Engenharia de Software

Capítulo 9

Análise e Projeto Orientados a Objetos

1. Introdução

Existem muitas definições para o que se chama, em desenvolvimento de software, de Orientação a Objetos. Estudando a bibliografia da área, observa-se que cada autor apresenta a sua visão do que entende por esta abordagem. Aí vão alguns conceitos:

- a orientação a objeto pode ser vista como a abordagem de modelagem e desenvolvimento que facilita a construção de sistemas complexos a partir de componentes individuais;
- o desenvolvimento orientado a objetos é a técnica de construção de software na forma de uma coleção estruturada de implementações de tipos abstratos de dados;
- desenvolvimento de sistemas orientado a objetos é um estilo de desenvolvimento de aplicações onde a encapsulação potencial e real de processos e dados é reconhecida num estagio inicial de desenvolvimento e num alto nível de abstração, com vistas a construir de forma econômica o que imita o mundo real mais fielmente;
- a orientação a objetos é uma forma de organizar o software como uma coleção de objetos discretos que incorporam estrutura de dados e comportamento.

Visando a que cada leitor passe a ter a sua própria definição do que significa Orientação a Objetos, vamos apresentar aqui os principais conceitos relacionados a esta tecnologia.

2. Orientação a objetos: conceitos

2.1. Características de Objetos

Um **objeto** é algo distinguível que contém *atributos* (ou propriedades) e possui um *comportamento*. Cada objeto tem uma identidade e é distinguível de outro mesmo que seus atributos sejam idênticos. Exemplos de objetos: o parágrafo de um documento, a janela num computador, o aluno Pedro neste curso, o carro do João. O conjunto de valores associados às propriedades do objeto definem o estado deste; o comportamento descreve as mudanças do estado do objeto interagindo com o seu mundo externo, através das *operações* realizadas pelo objeto.

As seções que seguem apresentam outras definições relacionadas à tecnologia de objetos que são de extrema importância à compreensão das atividades e metodologias baseadas em objetos.

2.2. Classe

Uma **classe** é o agrupamento de objetos com a mesma estrutura de dados (definida pelos *atributos* ou *propriedades*) e comportamento (*operações*) [RBP91]. Uma classe é uma abstração que descreve as propriedades importantes para uma aplicação e não leva em conta as outras. Exemplos de classes: Parágrafo, Janela, Aluno, Carro. Cada classe descreve um conjunto possivelmente infinito de objetos individuais. Cada objeto é uma *instância* de classe. Cada instância de classe tem seu próprio valor para cada um dos atributos da classe mas compartilha os nomes e as operações com as outras instâncias de classe.

2.3. Orientação a objetos

Ela se caracteriza principalmente pela abstração, encapsulamento, herança e polimorfismo.

2.4. Abstração

A abstração consiste em enfocar os aspectos mais importantes de um objeto (visão externa; o que é e o que ele faz), ignorando suas características internas (visão interna; como ele deve ser implementado).

2.5. Encapsulamento

O encapsulamento é o empacotamento de dados (atributos) e de operações sobre estes (métodos). No caso da orientação a objetos, os dados não podem ser acessados diretamente mas através de mensagens enviadas para as operações. A implementação de um objeto pode ser mudada sem modificar a forma de acessa-lo.

2.6. Herança

A herança consiste no compartilhamento de atributos e operações entre as classes numa relação hierárquica. Este mecanismo permite a uma classe ser gerada a partir de classes já existentes; por exemplo a classe **automóvel** herda da classe **veículo** algumas propriedades e operações. Uma classe pode ser definida de forma abrangente (como no caso do exemplo anterior) e posteriormente refinada em termos de sub-classes e assim sucessivamente. Cada subclasse *herda* todas as propriedades e operações da sua superclasse, (que não precisam ser repetidas) adicionando apenas as suas especificas.

Esta relação entre classes é uma das grandes vantagens de sistemas orientados a objetos por causa da redução de trabalho resultante durante o projeto e a programação destes.

Existem dois tipos de herança:

- herança simples, ondeuma subclasse tem somente uma superclasse;
- herança múltipla, na qual uma subclasse herda simultaneamente de várias superclasses.

2.7. Polimorfismo

O polimorfismo significa que uma mesma operação pode se comportar de forma diferente em classes diferentes. Exemplo de polimorfismo, a operação **calcular o perímetro** que é diferente para as instâncias de classe circulo e polígono ou a operação **mover** diferente para janela de computador e peça de xadrez. Uma operação que tem mais de um método que a implementa é dita **polimórfica**.

Nos sistemas orientados a objetos, o suporte seleciona automaticamente o método que implementa uma operação correto a partir do nome da operação e da classe do objeto no qual esta se operando, da mesma forma que no mundo real onde o objeto real "tem conhecimento" intrínseco do significado da operação a realizar. Essa associação em tempo de execução é chamada de ligação dinâmica (ou "dynamic binding").

3. Desenvolvimento orientado a objetos

O desenvolvimento orientado a objetos diz respeito aos procedimentos de concepção de sistemas a partir dos conceitos básicos anteriores. Ele corresponde às principais fases do ciclo de vida de software: análise, projeto e implementação.Existem várias técnicas para o desenvolvimento orientado a objetos cujas características serão citadas a seguir.

3.1. OMt — Object Modelling technique

A técnica OMT se distingue pela sua divisão em 4 fases: análise, projeto de sistema, projeto de objetos e implementação. Como principais características, destacam-se a separação clara entre análise e projeto, a inclusão de todos os conceitos da orientação a objetos e de alguns específicos do método. A fase de análise é baseada de 3 diagramas relacionados entre si e que representam os modelos de objetos, dinâmico e funcional.

3.2. técnica de Boocx

A técnica Booch é dividida em 3 fases: análise de requisitos, análise de domínios e projeto, com ênfase maior no projeto. Os diagramas seguintes são providos: de classes, de transição de estados, de objetos, temporais, de módulos e de processos.

3.3. técnica de Coad/Yourdon

Esta técnica utiliza um modelo único para todas as fases (OOA, OOD e OOP), o que torna mais simples e compreensível o desenvolvimento.

3.4. técnica de Sxlaer/Mellor

Ela fornece um conjunto integrado de modelos de análise e que depois são traduzidos (Recursive Design) durante o projeto.

3.5. técnica OOSE (Jacobson)

A técnica OOSE é centrada em casos-de-uso (use-cases) e permite durante a análise aprofundar o entendimento de como o sistema deve ser realmente utilizado.

3.6. técnica Fusão

A Fusão corresponde a uma integração das técnicas OMT e de Booch na qual os dois modelos de objeto e de interface visam representar os aspectos estáticos e dinâmicos do problema.

3.6. técnica UML

UML ou *Unified Modeling Language* é uma unificação dos métodos OMT, Booch e OOSE que está sendo submetido a OMG para a padronização.

A técnica OMT será descrita mais detalhadamente nos itens seguintes.

4. MODELAGEM ORIENTADA A OBJETOS

Um modelo é uma abstração que tem como propósito entender um problema antes de solucioná-lo. A partir de modelos, é possível simular e testar sistemas antes de construí-los, facilitar a comunicação com os usuários e os outros membros da equipe de desenvolvimento, visualizar e reduzir a complexidade dos problemas a tratar.

No modelo OMT, temos 3 visões combinadas que permitem representar o sistema:

- um **modelo objeto** para representar os aspectos estáticos, estruturais, de "dados" de um sistema;
- um **modelo dinâmico** para representar os aspectos temporal, comportamental e de "controle" de um sistema;
- um **modelo funcional** para representar os aspectos transformacionais e de "função" de um sistema.

Esses modelos não são independentes, existem interconexões limitadas e explícitas.

4.1. O modelo objeto

O modelo objeto descreve a estrutura de objetos no sistema: sua identidade, suas relações com os outros objetos, seus atributos e suas operações. Este modelo tenta capturar a estrutura estática dos componentes do mundo real que pretende-se representar. O modelo objeto tem uma representação gráfica na forma de diagramas contendo as classes de objetos com seus atributos e operações, e organizados segundo hierarquias e associações com os diagramas de outras classes.

4.1.1. Objetos e classes

Todos os objetos tem uma identidade e são distinguíveis. Eles são instancias de classes de objetos que descrevem grupos de objetos com similaridade nas propriedades (atributos), comportamento (operações), relações com os outros objetos e semântica. A notação gráfica para representar objetos, classes e suas relações é composta de dois tipos de diagramas, mostrados na figura 9.1:

- um **diagrama de classe**, que representa classes de objetos e tem a função de ser um esquema, um padrão ou um "template" para os dados;
- um **diagrama de instância**, utilizado para representar instâncias de classes e tem o objetivo de descrever como um conjunto particular de objetos se relaciona com outro.

O atributo é colocado na segunda parte da caixa (ver figura 9.2). Cada nome de atributo pode ser seguido por detalhes opcionais tais como tipo (precedido por "**:**") e valor default (precedido de "**=**"). Identificadores (explícitos) de objeto não são necessários no modelo objeto pois cada objeto já tem a sua propria identidade (a partir de seus valores).

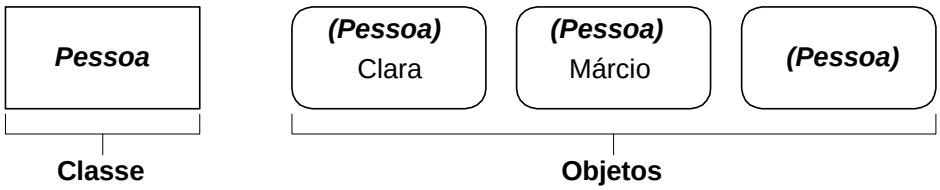

Figura 9.1 – Ilustração de Classes e Objetos.

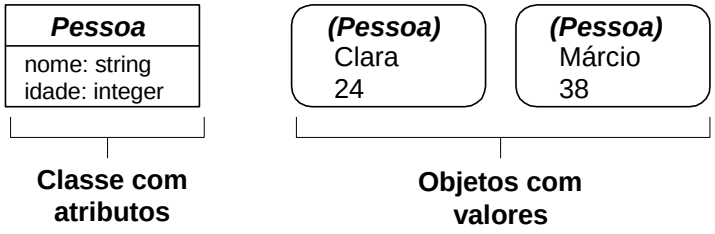

Figura 9.2 – Ilustração de Atributos e Valores.

Uma operação pode ser aplicada a ou por objetos numa classe. A mesma operação pode se aplicar a várias classes diferentes (polimorfismo). Um método é a implementação de uma operação para uma classe. Quando uma operação tem métodos em várias classes, eles devem ter a mesma assinatura (em número e tipos de argumentos). As operações se encontram na terceira parte da caixa, como ilustrado na figura 9.3. Cada operação pode ser seguida de detalhes opcionais tais como lista de argumentos (colocada entre parênteses após o nome, separados por "**,**") e tipo de resultado (precedido por "**:**"). A notação generalizada para classes de objetos se encontra na figura 9.4.

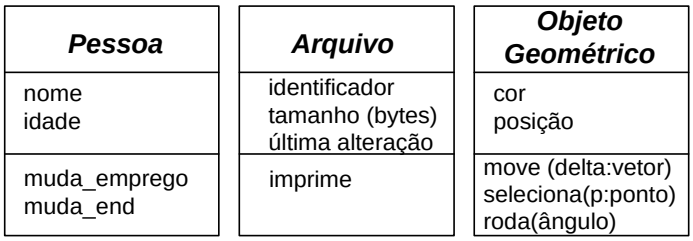

Figura 9.3 – Operações com Objetos.

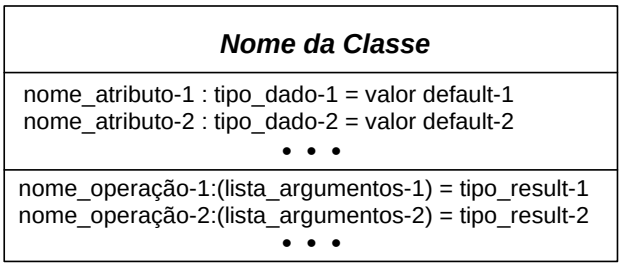

Figura 9.4 – Generalização da notação de modelagem de objetos.

4.1.2. Ligações e associações

A *ligação* ou *link* é uma conexão física ou conceitual entre instâncias de objetos. Por exemplo: Vitório Mazzola **trabalha para** UFSC. Uma ligação é uma túpla matemática correspondente a uma lista ordenada de instâncias de objetos. Uma ligação é uma instância de uma associação.

Uma *associação* ou *association* descreve um grupo de ligações com estrutura e semântica comuns. Por exemplo: uma pessoa **trabalha para** uma instituição. Todas as ligações de uma associação conectam objetos das mesmas classes. O conceito de associação é similar aquele usado na área de base de dados.

Associações e ligações são descritas por verbos. Associações são inerentemente bidirecionais e podem ser lidas nos dois sentidos. Por exemplo: num primeiro sentido pode-se ler Vitório Mazzola **trabalha para** UFSC e no sentido inverso, UFSC **emprega** Vitório Mazzola). Associações são muitas vezes implementadas em linguagens de programação como apontadores (atributo de um objeto que contêm referencia explícita a um outro) de um objeto para outro. Por exemplo: objeto *Pessoa* pode conter atributo *empregado* que aponta para um conjunto de objetos *Instituição*.

Uma ligação mostra a relação entre 2 ou mais objetos. A figura 9.5 mostra uma associação um-a-um e as ligações correspondentes. Observa-se, na notação OMT, que a associação corresponde a uma linha entre classes e que os nomes de associação são em itálico.

As associações podem ser binárias, terçárias ou de ordem maior. De fato, a maior parte é binária ou qualificada (i.e., é uma forma especial de terçária que será comentado a seguir). A simbologia OMT para terçária ou n-ário é um losango com linhas conectando este as classes relacionadas. O nome da associação é escrito dentre do losango; entretanto, os nomes de associação são opcionais. A figura 9.6 mostra exemplos de associações e ligações terçárias.

A **multiplicidade** especifica quantas instâncias de uma classe podem se relacionar a uma única instância de classe associada. A multiplicidade restringe o número de objetos relacionados. A multiplicidade depende de hipóteses e de como se define os limites do problema. Requisitos vagos tornam a multiplicidade incerta. Não é possível de determinar muito cedo a multiplicidade no processo de desenvolvimento de software; num primeiro tempo, determina-se objetos, classes, ligações e associações e depois decide-se a respeito de multiplicidade. A distinção mais importante é entre “um” e “vários”; entretanto existem símbolos especiais para “exatamente um”, “um ou mais”, “intervalos”, “”zero ou mais”, “zero ou um”, conforme mostra a figura 9.7.

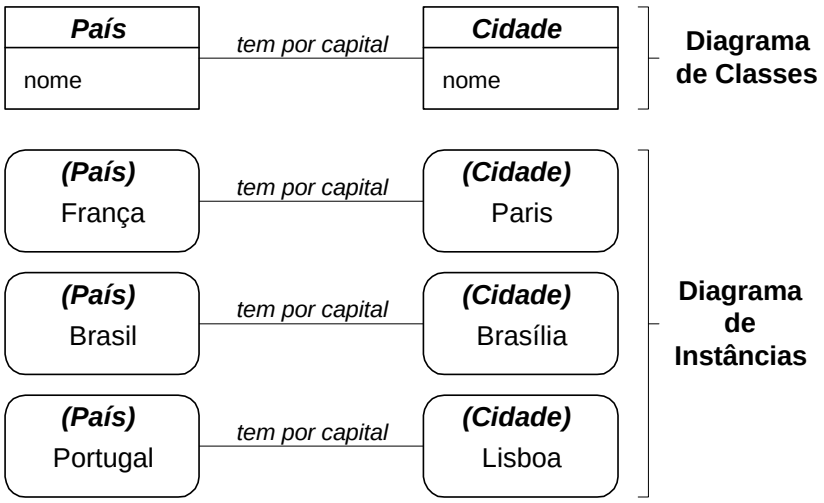

Figura 9.5 – Associação um-para-um e links.

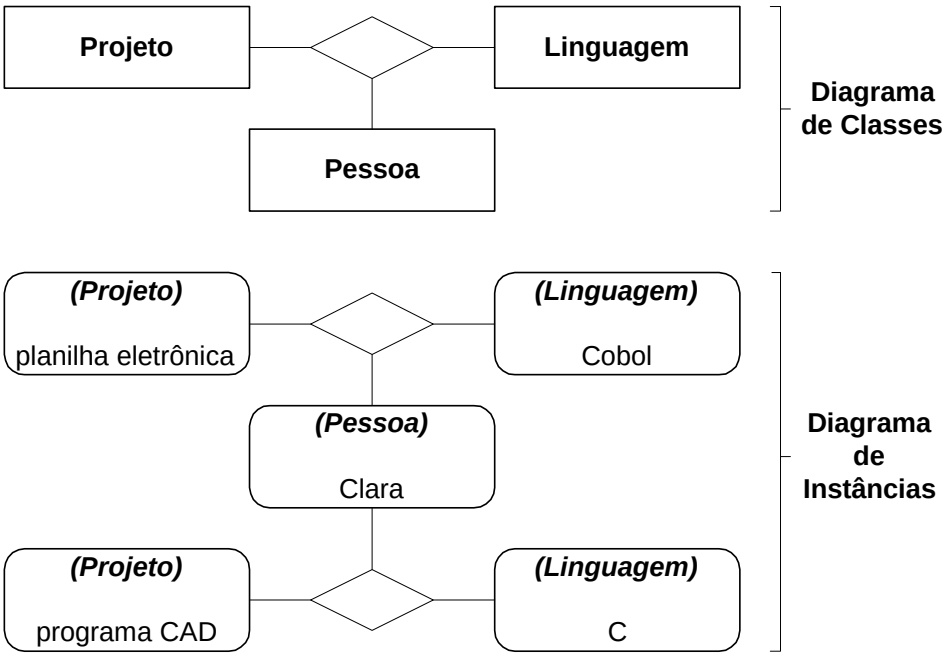

Figura 9.6 – Associação ternária e links.

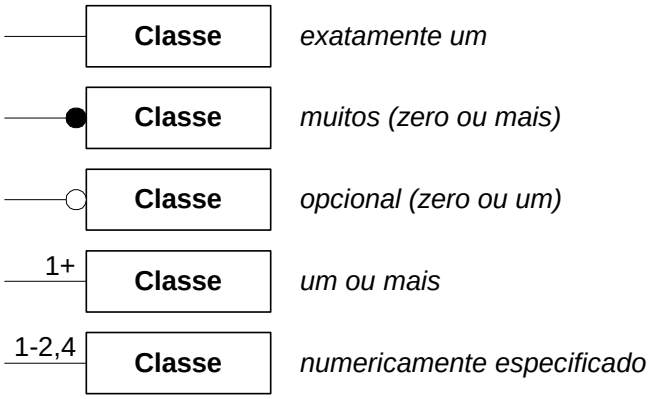

Figura 9.7 – Notação para associações múltiplas.

4.1.3. Ligações e associações avançadas

Atributo de ligação. Similarmente ao caso da classe onde um atributo é uma propriedade dos objetos desta, um atributo de ligação é uma propriedade das ligações numa associação. A figura 9.8 apresenta o caso onde *permissão-de-acesso* é um atributo da associação *Acessível-por*. Cada atributo de ligação tem um valor para cada ligação, conforme pode ser visto nesta figura ainda. É ainda possível ter atributos de ligações para associações vários-a-um conforme visto na figura 9.9 e também para ligações terçárias. Apesar de poder substituir os atributos de ligação por atributos de objetos da associação, é preferível manter atributos de ligação.

Modelagem de associação como classe. Às vezes, é útil modelar uma associação como classe. Cada ligação se torna uma instância da classe.

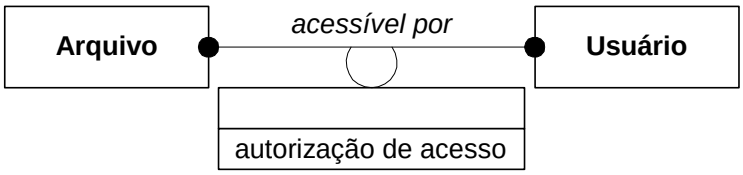

Figura 9.8 – Atributo de link para associação muitos-para-muitos.

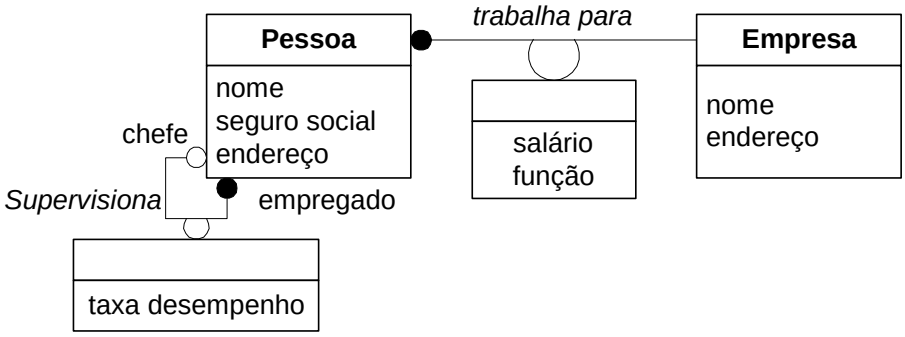

Figura 9.9 – Atributos de link para associação um para vários (papéis).

A figura 9.10 mostra um exemplo disto. Esta opção é particularmente útil quando ligações podem participar em associações com outros objetos ou quando ligações são sujeitas a operações.

Papel. Um *papel* ou *role* é uma extremidade de uma associação. Uma associação binária tem dois papeis, um em cada extremidade, cada um deles podendo ter seu nome. Cada papel numa associação binária identifica um objeto ou um conjunto de objetos associado com um objeto na outra extremidade. Nomes de papel são necessários em associações entre objetos da mesma classe, conforma consta na figura 9.9.

Ordenamento. Usualmente, do lado de uma associação "vários", os objetos não têm ordem explícita e podem ser vistos como um conjunto. Entretanto, às vezes, os objetos são explicitamente ordenados, o que é representado por *{ordered}* perto da indicação de multiplicidade.

Qualificação. Uma associação qualificada relaciona duas classes de objetos e um qualificador. O qualificador é um atributo especial que reduz a multiplicidade efetiva de uma associação, distinguindo objetos dentro do conjunto na extremidade "vários" da associação.

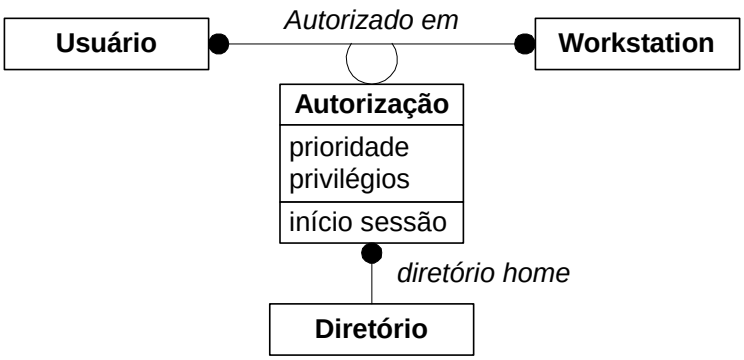

Figura 9.10 – Modelando uma associação como Classe.

4.1.4. Agregação

A agregação é uma relação “parte-todo” ou “uma-parte-de” na qual os objetos representando os componentes de algo são associados com um objeto representando a junção deles (“*assembly*”). Algumas propriedades da classe junção se propagam às classes componentes, possivelmente com alguma modificação local.

A agregação é definida como uma relação entre uma classe junção e uma classe componente. A agregação é então uma forma especial de associação. A simbologia da agregação é similar a da associação (linha), exceto pelo fato de um pequeno losango indicar o lado da junção da relação, conforme a figura 9.11.

4.1.5. Generalização e herança

Generalizações e herança são abstrações poderosas que permitem compartilhar similaridades entre as classes, apesar de preservar suas diferenças.

Generalização é uma relação do tipo “é um” entre uma classe e uma ou mais versões refinadas. A classe que está sendo refinada é uma superclasse e a classe refinada é uma subclasse. Por exemplo, **máquina de comando númerico** é uma superclasse de **torno**, **fresadora**. Atributos e operações comuns a um grupo de subclasses são reagrupados na superclasse e compartilhado por cada subclasse. Diz-se que cada subclasse herda as características de sua superclasse.

A notação para a generalização é um triângulo conectando uma superclasse a suas subclasses, conforme visto na figura 9.12. As palavras próximas ao triângulo no diagrama são discriminadores (i.e. atributos de um tipo enumeração que indica qual propriedade de um objeto está sendo abstraído por uma relação de generalização particular). Os discriminadores estão inerentemente em correspondência um-a-um com as subclasses da generalização.

É possível relação de herança em vários níveis. A herança em 2 ou 3 níveis (até 5 em alguns casos) de profundidade é aceitável; já a mais de 10 níveis é excessiva.

A generalização facilita a modelagem a partir de classes estruturadas e da captura do que é similar e do que é diferente nas classes. A herança de operação é de grande ajuda durante a implementação para facilitar a reutilização de código.

Utiliza-se a generalização para se referenciar à relação entre classes, enquanto a herança diz respeito ao mecanismo de compartilhar atributos e operações usando a relação de generalização. Generalização e especialização são dois pontos de vista da mesma relação, no primeiro caso vista a partir da superclasse e no segundo da subclasse: a superclasse generaliza a subclasse e a subclasse refina ou especializa a superclasse.

4.1.6. Módulo

Um módulo é uma construção lógica para reagrupar classes, associações e generalizações. Um módulo captura uma perspectiva ou uma vista de uma situação, como no caso de um **prédio**, existem várias visões correspondentes a **planta elétrica**, **hidráulica**, **de calefação**. Um modelo objeto consiste de um ou vários módulos. A noção de módulo força a particionar um modelo objeto em peças gerenciaveis. Uma mesma classe pode ser referenciada em módulos diferentes. De fato, referenciando a mesma classe em múltiplos módulos é o mecanismo para interligar módulos entre sí.

Figura 9.11 – Agregação.

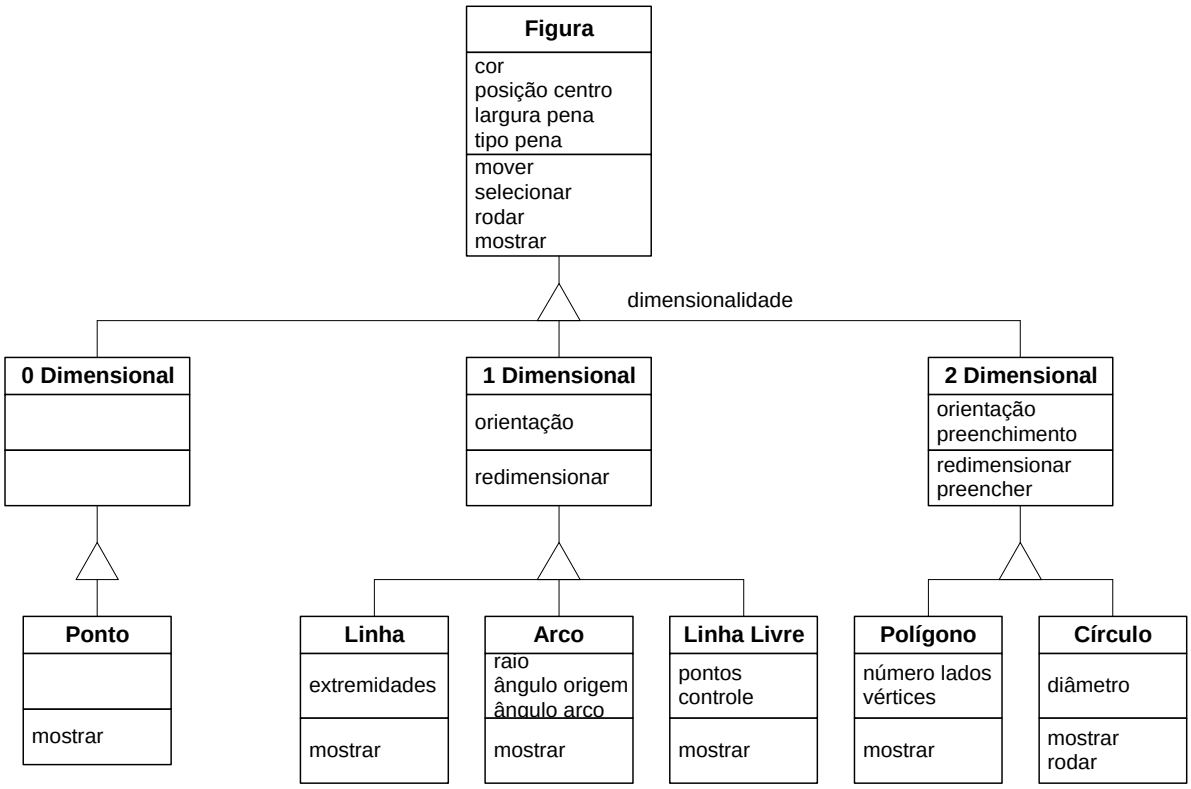

Figura 9.12 – Herança aplicada a figuras gráficas.

4.2. O modelo dinâmico

O modelo dinâmico descreve os aspectos do sistema que dizem respeito ao tempo e à seqüência de eventos (operações). Este modelo tenta capturar o controle, aspecto de um sistema que descreve as seqüências de operação que ocorrem em resposta a estímulos externos, sem levar em conta o que as operações fazem, quem as ativa e como são implementadas. Os conceitos utilizados nesta modelagem dinâmica são os de eventos que representam os estímulos externos e de estados que representam os valores de objetos.

A representação gráfica é um diagrama de estados que representa os estados e a seqüência de eventos permitidos num sistema para uma classe de objetos. Os estados e eventos podem ainda serem organizados de forma hierárquica e representados num diagrama de estados estruturado.

4.2.1. Eventos e estados

Um **estado** é caracterizado pelos valores dos atributos e pelas ligações mantidas por um objeto. Um **evento** corresponde a um estimulo individual de um objeto a um outro. O **diagrama de estados** representa o modelo de eventos, estados e transições de estado para um a classe dada. O **modelo dinâmico** consiste de vários diagramas de estados, um para cada classe com comportamento dinâmico importante; ele mostra o modelo de atividade para um sistema completo. Cada maquina de estados se executa de forma concorrente e pode mudar de estado independentemente. Os diagramas de estado para as várias classes combinam num modelo dinâmico único através de eventos compartilhados.

Um **evento** é algo que ocorre num instante de tempo e que não tem duração. Um evento pode preceder ou seguir outro evento ou pode não ter relação entre eventos (neste caso, são ditos concorrentes). Cada evento é uma ocorrência única; entretanto é possível reagrupa-los em classes de eventos e dar a cada uma delas um nome que indica uma estrutura e um comportamento comuns. Alguns eventos são simples sinais mas muito

outros tem atributos indicando a informação que eles transportam. O tempo no qual o evento ocorre é um atributo implícito de todos os eventos. Um evento transporta a informação de um objeto a outro; os valores de dados transportados por um evento são seus atributos. Os eventos incluem as condições de erro e as ocorrências normais.

Um **cenário** é uma seqüência de eventos que ocorre durante uma execução particular de um sistema. Ele pode incluir todos os eventos do sistema ou apenas eventos gerados por certos objetos no sistema. A seqüência de eventos e os objetos que trocam eventos podem ser mostrados juntos num diagrama de **rastro de eventos**. A figura 9.13 mostra o cenário e o rastro de eventos para uma chamada telefônica.

Um **estado** é uma abstração dos valores dos atributos e das ligações de um objeto. Um estado especifica a resposta do objeto à eventos de entrada. A resposta de um objeto à um evento pode incluir uma ação ou uma mudança de estado pelo objeto.
Um estado tem uma duração; ele ocupa um intervalo de tempo entre dois eventos. Na definição de estados, pode se ignorar atributes que não afetam o comportamento do objeto. O estado é caracterizado por um nome e uma descrição contendo a seqüência de eventos que leva ao estado, a condição que o caracteriza e os eventos aceitos neste estado com a ação que ocorre e o estado futuro. O estado pode incluir os valores de suas ligações.

O **diagrama de estados** relaciona estados e eventos. A mudança de estado causada por um evento é chamada de **transição**. A figura 9.14 mostra o diagrama de estados de uma linha telefônica. Os diagramas de estado podem representar ciclos de vida uma-vez (com um estado inicial e um estado final) que representam objetos com vida finita ou malhas continuas como na figura 9.14. Um modelo dinâmico é uma coleção de diagramas de estado que interagem entre si através de eventos compartilhados.

Uma **condição** é uma função booleana de valores objetos. Condições podem serem usados como guardas nas transições, sendo que uma transição guardada dispara quando o evento ocorre e que a condição de guarda é verdadeira.

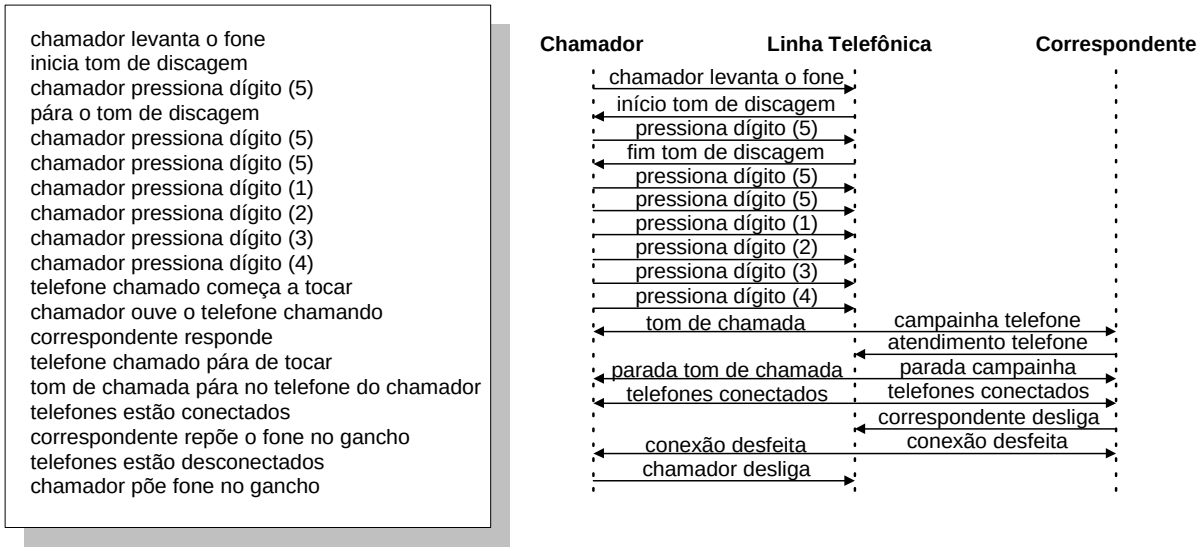

Figura 9.13 – Cenário e traço de eventos para uma chamada telefônica.

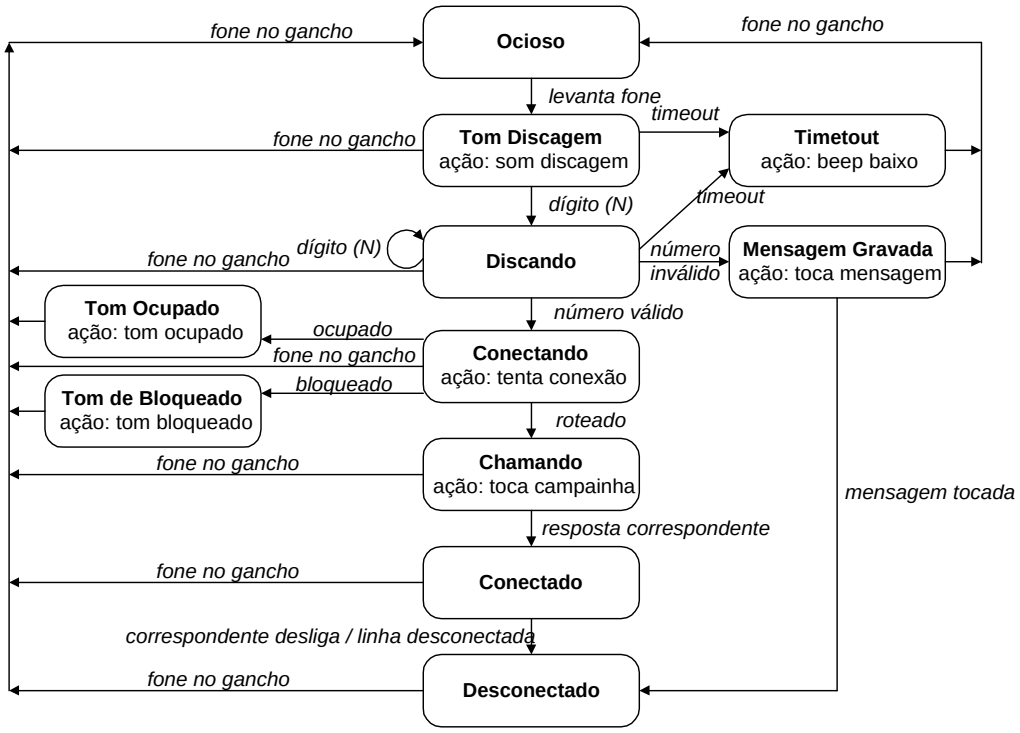

Figura 9.14 – Diagrama de estados de uma linha telefônica.

4.2.2. Operações

Uma descrição do comportamento de um objeto deve especificar o que o objeto faz em resposta a eventos. Operações associadas à estados ou transições são realizadas em resposta aos estados correspondentes ou a eventos.

Uma **atividade** é uma operação que leva tempo para se completar. Ela é associada a um estado. A notação "*do: A*" dentro de um caixa de estado indica que a atividade A inicia na entrada no estado e para na saída.

Uma **ação** é uma operação instantâneo e é associada a um evento. Uma ação representa uma operação cujo a duração é pequena comparada com a resolução do diagrama de estados. Ações podem também representar operações de controle interno. A notação para uma ação numa transição é um "/" seguido do nome da ação, após o evento que a causa. A figura 9.15 representa os diagramas de estados com operações.

4.2.3. Diagramas de estado aninhados

Os diagramas de estado podem ser estruturados para permitir descrições concisas de sistemas complexos. Através de uma **generalização**, é possível expandir, num nível mais baixo, adicionando detalhes, uma atividade descrita num nível superior. É possível organizar estados e eventos em hierarquias com herança de estrutura e comportamento comuns, como no caso da herança de atributos e operações em classes de objetos.

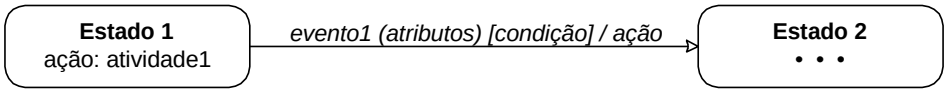

Figura 9.15 – Diagrama de estados (generalização).

A **agregação** permite quebrar um estado em componentes ortogonais, com interação limitada entre eles, da mesma foram como se tem uma hierarquia de agregação de objetos; a agregação é equivalente à concorrência de estados (estados concorrentes correspondem geralmente a agregações de objetos).

Uma **atividade** num estado pode ser expandida como um diagrama de estados de baixo-nível, cada estado representando um degrau da atividade. Atividades aninhadas são diagramas de estados *uma-vez* com transições de entrada e saída, de forma similar a subrotinas. Um diagrama de estados aninhado é uma forma de generalização (relação do tipo "*or*") sobre os estados. Os estados num diagrama aninhado são todos refinamentos (sub-estados) de um estado (super-estado) de um diagrama de alto-nível; a simbologia utilizada corresponde a uma caixa arredondada representando o super-estado e contendo todos seus sub-estados.

Eventos podem também ser expandidos em diagramas de estados subordinados. Os eventos podem ser organizados numa hierarquia de generalização com herança dos atributos de eventos. A hierarquia de eventos permite que diferentes níveis de abstração sejam usados em diferentes lugares num modelo.

4.2.4. Concorrência

Um modelo dinâmico descreve um conjunto de objetos concorrentes, cada um com seu próprio diagrama de estados. Os objetos num sistema são inerentemente concorrentes e podem mudar de estados de forma independente. O estado do sistema total é o resultado dos estados de todos os seus objetos. Um diagrama de estados no caso da junção ("*assembly*") de objetos através de agregação (relação "*and*") é uma coleção de diagrama de estados, concorrentes, um para cada componente. Entretanto, em vários casos, os estados dos componentes interagem; transições guardadas para um objeto podem depender do estado de um outro objeto.

A concorrência pode também ocorrer dentro do estado de um único objeto, quando o objeto pode ser particionado em subconjuntos de atributos ou ligações, cada um com seu próprio sub-diagrama; na notação adotada, os sub-diagramas são separados por linhas pontilhadas.

4.2.5. Modelagem dinâmica avançada

Ações de entrada e saída. Como alternativa a mostra ações nas transições, pode-se associar ações com a entrada ou a saída de um estado. O nome da ação de entrada seguindo *entry/* na caixa representando o estado e o nome da ação de saída seguindo *exit/* nesta são as notações adotadas.

Ações internas. Um evento pode forçar uma ação a ser realizada sem causar uma mudança de estado; o nome do evento é escrito na caixa representando o estado, seguido por um "/" e o nome da ação. A figura 9.16 representa a notação utilizada para os diagramas de estado.

Transição automática. Existe a possibilidade de representar transições automáticas que disparam quando a atividade associada ao estado fonte é completado; é representado por uma seta sem nome de evento. No caso do estado não ter atividade associada, a transição sem nome dispara logo após ter entrado no estado. Chama-se este tipo de transição automática de lambda (λ) transição.

Eventos de envio. Um objeto pode realizar uma ação de enviar um evento a um outro objeto. Um sistema de objetos interage entre si. Na notação OMT, pode se utilizar a palavra *Send* ou uma linha pontilhada que leva da transição a um objeto, conforme indicado na figura 9.16.

Sincronização de atividades concorrentes. Quando um objeto deve realizar duas ou mais atividades de forma concorrente, é necessário dividir ("split") o controle em partes concorrentes e depois juntá-las ("merge"), completando as atividades antes da progressão do objeto para um próximo estado. Uma seta que se ramifica para a divisão e uma seta com a extremidade ramificada para a junção são as representações na notação OMT.

4.3. O modelo funcional

O modelo funcional descreve os aspectos do sistema que dizem respeito com as transformações de valores: funções, mapeamentos, restrições e dependências funcionais. Este modelo captura *o que* o sistema faz sem levar em conta o *como* e o *quando* ele faz.

O modelo funcional é representado por vários diagramas de fluxo de dados (DFDs) que mostram as dependências entre valores e o calculo de valores de saída a partir de valores de entrada e de funções. O modelo funcional inclui também as restrições entre valores no modelo objeto.

4.3.1. Diagramas de Fluxo de Dados

O modelo funcional consiste de múltiplos diagramas de fluxo de dados que especificam o significado de operações e restrições. Um diagrama de fluxo de dados (DFD) mostra a relação funcional dos valores calculados pelo sistema, incluindo valores de entrada, saída e armazenamento de dados internos.

Um DFD contem *processos* que transformam dados, *fluxos de dados* que movimenta dados, objetos *atores* que produzem e consomem dados, objetos *armazenamento de dados* que estocam os dados.

Processos. Um processo transforma valores de dados. Um processo é implementado como um método de uma operação de uma classe de objetos. O objeto alvo é usualmente um dos fluxos de entrada, especialmente se a mesma classe de objeto é também um fluxo de saída. Em alguns casos, o objeto alvo é implícito.

Fluxos de dados. Um fluxo de dados conecta a saída de um objeto ou de um processo a entrada de um outro objeto ou processo Ele representa um valor de daos intermediário num calculo. O valor permanece sem mudança no fluxo de dados.

Atores. Um ator é um objeto ativo que conduz o diagrama de fluxo de dados produzindo ou consumindo valores. Atores são ligados as entradas e saídas de um diagrama de fluxo de dados. Eles podem serem vistos como fontes e receptores de dados.

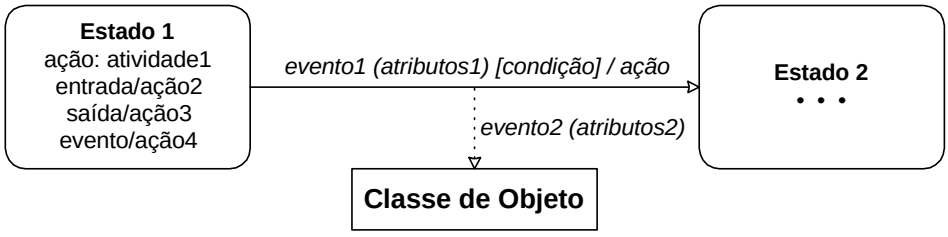

Figura 9.16 –Ilustração da notação extendida dos diagramas de estado.

Armazenadores de dados. Um armazenador de dados é um objeto passivo do diagrama de fluxo de dados que armazena dados para uma cesso futuro.> como no caso de um ator, um armazenador não gera operações sobre ele mesmo mas simplesmente responde a pedidos para armazenar e acessar dados. O acesso pode ser feito em ordem diferente do

armazenamento. Destaca-se que atores e armazenadores de dados são objetos que se diferenciam pelo seu comportamento e uso; atores podem ainda serem implementados como dispositivos externos e armazenadores como arquivos.

Diagramas de fluxo de dados aninhados. Um DFD é particularmente útil para mostrar a funcionalidade de alto-nível de um sistema e sua quebra em unidades funcionais menores. Um processo pode ser expandido num outro DFD no qual as entradas e saídas do processos o são também no novo diagrama. Eventualmente, o anhinhamento de diagramas termina com funções simples que devem ser especificadas como operações.

Fluxos de controle. Um DFD não mostra quais caminhos são executados e em que ordem. Decisões e sequenciamento são questões de controle que fazem parte do modelo dinâmico. As vezes, pode ser útil introduzir o fluxo de controle no DFD. O fluxo de controle é uma variável booleana que indica quando um processo pode ser realizado; o fluxo de controle (representado por uma linha pontilhada no DFD que vai de um processo que gera uma variável booleana ao processo a ser controlado) não é um valor de entrada para o processo ele mesmo.

4.3.2. Especificando operações

Processos em DFD devem eventualmente ser implementados como operações sobre objetos. Para cada nível baixo, um processo atômico é uma operação. Processos de nível elevado podem também ser considerados operações. Apesar que uma implementação possa ser organizada de forma diferente da que o DFD representa por causa de otimização. Cada operação pode ser especificada de várias formas como por exemplo: funções matemáticas, tabelas de valores de entrada e saída, equações especificando saída em termos de entrada, pré e pós condições, tabelas de decisão, pseudo-código e linguagem natural.

4.4. Relações entre modelos

Cada modelo descreve um aspecto do sistema mas contem referencias aos outros modelos. O modelo objeto descreve a estrutura de dados sobre a qual os modelos dinâmico e funcional operam. As operações no modelo objeto correspondem aos eventos no modelo dinâmico e as funções no modelo funcional.

O modelo dinâmico descreve a estrutura de controle dos objetos. As ações no diagrama de estados correspondem as funções no diagrama funcional. Os eventos no diagrama de estados se tornam as operações no modelo objeto.

O modelo funcional descreve as funções invocadas pelas operações no modelo objeto e ações no modelo dinâmico. As funções operam sobre as valores de dados especificados pelo modelo objeto. O modelo funcional mostra ainda as restrições sobre os valores objeto

5. ANÁLISE ORIENTADA A OBJETOS

A fase de análise diz respeito ao entendimento e à modelagem da aplicação e do domínio no qual ela opera. Esta fase focaliza *o que* necessita ser feito e não *como* deve ser feito. A etapa inicial da fase de análise consiste no estabelecimento dos requisitos do problema a ser resolvido, fornecendo uma visão conceptual do sistema proposto. O dialogo subsequente com o usuário, o conhecimento da área e a experiência adquirida do mundo real são elementos adicionais que servem de entrada para a análise. A saída desta fase de análise é um modelo “formal” que captura os aspectos essenciais do sistema: objetos e suas relações, fluxo dinâmico de controle e transformação funcional de dados. No caso da metodologia OMT, obtém-se os modelos objeto, dinâmico e funcional.

5.1. Modelo objeto

Para construir o modelo objeto, é necessário seguir os seguintes passos:

- identificar objetos e classes;
- preparar um dicionário de dados;
- identificar associações (incluindo agregações) entre objetos;
- identificar atributos de objetos e ligações;
- organizar e simplificar classes de objetos utilizando herança;
- verificar os caminhos de acesso;
- iterar e refinar o modelo;
- agrupar as classes em módulos.

5.1.1. Modelo dinâmico

Para construir o modelo dinâmico, é necessário seguir os seguintes passos:

- preparar cenários de seqüências de interação típicas;
- identificar eventos entre objetos;
- preparar um rastro de eventos para cada cenário;
- construir um diagrama de estados;
- equiparar eventos entre objetos para verificar a consistência.

5.1.2. Modelo funcional

Para construir o modelo funcional, é necessário seguir os seguintes passos:

- identificar valores de entrada e saída;
- construir um DFD mostrando dependências funcionais;
- descrever funções;
- identificar restrições (i.e dependências funcionais) entre objetos;
- especificar um critério de otimização

5.1.3. Acrescentando as operações

O estilo de análise apresentado não da ênfase nas operações; entretanto é necessário distinguir os vários tipos de operações durante a fase de análise: operações a partir do modelo objeto, de eventos, de ações e atividades no diagrama de estados, de funções (DFD).

6. PROJETO ORIENTADO A OBJETOS

O projeto orientado a objetos se divide em projeto do sistema e projeto do objeto, descritos sucintamente a seguir.

6.1. Projeto do Sistema

O projeto do sistema consiste em tomar decisões de alto nível sobre o *como* o problema será resolvido e a solução construída. O projeto de sistemas inclui decisões sobre a arquitetura do sistema i.e. sobre a organização do sistema em subsistemas , a alocação de subsistemas a componentes de hardware ou de software e sobre a política e a estratégia que formam o quadro no qual o projeto detalhada poderá ser desenvolvido.

O projetista do sistema deve tomar as seguintes decisões:

- organizar o sistema em subsistemas, a partir das noções de camadas (subdivisão horizontal) e de partições (subdivisão vertical);
- identificar a concorrência inerente ao problema e definir as tarefas concorrentes;
- alocar os subsistemas a processadores e tarefas;
- escolher uma abordagem para gerenciar os armazenadores de dados;
- determinar os mecanismos para manusear o acesso aos recursos globais;
- escolher a implementação do controle no software;
- determinar o manuseio das condições limites;
- estabelecer as prioridades entre os compromissos.

6.2. Projeto do objeto

A fase de projeto de objeto determina as definições completas das classes e associações utilizadas na implementação, bem como as interfaces e os algoritmos dos métodos utilizados par implementar as operações. Nesta fase, são adicionados objetos internos para a implementação e otimizado as estruturas de dados e os algoritmos. O projetista deverá escolher entre várias formas de implementação, levando em conta questões como minimização do tempo de execução, da memória e outras medidas de custo. A otimização do projeto deve ainda levar em conta as facilidades de implementação, manutenção e expansão.

Durante o projeto do objeto, o projetista deve realizar os passos seguintes:

- combinar os três modelos para obter as operações sobre as classes;
- projetar os algoritmos para implementar as operações;
- otimizar os caminhos de acesso aos dados;
- implementar o controle para as interações externas;
- ajustar a estrutura de classes para adicionar a herança;
- realizar o projeto das associações;
- determinar a representação do objeto;
- empacotar classes e associações nos módulos;
- documentar as decisões de projeto.