**- MÓDULO 3 -**
**MODELAGEM DE SISTEMAS ORIENTADA A OBJETOS COM UML**

## 1. INTRODUÇÃO

A partir de 1980, diversos métodos de desenvolvimento de sistemas surgiram para apoiar o paradigma orientado a objetos com uma grande diversidade de autores. No início da década de 90 os pesquisadores James Rumbaugh, Ivar Jacobson e Grady Booch uniram as melhores características destacadas em suas técnicas de modelagem e desenvolveram um padrão de referência para modelagem orientada a objetos - a Linguagem de Modelagem Unificada – *Unified Modeling Language*(UML).

A UML consiste da fusão dos métodos de Booch, Rumbaugh (OMT- *Object Modeling Technique*) e Jacobson (OOSE – *Object-Oriented Software Engineering*). A fusão iniciou com o trabalho de Rumbaugh e Booch, que criaram um método através de pontos fortes de cada um, surgindo o *Unified Method* - UM 0.8, apresentado ao público em 1995. Logo a seguir, em meados de 1996, Jacobson integrou-se ao grupo e lançaram a UML versão 0.9. A partir daí, criaram forças com cooperação de grandes empresas, lançando no mercado com aprovação da Agência Americana de Padrões - *Object Management Group*(OMG) em julho de 1997, considerando um padrão mundial. A concretização da UML aconteceu em 1997.

Conforme Booch; Jacobson e Rumbaugh (2000, p. 13), a UML é "**uma linguagem padrão para a elaboração da estrutura de projetos de software. A UML é uma linguagem para visualização, especificação, construção e documentação de artefatos que façam uso de sistemas complexos de software**".

A UML contempla uma representação gráfica, através das técnicas de modelagem que especificam vários elementos (objetos, classes, atributos, etc) da abordagem orientada a objetos. A UML não se limita a um Modelo de Engenharia de Software e não se vincula, exclusivamente, a uma etapa do processo de desenvolvimento, mas se apoia no desenvolvimento incremental, através de modelos que podem evoluir com a inclusão de novos detalhes.

Um modelo é uma descrição simplificada da realidade, apresentado a partir de uma perspectiva específica e criado para proporcionar uma melhor compreensão do sistema. Cada modelo pode ser expresso em diferentes níveis de precisão, constituindo um conjunto de diagramas consistentes entre si e acompanhados de técnicas de modelagem textuais. Um *diagrama* é uma visão sobre um modelo, o qual proporciona uma representação parcial do sistema.

Booch, Jacobson e Rumbaugh (2000) consideram as principais características da UML:

**a) Centrado na arquitetura***:* a arquitetura do sistema é utilizada como principal artefato para a conceituação, construção, gerenciamento e evolução do sistema em desenvolvimento, representando uma visão do projeto como um todo. O conceito de arquitetura de software incorpora os aspectos estáticos e dinâmicos mais importantes do sistema. A arquitetura é influenciada por muitos fatores, tais como a plataforma de software sobre a qual o sistema vai rodar (sistema operacional, sistema gerenciador de banco de dados, protocolos para comunicação em rede, etc.), blocos de construção reutilizáveis (*frameworks, componentes e patterns),* considerações de distribuição, sistemas legado e requisitos não funcionais;

**b) Orientado a Casos de Uso***:* os casos de uso são utilizados como o principal artefato para o estabelecimento do comportamento desejado do sistema;

**c) Processo Iterativo***:* contempla o gerenciamento de sequências de versões executáveis e incrementais, sendo que cada nova versão incorpora os aprimoramentos incrementais em relação às demais. Uma versão do sistema liberada resulta em uma iteração concluída.

A UML 2.0 abrange quatorze técnicas de modelagem, classificadas em estrutural e comportamental. As técnicas de modelagem **estruturais** enfatizam a estrutura dos elementos, a partir da identificação dos objetos, colaborando para modelagem estática do sistema. As técnicas de modelagem **comportamentais** enfatizam o comportamento e a interação entre os elementos do sistema, colaborando para modelagem dinâmica do sistema. A maioria das técnicas de modelagem é especificada no formato de diagramas complementadas com técnicas descritivas.

## 2. FUNDAMENTO DA ORIENTAÇÃO A OBJETOS

Antes de abordarmos a UML é importante relembrar alguns conceitos acerca da Orientação a Objetos.

As ideias fundamentais sobre a tecnologia baseada em objetos incluem: Objetos, Classes, Abstração, Encapsulamento, Herança, Comunicação com Mensagens, Métodos de organização, Polimorfismo. Cada uma dessas ideias serão discutidas a seguir.

### 2.1. Objetos

Objeto é um substantivo que pode ser abstrato ou real, sobre o qual armazenamos dados e operações que manipulam os dados, como uma pessoa, um avião, etc.

Um objeto é a ocorrência específica (instância) de uma classe e é similar a uma entidade de uma tabela no modelo relacional somente até o ponto onde representa uma coleção de dados relacionados com um tema em comum. O nome ou CPF de um empregado são dados que pertencem à entidade (Organização) ou ao objeto (Pessoas). Pessoas é uma mesma organização. Uma instância de Empregado é uma pessoa específica.

A diferença entre objeto e entidade é que a entidade se preocupa meramente com os dados, armazenando apenas um registro para cada entidade e o objeto se preocupa com os dados e os métodos através dos quais os dados estão manipulados e podem incluir tipos de registros múltiplos.

### 2.2. Classe

Classe é uma coleção de objetos que podem ser descritos com os mesmos atributos e as mesmas operações. Representa uma ideia ou um conceito simples e categoriza objetos que possuem propriedades similares, configurando-se em um módulo para a criação de novas instâncias.

### 2.3. Abstração

Quando usamos a abstração, admitimos que o que estamos considerando é complexo. Pois é uma das formas fundamentais de se lidar com complexidade. O resultado deste processo de abstração é conhecido como Modelo Conceitual.

Dentro do paradigma de Orientação a Objeto, abstração denota características essenciais a um objeto que o distingue de outros tipos de objetos provendo fronteiras bem definidas entre diferentes visões.

A abstração focaliza a visão externa de um objeto, separando-a em comportamento, descrito pela interface dos métodos da classe e implementação, composta pelas variáveis de instancia e corpo dos métodos.

### 2.4. Encapsulamento

O empacotamento de dados e métodos juntas é chamada de encapsulamento. O objeto esconde seus dados de outros objetos e permite que os dados sejam acessados somente por meio dos próprios métodos de objetos.

O encapsulamento esconde, dos usuários de um objeto, os detalhes da implementação interna. Os usuários compreendem quais operações podem ser solicitadas de um objeto, mas não conhecem os detalhes de como a operação é executada.

### 2.5. Herança

É o mecanismo de compartilhamento automático de métodos e dados entre classes, subclasses.

Exemplo são as classes Pessoa Física e Pessoa Jurídica que são subclasses da classe de Pessoa e, portanto, herdam as suas propriedades.

### 2.6. Mensagens

Uma mensagem é uma solicitação para executar a operação indicada, em um determinado objeto, e devolver a resposta. Esta mensagem contém o nome do objeto, o nome da operação e, às vezes, um grupo de parâmetros.

A programação baseada em objetos é uma forma de projeto modular no qual o mundo é visto em termos de objetos, operações, métodos e mensagens trocadas entre objetos.

### 2.7. Polimorfismo

Permite que o programa trate uniformemente objetos que pertencem a classe diferentes, isto é, o programa envia a mesma mensagem a objetos de diferentes classe resultando um comportamento diferente.

Existem alguns tipos de Polimorfismo, que estão citados abaixo:

- Paramétrico: quando uma função funciona uniformemente sobre uma gama de tipos.
- De Inclusão: através do mecanismo de herança de objeto pode pertencer a um número infinito de classes, aplicação do princípio de substituição.
- Por *Overloading*: mesmo nome de variável utilizado para denotar diferentes funções. É o contexto que decide que função utilizar.
- Por Coerção: operação semântica é utilizada para converter argumento para tipo necessário.

## 3. UML (Linguagem de Modelagem Unificada)

Para melhor compreender a UML é preciso entender o significado de cada palavra que a compõe:

- Linguagem: usada para expressar e comunicar ideias.
- Modelagem descrever um sistema em um alto nível de abstração.
- Unificada: porque se tornou o padrão mundial para modelagem de sistemas.

A UML, então, é uma linguagem gráfica para especificar, visualizar, construir e documentar os artefatos de software.

Apresenta as seguintes vantagens:

- Usa notação gráfica: mais clara que a linguagem natural (imprecisa) e código (muito detalhado);
- Ajuda a obter uma visão geral do sistema;
- Não é dependente de tecnologia;
- Diminui a fragmentação, aumenta a padronização

A UML suporta todo o ciclo de vida do software, tais como:

- modelagem do negócio (processos e objetos do negócio);
- modelagem de requisitos alocados ao software;
- modelagem da solução de software.

Convém lembrar que a UML não é uma metodologia.

A UML evoluiu ao longo dos anos, como se vê na lista a seguir:

- 1.1 OMG (*Object Management Group* – 1997)
- 1.3 (2000)
- 2.0 (2005)
- 2.4 (2010 - http://www.omg.org/spec/UML/2.4/)
- 2.4.1. (2011 - http://www.omg.org/spec/UML/2.4.1)
- 2.5 (2013 – http://www.omg.org/spec/UML/2.5/Beta2)

A última versão é a UML 2.5 e está estruturada conforme figura 1.

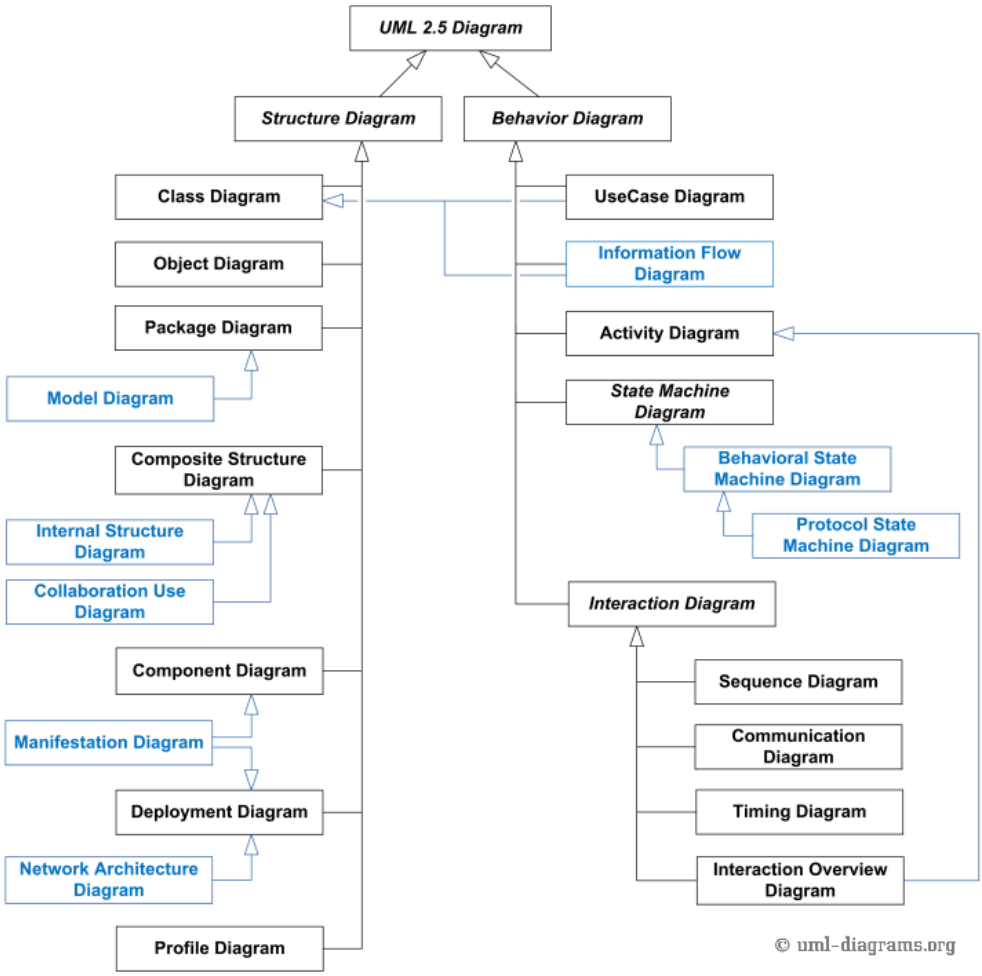


**Figura 1 - UML 2.5**
**Fonte: http://www.uml-diagrams.org/uml-25-diagrams.html**

A arquitetura da UML especifica dois conjuntos de diagramas: os diagramas de estrutura e os diagramas comportamentais.

### 3.1. Diagramas de Estrutura (Estáticos)

Os diagramas de estrutura mostram a estrutura estática do sistema e de suas partes em diferentes níveis de abstração e de implementação, bem como a forma como estão relacionadas entre si. Os elementos em um diagrama de estrutura representam os conceitos significativos de um sistema e podem incluir abstrações, aspectos do mundo real e conceitos de implementação.

Diagramas de estrutura não utilizam conceitos relacionados ao tempo e não mostram os detalhes de comportamento dinâmico. No entanto, eles podem mostrar relações entre objetos.

**Tabela 1 – Diagramas de Estrutura**

| |
|---|
| Diagrama de Classe |
| Diagrama de Objeto |
| Diagrama de Pacotes |
| Diagrama de Componentes |
| Diagrama de Implantação |

| Diagrama de Estrutura Composta |
| --- |
| Diagrama de Perfis |

### 3.1.1. DIAGRAMA DE CLASSE

É um diagrama estático da UML que reúne os elementos mais importantes de um sistema orientado a objetos. Nele são exibidos um conjunto de classes, interfaces e seus relacionamentos. As classes especificam tanto as propriedades quanto os comportamentos dos objetos.

A estrutura de uma classe se divide em três partes: o nome da classe, as propriedades (atributos) e as operações, como exemplificado na figura 2.

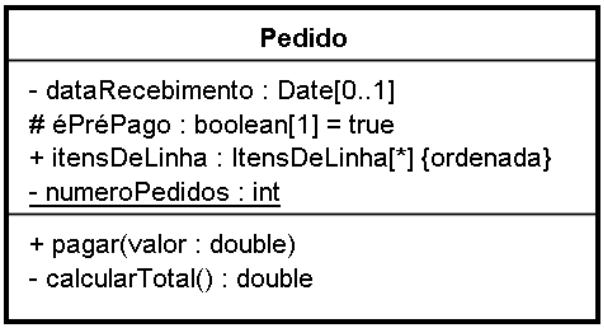


**Figura 2 - Classe**

Acima observa-se a representação mais comumente vista, contudo a UML permite representar as classes em um maior ou menor nível de abstração, apresentando mais ou menos detalhes, como se vê na figura 3.

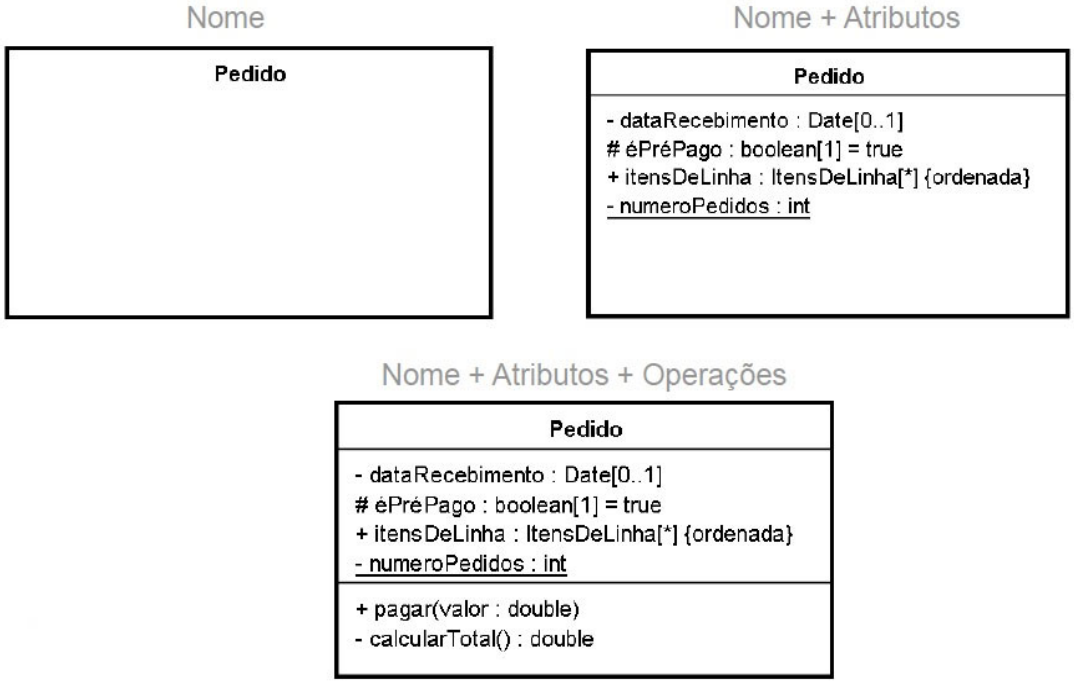


**Figura 3 - Formas de representação da classe**

3.1.1.1. Estrutura da Classe

A estrutura de uma classe é cheia de pormenores. Esta estrutura é composta das seguintes partes:

- Nome da classe: É o nome da classe. Pela notação deve iniciar com letra maiúscula.
- Atributos ou propriedades: Um atributo é um dado para o qual cada objeto tem seu próprio valor. São, basicamente, a estrutura de dados que vai representar a classe.
- Operações ou métodos: Funções ou comportamentos da classe. Importante observar a passagem de parâmetro e retorno das operações.
- Visibilidade: denota como um elemento da UML pode ser visto e utilizado por outro elemento, ou seja, são modificadores de visibilidade: privado(-), protegido(#) e público(+).

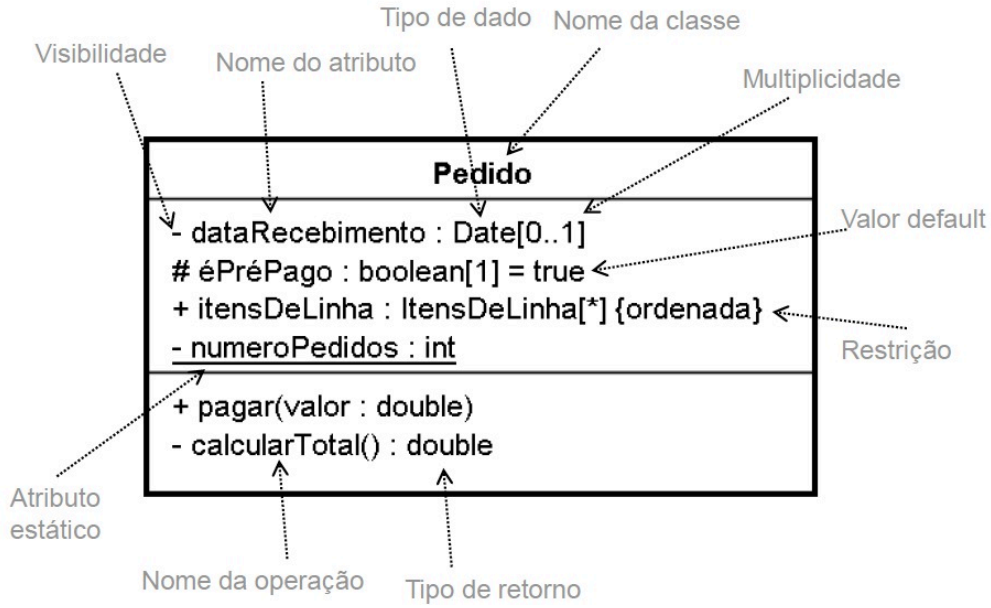


**Figura 4 - Estrutura de uma classe**

3.1.1.2. Atributo

Os Atributos em Programação Orientada a Objetos são os elementos que definem a estrutura de uma classe. Os atributos também são conhecidos como variáveis de classe, e podem ser divididos em dois tipos básicos: atributos de instância e de classe.

Nos **atributos de instância** cada objeto tem o seu próprio valor para o atributo. É o escopo default da UML. Em um **atributo de classe** o valor do atributo é comum a todos os objetos daquela classe. Para denotar este escopo o atributo deve ser sublinhado. Atributos de classe podem ser chamados também de atributos estáticos. Constantes são considerados atributos de classe quando estão fora de qualquer método.

Na especificação do atributo na classe é importante observar e conhecer sua especificação completa, como se vê abaixo:

| **Visibilidade nome: tipo [multiplicidade] = valor_default {lista de restrições}** |
|---|

- Nome: corresponde ao nome do atributo
- Tipo: domínio do atributo
- Multiplicidade: indicação de quantos objetos podem preencher a propriedade [min..max]
- Valor Default: valor do atributo, caso ele seja omitido no momento da criação
- Restrição: permite indicar propriedades adicionais. {*readOnly*}, {*ordered*}, {*unique*}, etc.

3.1.1.3. Operações/Métodos

Os métodos determinam o comportamento dos objetos de uma classe e são análogos às funções ou procedimentos da programação estruturada.

Assim como no atributo, é importante observar e conhecer sua especificação completa, como se vê abaixo:

| **Visibilidade nome (lista de parâmetros): tipo-de-retorno {lista restrições}** |
|---|

- Nome: corresponde ao nome da operação
- Lista de parâmetros: são os parâmetros da operação.
- Tipo de retorno: o tipo de dado retornado pela operação
- Restrição: permite indicar propriedades adicionais. ex: {*query*}.

Na orientação é possível implementar métodos abstratos e estáticos e de igual forma é necessário representa-los no diagrama de classe.

As **operações abstratas**, ou seja, que não têm uma implementação específica, devem ser escritas em itálico e as **operações estáticas** devem ser escritas com fonte sublinhada.

3.1.1.4. Modificadores de Visibilidade

Tanto atributos como operações são definidos por uma visibilidades:

- Público (+): O elemento é visível por qualquer classe;
- Protegido (#): O elemento é visível na própria classe e pelas subclasses da classe;
- Pacote (~): O elemento é visível apenas pela própria classe ou dentro do pacote onde a classe está localizada;
- Privado (-): O elemento é visível apenas pela própria classe.

3.1.1.5. Relacionamentos

Os relacionamentos ligam classes entre si, criando relações lógicas. Podem ser classificadas como:

- Associação:
  - Simples
  - Agregação
  - Composição
- Generalização
- Dependência
- Realização

a) Associação Simples: Indica que objetos de um elemento estão ligados a objetos de outro elemento. A navegabilidade pode ser unidirecional ou bidirecional.

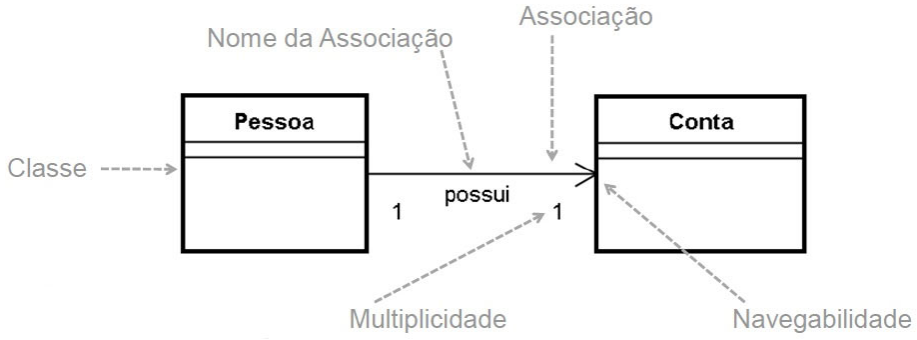


**Figura 5 - Associação Simples**

b) Associação Qualificada: Um qualificador de associação é um atributo do elemento-alvo capaz de identificar uma instância dentre as demais.

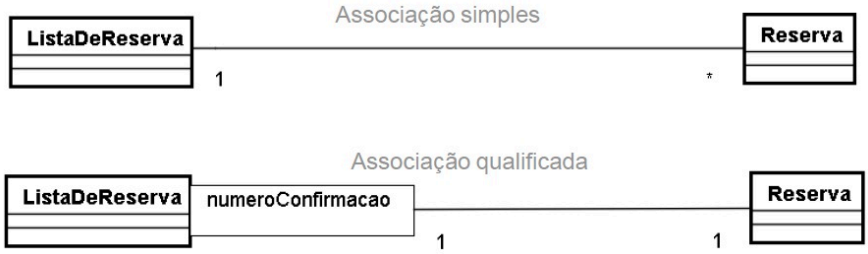


**Figura 6 - Associação Qualificada**

Uma associação pode mostrar as mesmas informações que um atributo, como se vê na figura 7.

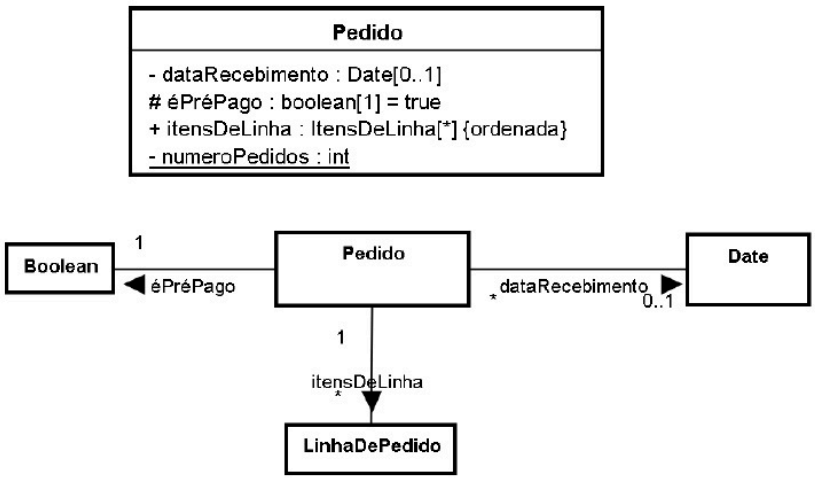


**Figura 7 - Associação x Atributo**

c) Agregação e Composição: São os relacionamentos conhecidos por “todo-parte”. Na **agregação** a parte existe sem o todo e na **composição** o todo controla o ciclo de vida da parte, e ela não pode ser compartilhada em outros relacionamentos.

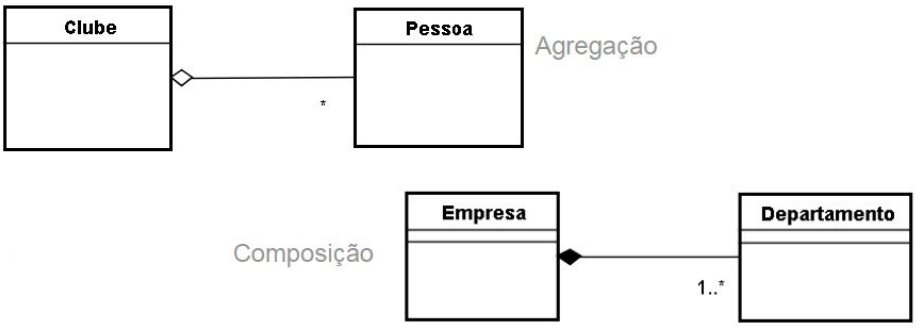


**Figura 8 - Agregação e Composição**

d) Dependência: Indica que mudança em um elemento pode causar mudanças no outro (uso).

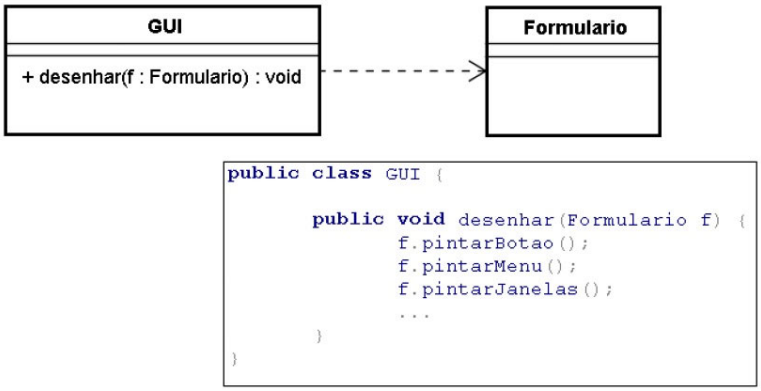


**Figura 9 - Dependência**

Pode ocorrer, também, entre uma classe e uma interface, como ilustra a figura 10.

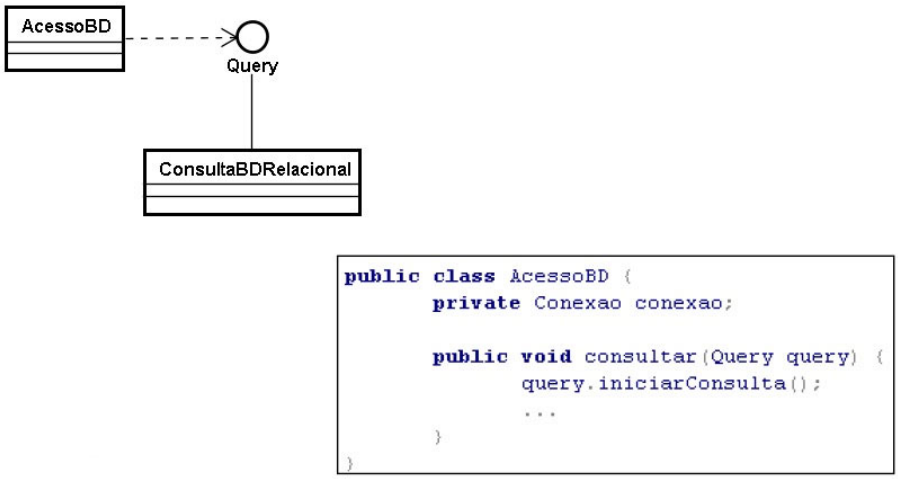


**Figura 10 - Dependência**

e) Generalização: É um relacionamento conhecido como “é um tipo de”.

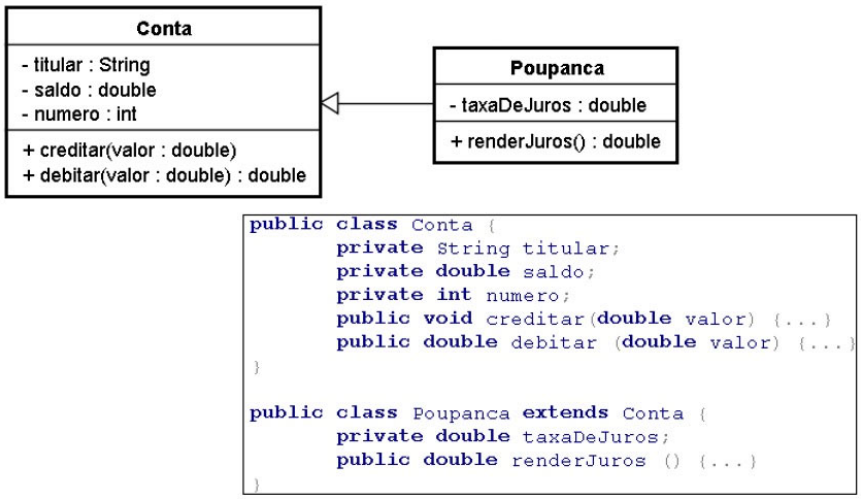


**Figura 11 - Generalização**

Pode ocorrer, também, entre interfaces, como ilustra a figura 12.

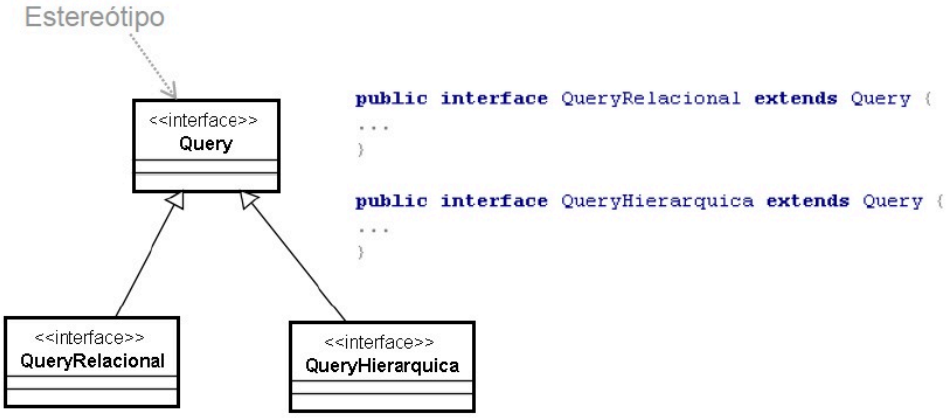


**Figura 12 - Generalização entre interfaces**

f) Realização: Há várias notações para realizações.

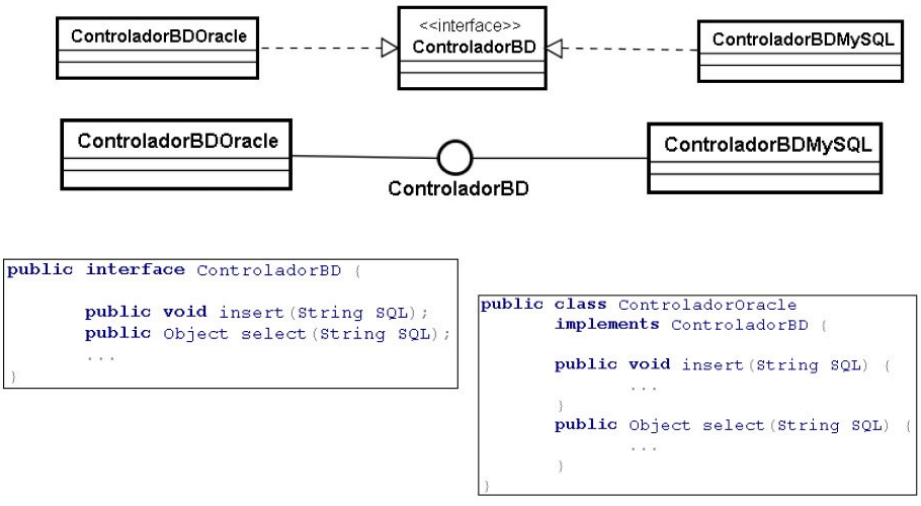


**Figura 13 - Realização**

A notação “bola-soquete” (UML 2.0) é utilizada para modelar uma dependência e uma realização entre duas classes e uma interface, como se vê na figura 14.

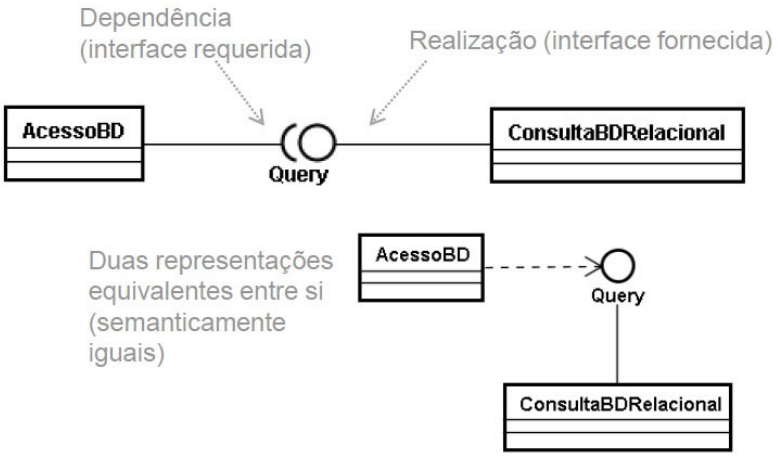


**Figura 14 - Realização (notação bola-soquete)**

A figura 15 exemplifica a utilização de inúmeros relacionamentos em um diagrama de classe.

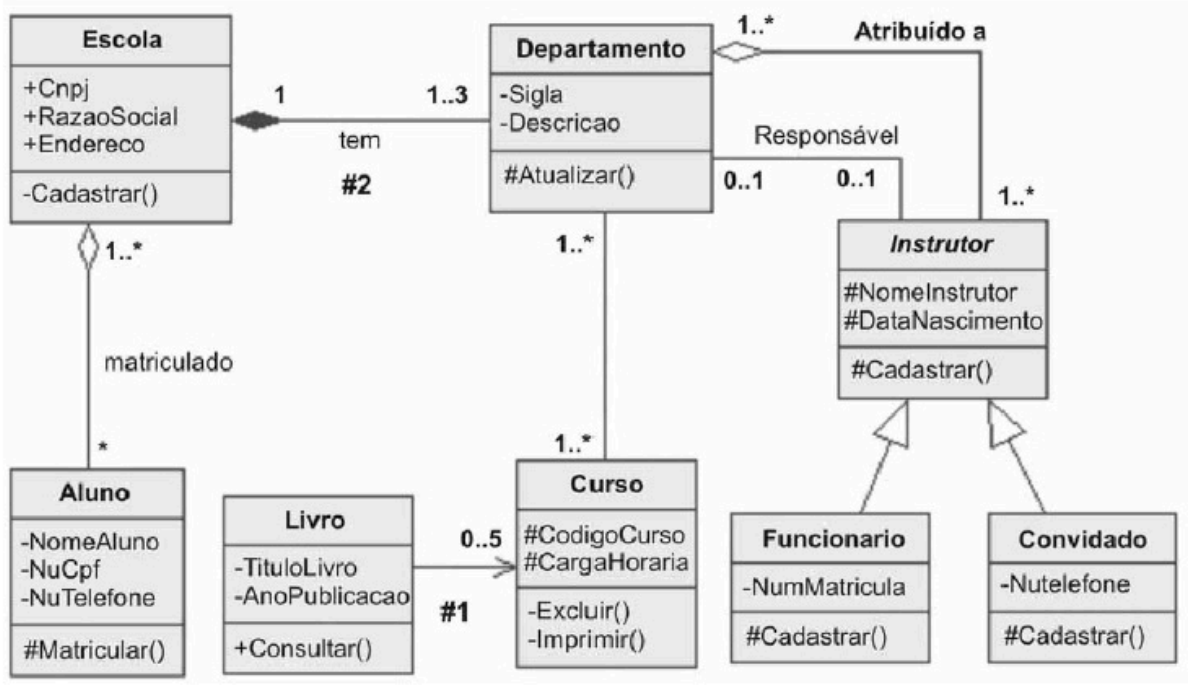


**Figura 15 - Diagrama de Classe**

### 3.1.2. DIAGRAMA DE OBJETO

O diagrama de objetos representa uma fotografia do sistema em um dado momento. Mostra os vínculos entre os objetos conforme estes interagem e os valores dos seus atributos.

Pode ser visto como uma “instância” do diagrama de classe.

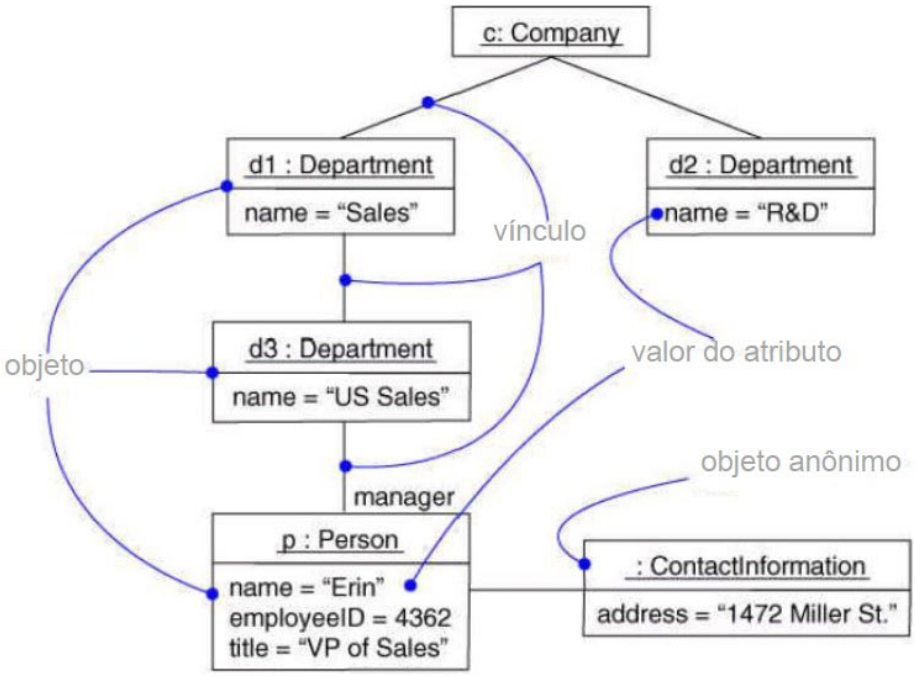


**Figura 16 - Diagrama de Objetos**

### 3.1.3. DIAGRAMA DE PACOTE

Pacotes são estruturas que permitem agrupar qualquer construção da UML em estruturas de alto nível. Pode mostrar:

- Pacotes e suas dependências;
- Interfaces entre os pacotes;
- Generalizações entre pacotes.

É possível representa-lo de duas formas, como se vê na figura 17 e 18.

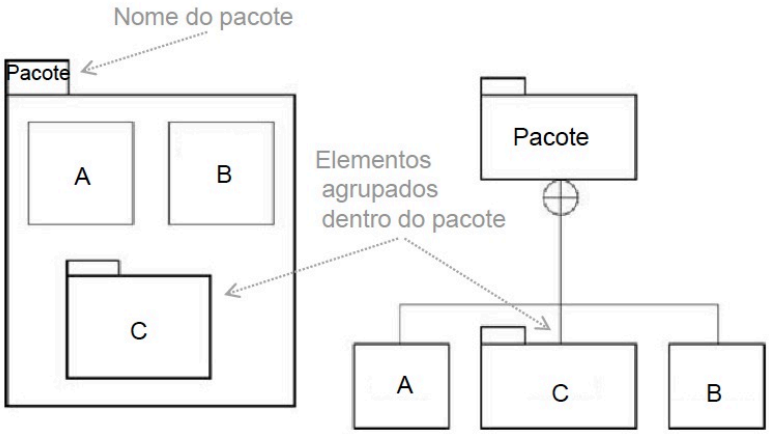


**Figura 17 - Diagrama de Pacotes**

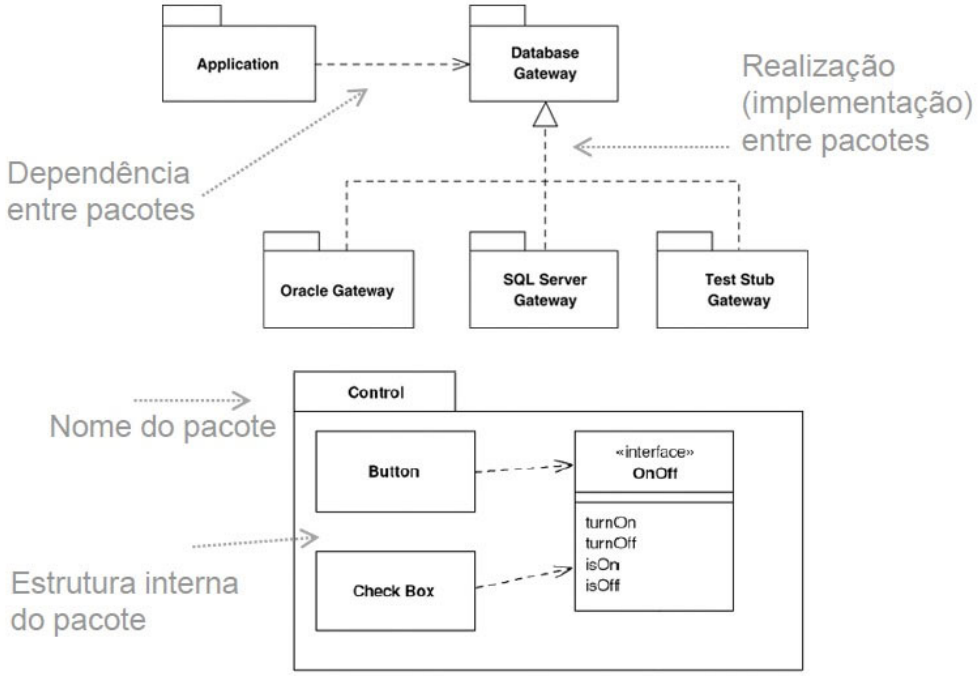


**Figura 18 - Diagrama de Pacotes (outra forma de representá-lo)**

### 3.1.4. DIAGRAMA DE COMPONENTES

Modela o sistema em termos de componentes e seus relacionamentos através de interfaces. Permite decompõe o sistema em subsistemas que detalham a estrutura interna. Alguns componentes existem em tempo de ligação, outros em tempo de execução.

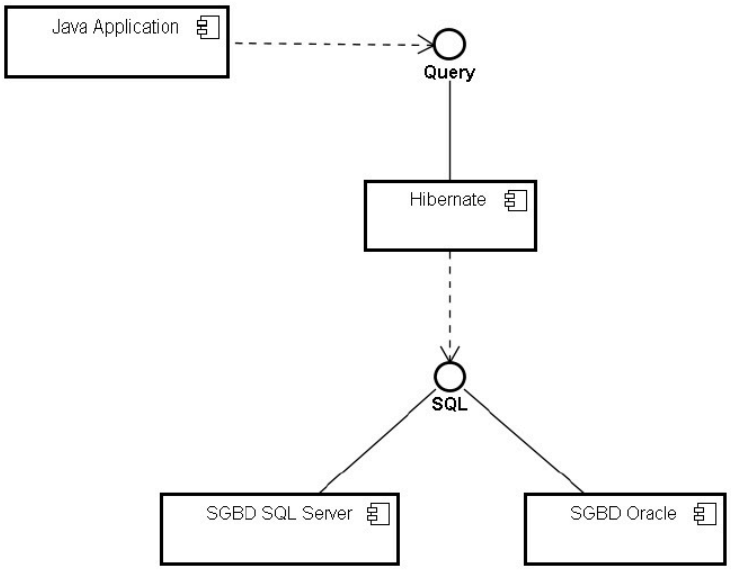


**Figura 19 - Diagrama de Componentes**

### 3.1.5. DIAGRAMA DE IMPLANTAÇÃO

Este diagrama modela a configuração física do sistema, revelando que pedaços de software rodam em que equipamentos de hardware. Inclui:

- Nós: que podem ser dispositivos (Hardware) ou ambientes de Execução;
- Artefatos: que podem ser código fonte, código binário, executáveis, etc.

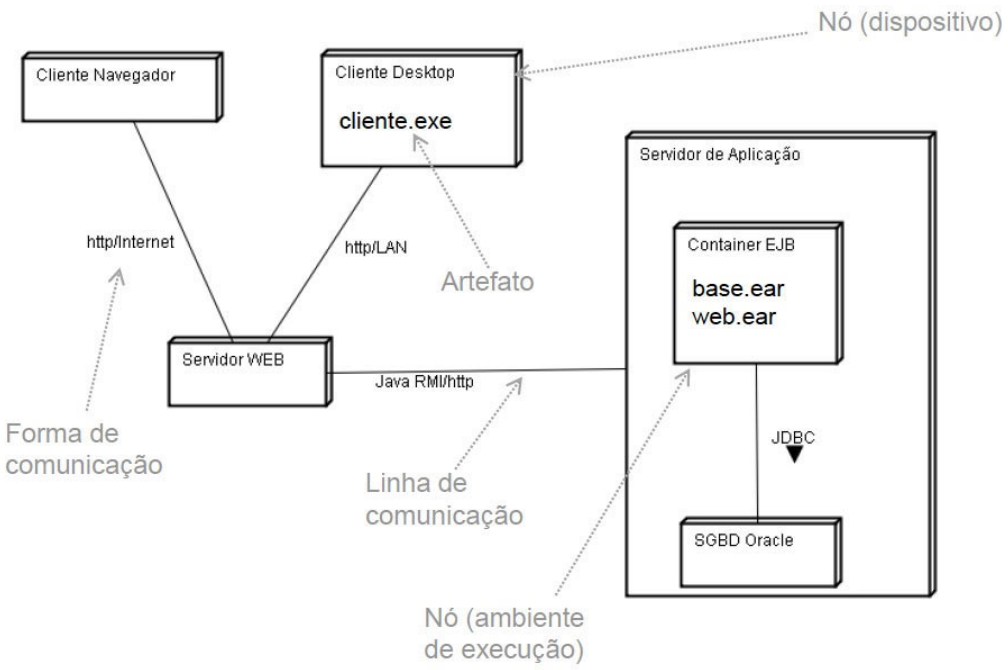


**Figura 20 - Diagrama de Implantação**

### 3.1.6. DIAGRAMA DE ESTRUTURA COMPOSTA

É utilizado para modelar colaborações entre interfaces, objetos ou classes. Pode ser usado para descrever:

- Estruturas de partes interconectadas;
- Estruturas de instâncias interconectadas.

Neste diagrama destaca-se o conceito de **parte** que representa o conjunto de uma ou mais instâncias contidas em outro elemento; e o conceito de **porta** que é o ponto de interação entre os elementos.

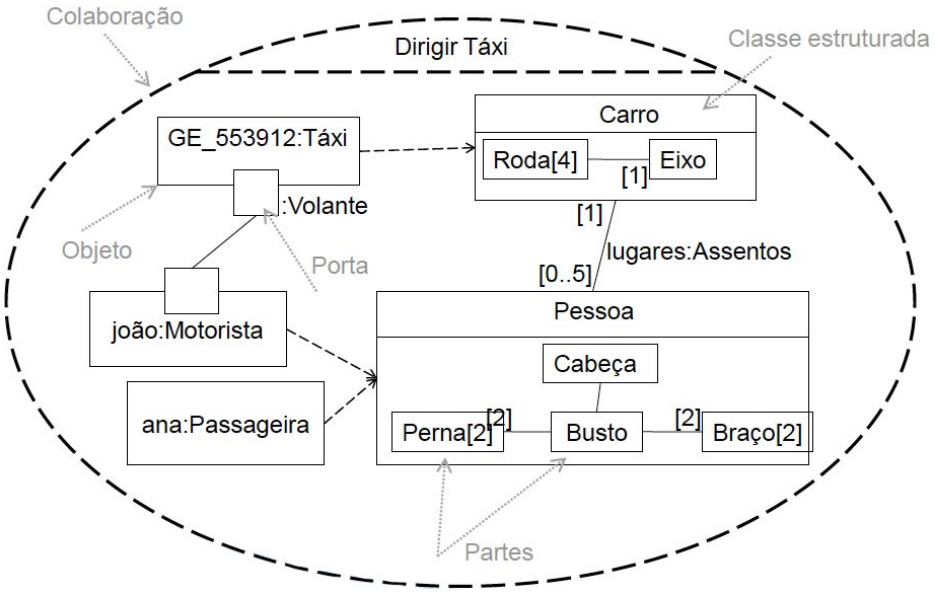


**Figura 21 - Diagrama de Estrutura Composta**

### 3.1.7. DIAGRAMA DE PERFIL

É um diagrama auxiliar que permite definir tipos padronizados de estereótipos, valores rotulados e restrições. A UML define o mecanismo de perfis como um "mecanismo leve de extensão" da linguagem. Permite adaptar os modelos UML para diferentes plataformas e domínios.

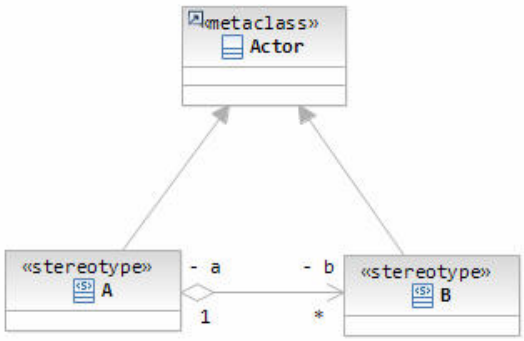


**Figura 22 - Diagrama de Perfil**

## 3.2. Diagramas Comportamentais (Dinâmicos)

Mostram a natureza dinâmica dos objetos do sistema, que pode ser descrita como uma série de mudanças no sistema com o passar do tempo.

**Tabela 2 - Diagramas de Comportamento**

| |
|---|
| Diagrama de Caso de Uso |
| Diagrama de Atividade |
| Diagrama de Máquina de Estados |

Diagramas de Interação
- Diagrama de Sequência
- Diagrama de Comunicação
- Diagrama de Tempo
- Diagrama de Interação Geral

### 3.2.1. DIAGRAMA DE CASO DE USO

Contém um conjunto de casos de uso e modela interações entre atores e o sistema e o próprio sistema.

É utilizado para descreve um conjunto de cenários, capturar requisitos do usuário e delimitar o escopo do sistema.

Por se tratar de um diagrama que já foi estudado em seções anteriores, não serão abordados detalhes de sua implementação.

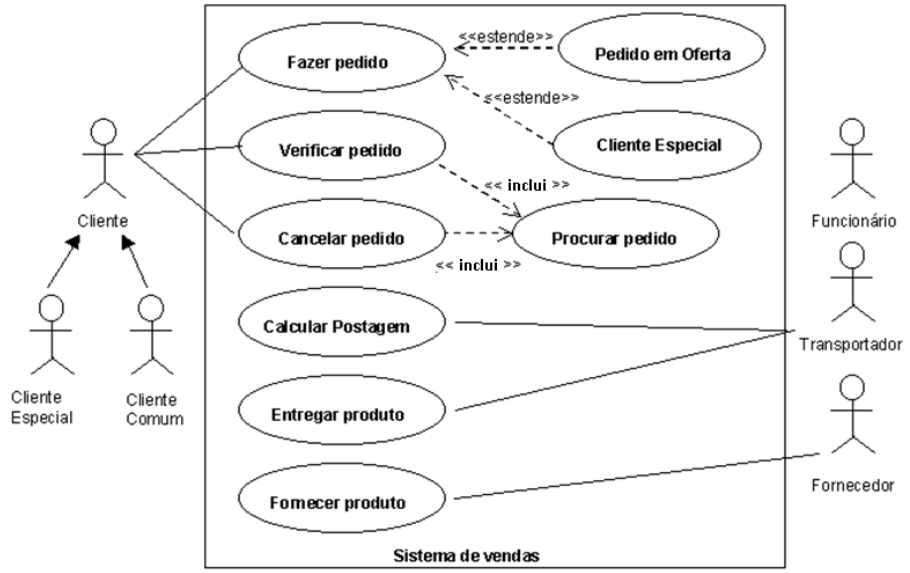


**Figura 23 - Diagrama de Caso de Uso**

### 3.2.2. DIAGRAMA DE ATIVIDADE

Descreve lógicas de procedimento, processos de negócio e fluxos de trabalho. Permite que seja mostrado que entidade é responsável por cada ação no diagrama, com uso de raias (*swimlanes*), ou seja quem faz o quê.

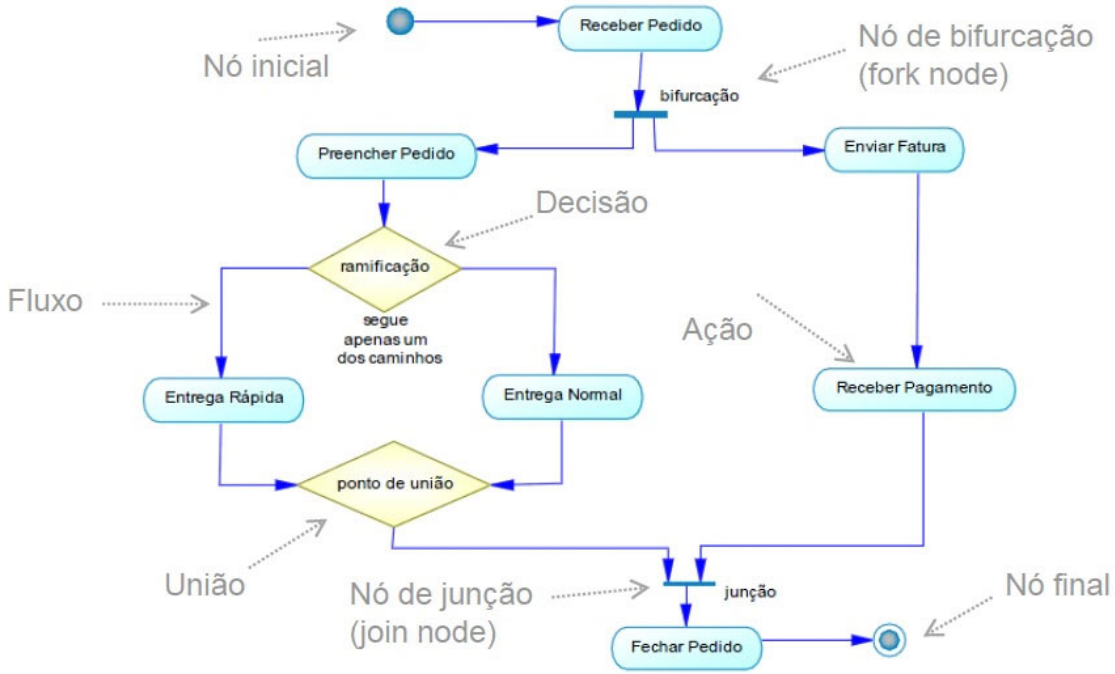


**Figura 24 - Diagrama de Atividade**

Outra forma de representá-lo é exemplificada na figura 25.

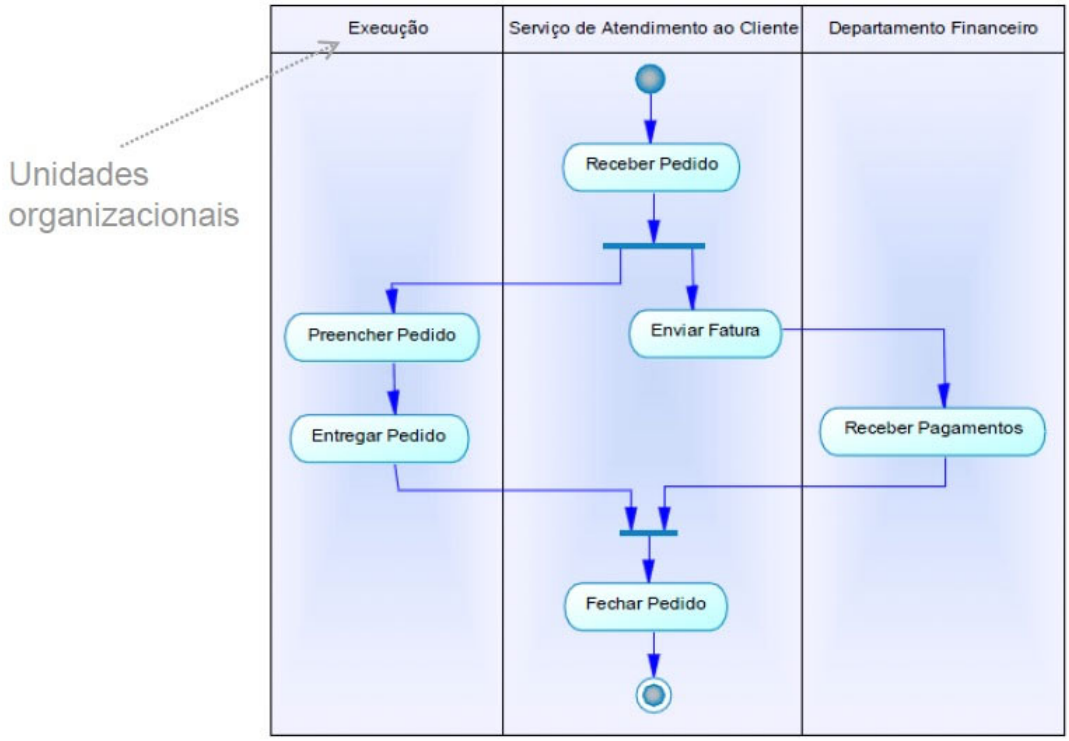


**Figura 25 - Diagrama de Atividade**

### 3.2.3. DIAGRAMA DE MÁQUINA DE ESTADO

Mostra os vários estados possíveis por quais um objeto pode passar. Ao longo do tempo um objeto muda de estado quando acontece algum evento interno ou externo ao sistema. Através da análise das transições entre os estados, pode-se prever todas as possíveis operações realizadas, em função de eventos que podem ocorrer.

#### 3.2.3.1. Elementos do Diagrama de Máquina de Estado

- Estados: Situações na vida de um objeto na qual ele satisfaz uma condição ou realiza alguma atividade;
- Transições: Estados são associados através de transições; Estas transições têm eventos associados. Sua sintaxe é:

| **evento [condição]/ação** |
|---|

- Ações: Ao passar de um estado para o outro o objeto pode realizar ações;
- Atividades: Executadas durante um estado.

Na figura 26 um escalonamento de processo é utilizado para exemplificar este diagrama.

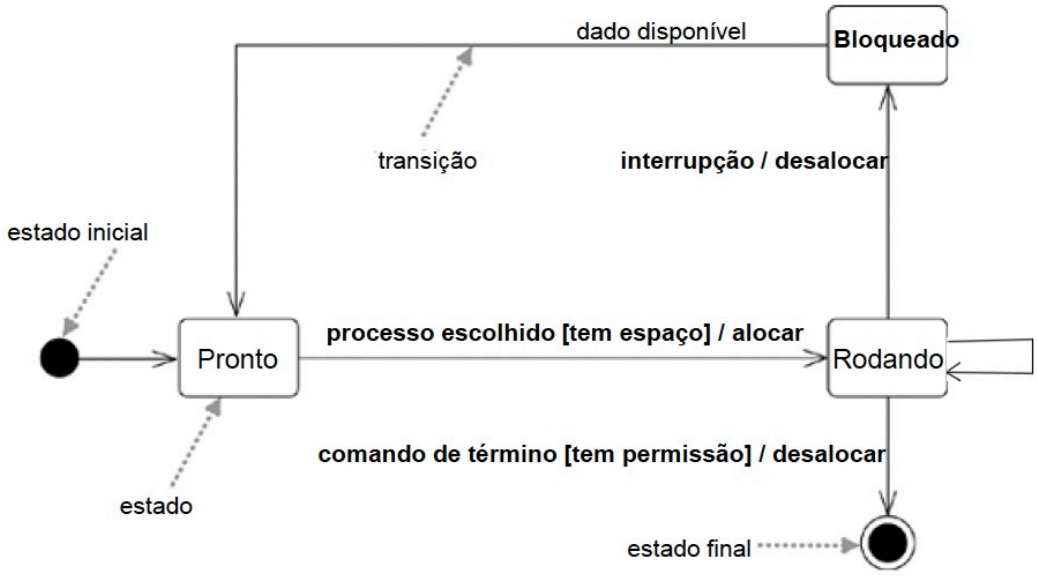


**Figura 26 - Diagrama de Máquina de Estado**

Outro exemplo é a demonstração de estados aninhados e concorrentes na figura 27.

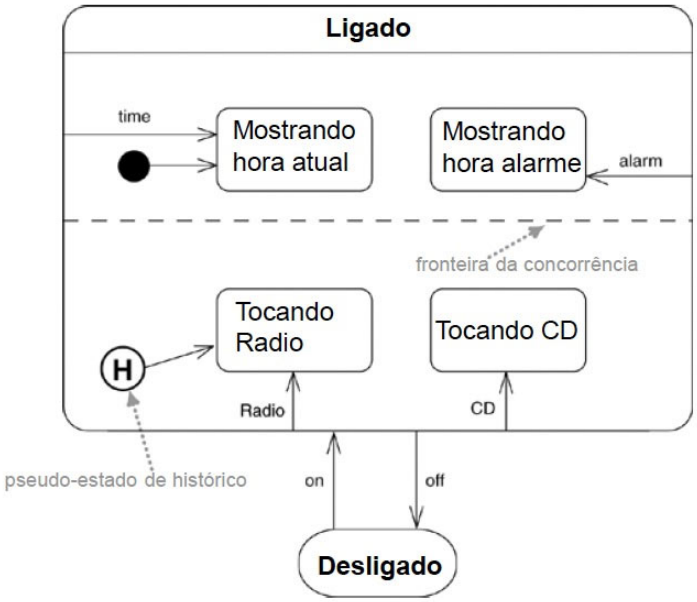


**Figura 27 - Diagrama de Máquina de Estado**

### 3.2.4. DIAGRAMA DE INTERAÇÃO

Diagramas de Interação são modelos que descrevem como o grupo de objetos colaboram em um determinado comportamento. Um diagrama de interação captura o comportamento entre objetos dentro um único caso de uso.

#### 3.2.4.1. Diagrama de Sequência

Captura o comportamento de um determinado cenário. Exibe os objetos e as mensagens trocadas entre eles e enfatiza a ordem temporal das mensagens. É o diagrama mais utilizado na etapa de Projeto Orientado a Objeto.

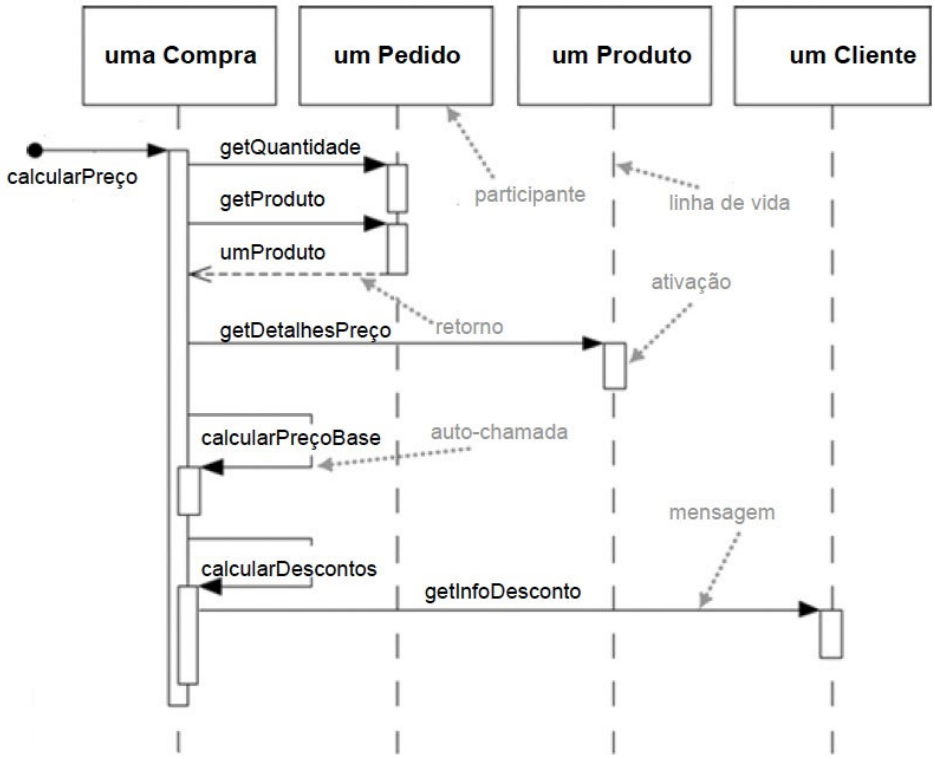


**Figura 28 - Diagrama de Sequência**

Outro exemplo de diagrama de sequência é exibido na figura 29 demonstrando a criação e destruição de objetos.

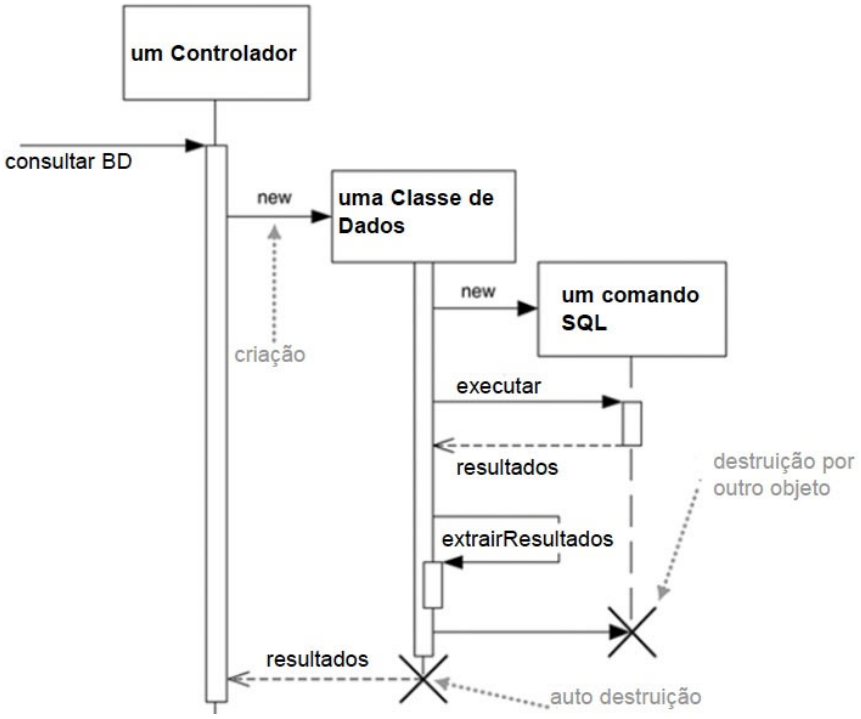


**Figura 29 - Diagrama de Sequência**

3.2.4.2. Diagrama de Comunicação

Captura o comportamento de um determinado cenário. Mostra os objetos e as mensagens trocadas entre eles e enfatiza a ordem estrutural das mensagens (relacionamentos entre objetos). É equivalente ao diagrama de sequência.

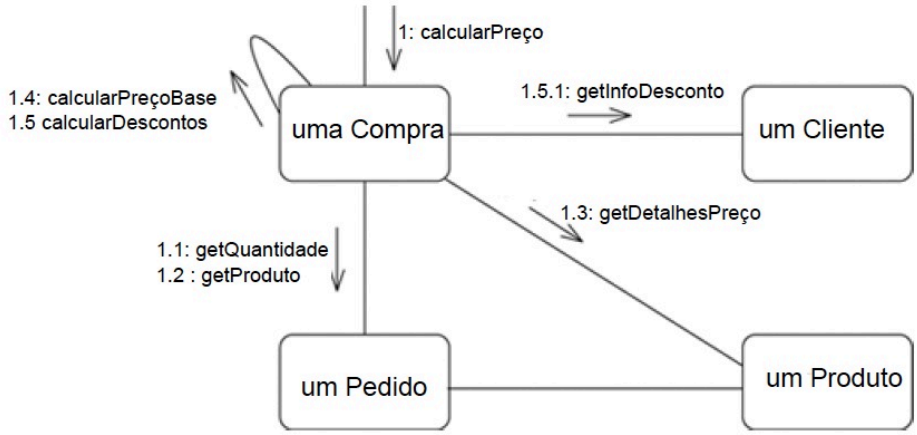


**Figura 30 - Diagrama de Comunicação**

3.2.4.3. Diagrama de Tempo

Captura o comportamento de objetos ao longo do tempo e a duração na qual eles permanecem em determinados estados. O foco se dá nas restrições de tempo das interações. É considerado uma mistura entre o diagrama de sequência e o diagrama de máquina de estados.

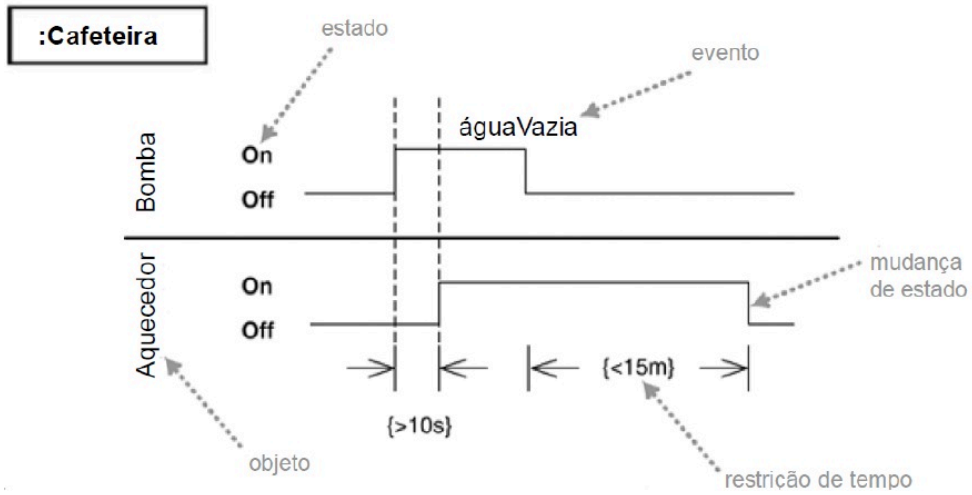


**Figura 31 - Diagrama de Tempo**

3.2.4.4. Diagrama de Interação Geral

Fornece uma visão geral do controle de fluxo entre objetos. É uma mistura entre diagramas de sequência e diagramas de atividade.

No exemplo da figura 32, se o Cliente for externo, os dados são buscados de um XML. Se for interno, os dados são buscados de um banco de dados. A sequência destes dois fluxos é detalhada. Ao final, é gerado um relatório.

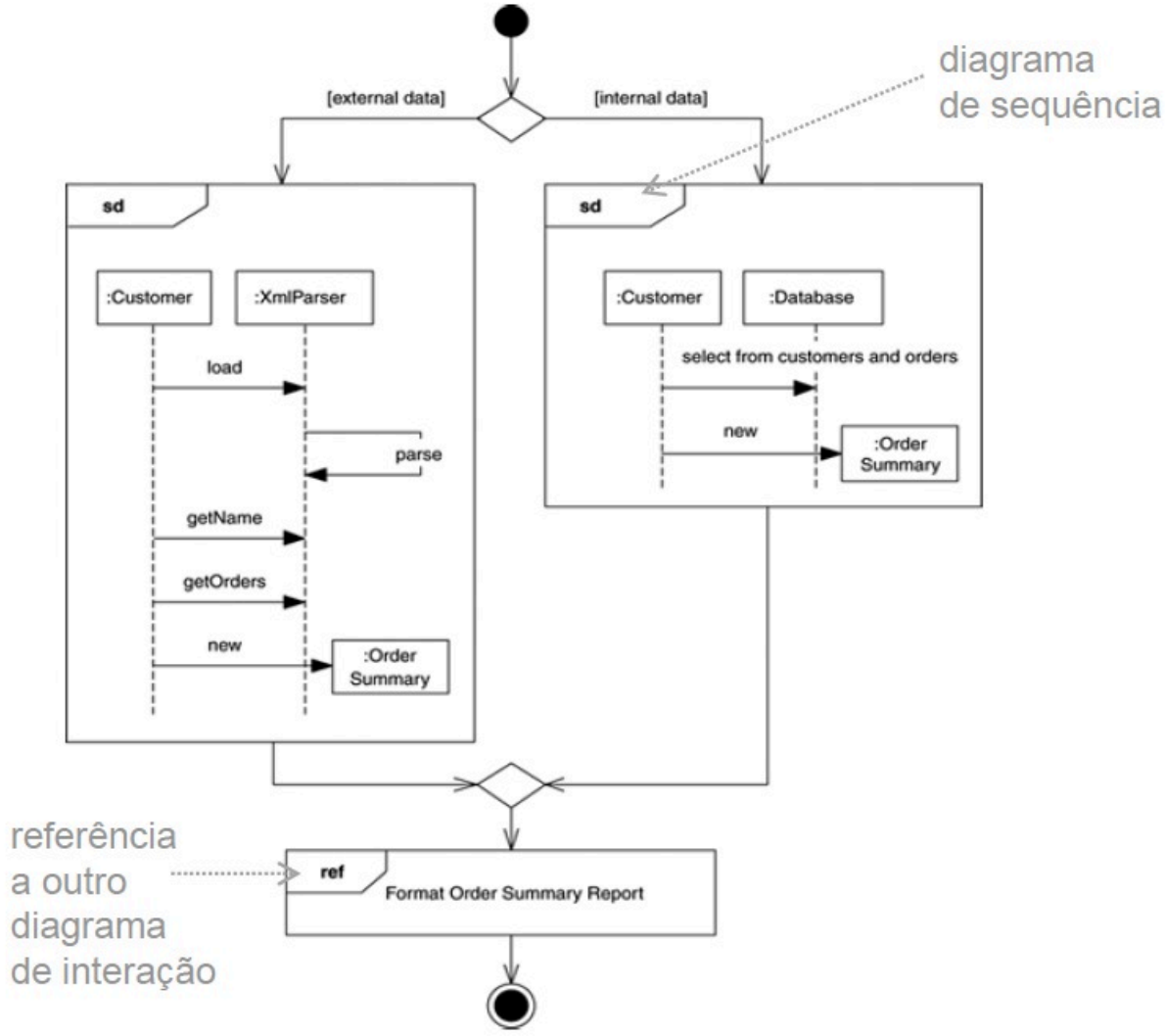


**Figura 32 - Diagrama de Integração Geral**

### 3.3. *Object Constraint Language* (Linguagem para Especificação de Restrições em Objetos)

É uma linguagem declarativa para descrever as regras que se aplicam aos modelos UML desenvolvida na IBM parte do padrão UML.

A linguagem possibilita o uso de expressões de restrições em um modelo orientado a objeto que não possam ser especificadas através dos diagramas. Por fornecer expressões livres das ambiguidades das linguagens naturais e menos difíceis que os métodos formais tradicionais, complementa os modelos UML.

Uma restrição (*constraint*) atua sobre um ou mais valores de um modelo orientado a objetos e apresenta as seguintes vantagens:

- Modelos mais completos, consistentes e precisos;
- Comunicação sem ambiguidade;

- Sintaxe e semântica formais.

Para melhor compreender o emprego da linguagem observe o exemplos que trata das regras (restrições) de um sistema de Universidade.

- A avaliação de supervisores acadêmicos deve ser maior que a nota dos seus supervisionados;
- A bolsa escolar dos alunos depende da sua avaliação acadêmica.

Estas regras podem ser escritas em *Object Constraint Language* e serem transformadas em código, scripts de bancos de dados ou outros modelos.

## 4. REFERÊNCIA BIBLIOGRÁFICA

PEDROSA, Fernando. Material Instrucional. 2013.

GUEDES, Gilleanes T. A. **UML 2 : uma abordagem prática**. 2ª edição. Editora Novatec. São Paulo, 2011.

UML 2.5 Diagrams Overview. Disponível em < http://www.uml-diagrams.org/uml-25-diagrams.html>. Acesso em 10 de abril de 2014.