Modelando com UML – Unified Modeling Language

AHMED ALI ABDALLA ESMIN[1]

[1]ILES – ULBRA – Instituto de Informática - Universidade Luterana do Brasil
Curso de Informática
Cx. Postal 271 – CEP 78.9860-000 Ji-Paraná (RO)
ahmed@ulbrajp.com.br

Resumo: Este artigo apresentada uma introdução sobre UML "Unified Modeling Language" que é a padronização das metodologias de desenvolvimento de sistemas baseados na orientação a objetos. A UML incorpora as noções do desenvolvimento de software totalmente visual.

Palavras-Chaves Programação Orientada a Objetos, Analise e Projeto Orientado ao Objetos, UML.artigo.

1 Introdução

Os conceitos da orientação a objetos são bem difundidos, desde o lançamento da 1ª linguagem orientada a objetos, a SIMULA. O desenvolvimento de sistemas de software de grande porte são suportados por métodos de análise e projeto que modelam esse sistema de modo a fornecer para toda a equipe envolvida uma compreensão única do projeto.

A UML (Unified Modeling Language) é o sucessor de um conjunto de métodos de análise e projeto orientados a objeto. A UML está, atualmente, em processo de padronização pela OMG (Object Management Group).

A UML é um modelo de linguagem, não um método. Um método pressupõe um modelo de linguagem e um processo. O modelo de linguagem é a notação que o método usa para descrever o projeto. O processo são os passos que devem ser seguidos para se construir o projeto.

A UML define uma notação e um meta-modelo. A notação representa a sintaxe da linguagem composta por elementos gráficos. Um meta-modelo é um diagrama de classe.

UML foi 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 adicionado novos conceitos e visões da linguagem (Figura1). Neste trabalho, será apresentada a notação UML através de algumas definições.

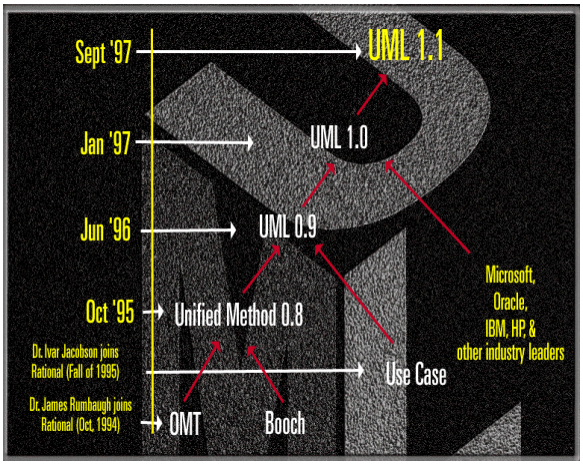


Figura 1: A História da UML

2 Desenvolvimento de um Sistema em UML

A UML suporta os cinco fases de desenvolvimento de Software: análise de requisitos, análise, projeto, implementação e testes. Estas fases não necessariamente devem ser executadas na ordem Seqüencial.

2.1 Análise de Requisitos

Nesta fase são capturados as necessidades dos usuário e o comportamento do sistema através de análise de

casos de uso chamados “Use Case”. As entidades externas ao sistema (em UML chamados de "atores externos") que interagem como sistema são modelados entre as funções que eles requerem, funções estas chamadas de "use-cases". Os atores externos e os "use-cases" são modelados com entre eles. O diagrama de "use-cases" É um diagrama usado para se identificar como o sistema se comporta em várias situações que podem ocorrer durante sua operação

2.2 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. Na análise, só serão modeladas classes que pertençam ao domínio do problema, classes que gerenciem banco de dados, comunicação, interface e outros não estarão presentes neste diagrama.

2.3 Projeto

Os resultado da análise é expandido nesta fase em soluções técnicas. Novas classes serão adicionadas para prover uma infra-estrutura técnica: a interface do usuário e de periféricos, gerenciamento de banco de dados, comunicação com outros sistemas, entre outros. As classes do domínio do problema modeladas na fase de análise são mescladas nessa nova infra-estrutura tornando possível alterar tanto o domínio do problema quanto a infra-estrutura. O design resulta no detalhamento das especificações para a fase seguinte.

2.4 Implementação

As classes são convertidas para código real em uma linguagem OO (procedural não é recomendado). Dependendo da capacidade da linguagem usada, essa conversão pode ser uma tarefa fácil ou não.

2.5 Testes

Idêntico a qualquer outro método de modelagem. Dividida em testes de unidades, testes de integração, teste de sistema e testes de aceitação.

3 A Notação da Linguagem

Para cada uma das fases do desenvolvimento utilizam-se em seu desenvolvimento cinco tipos de visões, nove tipos de diagramas e vários modelos de elementos que serão utilizados na criação dos diagramas e mecanismos gerais que todos em conjunto especificam e exemplificam a definição do sistema, tanto a definição no que diz respeito à funcionalidade estática e dinâmica do desenvolvimento de um sistema.

Antes de abordarmos cada um destes componentes separadamente, definiremos as partes que compõem a UML:

- Visões: As Visões mostram diferentes aspectos do sistema que está sendo modelado. A visão não é um gráfico, mas uma abstração consistindo em uma série de diagramas. Modelos de Elementos: Os conceitos usados nos diagramas são modelos de elementos que representam definições comuns da orientação a objetos como as classes, objetos, mensagem, relacionamentos entre classes incluindo associações, dependências e heranças.
- Mecanismos Gerais: Os mecanismos gerais provém comentários suplementares, informações, ou semântica sobre os elementos que compõem os modelos.
- Diagramas: Os diagramas são os gráficos que descrevem o conteúdo em uma visão. UML possui nove tipo de diagramas que são usados em combinação para prover todas as visões do sistema.

3.1 Visões

Para desenvolver um sistema complexo envolve vários aspectos, tais como: o aspecto funcional, não funcional e aspectos organizacionais para isto utiliza se de dos visões

Cada visão é descrita por um número de diagramas que contém informações que dão ênfase aos aspectos particulares do sistema. Os diagramas que compõem as visões contém os modelos de elementos do sistema. As visões que compõem um sistema são:

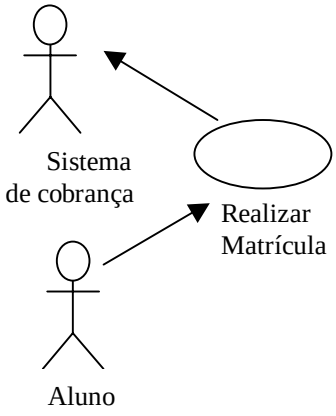
- Visão “use-case”: Descreve a funcionalidade do sistema desempenhada pelos atores externos do sistema (usuários).
- Visão Lógica: Descreve como a funcionalidade do sistema será implementada. É feita principalmente pelos analistas e desenvolvedores. Em contraste com a visão use-case, a visão lógica observa e estuda o sistema internamente. Ela 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.
- Visão de Componentes: É uma descrição da implementação dos módulos e suas dependências.

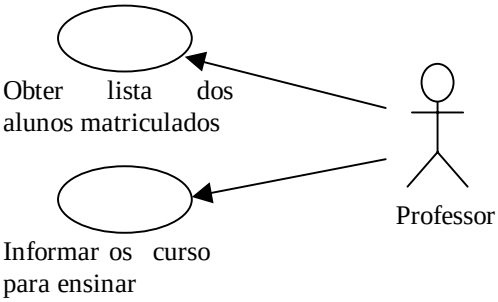
É principalmente executado por desenvolvedores, e consiste nos componentes dos diagramas.

- Visão de concorrência: Trata a divisão do sistema em processos e processadores. Este aspecto, que é uma propriedade não funcional do sistema, 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.
- Visão de Organização: Finalmente, a visão de organização mostra a organização física do sistema, os computadores, os periféricos e como eles se conectam entre si.

3.2 Comportamento do Sistema

O comportamento do sistema é capturado através de análise de casos de uso do sistema chamados "Use Case". Exemplo de um sistema matricula num curso





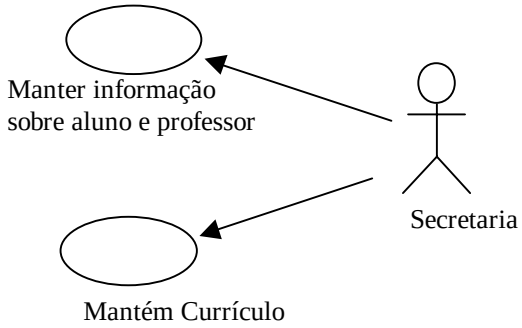


Figura 2: Exemplo de Diagrama de Use Case

O diagrama "Use Case" é um diagrama usado para se identificar como o sistema se comporta em várias situações que podem ocorrer durante sua operação. Os componentes deste diagrama são os atores e os "Use Case" (Figura3).

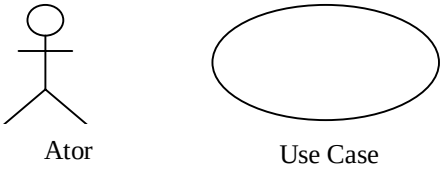


Figura 3: A notação usada pelo Diagrama de "Use Case"

- Ator: Representa qualquer entidade que interage com o sistema. Pode ser uma pessoa, outro sistema, etc.
- Use Case: é uma seqüência de ações que o sistema executa e produz um resultado de valor para o ator.

3.3 Objetos e Classes de Objetos

3.3.1 Objeto

O objeto Representa um entidade que pode ser física, ou de software. O objeto é descritos através do seguinte: Estado, Comportamento e Identidade

- A Identidade é identifica unicamente um objeto, mesmo que ele possua estados ou comportamento comuns a outros objetos.
- Estado de um objeto é mutável ao longo do tempo. É implementado por um conjunto de atributos, os valores desses atributos e ligações que o objeto pode ter com outros objetos.
- Comportamento de um objeto é modelado através de um conjunto de mensagens das operações executadas internamente pelo objeto.

3.3.2 Classe

É uma descrição de um grupo de objetos com atributos, comportamentos, relacionamentos com outros objetos e semântica comuns. Uma classe é uma abstração que enfatiza características relevantes dos objetos, suprimindo outras características. Portanto um objeto é sempre uma instância de uma classe.

A classe de objeto (Figura 3) é representada por um retângulo, subdividido em três áreas: A primeira contém o nome da Classe. A segunda contém seus atributos. A terceira contém suas operações.

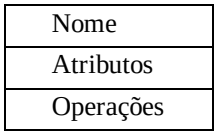


Figura4 : A notação de Classe

3.3.3 Estereótipos

Estereótipo é um elemento de modelagem que rotula tipos de Classes de Objeto. Uma Classe de Objetos pode ter um ou mais tipos de estereótipos. Os estereótipos mais comuns são:

- Classe Fronteiriça
- Classe de Entidade
- Classe de Controle
- Classe de Exceção
- Metaclasse
- Classe Utilitária

A notação usada pela UML para Estereótipos, é coloca-lo entre << >> na área reservada para o nome da classe .

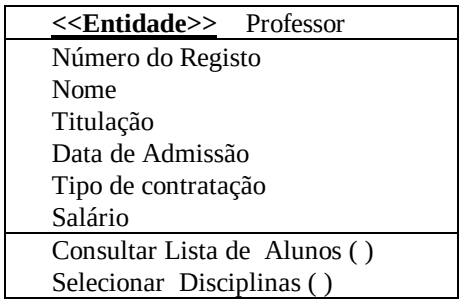


Figura5 : Exemplo de Estereótipos

Dentro de Estereótipos tem:

3.3.3.1 Classe fronteiriça

É uma classe que modela a comunicação entre sistemas e suas operações internas. Exemplos: Protocolo de Comunicação, Interface e Impressão, etc.

3.3.3.2 Classe de Entidade

É uma classe que modela objetos persistentes.

3.3.3.3 Classe de Controle

É uma classe que modela o comportamento de controle para uma ou mais "Use Case". Suas principais características são: Cria, ativa e anula objetos controlados, Controla a operação de objetos controlados, Controla a concorrência de pedidos de objetos controlados .

3.4 Interação entre objetos

A UML se utiliza de dois diagramas para representar a interação entre os objetos: Diagrama de Seqüência e Diagrama de Colaboração.

3.4.1 Diagrama de Seqüência

Mostra a interação entre os Objetos ao longo do tempo. Exemplo da matricula (Figura 5):

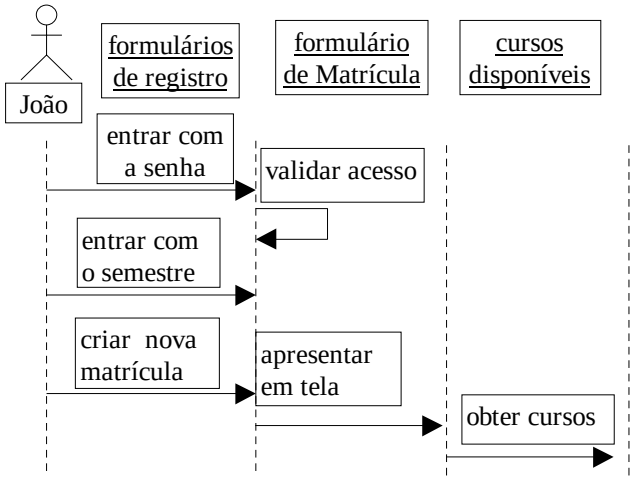


Figura6 : Diagrama de Seqüência: Exemplo realizar Matricula .

3.4.2 Diagrama de Colaboração

O Diagrama de Colaboração mostra a interação organizada entorno dos Objetos e suas ligações uns com os outros. (Figura 6)

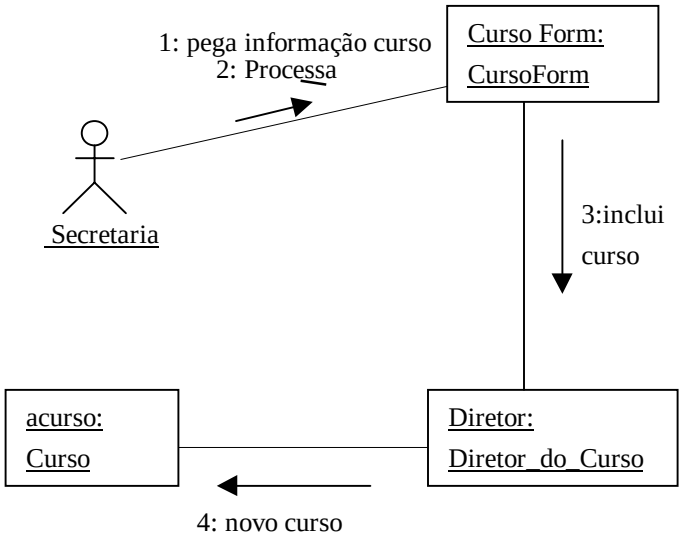


Figura7: Digrama de colaboração para manter as informações dos cursos.

3.4.3 Pacotes

Um pacote é um mecanismo de propósito geral para organizar elementos semanticamente relacionados em grupos. Todos os modelos de elementos que são ligados ou referenciados por um pacote são chamados de "Conteúdo do pacote".

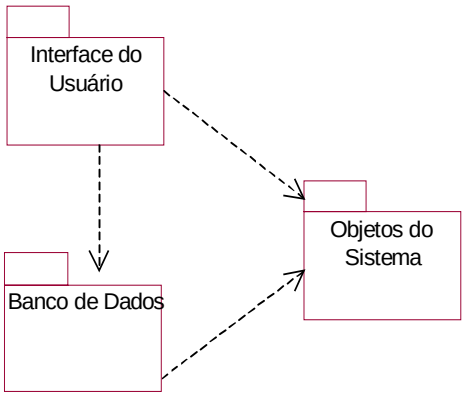


Figura8: Exemplo Pacotes com relacionamentos

Os relacionamentos permitidos entre pacotes são de dependência, refinamento e (herança.

3.4.4 Componentes

Diagramas de Componentes ilustram a organização e dependências entre componentes de software.

Um componente pode ser :

- Um componente de código fonte
- Um componente de run-time, ou
- Um componente executável

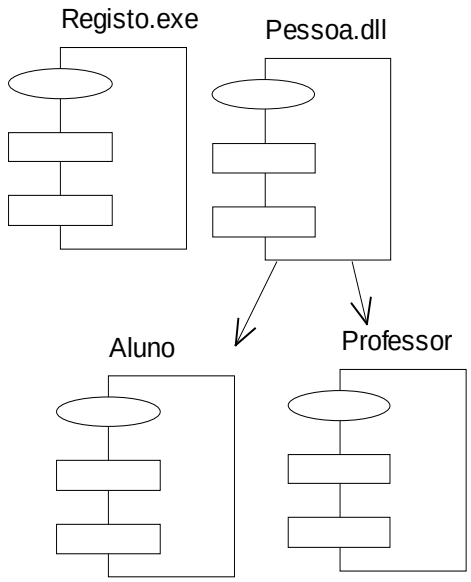


Figura9 : Exemplo de Diagrama de componentes

3.4.5 Relacionamentos

Os relacionamentos ligam as classes/objetos entre si criando relações lógicas entre estas entidades. Os relacionamentos podem ser dos seguintes tipos:

- Associação: É uma conexão entre classes, e Em UML, uma associação é definida com um relacionamento que descreve uma série de ligações.
- Generalização: É um relacionamento de um elemento mais geral e outro mais específico. O elemento mais específico pode conter apenas informações adicionais.
- Dependência e Refinamentos: Dependência é um relacionamento entre elementos, um independente e outro dependente.

3.4.5.1 Associações Normais

O tipo mais comum de associação é apenas uma conexão entre classes. É representada por uma linha sólida entre duas classes. A associação possui um nome (junto à linha que representa a associação), normalmente um verbo, mas substantivos também são permitidos.

Para expressar a multiplicidade entre os relacionamentos, um intervalo indica quantos objetos estão relacionados no link. O intervalo pode ser de zero para um (0..1), zero para vários (0..* ou apenas *), um para vários (1..*), dois (2), cinco para 11 (5..11) e assim por diante. É também possível expressar uma série de números como (1, 4, 6..12). Se não for descrito nenhuma multiplicidade, então é considerado o padrão de um para um (1..1 ou apenas 1).

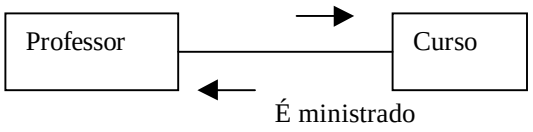


Figura 10: Exemplo de relacionamento

No exemplo acima vemos um relacionamento entre as classes Professo e Curso se relacionam por associação.

3.4.5.2 Associação Recursiva

É possível conectar uma classe a ela mesma através de uma associação e que ainda representa semanticamente a conexão entre dois objetos, mas os objetos conectados são da mesma classe. Uma associação deste tipo é chamada de associação recursiva.

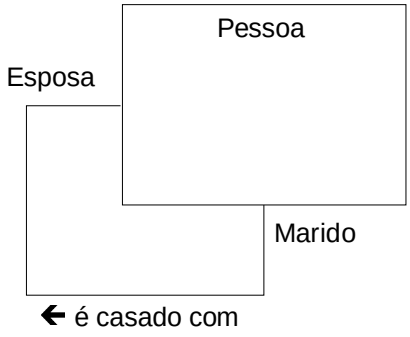


Figura 11: Exemplo de uma associação recursiva

3.4.5.3 Associação Qualificada

Associações qualificadas são usadas com associações de um para vários (1..*) ou vários para vários (*). O "qualificador" (identificador da associação qualificada) especifica como um determinado objeto no final da associação "n" é identificado, e pode ser visto como um tipo de chave para separar todos os objetos na associação. O identificador é desenhado como uma pequena caixa no final da associação junto à classe de onde a navegação deve ser feita.

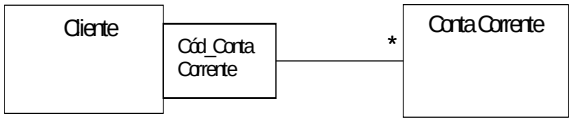


Figura 12: Representação de componentes

3.4.5.4 Associação Exclusiva

Em alguns modelos nem todas as combinações são válidas, e isto pode causar problemas que devem ser tratados. Uma associação exclusiva é uma restrição em duas ou mais associações. Ela especifica que objetos de uma classe podem participar de no máximo uma das associações em um dado momento. Uma associação exclusiva é representada por uma linha tracejada entre as associações que são parte da associação exclusiva, com a especificação "{ou}" sobre a linha tracejada.

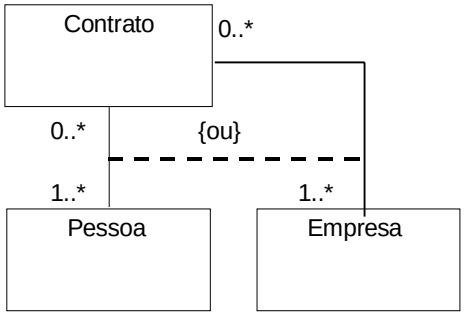


Figura 13: – Exemplo de uma associação exclusiva

3.4.5.5 Associação Ternária

Mais de duas classes podem ser associadas entre si, a associação ternária associa três classes. Ela é mostrada como um grade losango (diamante) e ainda suporta uma associação de classe ligada a ela, traçaria-

se, então, uma linha tracejada a partir do losango para a classe onde seria feita a associação ternária.

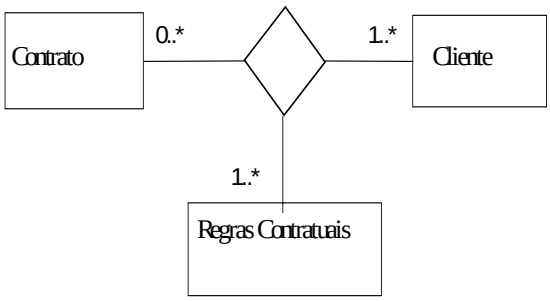


Figura 14 : Exemplo de uma associação ternária

No exemplo acima a associação ternária especifica que um cliente poderá possuir 1 ou mais contratos e cada contrato será composto de 1 ou várias regras contratuais.

3.4.6 Agregação

A agregação é um caso particular da associação. A agregação indica que uma das classes do relacionamento é uma parte, ou está contida em outra classe. As palavras chaves usadas para identificar uma agregação são: “consiste em”, “contém”, “é parte de”.

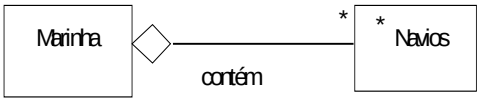


Figura 15: – Exemplo de uma agregação entre duas classes

Existem tipos especiais de agregação que são as agregações compartilhadas e as compostas.

Agregação Compartilhada: É dita compartilhada quando uma das classes é uma parte, ou está contida na outra, mas esta parte pode fazer estar contida na outra várias vezes em um mesmo momento.

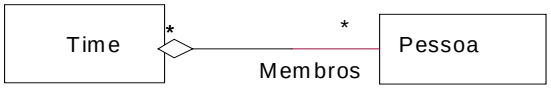


Figura 16: – Exemplo de uma agregação.

No exemplo acima uma pessoa pode ser membro de um time ou vários times e em determinado momento.

Agregação de Composição: É uma agregação onde uma classe que está contida na outra “vive” e constitui a outra. Se o objeto da classe que contém for destruído, as classes da agregação de composição serão destruídas juntamente já que as mesmas fazem parte da outra.

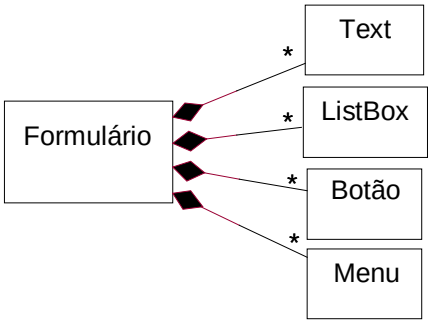


Figura 17 – Exemplo de uma agregação de composição

3.4.7 Generalização

A generalização é um relacionamento entre um elemento geral e um outro mais específico. O elemento mais específico possui todas as características do elemento geral e contém ainda mais particularidades. Um objeto mais específico pode ser usado como uma instância do elemento mais geral. A generalização, também chamada de herança, permite a criação de elementos especializados em outros.

Existem alguns tipos de generalizações que variam em sua utilização a partir da situação. São elas: generalização normal e restrita. As generalizações restritas se dividem em generalização de sobreposição, disjuntiva, completa e incompleta.

3.4.7.1 Generalização Normal

Na generalização normal a classe mais específica, chamada de subclasse, herda tudo da classe mais geral, chamada de superclasse. Os atributos, operações e todas as associações são herdadas.

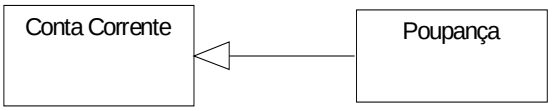


Figura 18 : Exemplo de uma generalização normal

Uma classe pode ser tanto uma subclasse quanto uma superclasse, se ela estiver numa hierarquia de classes que é um gráfico onde as classes estão ligadas através de generalizações.

A generalização normal é representada por uma linha entre as duas classes que fazem o relacionamento, sendo que coloca-se um seta no lado da linha onde encontra-se a superclasse indicando a generalização.

3.4.7.2 Generalização Restrita

Uma restrição aplicada a uma generalização especifica informações mais precisas sobre como a generalização deve ser usada e estendida no futuro. As restrições a seguir definem as generalizações restritas com mais de uma subclasse:

Generalizações de Sobreposição e Disjuntiva: Generalização de sobreposição significa que quando subclasses herdam de uma superclasse por sobreposição, novas subclasses destas podem herdar de mais de uma subclasse. A generalização disjuntiva é exatamente o contrário da sobreposição e a generalização é utilizada como padrão.

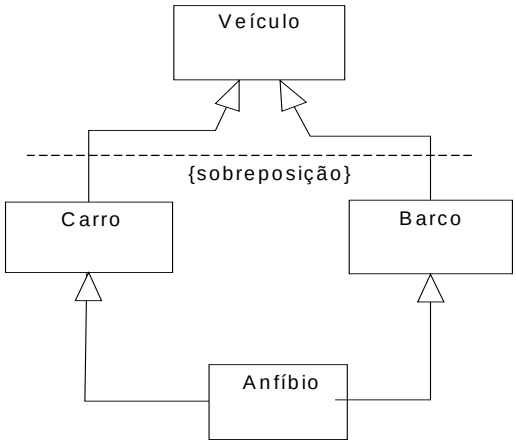


Figura 19 : Exemplo de uma generalização de sobreposição

- Generalizações Completa e Incompleta: Uma restrição simbolizando que uma generalização é completa significa que todas as subclasses já foram especificadas, e não existe mais possibilidade de outra generalização a partir daquele ponto. A generalização incompleta é exatamente o contrário da completa e é assumida como padrão da linguagem.

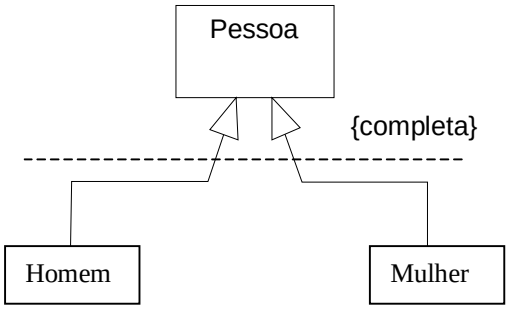


Figura 20: Exemplo de uma generalização completa

3.5 Diagrama de Classes

O diagrama de classes demonstra a estrutura estática das classes de um sistema onde estas representam as “coisas" que são gerenciadas pela aplicação modelada. Classes podem se relacionar com outras através de diversas maneiras: associação (conectadas entre si), dependência (uma classe depende ou usa outra classe), especialização (uma classe é uma especialização de outra classe), ou em pacotes (classes agrupadas por características similares). Todos estes relacionamentos são mostrados no diagrama de classes juntamente com as suas estruturas internas, que são os atributos e operações. O diagrama de classes é considerado estático já que a estrutura descrita é sempre válida em qualquer ponto do ciclo de vida do sistema. Um sistema normalmente possui alguns diagramas de classes, já que não são todas as classes que estão inseridas em um único diagrama e uma certa classes pode participar de vários diagramas de classes.

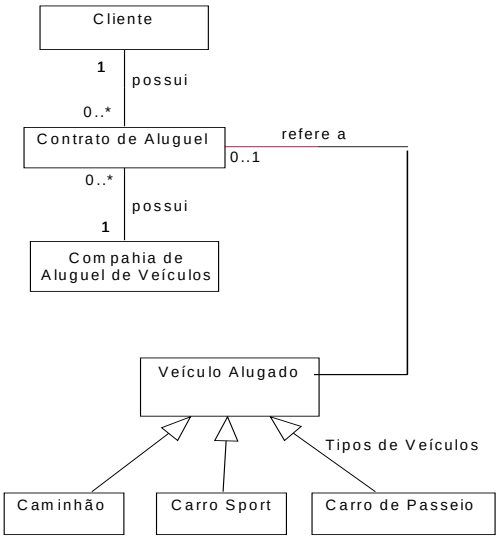


Figura 21: Exemplo de Diagrama de Classe

3.6 Diagrama de Estado

O comportamento de uma Classe de Objetos é representado através de um Diagrama de Transição de Estado, que descreve o ciclo de vida de uma dada classe, os eventos que causam a transição de um estado para outro e as ações resultantes da mudança de estado.

O espaço mostra os estados de uma dada Classe corresponde a enumeração de todos os estados possíveis de um objeto.

O estado de um Objeto é uma das possíveis condições na qual o objeto pode existir. O estado compreende todas as propriedades do objetos (estáticas) associadas aos valores correntes (dinâmico) de cada uma dessas propriedades.

A notação UML para representar o estado de uma classe corresponde a um retângulo com a bordas abauladas.

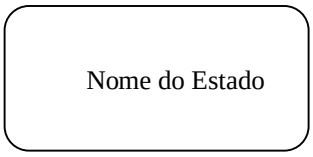


Figura 22: Estado

3.6.1 Estados e Atributos

Estados podem ser distinguidos pelos valores assumidos por certos atributos.

Exemplo - O número máximo de estudantes por curso, no "Use Case" Matrícula do Aluno, é igual a 10.

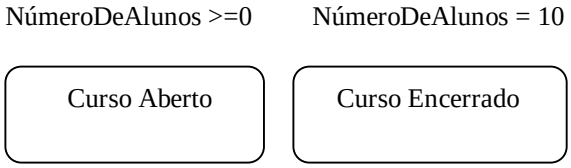


Figura 23: Estado e Atributo

3.6.2 Estados e Ligações

Estados também podem ser distinguidos pela existência de certas ligações.

Exemplo – A instância da Classe Professor pode ter dois estados:

Ensinando: quando o Professor esta ministrando um Curso.

Licenciado: quando não esta ministrando nenhum Curso.

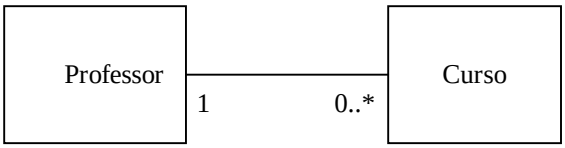


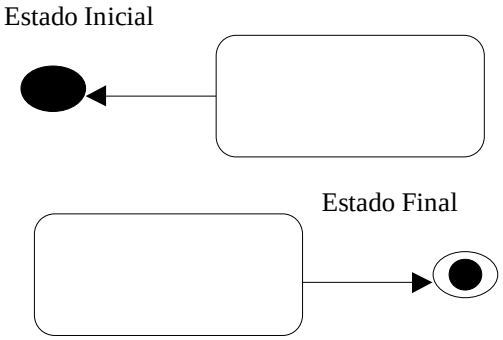
Figura 24: Estado e Ligação

Estados possíveis da Classe Professor.

3.6.3 Estados Especiais

- **Estado Inicial** é o estado atribuído a um objeto quando é criado. O estado Inicial tem as seguintes características:
  - É mandatório
  - Somente um estado Inicial é permitido.
  - O estado Inicial é representado por um círculo preenchido.

- **Estado Final** é o estado que indica o fim do ciclo de vida de um objeto. O estado Final tem as seguintes características:
    - É opcional.
    - Pode existir mais de um estado final.
    - O estado Final é representado por um "olho de boi".



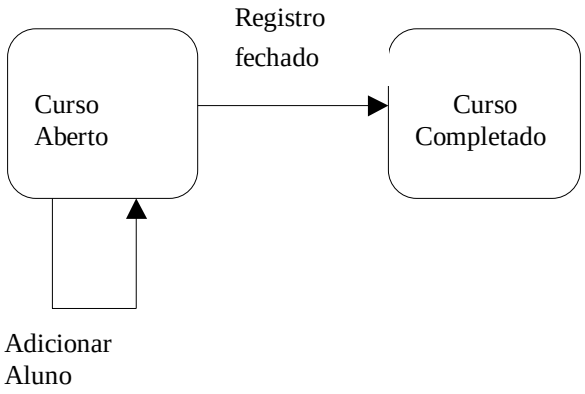
3.6.4 Eventos

Um evento é uma ocorrência que acontece em algum ponto no tempo e que pode modificar o estado de um objeto, podendo gerar uma resposta. Exemplo:

- Adicionar um aluno a um curso.
- Criar um novo curso.

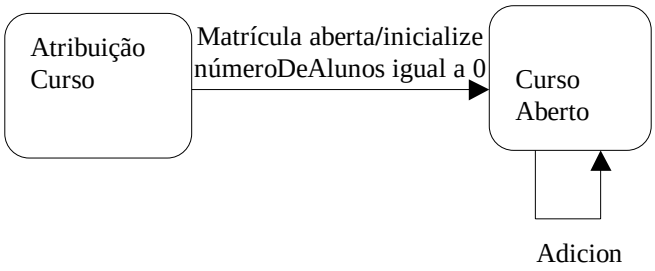
Transição

É a mudança do estado atual para o estado subseqüente como resultado de algum estímulo. O estado subseqüente pode ser igual ao estado original. Uma transição pode ocorrer em resposta a um evento. As transições rotuladas com o nome dos eventos.



3.6.5 Ações

É uma operação que está associada a uma transição, ocorrendo instantaneamente e que não pode ser interrompida. Nome de uma ação é mostrado, na seta indicativa da transição, precedida por um barra inclinada (/).



3.6.6 Envio de eventos a partir de outro evento

Um evento pode provocar o envio de outro evento. O nome do evento enviado é mostrado, na seta indicativa de transição, precedido por um circunflexo (^) seguido pelo nome da Classe para onde o evento será enviado, separados por um ponto. Exemplo: Evento1^Classe.Evento2, onde Evento1 é o evento que causou a transição e Evento2 é o evento gerado a partir da transição.

3.6.7 Envio de eventos a partir de atividade

Uma atividade também pode provocar o envio de um evento para um outro Objeto.

3.6.8 Transição Automática

Algumas vezes, o único propósito da existência de um estado é desenvolver uma atividade. Uma transição automática ocorre quando a atividade é completada. Se múltiplas transições automáticas existem, uma condição de guarda é necessária para cada transição e as condições de guarda devem ser mutuamente exclusivas.

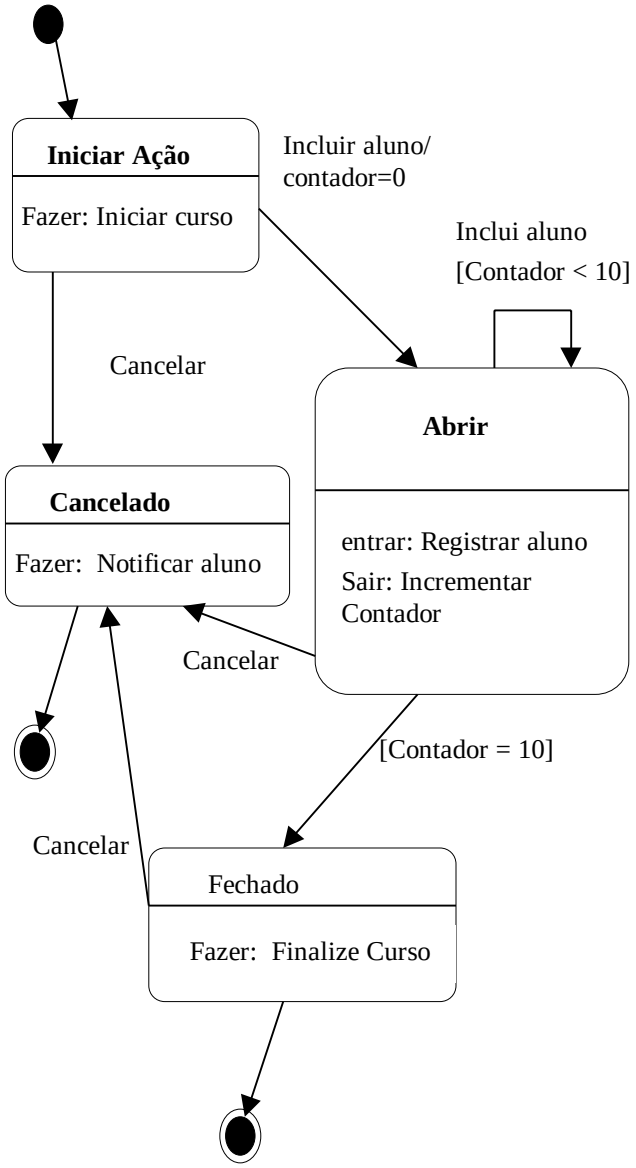


Figura 25: Exemplo de Diagrama de Estado para Matricula

4 Conclusão

A UML e incorporou muitos recursos com que permite a modelagem visual para sistemas complexas utilizando a orientação a objetos

Através da utilização da UML, os desenvolvedores e os cliente terão maior comunicação e aproveitamento dos modelos A reutilização é a palavra chave onde os modelos criados poderão ser facilmente reutilizados nos futuras aplicações.

Vários ferramentas case existentes já estão utilizando a UML como a linguagem principal.

5. References

COAD, Peter , NORTH, David, MAYFIELD, Mark. Object models : strategies, patterns, and applications. Englewood Cliffs : Prentice Hall, 1995.

COAD, Peter, YOURDON, Edward. Análise baseada em objetos. Rio de Janeiro: Campus, 1992.

COAD, Peter, YOURDON, Edward. Object-oriented design. Englewood Cliffs: Yourdon: Prentice Hall, 1991.

COAD, Peter, NICOLA, Jill. Object-oriented programming. Englewood Cliffs : PTR Prentice-Hall, 1993.

FURLAN, José David. Modelagem de objetos através da UML – the unified modeling language. São Paulo: Makron Books, 1998.

JACOBSON, Ivar. Object-oriented software engineering: a use case driven approach. Addison-Wesley, 1994.

MARTIN, James. Princípios de análise e projeto baseados em objetos. Rio de Janeiro: Campus, 1994.

MARTIN, James, ODELL, James J. Análise e projeto orientados a objeto. São Paulo : Makron Books, 1996.

RUMBAUGH, James et al. Modelagem e projetos baseados em objetos. Rio de Janeiro : Campus, 1994.

TUDOR, D.J., TUDOR, I.J. System analysis and design: a comparison of structured methods. Oxford: NCC Blackwell, 1995.

WINBLAD, Ann L et al. Software orientado ao objeto. São Paulo: Makron Books, 1993.

- Documentação oficial da UML. Disponível na Internet no endereço: http://www.rarional.com/uml