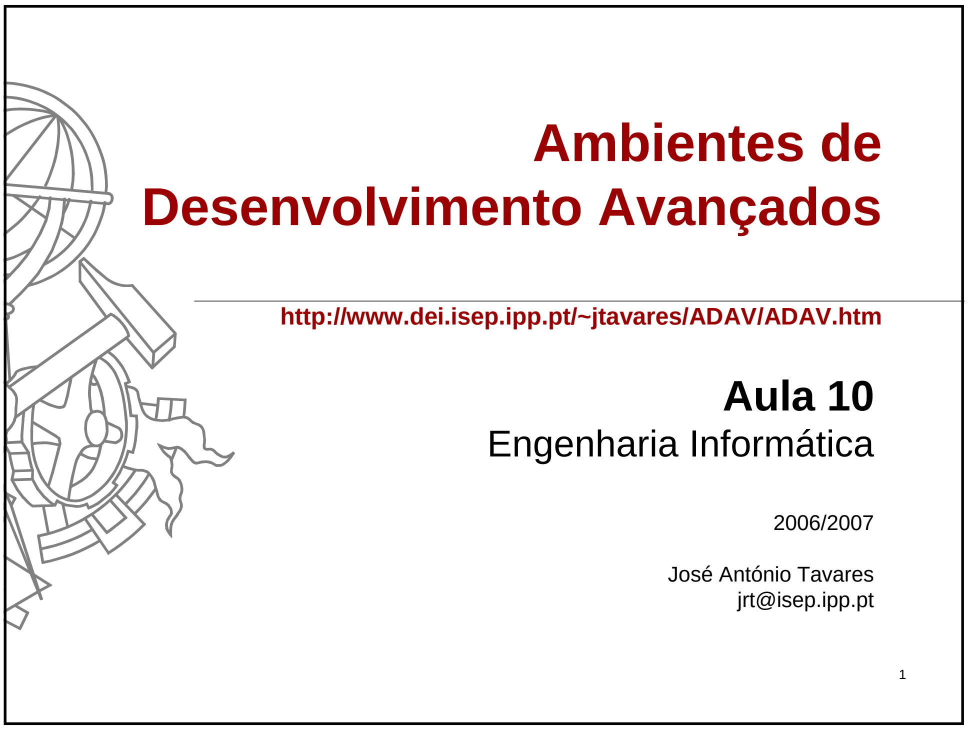


Ambientes de Desenvolvimento Avançados

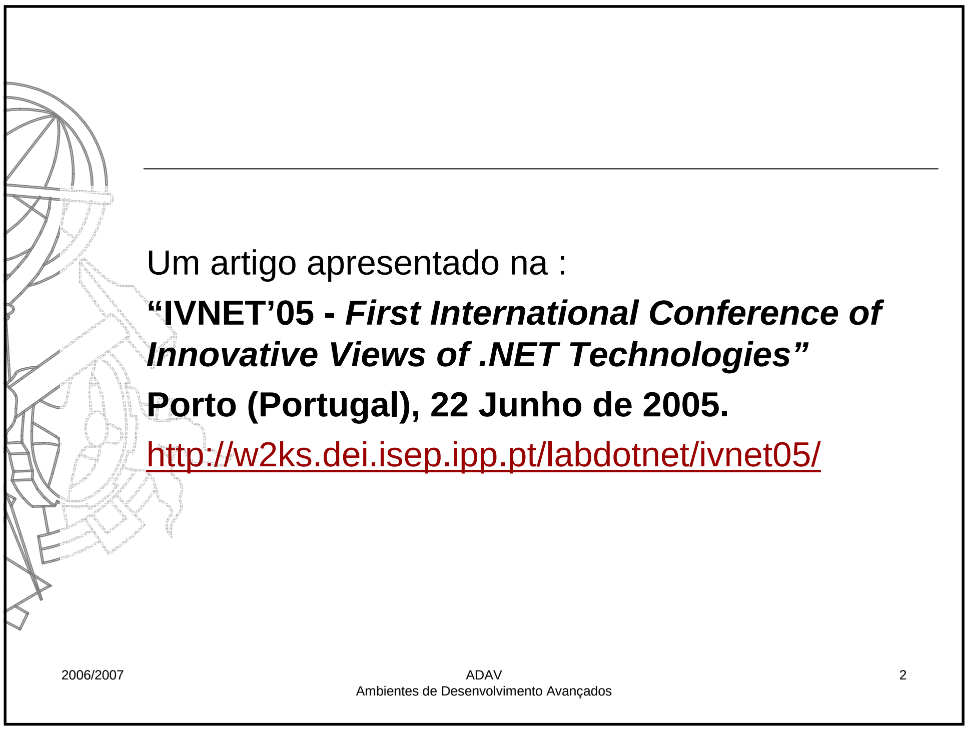
http://www.dei.isep.ipp.pt/~jtavares/ADAV/ADAV.htm

Aula 10

Engenharia Informática

2006/2007

José António Tavares
jrt@isep.ipp.pt

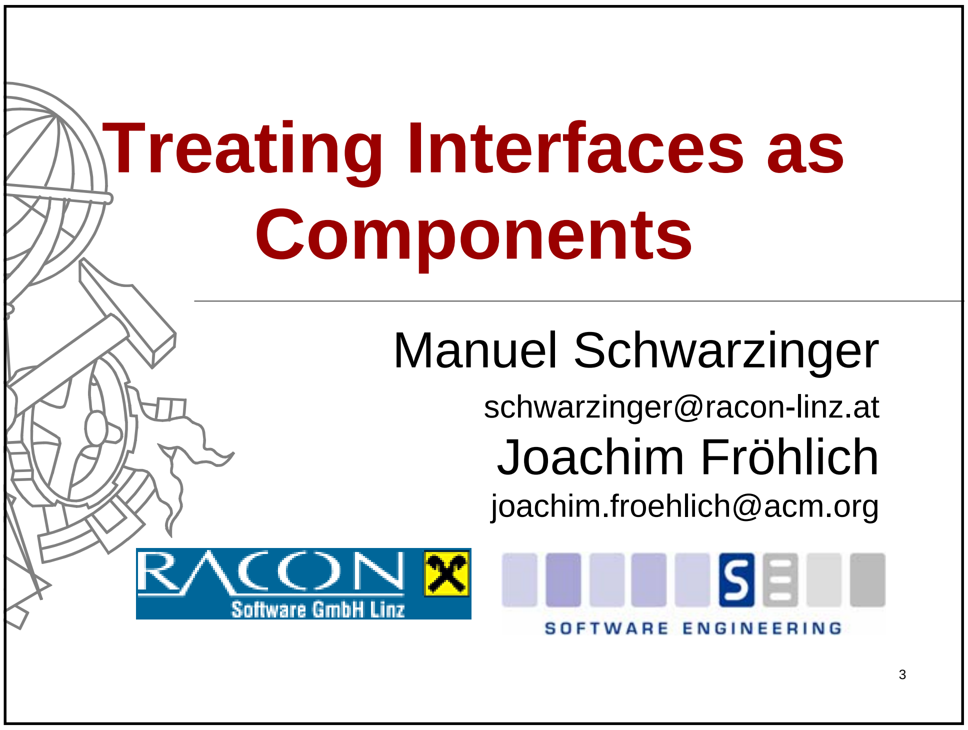


Um artigo apresentado na :

"IVNET'05 - *First International Conference of Innovative Views of .NET Technologies*"

Porto (Portugal), 22 Junho de 2005.

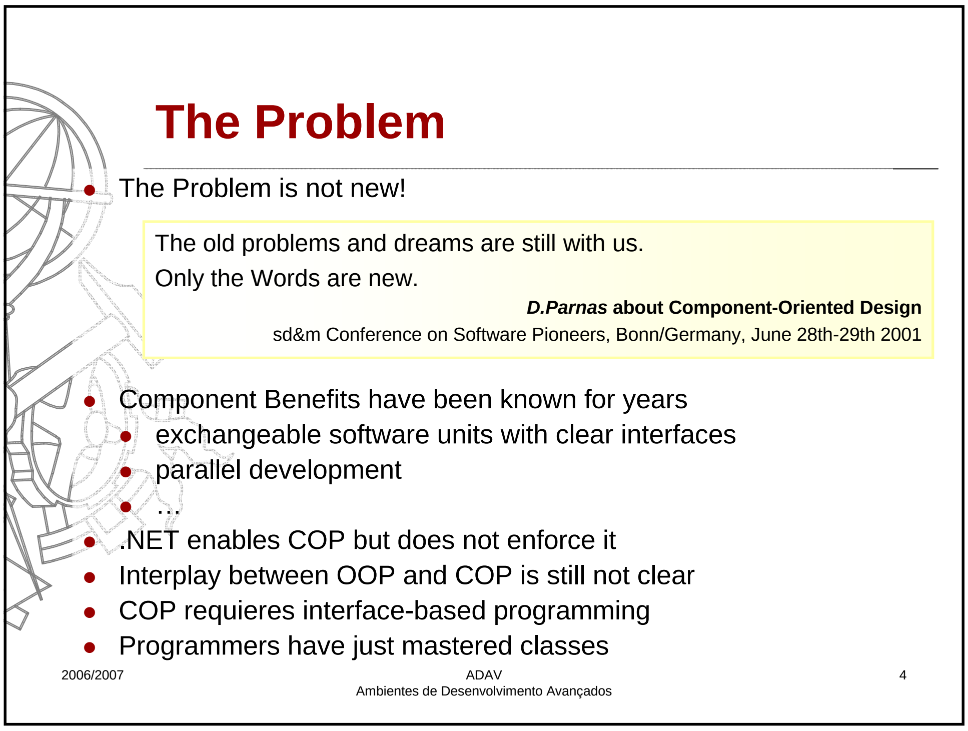
http://w2ks.dei.isep.ipp.pt/labdotnet/ivnet05/



Treating Interfaces as Components

Manuel Schwarzinger
schwarzinger@racon-linz.at
Joachim Fröhlich
joachim.froehlich@acm.org

3



The Problem

- The Problem is not new!

> The old problems and dreams are still with us.
> Only the Words are new.
>
> ***D.Parnas*** **about Component-Oriented Design**
> sd&m Conference on Software Pioneers, Bonn/Germany, June 28th-29th 2001

- Component Benefits have been known for years
  - exchangeable software units with clear interfaces
  - parallel development
  - …
- .NET enables COP but does not enforce it
- Interplay between OOP and COP is still not clear
- COP requieres interface-based programming
- Programmers have just mastered classes

2006/2007 ADAV
Ambientes de Desenvolvimento Avançados 4

The Goal

Components that

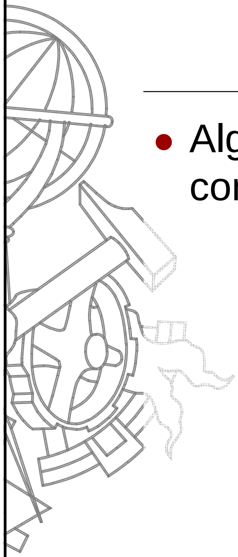
- are easy to use and implement
  … even for highly specialized application domains
- offer sharp interfaces
  … also with object-oriented techniques, but without classes
- adhere to the information hiding principle
  … of course
- decouple functional components completely
  … well, bind them well-defined
- allow (re)configurations
  … after development, before and during runtime

The Goal (2)

We have a dream of

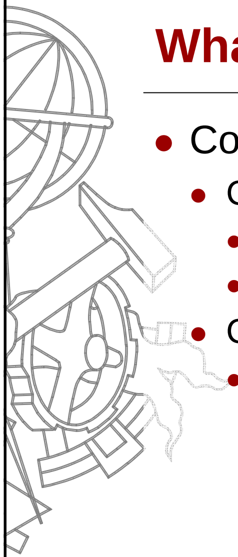
- sharp component (interface) specifications
- systematic unit, component and integration tests
- separate component implementations
- flexible configurations of system families
- hiding specific, platform-dependent component models

➔ We look for stability to be agile!

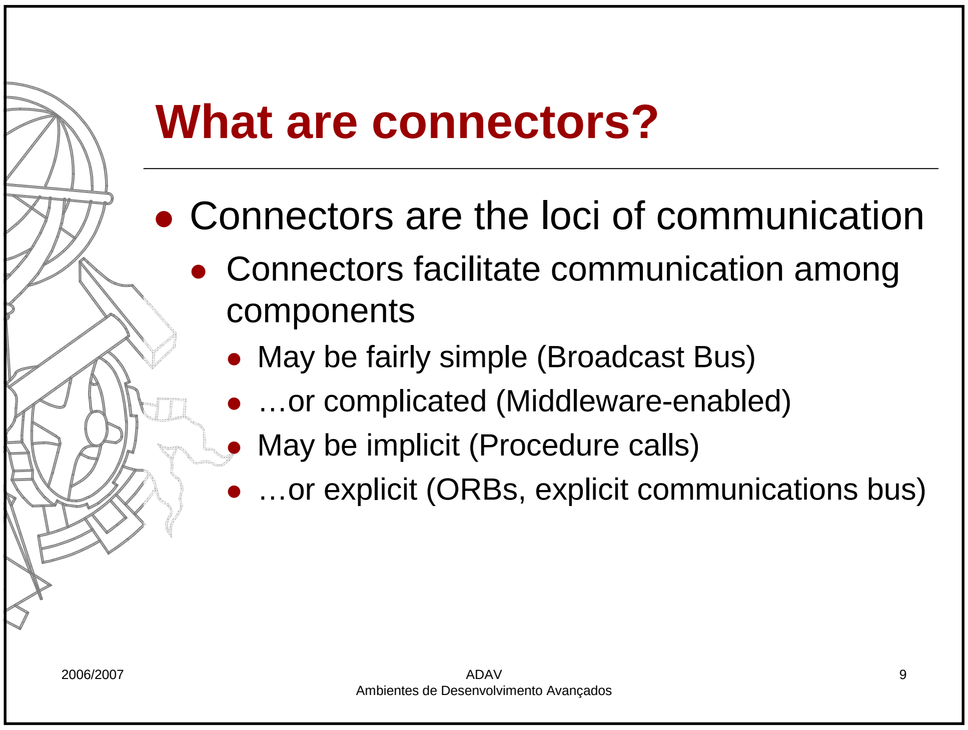


- Algumas definições/revisões de conceitos antes de continuar ...

What are components?



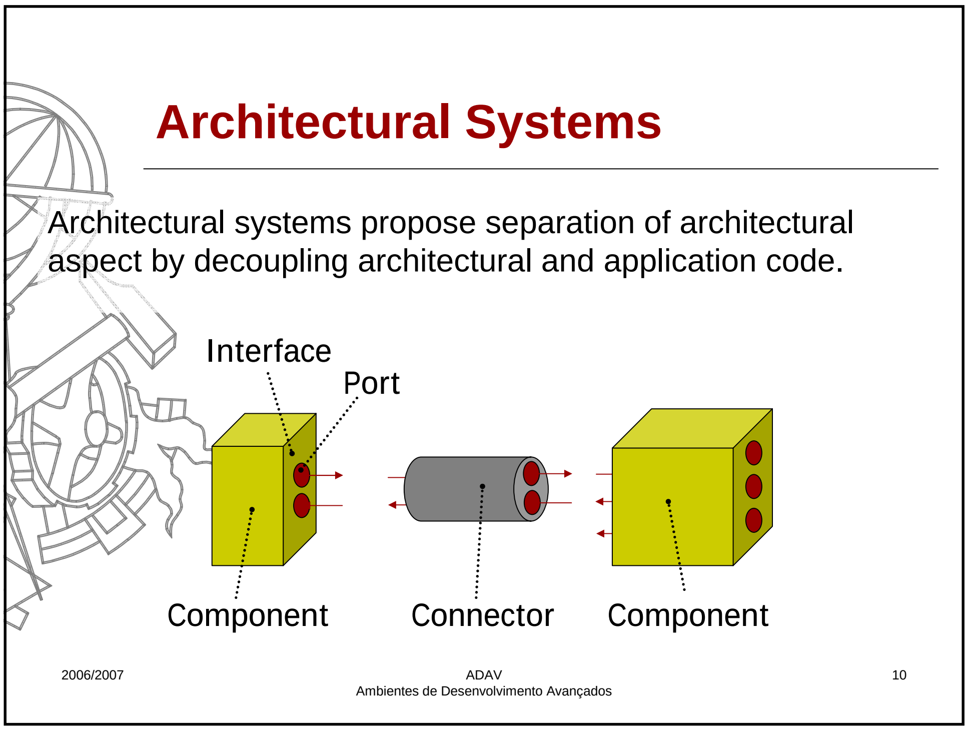
- Components are the loci of computation
  - Components "do the work" in the architecture
    - May be coarse-grained (an editor)
    - …or fine grained (a clock emitting ticks)
  - Components have interfaces
    - Provided *and* required in this class



What are connectors?

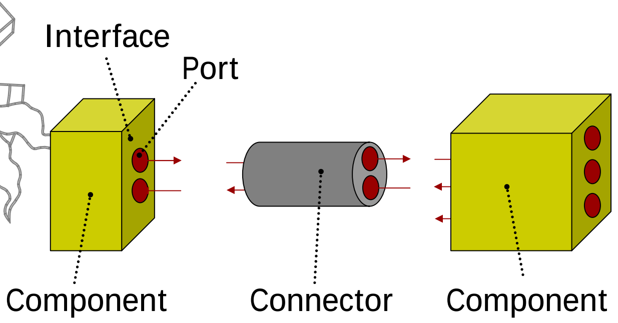
- Connectors are the loci of communication
  - Connectors facilitate communication among components
    - May be fairly simple (Broadcast Bus)
    - …or complicated (Middleware-enabled)
    - May be implicit (Procedure calls)
    - …or explicit (ORBs, explicit communications bus)

2006/2007 ADAV Ambientes de Desenvolvimento Avançados 9



Architectural Systems

Architectural systems propose separation of architectural aspect by decoupling architectural and application code.



2006/2007 ADAV Ambientes de Desenvolvimento Avançados 10

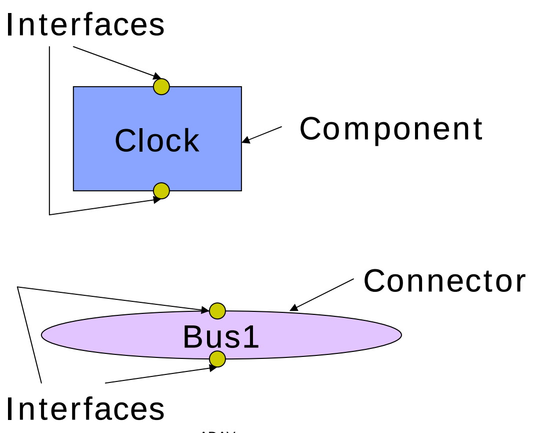
What are interfaces?

- Interfaces are the connection points on components and connectors
  - They define where data may flow in and out of the components/connectors
    - May be loosely specified (events go in, events go out)
    - …or highly specified (event protocols across the interface in CSP)

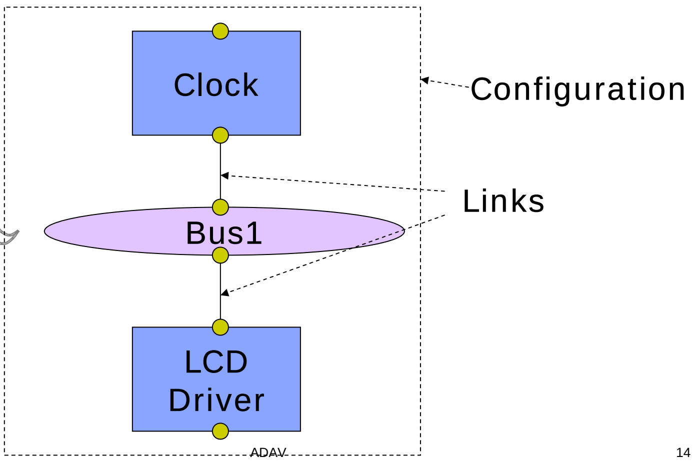
What are configurations?

- Configurations are arrangements of components and connectors to form an architecture.
- *Links* indicate connections between components and connectors
  - “If links had semantics, they’d be connectors.” --Dashofy’s 8th law of architectures

Examples:

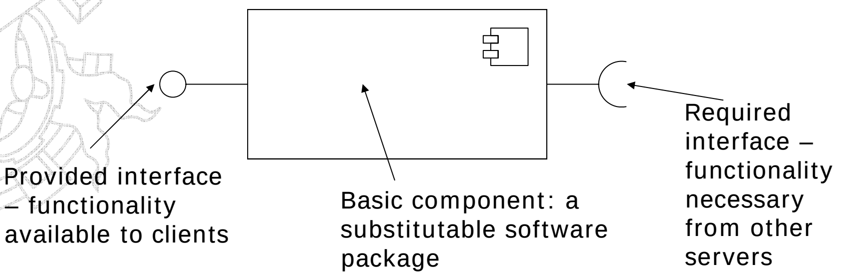


Examples:



Components in UML 2

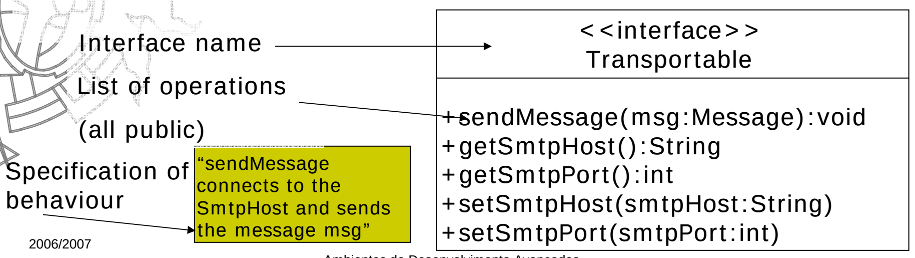
- The UML 2 has several ways of visually representing components.
- We will use the following visual notation for basic components


Provided interface – functionality available to clients
Basic component: a substitutable software package
Required interface – functionality necessary from other servers

- Let's investigate our definition of a component in more depth

Interfaces in UML 2

- Two ways
  - Component usage: lollipop and socket notation
  - Detailed signature and specification
- Detailed specification is given the same way as for UML classes, but with <<interface>> keyword at the top
- **Always provide detailed interface specifications in designs *before* referring to interface name in component diagrams**


Interface name
List of operations
(all public)
Specification of behaviour
"sendMessage connects to the SmtpHost and sends the message msg"
<<interface>>
Transportable
+sendMessage(msg:Message):void
+getSmtpHost():String
+getSmtpPort():int
+setSmtpHost(smtpHost:String)
+setSmtpPort(smtpPort:int)

UML Syntax for operations

Detailed UML interface specifications – recall the syntax for defining operations and input/output types

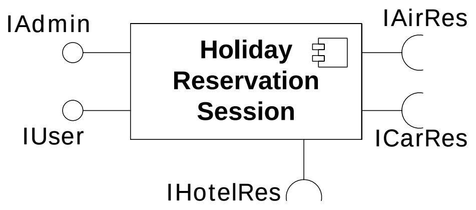
visibility name (parameter list) : return-type-expression

```
+ assignAgent (a : Agent) : Boolean
```

- visibility: public (+), protected (#), private (-)
  - Visibility is always public for interfaces
- *name*: string
- *parameter list of input arguments*
- *return-type-expression*

Example UML 2 component

- Holiday reservation component
  - 2 different kinds of clients: an ordinary user and a system administrator: 2 provided interfaces, one for each client
  - The component uses 3 different components to do its work, Hotel reservation, Car reservation and Air reservation: 3 required interfaces, one for each required component

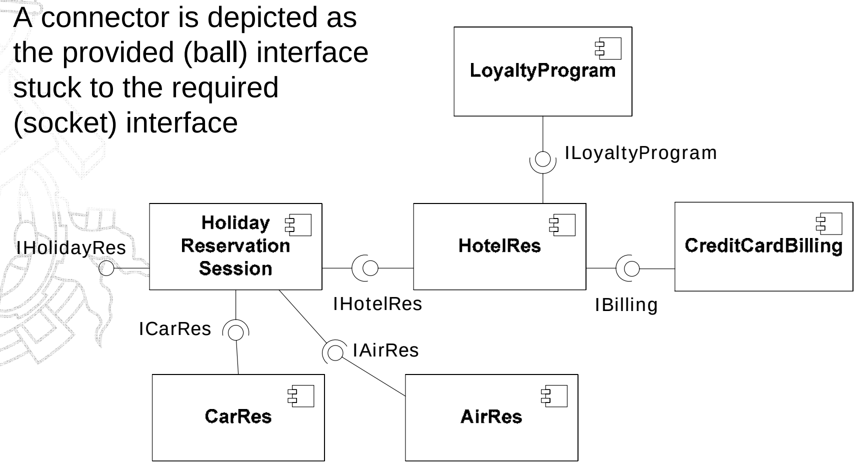


Connections

- UML 2 provides two kinds of connectors
  - Delegation connectors
  - Assembly connectors
- An assembly connector is a connection between two components that defines that one component provides the services that another component requires
- Defines communication between components
  - providing component acts as the server to the requiring component
  - The requiring, client component communicates by calling the methods of the provided interface on the server, providing component

Connectors in UML 2

- A connector is depicted as the provided (ball) interface stuck to the required (socket) interface



- Voltando ao ...
  “Treating Interfaces as Components“

The Idea (1)

Interfaces carry software architectures, not components

- stable interfaces lead to stable architectures
- … still not new, but still newsworthy

Connectors are Components with only systems of semantically related interfaces

… well, with almost only interfaces

Connectors may be seen as clutches with

- driving parts: functional *client* components
- driven parts: functional *provider* components

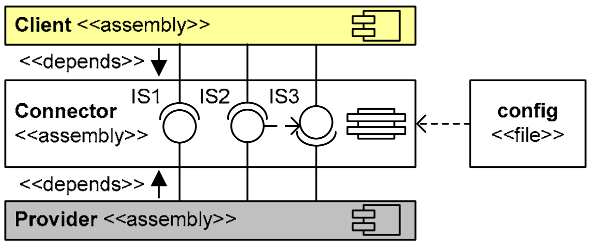
Connectors may offer services

… like antishock protection and gear changing

… i. e. logging, profiling, protocol checks …

The Idea (2)

Schematic representation of a connector with a system of call and callback interfaces

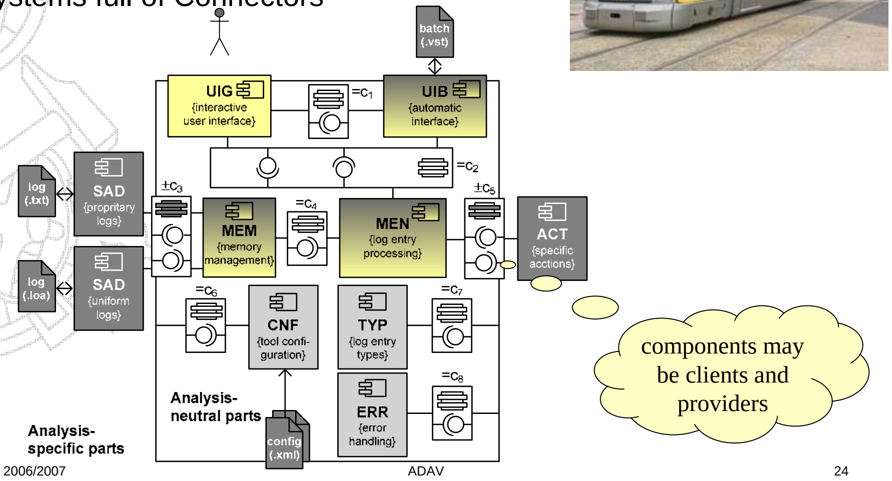


Characteristics:

- functional components do *not* depend on each other
- functional components depend on connectors
- connectors do *not* depend on functional components

The Idea (3)

Systems full of Connectors



A Solution

1. Design the Interfaces
2. Compile functional components and interfaces independently

```
csc /out:Connector.dll /t:library ...
csc /out:Provider.dll /t:library /r: Connector.dll
csc /out:Client.exe /t:exe /r:Connector.dll ...
```

3. Plug together functional components
   … before runtime e.g. simply with app.config

```
<configuration>
  <appsettings>
    <add key="ConnectorSubsystemX" value ="Provider.dll"/>
    ...
    </appsettings>
</configuration>
```

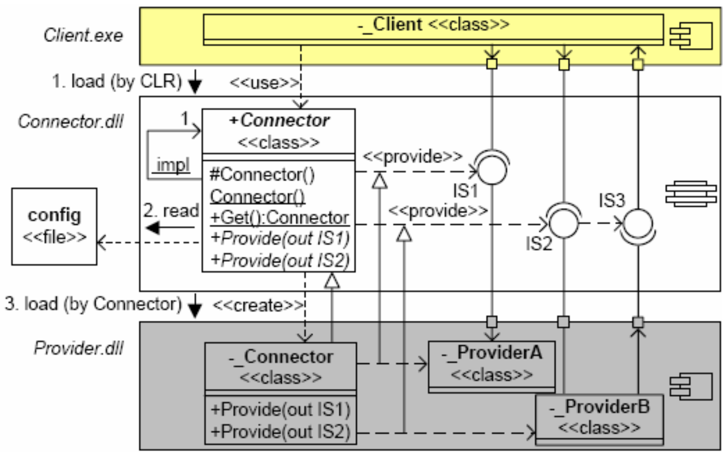
4. Run the application
   … without changing a line of code

An application scenario

```
using NumericSystem.Interface; // import the connector
...
Connector c = Connector.Get();
   // loads the provider, creates and registers
   // the connector object behind the scenes
ICalculator calculator;
INumber divisor, dividend, quotient;
c.Provide(out calculator);
calculator.Provide(out divisor);
calculator.Provide(out dividend);
divisor.numerator = 11;
dividend.numerator = 13;
quotient = calculator.Divide(divisor, dividend);
```

The design behind such an application scenario

Visibility indicators: + public, - internal



A Solution (2)

Tasks of a **connector**:

- Combine semantically coherent interfaces
- Establish the connection between a client and a provider

… without violating the information hiding principle

A **Connector** has at least one class

… the *provider-independent connector class*

Each provider has exactly one

… *provider-specific connector (sub)class*

… per supported connector

Exactly one object per provider-specific connector class

Clients use this object as a kind of factory

A Solution (3)

Connectors may factor out nonfunctional services
… from functional components.

Light connectors:
- only contain declarations of application-specific functions
  … organized in interfaces

Heavy connectors:
- wrap interfaces in proxy classes
- hook in services like logging, profiling …
- wrap call interfaces on the way *out*
- wrap callback interfaces on the way *in*

Connectors that may carry several providers: *Multitions*
… in contrast to *Singletons* (light and heavy) discussed so far

Review

We have been using connectors
- for a generic log analyzer
- in several student projects: mobile cash box, p2p chat ...

Our experience with others:
- heavy usage of OO hinders understanding of connectors
- need about 2 common programming sessions for confidence

Results compared to the goals
+ Ease of use
+ Sharp interfaces, information hiding principle
+ Complete separation of functional components
+ Hiding platform specifics
-/+ Dynamic exchange of components

Outlook

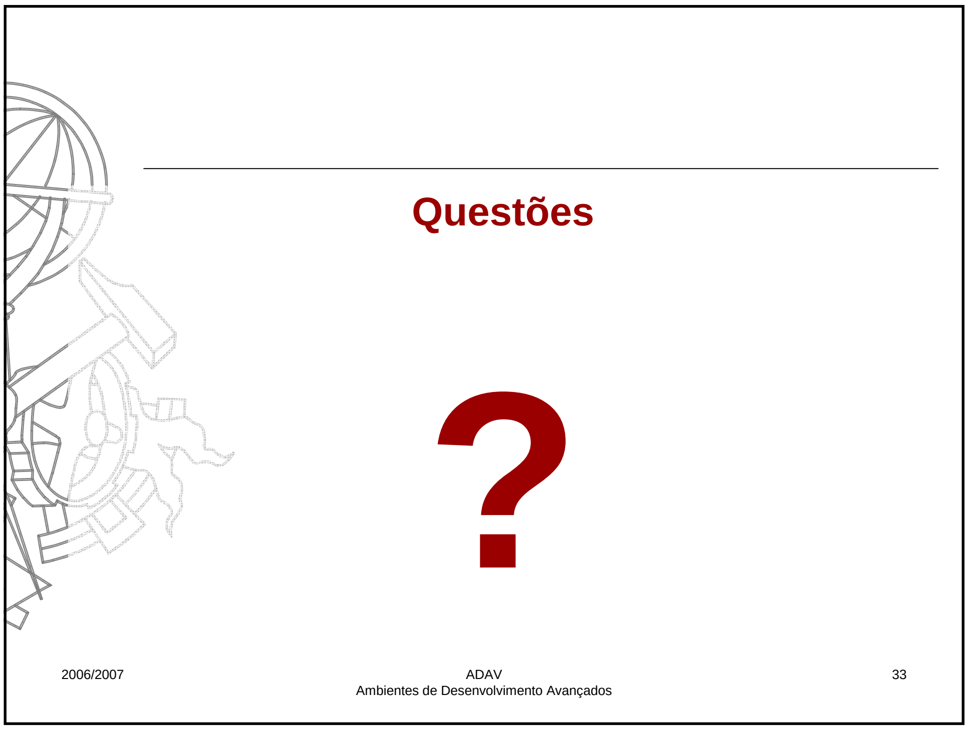
Focus: Experiments with Connectors of various expansion stages

- Load and unload providers dynamically
  … technically it works with Assemblies in AppDomains
  … and replacing interfaces with pure abstract classes
  … but what to do with stateful component instance?
- Expand on mechanisms for protocol checking
- Choose providers dynamically
  … according to an application/client-specific strategy
- Open programms for dynamic configuration from outside
- Monitor functional components
- …

Outlook (2)

We do not pretend to have achieved perfection – but we do have a system – and it **works**.

Michael Rennie
in „The Day The Earth Stood Still“ 1951



Questões

?