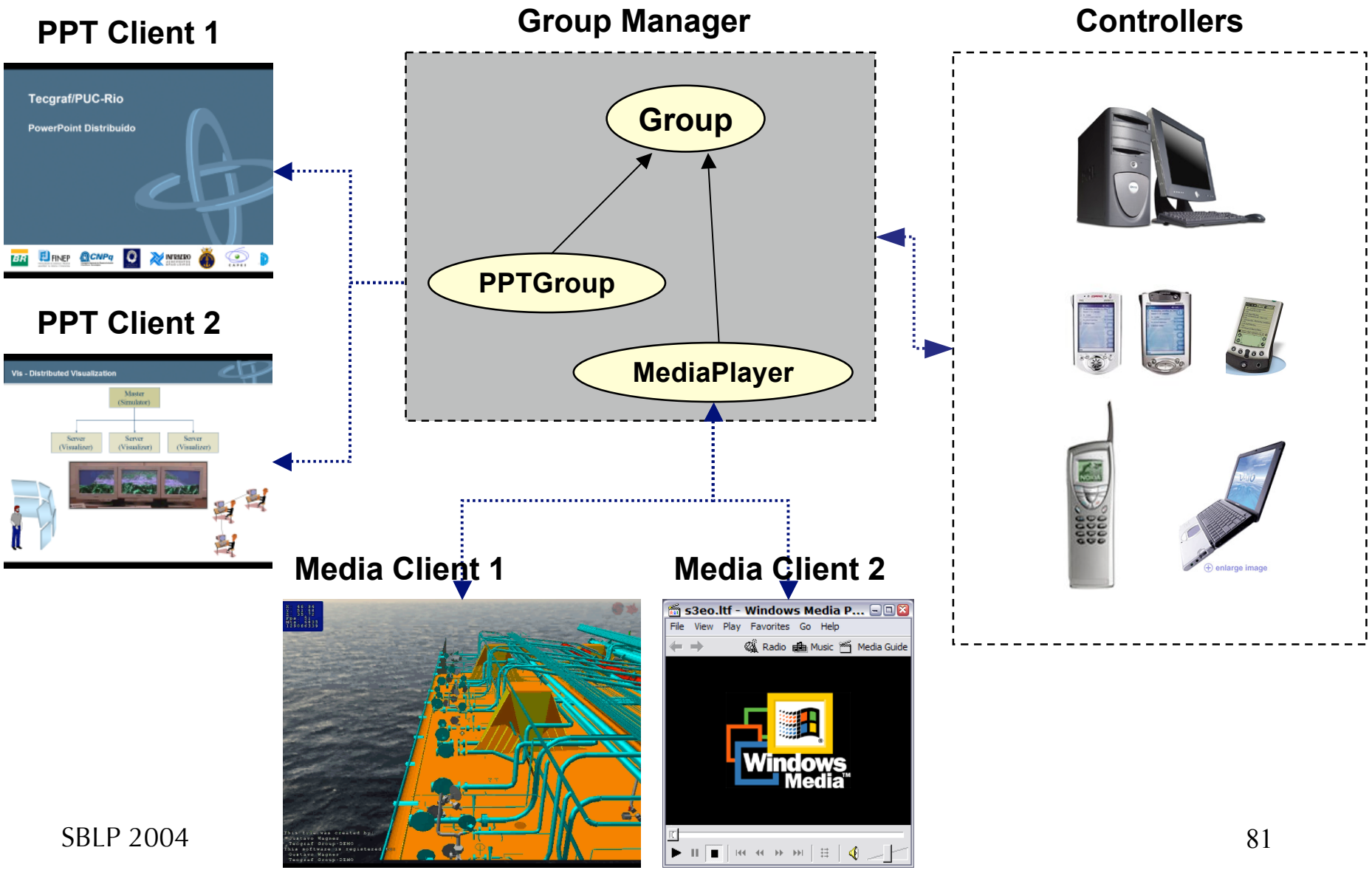




Vis - Colaboração Distribuída



Apresentações Distribuídas

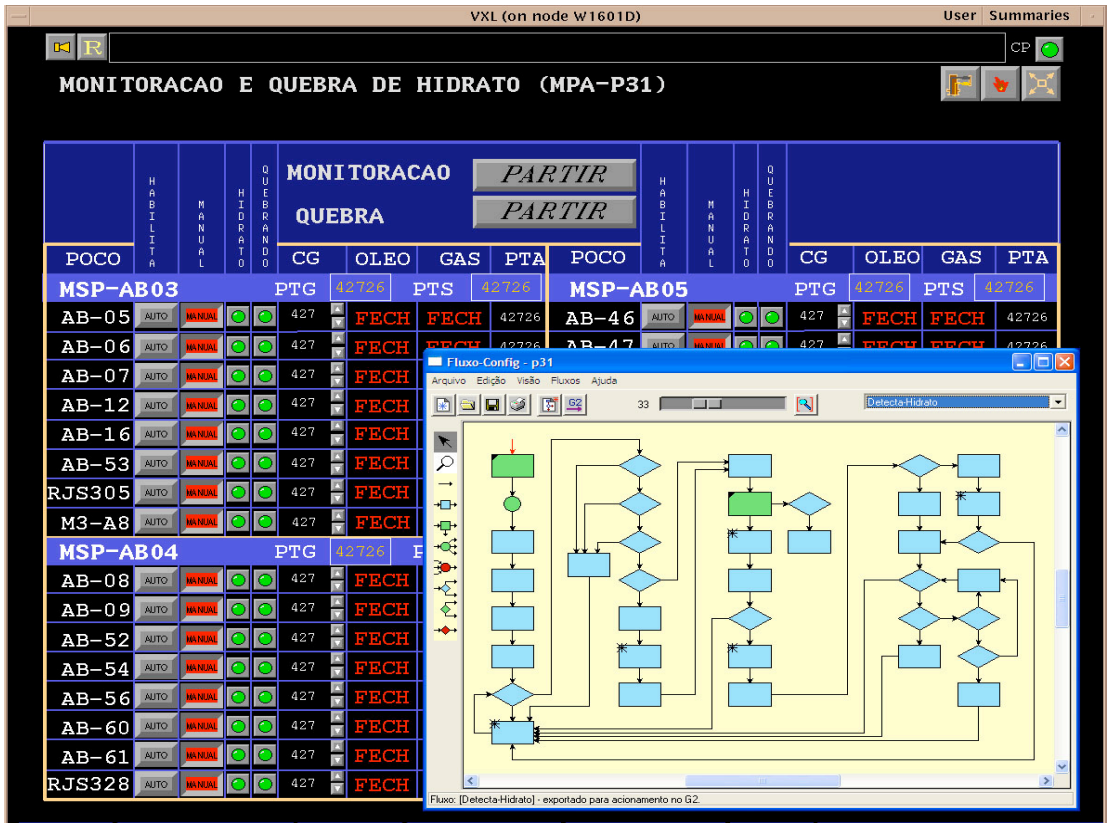


Computação Móvel - Controlador PPT

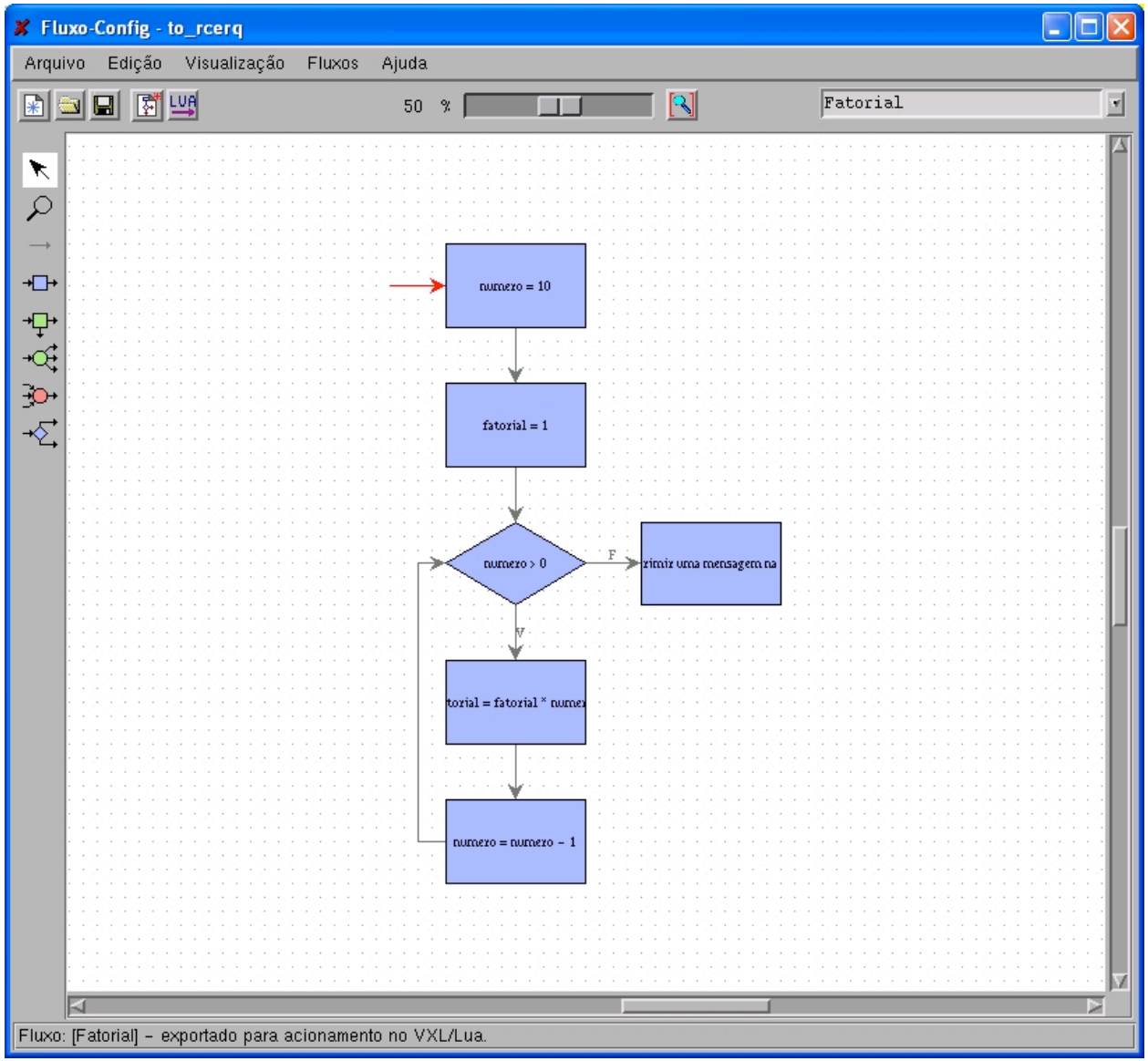
Computação Móvel - Mensagens instantâneas

Módulo de Procedimentos Automatizados (MPA)

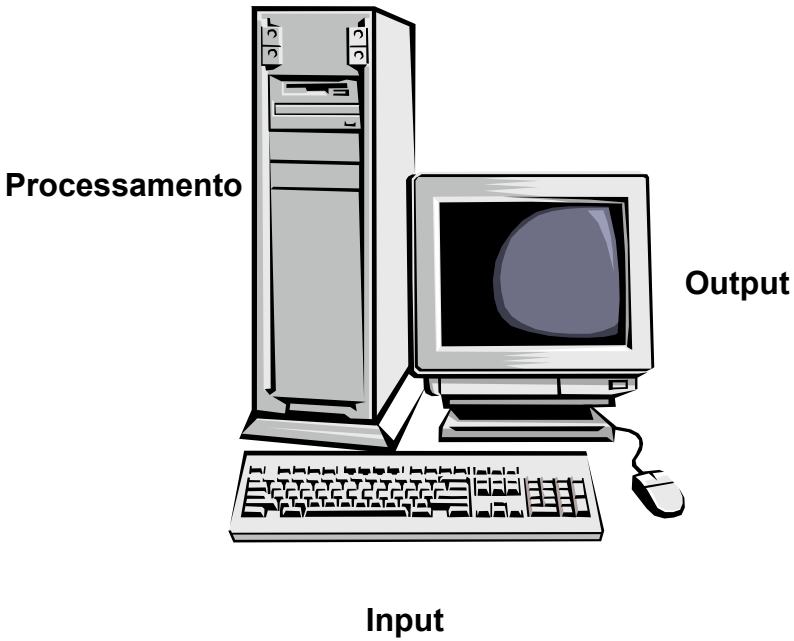
- Instalado na plataforma P-31



MPA - Editor de fluxos



Gaia - uma infra-estrutura de software para salas inteligentes

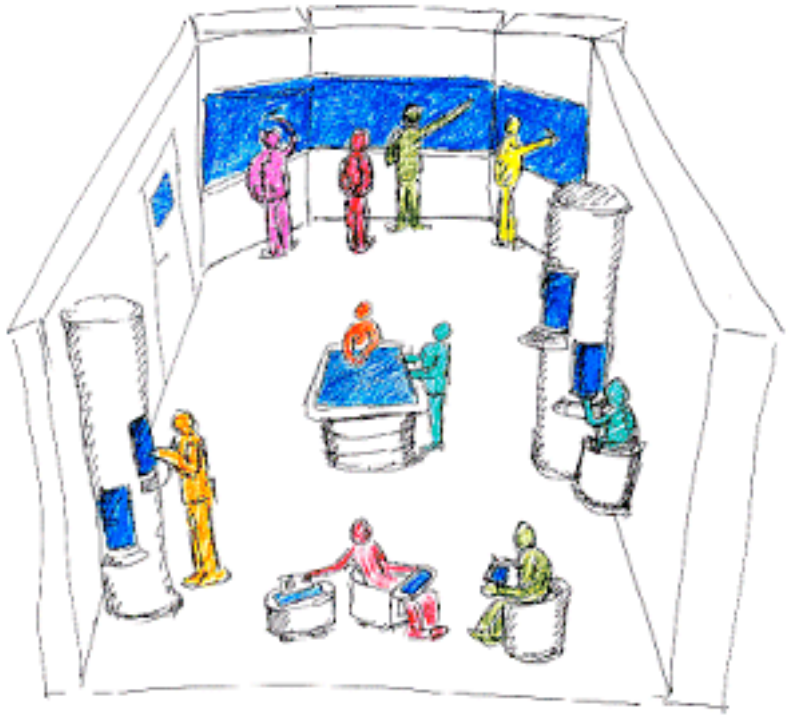


O computador que conhecemos

O novo computador

Aplicações para Espaços Ativos

- Devem fazer uso de todos os recursos disponíveis para dar apoio às atividades de seus usuários.
- Devem poder se adaptar de um espaço para outro.
- Podem ter a sua disposição um amplo e variado conjunto de dispositivos de entrada e saída.
- Podem precisar se adaptar dinamicamente a mudanças em seus ambientes de execução (tanto lógico quanto físico).



Gaia - e a nova rede…

Mais informações

http://www.tecgraf.puc-rio.br/luaorb
http://www.lua.org

rcerq@inf.puc-rio.br