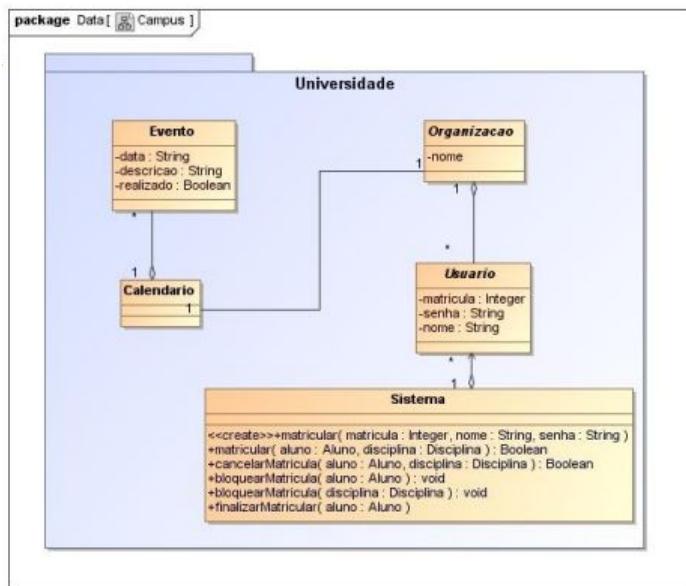


Estágio I

Diagrama de classes (estrutural)



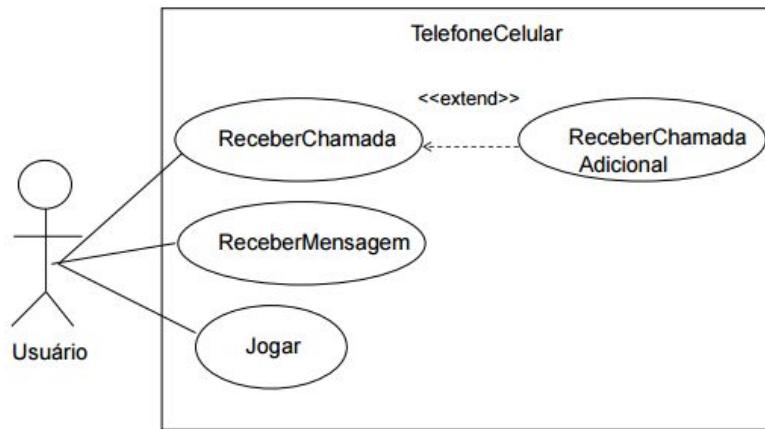
- Classe concreta é formada com a implementação das operações, diferente de uma classe abstrata (que não pode ser instanciada).
- Propriedades:
 - classe
 - leaf
 - root
 - multiplicidade
- Operações
 - leaf
 - isQuery
 - sequential
 - guarded
 - concurrent
- Interfaces são artefatos que possuem um conjunto de operações que especificam um serviço
- Relacionamento
 - generalização: subclasse/superclasse; herança múltipla.
 - associação: agregação é associação mais fraca; todo é responsável pela parte;
 - dependência: classe
 - realização
- Cada elemento do governo deve pertencer, no máximo, a um pacote.

Diagrama de objetos (estrutural)



- Diagrama que mostra um conjuntos de objetos e seus relacionamentos instanciados em dado momento.
- Para simulação e validação
- Objetos são semelhantes as classes mas não possuem operações
- Podem ser:
 - rotulados
 - anônimos
 - órfãos
 - ativos

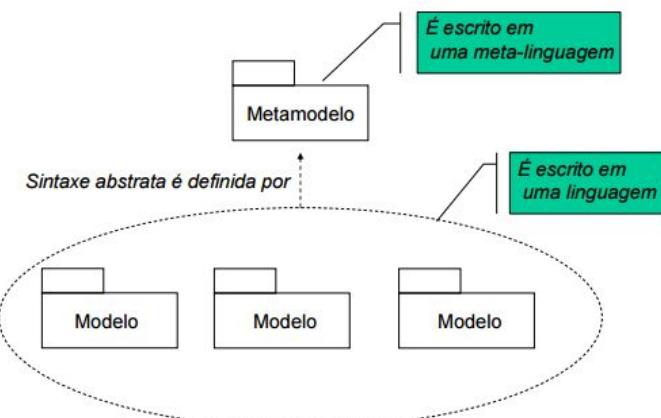
Diagrama de casos de uso (comportamental)



- Um caso de uso especifica o comportamento de um sistema, ou parte dele. Representa um requisito funcional de todo o sistema.
- Um ator representa um conjunto coerente de papéis que os usuários dos casos de uso desempenham quando interagem com eles.
- Variantes:
 - especialização
 - inclusão
 - extensão

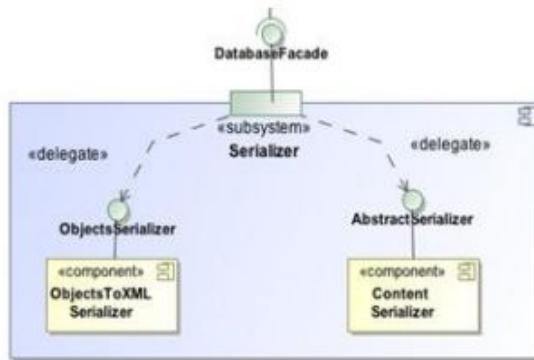
- Assunto, ator, associação
- Fluxo de eventos
 - tipos:
 - principal
 - alternativo
 - de exceção
 - componentes:
 - nome
 - identificador
 - descrição
 - objetivo
 - pré-condições
 - pós-condições
 - fluxo principal de ações
 - fluxo alternativo de ações
- Cenário é uma sequência específica de ações que ilustra o comportamento de um caso de uso
- Relacionamento
 - generalização
 - inclusão
 - extensão
- Mais abstrato, flexível e informal

Meta modelos



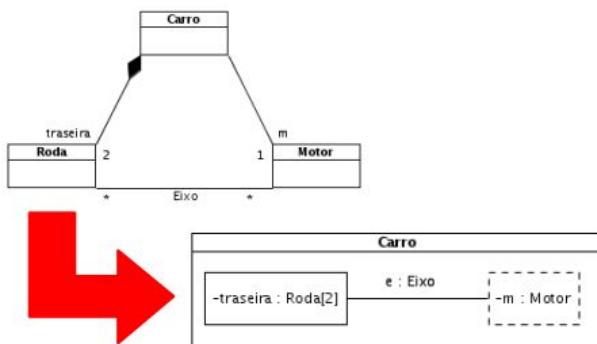
- nam

Diagrama de Componentes (estrutural)



- Representa a arquitetura do software
- Unidade modular com interfaces bem definidas
- Componente é como uma caixa preta, interfaces.
- Serviços provados e requeridos
- Componentes podem possuir:
 - portas
 - outros componentes
- ~setinhas~ Classes internas do componente
- Conectores ligam os componentes às portas internas e externas

Diagrama de estruturas compostas (estrutural)

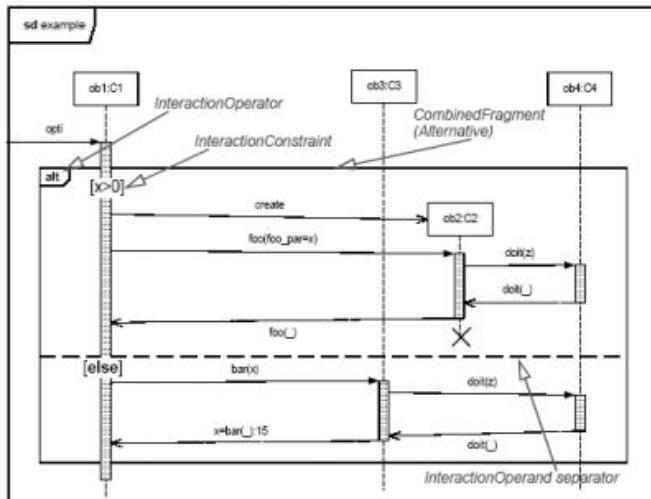


- Mostra a estrutura interna de um componente ou classes e suas possíveis colaborações.
- Permite visualizar a relação entre os elementos em tempo de execução.
- Elementos:
 - propriedade: retângulo contido no corpo de uma classe/componente (quando pontilhado é porque a propriedade só se relaciona com outras na mesma instância da classe/componente)
 - porta
 - conector
 - colaboração: associação de entidades específicas do contexto com os elementos da colaboração.

Diagrama de Interação

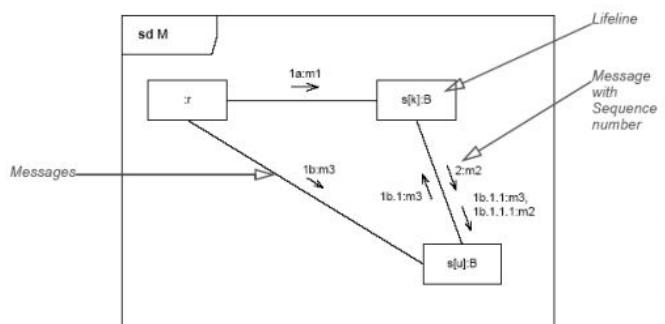
- Modela fluxo de controle dentro de uma classe, operação, componente, caso de uso, sistema....

Diagrama de sequência (comportamental)



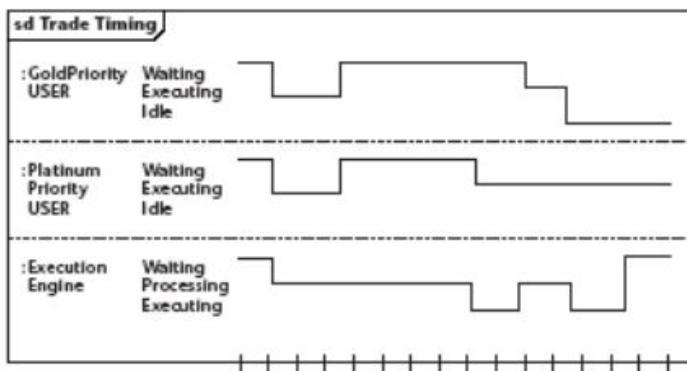
- Foco na troca de mensagens entre objetos ao decorrer do tempo.
- Fragmentos:
 - Alternative (alt)
 - Option (opt)
 - Parallel (par)
 - Critical region (critical)
 - Loop
 - Break
 - Negative
 - Assertion (assert)
 - Ignore/consider
- Interação: referencia à outro fragmento. (ref)

Diagrama de comunicação (comportamental)



- Enfatiza a organização estrutural dos objetos que enviam e recebem mensagens
- Interação entre objetos é explícita
- Um dos novos diagramas de UML 2.0
- Oferece uma visão geral do fluxo de controle

Diagrama de tempo (comportamental)



- Mostra as mudanças de estados em linhas de vida ao decorrer do tempo

Estágio II

OCL

- Surge da necessidade de uma linguagem formal, de fácil escrita e leitura e mais próxima da implementação do modelo.
- Linguagem escrita para a restrição sobre objetos, que pode ser aplicada em um ou mais valores de um sistema ou modelo orientados a objetos.
- Complementa UML.
- Referenciar: classificador. propriedade

Especificar invariantes

- Expressão cujo valor deve ser verdadeiro para todas as instâncias do seu contexto.
- Estereótipo <<invariante>>

```
context nome_tipo inv:  
expressao OCL
```

Operações

- Consulta: body

```
context ProgramaFidelidade::obtemServicos() : Set(Servico)  
body: parceiros.servicosOferecidos->asSet()
```

- Modificação: pre e post

```
context ProgramaFidelidade::cadastrar(c : Cliente) : void  
pre parametroOk: not cliente->includes(c)  
post resultadoOk: cliente = cliente@pre->including(c)
```

Regras de derivação

- Determinar valores de elementos variados

```
context Cartao:::nomeTratamentoCliente : String  
derive:  
proprietario.pronomeTratamento.  
concat(' ').concat(proprietario.nome)
```

Valores iniciais

```
context Conta::Pontos
init: 0
```

Definir novos atributos ou operações

```
context ProgramaDeFidelidade
def: getServicosPorNivel(nomeNivel : String) : Set
    (Serviço)
= nivelServico->select(nome = nomeNivel)
    .servicosDisponiveis->asSet()
```

Mais usos

- Especificar regras dedutivas
- Especializar inicializações de propriedades
- Descrever guardas
- Navegação
 - classe.papel.propriedade
- Especificar expressões sobre UML

Coleção

Tipos coleção

- Set: coleção com instâncias de um tipo OCL. Não tem elementos duplicados.
 - gerado da navegação sobre uma coleção com multiplicidade maior que 1.
- OrderedSet: Set ordenado
- Bag: Set com objetos duplicados.
 - gerado da navegação sobre uma coleção com multiplicidade (de ambas) maior que 1.
- Sequence: Bag ordenado
- Obs.: operações sobre uma coleção não a modificam, mas geram uma nova coleção.

Operações em coleções

- -> size()
- -> count(object)
- -> includes(object)
- -> includesAll(collection)
- -> excludes(object)
- -> excludesAll(collection)

- \rightarrow isEmpty
- \rightarrow including(object)
- \rightarrow excluding(object)

```
context ParceiroProgramma inv:
    quantidadeDeClientes = programaFidelidade.cliente->size()
```

Operações de iteração

```
colecao->operacao ( <expressão booleana> )
colecao->operacao ( elemento / <expressão booleana> )
colecao->operacao ( elemento : Tipo / <expressão booleana> )
```

- \rightarrow select: seleciona todos os elementos da coleção para qual a expressão booleana é verdadeira

context Cartao inv:

```
self.trasacoes->select(pontos > 100)->size() > 10
```

- \rightarrow collect: usada para obter uma nova coleção de tipos diferentes da original

context Conta inv:

```
transacoes->collect (pontos)->sum() > 20
```

- \rightarrow forAll: uma condição que deve ser verdadeira para todas as instâncias da coleção

context ProgramaFidelidade inv:

```
self.cliente->forAll( c1, c2 | c1 <> c2 implies
    c1.rg <> c2.rg)
```

- \rightarrow iterate:

```
collection->iterate( elemento : Tipo1;
    acum : Tipo2 = <expressao>
    / <expressao-com-elemento-e-acum>)

    self.servico.transacao->iterate(
        t: Transacao;
        acum: Integer = 0 /
        if t.isTypeOf(Credito) then
            acum + pontos
        else
            acum
        endif
    )
```

Tuplas

- A ordem de acesso não importa.
- Os tipos dos campos são opcionais.

```
 Tuple { nome : String = 'José', idade : Integer = 20 }.nome
       = 'José'
```

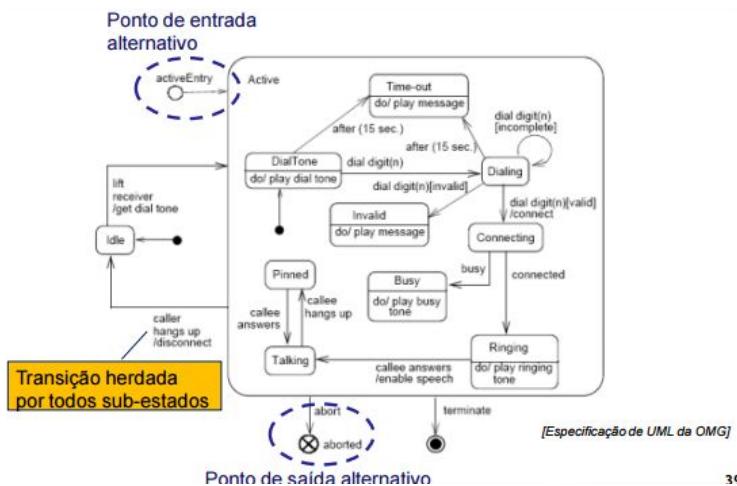
Limitações

- Restrições UML precisam ser acompanhadas de modelos UML
- Operações de quantificação e castings deixam a linguagem verborrágica
- Uma expressão que navega em duas relações e que, tecnicamente, é equivalente à uma de uma navegação é uma relação em OCL são distintas. ex.: pessoa.pais.pais é considerado diferente de pessoa.avos

Máquina de estados

- Especifica as sequências de estados pelas quais um objeto passa durante seu tempo de vida em resposta a eventos e, também, as respostas.

Máquina de estados comportamental



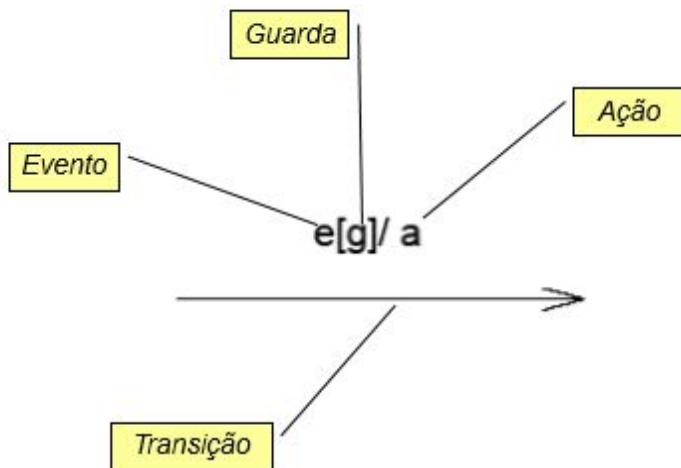
- Expressa o comportamento do objeto.

Estado

- Uma condição ou situação durante a vida do objeto.
- Composto por:
 - nome
 - ação de entrada
 - ação de saída
 - transições internas
 - sub-estados
 - eventos adiados
- Estado final possui apenas um nome.
- Podem ser simples ou compostos (quando há mais de uma região, sub-estados ou sub-máquinas).

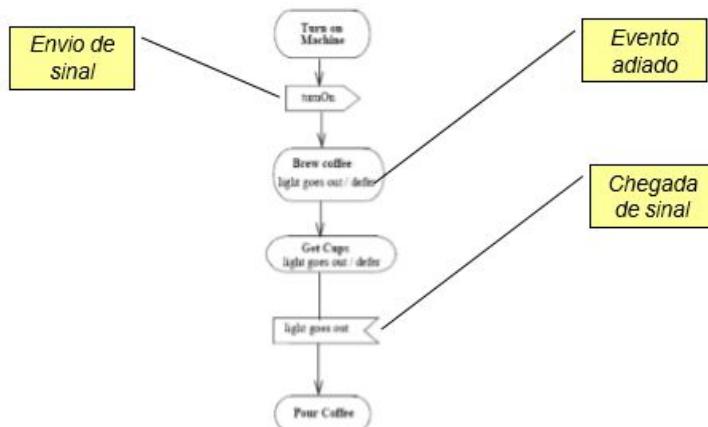
Transição

- Relacionamento que indica mudança de estados.
- Ocorre quando um evento específico ocorre ou condições são satisfeitas.
- Partes de uma transição:
 - Estado origem (Mais de um: fork)
 - Evento (opcional)
 - Guarda (opcional)
 - Ação (opcional)
 - Estado destino (Mais de um: join)



Evento

- A especificação de uma ocorrência que pode provocar transição de estado.
- Podem incluir:
 - chamadas
 - sinais
 - representa a especificação de um estímulo assíncrono comunicado entre distâncias



- passagem de tempo
 - mudança de estado
- Podem ser síncronos ou assíncronos.

Ação

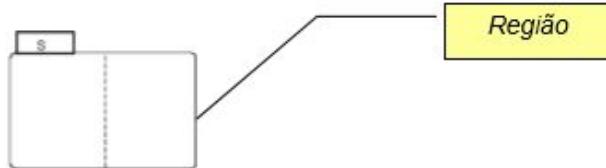
- Uma computação executável que gera um valor ou mudança de estado.
- Não pode ser interrompida por um evento.
- Internas ao estado:
 - entry
 - exit
 - do

Guarda

- Condição booleana que deve ser satisfeita ou não.
- Avaliada apenas uma vez e após o evento ocorrer

Região

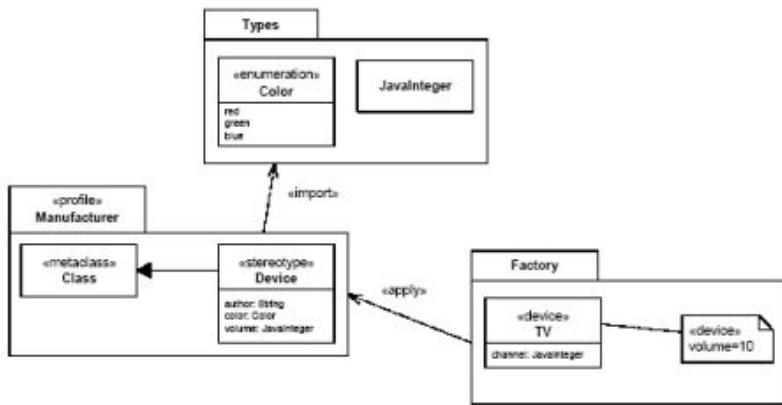
- Parte ortogonal de um estado composto ou de uma máquina de estados.
- Um estado é sequencial se tiver apenas uma região e ortogonal caso contrário.



Pseudo-estados

- Um tipo de estado que é usado para conectar múltiplas transições em um estado mais complexo.
- Tipos:
 - Junction
 - Fork
 - Join
 - Choice
 - Estado inicial
 - Nodo final
 - Histórico superficial
 - Histórico profundo
 - Ponto de entrada
 - Ponto de saída

Perfis UML



- Perfis viabilizam a adoção de UML para outros domínios
- Conjunto de técnicas e mecanismos que podem adaptar UML a um domínio específico.

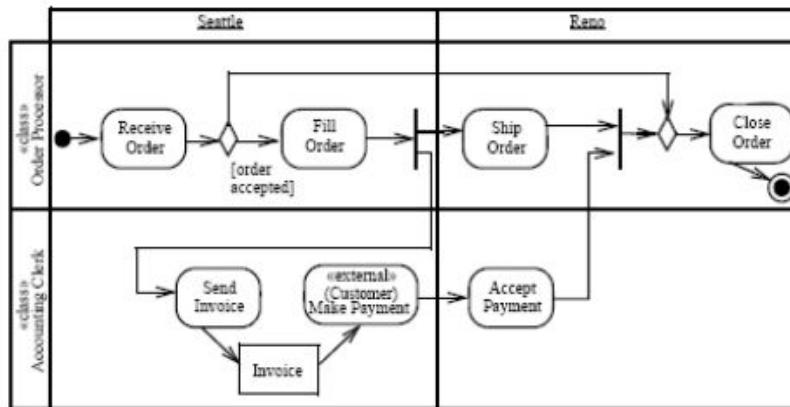
Estereótipo

- Etiqueta que pode ser adicionada a qualquer elemento de modelo UML.
- Package, class, association, interface, operation, etc.



Estágio III

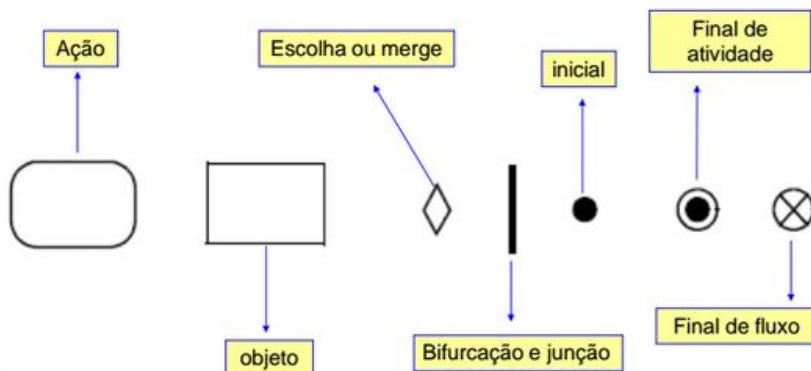
Diagrama de atividades (comportamental)



- Mostra o fluxo de controle de uma atividade para outra.
- Modela fluxo de objetos e de controle.
- Ideal para modelar algoritmos.
- Uma atividade é uma execução não atômica dentro de uma máquina de estados e são formadas por ações atômicas (que retornam um valor ou mudança de estado dos sistemas)

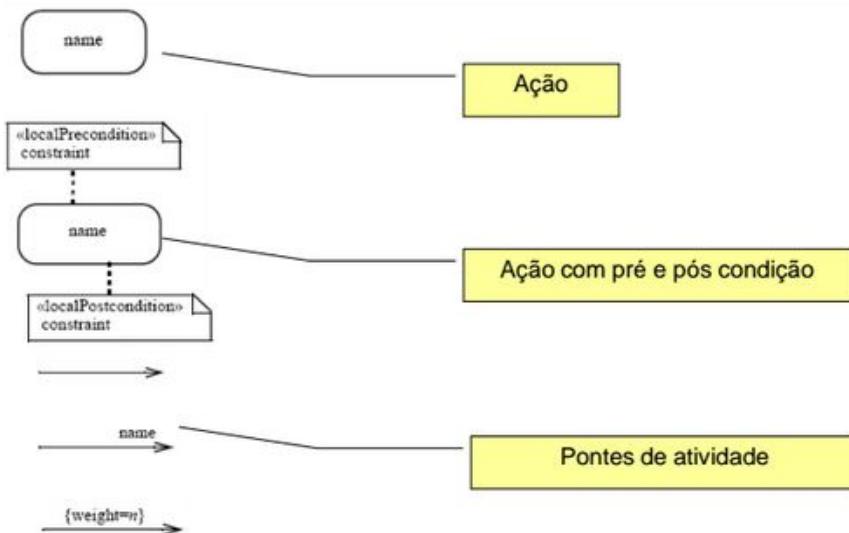
Nós

- São pontos de atividades conectados por pontes de atividades.
- Nós de ação, de objeto, de controle



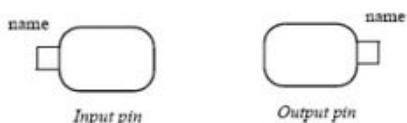
Ação

- Compõe uma atividade



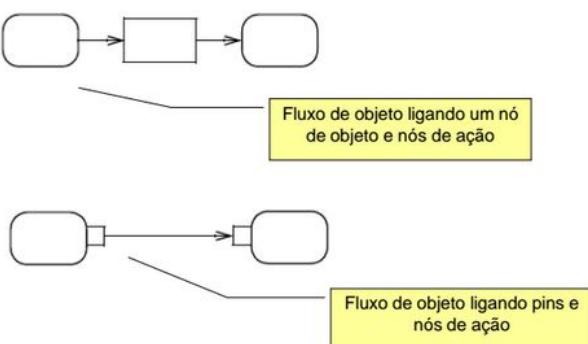
Pin

- É um nó que serve de entrada e saída de nós de ação.



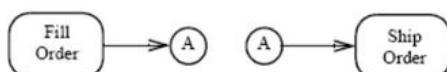
Fluxo de objetos

- Liga nós de objetos à nós de ação.



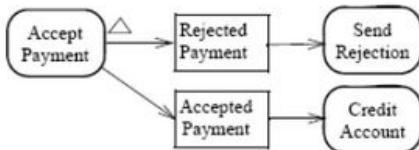
Conectores

- Conectores têm a função de simplificar a leitura do diagrama visualmente.

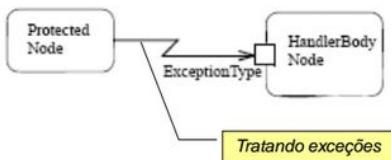


Exceções

- Ocorrência



- Tratamento



- Região interrompível

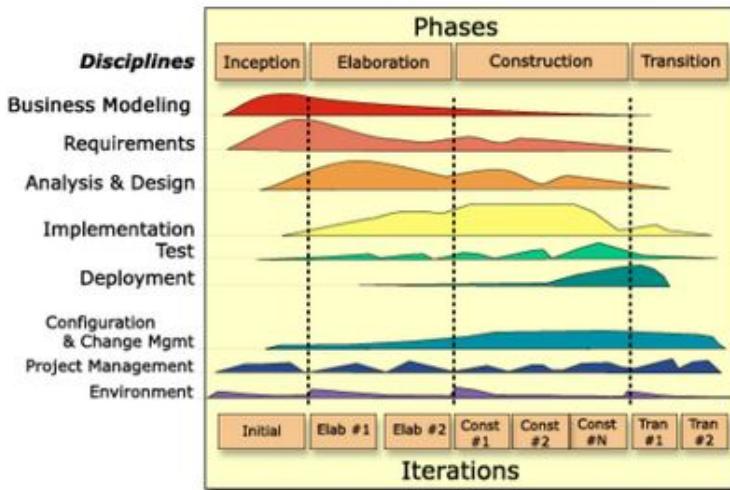


Partição de atividades - raias

- Divide um diagrama de atividades em grupos de atividades
- Raias representam entidades do mundo real
- Geralmente ocorrem com fluxos de controles concorrentes

RUP

- É um processo da engenharia de software que tem como objetivo garantir uma produção de software de alta qualidade e que atenda as expectativas do cliente dentro do prazo e custos estabelecidos.
- Definiu um conjunto de atividades, com seus responsáveis, artefatos de entrada e de saída, descrição sistemática e modelos ou templates.
- Processo iterativo e incremental
 - maior grau de reuso
 - time aprende ao decorrer das iterações
 - identificação de riscos mais cedo
 - mudanças mais fáceis de serem gerenciadas
- Guiado por casos de uso
- Baseado na arquitetura do sistema
- Arquitetura RUP:
 - eixo horizontal: tempo e ciclo de vida do processo
 - eixo vertical: atividades e workflows do processo



Workers (quem)

- Papéis desempenhados por um indivíduo ou um grupo num processo.

Atividades (como)

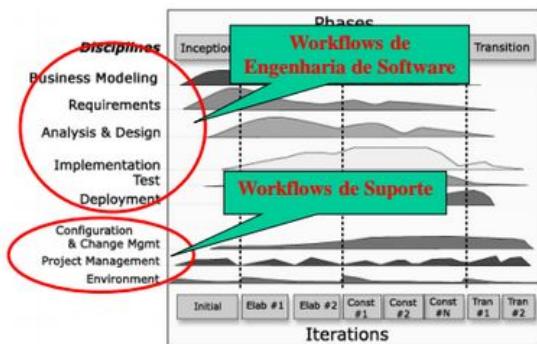
- Uma unidade de trabalho que um indivíduo naquele papel deve executar.
- Geralmente envolve um worker e produz poucos artefatos.

Artefatos (o quê)

- Um pedaço de informação produzida, modificada, utilizada por um processo.
- São usados como entradas pelos workers para executar uma atividade, assim como também são saídas de atividades.

Workflows (quando)

- Sequência de atividades que produz algum resultado de valor observável.
- Pode ser visto em diagrama:
 - de sequência
 - de comunicação
 - de atividades
 - de overview de interação
- Core workflows



Workflow de modelagem de negócios

- Entendimento a estrutura e dinâmica da organização, se certifica que os clientes, usuários e desenvolvedores possuem uma mesma visão do que será gerado e elicta os requisitos para se alcançar os objetivos.

Workflow de requisitos

- Definição dos requisitos dos sistema e de como gerenciar o escopo e possíveis mudanças de requisitos.

Workflow de análise e projeto

- Tradução os requisitos numa especificação de como implementar o sistema.

Workflow de implementação

- Desenvolvimento códigos.

Workflow de testes

- Verificação do sistema como um todo, com testes para avaliar a conformidade com os requisitos especificados.

Workflow de implantação

- Empacotamento, distribuição e instalação do produto e treinamento de usuários. Assim como planejamento de testes betas.

Ciclo de vida de processos

Concepção

- entendimento da necessidade e visão do projeto, dos requisitos
- ênfase no escopo do sistema (contexto, requisitos, restrições, etc.)
- produz uma arquitetura candidata

Elaboração

- especificação e abordagem dos pontos de maior risco
- ênfase na arquitetura
- implementação de protótipos
- entendimento dos casos de uso mais críticos do projeto
- selecionar e avaliar componentes

Construção

- desenvolvimento principal do sistema
- ênfase no desenvolvimento
- testar componentes e atestar releases do produto

Transição

- ajustes, implantação e transferência de propriedade do sistema
- ênfase na implantação
- se decide se um novo ciclo de desenvolvimento será iniciado

Processo dirigido à arquitetura

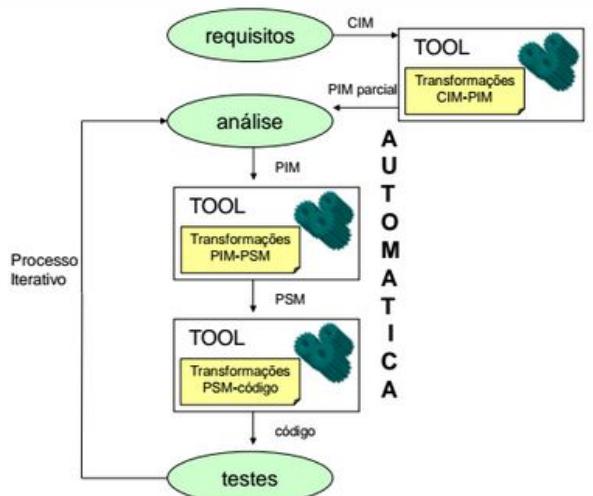
- Um só modelo não cobre todos os aspectos do sistema e muitos modelos atrapalham o entendimento geral do sistema.
- Arquitetura foca na estrutura, comportamento e contexto.
- Modelos versus Visões
- 5 visões
 - logical
 - implementation
 - process
 - deployment
 - use-case

Processo dirigido à casos de uso

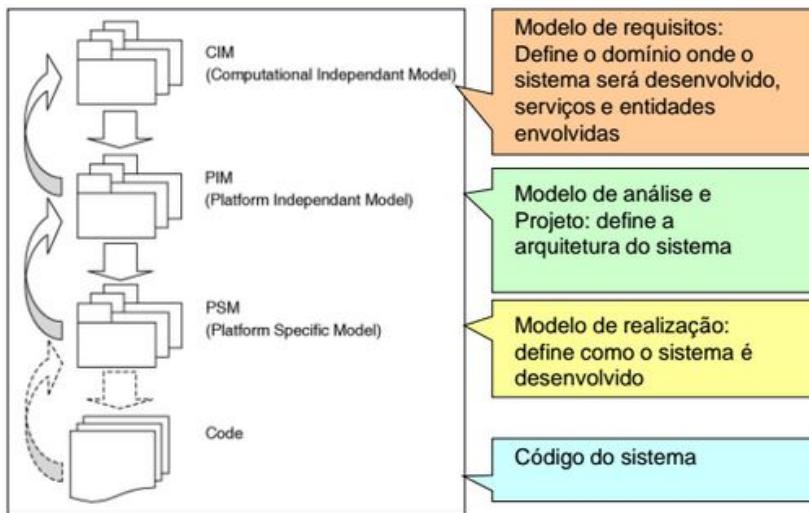
- Um caso de uso descreve um conjunto de sequências de ações cada um representando a interação dos atores e suas principais abstrações.
- Casos de uso são a base de todo o processo de desenvolvimento
- Os modelos de casos de uso é resultado do workflow de requisitos.

DDM (Desenvolvimento Dirigido por Modelos) E MDA (Model Driven Architecture)

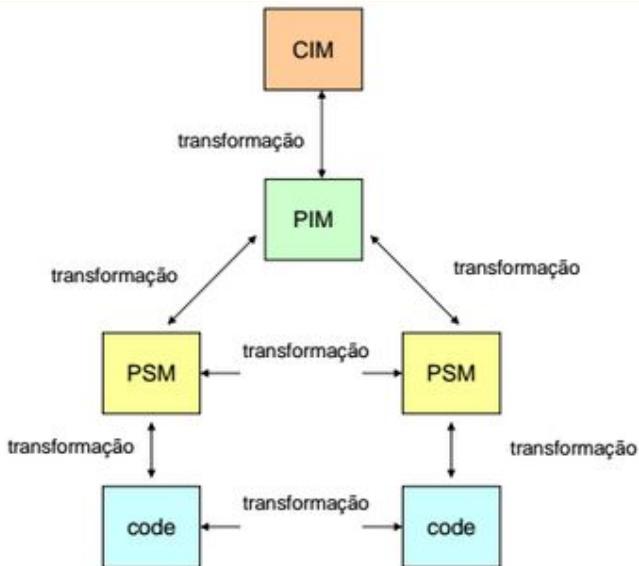
- Automatizar ciclo de vida de desenvolvimento de sistemas



Tipos de modelo



Transformação entre modelos



ATL

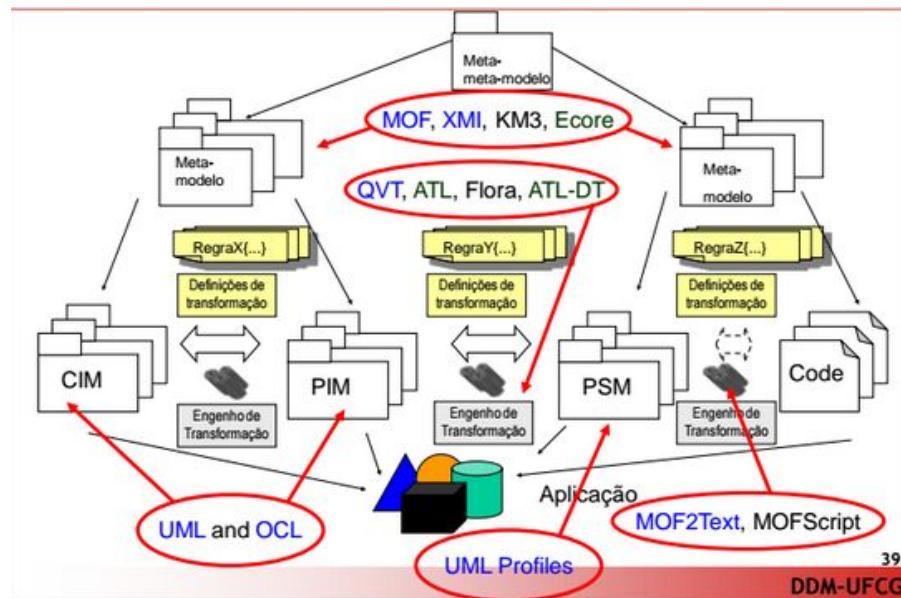
- Matched rules: transformações declarativas
- Called rules: transformações imperativas
- Linguagem amplamente baseada em OCL

Padrões OMG

- Meta-modelo
 - MOF
- Modelos
 - UML
- Linguagem para tornar MOF e UML mais precisos

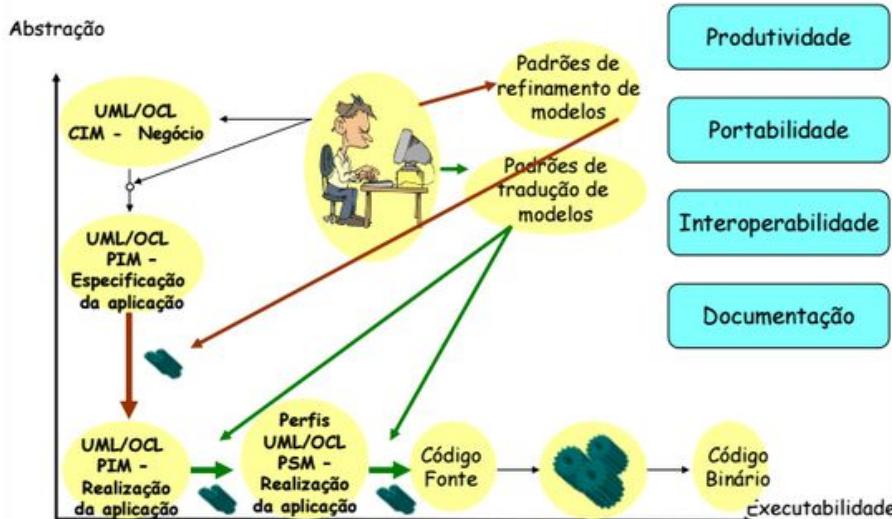
- OCL
- Linguagem para a interoperabilidade e persistência de modelos
 - XMI
- Linguagem de transformação e manipulação de modelos
 - QVT
- Linguagem de transformação textual
 - MOF2Text

Linguagens em DDM



39

DDM-UFCG



- Produtividade
 - melhor compreensão dos requisitos
- Portabilidade
 - tudo especificado no PIM é portável
- Interoperabilidade
 - transformações em diferentes PSMs
- Documentação

- PIM é a documentação em alto nível

Repositório de Modelos

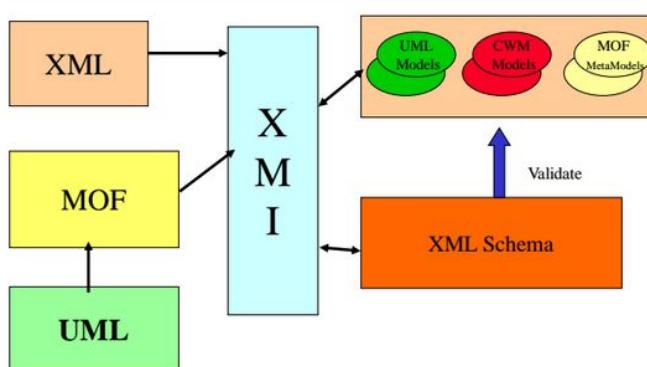
- Deve oferecer:
 - persistência
 - consulta ao modelo
 - checagem de conformidade modelos-metas
 - edições de modelos
 - controle de versões
 - controle de acesso
 - concorrência
 - tolerância à falhas
 - API

Manipulação de modelos baseados em LPOO

- JMI - Java Metadata Interface
 - baseado em MOF
 - herda interfaces reflexivas
- EMF - Eclipse Modeling Framework
 - fortemente acoplado à plataforma Eclipse
 - baseado no Ecore
- Interfaces
 - sob encomenda
 - específicas para o meta-modelo
 - reflexivas
 - gerais para qualquer tipo de meta-modelo

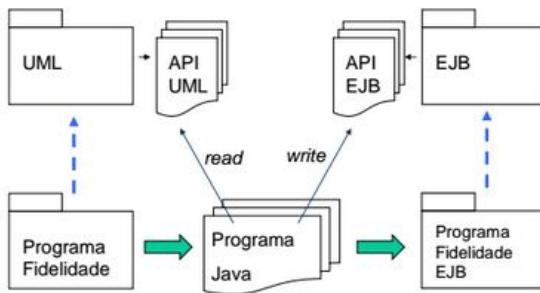
XMI

- Intercâmbio serializado de meta-dados.
- integra UML, MOF, XML

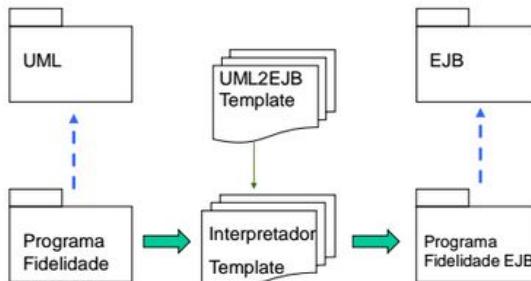


Transformações de Modelos

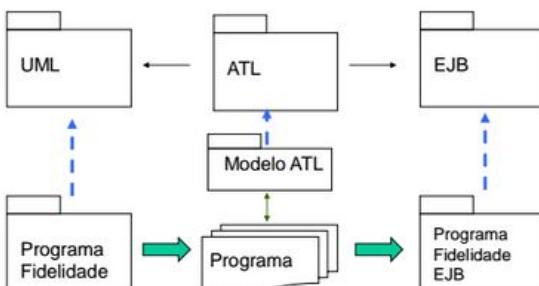
- Refinamento e refatoramento de modelos
- Engenharia reversa
- Geração de visões e de códigos
- Derivação de produtos em uma linha de produtos
- Não são transformações semânticas
- Técnicas baseada em:
 - Programação



- Templates



- Modelos



- Características:
 - linguagem de consulta
 - linguagem para a definição de transformações
 - sintaxe concreta
 - sintaxe abstrata
 - ser imperativo
 - ser declarativo
 - operar sobre modelos