Engenharia de Software

Modelação de Software

Luís Morgado

Instituto Superior de Engenharia de Lisboa

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Modelação de Software

- A modelação de software é um processo de criação de uma *representação abstracta* do software como um sistema, a qual serve de suporte para a análise e concepção do software a desenvolver
- Tem por base conhecimento (do domínio do problema e do domínio da solução)
- Tem por resultado uma representação de conhecimento, textual e/ou gráfica, que específica o sistema a realizar nas suas diferentes vertentes, nomeadamente, estrutural, dinâmica e comportamental
- É guiada por princípios, conceitos e modelos que que servem de base e orientam a forma de abordar questões e problemas e de conceber as respectivas soluções, por exemplo, o *modelo orientado a objectos*
- É suportada em linguagens e ferramentas específicas de modelação de software, nomeadamente a *linguagem UML*
- Tem como resultado um conjunto de modelos que definem a *arquitectura* do software a produzir
- Conceitos principais envolvidos: **abstracção**, **representação**, **modelo**

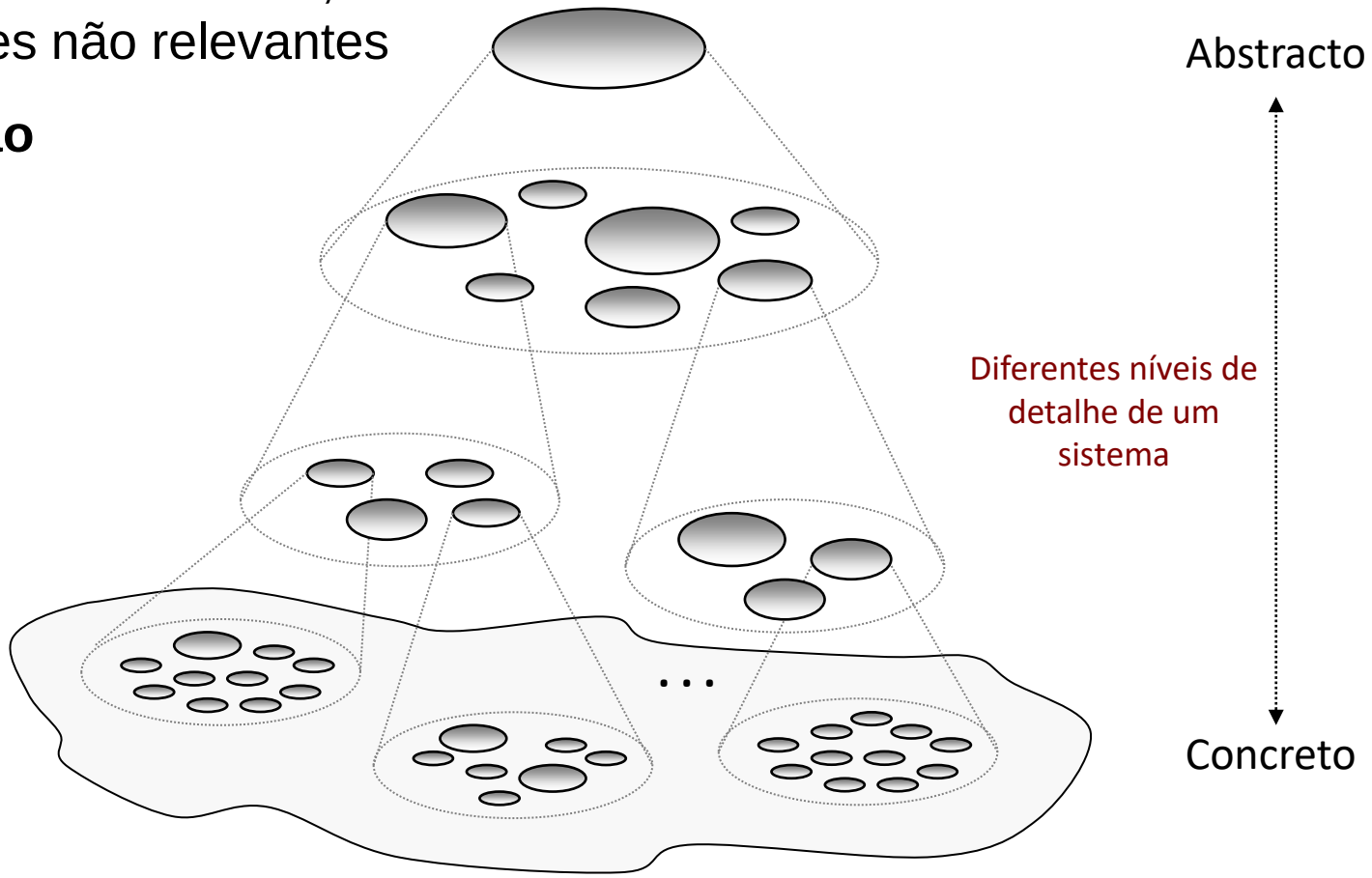
Abstracção

- Aspecto principal da modelação de sistemas
- Processo de descrição de conhecimento a *diferentes níveis de detalhe* (quantidade de informação) e *tipos de representação* (estrutura da informação) [Korf, 1980]
- **A abstracção é uma ferramenta base para lidar com a complexidade**
  - Identificação de características comuns a diferentes partes
  - Realçar o que é essencial, omitir detalhes não relevantes
  - Modelos, representações abstractas de um sistema
- **Desenvolvimento de um sistema complexo**
  - Criação de ordem de forma progressiva através de diferentes níveis de abstracção
  - Processo iterativo guiado por conhecimento

Abstracção

Ferramenta base para lidar com a complexidade

- **Realçar** o que é **essencial**, omitir detalhes não relevantes
- **Simplificação**
- **Focagem**

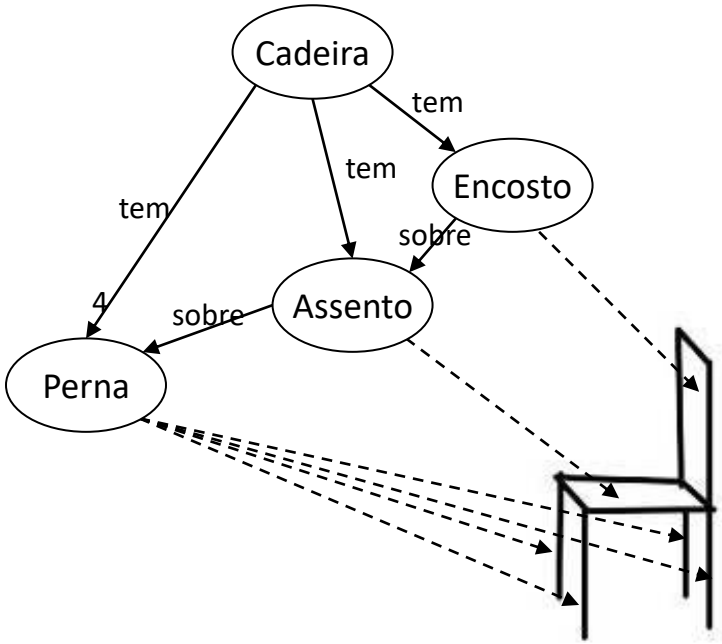


Representação

Forma de expressar o conhecimento acerca de um determinado domínio tendo por base os conceitos que o definem e as respectivas relações

- Pode ter um formato textual ou gráfico
- O formato gráfico aumenta a clareza da representação, facilitando a sua compreensão e manipulação
  - *Rede semântica*: expressa o significado do conhecimento envolvido com base em conceitos e relações entre conceitos

Domínio de representação



Representação de conhecimento referente a um determinado domínio

Linguagem de Representação

A representação de conhecimento envolve três aspectos principais, definidos pela *linguagem de representação*:

- **Notação**
  - Também designada ***sintaxe***, corresponde à especificação das convenções de *forma* de uma determinada representação, ou seja, quais os símbolos e arranjos de símbolos válidos
- **Denotação**
  - Também designada ***semântica***, corresponde à atribuição de *significado* às entidades especificadas na notação (conteúdo semântico)
- **Manipulação**
  - Corresponde à especificação da forma como os elementos da representação devem ser manipulados, de acordo com a sintaxe e a semântica associada, para *inferência* acerca do conhecimento representado

Exemplo: Linguagem UML

A linguagem UML (*Unified Modelling Language*) é um exemplo de uma linguagem de representação de sistemas, orientada em particular para a modelação de software

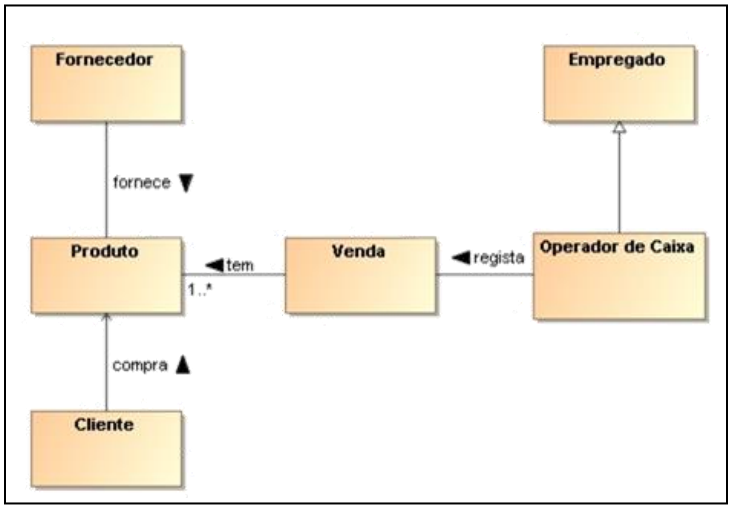


Diagrama de classes em linguagem UML

Conceitos (rectângulos) representam classes

Relações (linhas) representam associações entre classes

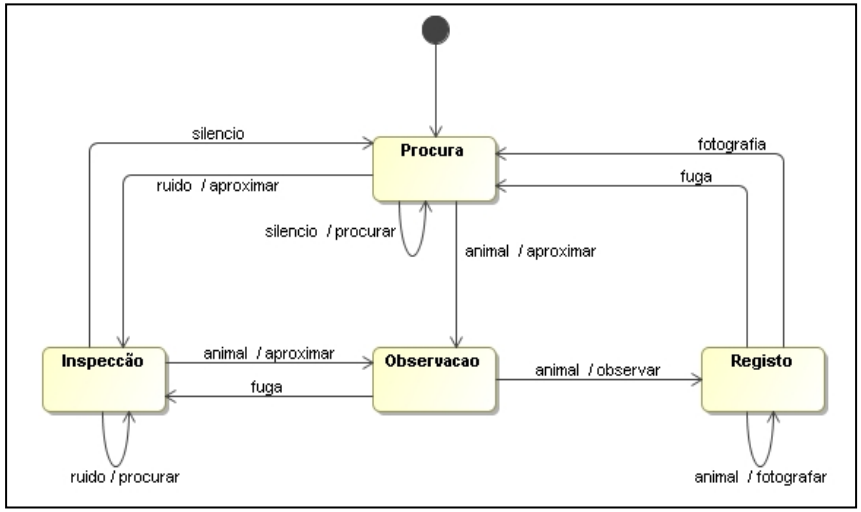


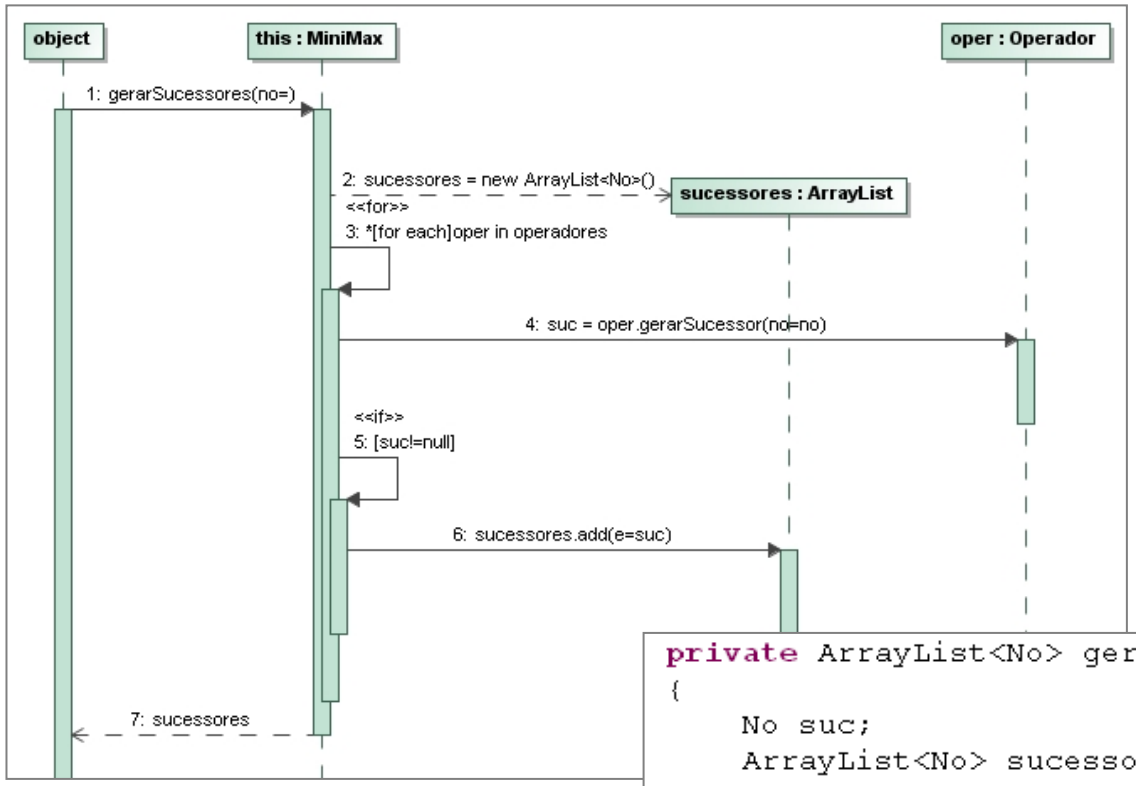
Diagrama de transição de estado em linguagem UML

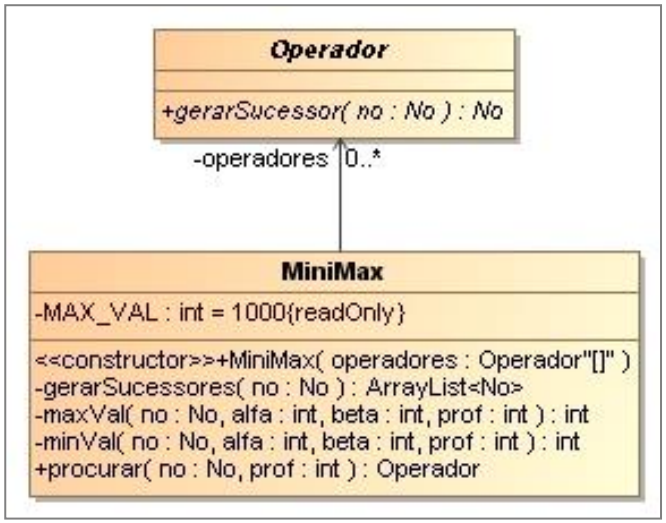
Conceitos (rectângulos com cantos redondos) representam estados

Relações (linhas com seta) representam transições entre estados

A *inferência* ocorre por transitividade das relações, por exemplo, "*uma venda tem um ou mais produtos*", ou, "*existe uma transição do estado Inspecção para o estado Procura pelo evento silêncio*"

Modelação de Software






```
private ArrayList<No> gerarSucessores(No no)
{
    No suc;
    ArrayList<No> sucessores = new ArrayList<No>();

    // Para todos os operadores gerar sucessor do nó
    for(Operador oper : operadores) {
        suc = oper.gerarSucessor(no);
        if(suc != null)
            sucessores.add(suc);
    }

    return sucessores;
}
```

Representação de um sistema computacional em diferentes perspectivas e níveis de abstracção

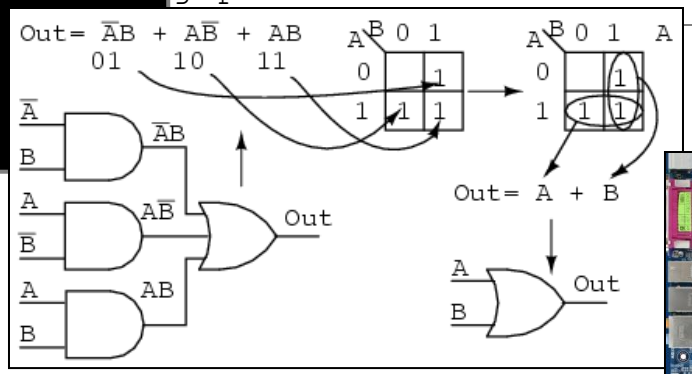
Modelação de Software

```
private ArrayList<No> gerarSucessores(No no)
{
    No suc;
    ArrayList<No> sucessores = new ArrayList<No>();

    // Para todos os operadores gerar sucessor do nó
    for(Operador oper : operadores) {
        suc = oper.gerarSucessor(no);
        if(suc != null)
            sucessores.add(suc);
    }
```

```
004113CE C7 05 AC 71 41 00 00 00 00 00
004113D8 83 3D A0 71 41 00 00
004113DF 7E 1F
004113E1 A1 AC 71 41 00
004113E6 03 05 9C 71 41 00
004113EC A3 AC 71 41 00
004113F1 A1 A0 71 41 00
004113F6 83 E8 01
004113F9 A3 A0 71 41 00
004113FE EB D8
00411400 5F
```

```
mov   dword ptr [_res (4171ACh)],0
cmp   dword ptr [_x (4171A0h)],0
jle   411400h
mov   eax,dword ptr [_res (4171ACh)]
add   eax,dword ptr [_y (41719Ch)]
mov   dword ptr [_res (4171ACh)],eax
mov   eax,dword ptr [_x (4171A0h)]
sub   eax,1
mov   dword ptr [_x (4171A0h)],eax
jmp   4113D8h
```

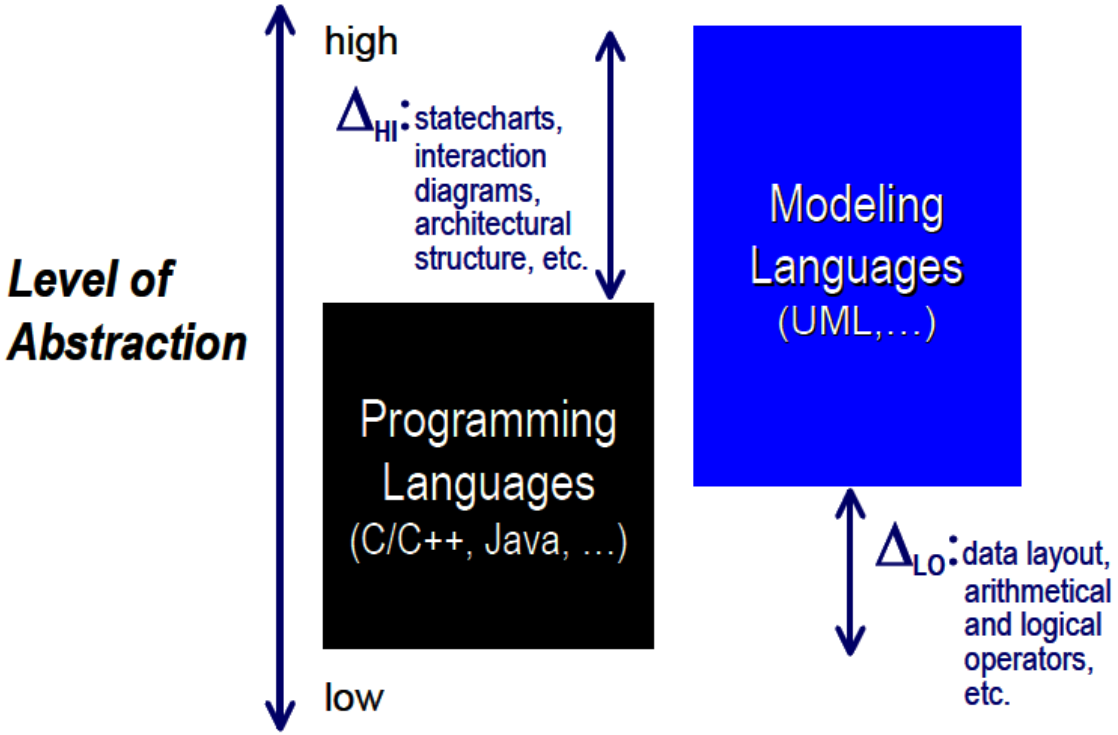


Nos níveis de descrição mais concretos as representações são progressivamente mais detalhadas, até corresponderem ao suporte de execução (*hardware*)

Linguagens de Modelação

Linguagens orientadas para a representação abstracta de sistemas sob a forma de modelos

As diferentes linguagens de representação de software estão orientadas para a descrição de software a diferentes níveis de abstracção, desde níveis mais detalhados, como as linguagens *assembly*, até níveis mais abstractos com linguagens de modelação de software, como a linguagem UML



[Selic, 2003]

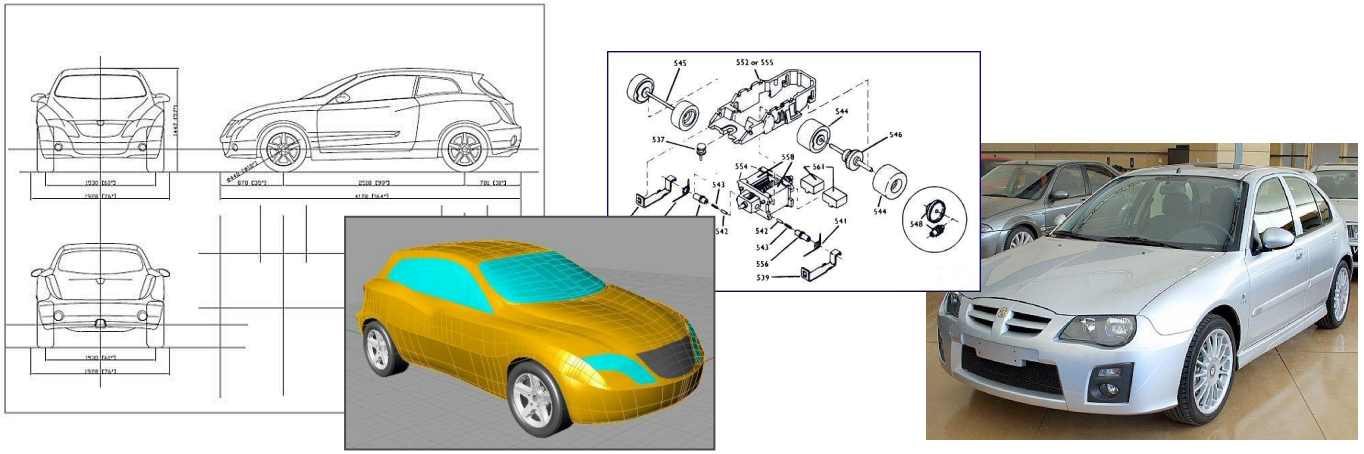
Elementos de descrição diferentes em cada nível

Δ_{HI} : Elementos de modelação não disponíveis em linguagens mais específicas

Δ_{LO} : Elementos disponíveis em linguagens mais específicas não disponíveis em linguagens de modelação

Modelo

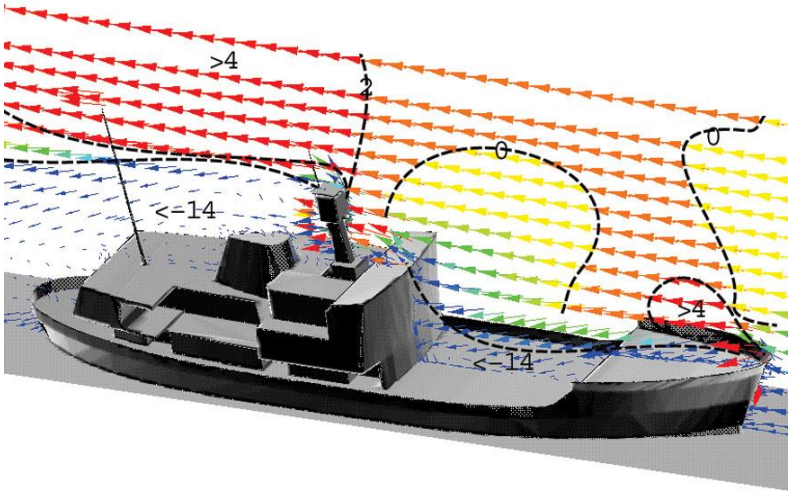
- **Representação abstracta de um sistema**
  - Especificação com base em conceitos abstractos das características fundamentais de um sistema
  - Representação de conhecimento acerca de um sistema
- **Meio para lidar com a complexidade**
  - Obtenção e sistematização progressiva de conhecimento
  - Compreensão e comunicação acerca do sistema
  - Especificação de referência para a realização do sistema
  - Documentação de um sistema



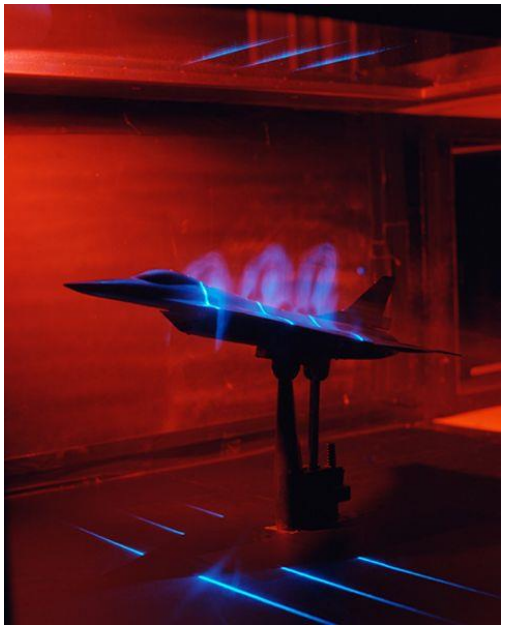
Modelos em Engenharia

Redução de incerteza / risco

- Elaboração de modelos para verificação de propriedades através de simulação
- Obtenção de conhecimento antes de construir o sistema concreto



[Yelland et al., 2002]

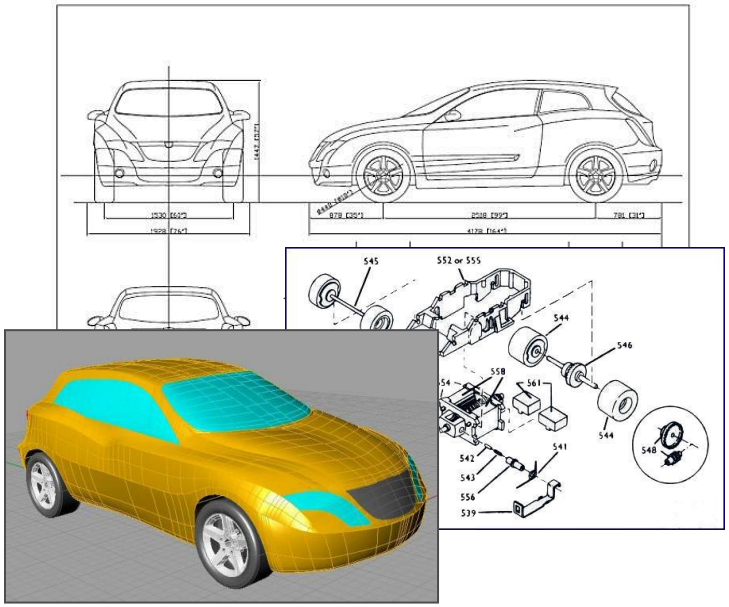


[aerospaceweb.org]

Modelos em Engenharia

O problema dos modelos

- **A realidade é muito mais rica que qualquer abstracção!**
- "… *the good thing about bubbles and arrows, as opposed to programs, is that they never crash.*" [Meyer, 1997]



HIATO SEMÂNTICO

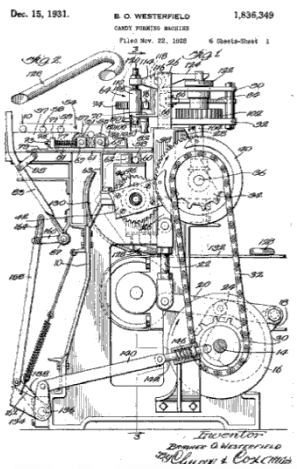
- Especificidades dos suportes de realização
- Efeitos de escala
- Métodos de construção
- Perícia dos construtores
- Falhas de comunicação

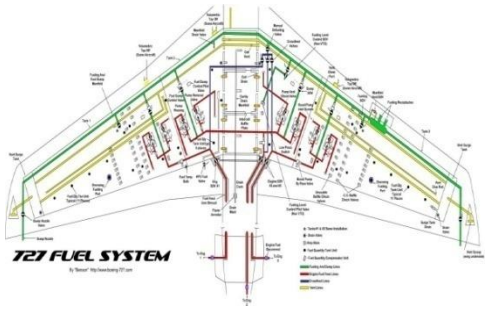
Discrepâncias entre modelo e realização
Falhas de operação

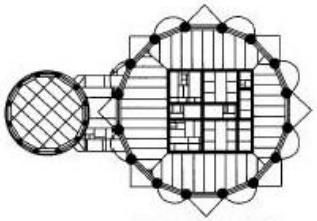
NECESSIDADE DE LIGAÇÃO EFICAZ ENTRE MODELOS E REALIZAÇÃO

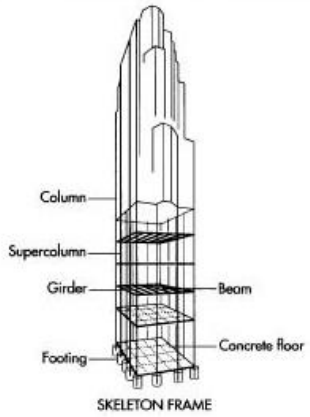
Modelação de um Sistema

DEFINIÇÃO DOS PADRÕES DE ORGANIZAÇÃO DO SISTEMA









CARACTERÍSTICAS IMPORTANTES DE UM MODELO

- **ABSTRACÇÃO**
  - Foco nos aspectos importantes, remoção de aspectos não relevantes
- **COMPREENSÃO**
  - Facilidade de compreensão e transmissão das ideias envolvidas
- **PRECISÃO**
  - Representação correcta e rigorosa do sistema
- **PREVISÃO**
  - Possibilidade de inferência de conhecimento correcto acerca do sistema descrito

Paradigmas de Modelação de Software

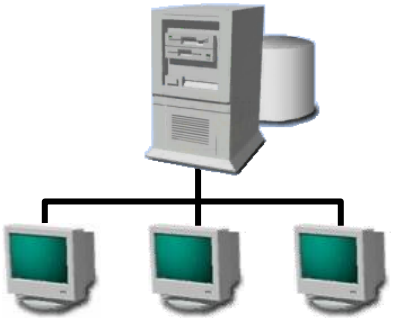
Vários paradigmas de modelação de software surgiram ao longo do tempo no sentido de criar meios adequados para a análise e concepção de software de cada vez maior complexidade

Paradigma, refere-se a um enquadramento conceptual, ou seja, conjunto de ideias, conceitos ou modelos que servem de base e orientam a forma de abordar questões e problemas numa determinada área e de conceber as respectivas soluções

Paradigma monolítico

- Começou nos primórdios do desenvolvimento de software, cerca da década de 1960
- É uma abordagem centralizada em que toda a lógica e dados estão juntos numa única unidade de software
- Estes sistemas foram criados para serem executados num único computador e não tinham necessidade de múltiplos componentes
- A arquitectura monolítica caracteriza-se por ter uma única base de código que contém toda a funcionalidade de uma aplicação e também um sistema de armazenamento de dados centralizado que é acessível a todos os elementos da aplicação

Computação Centralizada

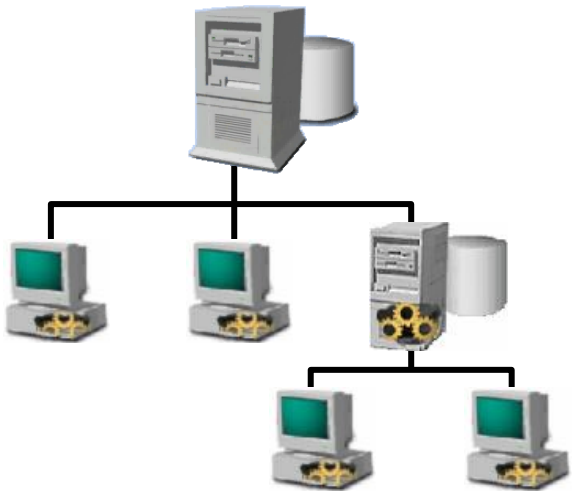


Paradigmas de Modelação de Software

Paradigma estruturado

- Começou a surgir no final da década de 1960 e divulgou-se nas décadas de 1970 e 1980, num período em que a complexidade do software começou a aumentar de forma significativa, nomeadamente, devido ao surgimento de arquitecturas descentralizadas, havendo necessidade de uma abordagem mais organizada ao desenvolvimento de software
- No paradigma estruturado, o software é organizado em módulos definidos numa hierarquia de abstracção, em que cada módulo tem uma tarefa específica a realizar, manipulando dados também organizados de forma estruturada
- Esta abordagem facilitou a o desenvolvimento de software, nomeadamente, tendo por base linguagens de programação estruturada

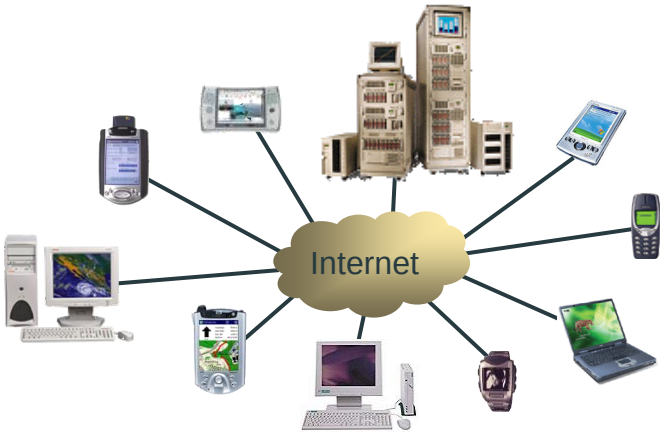
Computação Descentralizada



Paradigma orientado a objectos

- Começou a surgir no final dos anos 1960 e início dos anos 1970, nomeadamente, com o desenvolvimento da linguagem de programação Simula, sendo amplamente divulgado e adoptado nos anos 1990, com o surgimento da computação em rede
- A sua adopção foi motivada em particular pela necessidade de fazer face à complexidade crescente do desenvolvimento de software, nomeadamente, tendo por base uma maior facilidade de abstracção, modularização e reutilização do software produzido
- Define uma forma de organizar o software tendo por base os conceitos de *objecto* e *classe*, para representar entidades do domínio do problema e respectivas relações e interacções
- O paradigma orientado a objectos é hoje amplamente utilizado no desenvolvimento de software

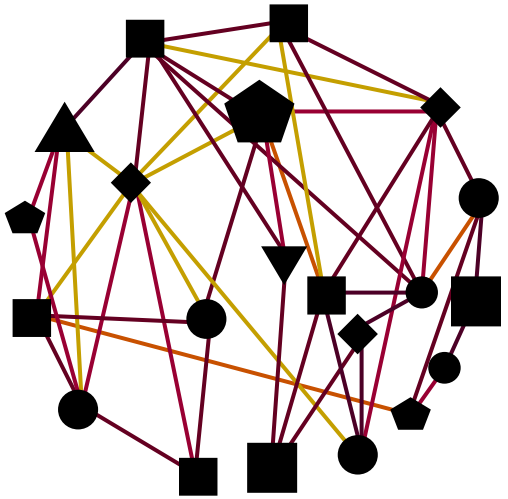
Computação em Rede



Paradigma baseado em agentes

- Começou a surgir na década de 1990 como uma nova abordagem ao desenvolvimento de software
- Tem por base a utilização de agentes autónomos para representar entidades do mundo real e respectivas interacções
- É uma abordagem centrada na representação de conhecimento de um domínio e na modelação das funcionalidades de um sistema sob a forma de objectivos de alto nível, concretizados por unidades computacionais autónomas, designadas *agentes*
- É uma abordagem baseada em conceitos e modelos de inteligência artificial, vocacionada, em particular, para a realização de sistemas distribuídos, como é o caso de sistemas compostos por múltiplas entidades heterogéneas organizadas em rede

Sistemas heterogéneos organizados em rede



Paradigmas de Modelação de Software

Comparação das características dos elementos básicos de modelação

Características dos elementos básicos	Paradigma Monolítico	Paradigma Estruturado	Paradigma Orientado por Objectos	Paradigma Baseado em Agentes
Comportamento	Não modular	Modular	Modular	Modular
Estado	Externo	Externo	Interno	Interno
Evocação	Externa	Externa (*Chamada*)	Externa (*Mensagem*)	Interna (*Objectivo*)

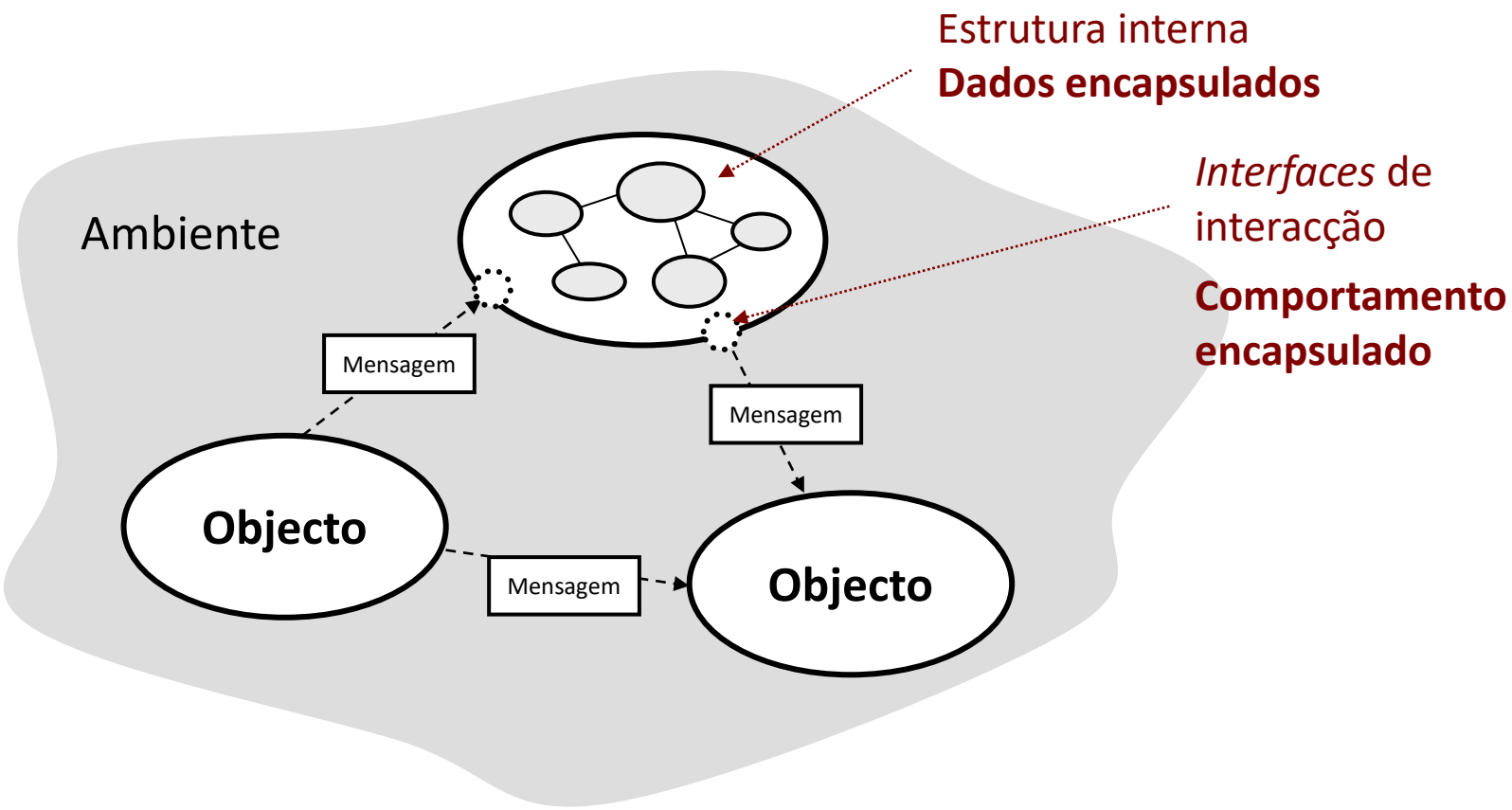
[Odell 2002]

Paradigma Orientado a Objectos

Na modelação orientada a objectos um sistema é representado como um conjunto de objectos que interagem para produzir o comportamento pretendido

Tem por base o conceito de *simulação da realidade*

Enfatiza a modularidade e o encapsulamento, sendo as interacções modeladas com base em mensagens e interfaces de interacção



Paradigma Orientado a Objectos

- A modelação orientada a objectos é um modelo de concepção de software baseado no conceito de *objecto*, o qual agrega de forma modular dados e comportamento
- Os objectos concretos (*instâncias*) são abstraídos nas suas características através do conceito de *classe*, o qual define um *tipo* de objectos
- Os dados são representados como *atributos* ou *propriedades* dos objectos
- O comportamento é representado como funções associadas aos objectos, designadas *métodos*
- Um objectivo principal da modelação orientada a objectos é associar e *encapsular* de forma modular dados e as funções que sobre eles operam

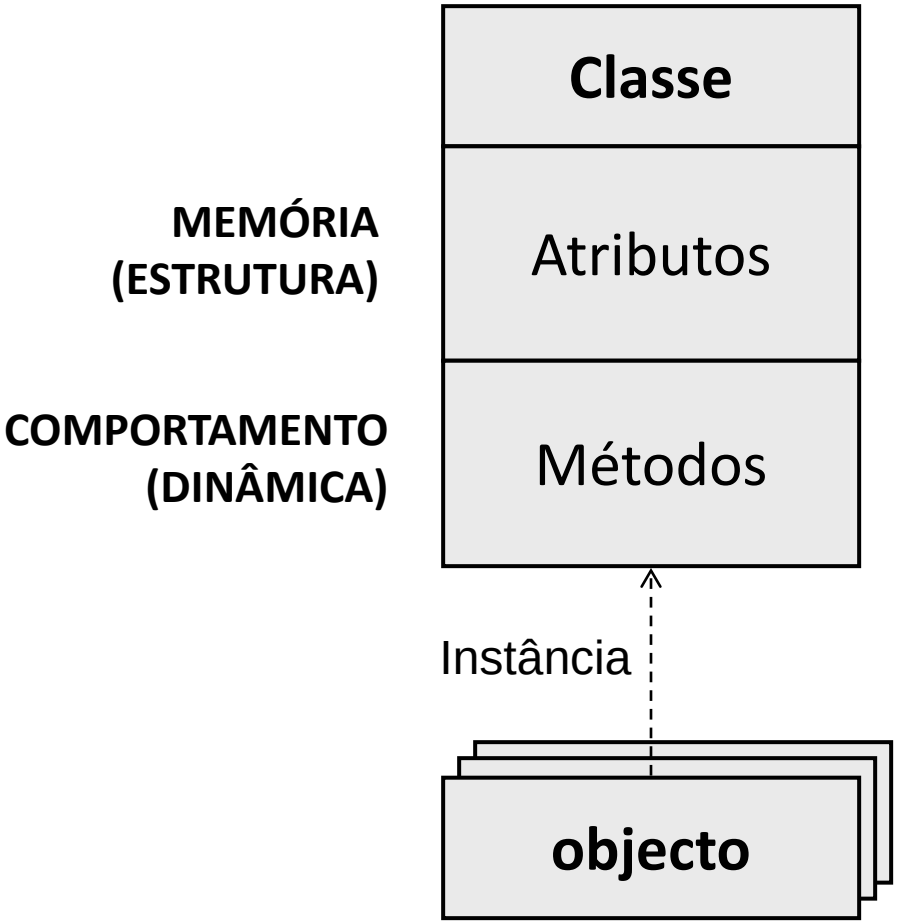
Paradigma Orientado a Objectos

Conceitos principais:

- **Abstracção**
  - Processo de definir a organização de um sistema a diferentes níveis de detalhe, omitindo aspectos não relevantes para cada nível
- **Encapsulamento**
  - Processo de agregar dados e funções que operam sobre esses dados numa única unidade, abstraindo os detalhes internos para o exterior
- **Herança**
  - Processo de definição de classes de objectos por especialização de outras classes existentes, as quais abstraem características gerais das classes especializadas
- **Polimorfismo**
  - Capacidade de objectos de uma mesma classe assumirem formas distintas, nomeadamente expressões comportamentais

Paradigma Orientados a Objectos

MODELO COMPUTACIONAL



HERANÇA

- Super-classes
- Sub-classes
  - Os objectos de uma sub-classe partilham todas as características dos objectos da respectiva super-classe

POLIMORFISMO

- Capacidade de assumir múltiplas formas
  - Classes
  - Operações

Paradigma Orientado a Objectos

Vantagens:

- Promove a organização e clareza dos modelos e do software produzido
- Promove a reutilização de software, reduzindo o esforço de desenvolvimento
- Promove uma maior facilidade de manutenção e modificação do software
- O encapsulamento permite abstrair detalhes de implementação de classes e módulos, facilitando a alteração da organização interna sem afectar a interacção com o exterior
- A herança promove a eliminação de redundância e a reutilização
- O polimorfismo proporciona flexibilidade na organização e interacção das partes de um sistema

Desvantagens:

- Requer maior conhecimento técnico
- Esforço de aprendizagem maior

Princípios de Modelação

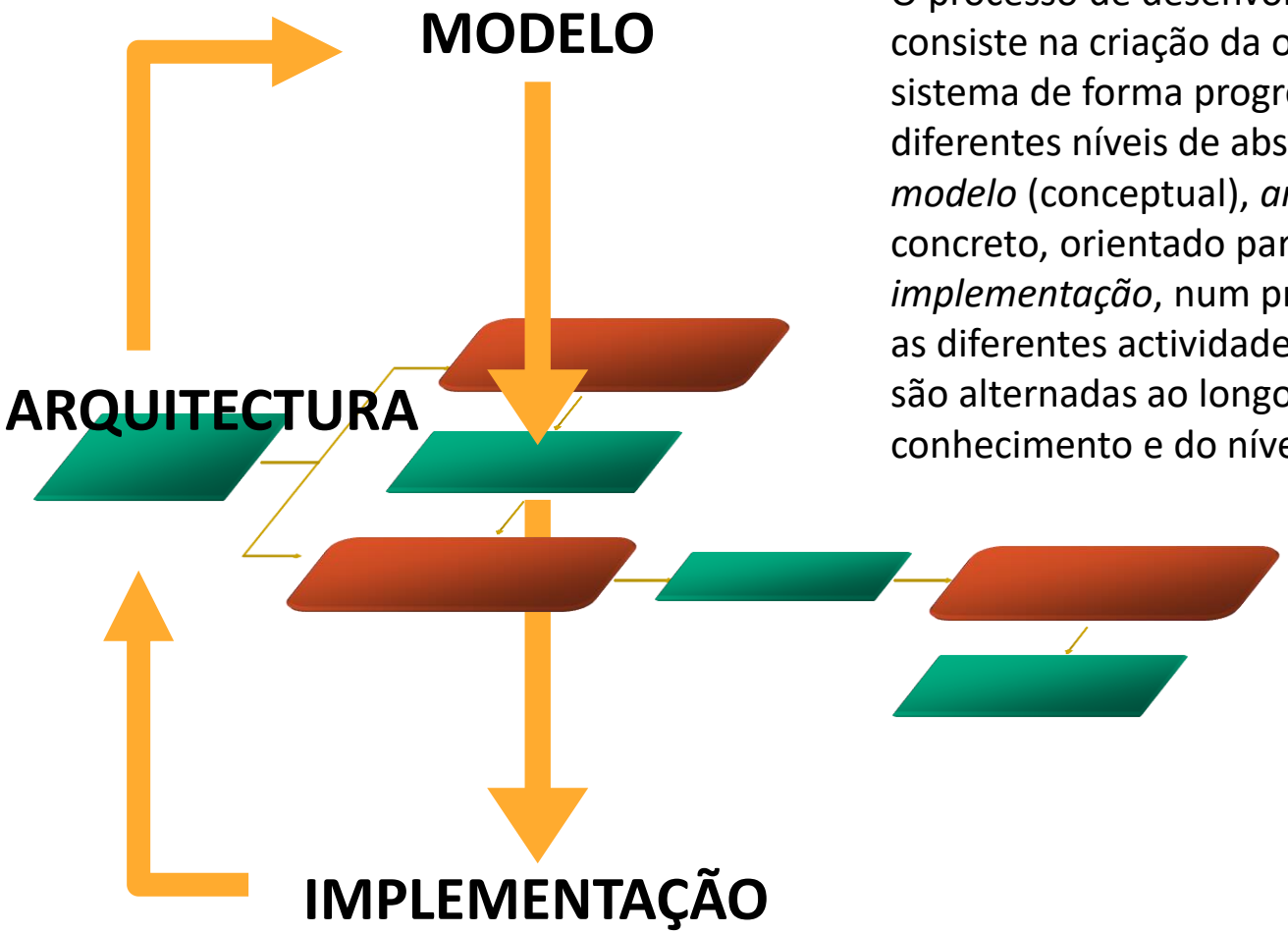
Os princípios de modelação definem conceitos fundamentais que orientam o processo de criação de modelos de um sistema, contribuindo para garantir que os modelos representam de forma correcta e precisa o sistema que está a ser desenvolvido

- **Abstracção**
  - Focar aspectos essenciais do sistema que está a ser modelado, ignorando os detalhes não relevantes
- **Decomposição**
  - Dividir o sistema em partes mais pequenas e mais fáceis de compreender e descrever, que podem ser modeladas separadamente
- **Composição**
  - Combinar partes mais pequenas para criar partes mais complexas

Princípios de Modelação

- **Completude**
  - Garantir que o modelo inclui todas as informações necessárias para representar com precisão o sistema que está a ser desenvolvido
- **Consistência**
  - Garantir que o modelo está livre de contradições ou conflitos
- **Correcção**
  - Garantir que o modelo representa com exactidão o sistema que está a ser desenvolvido e satisfaz os requisitos do sistema
- **Rastreabilidade**
  - Garantir que o modelo é rastreável até aos requisitos do sistema
- **Heurística**
  - Utilizar soluções conhecidas (padrões) e boas práticas para orientar o processo de modelação

Processo de Desenvolvimento



O processo de desenvolvimento de software consiste na criação da organização de um sistema de forma progressiva, através de diferentes níveis de abstracção, nomeadamente *modelo* (conceptual), *arquitectura* (modelo concreto, orientado para a implementação) e *implementação*, num processo iterativo em que as diferentes actividades de desenvolvimento são alternadas ao longo do tempo em função do conhecimento e do nível de detalhe envolvido

Criação da organização de um sistema de forma progressiva

PROCESSO ITERATIVO

BIBLIOGRAFIA

[Pressman, 2003]

R. Pressman, *Software Engineering: a Practitioner's Approach,* McGraw-Hill, 2003.

[Schach, 2010]

S. Schach, *Object-Oriented and Classical Software Engineering*, 8th Edition, McGraw-Hill, 2010.

[Booch et al., 1998]

G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1998.

[Miles & Hamilton, 2006]

R. Miles, K. Hamilton, *Learning UML 2.0*, O'Reilly, 2006.

[Eriksson et al., 2004]

H. Eriksson, M. Penker, B. Lyons, D. Fado, *UML 2 Toolkit*, Wiley, 2004.

[Douglass, 2009]

B. Douglass, *Real-Time Agility: The Harmony/ESW Method for Real-Time and Embedded Systems Development*, Addison-Wesley, 2009.