DESENVOLVIMENTO DE SISTEMAS EMBEBIDOS

(MESTRADO EM INFORMÁTICA)

- SESSÃO 4: Modelação de Sistemas com Objectos -

JOÃO MIGUEL FERNANDES
Email: miguel@di.uminho.pt
URL: http://www.di.uminho.pt/~miguel

Sumário

1. **Enquadramento**
2. **Métodos Funcionais**
3. **Métodos OO**
4. **Características OO**
5. **Objectos**
6. **UML**

1. Enquadramento (1/2)

- **Objectivos deste módulo**
  - **Introduzir o termo "objecto".**
  - **Apresentar as características mais importantes do paradigma dos objectos para tratar a complexidade dos sistemas.**
  - **Mostrar os mecanismos de modelação disponibilizados pela linguagem UML.**
- **Audiência alvo**
  - **licenciados (com ou sem formação na área das TSI) com responsabilidades e experiência comprovada (desejável!) na análise, concepção e implementação de sistemas baseados em software**

1. Enquadramento (2/2)

- **Bibliografia recomendada**
  - **Booch G. (1994). "*Object-Oriented Analysis and Design with Applications*". Benjamin/Cummings, 2ª edição. ISBN 0-8053-5340-2.**
  - **Cox B.J., Novobilski A.J. (1991). "Object-Oriented Programming: An Evolutionary Approach". Addison-Wesley. ISBN 12667792.**
  - **Booch G., Rumbaugh J., Jacobson I. (1999). "*The Unified Modeling Language User Guide*". Object Technology. Addison-Wesley. ISBN 0-201-57168-4.**
  - **Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W. (1991). "*Object-Oriented Modeling and Design*". Prentice-Hall International. ISBN 0-13-630054-5.**
  - **www.omg.org/uml**

2. Métodos Funcionais (1/5)

- **Métodos**
  - **os métodos a serem aplicados, no âmbito das metodologias de desenvolvimento, correspondem a conjuntos de actividades que organizam a execução de determinadas fases do ciclo de vida do sistema**
  - **se a maior parte dos métodos segue a abordagem multi-vista, já o mesmo consenso não se verifica quanto à ordem de construção das vistas e dos seus conteúdos**

2. Métodos Funcionais (2/5)

- **Métodos funcionais #1**
  - **os métodos funcionais baseiam-se fundamentalmente na modelação do fluxo de dados, recomendando, o processo de especificação, que se inicie o desenvolvimento do sistema pela análise dos dados**
  - **esta análise inicia-se com a caracterização do ambiente do sistema, bem como das suas entradas e saídas, o que leva à definição de um diagrama de contexto do sistema**
  - **as actividades internas ao sistema são decompostas por refinamentos sucessivos de DFDs até que cada actividade acaba por ser especificada num formato textual**

2. Métodos Funcionais (3/5)

- **Métodos funcionais #2**
  - **uma vez que os DFDs não se adequam à especificação do comportamento temporal das actividades, podem ser adicionados STDs aos DFDs para possibilitar a especificação dos estados de todas as actividades**
  - **no que diz respeito à concepção, os métodos funcionais suportam a obtenção da arquitectura conceptual do sistema e a implementação é baseada na transformação dos modelos em programas, codificados à luz do paradigma da programação estruturada**

2. Métodos Funcionais (4/5)

- **Métodos funcionais #3**
  - **os métodos funcionais baseiam-se numa decomposição *top-down* e num refinamento sucessivo dos modelos do sistema, obtendo-se, em cada passo do processo, uma descrição completa do sistema, com níveis de maior detalhe**
  - **esta abordagem tem como principal objectivo a redução da complexidade de sistemas de alguma dimensão a um número controlado de pequenos módulos fáceis de manusear conceptualmente**

2. Métodos Funcionais (5/5)

- **Métodos funcionais #4**
  - **a abordagem dos métodos funcionais tem, no entanto, duas grandes desvantagens:**
    - **as decisões mais críticas relacionadas com a estrutura global do sistema têm que ser efectuadas muito cedo, no processo de desenvolvimento, numa altura em que o problema ainda não está bem percebido**
    - **uma vez que a decomposição é efectuada principalmente no domínio funcional, as estruturas de dados mais importantes tendem a ser globais a todo o sistema, o que coloca problemas significativos à fase de manutenção, quando as representações dos dados têm que ser alteradas como resposta a alterações no ambiente ou nos requisitos do sistema**

3. Métodos OO (1/13)

- **Métodos orientados por objectos #1**
  - **os métodos OO exploram uma abordagem (completamente diferente dos métodos funcionais) inerentemente baseada no encapsulamento da informação e na abstracção dos tipos de dados**
  - **a principal característica dos métodos OO é que a estruturação conceptual das representações, podendo ser realizada segundo uma abordagem *bottom-up*, deve modelar a estrutura do mundo real, no qual se baseia o problema em causa**

3. Métodos OO (2/13)

- **Métodos orientados por objectos #2**
  - **cada objecto é uma abstracção de uma entidade do mundo real, com um estado interno e com uma interface (a única parte do objecto visível externamente) que recebe os estímulos do exterior e que responde segundo a dinâmica interna do objecto**
  - **desta forma, uma rede de objectos cooperantes conceptualiza um modelo natural do problema, estando todas as características dos objectos reais encapsuladas nos seus modelos computacionais**
  - **assim, qualquer alteração no ambiente só tem consequências locais no interior de objectos particulares, não existindo a propagação para fora das fronteiras dos objectos em causa**

3. Métodos OO (3/13)

- **Métodos orientados por objectos #3**
  - **os métodos OO baseiam a especificação de sistemas nos meta-modelos orientados por objectos**
  - **a fase de concepção permite obter modelos arquitecturais menos dependentes da tecnologia de implementação, sendo esta baseada no paradigma da programação orientada por objectos**

3. Métodos OO (4/13)

- ***Methods war***
  - **a partir do início dos anos 90, iniciou-se uma verdadeira "corrida aos métodos" (*methods war*), surgindo uma quantidade enorme de métodos OO como resposta ao descontentamento então existente relativamente aos métodos funcionais**
  - **uma parte representativa dos métodos entretanto propostos foi sumariamente compilada em [Hutt 1994] pela *OMG* (*Object Management Group*), que consiste numa organização que, através do seu *Object Analysis and Design Special Interest Group*, se dedica ao estudo de métodos orientados por objectos**

3. Métodos OO (5/13)

- **Exemplos (OMT #1)**
  - **o método *object modeling technique*, concebido pela equipa de James Rumbaugh na *General Electric*, define quatro fases para chegar à implementação do sistema, tipicamente em *C++***
  - **a primeira fase gera uma especificação do sistema recorrendo a três vistas ortogonais:**
    - **o *modelo dos objectos* que corresponde a uma vista estática do sistema e que recorre aos conceitos de generalização, especialização, classe, herança, agregação, decomposição e relações para estruturar os objectos e definir as interdependências entre eles**
    - **o *modelo dinâmico* que descreve o comportamento de cada objecto recorrendo à linguagem *StateCharts***
    - **o *modelo funcional* que recorre a DFDs para descrever as transformações de dados efectuadas pelos objectos, bem como as suas dependências funcionais e restrições**

3. Métodos OO (6/13)

- **Exemplos (OMT #2)**
  - **o modelo funcional realiza a integração das três vistas, uma vez que as funções que representam as transformações dos dados são invocadas como acções no modelo dinâmico e correspondem às operações dos objectos no modelo dos objectos**
  - **as três fases seguintes definem a arquitectura conceptual do sistema, refinam a concepção de cada tipo de objecto e finalmente chegam à implementação dos objectos por aplicação dos princípios da programação orientada por objectos**
  - **este método garante uma boa continuidade dos modelos, consegue controlar adequadamente a complexidade e suporta objectos intrinsecamente concorrentes**

3. Métodos OO (7/13)

- **Exemplos (OOD #1)**
  - **o método *object-oriented design*, desenvolvido por Grady Booch na *Rational*, não aborda a fase da análise, pelo que a definição do problema e a especificação dos requisitos deverão ser efectuados recorrendo, por exemplo, a um dos métodos funcionais ou outro qualquer orientado por objectos**
  - **após a fase de análise, este método realiza a identificação dos objectos existentes no problema, bem como as suas propriedades**
  - **esta fase é particularmente difícil e é muito dependente da forma como foi realizada a fase de análise**

3. Métodos OO (8/13)

- **Exemplos (OOD #2)**
  - **segue-se a identificação das operações realizadas por cada objecto e das relações com os outros objectos, integrando cada objecto na estrutura da solução**
  - **seguidamente define-se a especificação para cada objecto, através da caracterização da sua interface com o ambiente**
  - **por último, implementam-se os objectos codificando os seus comportamento e os seus estados internos**
  - **por vezes, considera-se o *OOD* como um conjunto de técnicas e notações, em vez de uma metodologia, uma vez que não é fácil identificar qual o modelo do processo de desenvolvimento que lhe está subjacente**

3. Métodos OO (9/13)

- **Exemplos (OOSE #1)**
  - **o método *object-oriented software engineering* (também designado de *Objectory*), concebido pela equipa de Ivar Jacobson na *Objective Systems*, baseia todo o seu processo de desenvolvimento nas funcionalidades oferecidas pelo sistema através da sua interface com o mundo envolvente (*use case-driven approach*)**
  - **como o *OOSE* define os modelos de análise e de concepção com base em sequências de interacção que os utilizadores estabelecem com o sistema, organizadas em cenários típicos de utilização (*usage scenarios*), são produzidos sistemas mais robustos e adaptativos**

3. Métodos OO (10/13)

- **Exemplos (OOSE #2)**
  - **este método define cinco fases:**
    - ***análise de requisitos*, em que se caracteriza a utilização do sistema e se identificam os vários actores que com ele se relacionam**
    - ***análise de robustez*, em que se constrói o primeiro modelo OO do sistema baseado em objectos do tipo controlo, interface e entidade**
    - ***concepção*, em que se definem as interfaces dos objectos e a semântica das operações, refinando os modelos anteriores no sentido da plataforma de implementação através da definição de *packages***
    - ***implementação*, em que se sintetizam os *packages* para a plataforma de implementação recorrendo a linguagens, normalmente, OO;**
    - ***teste*, em que se validam permanentemente todas as decisões tomadas ao longo do processo de desenvolvimento**

3. Métodos OO (11/13)

- **Outros exemplos**
  - ***OOA/D: object-oriented analysis and design* da *Object International* [Coad *et al.* 1991a, Coad *et al.* 1991b]**
  - ***Fusion: object-oriented development* da *Hewlett-Packard* [Coleman *et al.* 1994]**
  - ***SOMA: semantic object modeling approach* da *BIS Information Systems* [Graham 1993]**
  - ***OOIE object-oriented information engineering* da *James Martin and Co.* e da *Intellicorp* [Martin *et al.* 1993]**
  - ***Shlaer/Mellor: object-oriented analysis* da *Project Technology* [Shlaer *et al.* 1992]**
  - ***OO/SSADM: object oriented SSADM* da *CCTA* [Robinson *et al.* 1994]**

3. Métodos OO (12/13)

- **UML #1**
  - **os métodos *OMT*, *OOD* e *OOSE* foram destacados por se terem tornado inequivocamente os mais conhecidos a nível mundial e como tal terem estado na base do surgimento de uma notação unificada, normalizada pela *OMG*, designada de *UML: unified modeling language***
  - **a linguagem *UML*:**
    - **suporta objectos e classes e muitos tipos de associações entre eles, como a agregação, dependência, herança, instanciação e outras**
    - **permite a modelação de cenários para descrever, de uma forma detalhada, os requisitos comportamentais do sistema**
    - **recorre a *StateCharts* para modelar o ciclo de vida dos objectos**
    - **dá acesso à meta-classificação de elementos para definir novos tipos a incorporar no modelo**

3. Métodos OO (13/13)

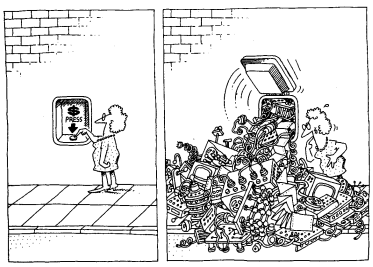
- **UML #2**
  - **desta forma, a linguagem *UML* não define um método, mas "limita-se" a disponibilizar uma notação normalizada que qualquer método OO pode utilizar, podendo ser a solução para a aborrecida confusão de notações e linguagens próprias resultantes da enorme quantidade de métodos OO existentes actualmente**
  - **uma vez que a empresa *Rational* foi a grande promotora da definição da linguagem *UML*, é frequente designar numericamente a versão *Rational* da linguagem *UML*, em vez da correspondente versão numérica definida pela *OMG* (a versão *Rational UML v 1.1* corresponde à versão *OMG UML v 1.0*)**

4. Características OO (1/24)

- **Principais conceitos associados aos objectos**
  - **Identidade,**
  - **Classificação,**
  - **Abstracção,**
  - **Encapsulamento,**
  - **Mensagens,**
  - **Polimorfismo,**
  - **Herança e hierarquia,**
  - **Agregação e composição.**

4. Características OO (2/24)

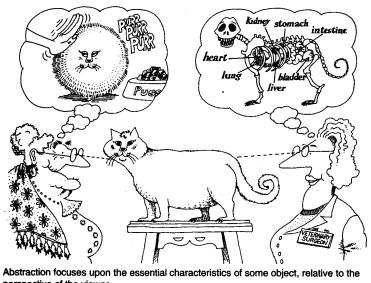
- simplicidade na interface -



The task of the software development team is to engineer the illusion of simplicity.

4. Características OO (3/24)

- abstracção -



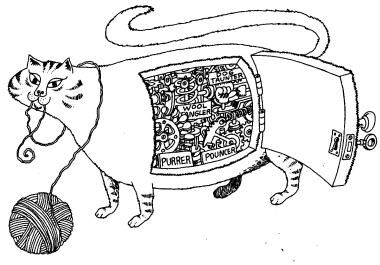
Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

4. Características OO (4/24)

- **Abstracção**
  - **A abstracção consiste na selecção dos aspectos essenciais duma entidade, ignorando aqueles tidos por irrelevantes.**
  - **O analista do sistema deve ter conhecimento dos vários aspectos duma dada entidade, para depois escolher aqueles que lhe pareçam importantes para o sistema a desenvolver.**
  - **A escolha de quais as propriedades essenciais depende do propósito do sistema. Aquilo que numa situação pode ser relevante, pode noutras ser completamente indiferente.**
  - **Quando se usa abstracção, está a admitir-se que o objecto em causa é complexo. Não se pretende modelar esse sistema na totalidade, mas antes escolher parte das suas características.**
  - **Trata-se duma técnica muito importante para permitir lidar com a complexidade dos sistemas.**

4. Características OO (5/24)

- encapsulamento -



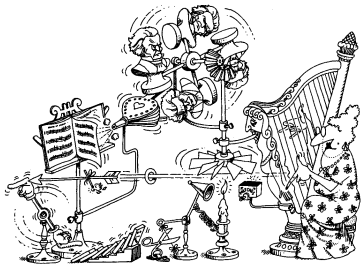
Encapsulation hides the details of the implementation of an object.

4. Características OO (6/24)

- **Encapsulamento**
  - **O encapsulamento consiste em separar o comportamento externo dum objecto dos seus pormenores de implementação.**
  - **A abstracção permite que o projectista se concentre naquilo que deve ser feito. O encapsulamento possibilita que se façam alterações num objecto, sem interferência nos restantes.**
  - **Cada classe deve ser composta por duas partes: uma interface e uma implementação.**
  - **O encapsulamento é o processo de esconder as propriedades dum objecto que não contribuem para a sua caracterização.**
  - **O encapsulamento evita que sejam criadas interdependências num programa, que depois obrigariam a grandes reformulações, quando se pretendesse proceder a alterações, nos requisitos do sistema.**

4. Características OO (7/24)

- mecanismos de interacção -



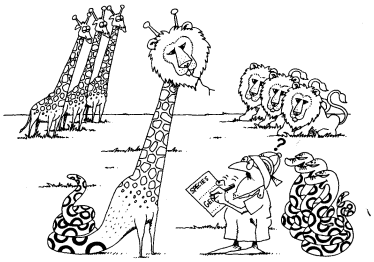
Mechanisms are the means whereby objects collaborate to provide some higher-level behavior.

4. Características OO (8/24)

- **Agregação e composição**
  - **As relações inter-objectos que se expressam, durante a utilização do sistema, através da troca de mensagens entre eles, designam-se por associações.**
  - **Quando um objecto recorre aos serviços dum outro, deve estabelecer-se entre eles uma associação.**
  - **A agregação (composição) representa uma forma especial de associação, que se verifica quando um objecto contém, física ou logicamente, outros.**
  - **Um agregado é mais do que a simples soma das suas partes.**
  - **Um agregado tem os seus próprios atributos e operações que não têm de estar directamente relacionados com as partes.**
  - **Apesar de ser um conceito intuitivo, a agregação nem sempre é bem utilizada na modelação de sistemas.**

4. Características OO (9/24)

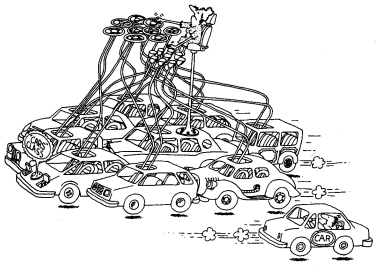
- classificação -



Classification is the means whereby we order knowledge.

4. Características OO (10/24)

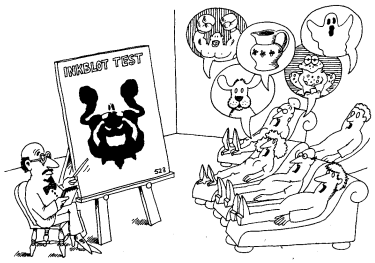
- classe -



A class represents a set of objects that share a common structure and a common behavior.

4. Características OO (11/24)

- critério de classificação -



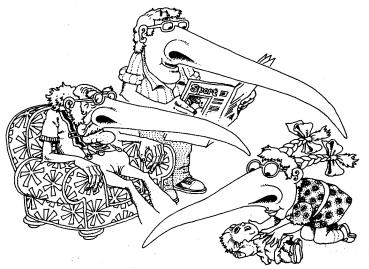
Different observers will classify the same object in different ways.

4. Características OO (12/24)

- **Classificação**
  - **A classificação significa que os objectos com a mesma estrutura de dados e o mesmo comportamento são agrupados numa mesma classe.**
  - **Um atributo é um elemento de modelação que descreve uma instância, caracterizando algo de importante e significativo acerca da natureza dessa instância.**
  - **Uma operação é uma acção ou transformação desempenhada por um objecto ou a que o objecto se sujeita.**
  - **Uma classe (de objectos) é uma abstracção que, para uma dada aplicação, define apenas as propriedade mais relevantes e ignora as restantes.**
  - **A escolha de quais as classes com relevância para uma determinada aplicação é arbitrária e depende fortemente desta.**

4. Características OO (13/24)

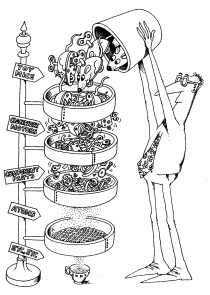
- herança -



A subclass may inherit the structure and behavior of its superclass.

4. Características OO (14/24)

- hierarquia -



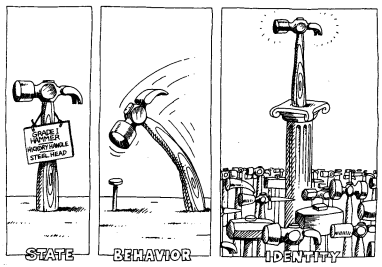
Abstractions form a hierarchy.

4. Características OO (15/24)

- **Herança e hierarquia**
  - **A herança é definida como o mecanismo que facilita a construção de novas classes (subclasses) a partir de outras classes (superclasses) e permite a especialização e a generalização de componentes.**
  - **A herança é um mecanismo valioso e poderoso que facilita a reutilização e permite expressar os aspectos comuns entre classes.**
  - **A herança inclui quatro aspectos principais:**
    - **Uma subclasse herda os atributos das suas superclasses.**
    - **Uma subclasse herda as operações das suas superclasses.**
    - **Uma subclasse pode adicionar novos atributos àqueles existentes nas superclasses.**
    - **Uma subclasse, relativamente aos métodos definidos nas superclasses, pode reescrevê-los ou adicionar novos.**

4. Características OO (16/24)

- objecto -



An object has state, exhibits some well-defined behavior, and has a unique identity.

4. Características OO (17/24)

- **Identidade**
  - **A identidade significa que os dados do sistema podem ser quantificados em entidades discretas, a que se dá o nome de objectos.**
  - **Cada objecto tem a sua própria identidade, o que significa que dois objectos são distintos, mesmo que tenham todos os seus atributos iguais.**
  - **O termo identidade pressupõe que cada objecto é distinto pela sua própria existência e não pelo valor dos seus atributos.**
  - **Os objectos podem ser coisas reais (tangíveis), como maçãs, cães, televisores, impressoras, ou máquinas, mas também podem ser entidades puramente conceptuais como contas bancárias, casamentos, ou listas.**

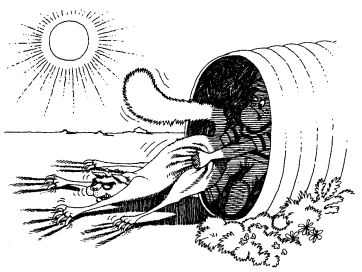
4. Características OO (18/24)

- modularidade -

Modularity packages abstractions into discrete units.

4. Características OO (19/24)

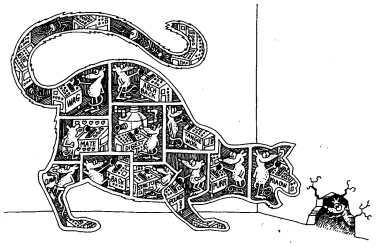
- persistência -



Persistence saves the state and class of an object across time or space.

4. Características OO (20/24)

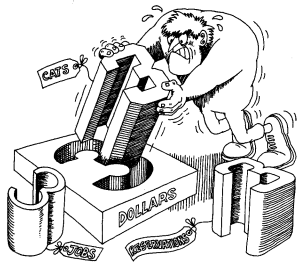
- concorrência -



Concurrency allows different objects to act at the same time.

4. Características OO (21/24)

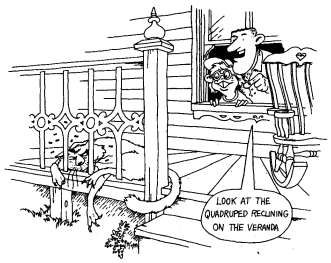
- tipos -



Strong typing prevents mixing abstractions.

4. Características OO (22/24)

- classe vs. objecto -



Classes and objects should be at the right level of abstraction: neither too high nor too low.

4. Características OO (23/24)

- **Mensagens**
  - **Uma acção é iniciada, em linguagens orientadas ao objecto, através da transmissão duma mensagem ao objecto responsável por essa acção.**
  - **Se um objecto aceita uma mensagem que lhe é enviada, está a assumir o compromisso de desempenhar essa operação.**
  - **Todos os objectos duma determinada classe, como resposta a uma mensagem do mesmo tipo, usam o mesmo método.**
  - **Os sistemas orientados ao objecto pressupõem um modelo de comunicação do tipo cliente/servidor.**
  - **Este modelo de comunicação pressupõe uma dependência reduzida entre o cliente e o servidor, no sentido deste não precisar dum conhecimento prévio de quais os clientes que a ele recorrem para desempenhar uma operação em que é especialista.**

4. Características OO (24/24)

- **Polimorfismo**
  - **O polimorfismo significa que a mesma operação pode apresentar comportamentos distintos para classes diferentes.**
  - **Se uma dada operação é polimórfica, então existe mais do que um método que a implementa.**
  - **Em termos práticos, e segundo uma outra perspectiva, o polimorfismo significa também que o emissor duma dada mensagem não precisa de conhecer a classe a que pertence o objecto receptor.**
  - **Outro conceito relacionado com o polimorfismo é a sobreposição de operadores (operator overloading) que se refere à possibilidade de se usar o mesmo símbolo para propósitos distintos, consoante o contexto de utilização.**

5. Objectos (1/11)

- **Visão informal dos objectos**
  - **A orientação ao objecto é uma técnica para modelação de sistemas, que faculta um conjunto significativo de conceitos e mecanismos para esse fim.**
  - **Quando se recorre ao paradigma dos objectos para modelar um dado sistema, passa-se a olhar para este como um conjunto de objectos que interagem entre si.**
  - **Uma vez que, no dia-a-dia, as pessoas também lidam com objectos, é relativamente simples pensar de forma semelhante quando se pretende construir um modelo.**
  - **Um sistema, no âmbito da modelação por objectos, é composto por vários objectos, que correspondem a entidades reais.**

5. Objectos (2/11)

- **Visão informal dos objectos**
  - **Um objecto contém um conjunto de dados (informação), que se designam atributos, que o descrevem de alguma forma.**
  - **Para manipular estes atributos, devem definir-se as operações que afectam ou permitem usar aqueles.**
  - **A única parte do objecto visível do exterior são as suas operações, mais propriamente as respectivas interfaces, estando todo o resto escondido.**
  - **A relação ou associação entre os objectos também tem de ser modelada.**
  - **Podem existir outros tipos de relação entre os objectos, nomeadamente a composição ou a agregação.**

5. Objectos (3/11)

- **Visão informal dos objectos**
  - **O comportamento temporal do modelo dos objectos é conseguido, devido às relações dinâmicas entre objectos, pois estas permitem que os objectos enviem estímulos (mensagens) uns aos outros.**
  - **Um objecto, ao receber uma mensagem, realiza uma determinada operação e pode provocar novas mensagens em outros objectos.**
  - **Os objectos só respondem às mensagens para as quais estão preparados.**
  - **Os pedidos que são feitos a um objecto podem ou não ser satisfeitos, dependendo este facto das operações que aquele está habilitado a desempenhar.**

5. Objectos (4/11)

- **Definição formal de objecto**
  - **Objecto: Tudo aquilo que se apresenta aos sentidos; aquilo que se apresenta à vista. (...) Em sentido geral, qualquer coisa; peça, artigo (...).**
    Grande Enciclopédia Portuguesa e Brasileira (1975). Editorial Enciclopédia, Lisboa e Rio de Janeiro.
  - **Objecto: Do latim *objectum*, do verbo *objicere* (lançar ou pôr em frente, propor, expor, opor). Etimologicamente, objecto é o que está à frente a, o que se opõe ao sujeito. Por extensão, significa o que é, as coisas. Palavra do vocabulário comum que se restringe em sentidos muito precisos nas diversas disciplinas científicas e filosóficas (...).**
    Enciclopédia Luso-Brasileira de Cultura (1973).Verbo Editora, Lisboa, Portugal.

5. Objectos (5/11)

- **Definição formal de objecto**
  - **O conceito de objecto, no projecto de software, surgiu, pela primeira vez, num construtor da linguagem de programação Simula, nos anos 60.**
  - **Mais recentemente, este conceito serviu de base a diversas linguagens de programação orientadas ao objecto (SmallTalk, ObjectiveC, Eiffel, C++ e Java), bem como a outras linguagens não orientadas ao objecto (Ada e Modula-2).**
  - **Diversos métodos de análise e concepção fizeram também uso, como elemento basilar, do conceito de objecto.**
  - **Um objecto pode ser visto como uma entidade, cujo comportamento se caracteriza pelo conjunto de acções que executa e que requer a outros objectos.**

5. Objectos (6/11)

- **Definição formal de objecto**
  - ***We defined an object as a concept, abstraction or thing with crisp boundaries and meaning for the problem at hand. Objects serve two purposes: They promote understanding of the real world and provide a practical basis for computer implementation. Decomposition of a problem into objects depends on judgment and the nature of the problem.***
    James Rumbaugh, 1991.
  - ***An object represents an individual, identifiable item, unit or entity, either real or abstract, with a well defined role in the problem domain. An object may be recognized by the data it carries, by its behavior (i.e. how it responds to events in its environment), and by the processing that it performs.***
    Mark M. Smith e Stephen R. Tockey, 1988.

5. Objectos (7/11)

- **Definição formal de objecto**
  - ***An object has state, behavior, and identity; the structure and behavior of similar objects are defined in their common class.***
    Grady Booch, 1991.
  - ***An object is a model of a real-world entity or a software solution entity that combines data and operations in such a way that data are encapsulated in the object and are accessed through the operations. An object thus provides operations for other objects, and may in turn also require operations of another object. An object may have state, either explicitly to provide control or implicitly in terms of the value of the internal data. An object may be a class or an instance created as a parameterisation of a class.***
    Peter J. Robinson, 1992.

5. Objectos (8/11)

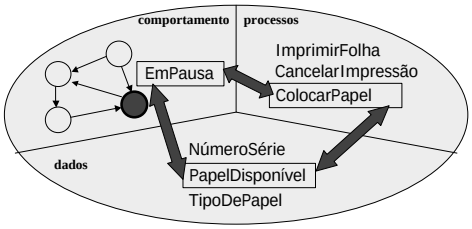
- **Características dos objectos**
  - **Encapsula informação.**
  - **Indica o tipo dos seus atributos.**
  - **Disponibiliza operações a outros objectos.**
  - **Pode requerer e usar operações doutros objectos.**
  - **Tem visibilidade reduzida dos outros objectos.**
  - **Pode decompor-se noutros (sub-)objectos que, conjuntamente, exibem o mesmo comportamento.**
  - **É um modelo duma entidade real ou uma entidade relevante para a solução final em software (ou hardware).**
  - **É referido pelo seu nome.**
  - **Pode ser uma instância duma classe.**
  - **Pode implementar-se com um tipo abstracto de dados.**
  - **Pode ter estado (visto como uma máquina de estados).**

5. Objectos (9/11)

- **Perspectivas de modelação dos objectos**
  - **Um objecto inclui, em simultâneo, as três perspectivas fundamentais sob as quais um sistema é, actualmente, modelado:**
    - **os dados que incorpora,**
    - **o comportamento dinâmico que exibe, e**
    - **os processos que realiza.**

5. Objectos (10/11)

- **Relação entre perspectivas de modelação**



5. Objectos (11/11)

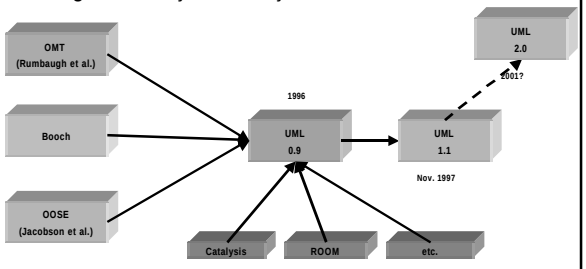
- **Aspectos dum objecto (sistema)**
  - **Dados ou estrutura: Descrevem as características estáticas do objecto, nomeadamente os seus atributos relevantes para a aplicação em causa.**
  - **Controlo ou comportamento global: Indica as dependências temporais e dinâmicas entre a ocorrência de eventos (externos) e a execução das acções internas, o que implica alterações no estado e nos dados do objecto.**
  - **Processos ou comportamento local: Descrevem as actividades internas do objecto, em função dos estímulos externos a que está sujeito.**
- **Os dados dum objecto são, normalmente, a primeira perspectiva a ser considerada, pois são considerados mais estáveis que as funções ou o comportamento.**

6. UML (1/32)

- **O que é UML**
  - **UML é uma linguagem para expressar a funcionalidade, a estrutura e as relações de sistemas complexos.**
  - **Standard OMG (Object Modeling Group) para modelação de sistemas.**
  - **Em 2001, é previsível que se torne um standard ISO.**
  - **A definição de UML inclui o respectivo meta-modelo, o que permite conhecer, caso se use uma linguagem formal para o definir, a respectiva semântica.**
  - **O meta-modelo UML não está definido de forma formal, pois a sintaxe dos construtores está especificada numa linguagem precisa, mas a respectiva semântica só foi descrita em inglês.**
  - **UML pode classificar-se, actualmente, como semi-formal.**

6. UML (2/32)

- **Origem e evolução da notação UML**



6. UML (3/32)

- **Diagramas**
  - **A notação UML é composta por diversos diagramas que permitem descrever os aspectos mais relevantes dos sistemas a implementar segundo a abordagem orientada ao objecto.**
  - **Cada um dos diagramas foca uma dada vista do sistema e propositadamente enfatiza alguns aspectos e negligencia outros.**
  - **A notação UML é independente da linguagem de programação e do processo de desenvolvimento adoptados**
  - **UML está dotada dum mecanismo de extensão (estereótipos) que facilita a sua adaptação a situações que apresentem mecanismos não previstos inicialmente na notação.**
  - **Um estereótipo é a classe duma entidade no meta-modelo UML.**

6. UML (4/32)

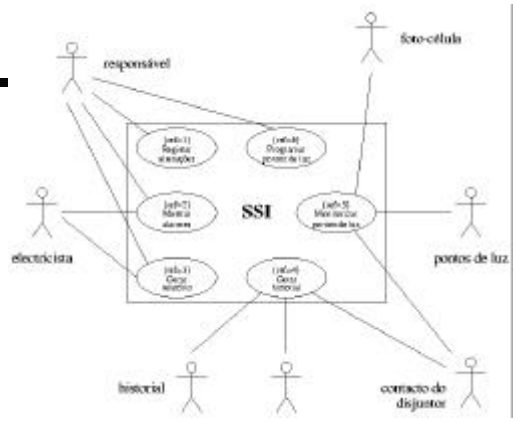
- **Diagramas**
  - **No contexto dos sistemas embebidos, os seguintes diagramas UML foram considerados indispensáveis para especificar e documentar os vários aspectos que importa considerar na modelação dum sistema complexo:**
    - **Diagramas de casos de uso (use case diagrams).**
    - **Diagramas de classes (class diagrams).**
    - **Diagramas de objectos (object diagrams).**
    - **Diagramas de interacção (interaction diagrams).**
    - **Diagramas de estados (state diagrams).**

6. UML (5/32)

- **Diagramas**
  - **Diagrama de casos de uso: Mostrar um conjunto de casos de uso e de actores e as respectivas relações. São importantes para organizar e modelar as funcionalidades do sistema.**
  - **Diagrama de classes: Apresentar um conjunto de conceitos, tipos e classes e respectivas relações.**
  - **Diagrama de objectos: Exibir um conjunto de instâncias e a forma como elas se relacionam. Como sucede com os diagramas de classes, a estrutura estática do sistemas é mostrada, mas a ênfase é colocada em configurações estáticas num dado instante.**
  - **Diagrama de interacção: Revelar como vários objectos colaboram (trocam mensagens) num dado caso de uso.**
  - **Diagrama de estados: Especificar o comportamento dum objecto, englobando, potencialmente, muitos casos de uso.**

6. UML (6/32)

- **Diagramas de casos de uso**
  - **Elementos básicos:**
    - **casos de uso (as funcionalidades que esse sistema deve desempenhar)**
    - **actores (o que existe fora do sistema).**



6. UML (8/32)

- **Diagramas de casos de uso**
  - **Um caso de uso é uma típica interacção entre um utilizador e um sistema computacional.**
  - **Os casos de uso permitem captar, junto dos clientes, os requisitos do utilizador, dado que o vocabulário usado é o dos clientes e não o dos projectistas.**
  - **Os casos de uso dum sistema constituem uma decomposição funcional do comportamento desse sistema, sem lhe impor qualquer estrutura interna.**
  - **UML dá uma grande importância aos diagramas de casos de uso, uma vez que é com base neles que, segundo os seus proponentes, se pode planear e sustentar todo o desenvolvimento do sistema em causa.**
  - **Devem usar-se verbos para caracterizar os casos de uso, o que indica que há uma dada operacionalidade associada.**

6. UML (9/32)

- **Diagramas de casos de uso**
  - **Um actor representa um papel que um dado utilizador pode ter em relação a um determinado sistema, quando com ele interactua.**
  - **Os casos de uso são realizados por actores, podendo um mesmo actor desempenhar vários casos de uso e sendo também possível que um dado caso de uso seja executado por vários actores.**
  - **Note-se que várias pessoas podem ser representadas pelo mesmo actor no diagrama.**
  - **Um actor é assim uma representação abstracta dum tipo de pessoa que interage com o sistema.**
  - **Note-se também que a mesma pessoa pode desempenhar vários papéis relativamente ao mesmo sistema, papéis esses que darão origem a actores distintos no respectivo diagrama.**

6. UML (10/32)

- **Diagramas de casos de uso**
  - **A identificação dos actores do sistema facilita a definição das funcionalidades do sistema, feita através da especificação dos casos de uso.**
  - **Um caso de uso consiste numa forma particular de usar o sistema, representando parte da sua funcionalidade total.**
  - **Cada caso de uso constitui um conjunto completo de eventos, iniciado por um actor, e explicita a interacção que se pode observar entre esse actor, o sistema e, eventualmente, outros actores.**
  - **O conjunto de todos os casos de uso permite especificar todas as formas distintas de interagir com o sistema.**

6. UML (11/32)

- **Diagramas de classes**
  - **Para sistemas OO, são necessários diagramas de classes, para indicar as classes existentes e as suas relações.**
  - **Este tipo de diagrama é sempre contemplado por todas as metodologias OO, uma vez que o conceito de classe é fundamental neste paradigma.**
  - **Um diagrama de classes tem por objectivo evidenciar a estrutura estática de conceitos, tipos e classes.**
  - **Os conceitos mostram como os utilizadores vêem o domínio da aplicação, independentemente da forma como eles são, na prática, implementados.**
  - **Estes diagramas permitem também descrever os tipos de objectos (as classes) que o sistema pode contemplar e as formas de inter-relação entre eles.**

6. UML (12/32)

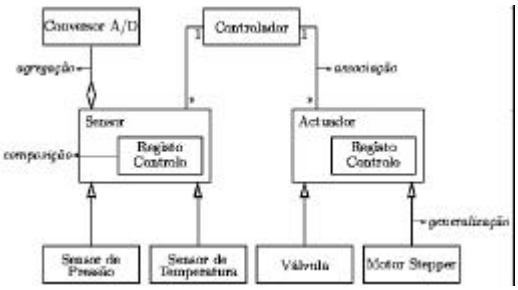
- **Diagramas de classes**
  - **Podem ainda ser mostrados os atributos e as operações de cada uma das classes, caso tal seja relevante para o nível de abstracção em causa.**

6. UML (13/32)

- **Diagramas de classes**
  - **Em UML, existem 4 tipos de relações entre objectos, que podem ser mostradas entre as classes:**
    - **Associação: Relação entre objectos que se manifesta em tempo de execução através da troca de mensagens entre eles. Quando um objecto usa os serviços dum outro, devem ligar-se estes com uma associação.**
    - **Agregação: Forma especial de associação, que se verifica quando um objecto contém, física ou logicamente, outros. Os componentes podem ser partilhados por vários agregados.**
    - **Composição: Forma mais restrita de agregação, pois os componentes dum agregado por composição não podem ser partilhados por outros agregados. O agregado é responsável pela criação/destruição dos seus componentes.**
    - **Generalização: Verifica-se quando uma classe é uma especialização duma outra. A subclasse herda todas as características da superclasse, podendo adicionar novos atributos ou operações.**

6. UML (14/32)

- **Diagramas de classes**

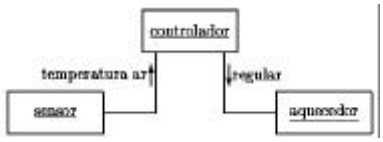


6. UML (15/32)

- **Diagramas de objectos**
  - **Para documentar os objectos concretos que compõem o sistema, bem como as suas inter-relações, são usados diagramas de objectos.**
  - **Estes são, por natureza, estáticos, pois apenas mostram um conjunto de objectos que trocam mensagens entre si, omitindo o fluxo de controlo e a ordem dos eventos.**
  - **Os diagramas de objectos são idênticos aos diagramas de classes, exceptuando o facto de mostrarem instâncias (ou objectos) em vez de classes.**
  - **Os objectos são desenhados como rectângulos, sendo os nomes sublinhados para mais facilmente os distinguir de classes.**

6. UML (16/32)

- **Diagramas de objectos**



ão também necessárias representações gráficas que permitam captar os aspectos dinâmicos relativos à troca de mensagens entre objectos, a que se chamam diagramas de interacção.

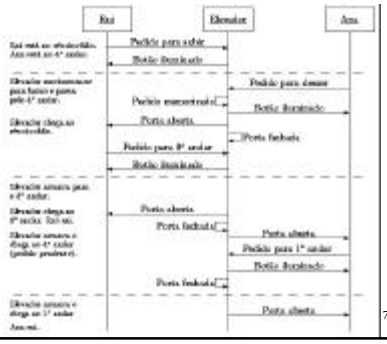
- Os diagramas de estado associados a classes não servem para este efeito, pois descrevem as mudanças internas de estado das instâncias dessas classes e não entre um conjunto de objectos.
- Um diagrama de interacção representa uma instância dum caso de uso.
- Os diagramas de interacção descrevem a forma como um grupo de objectos comunica entre si.
- Tipicamente, um diagrama de interacção capta o comportamento dum dado cenário, mostrando os objectos e as mensagens que são trocadas entre eles, nesse caso de uso.

6. UML (18/32)

- **Diagramas de interacção**
  - Existem dois tipos diferentes de diagramas de interacção: diagramas de sequência e diagramas de colaboração.
  - Estes diagramas permitem captar os requisitos do sistema, podendo portanto ser utilizados na fase de análise.
  - Podem também usar-se, durante o teste do sistema, para comparar o funcionamento real do sistema (ou do protótipo, ou do modelo executável) com aquele que foi especificado.
  - Um diagrama de sequência mostra a sequência de mensagens trocadas entre objectos.

6. UML (19/32)

- **Diagramas de interacção**
  - diagrama de sequência

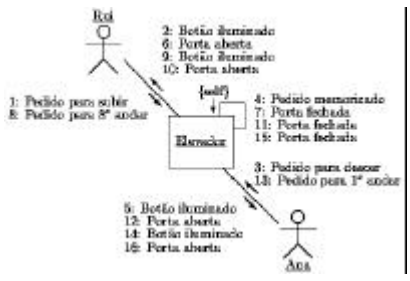


6. UML (20/32)

- **Diagramas de interacção**
  - Um diagrama de colaboração também pode ser usado para descrever possíveis cenários dum sistema.
  - Os diagramas de colaboração podem ser vistos como diagramas de objectos que apresentam as mensagens que circulem entre eles e por que ordem.
  - O mesmo tipo de informação é mostrado pelos diagramas de sequência e de colaboração.
  - A diferença reside no facto de os primeiros focarem a sequência de mensagens, enquanto que os últimos enfatizam a estrutura dos objectos que interactuam.
  - Nos diagramas de colaboração, identificar a sequência de mensagens não é tão óbvio como sucede nos diagramas de sequência.

6. UML (21/32)

- **Diagramas de interacção**
  - diagrama de colaboração



6. UML (22/32)

- **Diagramas de estados**
  - **Um diagrama de classes não permite determinar o comportamento dinâmico das instâncias dessas classes**
  - **Para as classes mais complexas, o uso duma notação que tenha subjacente um meta-modelo baseado em estados é crucial.**
  - **O uso de diagramas de estado, para definir o comportamento dum sistema, popularizou-se no domínio do hardware, mas a sua capacidade de modelação é útil noutras disciplinas (software, compilação, comunicações, sistemas operativos, simulação, multimédia, interfaces homem-máquina, etc.).**
  - **Os diagramas de estados definem o comportamento dinâmico duma classe, explicitando todos os estados em que cada um desses objectos se pode encontrar e as transições entre estados provocadas por condições a que esse objecto é sensível.**

6. UML (23/32)

- **Diagramas de estados**
  - **Num diagrama de estados convencional, em cada instante, um e um só estado está activo.**
  - **Um estado dum sistema é representado por um conjunto de variáveis, cujos valores contêm toda a informação necessária sobre o passado do sistema e que, simultaneamente, condicionam o comportamento futuro do sistema.**
  - **Um estado representa um período de tempo durante o qual o sistema exibe um tipo específico de comportamento.**
  - **Uma definição de estado é a seguinte: "*A state is an ontological condition that persists for a significant period of time, is distinguishable from other such conditions, and is disjoint with them. A distinguishable state means that it differs from other states in the events it accepts, the transitions it takes as a result of accepting those events, or the actions it performs.*"**

6. UML (24/32)

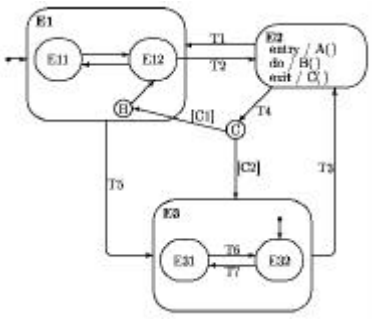
- **Diagramas de estados**
  - **Em UML, existem dois formalismos diferentes para especificar máquinas de estados: state-charts e diagramas de actividades.**
  - **Os state-charts são usados quando a transição entre estados dispara devido principalmente à ocorrência dum evento significativo.**
  - **Os diagramas de actividades mostram-se apropriados quando a transição de estados ocorre, sobretudo, por causa da conclusão da actividade executada no estado e não devido à ocorrência de eventos, sejam eles síncronos ou assíncronos.**
  - **Uma vez que os sistemas de interesse neste trabalho incluem sistemas reactivos, restringir-se-á a discussão aos statecharts.**

6. UML (25/32)

- **Diagramas de estados**
  - **Os state-charts estendem os diagramas de estados mais convencionais em três vertentes relacionadas com hierarquia, concorrência e comunicação.**
  - **Apesar destas extensões permitirem obter modelos mais simples, compactos e legíveis, os state-charts são matematicamente equivalentes a máquinas de estados.**

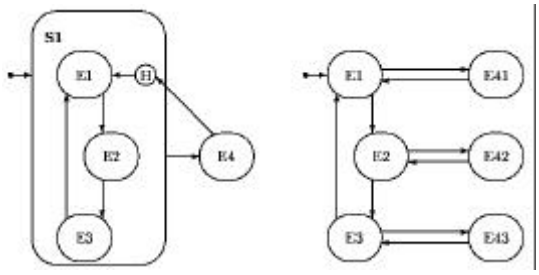
6. UML (26/32)

- **Diagramas de estados**
  - **statechart**



6. UML (27/32)

- **Diagramas de estados**
  - **statechart**

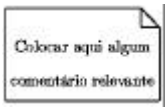


6. UML (28/32)

- **Outros mecanismos de modelação**
  - **A definição de UML baseou-se, em forte medida, nas notações associadas às metodologias de Booch, Objectory e OMT.**
  - **Houve, na sua definição, a preocupação de consolidar essas notações, mas também de introduzir alguns mecanismos avançados de modelação.**
  - **Nesse sentido, foram adicionados a UML, entre outros, os estereótipos, os valores etiquetados e as restrições.**
  - **Estes conceitos unificam uma série de características das notações daquelas três metodologias e permitem estender a notação UML.**

6. UML (29/32)

- **Outros mecanismos de modelação**
  - **Uma nota de texto (*text note*) é um elemento gráfico que pode ser colocado em qualquer tipo de diagrama e que não tem qualquer valor semântico.**
  - **A sua utilização reduz-se à colocação de comentários e notas para aumentar a legibilidade dos modelos.**
  - **Por exemplo, relativamente a uma classe, é conveniente introduzir uma nota de texto onde se indicam comentários relativos àquela, incluindo, por exemplo, responsabilidades, propósito, restrições (pré e pós-condições) e regras.**



6. UML (30/32)

- **Outros mecanismos de modelação**
  - **Um estereótipo (*stereotype*) é um mecanismo que permite estender a notação UML, por forma a incluir elementos não previstos na notação base.**
  - **Os estereótipos permitem rotular uma dada classe, tratando-se duma forma de classificar as classes (meta-classificação).**
  - **Cada classe pode ter associados zero ou mais estereótipos.**
  - **Os estereótipos duma classe são listados por cima do nome.**
  - **Os estereótipos são indicados entre aspas.**
  - **Exemplos: <<control>>, <<uses>> e <<extends>>.**
  - **Todos os mecanismos de modelação definidos em UML podem ser estereotipados, sendo, por ventura, o exemplo mais óbvio as mensagens.**

6. UML (31/32)

- **Outros mecanismos de modelação**
  - **Um valor etiquetado (*tagged value*) é uma par (etiqueta,valor) que se associa a qualquer elemento de modelação para guardar informação.**
  - **Os valores etiquetados constituem uma forma simples de estender o meta-modelo UML e podem ser usados, por exemplo, para passar informação aos utilitários de geração automática de código.**
  - **É uma possibilidade a explorar no desenvolvimento de sistemas embebidos (fases de concepção e implementação), sobretudo durante a partição hardware/software.**
  - **Cada valor etiquetado é composto por uma etiqueta e por um valor, com a seguinte forma: {etiqueta = valor}.**
  - **Exemplo: {autor = Miguel}.**

6. UML (32/32)

- **Outros mecanismos de modelação**
  - **Uma restrição é uma condição adicional que se aplica a um dado elemento de modelação e é sempre indicada entre chavetas "{}".**
  - **Por exemplo, a restrição {abstract} é usada numa classe para indicar que se trata duma classe abstracta**
  - **A restrição {instantiable} é usada para indicar que a classe é concreta.**
  - **Nos diagramas de sequência, podem usar-se restrições para explicitar marcas temporais.**