UML – Unified Modeling Language

Marcelle Mussalli Cordeiro
{mmussalli@gmail.com}

Objetivo do Curso

- Fornecer ao profissional que pretende utilizar as técnicas da linguagem UML
  - Uma visão clara de modelagem.
  - Apresentar o paradigma da Orientação a Objetos.
  - Apresentar os processos de desenvolvimento.
  - Apresentar a descrição dos Diagramas utilizados para a construção de sistemas.
  - Aplicação destes conceitos em um estudo de caso.

O que é modelagem?

- Visualizar um sistema como ele é ou como se deseja que ele seja.
- Especificar o comportamento e a estrutura de um sistema.
- Guiar a construção do sistema.
- Documentar as decisões tomadas pela avaliação das alternativas propostas.

O que é modelagem?

- Captura dos processos (Use Cases).
- Melhora da comunicação entre as partes envolvidas (Especialistas e Analistas).
- Gerência da complexidade permitindo exibir várias Visões dos elementos de modelo.
- Definição da Arquitetura lógica independente das possíveis implementações.
- Permitir o reuso pela criação de componentes.

O que é modelagem?

- Servir como linguagem para comunicar decisões que não são óbvias ou que não podem ser inferidas.

- Prover uma semântica rica o suficiente para capturar o que é importante a nível tático e estratégico.

O que é modelagem?

- Auxilia:
  - Avaliação dos riscos.
  - Definição do problema.
  - Gerenciamento do projeto.

O que é UML?

- Linguagem de Modelagem Unificada.
- Surgiu em meados de 1996 com o propósito de criar uma notação padronizada para modelagem de sistemas orientados a objetos
- É o sucessor de um conjunto de métodos de análise e projeto orientados a objeto.
- Família de notações gráficas que ajuda na descrição e no projeto de sistemas de software.
- As linguagens de programação não estão num nível de abstração suficientemente alto para facilitar as discussões sobre o projeto.
- Trata-se de um padrão aberto, controlado pela OMG, um consórcio de empresas formado para estabelecer padrões que suportassem interoperabilidade, especificamente de sistemas OO.

O que é UML?

- A UML é a padronização da linguagem de desenvolvimento orientado a objetos para visualização, especificação, construção e documentação de sistemas.

- Pode ser usada com todos os tipos de processos, em todo o ciclo do desenvolvimento do software.

Histórico

- Para possibilitar o aproveitamento dos reais benefícios da OO, vários metodologistas, como Grady Booch, Ivar Jacobson, Peter Coad, James Rumbaugh e Wirfs-Brock apresentaram uma linguagem e uma seqüência de passos para a abordagem da OO, dando início a uma guerra de métodos.

- Booch
  - Caráter amplo com ênfase nos aspectos das ferramentas de análise e desenho OO, inclui modelagem de objetos, modelagem de análise, desenho da aplicação, desenho da implementação e ciclo de vida dos processos. Simbologia muito complexa para desenhar a mão.

- Rumbaugh
  - OMT (Object Modelling Technique), inicialmente para o desenvolvimento de sistemas de tempo real e sistemas específicos, como um compilador, podendo também ser aplicado para sistemas de informação.

Histórico

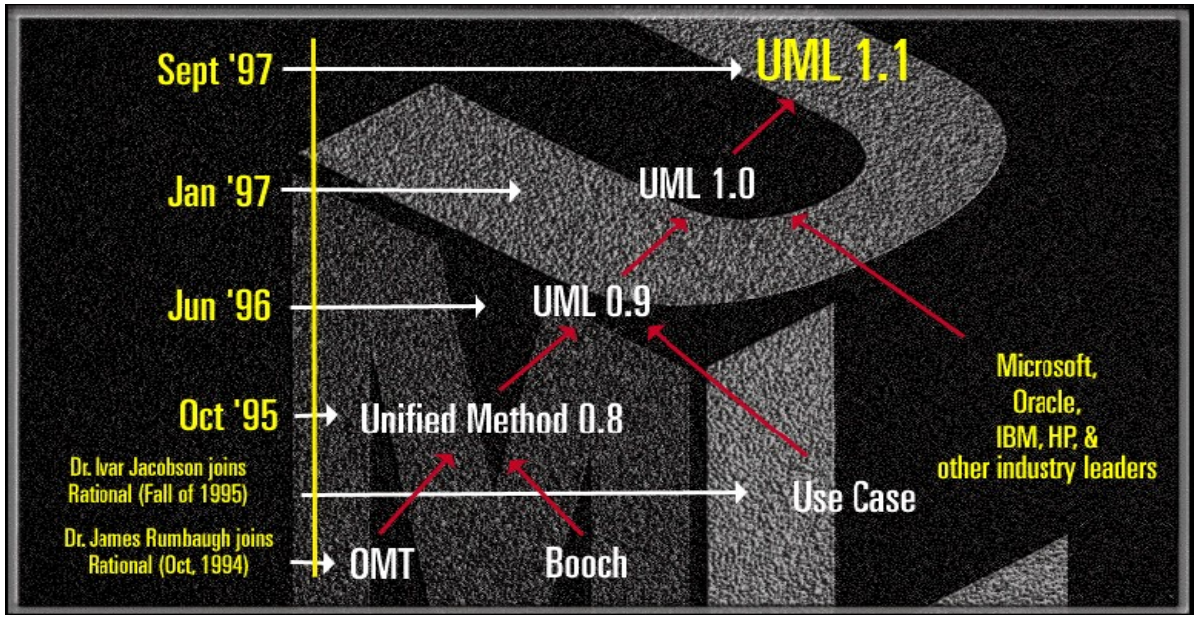
- Jacobson
  - OOSE e o Objectory. Esses dois métodos utilizam-se do conceito de use-case. O método OOSE é um método orientado a objetos, já o objectory é usado na construção de sistemas tão diversos quanto eles forem.

- Em 1995, Booch e Rumbaugh combinaram seus métodos na forma de uma notação comum e criaram o *Unified Method* (UM). Um pouco depois, a Rational compra a *Objectory* e Jacobson junta-se à equipe, integrando o use-case. Foi lançada a versão 0.8.

- Finalmente, em 1997, a UML versão 1.1 foi submetida a OMG para padronização.

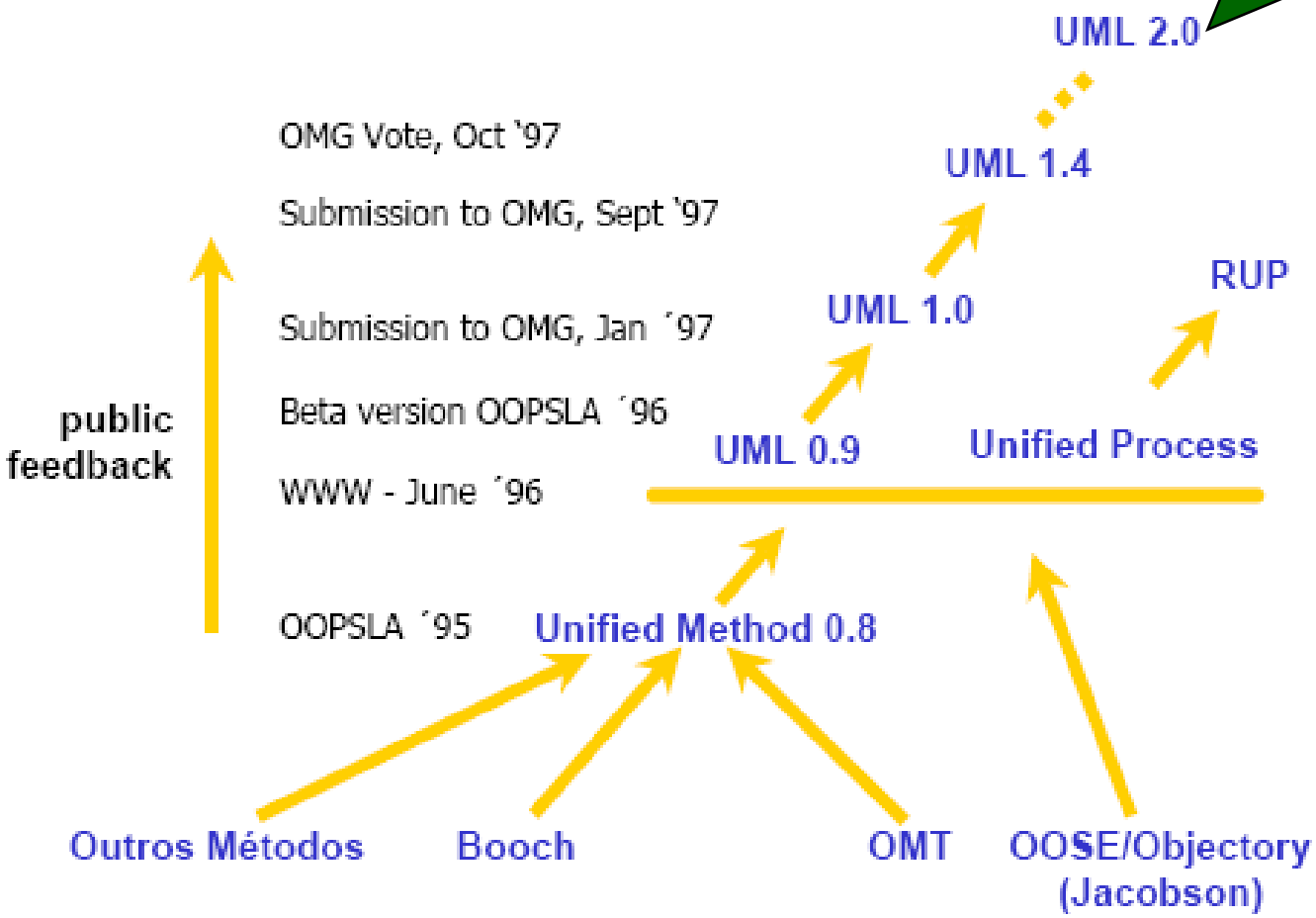
Histórico

- Desenvolvida por Grady Booch, James Rumbaugh, e Ivar Jacobson que são conhecidos como "os três amigos". A UML é a junção do que havia de melhor nestas três metodologias adicionando novos conceitos e visões da linguagem.



Histórico

De 2000 a 2003 foi produzida a UML 2.0 e a OMG a adotou como padrão no início de 2005



Utilizações da UML

- Esboço
  - Evidencia informações importantes
- Projeto
  - Abrange todo tipo de informação
- Linguagem de programação
  - Especificação de todo um sistema utilizando UML

Utilizações da UML

- ✓ Esboço
  - Evidencia informações importantes
- Projeto
  - Abrange todo tipo de informação
- Linguagem de programação
  - Especificação de todo um sistema utilizando UML

Esboço

- Essência: seletividade, “explorativo”
- Auxilia a transmissão de aspectos do sistema.
- Utilizado no desenvolvimento:
  - Utilização dos diagramas antes da codificação.
  - Propósito de expor problemas, idéias e alternativas para a equipe.
- E também na engenharia reversa:
  - Construção do diagrama a partir de um código já existente, para ajudar em seu entendimento.

Utilizações da UML

- ✓ Esboço
  - Evidencia informações importantes
- ✓ Projeto
  - Abrange todo tipo de informação
- Linguagem de programação
  - Especificação de todo um sistema utilizando UML

Projeto

- Essência: completeza, “definitivo”
- Objetivo de reduzir a programação a uma atividade simples e completamente mecânica.
- Utilizado no desenvolvimento:
  - Aplicação da diferença entre *projetista* e *programador*.
  - O projeto deve ser completo, com todas as decisões expostas para que o programador possa segui-lo de forma simples e direta.
- E também na engenharia reversa:
  - Transmissão de informações detalhadas sobre o código em documentos de papel ou via um navegador gráfico interativo.
  - Os projetos podem mostrar, de forma gráfica, cada detalhe sobre uma classe.

Utilizações da UML

- ✓ Esboço
  - Evidencia informações importantes
- ✓ Projeto
  - Abrange todo tipo de informação
- ✓ Linguagem de programação
  - Especificação de todo um sistema utilizando UML

Linguagem de programação

- À medida que a programação se torna mais mecânica, torna-se evidente a necessidade de *automatização.*
- Uso de ferramentas CASE para geração de código.
- Ponto em que todo o sistema é especificado na UML.
- Diagramas UML são compilados diretamente para código executável, e assim, a UML se torna o código fonte.
- Dessa forma, noções de desenvolvimento e engenharia reversa não se aplicam, já que UML e código fonte são a mesma coisa.

MDA – Model Driven Architecture

- Estratégia padrão para usar a UML como linguagem de programação.
- Também controlado pelo OMG.
- Utiliza a UML como linguagem de modelagem básica.
- Portabilidade, interoperabilidade e reutilização através das diferentes abstrações do sistema
- Provê uma maneira e ferramentas para:
  - Especificar um sistema independentemente da plataforma que o suporta.
  - Especificar a plataforma.
  - Escolher uma plataforma para o sistema.
  - Transformar a especificação do sistema para a especificação dependente de plataforma.

MDA – Model Driven Architecture

- CIM (Computation Independent Model)
  - Modelagem considerando apenas os requisitos, regras de negócio do sistema, sem se preocupar com a arquitetura
  - Auxilia o entendimento do problema e a comunicação entre desenvolvedores e usuário.
- PIM (Plataform Independent Model)
  - Descrição do sistema independente da plataforma.
- PSM (Plataform Specific Model)
  - Combina as especificações do PIM com detalhes que descrevem como o sistema usa um tipo particular de plataforma.
  - Um PIM pode ser utilizado para a geração de vários PSMs.

MDA – Model Driven Architecture

- Divide o trabalho de desenvolvimento em 2 áreas principais:
  - Os modeladores representam uma certa aplicação por meio da criação de um PIM (Platform Independent Model) – modelo da UML independente de qualquer tecnologia específica.
  - Uso de ferramentas para a transformação do PIM em um PSM (Platform Specific Model).
- Se a transformação for totalmente automatizada, teremos a UML como linguagem de programação.

PIM → PSM → "código final"

UML Executável

- Semelhante à MDA, começa com um modelo independente de plataforma, equivalente ao PIM.
- Utilização de um compilador (portanto, um processo totalmente automático) de modelos para transformar esse modelo UML em um sistema que possa ser distribuído em um único passo.
- Não usa o padrão UML completo, portanto, é mais simples.

Reflexão

- As ferramentas são maduras o suficiente?
- O ganho de produtividade com o uso da UML como linguagem de programação é realmente significativo em comparação com outra linguagem? É necessário também haver uma massa crítica de usuários para ser considerado de uso comum. Um bom exemplo disso é o caso do Smalltalk.
- Como modelar lógica comportamental? A UML 2 oferece três espécies de modelagem comportamental: diagramas de interação, de estados e de atividades. Qual se tornará bem sucedida?

UML Profile

- A UML é uma linguagem de modelagem utilizada largamente em vários tipos de aplicações.
- Cada domínio tem necessidades particulares, que podem ser tratadas através de extensões da UML agrupada em *profiles UML*.
- Extensão da UML adicionando uma semântica aos elementos.
- *Profiles* especializam um modelo UML para uso em um domínio particular.
- Tem como finalidade ajudar na modelagem, no desenvolvimento automático e na checagem automática de um modelo, garantindo assim a qualidade do desenvolvimento de software.
- Os mecanismos de extensão mais utilizados são os estereótipos, tagged values e restrições.

UML Profile - Exemplo

- Extensão da UML para adequação do paradigma de sistemas de tempo real.
- *Profile for Schedulability Performance and Time Specification*(SPT)
  - Desenvolvido pela OMG, com o objetivo de normalizar a utilização da UML para sistemas de tempo real.
  - Propõe um *framework* que padroniza os elementos e a forma de utilização das capacidades da UML para representar conceitos e práticas no domínio das aplicações de tempo real.
  - Modelar as capacidades que permitem analisar quantitativamente os sistemas de tempo real (escalonabilidade e desempenho).

Processo de desenvolvimento

- RUP - Rational Unified Process
- XP - eXtreme Programming

Fases de desenvolvimento

- A UML suporta as 5 disciplinas de desenvolvimento de software:
  - Análise de requisitos
  - Análise
  - Projeto
  - Implementação
  - Testes
- Estas disciplinas não são necessariamente realizadas na ordem seqüencial.

Análise de requisitos

- Esta etapa se caracteriza pela definição do comportamento do sistema, ou seja, como o sistema age ou reage, descrevendo o relacionamento entre o ambiente e o sistema.
- Captura das necessidades do usuário e não uma proposta de solução. O usuário deve indicar os requisitos prioritários para o sistema.
- Técnicas UML que são úteis nesse ponto:
  - Casos de Uso, descrevem a interação das pessoas com o sistema.
  - Diagrama de Atividades, exibe o fluxo de trabalho, interação das atividades humanas com o software e o contexto do caso de uso.

Análise de requisitos

- O mais importante nesse ponto é a comunicação com os usuários e clientes.
- Manter a notação em um mínimo para que possa ser compreendida.
- Violar regras UML é comum para que haja uma melhor comunicação.
- Desenhar diagramas que não são entendidos pelos especialistas do domínio é inútil!

Análise

- Nesta fase são identificados as classes , objetos e os mecanismos que estarão presentes no domínio do problema.

- As classes são modeladas e interligadas através de relacionamentos utilizando o Diagrama de Classe.

- Serão modeladas classes que pertençam ao domínio do problema.

- Classes que gerenciem banco de dados, comunicação, interface e outros não estarão presentes neste diagrama.

Projeto

- Resultados da análise serão expandidos em soluções técnicas.
- Novas classes serão adicionadas para prover infra-estrutura técnica, como interface do usuário e de periféricos, gerenciamento de banco de dados.
- Técnicas úteis:
  - Diagrama de Classes, perspectiva de software, mostra as classes existentes e como se relacionam.
  - Diagrama de Seqüência para cenários comuns.
    - Dica: Escolher cenários mais importantes e interessantes dos casos de uso e utilizar cartões CRC ou diagramas de seqüência para descobrir o comportamento do software.

Projeto

- Diagramas de Pacote
  - mostrar a organização em larga escala do software.
- Diagramas de Estados
  - classes com históricos de vida complexos.
- Diagramas de Distribuição
  - mostrar o layout físico do software.

Implementação

- As classes são convertidas para código real em uma linguagem OO.

Testes

- Testes de Unidade
- Testes de Integração
- Testes de Aceitação

Documentação

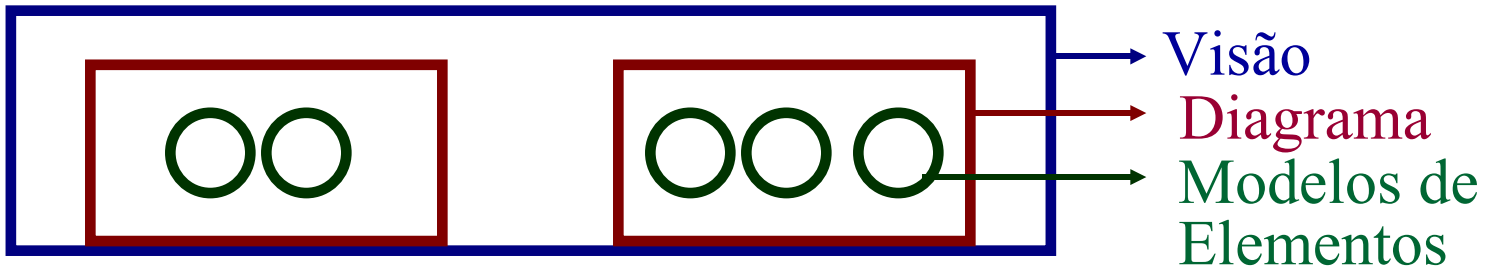
- Uso dos diagramas UML para obter um entendimento global do sistema.
- Atenção e cuidado na produção de diagramas com o objetivo de documentação.
- “Memorandos bem escritos e cuidadosamente selecionados podem facilmente substituir uma documentação de projeto abrangente tradicional. Esta última raramente brilha, exceto em pontos isolados. Eleve esses pontos... e esqueça o resto.” (Cunningham, p.384)

Notações e Metamodelos

- Notação:
  - Material gráfico encontrado nos modelos, sintaxe gráfica encontrada nas modelagens, como classe, associação e multiplicidade.
  - A maioria das linguagens gráficas de modelagem tem muito pouco rigor, não possuem definições formais de notações.
- Metamodelo:
  - Modelo do modelo.
  - Objetivo de melhorar o rigor das metodologias sem sacrificar sua utilidade.
  - Diagrama de Classe

Notação

- Partes que compõem a UML
  - Visões
  - Modelos de Elementos
  - Mecanismos Gerais
  - Diagramas



Visões

- Mostram diferentes aspectos do sistema que está sendo modelado.
- Cada visão mostrará aspectos particulares do sistema dando enfoque a ângulos e níveis de abstrações diferentes.
- Não é um gráfico, mas uma abstração consistindo em uma série de diagramas.
- Cada visão é descrita por um número de diagramas que contém informações que dão ênfase aos aspectos particulares do sistema.
- Um sistema é composto por diversos aspectos:
  - **Funcional** – Sua estrutura estática e suas interações dinâmicas.
  - **Não Funcional** – requisitos de tempo, confiabilidade, desenvolvimento
  - **Organizacionais** - organização do trabalho, mapeamento dos módulos de código, etc.

Tipos de Visões

- Visão de Use-cases
- Visão de Componentes
- Visão Lógica
- Visão de Concorrência
- Visão de Organização

Tipos de Visões

- ✓ Visão de Use-cases
- Visão de Componentes
- Visão Lógica
- Visão de Concorrência
- Visão de Organização

Visão Use-Case

- Descreve a funcionalidade do sistema desempenhada pelos atores.
- Visão central, base do desenvolvimento das outras visões do sistema.
- Essa visão é montada sobre:
  - Diagramas de Use-Case
  - Diagramas de Atividade (eventualmente)

Tipos de Visões

- ✓ Visão de Use-cases
- ✓ Visão de Componentes
- Visão Lógica
- Visão de Concorrência
- Visão de Organização

Visão de Componentes

- Descrição da implementação dos módulos e suas dependências.
- Componentes de código fonte, runtime e executável.
- É principalmente executado por desenvolvedores, e consiste nos componentes dos diagramas.
- Captura as informações do modelo físico de implementação, tais como arquivos de programas e sub-sistemas.

Tipos de Visões

- ✓ Visão de Use-cases
- ✓ Visão de Componentes
- ✓ Visão Lógica
- Visão de Concorrência
- Visão de Organização

Visão Lógica

- Descreve como a funcionalidade do sistema será implementada.
- Análise da estrutura interna do sistema pelos desenvolvedores.
- Descreve e especifica a estrutura estática do sistema (classes, objetos, e relacionamentos) e as colaborações dinâmicas quando os objetos enviarem mensagens uns para os outros para realizarem as funções do sistema.
- A estrutura estática :
  - Diagramas de Classes e Objetos.
- A modelagem dinâmica :
  - Diagrama de Seqüência, Colaboração
  - Diagramas de Estado, Atividade.

Tipos de Visões

- ✓ Visão de Use-cases
- ✓ Visão de Componentes
- ✓ Visão Lógica
- ✓ Visão de Concorrência
- ▪ Visão de Organização

Visão de Concorrência

- Trata a divisão do sistema em processos e processadores.
- Permite uma melhor utilização do ambiente onde o sistema se encontrará, se o mesmo possui execuções paralelas, e se existe dentro do sistema um gerenciamento de eventos assíncronos.
- Uma vez dividido o sistema em linhas de execução de processos concorrentes (threads), esta visão de concorrência deverá mostrar como se dá a comunicação e a concorrência destas threads.
- A visão de concorrência é suportada :
  - Diagramas Dinâmicos, que são os diagramas de estado, seqüência, colaboração e atividade
  - Diagramas de Implementação, que são os diagramas de componente e diagramas de implantação.

Tipos de Visões

- ✓ Visão de Use-cases
- ✓ Visão de Componentes
- ✓ Visão Lógica
- ✓ Visão de Concorrência
- ✓ Visão de Organização

Visão de Organização

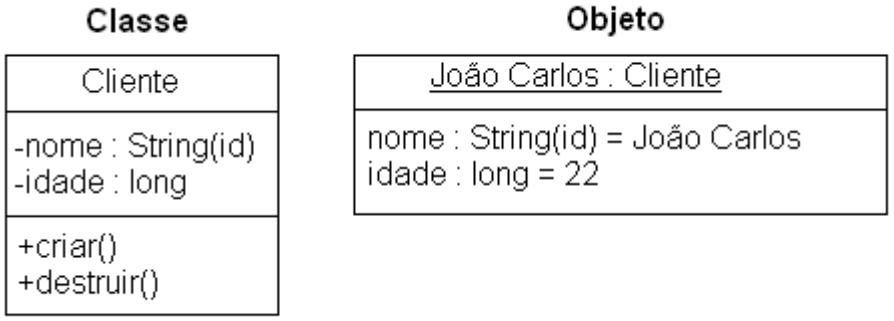
- Mostra a organização física do sistema, os computadores, os periféricos e como eles se conectam entre si.
- Esta visão será executada pelos desenvolvedores, integradores e testadores.
- Será representada pelo diagrama de implantação.
  - Mostra a arquitetura do sistema em tempo de execução composta de seus processadores, dispositivos e componentes de software.

Modelos de Elementos

- Conceitos usados nos diagramas.
- Representam definições comuns da orientação a objetos.
  - Classe
  - Objeto
  - Estado
  - Pacote
  - Componente
  - Mensagem
  - Relacionamentos – Associações, Generalizações (Herança)

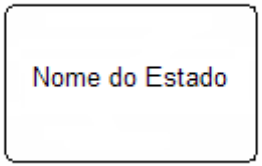
Classes e Objetos

- Classe é a descrição das propriedades e comportamentos de um objeto que identificamos no mundo real.
- Objeto é a instância da classe.
- Objeto é o elemento que podemos manipular.



Estado

- Representa o resultado de atividades executadas pelo objeto.
- Qualquer ocorrência com o objeto é chamada de *evento*.

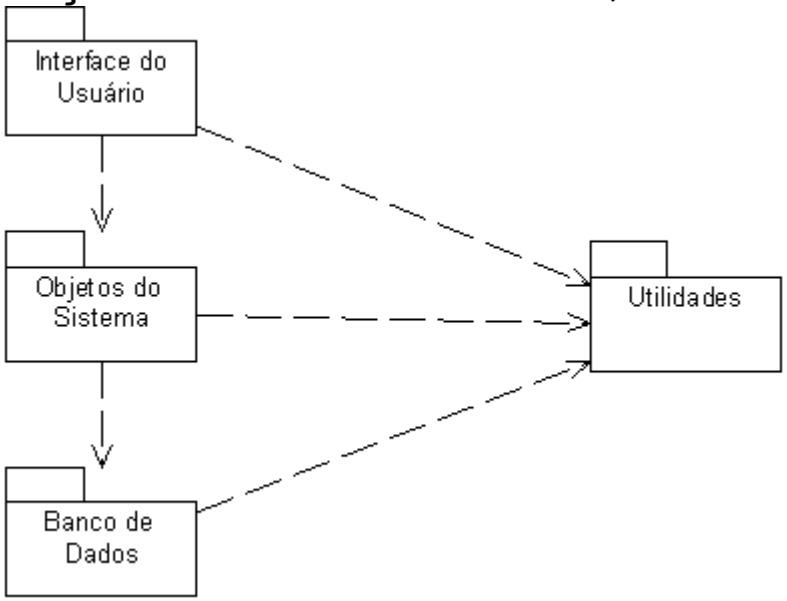


Pacote

- Mecanismo de agrupamento dos modelos de elementos.
- UML: "Um mecanismo de propósito geral para organizar elementos semanticamente relacionados em grupos."
- Um pacote possui vários modelos de elementos, e isto significa que estes não podem ser incluídos em outros pacotes.
- Pacotes podem importar modelos de elementos de outros pacotes.
- Quando um modelo de elemento é importado, refere-se apenas ao pacote que possui o elemento.

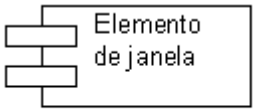
Pacote

- Possuem relacionamentos com outros pacotes.
- Relacionamentos: dependência, refinamento e generalização (herança).
- O fato de um pacote ser composto de modelos de elementos cria uma agregação de composição. Se este for destruído, todo o seu conteúdo também será.

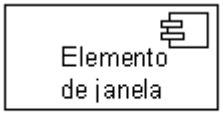


Componentes

- Nas versões anteriores da UML, os componentes eram usados para representar estruturas físicas, como as DLLs. Isso não é mais verdade, agora usa-se artefatos.
  - Pode ser tanto um código em linguagem de programação como um código executável já compilado.
  - Por exemplo, em um sistema desenvolvido em Java, cada arquivo.Java ou.Class é um componente do sistema, e será mostrado no diagrama de componentes que os utiliza.
- Representam peças que podem ser adquiridas e atualizadas independentemente.



Notação da UML 1



Notacão da UML 2

Mensagem

- São estímulos enviados aos objetos solicitando que alguma operação seja realizada por um dado objeto.
  - Nome da mensagem
  - Parâmetros
- A mensagem especifica O QUE deve ser feito.
- O comportamento de um objeto é dado pelo conjunto de mensagens que ele pode responder.

Relacionamentos

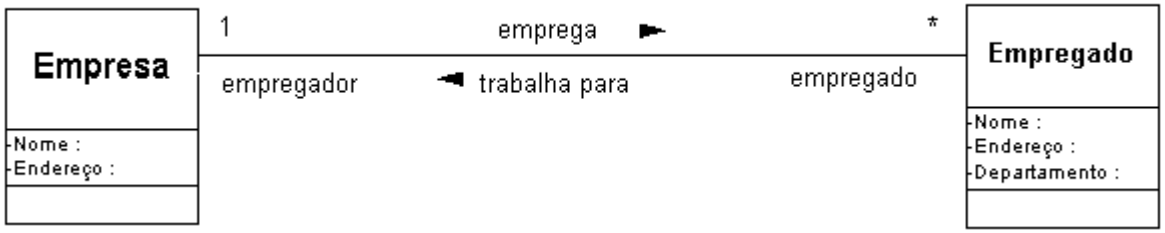
- Ligam as classes e objetos entre si criando relações lógicas entre estas entidades.
- Associação
- Generalização
- Dependência
- Refinamento
- Agregação

Relacionamentos

- ✓ Associação
- Generalização
- Dependência
- Refinamento
- Agregação

Associação

- Conexão entre classes.
- Descreve ligações, onde a ligação é definida como a semântica entre as duplas de objetos ligados.

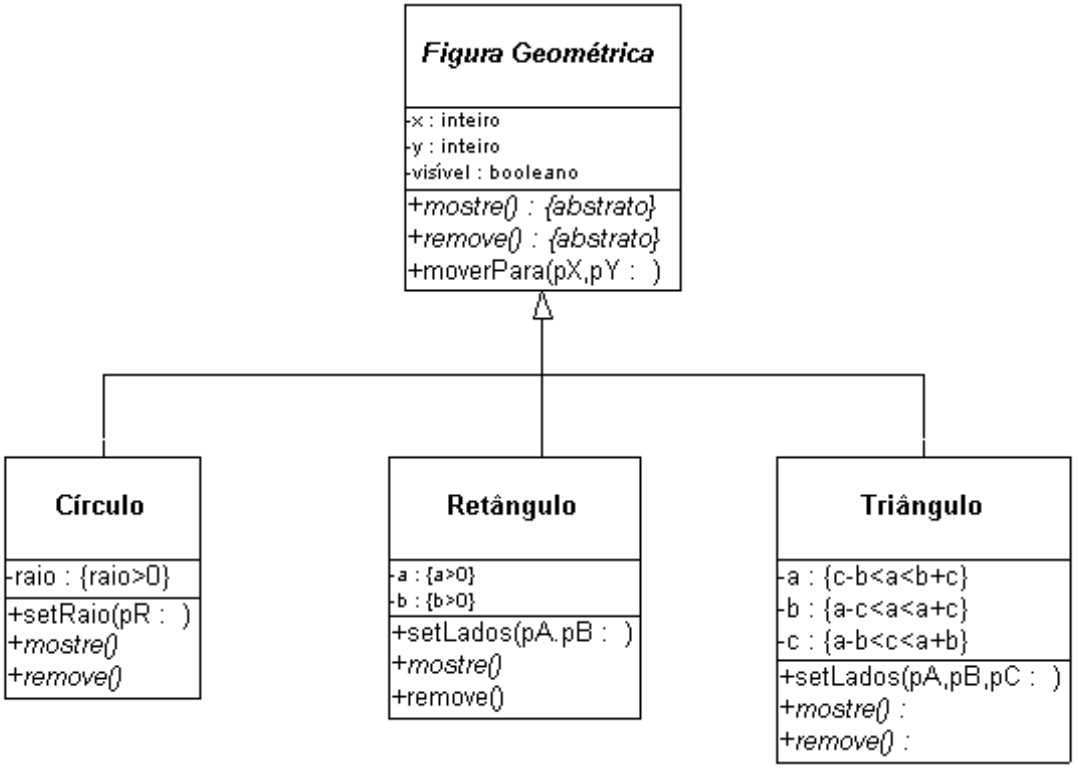


Relacionamentos

- ✓ Associação
- ✓ Generalização
- Dependência
- Refinamento
- Agregação

Generalização

- Elemento mais geral (pai) e outro mais específico (filho).
- O filho possui as características do pai mais informações adicionais.
- Classes semelhantes.
- Herança.

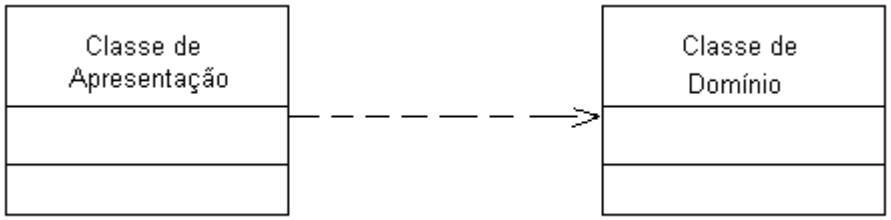


Relacionamentos

- ✓ Associação
- ✓ Generalização
- ✓ Dependência
- Refinamento
- Agregação

Dependência

- Elemento independente e outro dependente.
- Uma subclasse é dependente de uma superclasse.
- Significa que modificações na superclasse acarretam em modificações na subclasse e não o contrário.

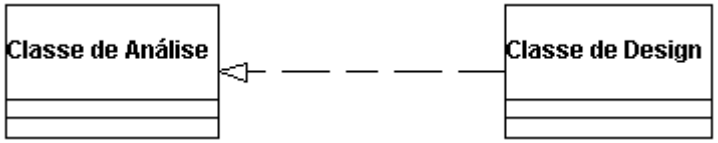


Relacionamentos

- ✓ Associação
- ✓ Generalização
- ✓ Dependência
- ✓ Refinamento
- ▪ Agregação

Refinamento

- Duas descrições de uma mesma entidade, mas em níveis de abstração diferentes.
- Uma relação que representa a especificação mais completa de algo que já foi especificado em um certo nível de detalhe.

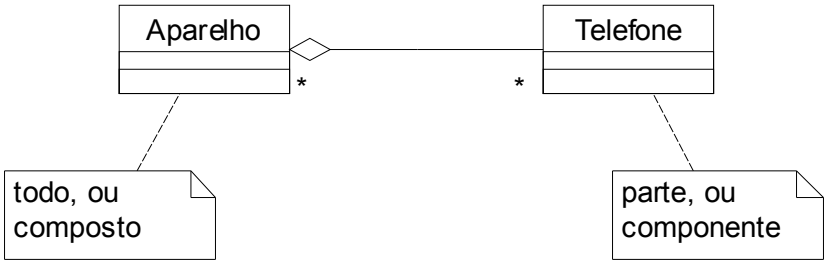


Relacionamentos

- ✓ Associação
- ✓ Generalização
- ✓ Dependência
- ✓ Refinamento
- ✓ Agregação

Agregação

- Forma especial de associação em que as classes envolvidas representam uma hierarquia todo-partes.



Mecanismos Gerais

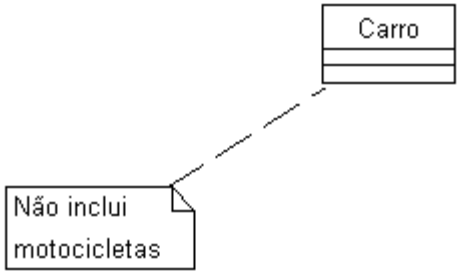
- Provém comentários suplementares, informações, ou semântica sobre os elementos que compõem os modelos.
- Ornamentos
- Notas

Ornamentos Gráficos

- São anexados aos modelos de elementos em diagramas e adicionam semânticas ao elemento.
- Um exemplo de um ornamento é o da técnica de separar um tipo de uma instância.
  - Quando um elemento representa um tipo, seu nome é mostrado em negrito.
  - Quando o mesmo elemento representa a instância de um tipo, seu nome é escrito sublinhado e pode significar tanto o nome da instância quanto o nome do tipo.
- Especificação de multiplicidade de relacionamentos, onde a multiplicidade é um número que indica quantas instâncias de um tipo conectado pode estar envolvido na relação.

Notas

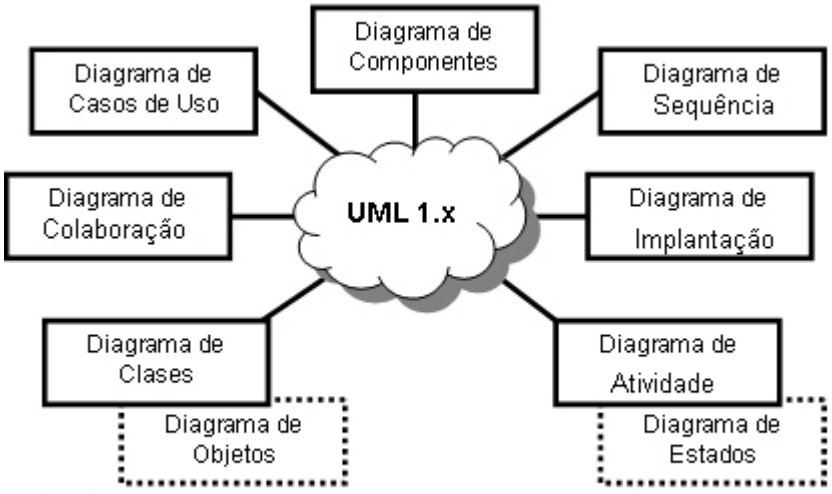
- São comentários nos diagramas.
- Podem ser isoladas ou vinculadas, com uma linha tracejada, aos elementos que estão sendo comentados.
- Aparecem em qualquer tipo de diagrama.



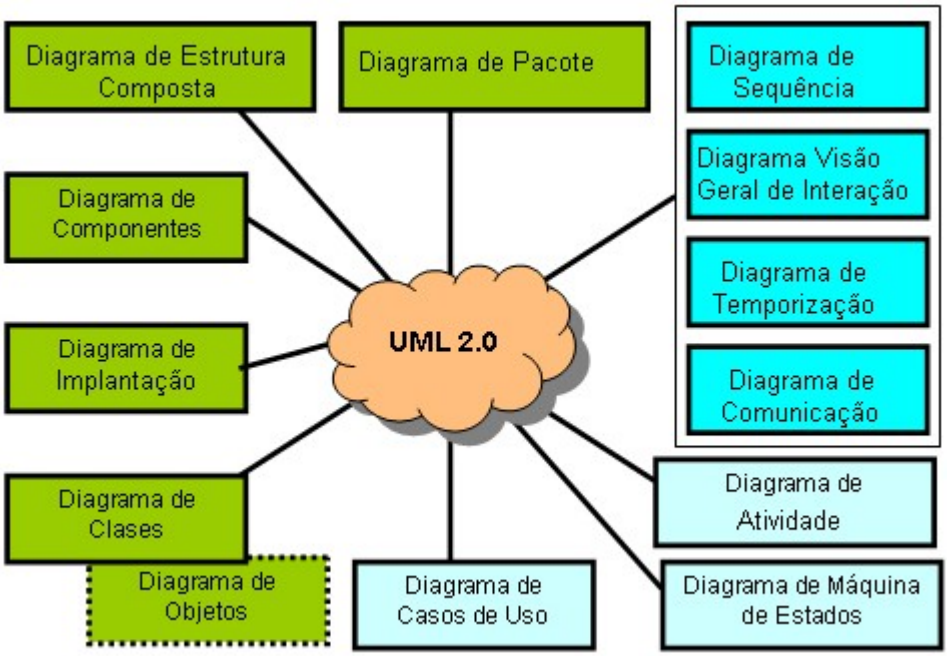
Diagramas

- O modo para descrever os vários aspectos de modelagem para a UML é através da notação definida pelos seus vários tipos de diagramas.
- Um diagrama é uma apresentação gráfica de uma coleção de modelos de elementos, frequentemente mostrado como um gráfico conectado de arcos (relacionamentos) e vértices (outros elementos do modelo).

Diagramas UML 1.x



Diagramas UML 2.0



Mudanças da UML 1.x para 2.0

- A UML 2 representa a maior mudança que aconteceu na UML.
- Introdução de novos tipos de diagramas.
- O diagramas de Objetos e de Pacotes já eram desenhados, mas não eram tipos de diagramas oficiais.
- Modificação do nome dos diagramas de Colaboração para Comunicação.
- Foram adicionados novos componentes que permitem a representação de sistemas de tempo real. Diagrama de temporização, expandido o universo da classificação das ações e adicionado componentes nos diagramas já existentes para a representação do tempo.

Ferramentas CASE

- Rational Rose
- ARGOUML

O Futuro e a UML

- A UML já é a base para muitas ferramentas de desenvolvimento, incluindo modelagem visual, simulações e ambientes de desenvolvimento.
- A integração que a UML trouxe vai acelerar o uso do desenvolvimento de softwares orientados a objetos.
- Grande aumento no desenvolvimento de Sistemas OO.
- Softwares Complexos tornam-se mais simples com uma linguagem de modelagem visual.
- Modelagem visual robusta para todas as fases do desenvolvimento do software.
- Mais facilidade na comunicação entre desenvolvedores.
- Ferramentas CASE mais poderosas facilitando cada vez mais a programação.

UML – Unified Modeling Language

Marcelle Mussalli Cordeiro
{mmussalli@gmail.com}