Utilização de reflexão computacional para implementação de aspectos não funcionais em um gerenciador de arquivos distribuídos

Fabrício Jailson Barth

Orientador: Maurício Capobianco Lopes

Universidade Regional de Blumenau

Blumenau, Julho de 2000

Roteiro da apresentação:

- Introdução
- Reflexão Computacional
- Ferramentas reflexivas para a linguagem Java
- Gerenciador de arquivos distribuídos
- Modelagem e implementação
- Conclusões e sugestões
- Referências bibliográficas

Introdução

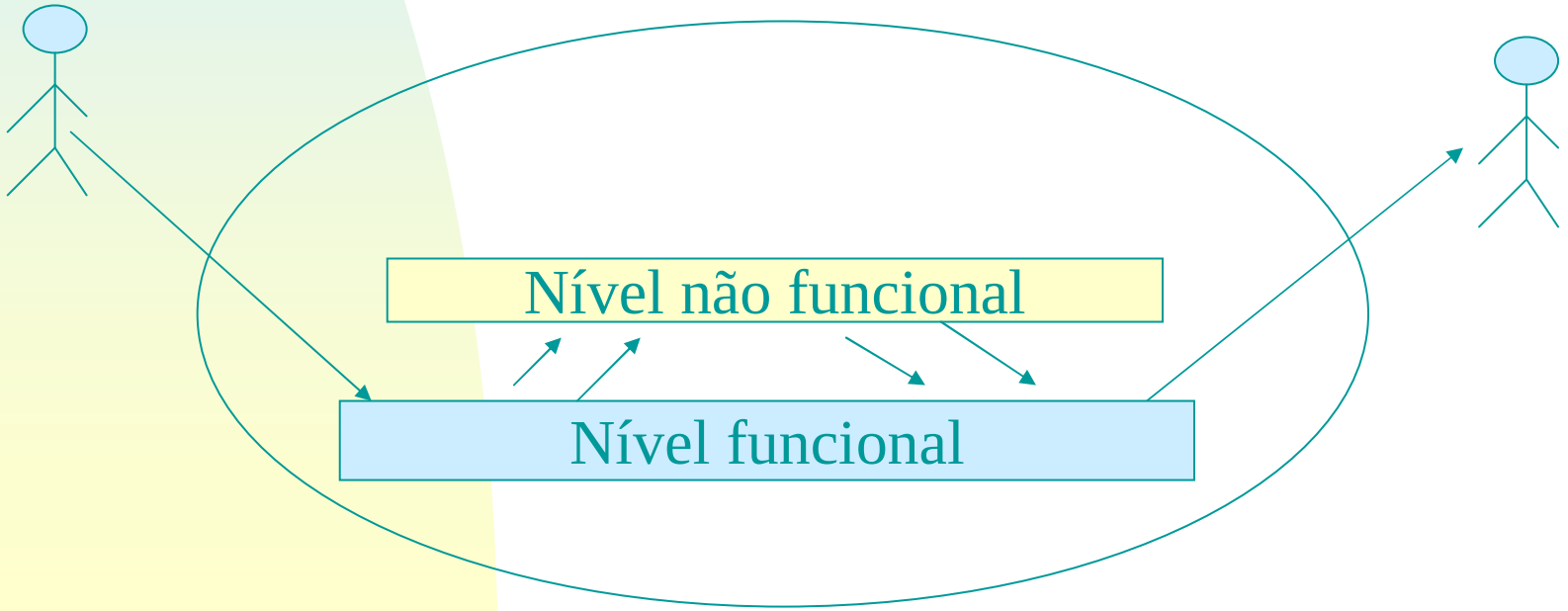
- **Características de sistemas modernos**
- **Separação de funcionalidades** [KIC1991]
  - Aspectos funcionais
  - Aspectos não funcionais
- **Reflexão Computacional / Meta-níveis / Meta protocolos**
- **Problema:** adaptação de requisitos não funcionais a um gerenciador de arquivos distribuídos
- **Conceitos características e vantagens/desvantagens**
- **Redução de complexidade, adaptabilidade e reutilização**

Reflexão Computacional

- Toda atividade de um sistema computacional realizada sobre si mesmo, e de forma separada das computações em curso, com o objetivo de resolver seus próprios problemas [LIS1997]
- Capacidade de um sistema computacional de interromper o processo de execução, realizar computações ou fazer deduções no meta-nível e retornar ao nível de execução traduzindo o impacto das decisões [STE1994]

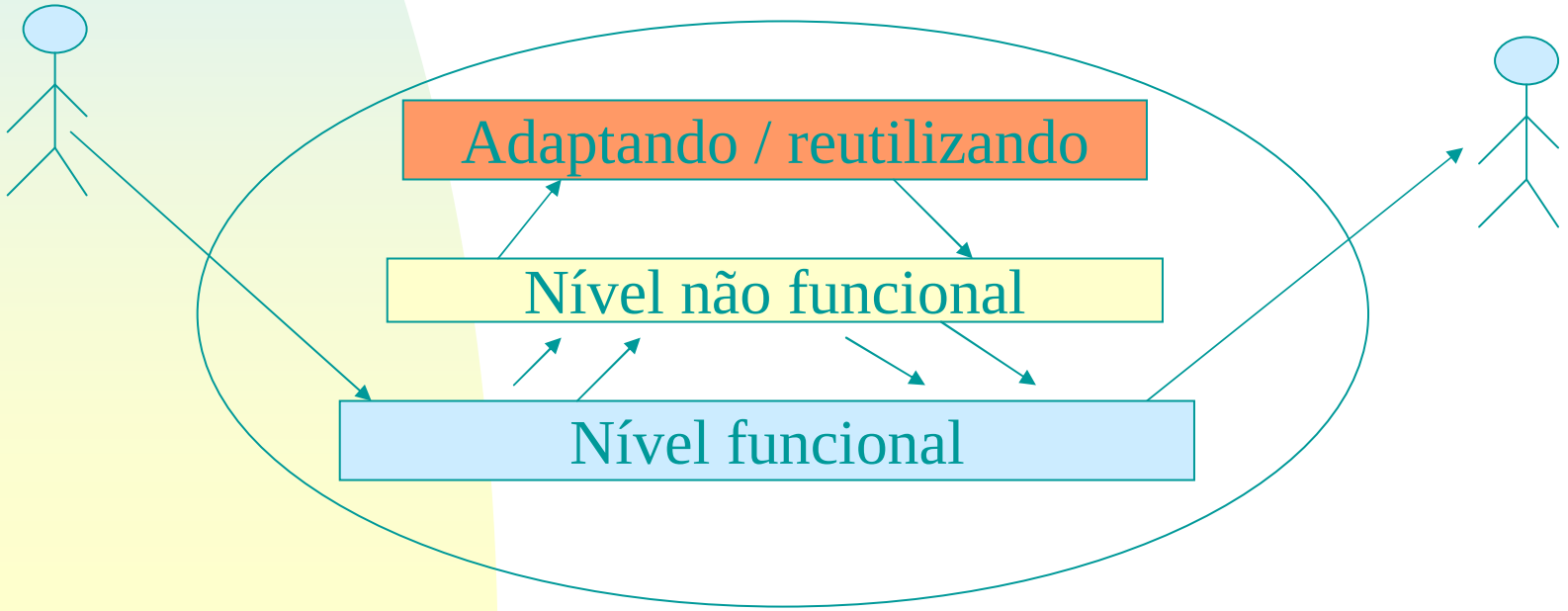
Reflexão Computacional

- Idéia básica do paradigma
- Nova arquitetura de software



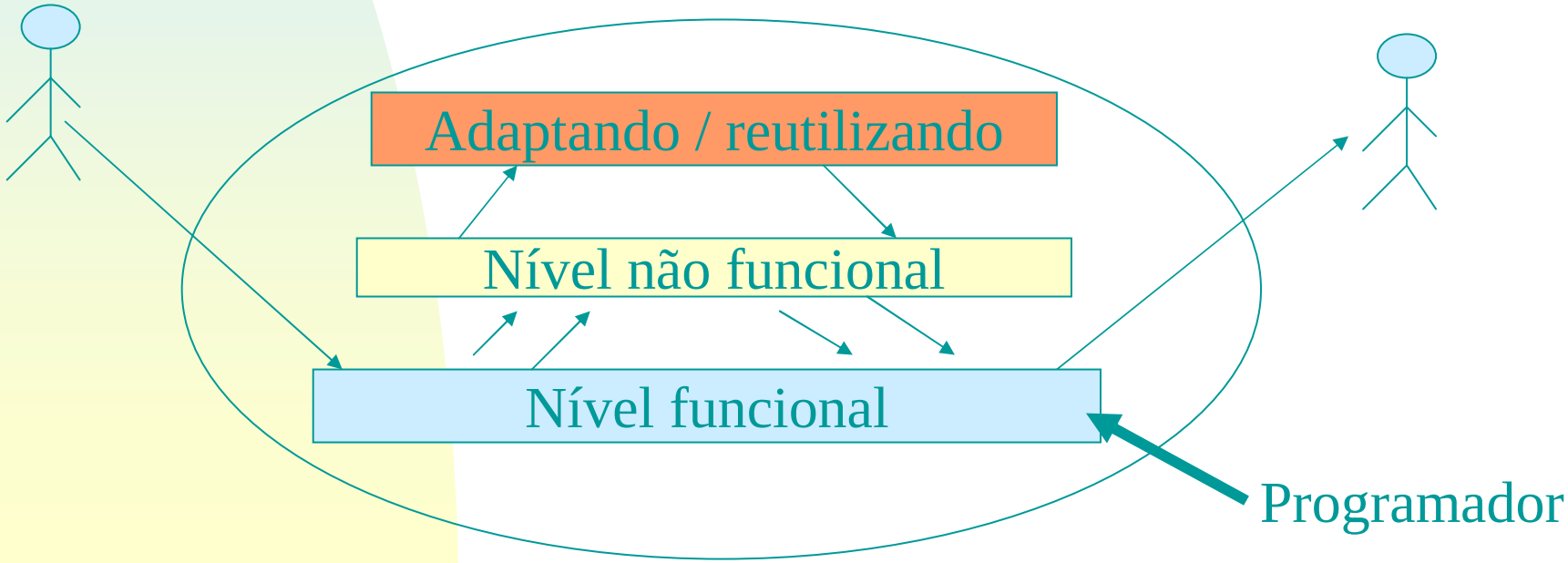
Reflexão Computacional

- Idéia básica do paradigma
- Nova arquitetura de software



Reflexão Computacional

- Idéia básica do paradigma
- Nova arquitetura de software

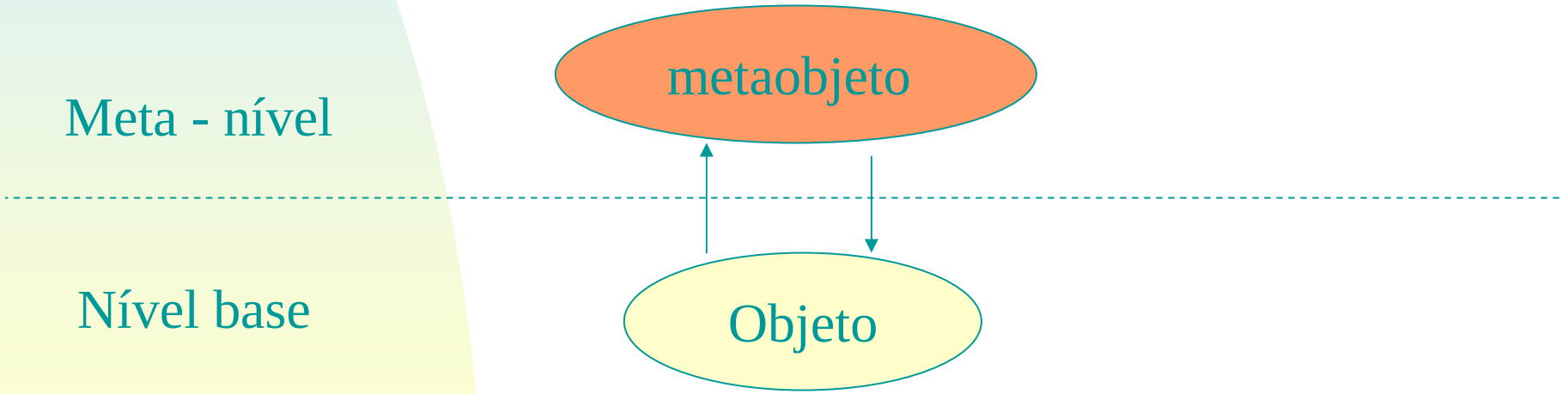


Reflexão Computacional

- **Idéia básica sobre reflexão computacional** [WU1997]:
  - separar as funcionalidades básicas das não básicas através de níveis arquiteturais;
  - as funcionalidades básicas devem ser satisfeitas pelos componentes da aplicação;
  - as não básicas dever ser satisfeitas pelos meta componentes;
  - as capacidades não funcionais são adicionadas aos componentes da aplicação através de seus metaobjetos específicos;
  - o componente base pode ter a sua estrutura e comportamento alterados em tempo de execução, ou de compilação.

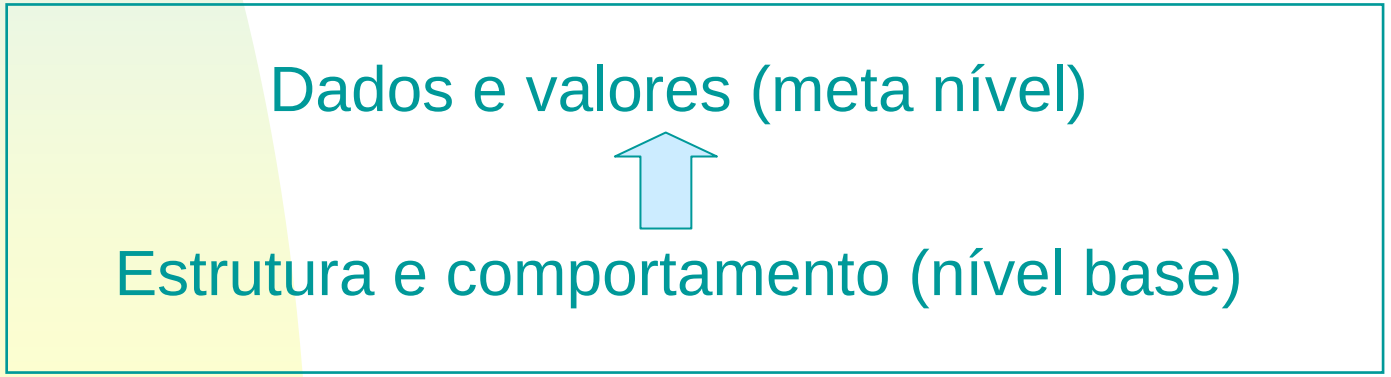
Reflexão computacional no modelo de objetos

- Interpretador de uma linguagem para a própria linguagem
- Abstração do modelo de objeto



Nível base / Meta nível

- **Informações**:
  - classes de um objeto;
  - herança;
  - propriedades;
  - comportamento.

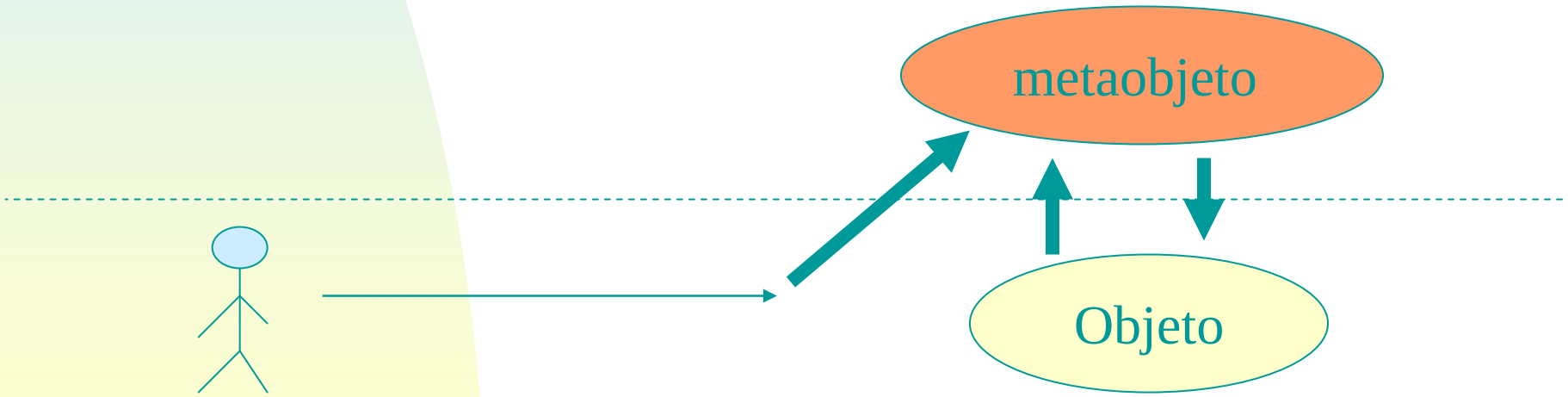


Consolidação de um sistema reflexivo

- **Um sistema reflexivo consolida-se depois de completado três estágios**:

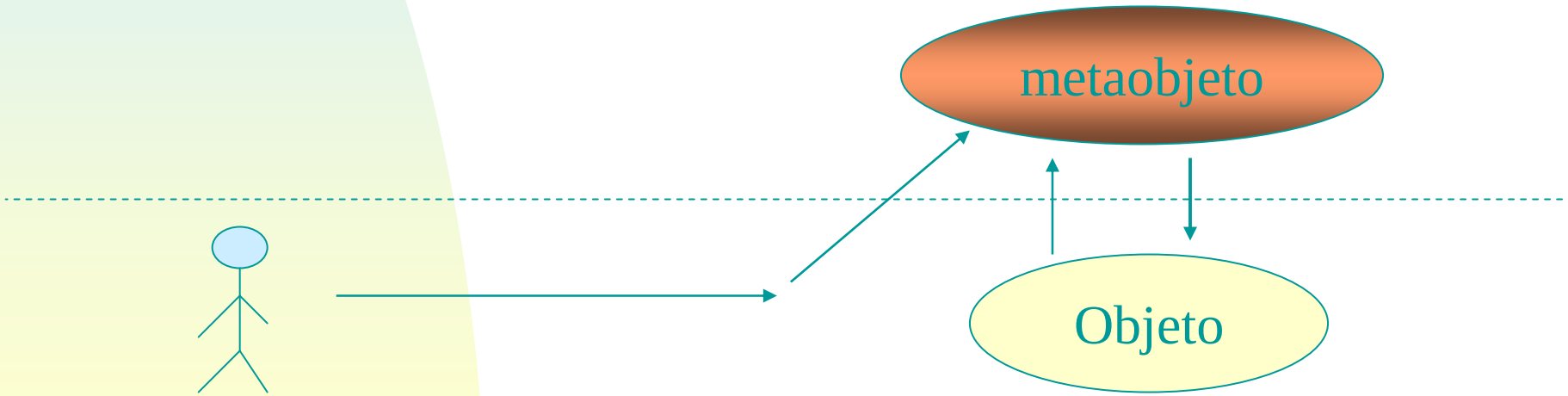
Consolidação de um sistema reflexivo

- Obter uma descrição abstrata do sistema tornando-a suficientemente concreta para permitir operações sobre ela;



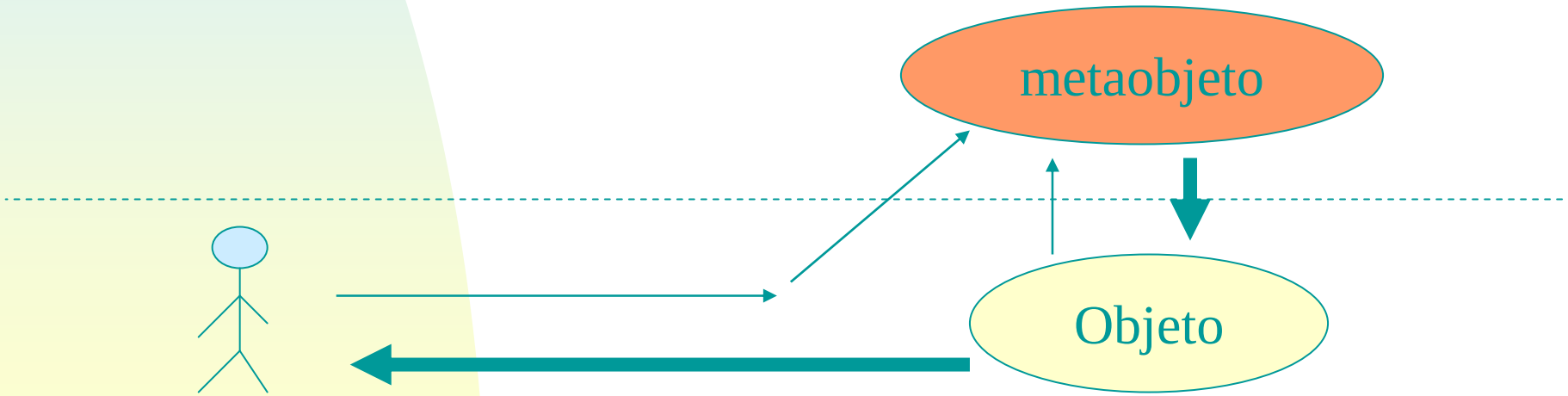
Consolidação de um sistema reflexivo

- Utilizar esta descrição concreta para realizar alguma manipulação;



Consolidação de um sistema reflexivo

- Modificar a descrição obtida com os resultados da reflexão computacional, retornando a descrição modificada ao sistema.

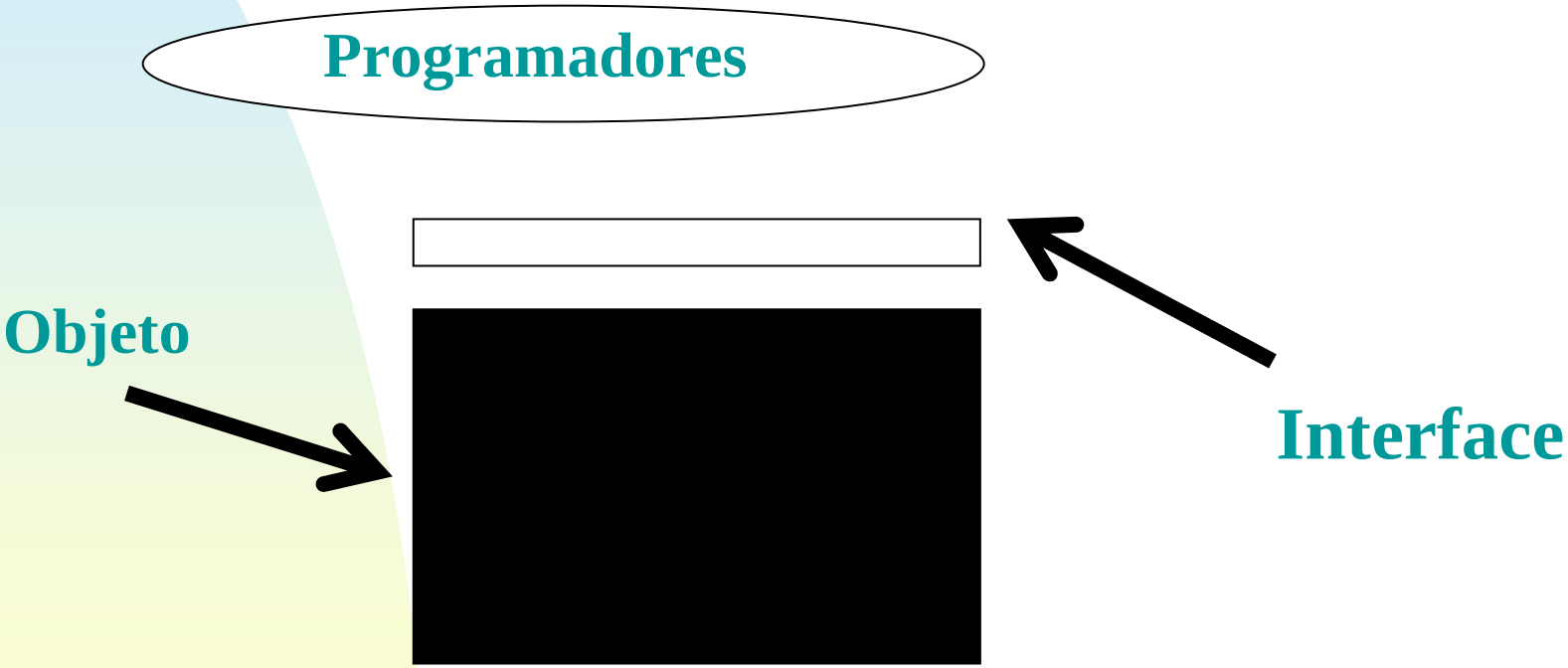


Protocolos de metaobjetos (MOP)

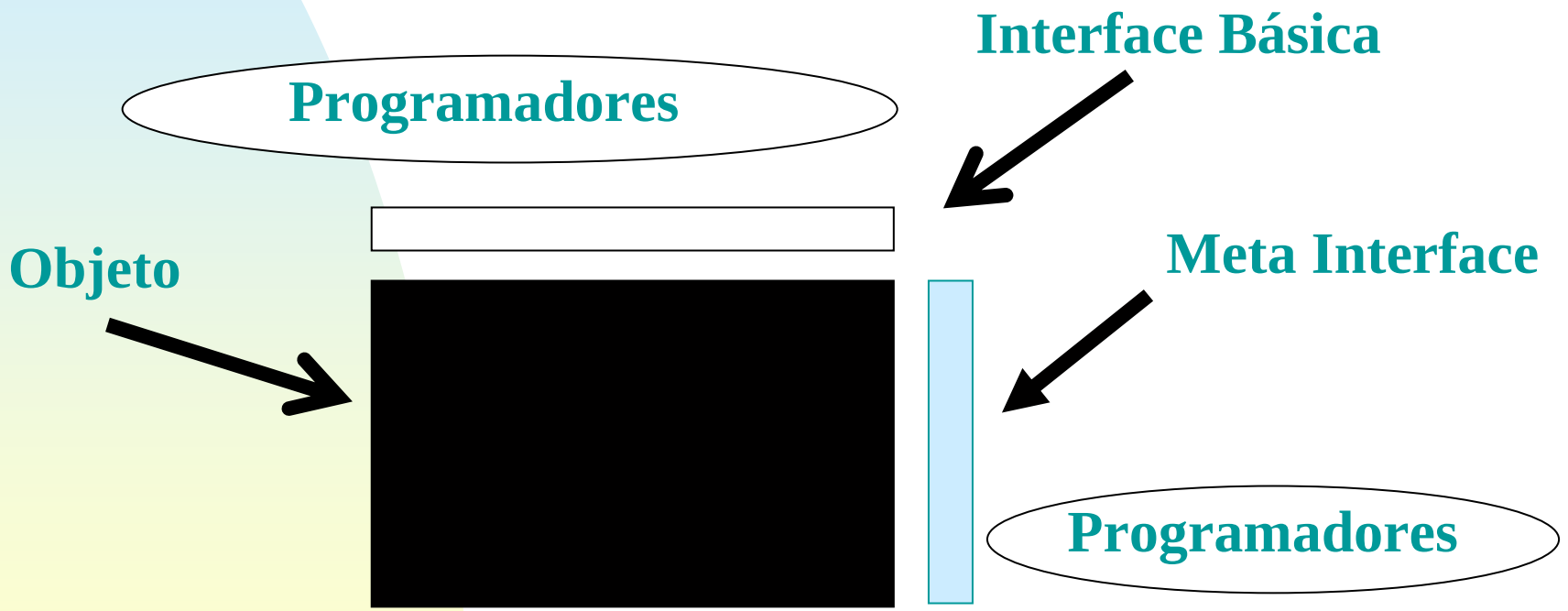
- Mecanismo responsável por realizar a comunicação entre o nível base e o meta-nível

- Preocupações:
  - quais entidades devem ser transformadas em algo que possa sofrer operações no meta-nível ?
  - como o relacionamento entre o nível base e o meta-nível é implementado ?
  - quando o sistema passa para o meta-nível ?

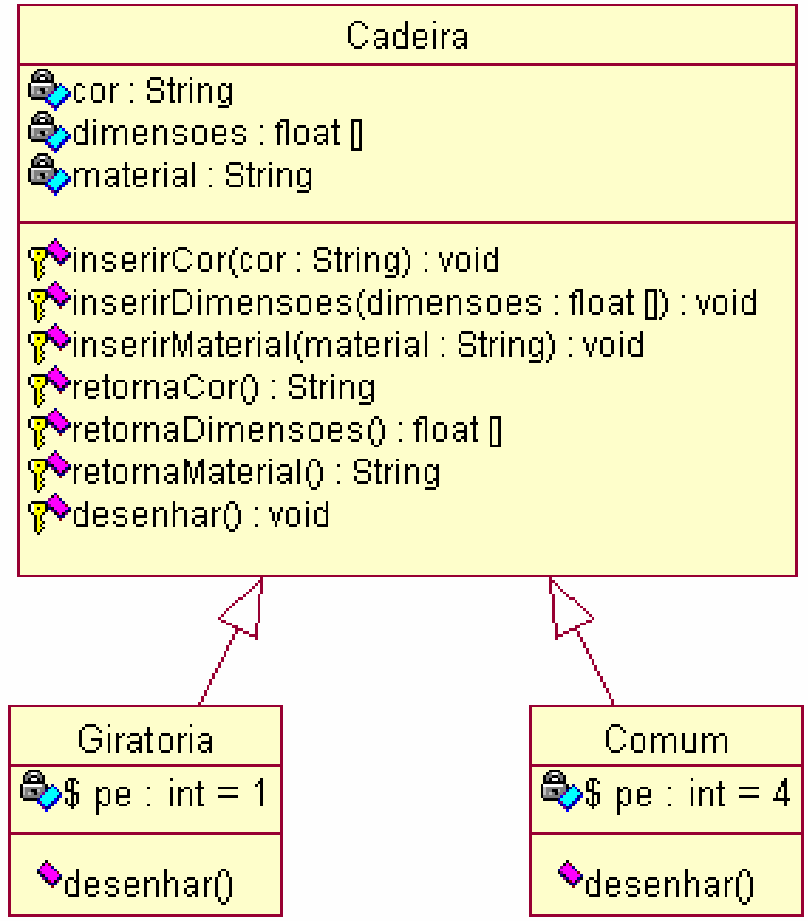
Protocolos de metaobjetos - objetos fechados



Protocolos de metaobjetos - objetos abertos



Exemplo de objeto fechado



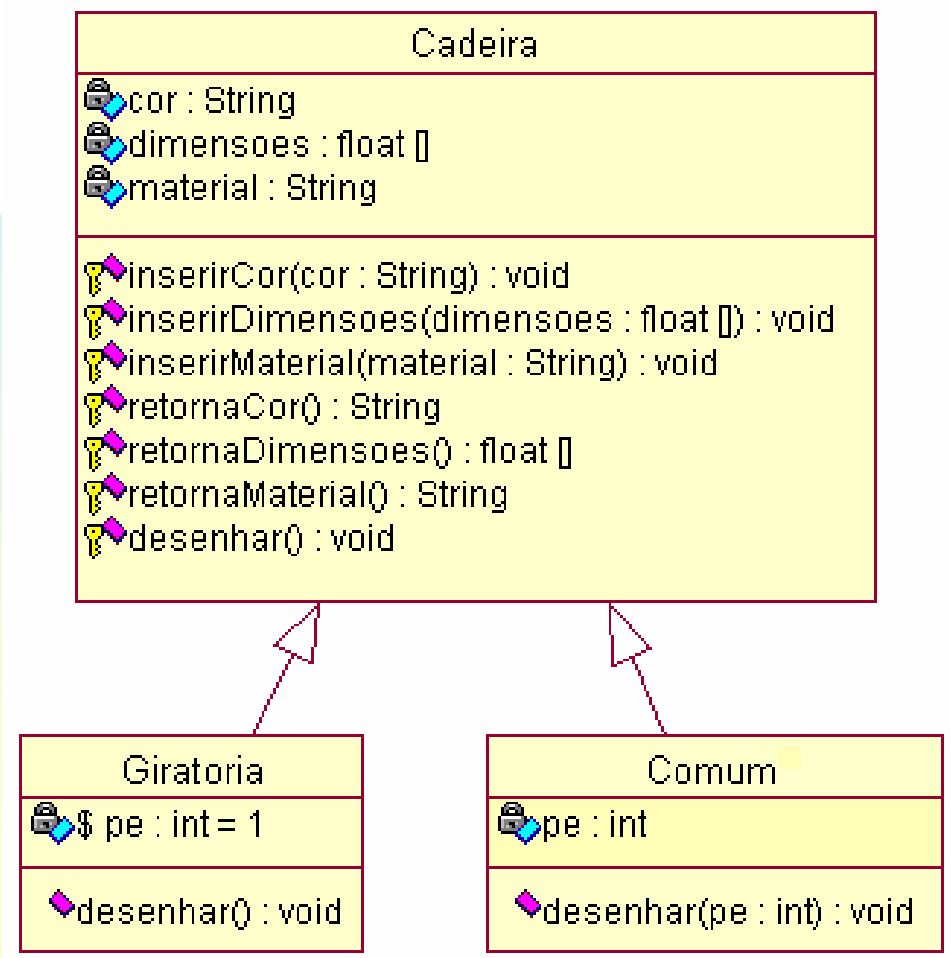
Meta interface de uma linguagem hipotética

Método	Descrição
Classe obtemClasse (String classe)	Através do nome da classe o método retorna um objeto representando a classe
Classe [] retornaSuperClasse ()	Método que retorna um vetor de super classes da classe alvo.
Classe [] retornaSubClasse ()	Método que retorna um vetor de sub classes da classe alvo
Metodo [] retornaMetodos ()	Método que retorna um vetor de todos os métodos da classe alvo
Atributo [] retornaAtributo ()	Método que retorna um vetor de todos os atributos da classe alvo.
void alteraEstadoAtributo (Atributo atributo)	Método que permite alterar o estado (privado, público ou protegido) do atributo listado como parâmetro
void alteraNomeClasse ()	Método que permite alterar o nome da classe alvo.
void adicionaCorpoMetodo (String codigo, int linha)	Método que permite alterar o comportamento de um determinado método, adicionando determinada linha de código

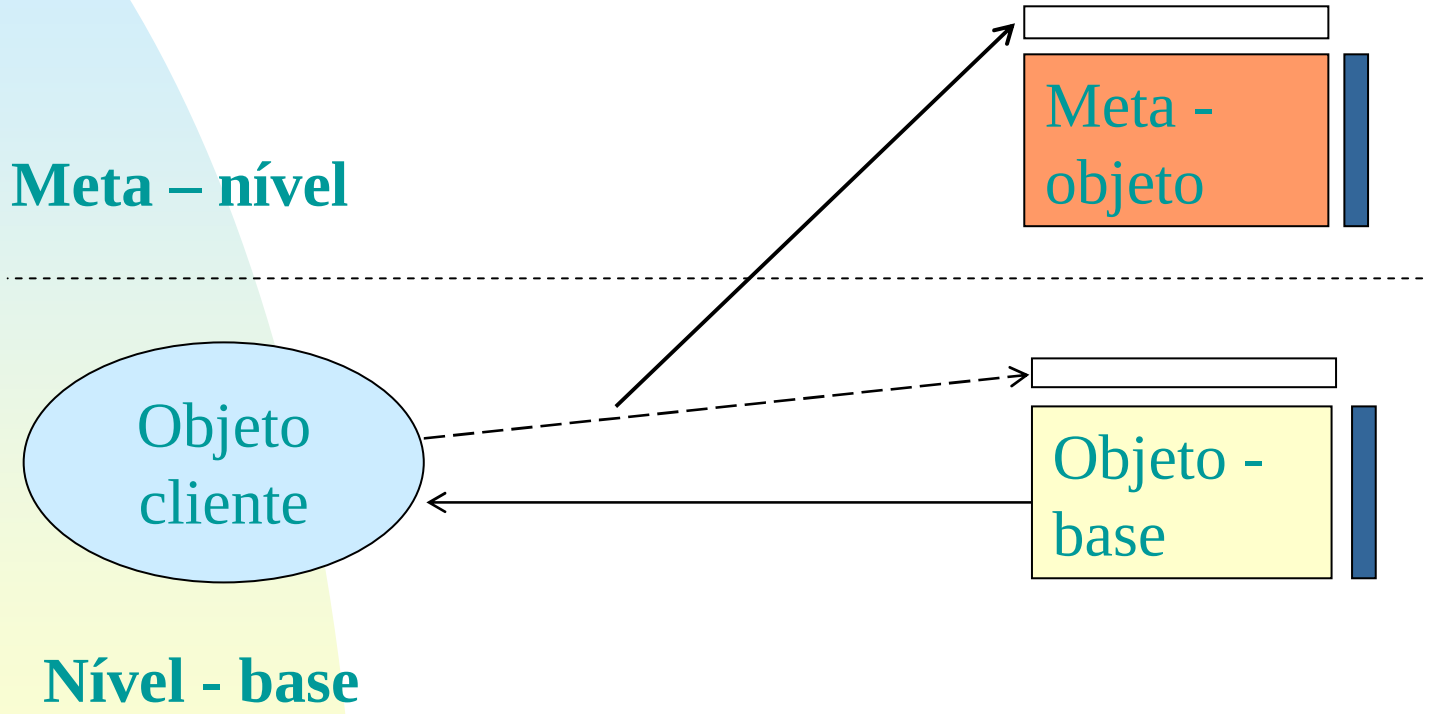
Código fonte utilizando uma meta interface

```
public Classe metaInterface () {
        Classe classe obtemClasse("Comum");
        // a variavel atributo recebe todos os atributos da classe Comum
        Atributo [] atributos = classe.retornaAtributo();
        for (int i=0; i<atributos.tamanho(); i++){
                if (atributos[i].nome = "pe"){
                        //se o nome do atributo for pe, ele deleta o atributo
                        // e cria outro atributo com estrutura diferente
                        classe.apagaAtributo("pe");
                        classe.adicionaAtributo(pe, int, private);
        }}
        // a variável metodos recebe todos os métodos da classe Comum
        Metodo []  metodos = classe.retornaMetodos ();
        for (int i = 0; i< metodos.tamanho(); i++) {
                if (metodos[i].nome == "desenhar"){
                        // o método que tiver o nome igual a desenhar recebe mais um parâmetro
                        // a sua chamada, do tipo inteiro e com  o nome de pe
                        metodos[i].adicionaParametro(int, pe);
                        // neste mesmo método, primeira linha, é adicionado novo código
                        metodos[i].adicionaCorpo ("this.pe = pe",0);
        }}
        //retorna a classe modificada para que a aplicação possa utilizá-la
        return classe;
}
```

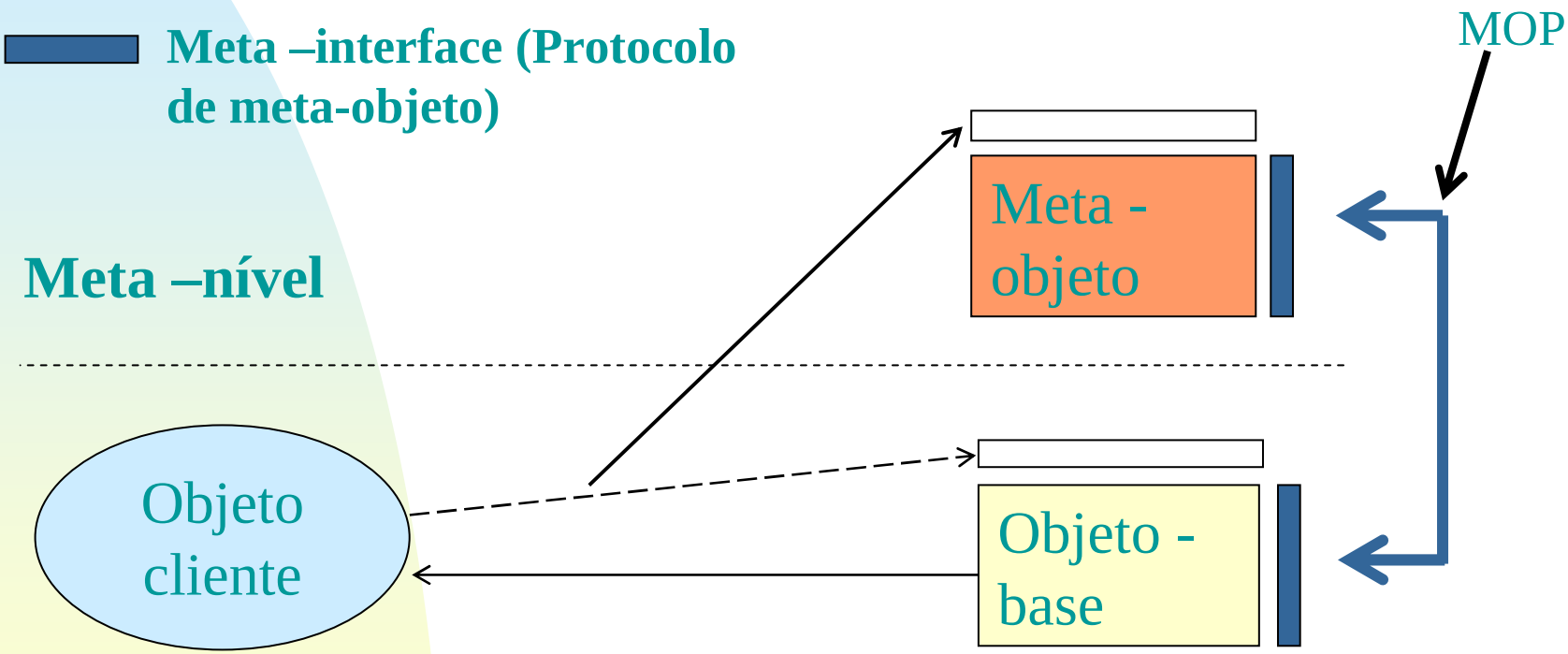
Objeto transformado



Protocolo de metaobjetos (MOP)



Protocolo de metaobjetos (MOP)

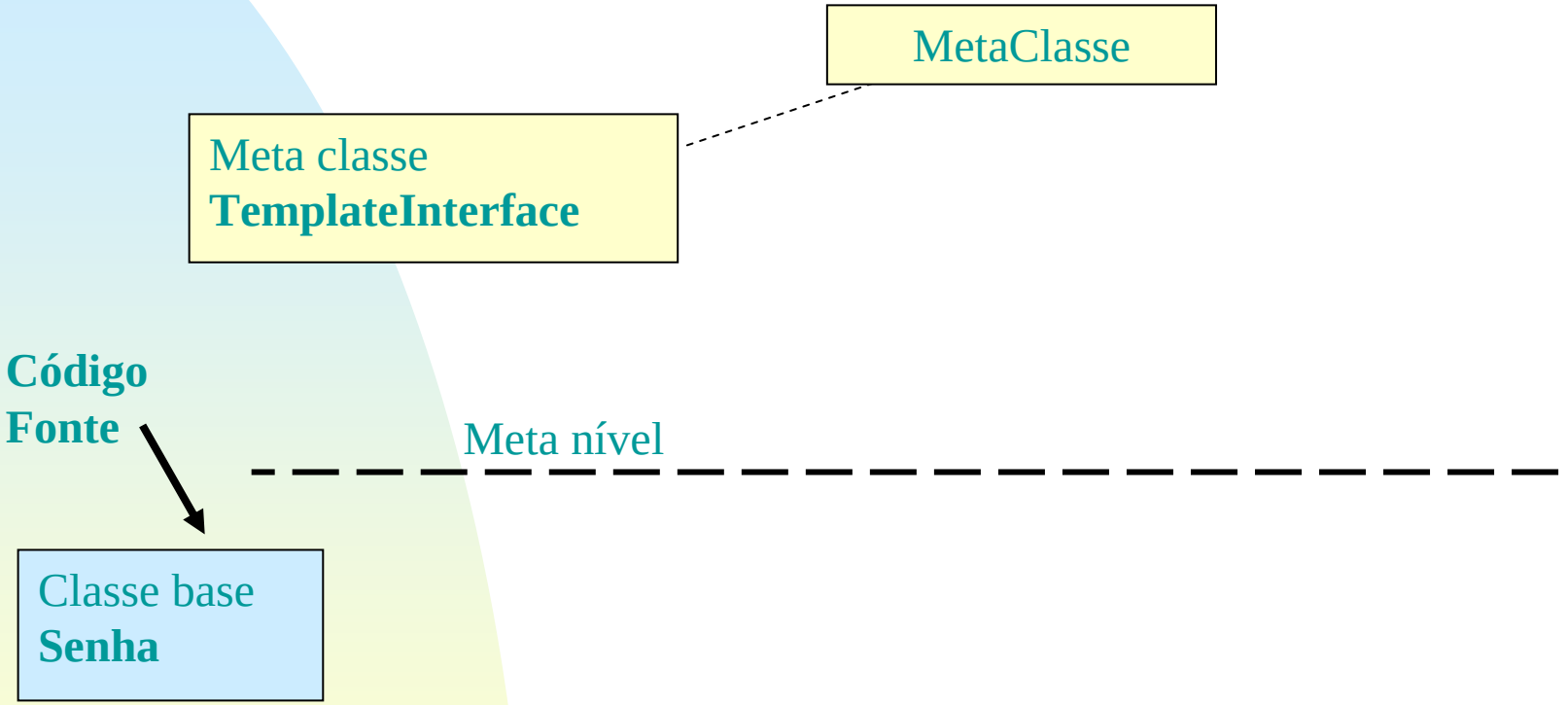


Protocolo de metaobjetos (MOP)

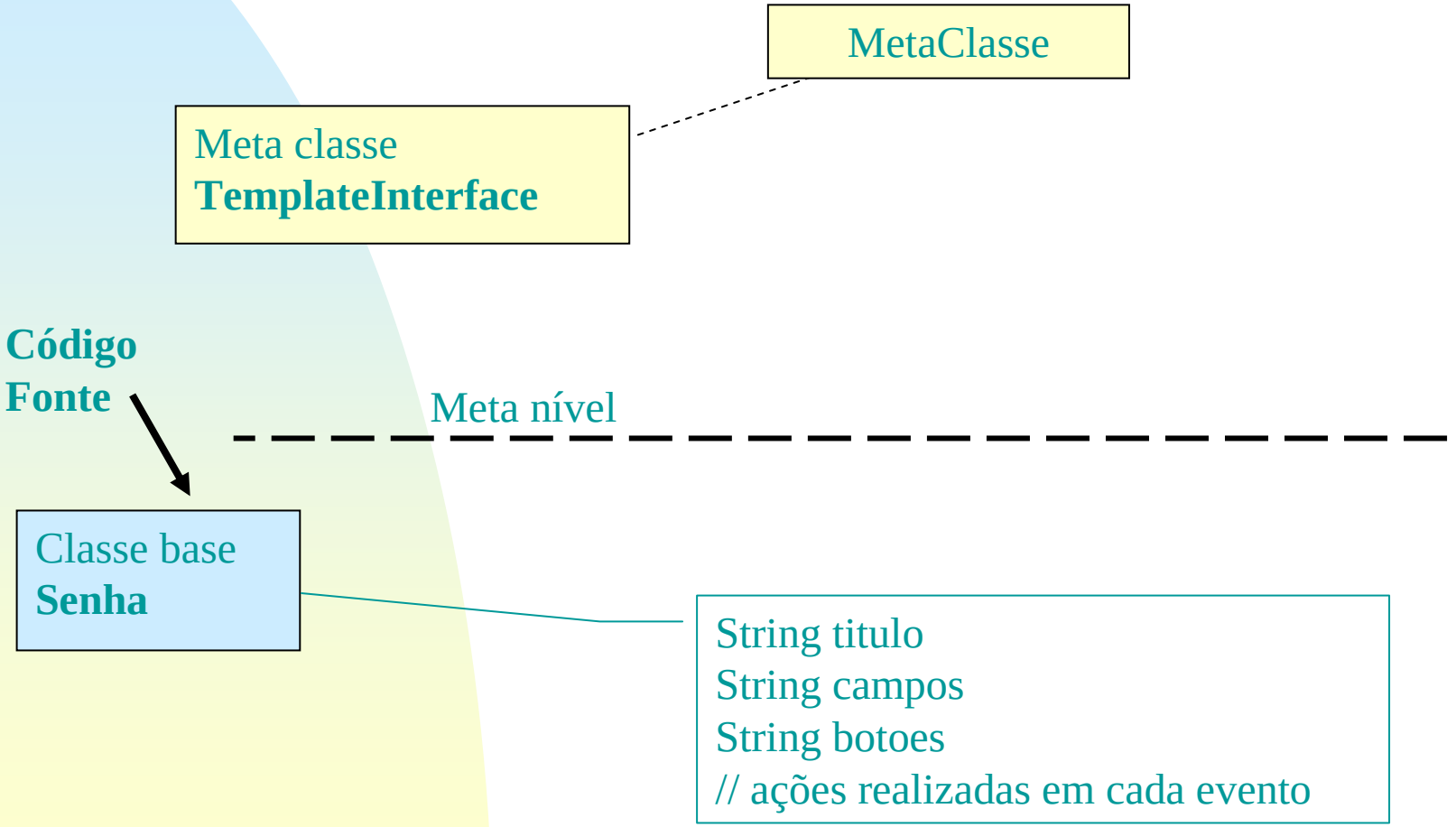
- Segundo [KIC1991], o MOP pode ser dividido em:
  - **Protocolo de introspecção**;
  - **protocolo de invocação**;
  - **protocolo de intercessão**.

- **Reificação**: é a transformação de informações sobre a execução de um programa orientado a objetos em dados disponíveis ao próprio programa.

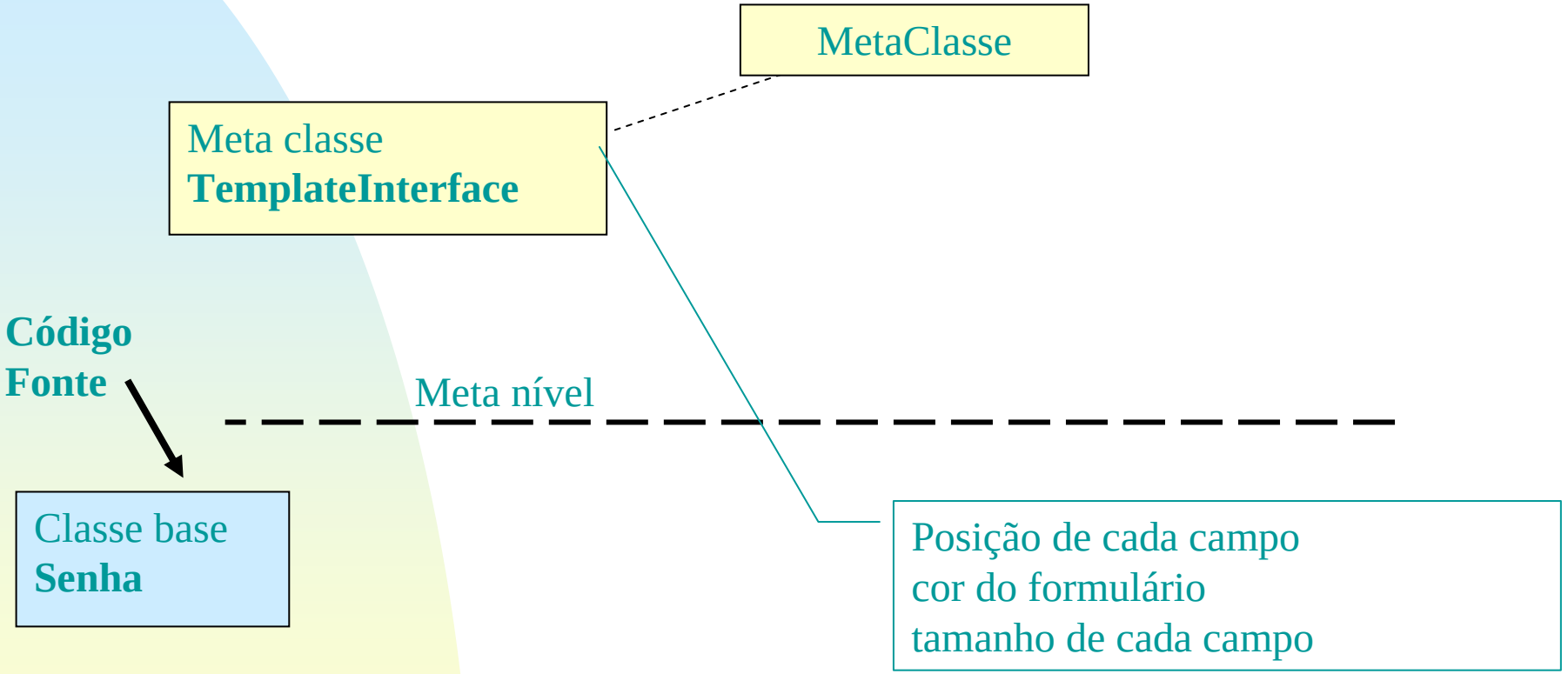
Protocolo de metaobjetos (MOP)



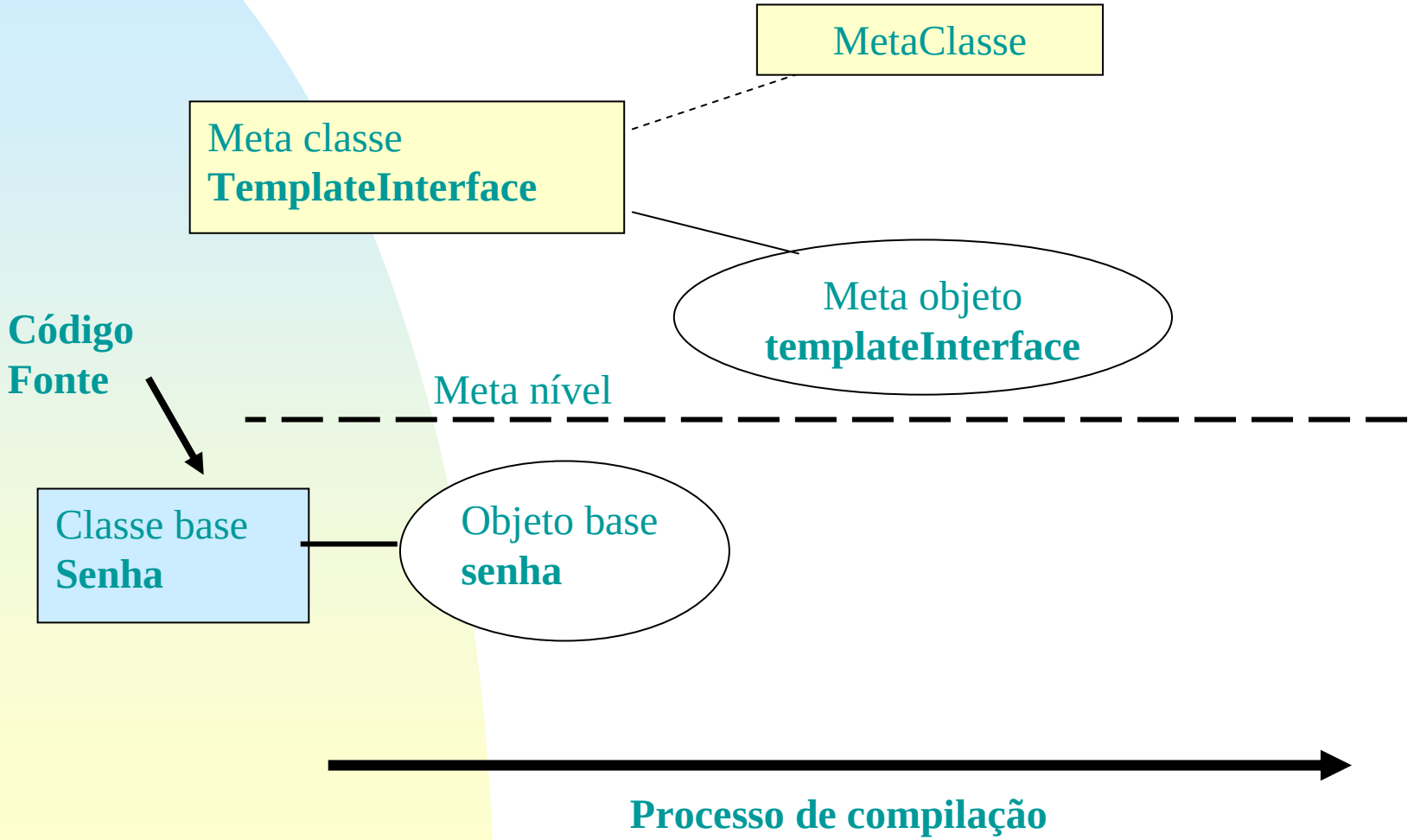
Protocolo de metaobjetos (MOP)



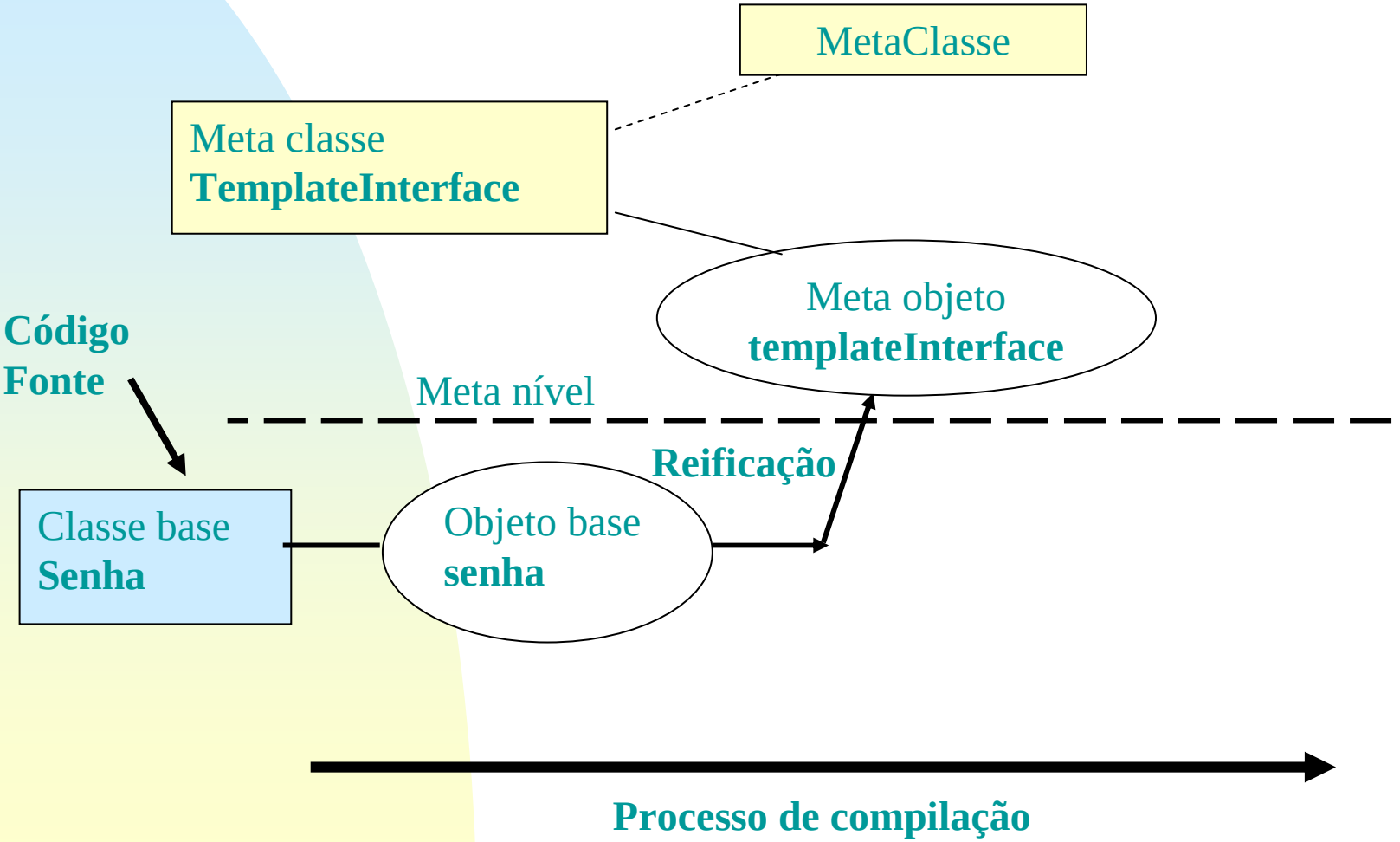
Protocolo de metaobjetos (MOP)



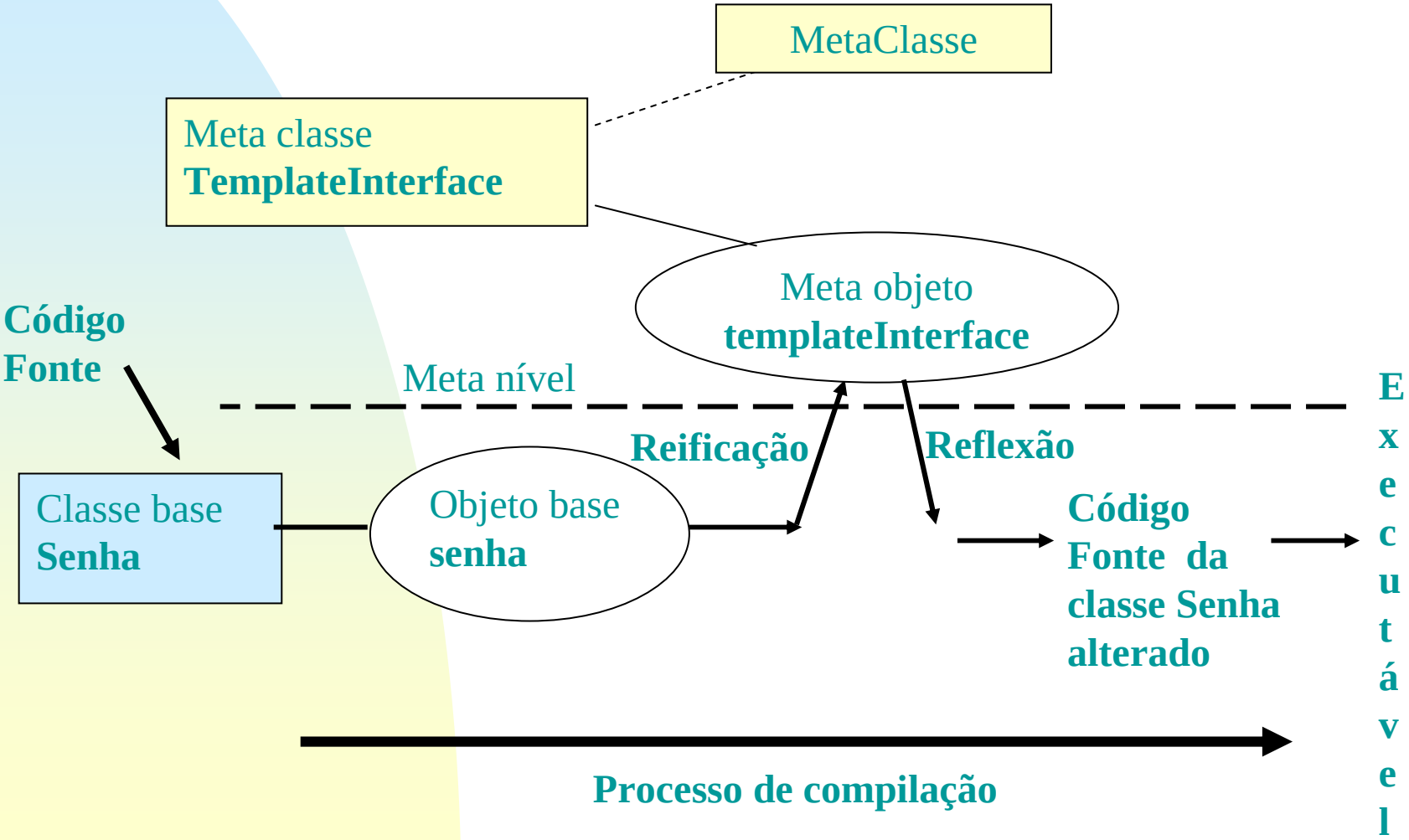
Protocolo de metaobjetos (MOP)



Protocolo de metaobjetos (MOP)



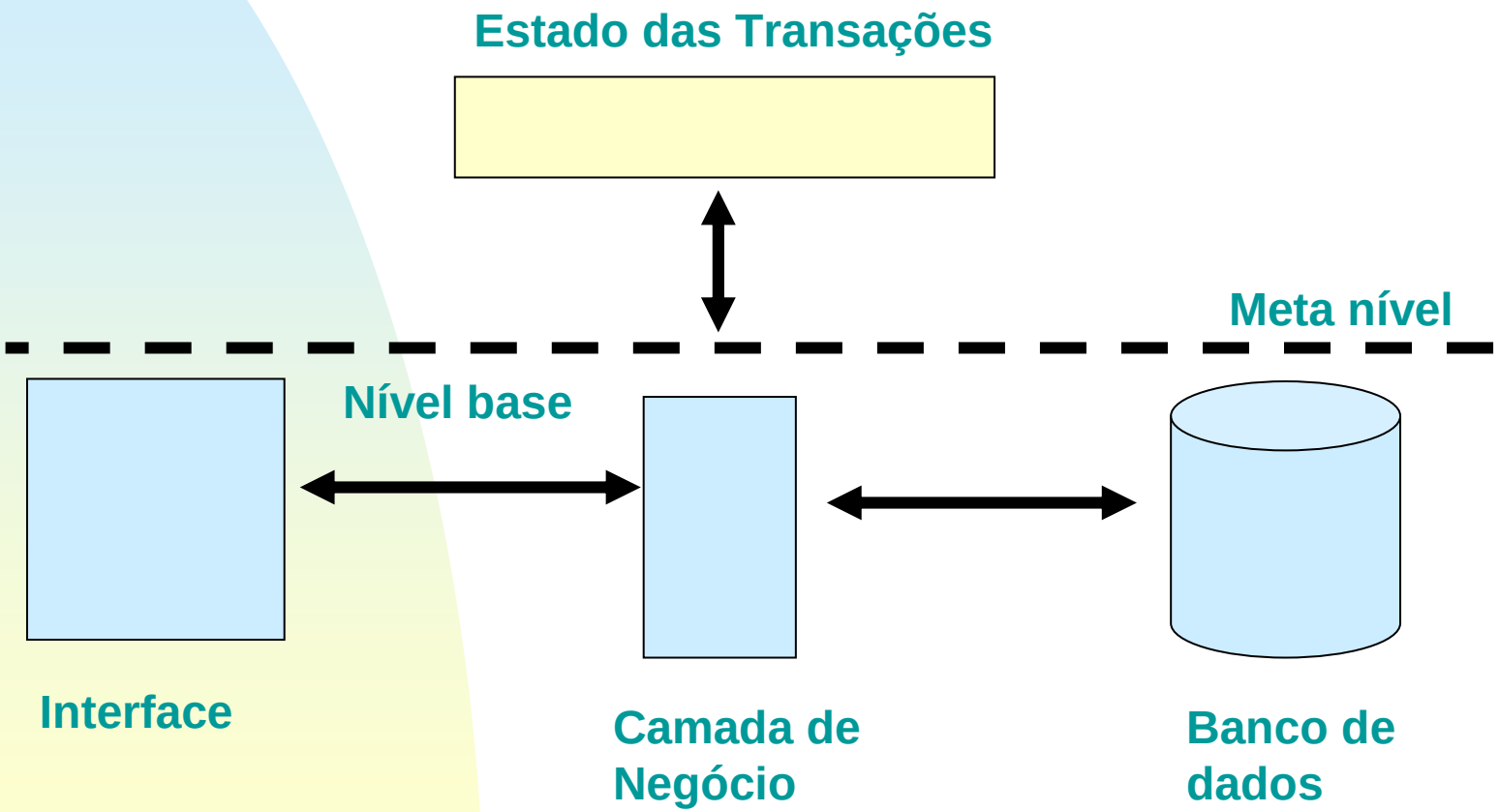
Protocolo de metaobjetos (MOP)



Protocolo de metaobjetos (MOP)

- **Quais entidades devem ser transformadas em algo que possa sofrer operações no meta-nível ?**
  - Processo de reificação.
- **Como o relacionamento entre o nível base e o meta-nível é implementado ?**
  - Interface básica / Meta interface
- **Quando o sistema passa para o meta-nível ?** [MAE1987]
  - Objeto de nível base
  - Ao sistema

Exemplo de aplicação de *n-camadas* reflexiva



Protocolo de metaobjetos (MOP)

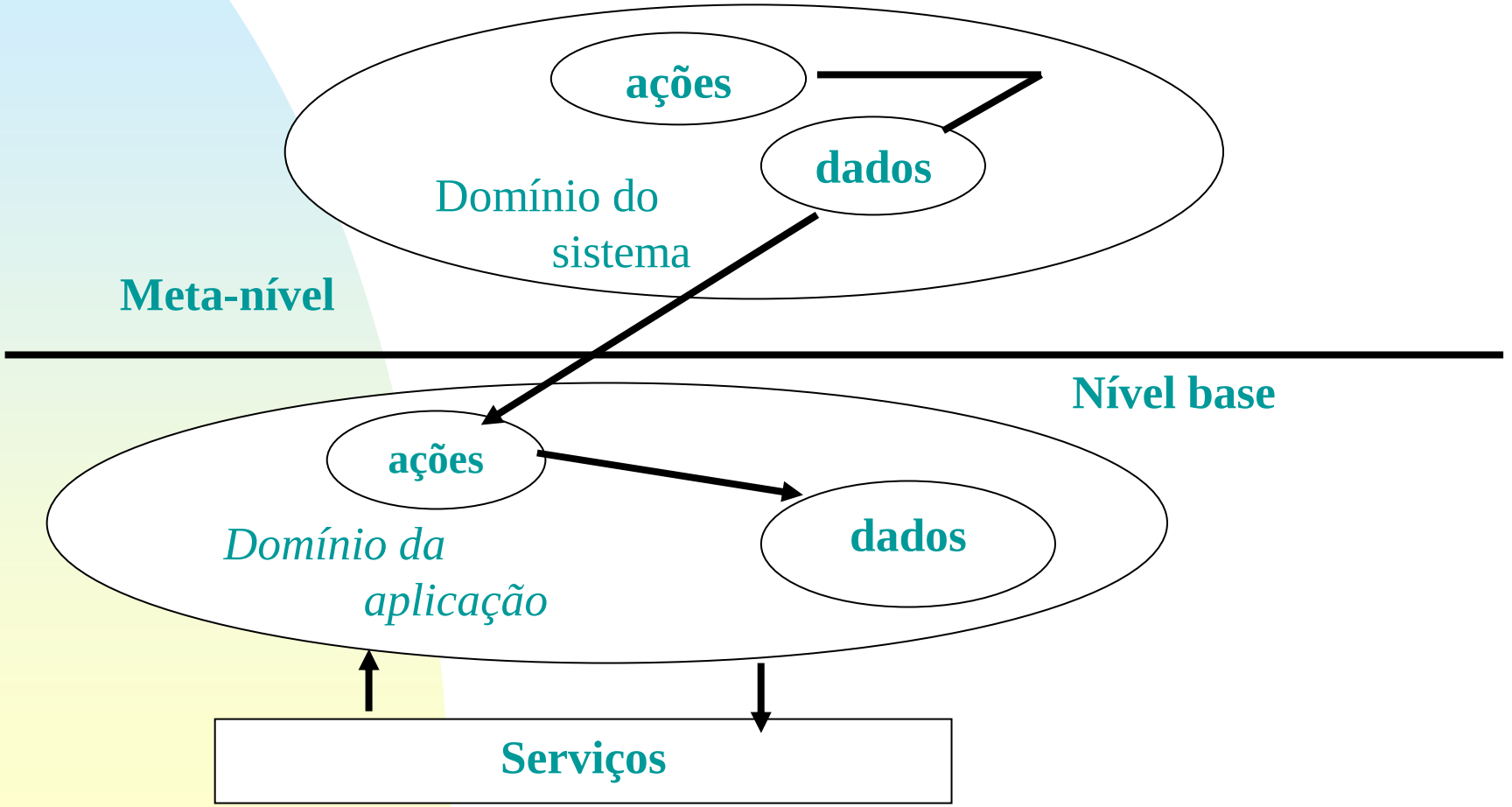
- **Protocolo de metaobjetos em tempo de compilação**
  - Realizam o processo de reificação e reflexão em tempo de compilação
  - Compilador próprio
  - Associação parte do nível base
- **Protocolo de metaobjetos em tempo de execução**
  - Realizam o processo de reificação e reflexão em tempo de execução
  - Não possuem compilador próprio
  - Associação parte do sistema

Arquitetura reflexiva

- Segundo [BUZ1998], a reflexão computacional define uma arquitetura composta por meta-níveis, onde se encontram ações a serem realizadas sobre um sistema alvo localizado no nível base (ou da aplicação).

- Abstração reflexiva

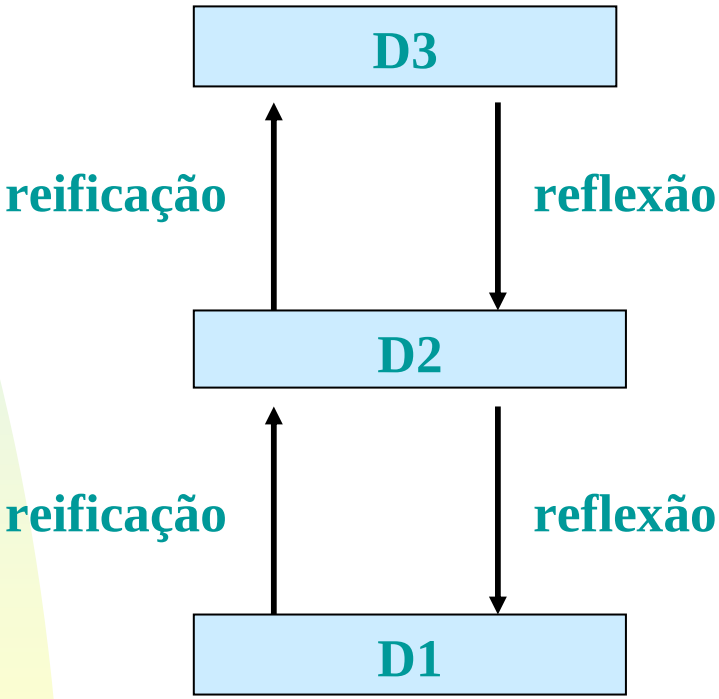
Arquitetura reflexiva



Arquitetura reflexiva

- **Vantagens**:
  - Redução de complexidade;
  - separação conceitual;
  - reutilização;
  - adaptabilidade.

Torre reflexiva



Reflexão estrutural

- Reflexão estrutural de uma classe **x** é toda a atividade realizada em uma meta-classe **Mcx**, com o objetivo de obter informações e realizar transformações sobre a estrutura estática da classe **x** [FER1989]

- Informações e transformações típicas:
  - obter informações sobre a classe **x**;
  - alterar a classe **x**;
  - atuar sobre as classes.

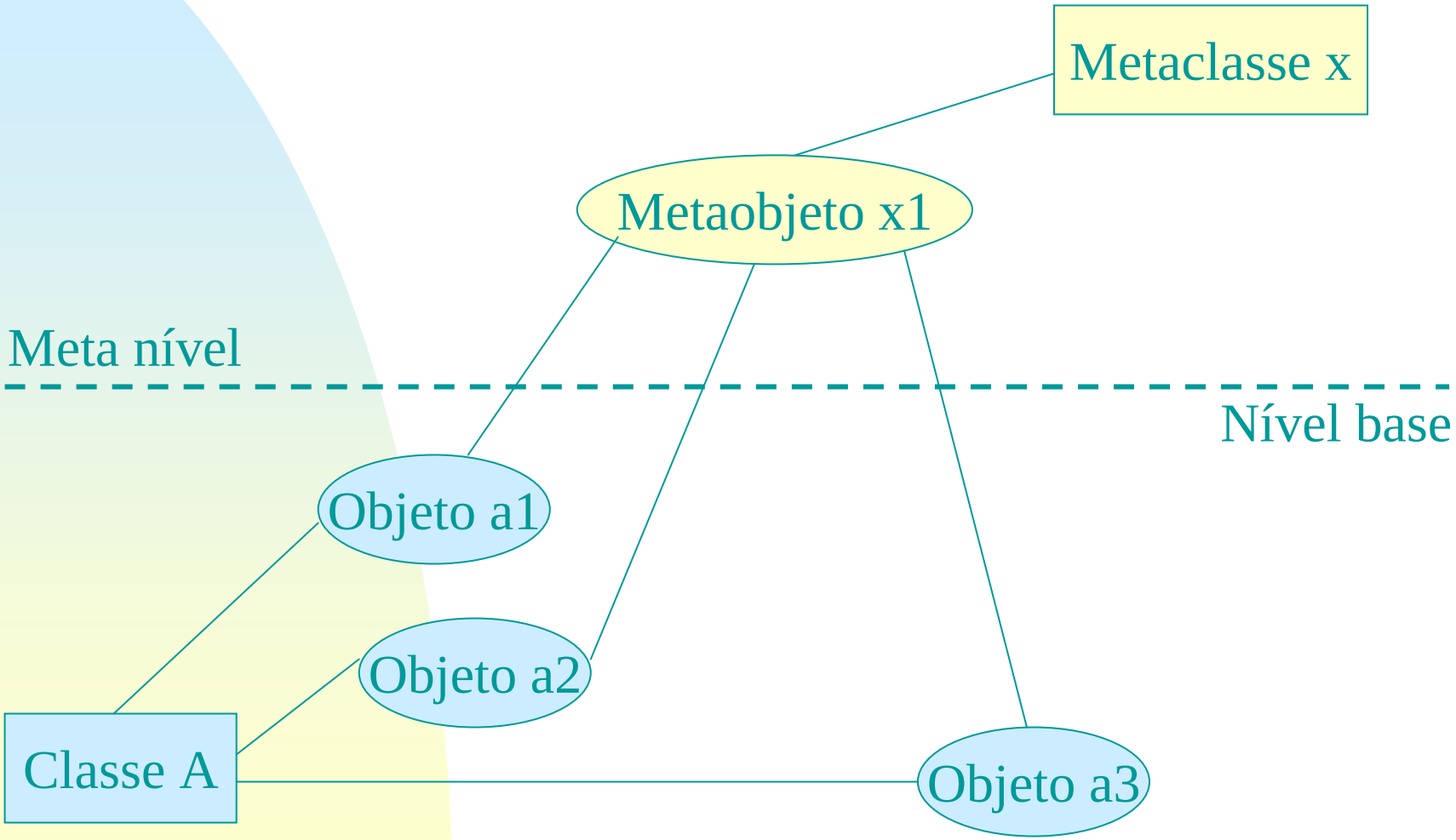
Reflexão comportamental

- Reflexão comportamental de um objeto **O** é toda atividade realizada por um metaobjeto **Mox** com o objetivo de obter informações e realiar transformações sobre o comportamento do objeto **O**.

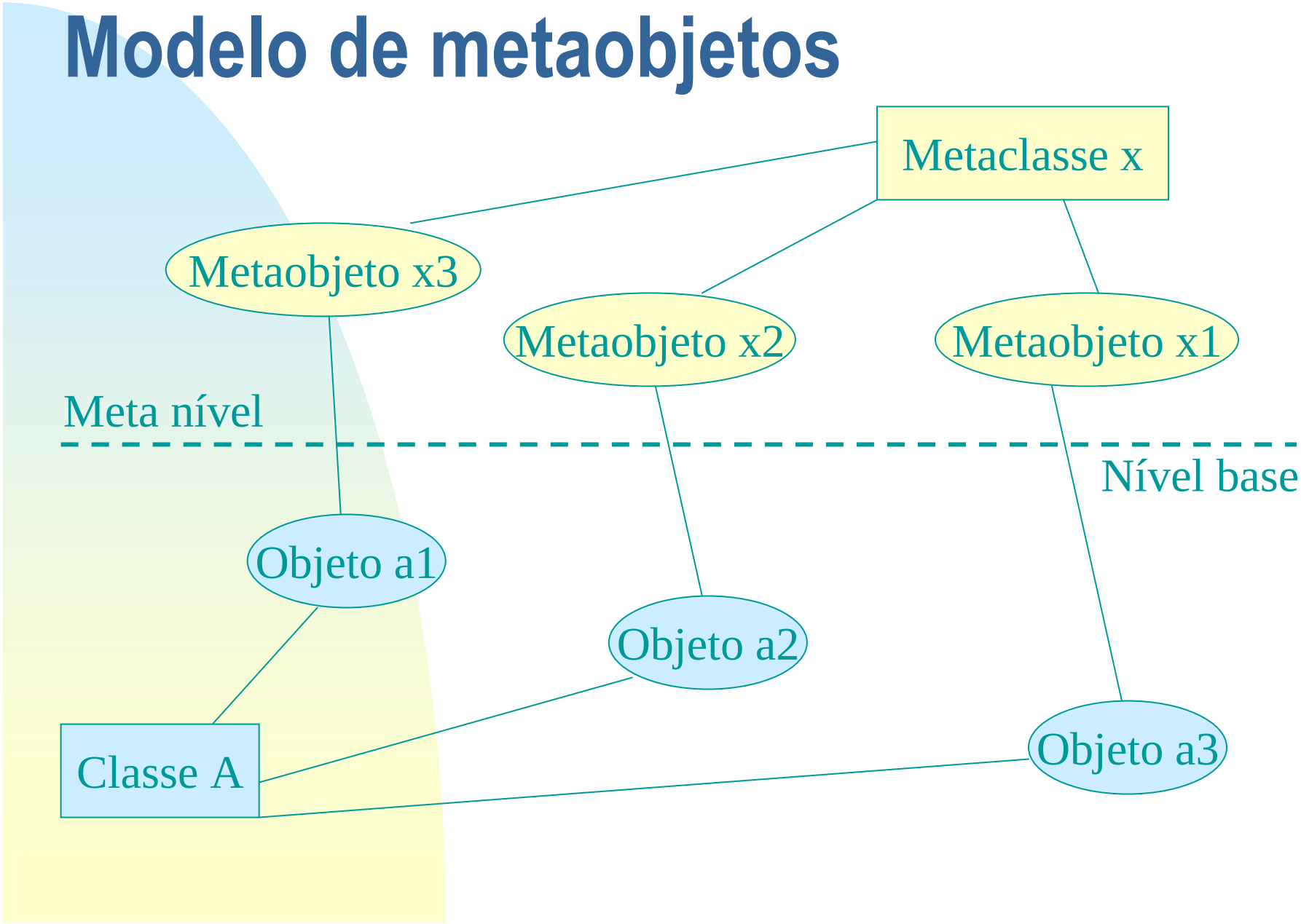
Modelos de reflexão

- Modelo de metaclasses
- Modelo de metaobjetos
- Modelo de metacomunicações

Modelo de metaclasses



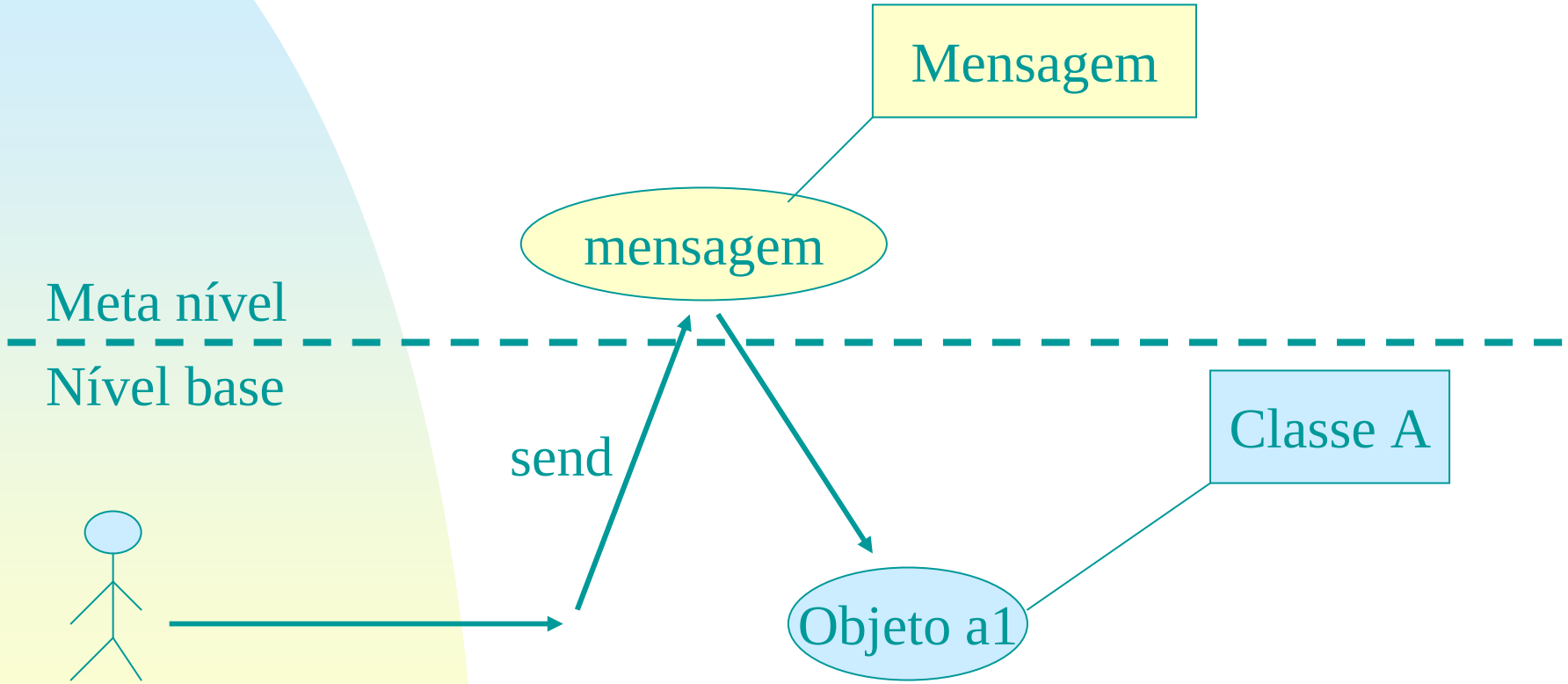
Modelo de metaobjetos



Modelo de metaobjetos

- **Características principais**:
  - individualidade;
  - separação de classes;
  - associação dinâmica;
  - definição circular.

Modelo de metacomunicações



Linguagens e ambientes reflexivos

- **Smalltalk**
  - Object / Metaclass
  - Todas as entidades são objetos
- **OpenC++**
  - Linguagem C++
  - Pré-processador
  - Modelo de metaobjetos
  - MetaObj

Linguagens e ambientes reflexivos

- ***Common LISP Object System* (CLOS)**
  - Estende a linguagem LISP para o modelo de objetos
  - Todas as características de uma classe são herdadas
  - Permite a redefinição da própria linguagem

- **Java**
  - Object
  - *java.lang.reflect*
  - Reflexão estrutural

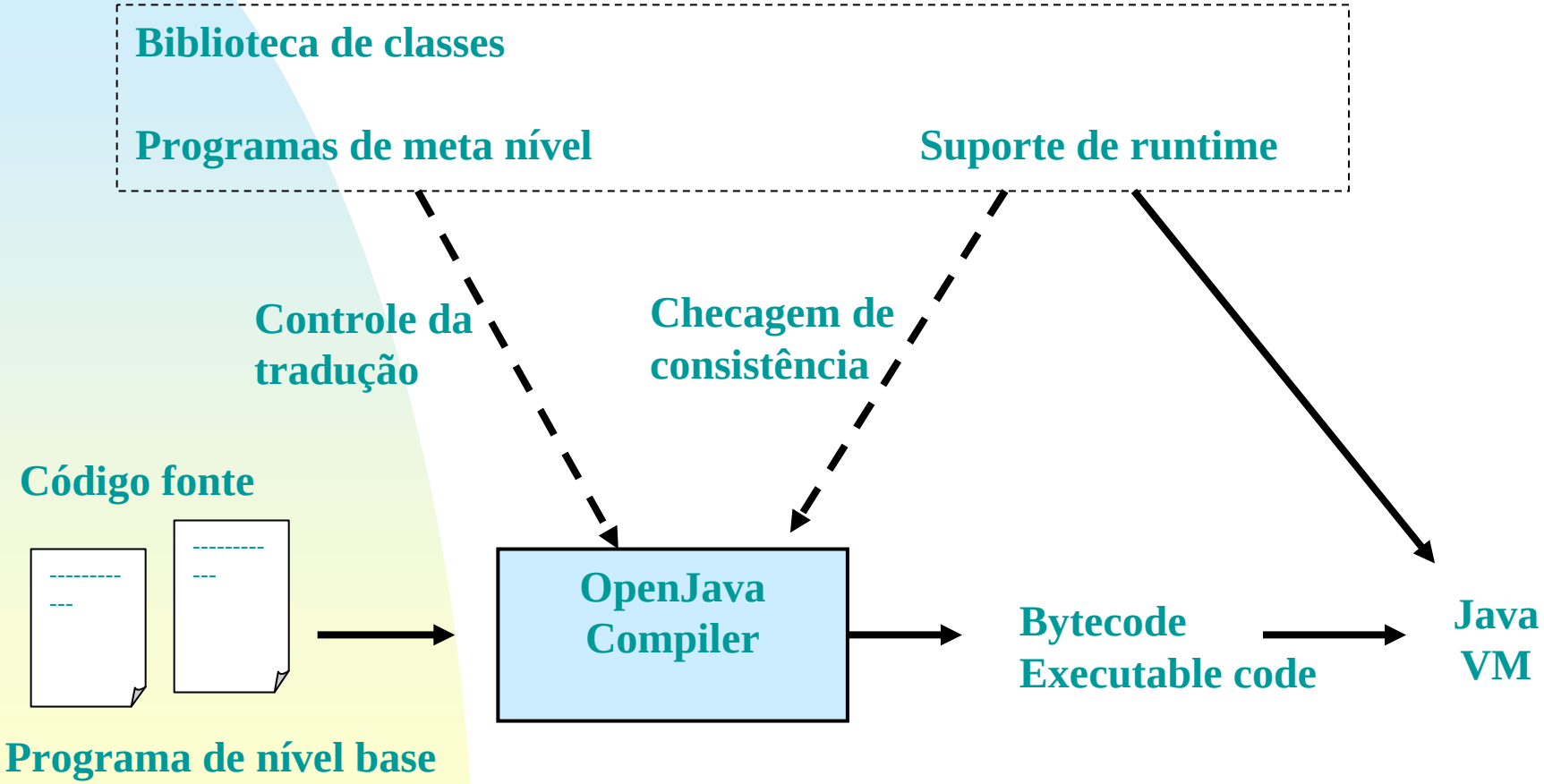
Ferramentas reflexivas

- OpenJava
  - Metaprotocolo em tempo de compilação
- Javassist
  - Metaprotocolo em tempo de execução

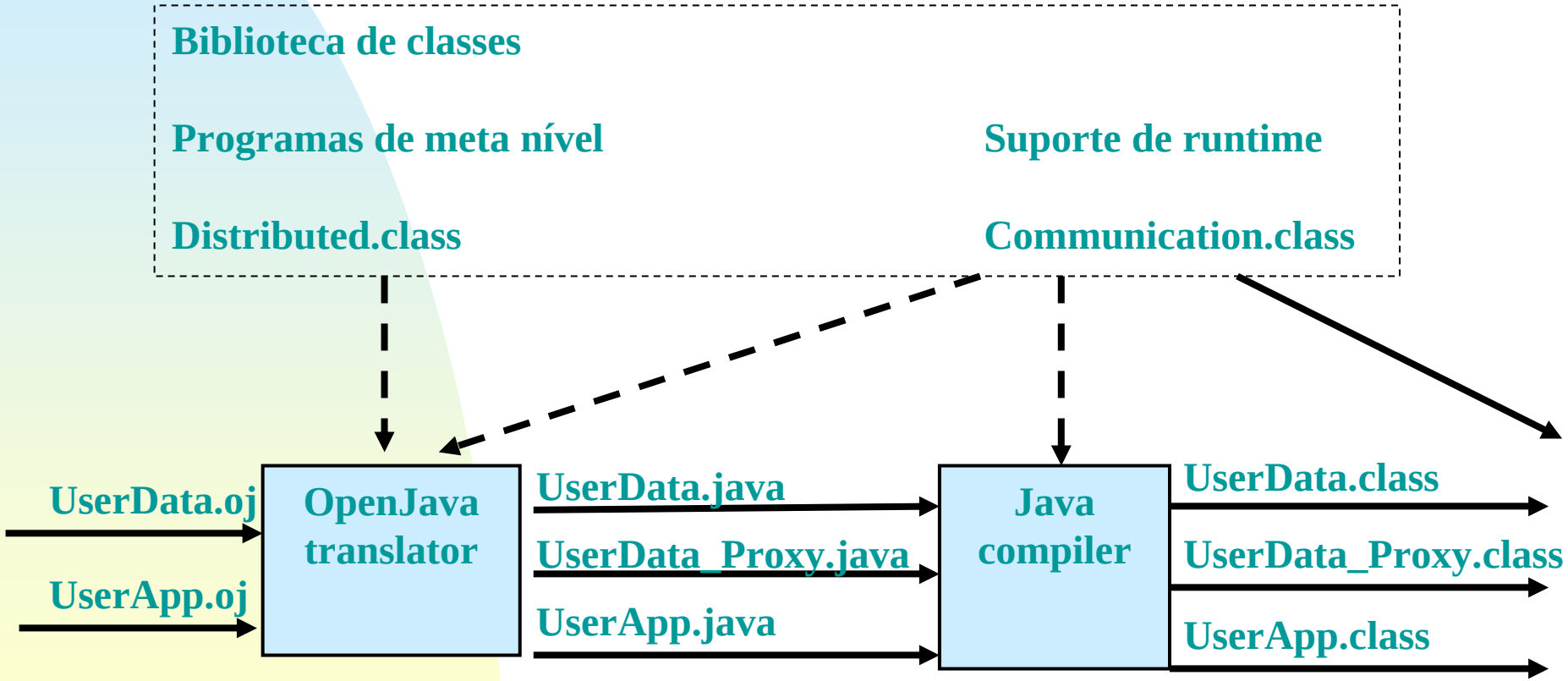
OpenJava

- Linguagem extensível baseada em Java
- Estrutura idêntica a do OpenC++
- Compilador *ojc*
- Aceita código fonte e gera código executável
- Recorre a bibliotecas de meta nível além das regulares

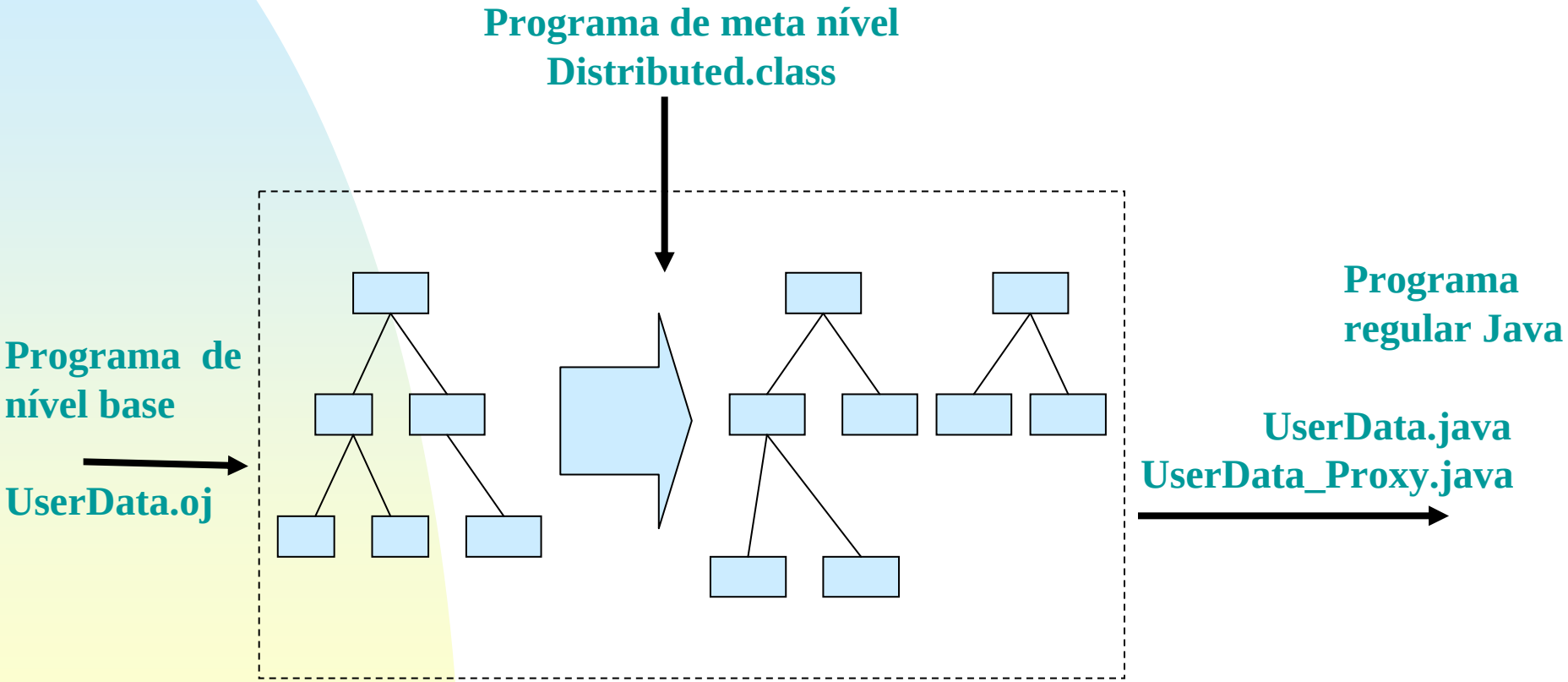
OpenJava - processo de compilação



OpenJava - módulos do compilador



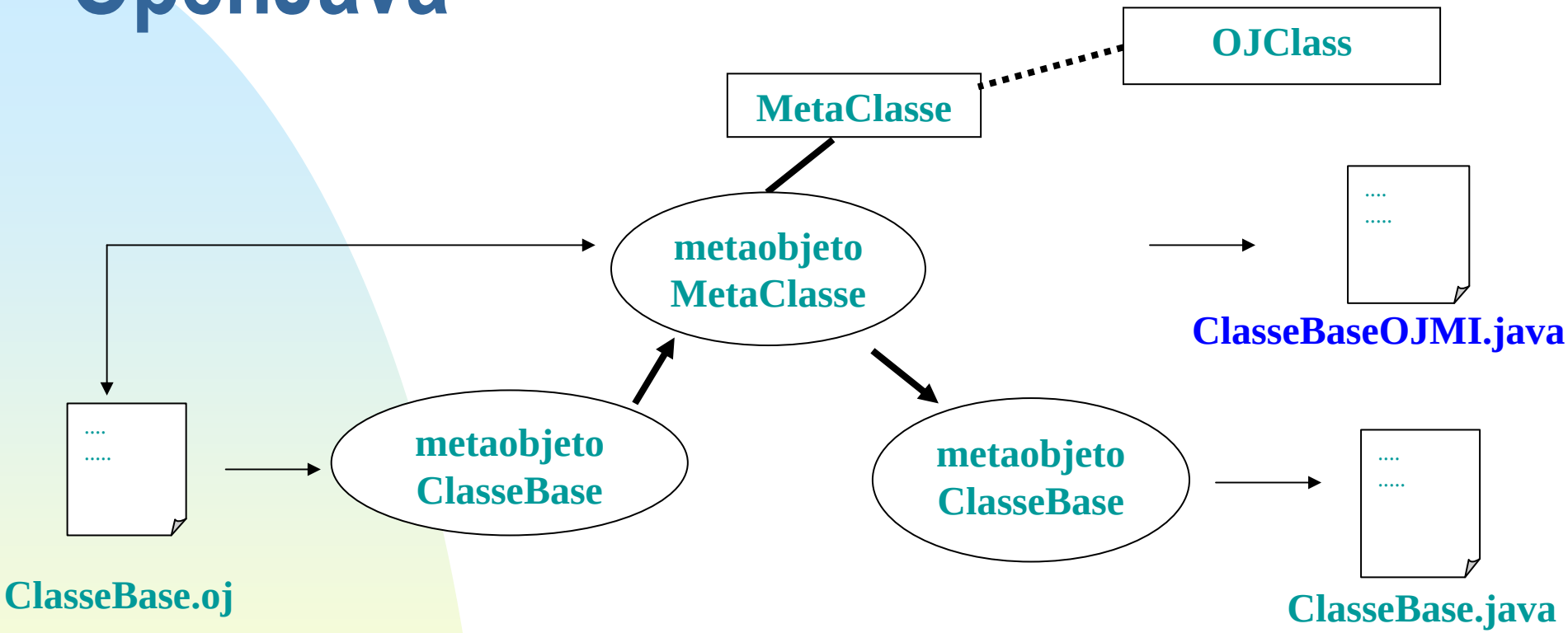
OpenJava - tradutor



OpenJava (código fonte exemplo)

```
public class ClasseBase instantiates MetaClasse {
        public static void main (String[] args) {
                hello ();
        }
        static void hello () {
                System.out.println (“Nível base...”);
}}

import openjava.mop.*;
import openjava.ptree.*;
public class MetaClasse instantiates Metaclass extends OJClass {
        public void translateDefinition () throws MOPException {
                OJMethod [] methods = geteclaredMethods ();
                for (int i = 0; i < methods.length; ++i) {
                        Statement printer = makeStatement (“System.out.println(\“Este é o método ”+
                        methods[i]+”\” );”);
                        Methods[i].getBody().insertElementAt (printer, 0);
}}}
```

OpenJava



Herança - - - - - -

Instância normal ——

Instância reflexiva ⟷

Informação sobre o meta nível

OpenJava API

- OJClass / OJField / OJMethod / OJConstructor
- OpenJava versão 1.0 (20/jan/2000)

Javassist

- Javassit é uma ferramenta que visa facilitar o desenvolvimento de aplicações que utilizam a linguagem Java. Permite que programadores possam automatizar alguns tipos de definições de classe e permite realizar reflexão computacional em tempo de execução.

- API do Javassit:
  - javassist;
  - **javassist.reflect**;
  - javassist.rmi;
  - javassist.tool.

javassist.reflect

Interface

Metalevel	Interface que fornece o metaobjeto e a metaclasse do objeto respectivo

Classes

ClassMetaobject	Corresponde a Metaclasse do modelo
Compiler	Possibilita realizar reflexão em *bytecodes*
Implement	Implementa mecanismos de reflexão
Metaobject	Corresponde ao metaobjeto do modelo
ReflectLoader	Permite a seleção dos objetos reflexivos
ReflectServer	Implementa uma classe reflexiva para aplicações distribuídas
Sample	Fornece um template para uma classe reflexiva

Exceções

CannotCreateException	Lança uma exceção quando não é possível instanciar um metaobjeto
CannotInvokeException	Lança uma exceção quando não é possível invocar um método

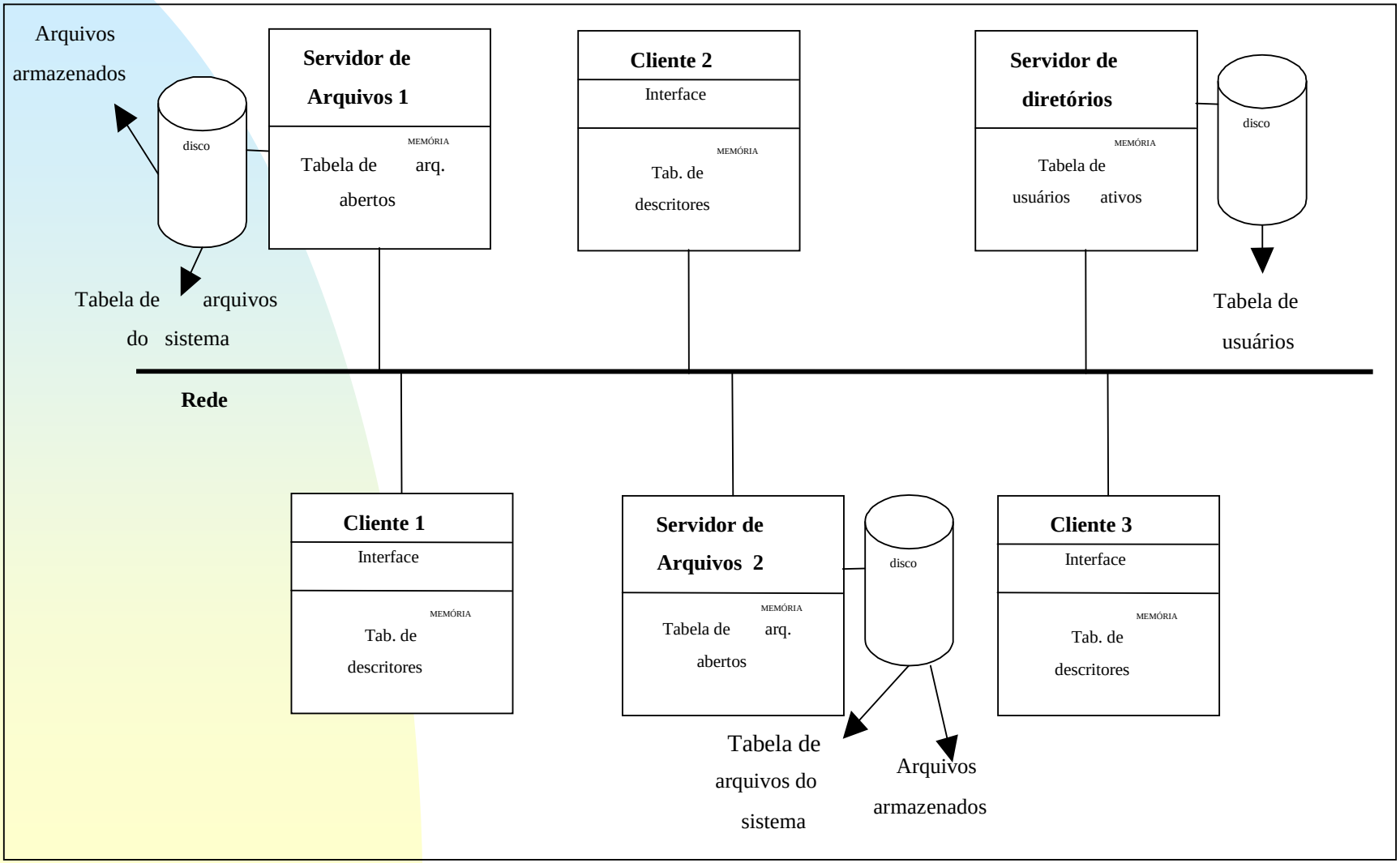
Javassist (código fonte exemplo)

```
// Classe principal do sistema
import javassist.Loader; import javassist.Reflect.ClassMetaobject;
import javassist.Reflect.ReflectLoader;
public class Main{
        public static void main (String[] args) throws Throwable {
                Loader c1 = (Loader)Main.class.getClassLoader();
                ReflectLoader loader = new ReflectLoader();
                loader.makeReflective ("ClasseBase",MetaClasse.class,Metaobject.class);
                c1.addUserLoader (loader);
                c1.run ("ClasseBase",args);
}}

// Metaclasse
import javassist.*; import javassist.reflect.*;
public class MetaClasse extends Metaobject {
        public MetaClasse (Object classe, Object[] args) {
                super (classe, args);
                System.out.println ("Metaobjeto instanciado: "+classe.getClass().getName());
        }
        public Object trapMethodCall (int identifier, Object[] args) {
                System.out.println (" Alterando o método: "+ getMethodName(identifier) +" em
                "+getClassMetaobject().getName());
                return super.trapMethodCall (identifier, args);
}}

// Classe base
public class ClasseBase {
        public static void main (.....) { .....}
        public ClasseBase () { .... }
}
```

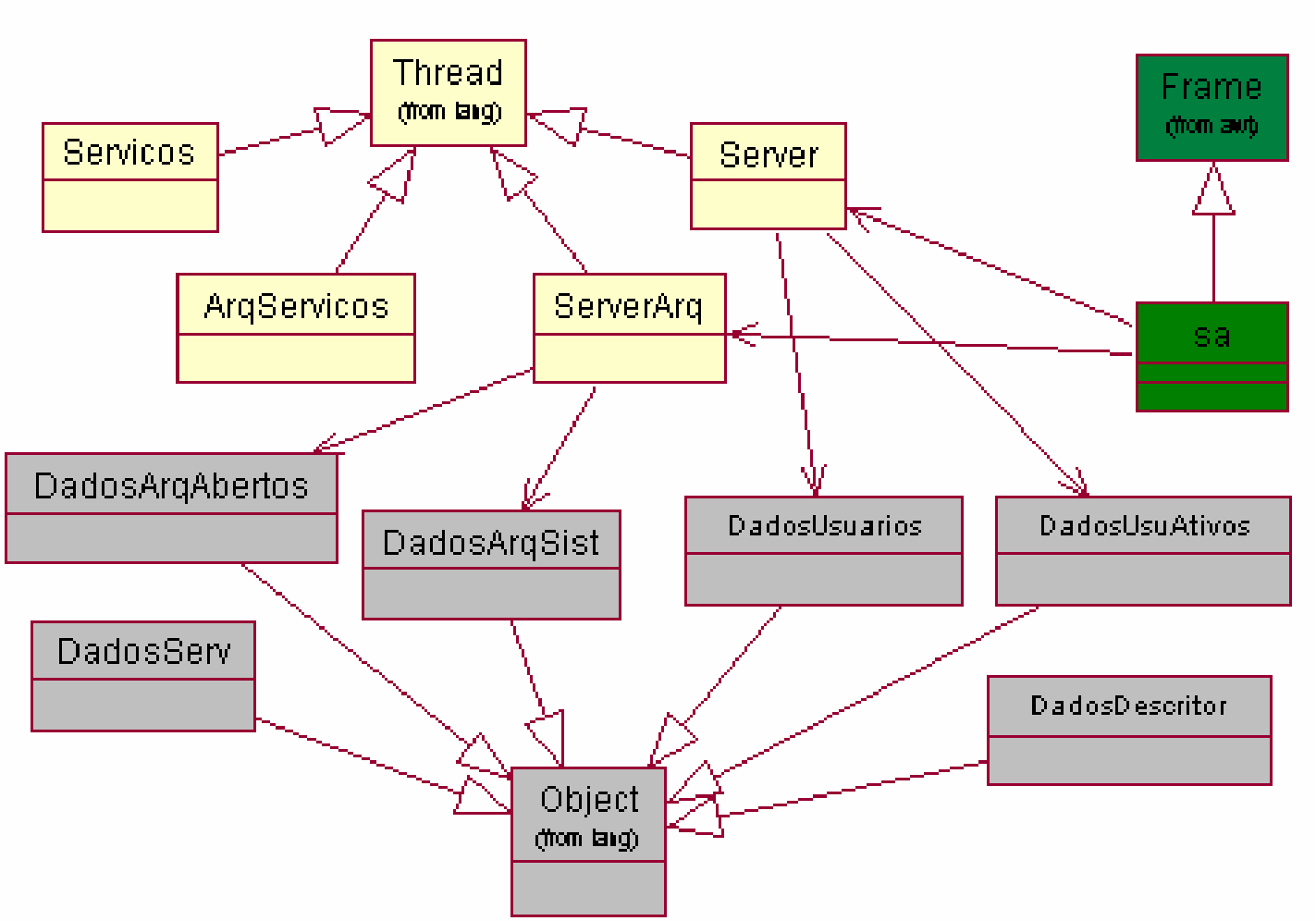
Javassist

- Baseada em experiências sobre o OpenC++ e OpenJava
- Nenhuma ferramenta para suporte, nem compilador
- Javassist / MetaXa / Dalang
- Javassist 0.6 (04/abril/2000)

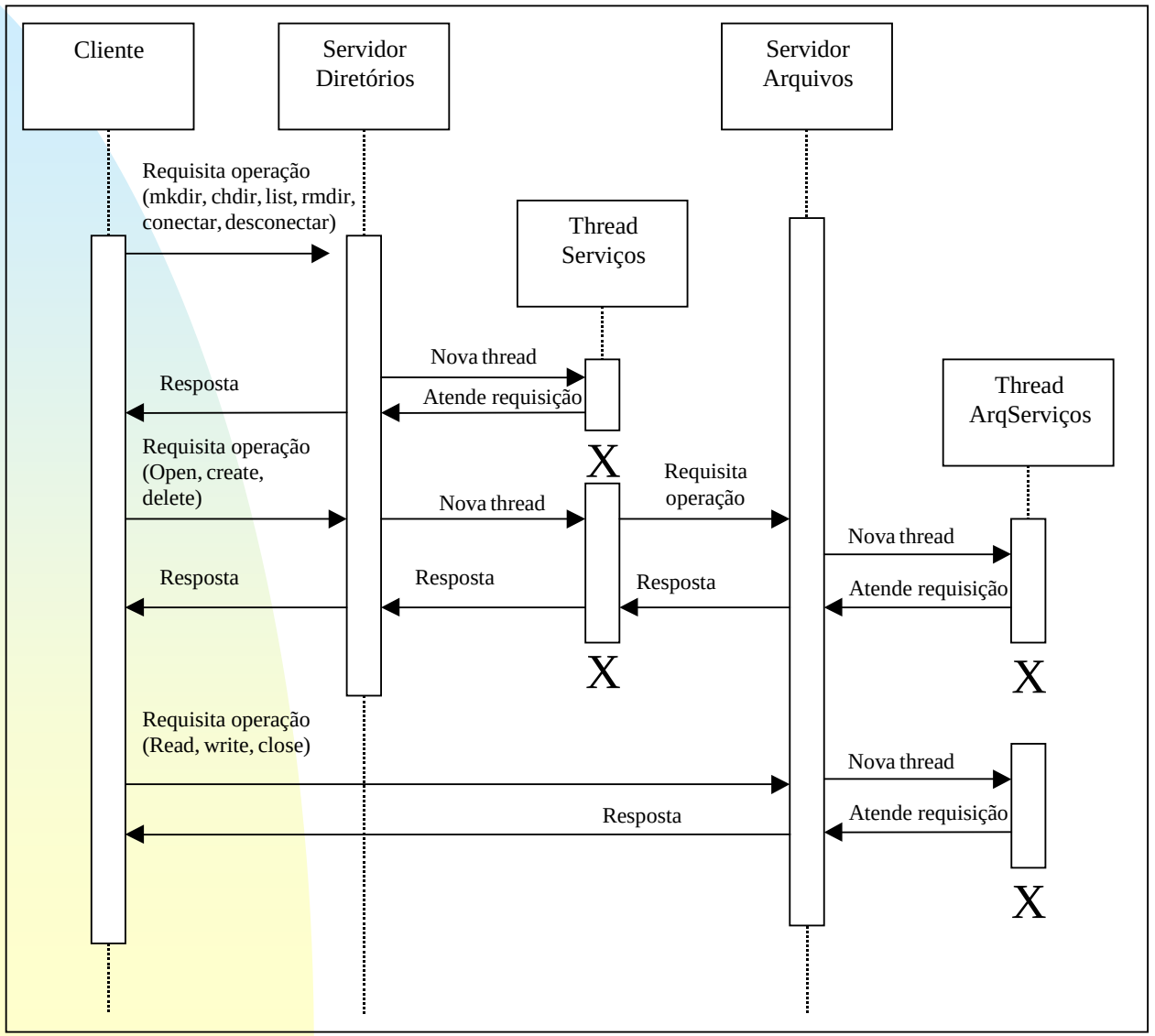
Gerenciador de arquivos



Gerenciador de arquivos (diagrama de classes)



Gerenciador de arquivos (interação entre objetos)



Modelagem e implementação

- Gerenciador de arquivos distribuídos - Java

- Comportamento para armazenamento de arquivos
- *Log* do sistema
- Tratamento de erros

- *Unified Modeling Language* (UML)
  - Diagrama de classes
  - Diagrama de interação entre objetos

Arquitetura do protótipo

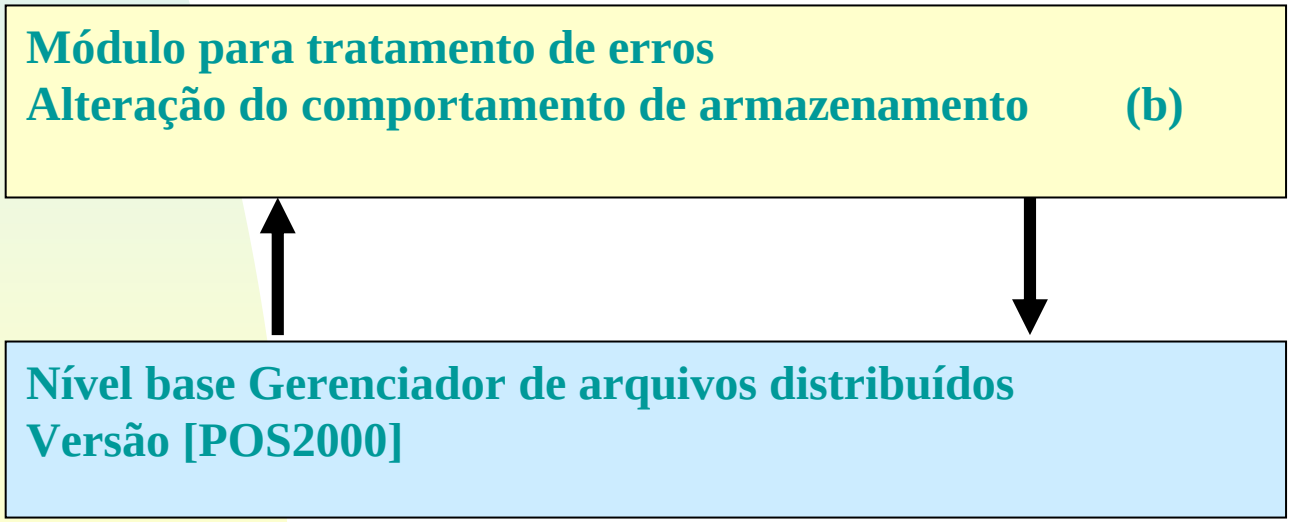
Nível base Gerenciador de arquivos distribuídos
Versão [POS2000]

Arquitetura do protótipo

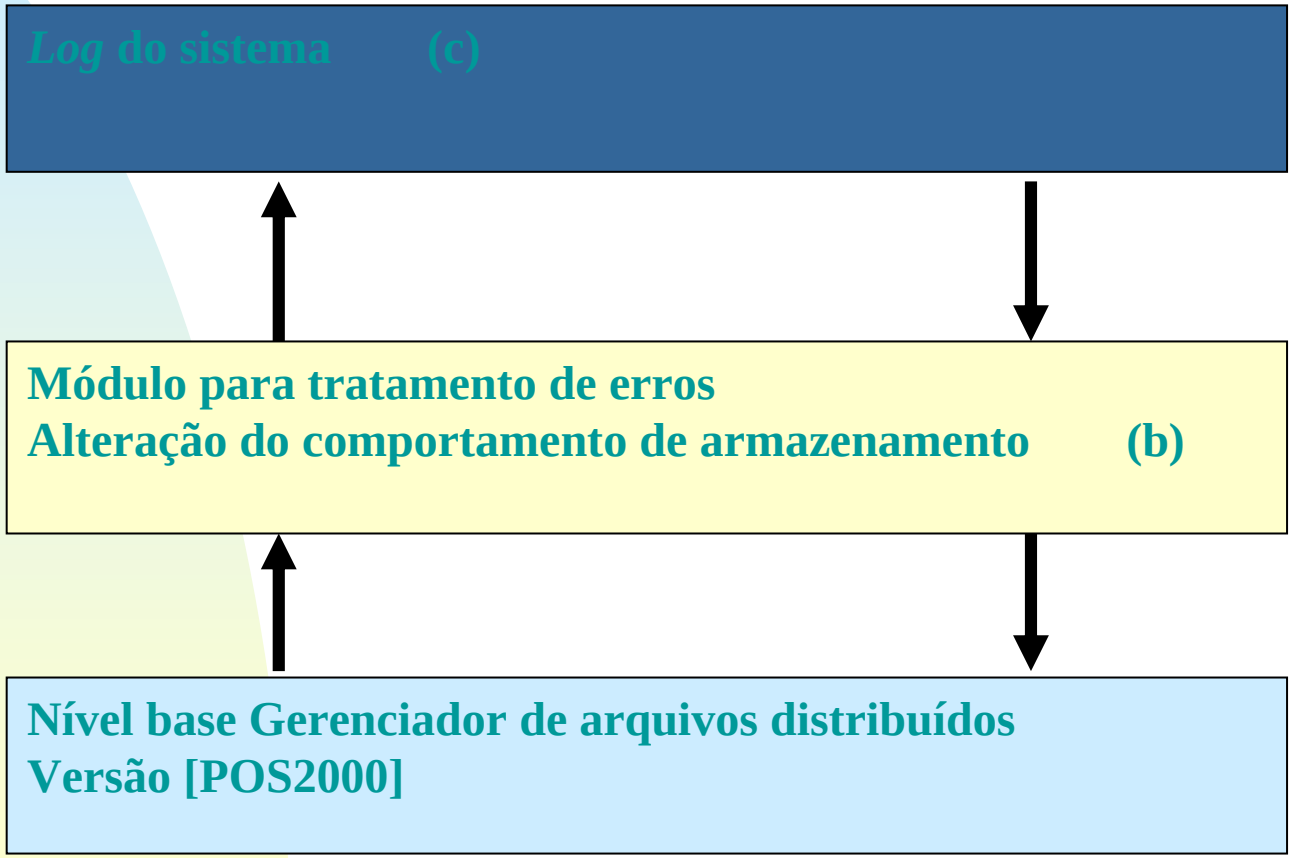
Módulo para tratamento de erros
Alteração do comportamento de armazenamento (b)

Nível base Gerenciador de arquivos distribuídos
Versão [POS2000]

Arquitetura do protótipo



Arquitetura do protótipo



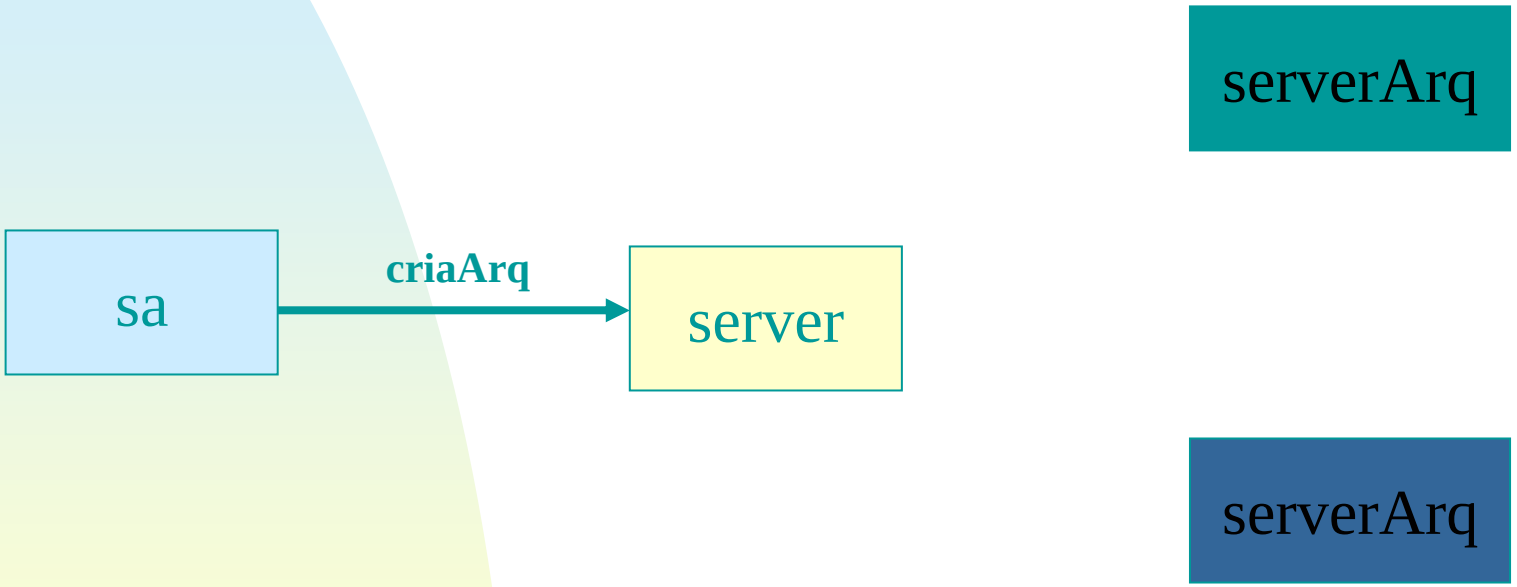
Ferramentas Modelagem/Implementação

- Rational Rose 2000
- OpenJava versão 1.0
- Javassist versão 0.6
- JDK 1.2.2

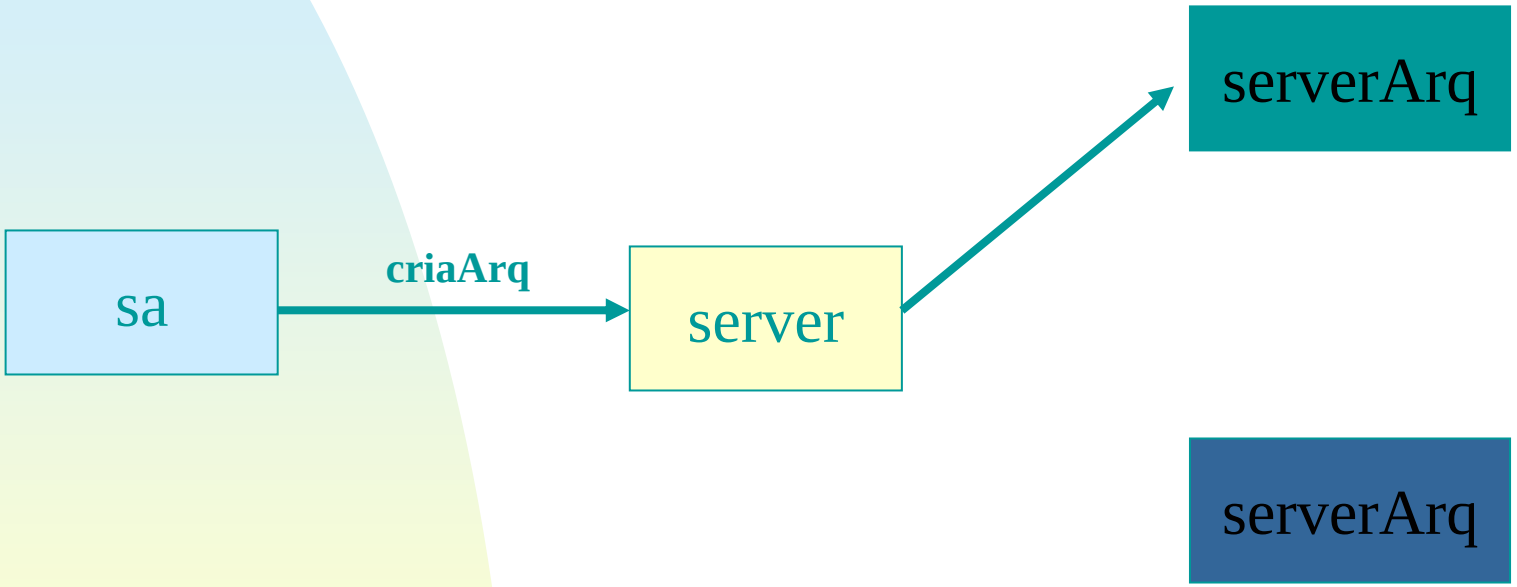
Modelagem e implementação

- **Escolha do Servidor para armazenamento de arquivos**
- ***Log* do sistema**
- **Módulo para tratamento de erros**

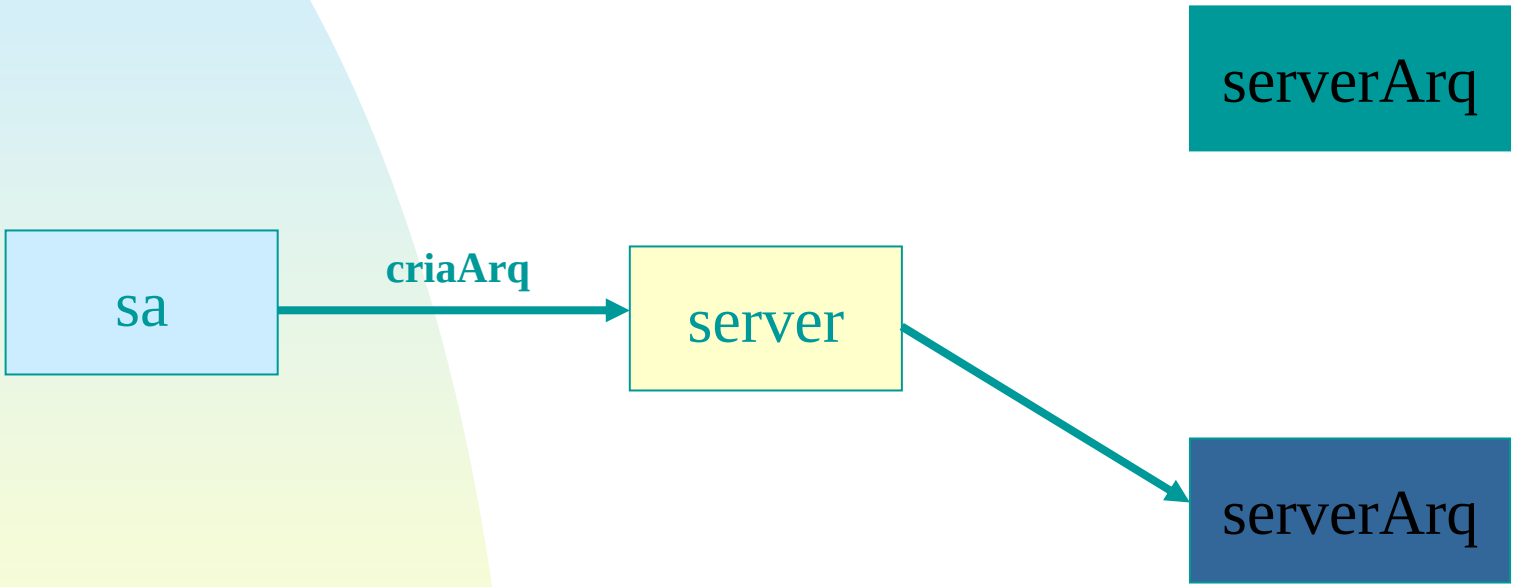
Escolha do servidor para armazenamento de arquivos



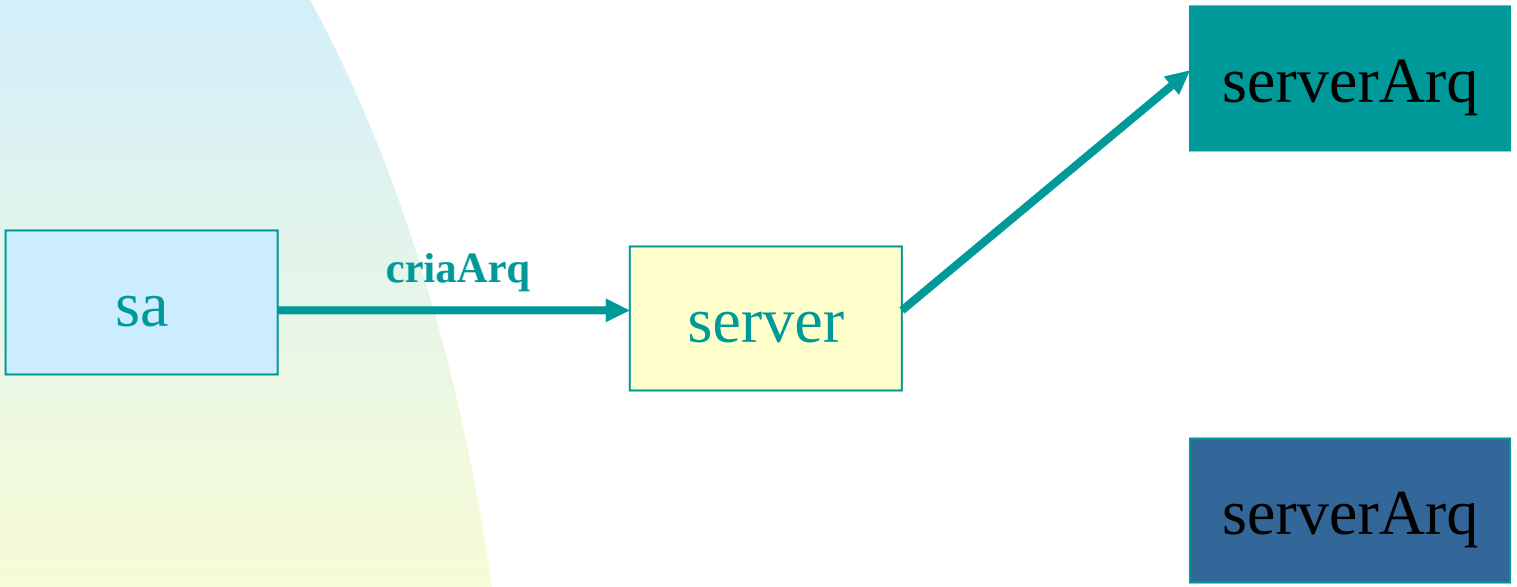
Escolha do servidor para armazenamento de arquivos



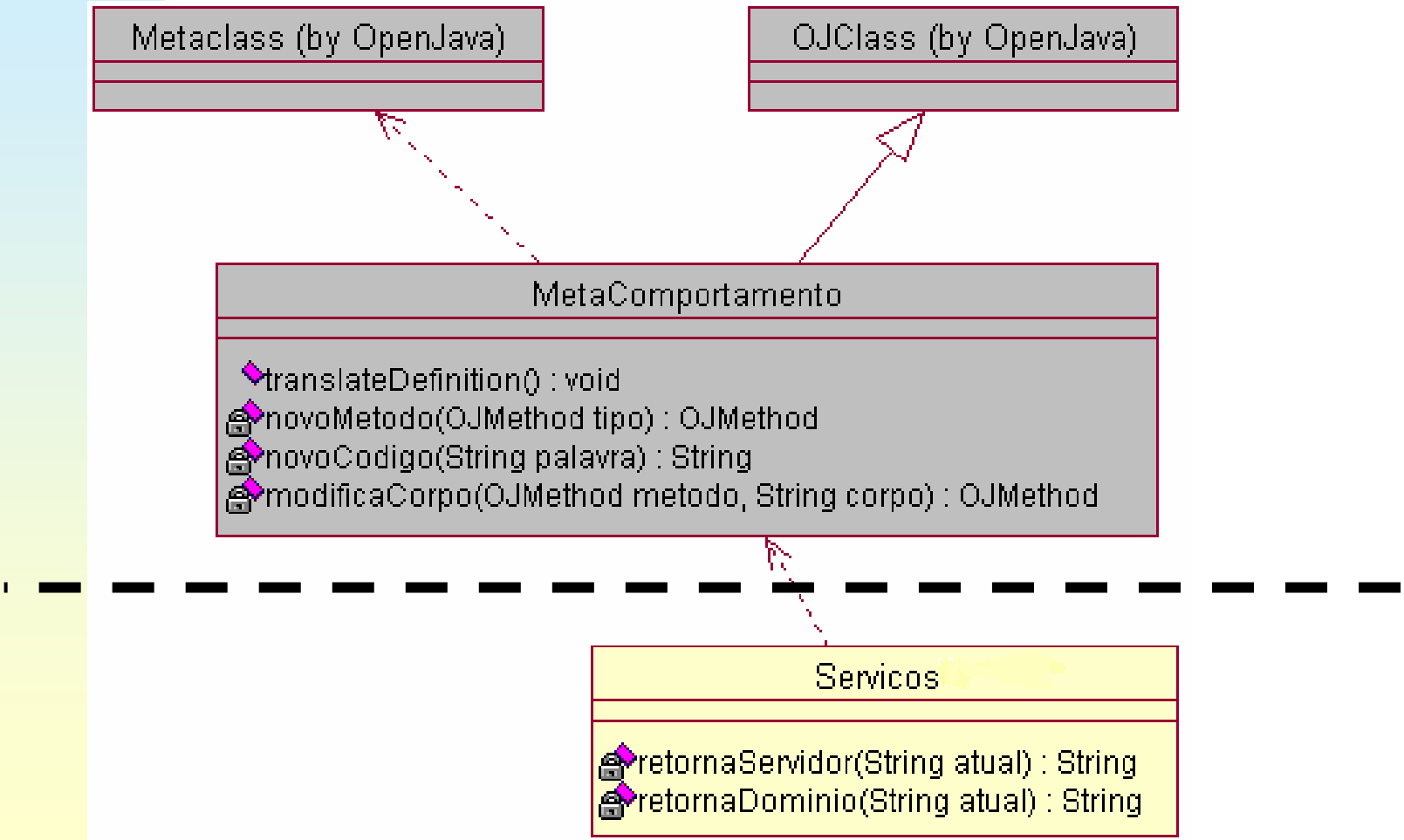
Escolha do servidor para armazenamento de arquivos



Escolha do servidor para armazenamento de arquivos



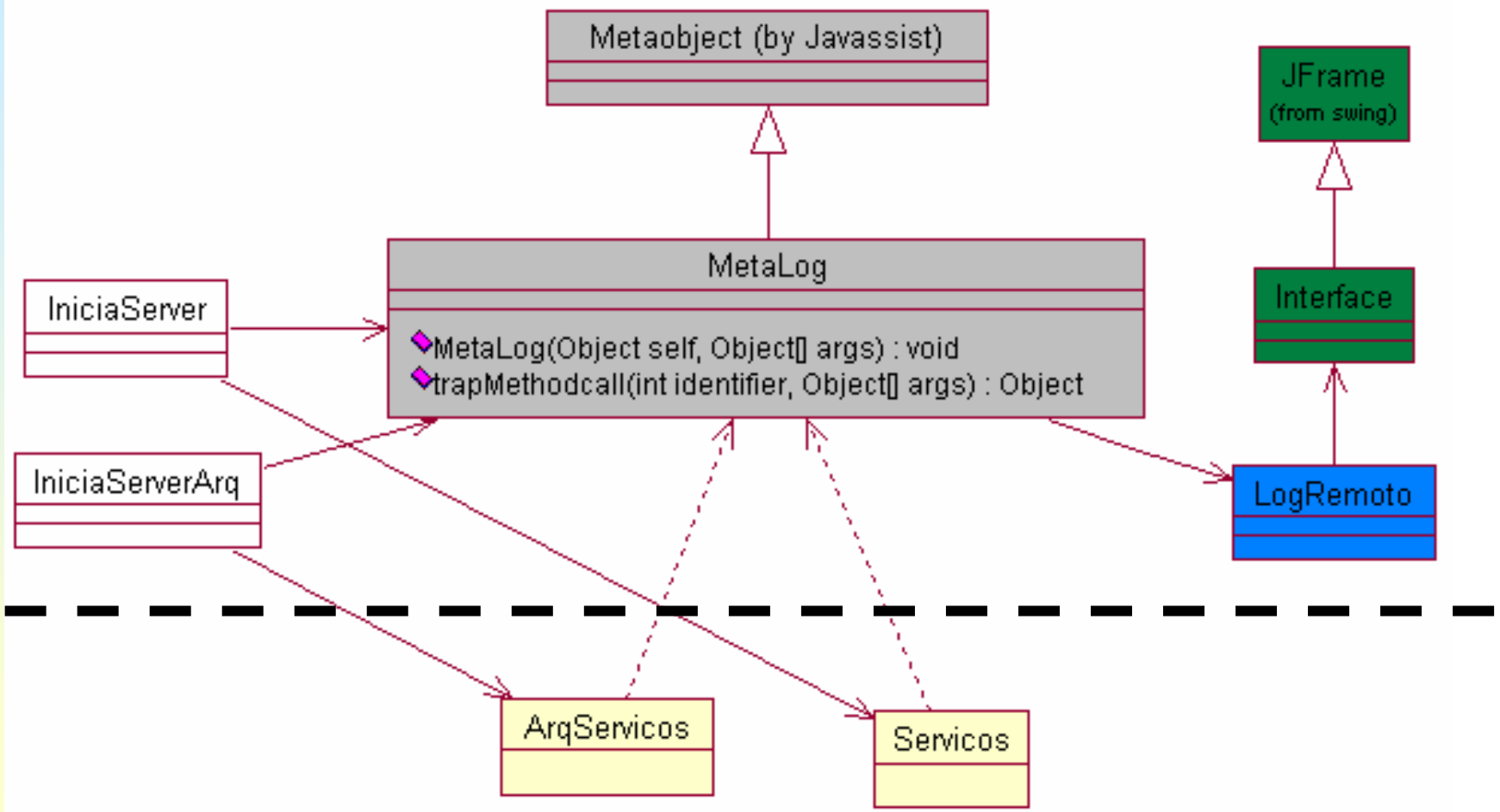
Escolha do servidor para armazenamento de arquivos (diagrama de classes)



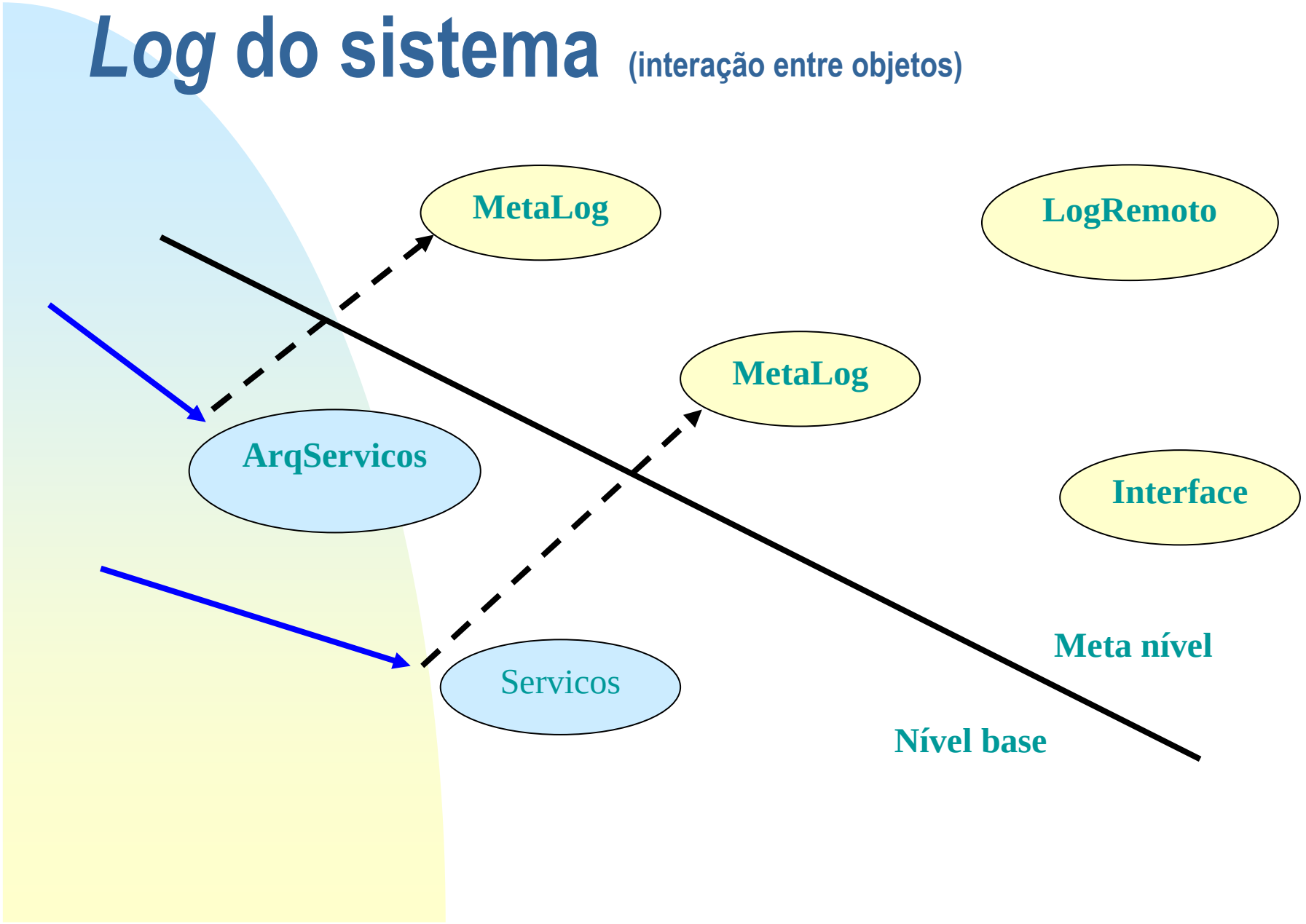
Escolha do servidor para armazenamento de arquivos (fragmento de código)

```
public String retornaServidor(java.lang.String oj_param0 ){
   if (espacoDisco( Dominio1 ) <= espacoDisco( Dominio2 )){
       return Servidor1;
   } else {
       return Servidor2;
   }
}

public long espacoDisco( java.lang.String oj_param0 ){
   //comportamento do método espacoDisco
   return espaco;
}

public String retornaDominio( java.lang.String oj_param0){
   if (espacoDisco( Dominio1 ) <= espacoDisco( Dominio2 )){
       return Dominio1;
   } else {
       return Dominio2;
   }
}
```

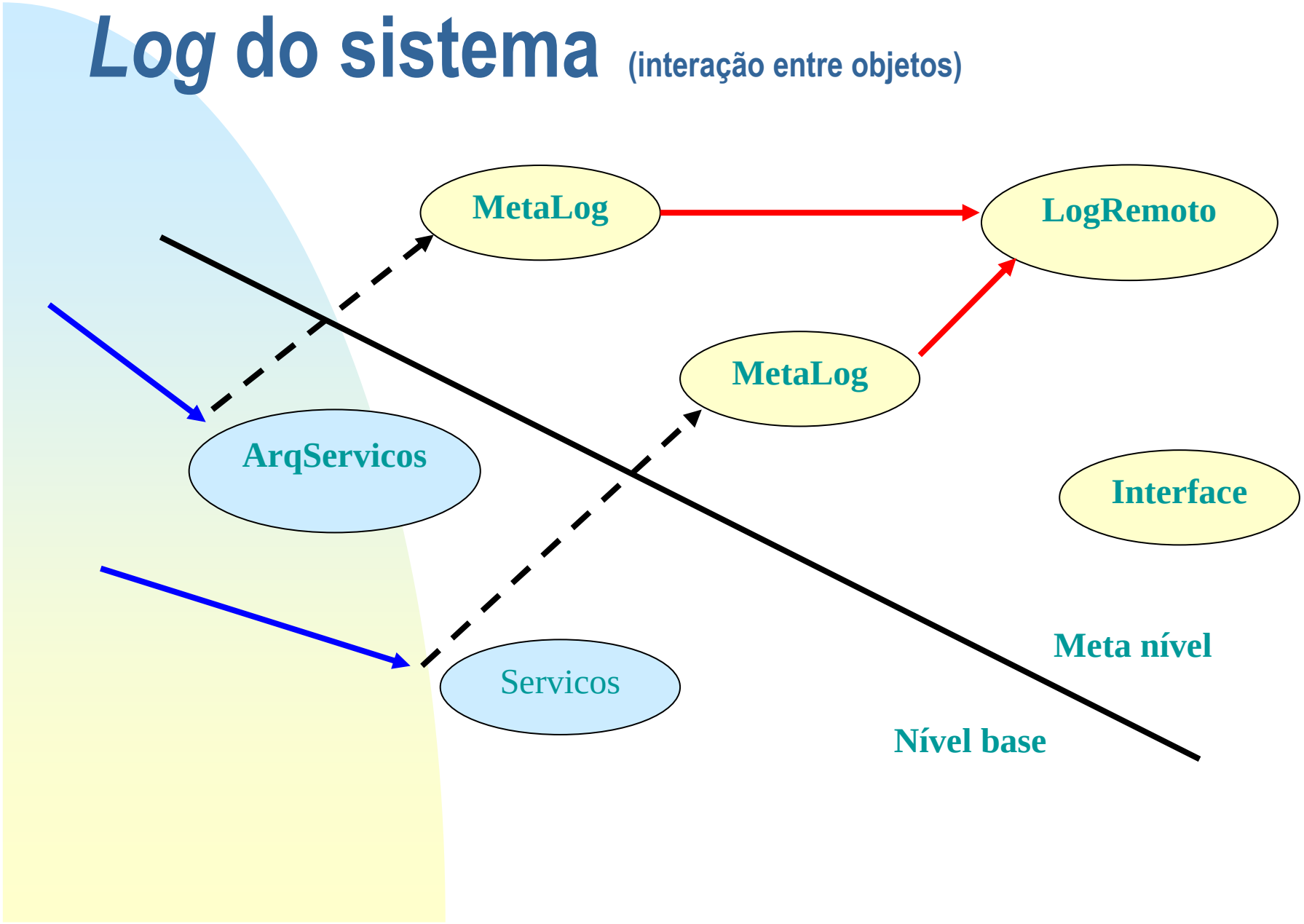
Log do sistema (diagrama de classes)



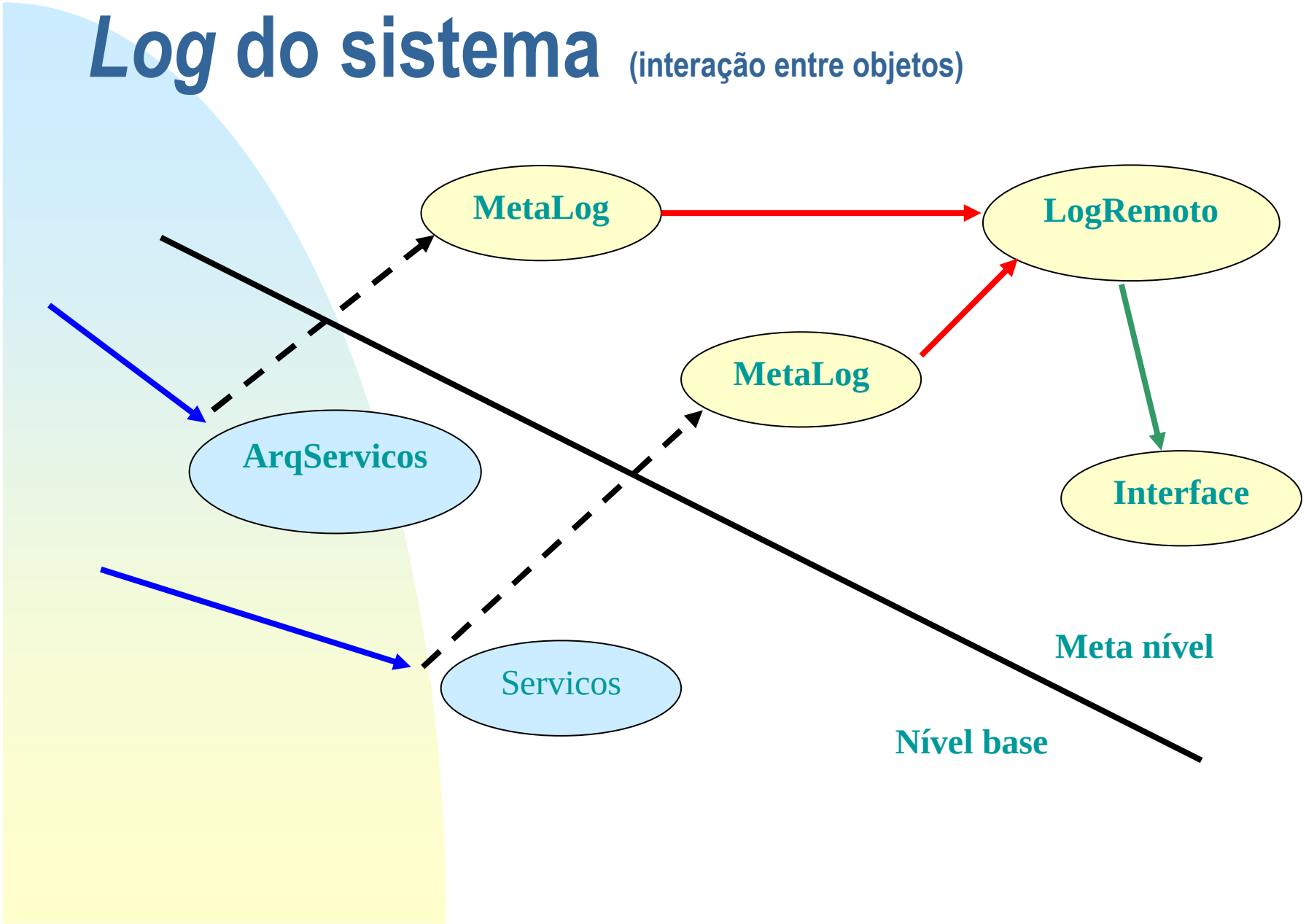
Log do sistema (interação entre objetos)



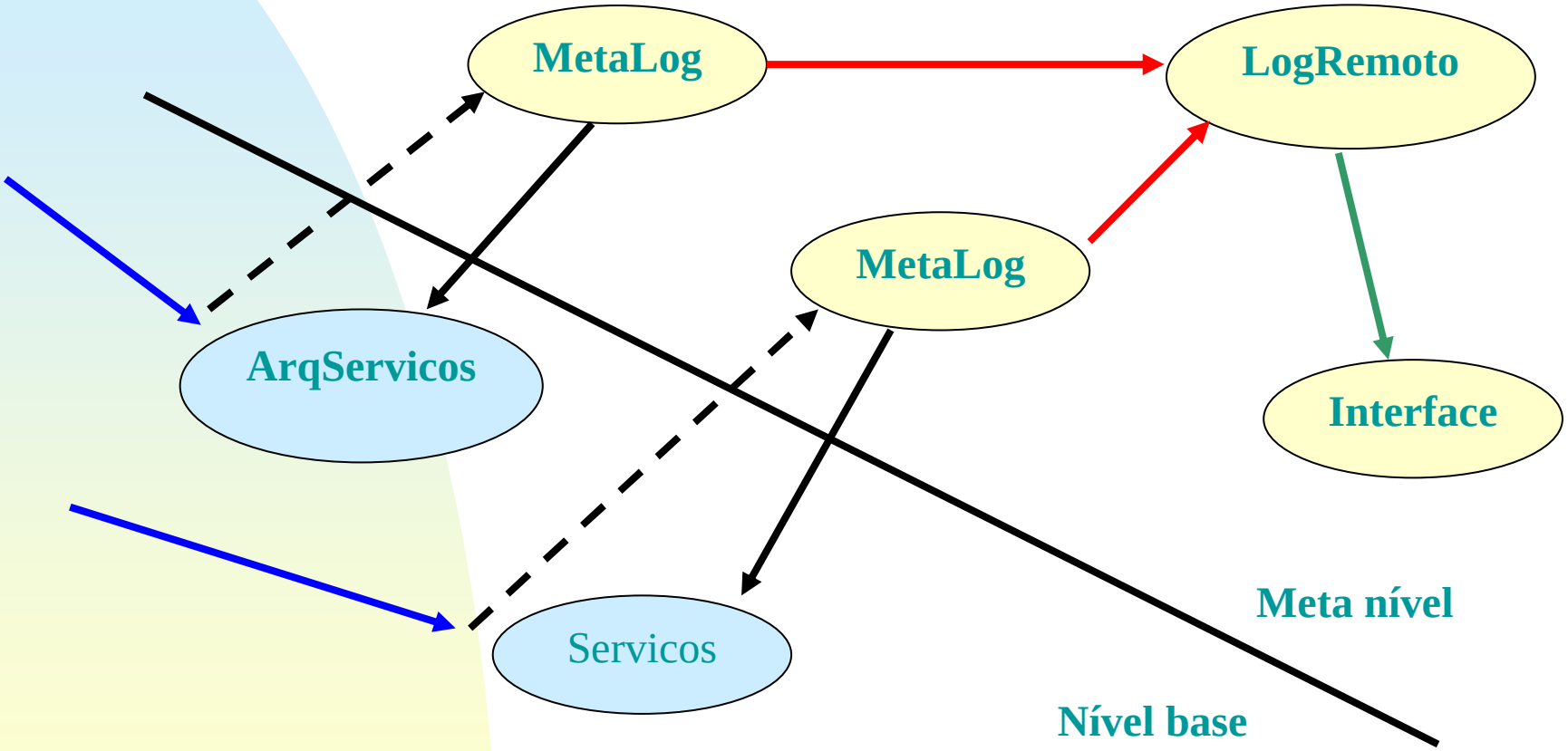
Log do sistema (interação entre objetos)



Log do sistema (interação entre objetos)



Log do sistema (interação entre objetos)



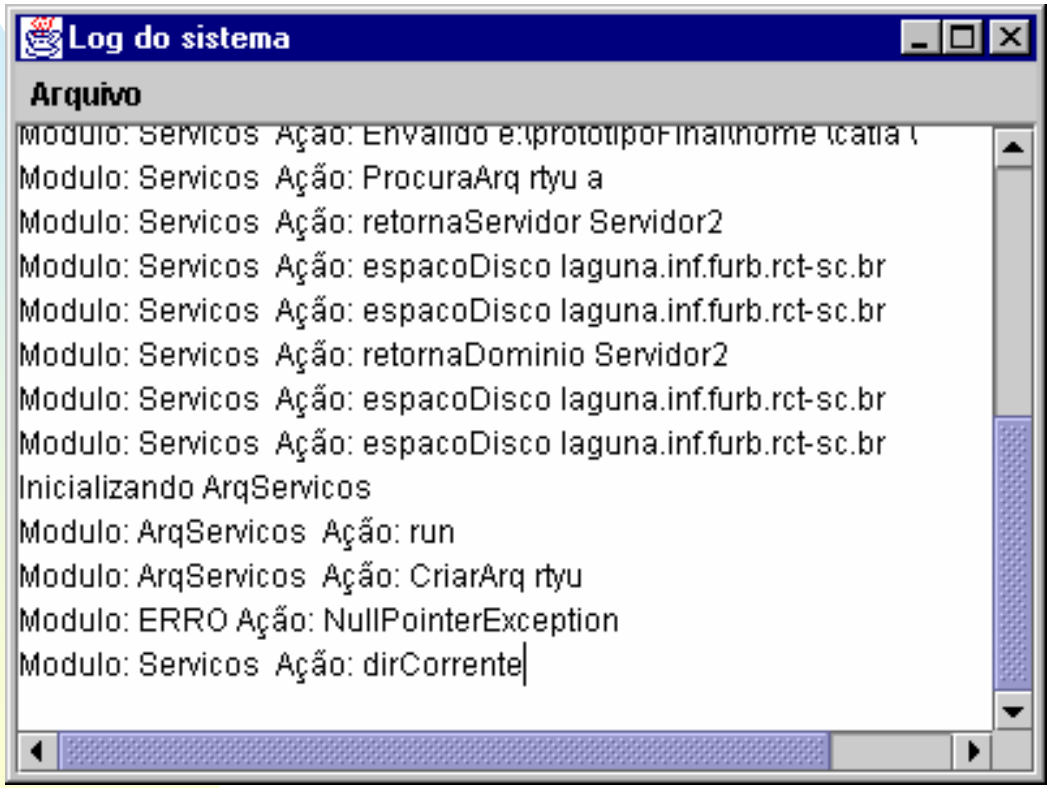
Log do sistema (fragmento de código fonte)

```
public class IniciaServer {
  public static void main(String[] args) throws Throwable {
    Loader classe = (Loader)IniciaServer.class.getClassLoader();
    ReflectLoader loaderServicos = new ReflectLoader();
    ReflectLoader loaderErro = new ReflectLoader();
    loaderServicos.makeReflective("Servicos", MetaLog.class ,ClassMetaobject.clas
    loaderErro.makeReflective("MetaErro", MetaLog.class ,ClassMetaobject.class);
    classe.addUserLoader(loaderServicos);
    classe.addUserLoader(loaderErro);
    classe.run("Server", args);
}}
```

Log do sistema (fragmento de código fonte)

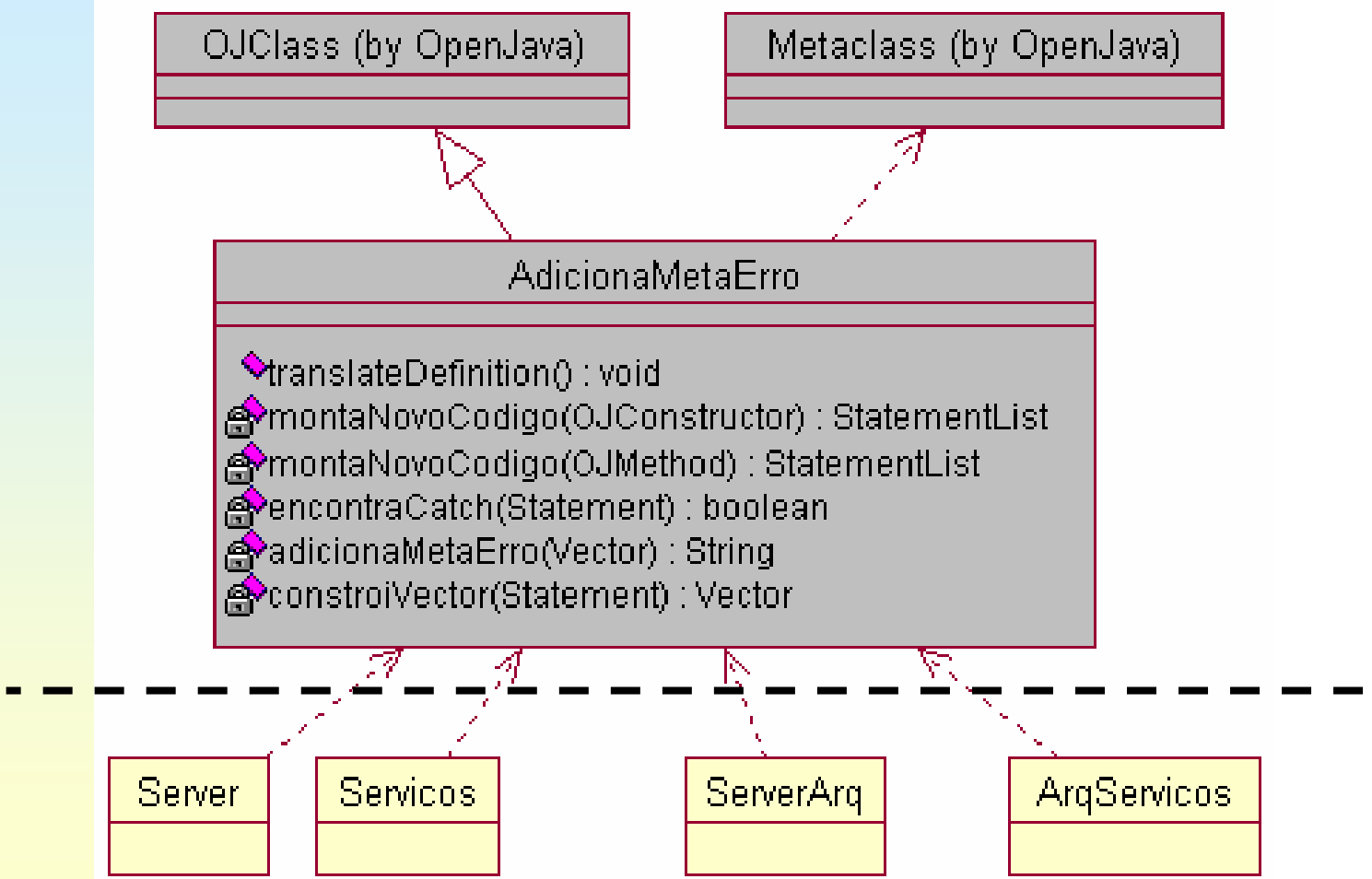
```
public MetaLog(Object self, Object[] args) throws CannotInvokeException {
  super(self, args);
  try{
    File arq = new File("vazio");
    String separador = arq.separator;
    DirTrabalho = args[1].toString();
    File arqhost = new File(DirTrabalho + separador + "host");
    RandomAccessFile rhost = new RandomAccessFile(arqhost,"r");
    String dadosHost = (rhost.readLine()).trim();
    StringTokenizer token = new StringTokenizer(dadosHost,"/");
    dadosHost = token.nextToken();
    dadosHost = token.nextToken();

    st = new Socket(dadosHost,1023);
    toServer = new PrintStream(st.getOutputStream());
    fromServer = new DataInputStream(st.getInputStream());
    toServer.println("Inicializando " +   self.getClass().getName()+"\n");

    toServer.close();
    fromServer.close();
    st.close();
 }catch(Exception e){ System.out.println("ERRO: "+e); }
}
```

Log do sistema (interface)



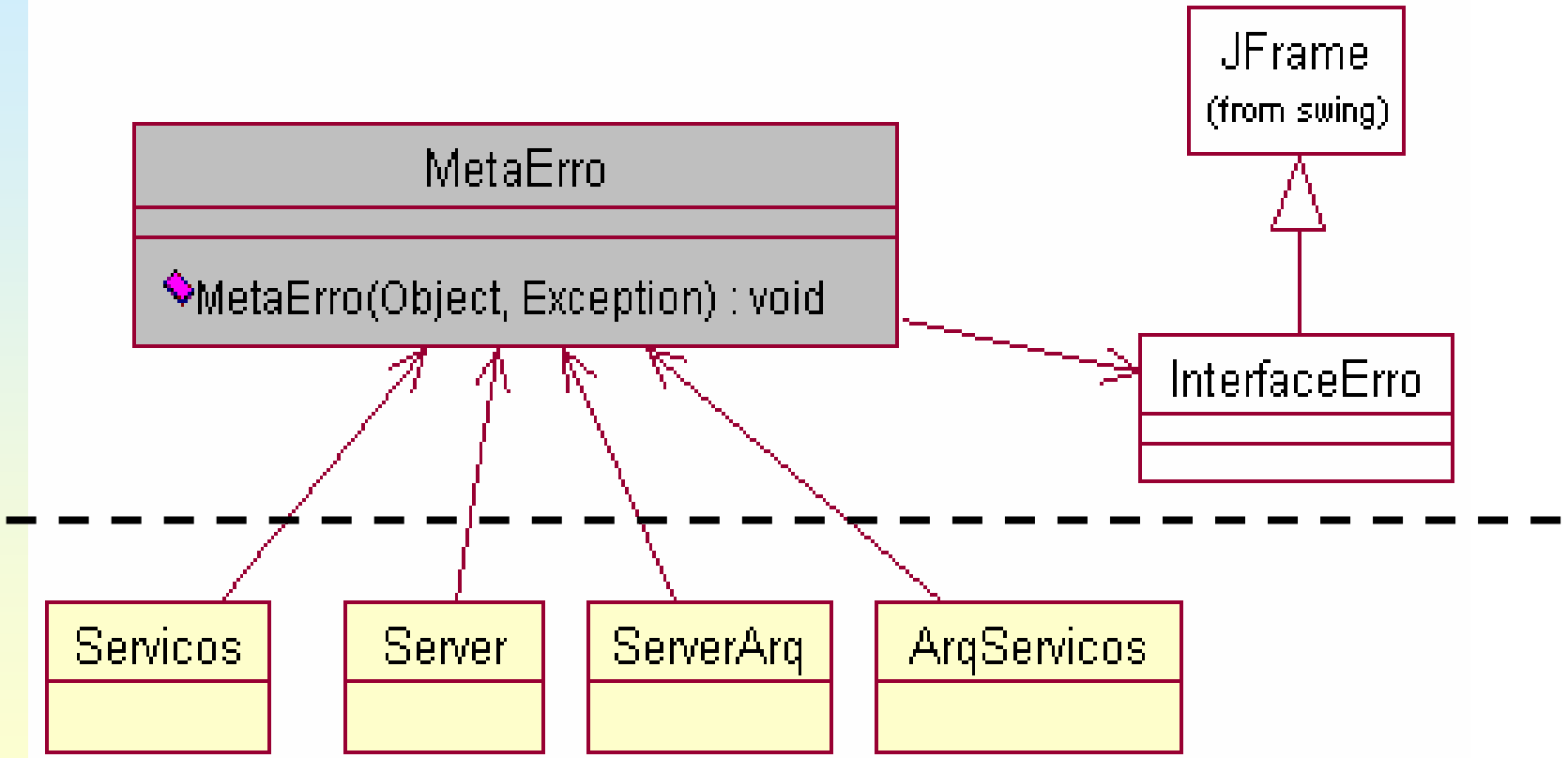
Módulo para tratamento de erros

(diagrama de classes - compilação)



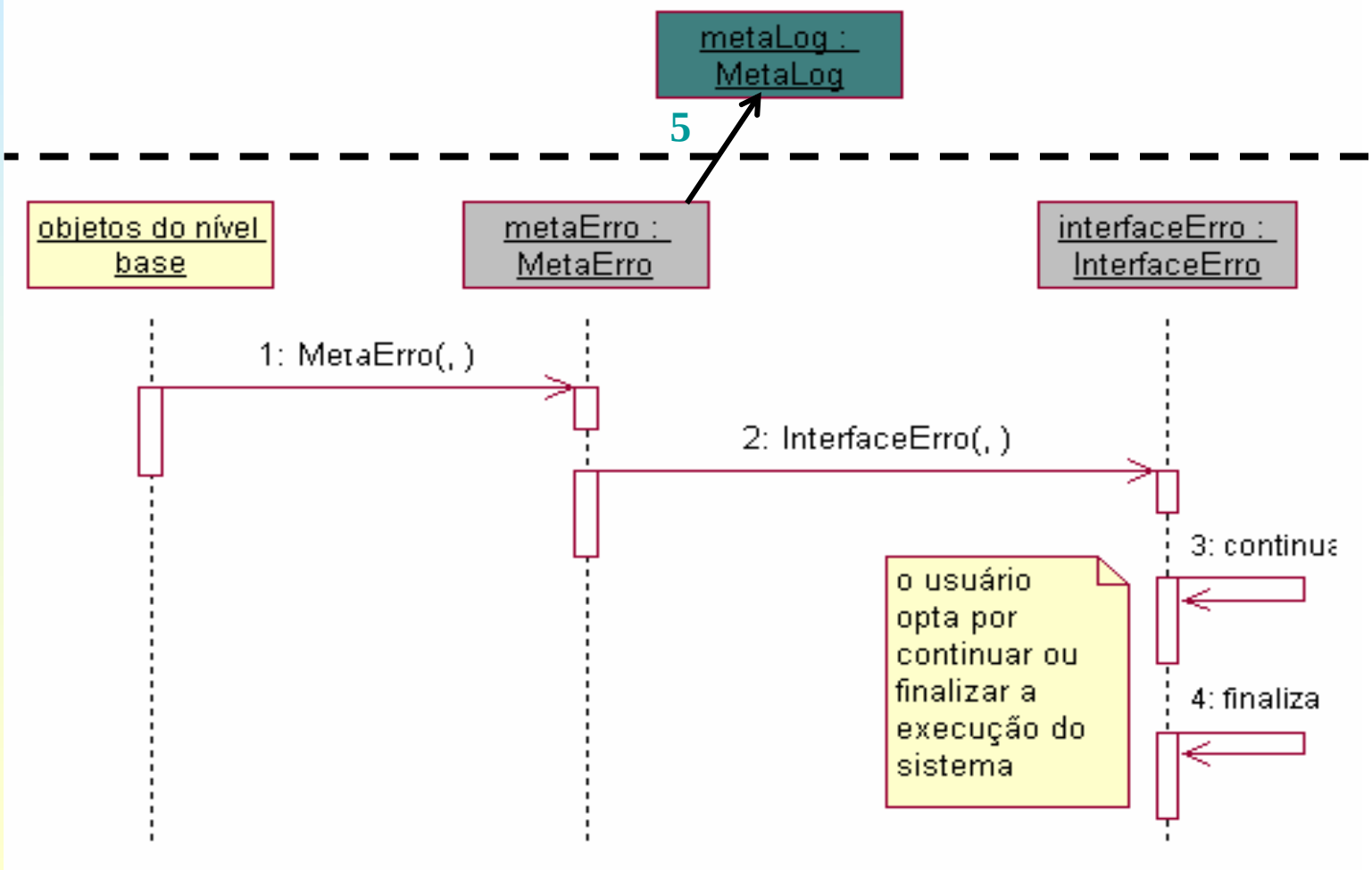
Módulo para tratamento de erros

(diagrama de classes - execução)



Módulo para tratamento de erros

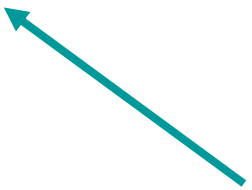
(diagrama de interação entre objetos)



Módulo para tratamento de erros

(fragmento de código fonte)

```
try{

  //codigo com possibilidades de erro
}catch (Exception e){

  new MetaErro(this,e);
}
```



Módulo para tratamento de erros (interface)

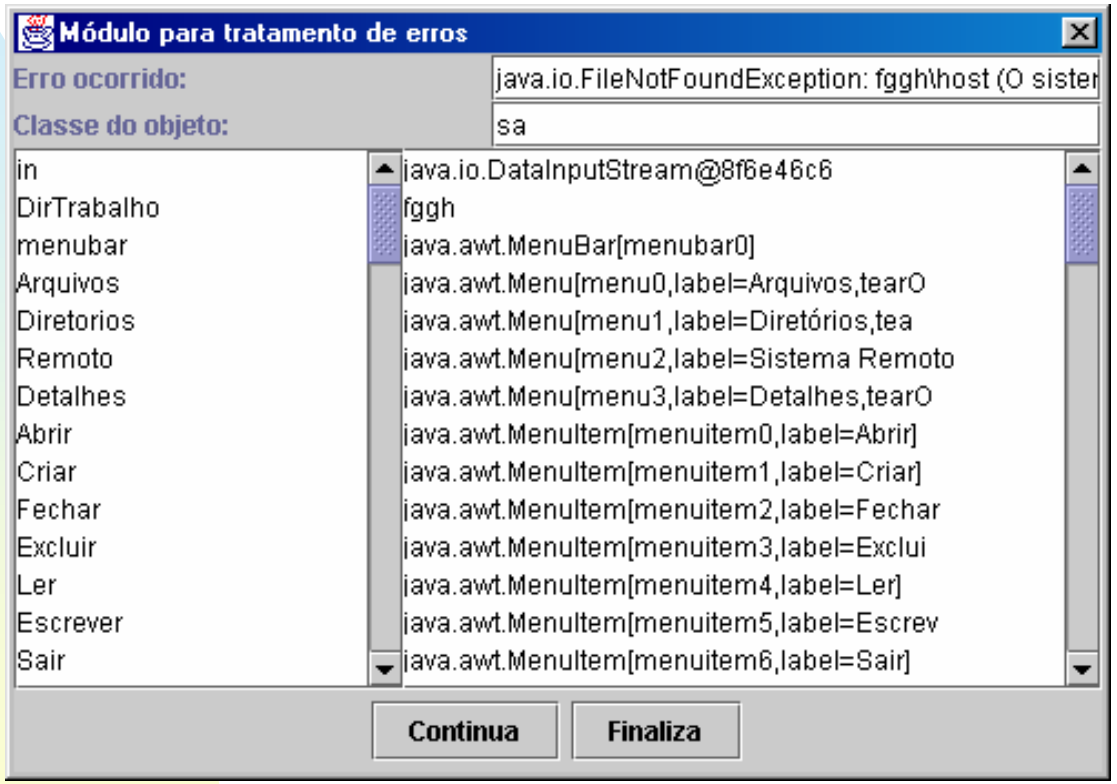


Diagrama de classes consolidados

(diagrama de classes - compilação)

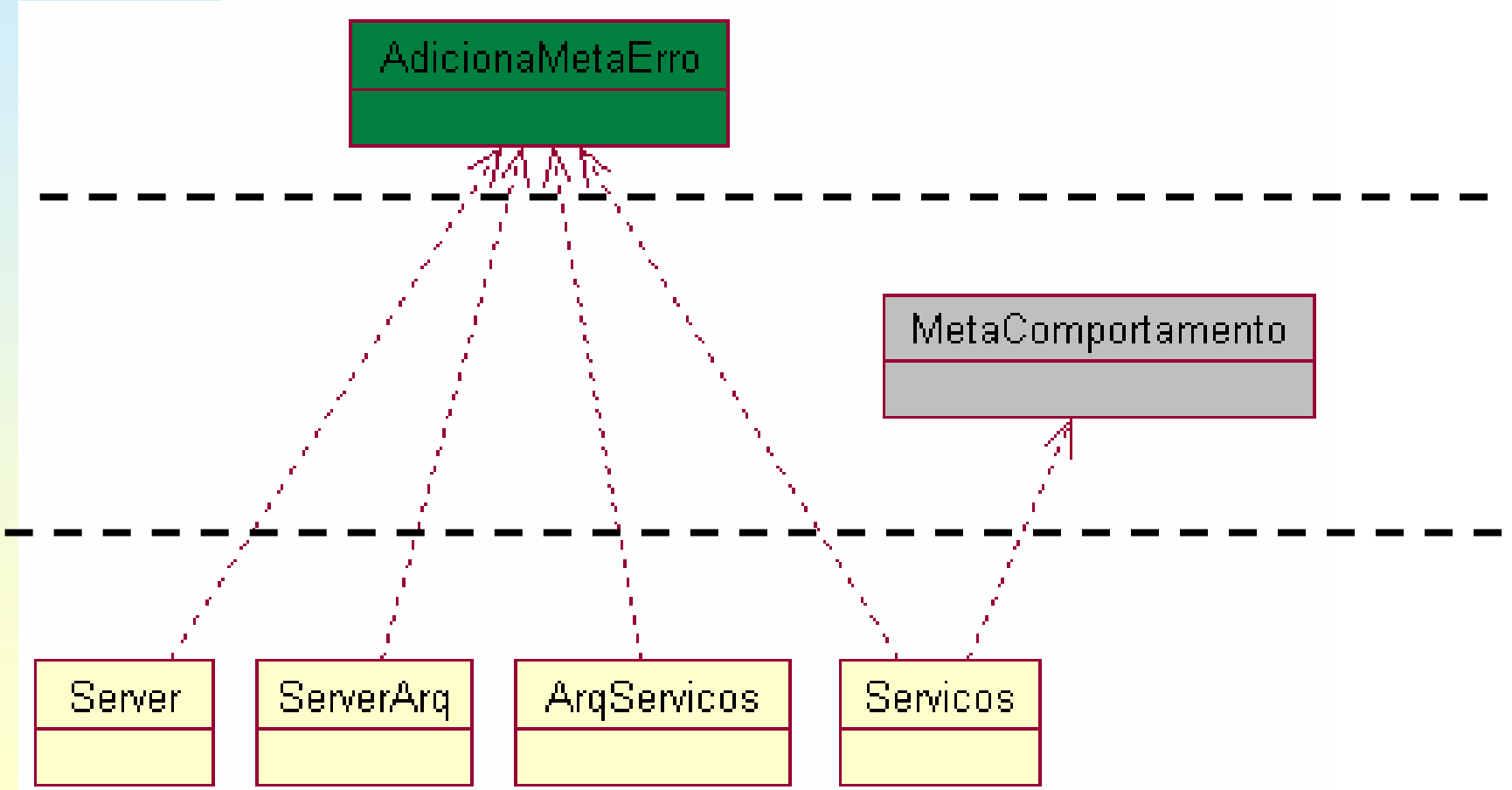
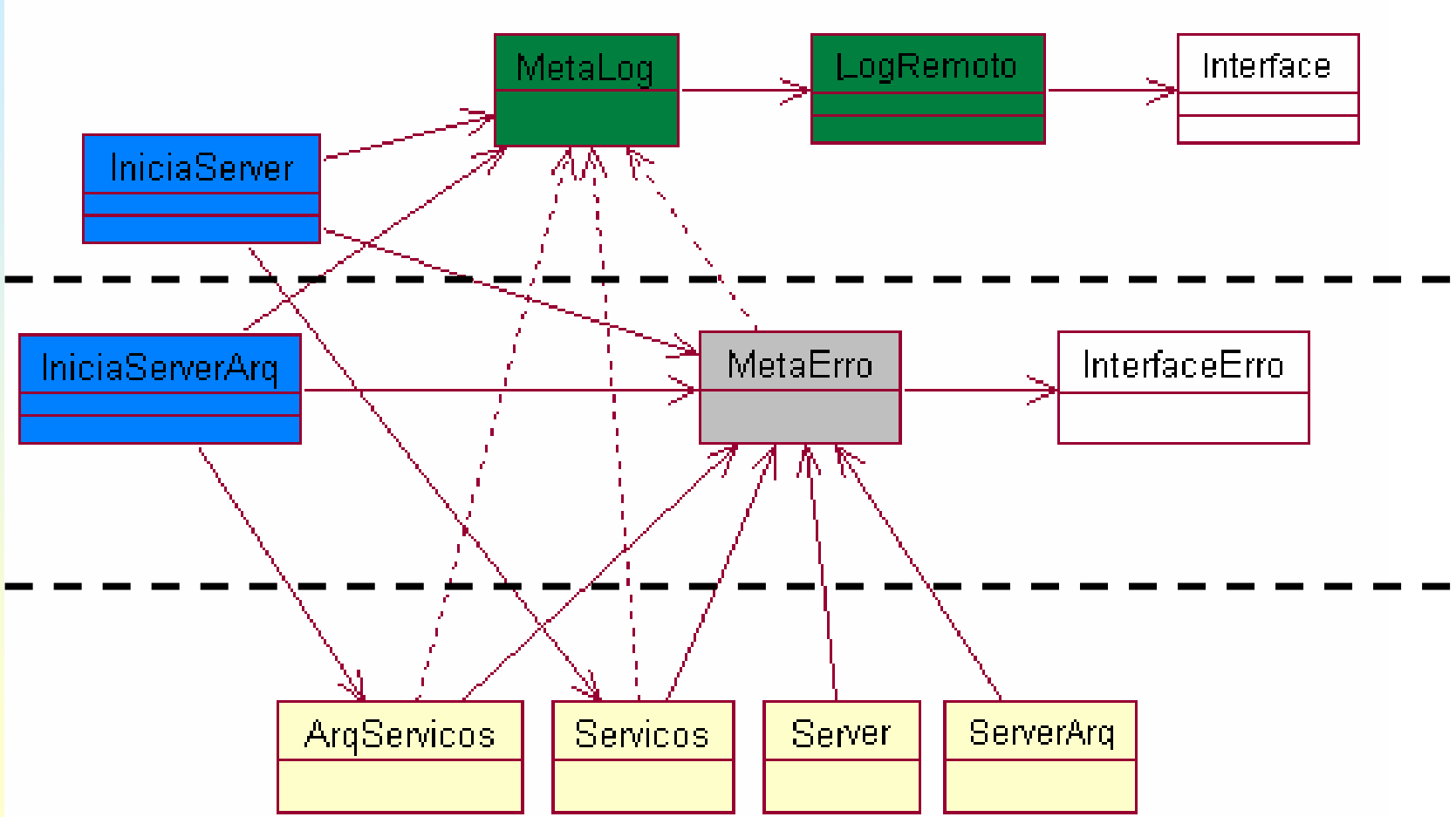


Diagrama de classes consolidados

(diagrama de classes - execução)



Conclusões e sugestões

- **Características, vantagens e desvantagens**
- **Exemplos e implementação**
  - Adaptação
  - Reusabilidade
  - Complexidade
- **Teoria x Prática**
- **Tempo de compilação x Tempo de execução**
- **OpenJava x Javassist**

Conclusões e sugestões

- **Protótipo implementado**
  - Objetivos
  - Restrições
- **Sugestões**
  - Estudar e avaliar ambientes reflexivas
  - Implementação de objetos persistentes, réplicas de servidores, armazenamento de transações
  - Sistemas tolerantes a falhas / Tempo real
  - Módulo de tratamento de erros (introspecção/intercessão)

Referências bibliográficas

- [KIC1991] KICZALES, G. **The art of the metaobjects protocol.** Cambridge: MIT Press, 1991.
- [KIC1996] KICZALES, G. **Open implementations and metaobject protocol.** Relatório de pesquisa da Xerox Corporation.
- [CHI2000] CHIBA, S. **Welcome to Javassist 0.6.** Http://www.hlla.is.tsukuba.ac.jp
- [LIS1997] LISBÔA, M. **Arquiteturas de meta-nível.** Tutorial XI Simpósio Brasileiro de Engenharia de Software. Fortaleza, CE.
- [MAE1987] MAES, P. **Concepts and experiments in computacional reflection.** ACM SIGPLAN Notices.
- [TAT2000] TATSUBORI, M. **Welcome to OpenJava 1.0.** Http://hlla.is.tsukuba.ac.jp/~mich/openjava/