Aspectos técnicos do desenvolvimento baseado em componentes

Um novo processo de desenvolvimento

- O uso de componentes traz mudanças no processo de desenvolvimento
- Além de desenvolver um produto, queremos criar abstrações reutilizáveis e aumentar o *patrimônio* da empresa
- Isso requer uma nova forma de pensar dos programadores
- Também requer novos incentivos para os envolvidos

Novas etapas surgem:

- Seleção de componentes e possível adaptação
  - Pode ser in-house, out-sourced ou compra
- "Assembly time" para construção de aplicações
  - A composição é feita numa etapa separada da construção de componentes
  - Antes, tudo era feito na mesma etapa
- "Deployment time" para configurar componentes no ambiente final
- Novas ferramentas são necessárias para as novas etapas
  - Ferramentas visuais de assembly e de deployment
  - Uso do design pattern "Prototype" para instanciar objetos graficamente
- Configuration Management é muito mais importante do que antes porque tem muito mais componentes individuais do que antes (aplicações individuais)
- A fase de integração era a última fase antes de componentes
  - Agora, ela passa a ser a fase principal de

construção de aplicações mas se chama a fase de composição ou assembly
    - A velha fase de integração final antes da entrega não existe mais, já que os pedaços são entregues separadamente e juntados em tempo de execução
  - Composição é o foco, em vez de codificação

Novos papeis no desenvolvimento

- Quem coordena os esforços de reutilização entre times e decide que componentes são desenvolvidos/out-sourced/comprados
- Quem faz componentes
- Quem monta aplicações com componentes
  - Pode ser integradores de sistema também
  - Software houses (SAP, Oracle, p.ex.) estão rearquiteturando as soluções para usar componentes
- Quem instala e configura componentes em ambientes finais
  - Lembra que os componentes podem ser instalados em servidores e várias aplicações diferentes podem usá-los

Comparação técnica de OOP e COP

- Resumo de Orientação a Objeto
  - Classes e objetos
  - Algum "Information hiding"
  - Herança
  - Polimorfismo com Late Binding
    - Para ter extensibilidade (adicionar novos pedaços) sem repensar tudo
    - Podemos codificar a parte cliente através de um conhecimento geral (apenas) do comportamento esperado
    - A extensão (um método polimórfico, p. ex.) pode ser plugada onde este conhecimento geral era

esperado

- Para entender melhor a diferença com Component-Oriented Programming, precisamos falr um pouco mais de *binding*:

Tempo de Binding	Como funciona
Tempo de codificação	✍ Codifição simples sem sequer separação procedural
Tempo de compilação	✍ Separação padrão em procedimentos/funções ✍ As chamadas são amarradas (bound) a uma implementação em tempo de compilação
Tempo de link-edição	✍ Unidades separadas de compilação com interfaces (ex. "protótipos" em C) declaradas separadamente das implementações ✍ As chamadas são compiladas usando as interfaces ✍ As chamadas são amarradas (bound) a uma implementação em tempo de link-edição para gerar um único módulo executável
Ligação dinâmica	✍ Unidades separadas de compilação que não precisam se link-editadas num único módulo executável para permitir a execução ✍ Um módulo compilado (DLL) pode ser linkado dinamicamente a um sistema que esteja executando ✍ As chamadas são compiladas usando as interfaces ✍ As chamadas são amarradas (bound) a uma implementação em tempo de execução ✍ A implementação é carregada em tempo de execução ✍ Java é assim (DLL é o arquivo .class ou .jar)
Reflexivo	✍ Chamadas não são compiladas com interfaces em tempo de compilação ✍ Em tempo de execução, tem-se informação suficiente sobre as interfaces disponíveis ✍ Em tempo de execução, as chamadas são feitas a tais interfaces usando "reflexão" (olhar dentro do "módulo" para descobrir o que se quer) ✍ O binding é obviamente dinâmico ✍ Normalmente, os nomes das chamadas são contidas em strings; os nomes das chamadas podem ser montadas dinamicamente e executadas ✍ Java usa reflexão

- Resumo de Orientação a Componente
  - Mais "Information hiding"
    - Apenas interfaces são usadas. Ponto.
    - Unidade de packaging reforça a encapsulação (tudo está lá dentro)
  - Polimorfismo com Very Late Binding
    - Software baseado em Componentes é extensível por definição, i.e. nunca está completo

- Novos componentes podem ser acrescentados a qualquer momento: precisa de Very Late Binding
- Não pode precisar de um teste de integração final porque esta fase não existe!

- Very Late Binding inclui fazer o seguinte em tempo de execução:
  - Achar o objeto
  - Carregá-lo
  - Dinamicamente chamar a implementação de suas interfaces
- Reflexão é necessária para descobrir interfaces e chamar as implementações das interfaces em tempo de execução

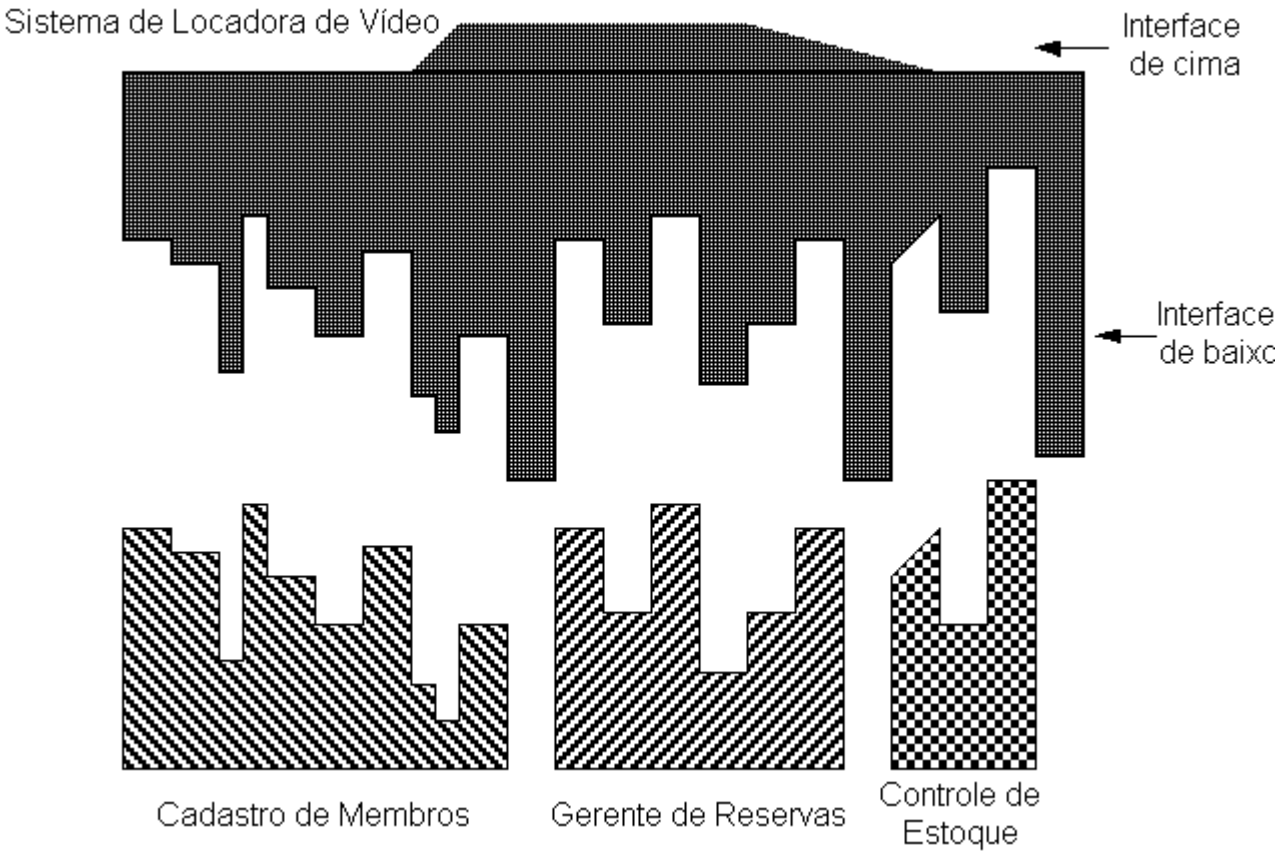
- Não tem herança entre componentes (ou, pelo menos, é muito mal visto)
  - Para diminuir acoplamento
  - Pode ter herança internamente
- Segurança
  - A instalação de novos componentes no ambiente do usuário não deve invalidar os componentes já instalados
  - Isso leva ao "Open-Closed Principle"
    - "Um Componente deve ser aberto à extensão mas fechado à modificação"
  - Portanto, gerenciamento de memória é muito mais importante em COP
    - Tem que ter garbage collection porque um componente nunca pode saber quando um outro componente pode ser liberado

Década	**Tecnologia de Programação**	**Novidade**
1940	Código de máquina	Máquinas programáveis
1950	Assembly	Linguagens simbólicas
1960	Linguagens de alto nível	Expressões e independência de máquina
		Tipos estruturados e estruturas de

1970	Programação estruturada	controle
1980	Programação modular	Separação da interface e da implementação
1990	Programação Orientada a Objeto	Polimorfismo
2000	Programação Orientada a Componente	Composição dinâmica e segura

Supporte de Linguagem

- Lembre que um componente é uma unidade de
  - Composição
  - Packaging
  - Deployment
- Precisamos de um suporte adequado das linguagens de programação para dar suporte a esses conceitos
- Um dos aspectos mais importantes é o encapsulamento do componente inteiro
  - Para que a interferência com outros componentes (de outros fabricantes) possa ser controlada estaticamente (i.e., sem inspecionar o resultado da fusão)
- Aliados a isso, precisamos de mecanismos de segurança para que componentes não possam afetar a integridade de outros
  - Inclui integridade de memória a nível de componente
- Finalmente, precisamos de um mecanismo para definir interfaces
  - O que ofereço (interface de cima)
  - O que preciso (interface de baixo)



- Como estão as linguagens de hoje?
  - Não suportam COP bem
  - C++ e Smalltalk não têm interfaces
  - Em Java, temos definição de interfaces mas não podemos definir a interface de uma unidade completa (interface de um package, se usarmos um package para implementar um componente)
    - Não podemos dizer que "O package X implementa a interface Y"
  - Smalltalk não permite agrupar classes
  - Em todas essas linguagens, não posso dizer "Me dê um componente com esta especificação"
    - Tenho que especificar algo concreto, não apenas uma interface
- Como CORBA atende?
  - CORBA dá interfaces, e permite pedir um objeto que tenha uma certa interface
    - Objetos distribuídos são componentes
  - Problema: CORBA não é uma linguagem de

programação

- É uma camada acima de uma linguagem
- Fica pesado demais fazer tudo com CORBA (usar centenas de componentes)
- CORBA é independente de plataforma: overhead de comunicação é grande e CORBA é portanto lento
- CORBA é caro (tem que pagar licenças de run-time)

comp-2 programa anterior próxima