Ambientes de Desenvolvimento Avançados

Aula 5

Engenharia Informática

2005/2006

José António Tavares
jrt@isep.ipp.pt

1

O que é um componente e o que não é?

Capítulo 4 de:

Szyperski, Clemens et al. Component Software - *Beyond Object-Oriented Programming.* Second Edition

1

O que é um componente?

- **"A software package which offers *service* through *interfaces*"**
[Peter Herzum and Oliver Sims, "Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise", John Wiley & Sons, Incorporated, 1999].

- **"A coherent package of software artifacts that can be independently developed and delivered as a unit and that can be composed, unchanged, with other components to build something larger"**
[D.F. D'Souza and A.C. Wills, "Objects, Components, And Frameworks with UML – The Catalysis Approach" Addison-Wesley, 1998].

- **"A component is a unit of composition with contractually specified interfaces and *explicit context dependencies* only. A software component can be deployed independently and is subject to composition by third parties."**
[C. Szyperski, "Component Software: Beyond Object-Oriented Programming" Addison-Wesley, 1998].
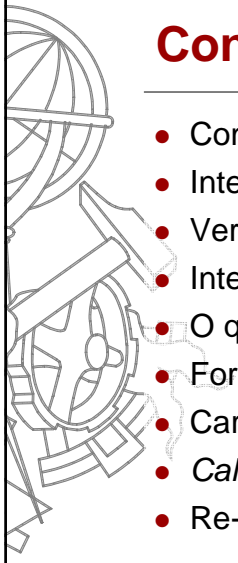
O que não é um componente?

Component isn't an object, not in sense of simply being an object in a Java or C++ program, although it is true at runtime.

Componentes, Interfaces e re-entrada.

Capítulo 5 de:

Szyperski, Clemens et al. Component Software - *Beyond Object-Oriented Programming.* Second Edition

Conteúdo

- Componentes e interfaces
- Interfaces directas e indirectas
- Versões
- Interfaces como contrato
- O que pertence a um contrato?
- Formalidade ou informalidade?
- Características não documentadas
- *Callbacks* e contractos
- Re-entrada nos objectos

Componentes e interfaces

- Interfaces are the means by which components connect. Technically, an interface is a set of named operations that can be invoked by clients.

- Each operation's semantics is specified, and this specification plays a dual role as it serves both providers implementing the interface and clients using the interface.

Componentes e interfaces

- A component may either directly provide an interface or implement objects that, if made available to clients, provide interfaces.

- Interfaces **directly** provided by a component correspond to procedural interfaces of traditional libraries. Such **indirectly** implemented interfaces correspond to object interfaces.

Interfaces Directas e indirectas

- A procedural (direct) interface to a component is modeled as an object interface of a static object within the component.

- An object (indirect) interface introduces an indirection called method dispatch or, sometimes, dynamic method lookup.
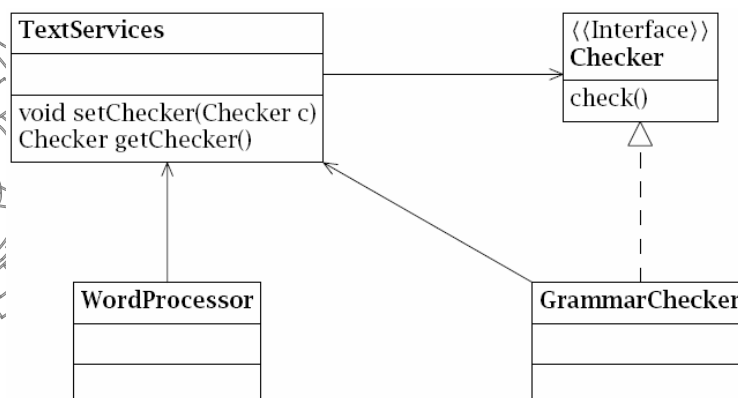
Interfaces Directas e indirectas

Example of indirection: classes


```
TextServices
─────────────────────────
void setChecker(Checker c)
Checker getChecker()
```

```
⟨⟨Interface⟩⟩
Checker
──────────
check()
```

```
WordProcessor
```

```
GrammarChecker
```

5

Interfaces Directas e indirectas

Example of indirection: messages
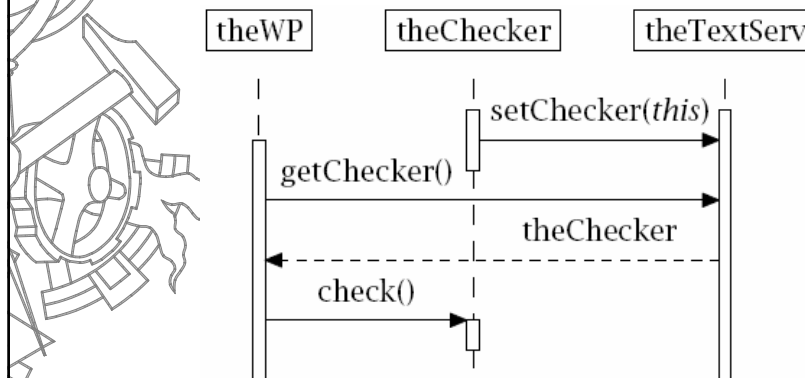
Versões

- Traditional version management assumes that the versions of a component evolve at a single source. In a free market, the evolution of versions is more complex and management of version numbers can become a problem in its own right.

- With direct interfaces it suffices to check versions at bind time, which is when a service is first requested.
- In indirect interfaces couple arbitrary third party.
- In a versioned system, care must be taken to avoid indirect coupling of parties that are of incompatible versions.
- The goal is to ensure that older and newer components are either compatible or clearly detected as incompatible.

Interfaces como contrato

- Interfaces can be viewed as contracts between provider and consumer;
- The contract states what the client needs to do to use the interfaces and what the provider has to implement to meet the services promised by the interface;
- A contract is an appropriate approach, with **pre-** and **post-conditions** attached to every operation
  - The client has to establish the pre-condition before calling the operation and the provider can rely on the precondition being met whenever the operation is called
  - The provider has to establish the post-condition before returning to the client and the client can rely on the post-condition being met whenever the call to the operation returns
- Pre- and post-conditions are not the only way to form contracts.

O que pertence a um contrato?

- contract = signature + behavioral specification;
- specifies *requirements* and *guarantees*, perhaps using pre- and post-conditions;
- *refinements* (eg revisions) may *weaken preconditions* and/or *strengthen post conditions*
- might also specify *non-functional requirements* (eg speed, time complexity, space)
- might also specify *safety* ("this bad thing will never happen") and *progress* ("this good thing will eventually happen") properties
- should be *rigorous*; may be *formal*

Formalidade ou informalidade?

- None of the real-world laws are formal. New "interpretations" are found every day and tested in court.

- Interface contracts should be as formal as possible to derive all necessary information and to enable formal verification – this is complex and, therefore, rarely used in practice;
- Different parts of a system can be specified using different degrees of formality – the preciseness of the specification have to be balanced against the critically of the target part.

Características não documentadas

- always possible to observe behavior of implementation (eg testing, debugging, espionage)
- may provide more information than specification
- depending on such information is dangerous
- no guarantee that later versions will behave the same
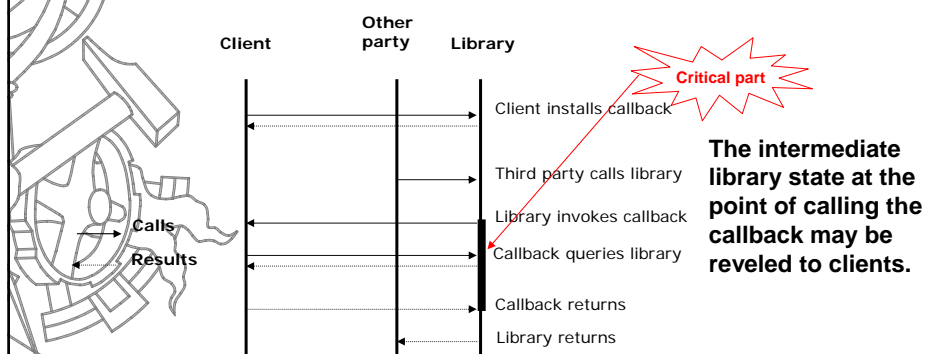- no guarantee even that this version always behaves the same

Callbacks e contractos

- *Callback* or *up-call* is procedure *registered with* and subsequently called by a library
- *Callbacks* are a common feature in procedural libraries that have to handle asynchronous events.
- A callback usually reverses the direction of the flow of controls, so a lower layer calls a procedure in a higher layer.
- The resulting contract are far less manageable than simple pre- and post-conditions.
- Validity of the library state is specified as part of a contract.

Callbacks e contractos

Client   Other party   Library

Critical part

Client installs callback

Third party calls library

Library invokes callback

Callback queries library

Callback returns

Library returns

Calls

Results

The intermediate library state at the point of calling the callback may be reveled to clients.

```
public delegate void MyDelegate();    // delegate declaration

public interface I {
    event MyDelegate MyEvent;
    void FireAway();
}

public class MyClass: I {
    public event MyDelegate MyEvent;

    public void FireAway()
    {
        if (MyEvent != null)
            MyEvent();
    }
}

public class MainClass {
    static private void f()     {
        Console.WriteLine("Called when the event fires.");
    }

    static public void Main ()     {
        I i = new MyClass();

        i.MyEvent += new MyDelegate(f);
        i.FireAway();
    }
}
```

Que é têm de especial os *callbacks*?

- in layered architecture, calls originate in higher (more abstract) layer and move downwards
- library operations complete before returning to client, who cannot observe intermediate states
- callback usually reverses this flow
- intermediate state of library becomes visible
- client may observe, or even modify, library's intermediate state
- client certainly observes identity and ordering of callbacks
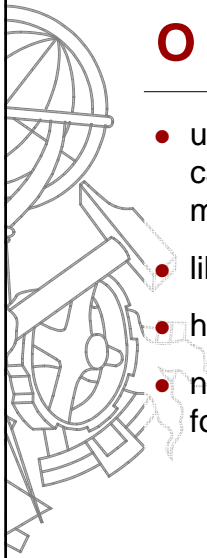
O que é que se pode fazer

- unrealistic to restrict behavior of client during callback (most non-trivial callbacks query library for more information before taking appropriate action)

- library state must remain valid while observable

- hence must remain valid during callbacks

- no longer sufficient to give pre- and post-conditions for library
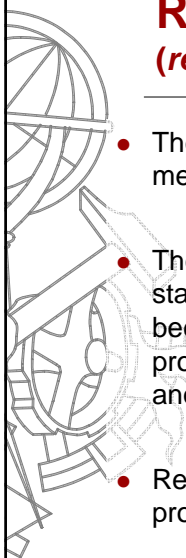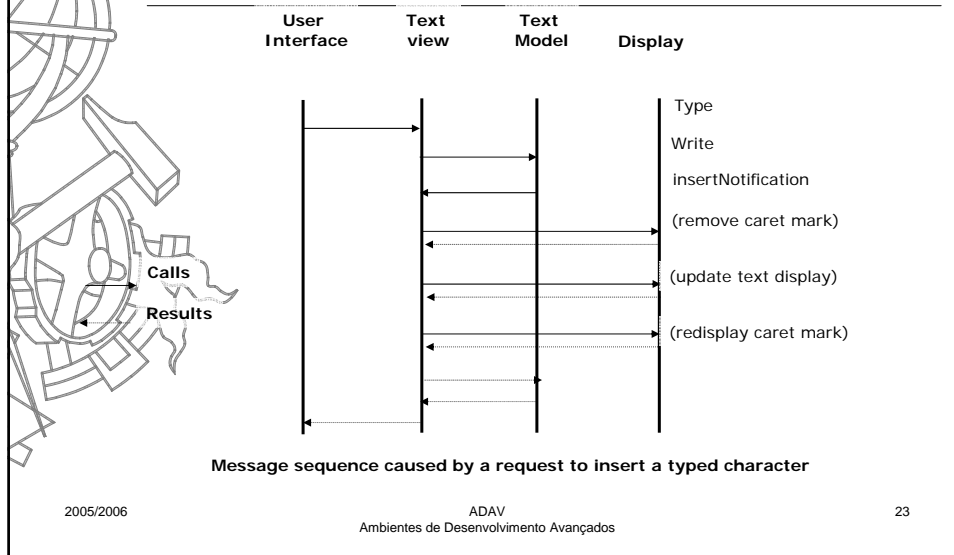
Re-entrada nos objectos
(*re-entrance*)

- The object re-entrance is the situation in which an object´s method is invoked while another method is still executing.

- The real problem is observation of an object undergoing a state transition with inconsistent intermediate states becoming visible. Considering object re-entrance, the problem is when an object´s method is invoked while another method is still executing.

- Recursion and re-entrances become even more pressing problem when crossing the boundaries of components.

Re-entrada nos objectos
(*re-entrance*)

User Interface	Text view	Text Model	Display
			Type
			Write
			insertNotification
			(remove caret mark)
			(update text display)
			(redisplay caret mark)

Calls

Results

Message sequence caused by a request to insert a typed character
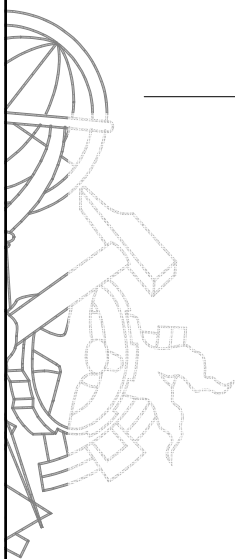

Re-entrada nos objectos

- **Multi-threading**
  - problems of recursive re-entrance similar to those of concurrent interaction
  - perhaps helps to make objects *thread-safe*? (ie protected from unwanted interference from concurrent activities)
  - no! thread safety addresses only external re-entrance
  - locking prevents other objects from invoking our methods, but cannot prevent us from invoking our own (or self-inflicted deadlocks would result)

Questões

?