Engenharia de Software II- Parte 1

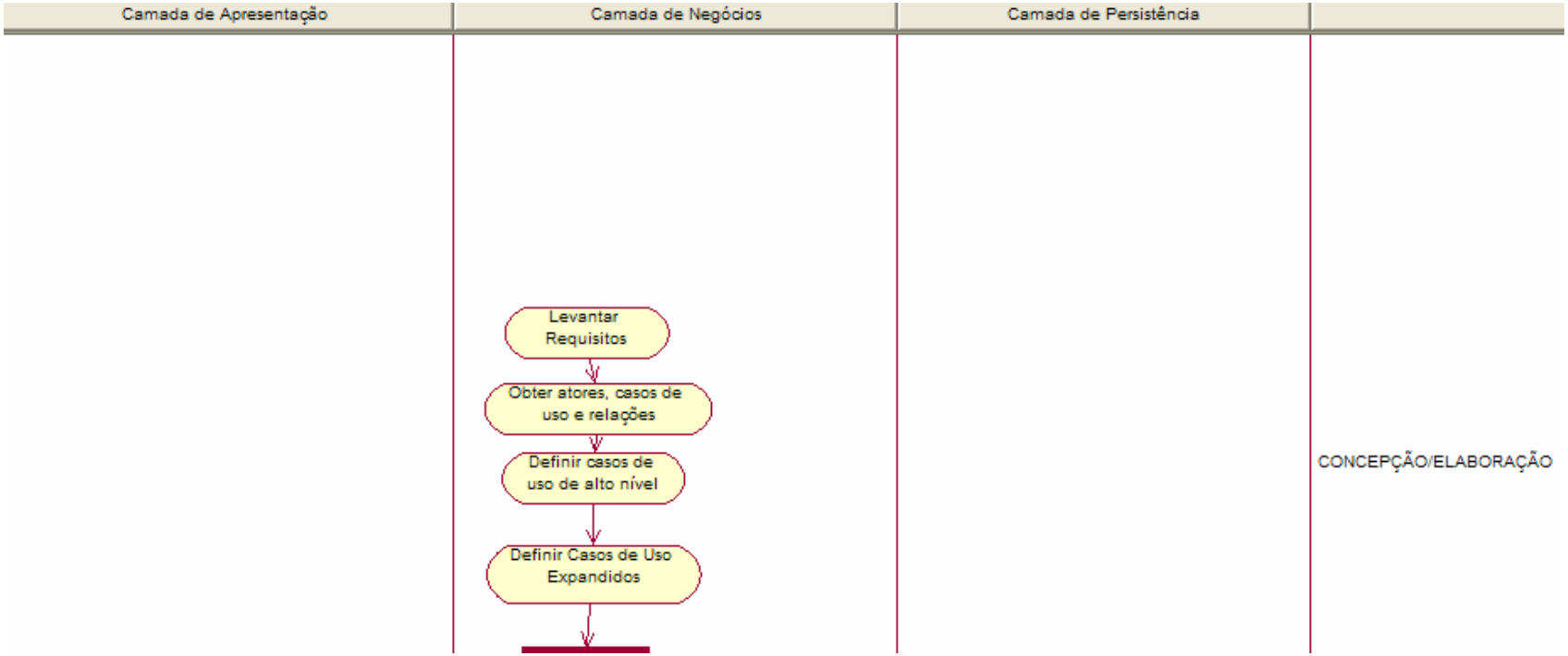
Descrição do sistema-exemplo Revista Digital

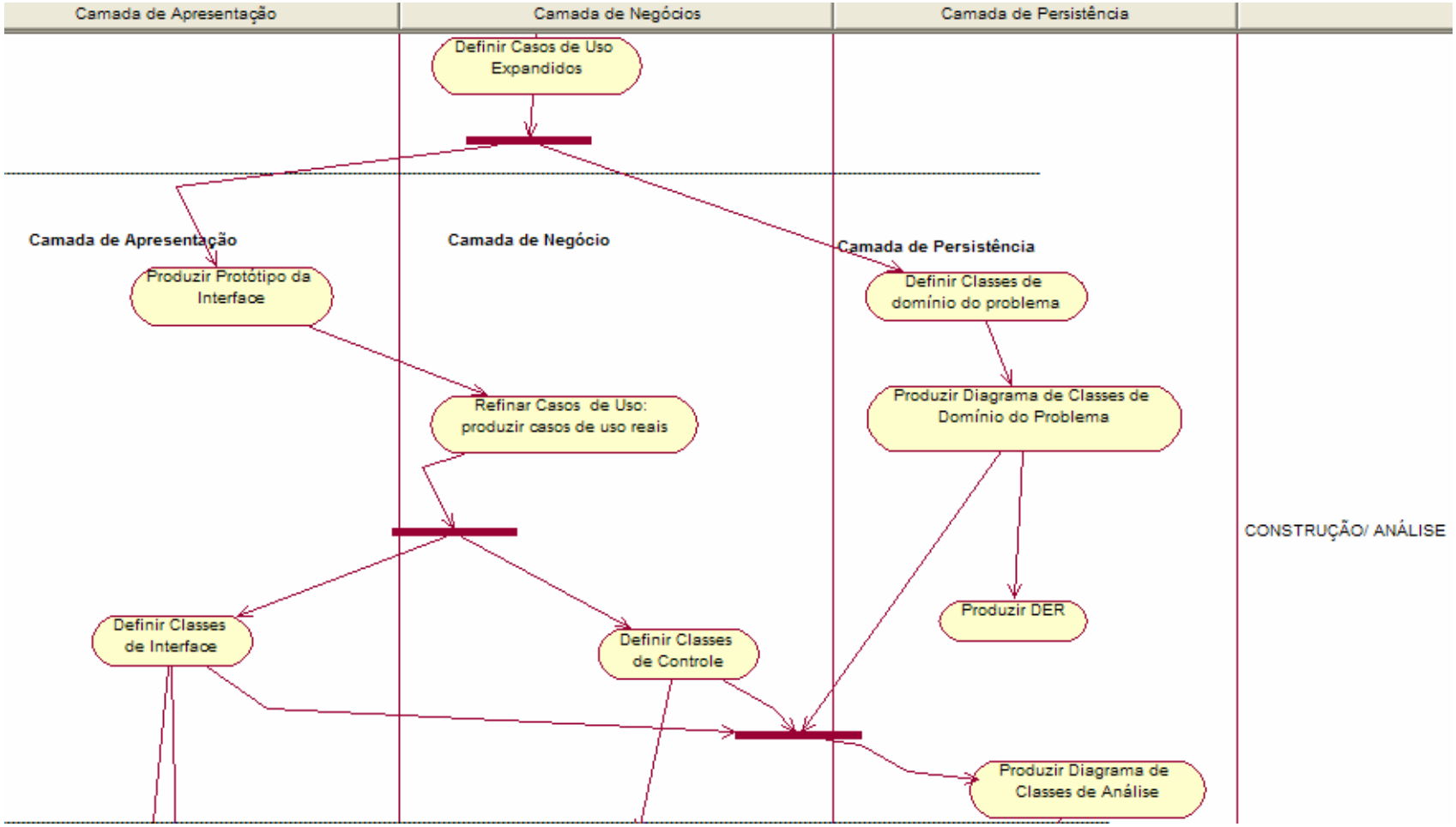
O objetivo deste sistema é permitir a professores e alunos uma ferramenta para submissão e acesso a artigos produzidos por professores da PUC Minas pertencentes aos diversos cursos da mesma. As funcionalidades requeridas são as seguintes:

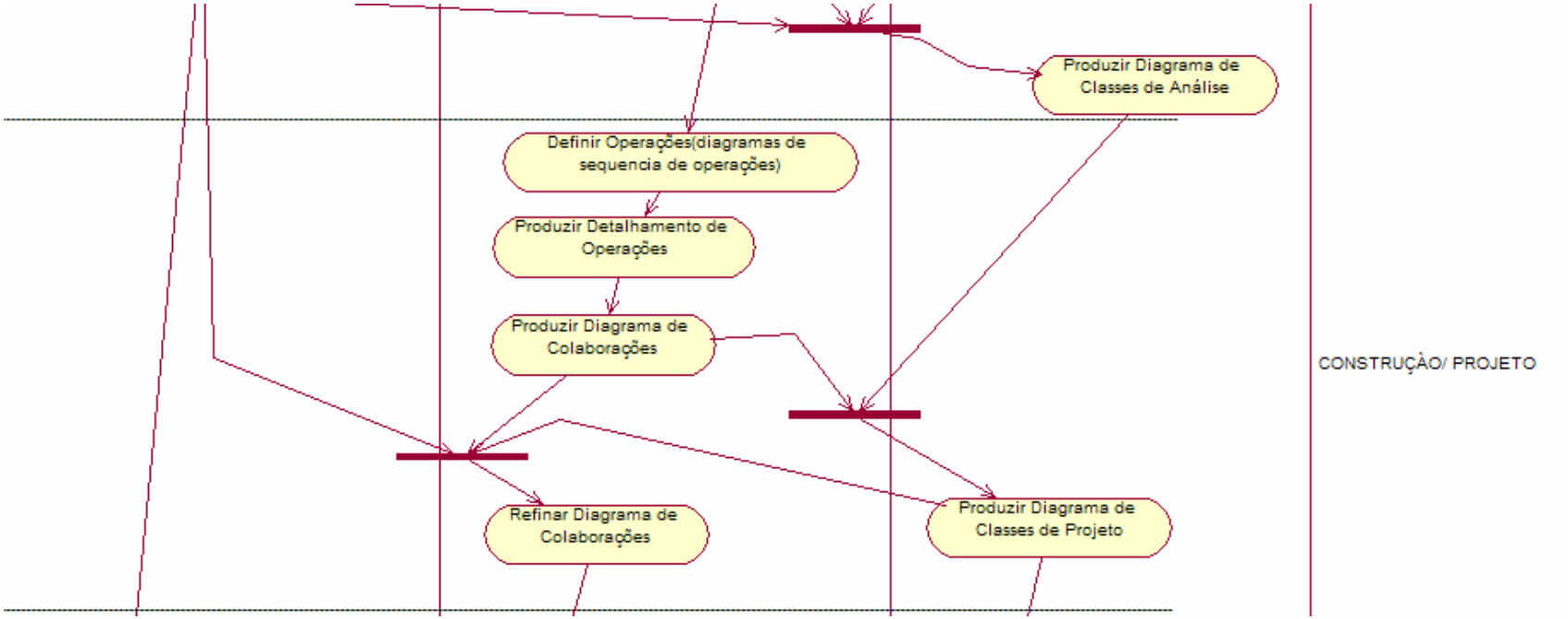
-Autenticação de usuários em três níveis: leitor, autor e administrador.
-Pesquisar documentos por título, nome do autor, código RT e palavras-chaves.
-Fazer o download de documentos pesquisados.
-Submissão (upload) de arquivos para revisão.
-Publicação de arquivos em cadernos periódicos.
-Definir revisores de documentos.
-Registrar autores e revisores de documentos.
-Permitir o registro da revisão do documento.
-Permitir a inclusão/exclusão/busca/alteração de administradores, leitores, revisores, cursos e tópicos (assuntos)

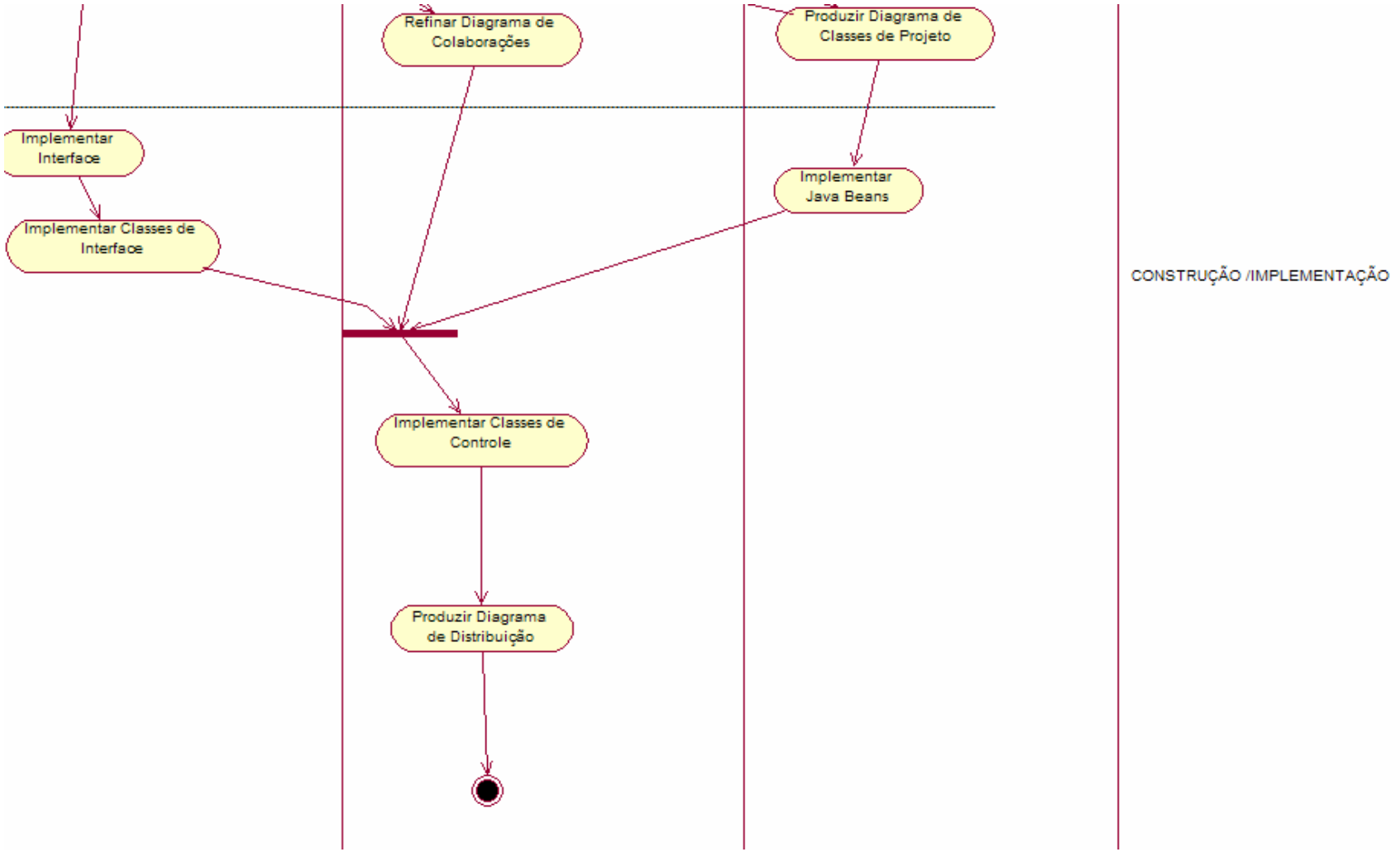
1-Processo de desenvolvimento adotado

Um processo é, em essência, uma cadeia de atividades a serem realizadas durante o desenvolvimento do software. O processo que aqui propomos apresenta atividades para as disciplinas de Requisitos, Análise, Projeto e Implementação do Rational Unified Process (RUP). As outras disciplinas, ainda que importantes e essenciais, não estão no escopo deste curso.
A arquitetura 3 camadas é ideal a aplicações web. O processo aqui apresentado distribui as atividades nas 3 camadas (apresentação, negócio e persistência), fato que permite explorar o desenvolvimento independente por equipes distintas. Assim sendo, as atividades estão distribuídas em duas dimensões: as etapas do RUP e a camada onde se localizam.


Camada de Apresentação
Camada de Negócios
Camada de Persistência
Levantar Requisitos
Obter atores, casos de uso e relações
Definir casos de uso de alto nível
Definir Casos de Uso Expandidos
CONCEPÇÃO/ELABORAÇÃO


Camada de Apresentação
Camada de Negócios
Camada de Persistência
Definir Casos de Uso Expandidos
Camada de Apresentação
Camada de Negócio
Camada de Persistência
Produzir Protótipo da Interface
Definir Classes de domínio do problema
Refinar Casos de Uso: produzir casos de uso reais
Produzir Diagrama de Classes de Domínio do Problema
CONSTRUÇÃO/ ANÁLISE
Produzir DER
Definir Classes de Interface
Definir Classes de Controle
Produzir Diagrama de Classes de Análise


Produzir Diagrama de Classes de Análise
Definir Operações(diagramas de sequencia de operações)
Produzir Detalhamento de Operações
Produzir Diagrama de Colaborações
Refinar Diagrama de Colaborações
Produzir Diagrama de Classes de Projeto
CONSTRUÇÀO/ PROJETO


Refinar Diagrama de Colaborações
Produzir Diagrama de Classes de Projeto
Implementar Interface
Implementar Java Beans
Implementar Classes de Interface
CONSTRUÇÃO /IMPLEMENTAÇÃO
Implementar Classes de Controle
Produzir Diagrama de Distribuição

2-Etapa de Concepção/Elaboração

Esta é a primeira etapa do processo de desenvolvimento adotado e corresponde essencialmente a atividades que visem:

-Analisar a viabilidade do projeto.

-Identificar a estrutura e a dinâmica da organização na qual o sistema vai ser instalado.

-Identificar os problemas da organização e propor melhorias.

-Levantar requisitos.

-Especificar os requisitos levantados.

Neste curso, detalhamos as atividades relacionadas a especificação de requisitos. As outras fogem ao escopo deste curso.

2.1-UML- Modelo de Casos de Uso

2.1.1-Definições

O modelo de casos de uso constitui o principal artefato da UML, pois é utilizado em todas as fases do desenvolvimento da aplicação, servindo de base para determinação de:

-Classes na fase de análise

-Operações, na fase de projeto

-Testes de verificação e validação de requisitos.

-Manual do usuário

Na essência,um *caso de uso* é uma interação entre um usuário e um sistema computacional, de tal forma que sejam representadas todas as funções visíveis ao usuário, bem como seus objetivos. Um *ator* é um papel que um usuário representa com respeito ao sistema. Normalmente um caso de uso é utilizado por um ator. Um único ator pode utilizar vários casos de uso, da mesma forma, um caso de uso pode interagir com vários atores. Não precisam necessariamente ser humanos. Podem, por exemplo ser sistemas externos que utilizam informações do sistema sendo modelado.Os atores podem ter vários papéis com relação ao caso de uso. Eles podem obter valores ou podem apenas participar do mesmo. O diagrama de casos de uso representam as interações entre casos de uso e atores. Estas interações são representadas por setas ligando o caso de uso ao ator. A Figura 2.1 mostra a notação utilizada no modelo de casos de uso.

Um caso de uso é um processo relativamente grande. Isto significa que envolve normalmente muitos passos, não sendo nunca constituído por um único passo ou atividade individual. Seu detalhamento é feito por meio de descrições textuais que podem ser sucessivamente refinadas durante o processo de desenvolvimento, começando por descrições de alto nível (casos de uso de alto nível) e chegando a descrições minuciosas (casos de uso reais), melhor descritas adiante.

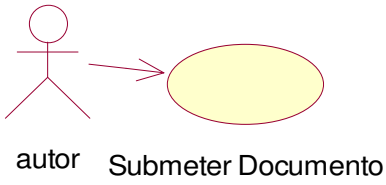


Figura 2.1- Notação da UML para ator (autor), interação (seta) e caso de uso (Submeter Documento)

2.1.2-Relações entre casos de uso

As relações entre casos de uso podem ser do tipo *usa* ou *estende.*
A relação *estende* (*extends)* é utilizada para indicar que determinado caso de uso pode ser inserido em outro, estendendo sua descrição. Ambos serão totalmente independentes, ou seja, funcionalidades adicionais de extensão podem ser adicionadas sem qualquer alteração no caso de uso estendido, ou seja, este é completo. As situações em que normalmente se usa a ligação estende são as seguintes:

-Para modelar variações com que um caso de uso pode ser executado. Neste caso, na descrição do use case estendido deve-se colocar referências aos casos de uso extensores, no seguinte formato:

Caso <condição> iniciar < nome do caso de uso extensor>

-Para modelar situações excepcionais que somente ocorram em certos casos. Neste caso, a primeira linha da descrição do caso de uso que estende tem o seguinte formato padrão:

Caso de uso <nome> é uma situação excepcional de<nome do caso de uso estendido> na linha <número da linha> quando <evento do qual deriva a situação excepcional>

-Para modelar a situação em que vários casos de uso podem ser inseridos em outro, como em um sistema de menus, por exemplo.

Na Figura 2.2 o caso de uso Pesquisar tem sua funcionalidade estendida por variações que correspondem às diversas formas possíveis de pesquisa.

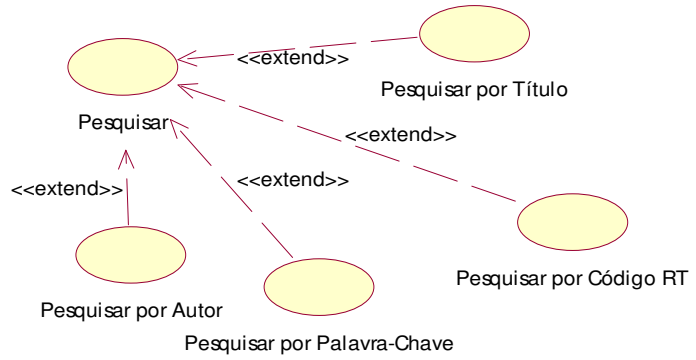


Figura 2.2-Utilização da ligação estende (extend)

A relação *usa (uses* ou *include)* denota a relação de um caso de uso com outro cujo comportamento é complexo e comum a vários casos de uso. Assim sendo, ocorre a reutilização de um caso de uso, por outros que necessitem de sua funcionalidade. Se um caso de uso A *usa* um caso de uso B, no momento em que B é invocado, a execução de A pára, inicia-se a execução de B e, ao término desta, a execução retorna para A no ponto em que foi interrompida para a execução de B.
A Figura 2.3 mostra a utilização da ligação *usa (include)*. Os casos de uso de cadastro têm uma funcionalidade que se repete em todos eles, representada pelo caso de uso *Cadastrar.* Este caso de uso permite inserir, alterar, excluir e pesquisar dados no banco de dados.

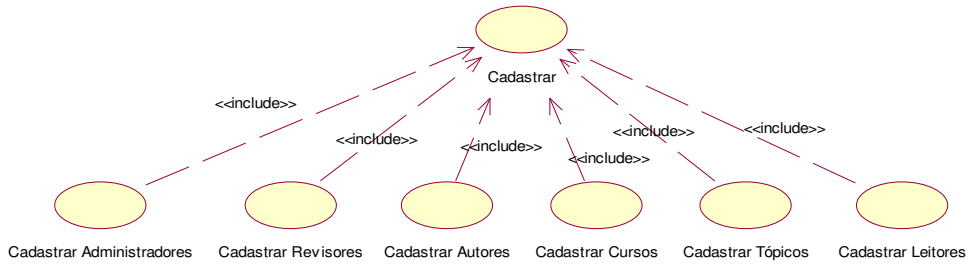


Figura 2.3 – Utilização da relação usa (include)

2.1.3-Herança entre atores

Atores que têm um papel em comum podem ser representados no diagrama de casos de uso pela notação mostrada na Figura 2.3. Nela, os atores autor, leitor, revisor e administrador têm papéis em comum, que são denotados pelo ator usuário. Na Figura 2.5, podemos ver que um dos papéis em comum é acessar os casos de uso Pesquisar e Autenticar. Repare que conseguimos com esta notação economizar um número substancial de linhas que seriam necessárias para conectar cada um dos quatro atores aos dois casos de uso.

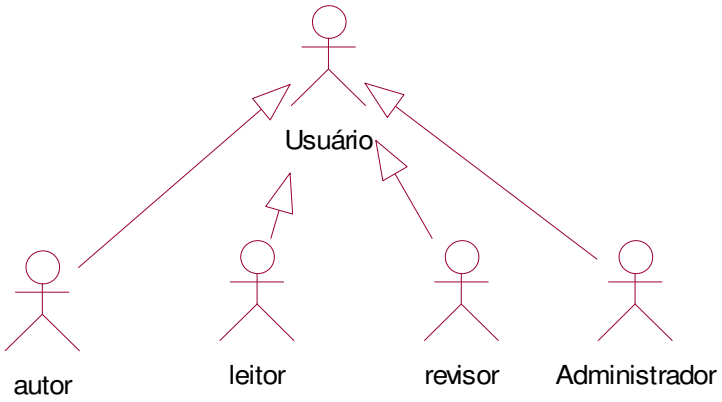


Figura 2.4-Herança entre atores

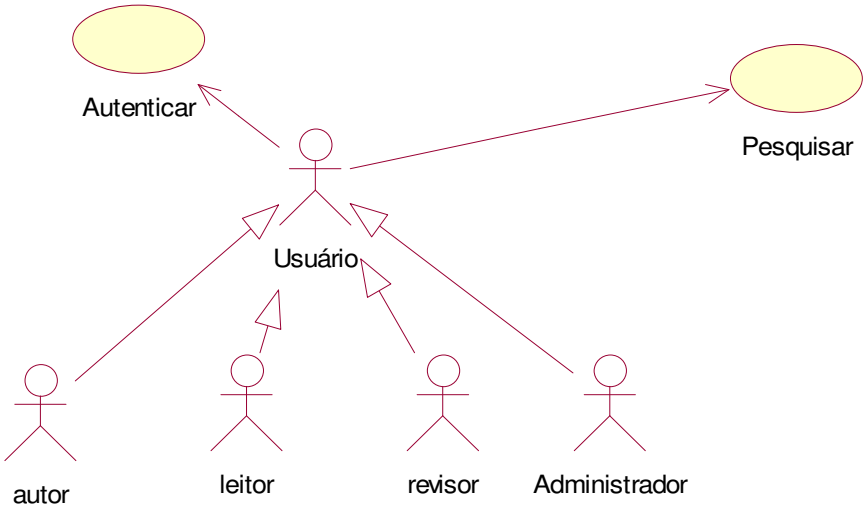


Figura 2.5-O ator usuário acessa os casos de uso Pesquisar e Autenticar, comuns aos 4 atores filhos.

2.2-Atividade Levantar Requisitos

2.2.1-Definição:Requisitos

Em essência, um requisito é uma necessidade do usuário. Levantar requisitos é uma atividade árdua, que envolve normalmente uma grande interação entre o analista e o usuário, exigindo técnicas específicas (que não estão no escopo deste livro). Devem ser definidos de maneira clara e objetiva, sem ambiguidades.
Projetos bem sucedidos advêm de requisitos bem definidos. Requisitos de alta qualidade têm as seguintes características:
-São corretos, ou seja, correspondem realmente ao que o usuário quer;
-São completos, ou seja, representam tudo o que o usuário necessita;
-São consistentes, ou seja, não existe conflito entre eles;
-São testáveis, ou seja, são passíveis de serem verificados quanto a quaisquer destas características;
-São alteráveis, ou seja, podem ser modificados sem que isto cause transtorno ao desenvolvimento em si ou a outros requisitos.
-São precisos, ou seja, sua interpretação é livre de ambigüidades.
Segundo [SOM01], requisitos de sistemas de software são classificados como:

-*Funcionais:* correspondem às funções que o sistema deve possuir, reações a determinadas entradas e o comportamento em situações específicas.
-*Não funcionais:* são restrições de tempo, memória, segurança, desempenho, qualidade, dentre outros.
-*Requisitos de domínio:* são requisitos relativos ao domínio de aplicação do sistema, refletindo suas características.

2.2.2-O levantamento de requisitos

Esta atividade é extensa e seu detalhamento foge ao escopo deste curso. Em essência, a atividade de levantamento de requisitos se faz mediante a utilização de:
-*Entrevistas com o usuário:* consiste em um diálogo entre o entrevistador e o entrevistado, dentro de um roteiro previamente elaborado. A entrevista deve ser adaptada ao nível profissional, cultural e social do entrevistado.
-Análise documental: consiste na análise de formulários, relatórios, contratos e documentos essenciais ao usuário;
-Questionários: são questões sistematizadas e bem direcionadas, dispostas em impressos padronizados, a serem aplicadas aos usuários do sistema. É uma alternativa à entrevista principalmente em situações em que os elementos do universo desejado sejam muitos, ou ainda, estejam geograficamente dispersos, quando há pouco tempo,ou quando as respostas exijam cálculo ou um tempo maior para serem respondidas.
-Observação direta: serve para complementar ou enriquecer as conclusões obtidas através dos meios anteriores, através da observação direta do meio que está sob levantamento.
-*Reuniões JAD*- são reuniões realizadas para que os requisitos sejam obtidos por um grupo de pessoas composto por usuários, analistas e especialistas do domínio do problema.

2.3-Atividade obter os atores, casos de uso e relações

Os passos a seguir direcionam na obtenção sistemática de casos de uso, atores e interações:

1-Definem-se os atores que irão interagir com o sistema. Atores podem ser:
- -Papéis a serem realizados pelo usuário
- -Unidades organizacionais
- -Grupos de usuários
- -Dispositivos de hardware
- -Sistemas externos com os quais interagir

No exemplo da Revista Digital os atores realizam papéis dentro do domínio do problema. Estes papéis são os seguintes:

Autor: responsável pela submissão de artigos a serem publicados pela revista.
Leitor: pesquisa artigos e faz download dos mesmos.
Revisor: responsável por fazer as revisões dos artigos submetidos pelos autores.
Administrador: responsável por cadastrar todos os dados do sistema, montar cadernos e distribuir os artigos pelos revisores.
Usuário: corresponde aos papéis comuns realizados pelos atores acima, como autenticar, fazer pesquisa e fazer download de documentos.

2-A partir do levantamento de requisitos, capturam-se os principais eventos externos do sistema, sob a visão do usuário. Um evento externo corresponde a uma ação que do usuário ao utilizar o sistema.
No caso da Revista Digital, temos:

1-Usuário faz autenticação.
2-Usuário faz pesquisa por título.
3-Usuário faz pesquisa por nome do autor.
4-Usuário faz pesquisa por código RT.
5-Usuário faz pesquisa por palavra-chave.
6- Usuário faz download de documentos.
7-Autor submete documentos.
8-Autor registra autores do documento.
9-Autor exclui autores do documento.
10-Revisor faz revisões de documentos.
11-Administrador aloca revisores a documentos.
12-Administrador monta cadernos.
13-Administrador cadastra administradores.
14--Administrador cadastra revisores.
15-Administrador cadastra autores.
16-Administrador cadastra cursos.
17-Administrador cadastra tópicos.
18-Administrador cadastra leitores.

3-Cada evento da lista de eventos dá origem a um caso de uso e às relações entre o ator e o caso de uso.

Cada evento acima dá origem a um caso de uso:

Casos de uso	Atores
Autenticar	Usuário
Pesquisar por título	Usuário
Pesquisar por nome do autor	Usuário
Pesquisar por código RT	Usuário
Pesquisar por palavra-chave	Usuário
Fazer download de documentos	Usuário
Submeter documentos	Autor
Registrar autores	Autor
Excluir autores	Autor
Fazer revisões de documentos	Revisor
Alocar revisores a documentos	Administrador
Montar cadernos	Administrador
Cadastrar administradores	Administrador
Cadastrar revisores	Administrador
Cadastrar autores	Administrador
Cadastrar cursos	Administrador
Cadastrar tópicos	Administrador
Cadastrar leitores	Administrador

4-Desmembrar casos de uso com alto grau de complexidade, ou agregar casos de uso que sejam simples e vinculáveis a outros.

No exemplo, os casos de uso excluir autores e registrar autores são partes do processo de submissão de um documento, por isso serão agregados ao caso de uso submeter documentos. Na descrição deste último far-se-á referência aos agregados em subseções.

5-Criam-se, onde necessário, generalizações <EXTENDS> ou reutilizações <INCLUDE> de forma a tornar o diagrama mais robusto.

No exemplo, os casos de uso de pesquisa podem ser ligados ao caso de uso denominado *pesquisar* que será estendido por eles (ligação extend).
Podemos criar um caso de uso comum aos casos de uso de cadastro, denominado *Cadastrar,* conectado a eles por ligações *usa(include).* Ele vai agregar funcionalidades comuns a todos eles, quais sejam, incluir, excluir, e pesquisar dados.

Na Figura 2.6 temos o Diagrama de Casos de Uso resultante da realização das atividades supra descritas.

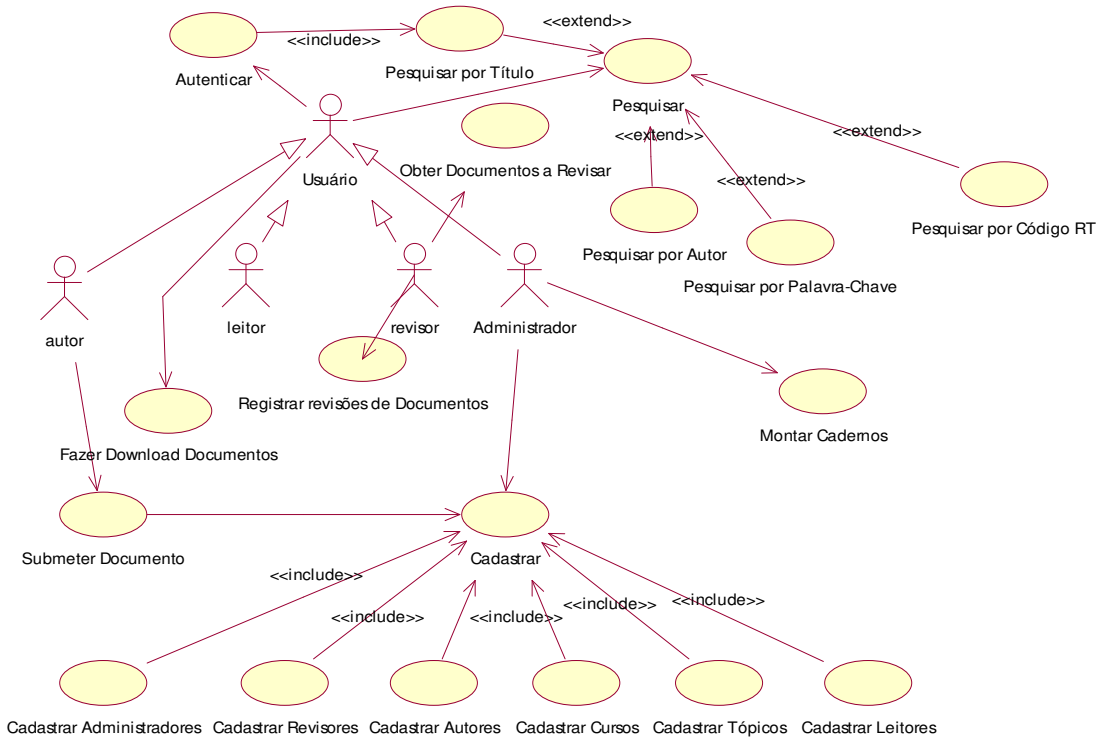


Figura 2.6-Diagrama de casos de uso da Revista Digital

2.4-Atividade Definir Casos de Uso de alto nível

Os chamados casos de uso de alto nível [LAR00], contêm um texto narrativo sucinto que descreve o processo ao qual se referem. Constituem uma primeira abordagem acerca da funcionalidade do caso de uso, que servirá como base para refinamentos efetuados em etapas posteriores (casos de uso expandidos e reais).
Seja por exemplo o caso de uso Submeter Documento. Sua descrição de alto nível é a seguinte:

O autor seleciona o documento a ser submetido a revisão. Em seguida comanda o registro de dados básicos, de autores e finalmente envia este documento para o setor de revisões da revista digital.

Repare que se pretende com esta descrição uma visão geral do processo, sem detalhes de como a seleção e o envio serão efetivados.

2.5-Atividade Definir Casos de Uso Expandidos

Um caso de uso expandido [LAR00] constitui uma versão mais detalhada do caso de uso de alto nível, permitindo uma compreensão mais profunda dos processos e dos requisitos. A descrição é feita na forma de *fluxo de eventos.* Um fluxo de eventos é uma seqüência de ações que realizam a funcionalidade representada pelo caso de uso. Na abordagem aqui descrita, distinguimos dois tipos de fluxo de eventos: as *ações realizadas pelo ator* e a *resposta do sistema* a estas ações. Estes fluxos são numerados seqüencialmente e dispostos em duas colunas paralelas. Abaixo temos o modelo de descrição do caso de uso expandido:

Na parte superior da descrição temos um resumo do caso de uso:

Caso de uso : nome do caso de uso.
Atores: lista de atores que interagem com este caso de uso.
Finalidade: objetivo do caso de uso.
Visão geral: descrição sucinta, a mesma utilizada no de alto nível.
Pré-condições: lista de condições que tenham sido satisfeitas para que o caso de uso seja executado.
Pós-condições: lista de condições obtidas após a realização do caso de uso.

Na parte média temos a seqüência de eventos, que vai descrever as ações do ator e a resposta do sistema às mesmas. Normalmente esta seqüência de eventos é típica, não descrevendo situações alternativas. O formato desta seção é o seguinte:

Seqüência típica de eventos

Ação do ator	**Respostas do Sistema**
Descrição das ações numeradas dos atores.	Respostas numeradas do sistema

Se houver pontos de decisão dentro da seqüência típica de eventos duas situações devem ser consideradas:
-Se um dos ramos de decisão for um caso bastante típico e os outros forem situações raras então o ramo típico permanece na seqüência típica e as alternativas são descritas numa seção denominada *Alternativas:*

Seqüências alternativas

Enumerar as ações alternativas.

-Se os pontos de decisão representam situações equiprováveis, criam-se *subseções,* que vão possuir, cada uma, sua seqüência típica de eventos. Na seção principal criam-se desvios para as subseções, utilizando-se a estrutura condicional *Se <condição> ver seção <nome da subseção>.*Esta subseções podem, inclusive ter também seqüências alternativas. Se estas subseções forem

comuns a vários casos de uso, podem se tornar casos de uso independentes, conectados ao principal por meio das relações *usa.* Na descrição do caso de uso, ao invés da expressão *ver seção <nome da subseção>,* colocamos *iniciar <nome do caso de uso>.*

A seguir temos um exemplo de uma descrição expandida para o caso de uso Submeter Documento.

Caso de uso : Submeter Documentos
Atores: autor
Finalidade: Enviar documento para equipe de revisão.
Visão geral: O autor seleciona o documento a ser submetido a revisão. Em seguida comanda o registro de dados básicos, de autores e finalmente envia este documento para o setor de revisões da revista digital.
Pré-condições: Autor cadastrado no sistema. Autor preencheu dados do documento a ser submetido. Documento disponível para envio.
Pós-condições: Documento enviado e armazenado para que possa ser revisado.

Seqüência típica de Eventos

Ações do ator	Resposta do Sistema
1-Iniciar caso de uso *Pesquisar por Título* 2-Se não houver cadastrado o documento ver seção *Cadastrar Dados Documento* 3-Se não houver registrado os autores do documento, ver seção *Registrar Autores do Documento* 4-O autor, então comanda o envio do documento.	5-Grava o documento submetido no servidor, para ser posteriormente revisado.

Seqüências alternativas

Linha 5: Problemas no envio (upload) do documento. Indicar erro de comunicação.

Subseção Cadastrar Dados Documento

Seqüência típica de Eventos

Ações do ator	Resposta do Sistema
1-Depois que o sistema retornou a lista de documentos do autor, este, se	

já não houver feito, pode: a-Selecionar um documento para alterar dados. Após seleção feita, iniciar *Cadastrar,* enviando para alteração os dados *Código RT, Título, Resumo, Data de Submissão, Tópico e Palavras-chaves.* b-Selecionar opção para inserir dados de um novo documento. Iniciar *Cadastrar* para inserir os dados *Código RT, Título, Resumo, Data de Submissão, Tópico e Palavras-chaves.*	

Observação: *repare que nesta subseção estamos invocando o caso de uso Cadastrar, que se relaciona com o caso de uso Submeter Documentos por meio de uma relação usa (include-ver diagrama completo e descrição do caso de uso Cadastrar no próximo item).*

Subseção Registrar Autores do Documento

Ações do ator	Resposta do Sistema
1-O autor deve inicialmente pesquisar o nome do(s) autor(es) do documento. Para tal pode:	
1.a.1-Deixar o campo de nome em branco e solicitar a pesquisa	1.a.2 – Retornar a lista de todos os autores cadastrados no sistema
1.b.1-Digitar o nome do autor ou uma parte do mesmo e solicitar a pesquisa.	1.b.2- Retornar o(s) nome(s) dos autores encontrados no banco que mais se assemelhem ao solicitado.
2-Para cada autor do documento:	
2.a.1 – Seleciona seu nome da lista e solicita sua inclusão.	2.a.2-Inserir autor na lista de autores do documento.
2.b.1-Se desejar excluir um autor, seleciona e solicita sua exclusão.	2.b.2-Excluir o autor da lista de autores.

Observação: *Na linha 2, repare que utilizamos uma estrutura de repetição, indicando que ações subseqüentes podem ser feitas tantas vezes quantos forem os autores a serem registrados. Repare também na sugestão para estruturas de decisão nas linhas 1.a.1 e 2.a.1*

Seqüência alternativa

Linha 2.1- Autor selecionado já está na lista de autores. Sistema emite mensagem informando que autor já foi incluído.

3 -Etapa de Concepção-Análise

Na etapa anterior obtivemos como artefatos principais o diagrama de casos de uso estendidos e as descrições dos casos de uso. Nesta etapa iniciamos o desenvolvimento incremental, em que cada caso de uso é refinado de forma a produzirem-se descrições reais.

3.1-UML – Diagramas de Classes e Objetos

3.1.1-Definição

Um objeto é um conceito pertinente a um domínio específico, abstraído na forma de *procedimentos,* operações que são capazes de executar, e *atributos* que o caracterizam. As operações são invocadas através de mensagens trocadas entre objetos, ou seja um objeto colabora com outro executando uma operação que é de sua responsabilidade, que lhe foi solicitada.
Uma classe é uma representação de um conjunto de objetos que agregam características semelhantes. O diagrama de classes é o artefato da UML que mostra as diversas classes que compõem o sistema e, principalmente, as relações entre estas. Fornece, portanto, uma visão estática do sistema sob modelamento. Os diagramas de classes são produzidos em diversos níveis, num processo incremental. Os tipos de diagramas seguintes traduzem níveis de abstração distintos, que se superpõem:

-Diagrama de Classes de Domínio de Problema: traduz abstrações específicas de classes que são nativamente pertinentes ao domínio no qual o sistema se insere.

-Diagrama de Classes de Análise: contém o diagrama anterior e abstrações e considerações específicas propostas pelo processo de análise.

-Diagrama de Classes de Projeto: é o diagrama de análise modificado e acrescido de abstrações e considerações específicas propostas pelo processo de projeto.

O diagrama de objetos, por sua vez, modela instâncias de classes contidas em um diagrama de classes. Portanto, eles denotam o estado de um determinado diagrama de classes em um ponto determinado no tempo.

Nos itens seguintes apresentaremos a notação que a UML propõe para diagramas de classes e de objetos em seus diversos níveis. Mais adiante, descreveremos o processo de criação de cada um dos níveis de abstração acima citados.

3.1.2-Notação

3.1.2.1-Classes e Objetos

Classes e objetos são representados por retângulos contendo um nome identificador. As particularidades de notação são as seguintes:

-Quando deseja-se representar um objeto sem referência à classe à qual pertence, colocamos seu nome sublinhado dentro do retângulo:

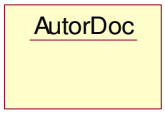


Figura 3.1: Notação para objeto sem referência a sua classe

-Quando deseja-se representar um objeto e a classe à qual pertence, utiliza-se a notação *<nome objeto>:<classe>:*

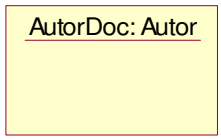


Figura 3.2: Notação para objeto com referência à classe à qual pertence

-Quando deseja-se representar o objeto, não sendo relevante o nome de sua instância, representa-se apenas o nome da classe à qual pertence, precedido por um sinal de dois pontos:

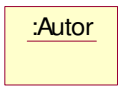


Figura 3.3: Notação para objeto com omissão do nome de sua instância.

-Uma classe é representada por um retângulo dividido em até três seções. Na seção superior é colocado o nome da classe. Nas seções seguintes alocam-se, respectivamente de cima para baixo:

-Atributos: constituem a caracterização dos objetos-instâncias das classes. A sintaxe para representar atributos é a seguinte:

visibilidade nome : tipo = valor default

onde

visibilidade: veja o conceito abaixo em operações
nome: identificador do atributo
tipo : tipo do identificador

Visibilidade e tipo são opcionais.

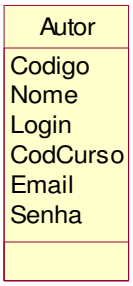


Figura 3.4- Atributos de classe

A sintaxe para representar atributos é a seguinte:

visibilidade nome : tipo = valor default

onde

visibilidade: +(pública),#(protegida) ou -(privada)
nome: identificador do atributo
tipo : tipo de dado do identificador

-Operações: Denotam as ações passíveis de serem realizadas por qualquer objeto pertencente à classe. São representadas segundo a seguinte sintaxe:

visibilidade nome(lista de parâmetros): tipo de retorno {valores aplicáveis}

onde

visibilidade : +(pública),#(protegida) ou -(privada)
nome: string com o nome da operação
lista de parâmetros : lista de argumentos com a mesma sintaxe dos atributos.
tipo de retorno: tipo de dado de retorno, quando for o caso.
valores aplicáveis: lista de valores aplicáveis à operação.

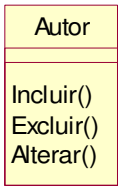


Figura 3.5- Operações de classe

As seções de atributos e operações não necessitam obrigatoriamente ser mostradas no diagrama de classes.

3.1.2.2-Associações

Associações são, na essência, "relacionamentos estruturais entre classes de tipos diferentes". Os diversos tipos suportados pela UML estão relacionados nos subitens a seguir.

3.1.2.2.1-Associações de ocorrência

Corresponde a associações entre classes que descrevem o número de ocorrências (cardinalidade) de uma em relação a outra. A Figura 3.6 a seguir mostra algumas cardinalidades comuns a associações de ocorrência.

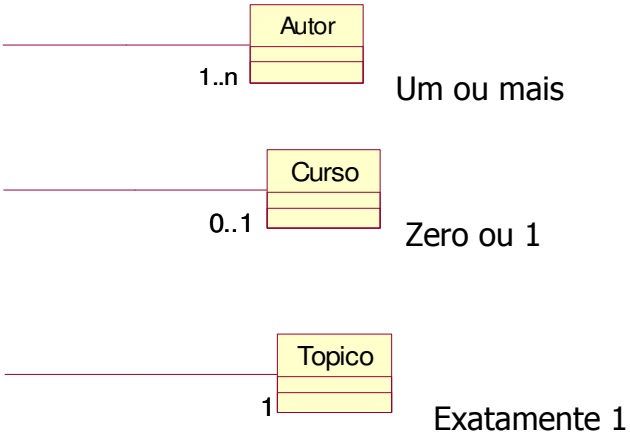


Figura 3.6-Algumas associações entre classes

*Observação – A cardinalidade 0..n muitas vezes é substituída por um * na notação da UML.*

3.1.2.2.2-Associações todo-parte

Correspondem a relações entre uma classe (todo) e suas partes constituintes. Identificamos dois tipos de associações todo-parte:

Agregação: as partes podem existir sem o todo (Figura 3.7).

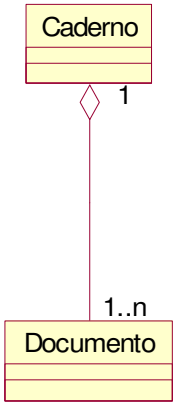


Figura 3.7- Exemplo de agregação. Documento existe independentemente da existência de caderno.

Composição: as partes dependem do todo para existir. O fim do todo significa o fim das partes (Figura 3.8).

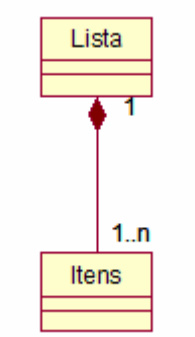


Figura 3.8- Exemplo de composição: a parte Itens depende do todo Lista para existir: a eliminação de Lista provoca a eliminação de Itens.

3.1.2.2.3-Associações múltiplas

São associações que ocorrem entre três ou mais classes. Cada instância deste tipo de associação é uma n-tupla de valores das respectivas classes. Na Figura 3.9, temos sua representação.

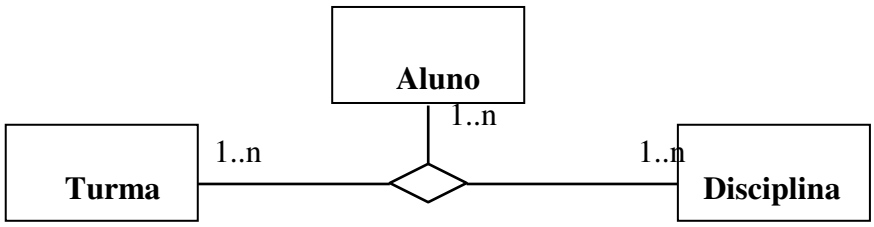


Figura 3.9-Associação ternária

3.1.2.2.4-Navegabilidade

A navegabilidade entre duas classes define a visibilidade que as respectivas instâncias vão ter entre si. É representada por uma seta na associação entre as classes. A classe na origem tem visibilidade sobre a classe do destino, sem reciprocidade. Na Figura 3.10, a classe Autor tem visibilidade sobre as instâncias de Curso, ou seja, tem referências a instâncias de curso. Por sua vez, Curso não tem nenhuma visibilidade sobre instâncias de autor.

Figura 3.10- Navegabilidade entre duas classes

3.1.2.2.5-Generalizações-especializações

São associações entre classes que guardam entre si relações de herança, ou seja, uma classe pode herdar de outra, atributos e operações que lhe sejam visíveis (que sejam de nível de encapsulamento protegido ou público). Na Figura 3.11 temos um exemplo em que 3 classes (Autor, Leitor, Revisor) herdam características da classe usuário. *Discriminador* corresponde a um nome que se dá para a relação de herança dentro do contexto do domínio de problema. Por exemplo, no contexto da Revista Digital, poderia ser *papel. Restrições* podem ser alocadas à generalização, colocadas entre chaves, sobre uma linha pontilhada (ver Figura 3.11). Podem ser dos seguintes tipos:

{Sobreposição} : subclasses podem ocorrer simultaneamente.
{Disjunção}: subclasses são mutuamente exclusivas.
{Completo}: todas as subclasses estão representadas.
{Incompleto}: algumas subclasses foram representadas.

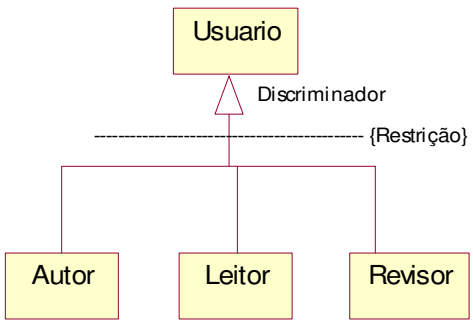


Figura 3.11- Estrutura Generalização-Especialização

A herança múltipla pode ser representada conforme mostrado na Figura 3.12. A classe TextBox herda características de um editor de textos e de uma janela.

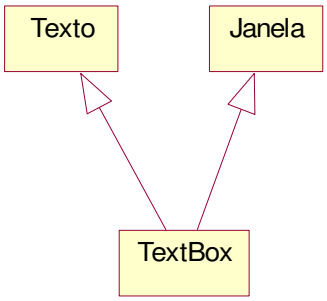


Figura 3.12-Estrutura Generalização-Especialização para herança múltipla

3.1.2.3-Mecanismos de extensão

Apesar de bastante completa, a UML não é capaz de representar todas as nuances de um ambiente ou modelo em todos os domínios possíveis. Por esse motivo a UML possui os chamados mecanismos de extensão, descritos a seguir.

3.1.2.3.1-Regras de restrição

A UML provê *regras de restrição* a serem aplicadas a classes e seus constituintes (atributos e procedimentos).As associações, atributos e generalizações em si já especificam restrições, mas nem todas podem ser expressas por meio de tais estruturas. A UML não define uma sintaxe rígida para especificar restrições, apenas determina que devem ser colocadas entre chaves({}).
Estas restrições podem ser definidas utilizando-se linguagens informal ou formal, predicados ou fragmentos de código-fonte. Na Figura 3.13 abaixo temos um exemplo de uso das mesmas. O valor do atributo ComRes sendo 3, determina a colocação da data de hoje (DATAHOJE) no campo DocDatApr. (No contexto do sistema Revista Digital corresponde a alocar a data de hoje à data de aprovação do documento, caso ele tenha sido aprovado na última revisão).

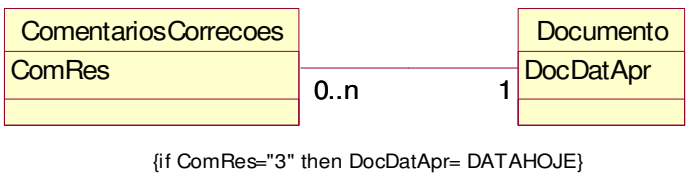


Figura 3.13-Regra de Restrição aplicada aos atributos ComRes e DocDatApr. Se o valor de ComRes for "3" faz DocCatApr ser igual a DATAHOJE

3.1.2.3.2-Estereótipos

O *estereótipo* é uma extensão ao vocabulário da UML cujo objetivo é a criação de novos blocos de construção a partir dos existentes [BOO99]. A forma mais comum é a utilizada para designar determinada classificação de alto nível para objetos ou classes que possuam naturezas ou objetivos afins. Por exemplo,na abordagem original de Jacobson, objetos são classificados em três tipos: objetos de interface, objetos de controle e objetos-entidades. São sugeridas regras para comunicação entre estes objetos e cada um é representado por um ícone diferente. Na UML isto não existe e estes tipos de objetos tornaram-se estereótipos.
Estereótipos são usualmente mostrados entre os sinais "<<" e ">>", como por exemplo, em <<controle>> (ver Figura 3.14).

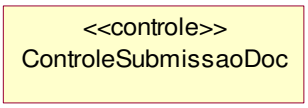


Figura 3.14-Representação de estereótipo de classe de controle.

Um outro exemplo notável de estereótipo é o <<metaclasse>>, que denota a classe da qual suas instâncias também são classes.

3.1.2.4-Classificação Múltipla

A classificação múltipla refere-se à situação em que uma classe mãe possui classes derivadas em contextos diferentes. Utiliza-se o discriminador para designar o contexto em que uma herança específica está incluída. Na Figura 3.16, no contexto do cargo em que ocupa na empresa, Funcionário pode ser classificado como Analista, Programador e Gerente. Já no contexto do departamento de recursos humanos, definem-se subclasses para discriminar a forma na qual a contratação do funcionário foi efetuada, ou seja, como Celetista ou Consultor.

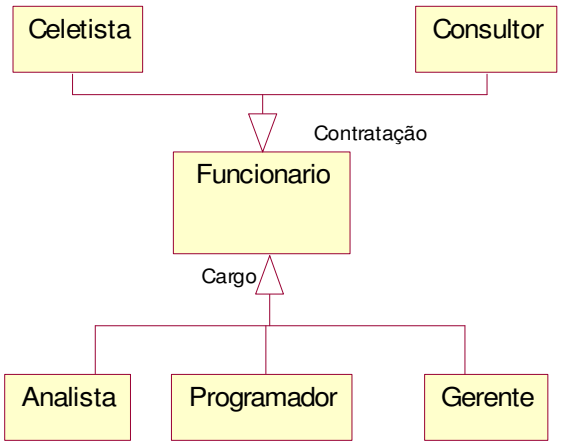


Figura 3.16- Exemplo de classificação múltipla

3.1.2.5-Classificação Dinâmica

Ocorre quando há a possibilidade de um objeto modificar seu tipo na estrutura de subclasse. Na Figura 3.17 a seguir, o funcionário pode modificar seu cargo na empresa, evoluindo de programador para analista e deste para gerente. O denotador de estereótipo <<dinâmica>> designa este tipo de classificação.

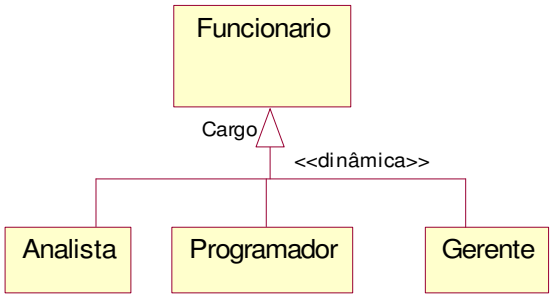


Figura 3.17- Exemplo de classificação dinâmica

3.1.2.6-Notação para classes abstratas

Já vimos que classes abstratas não possuem instâncias. Elas são denotadas colocando-se a palavra *abstrata* entre chaves ({abstrata}) logo abaixo do nome da classe. Na Figura 3.18, a classe BasePersistencia é abstrata, ou seja, não existem instâncias de objetos dela derivados.

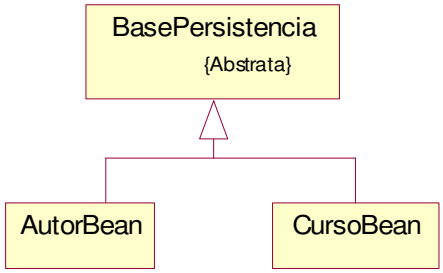


Figura 3.18- Exemplo de classe abstrata

3.1.2.7-Notação para interfaces

No contexto de linguagens de programação, uma interface é uma classe sem implementação, não possuindo atributos e contendo apenas a declaração das operações. Uma ou mais classes implementam as operações especificadas na interface. A notação para designar uma interface coloca o estereótipo <<interface>> sobre o nome da mesma. Na Figura 3.19, a interface Janela é implementada pelas classes Caixa e Diálogo. Repare na associação pontilhada: corresponde à representação para implementação da interface. Na Figura 3.20 apresentamos outra notação para interface: a classe ConnectionHolder implementa a interface HttpSessionBindingListener.

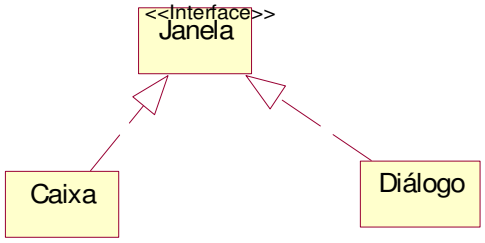


Figura 3.19 - Notação para interface e implementação da interface

Figura 3.20 – Outra notação para Interface

3.1.2.8-Classes Associativas

As classes associativas são derivadas de uma associação.Neste caso, esta possuirá atributos e operações. Classes associativas ocorrem na situação em que a associação entre as classes existe no domínio do problema. Na Figura 3.21, existe um relacionamento 0 ou 1 para muitos entre Empresa e Pessoa. No domínio do problema, esta relação é o Emprego da pessoa (classe associativa), caracterizado pelos atributos Cargo e Salário.

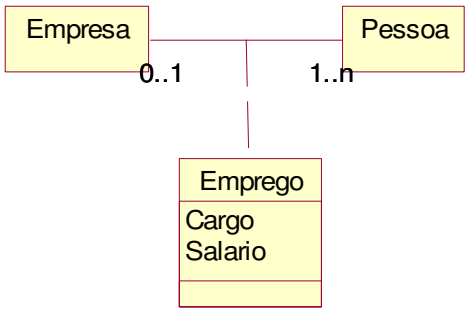


Figura 3.21-Classe associativa

3.1.2.9-Classes Parametrizadas

Uma classe parametrizada é um gabarito para criação de classes que seguem seu formato. Ela declara parâmetros padrões que permitirão a sua adaptação a contextos específicos. Na Figura 3.22 temos sua notação.

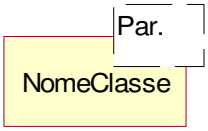


Figura 3.22-Classe Parametrizada.

3.1.2.10-Atributos Derivados

Atributos Derivados são aqueles que podem ser calculados a partir de outros atributos. Por exemplo, um atributo *idade* pode ser derivado a partir do atributo *data de nascimento.* Tais atributos são denotados por uma barra (/) colocada antes de seu nome.A forma de derivação pode ser explicitada a partir de uma regra de restrição alocada ao pé da classe.

3.1.2.11- Instanciação

Esta notação representa a possibilidade de uma classe instanciar outra. A seta tracejada parte da classe que cria a instância.

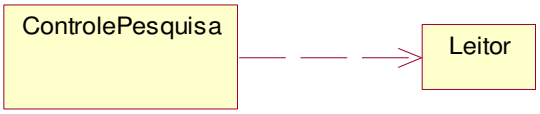


Figura 3.23-Classe ControlePesquisa instancia classe Leitor

3.1.2.12-Alocando notas ao diagrama

É possível alocarem-se notas e observações no diagrama de classes. Na Figura 3.24 temos uma nota que se refere a um pseudocódigo alocado a uma operação.

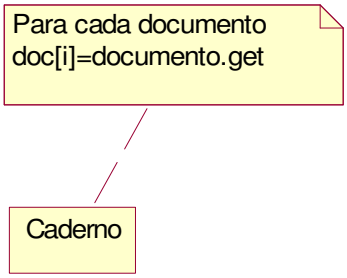


Figura 3.24-Nota de pseudocódigo alocada a uma classe

3.2-UML – Diagramas de Pacotes

Pacotes são artefatos da UML que permitem dividir um determinado modelo de classes em subsistemas. Cada um destes subsistemas contém classes que guardam afinidades funcionais entre si e este é o critério utilizado para particionamento. Os diagramas de pacotes denotam as relações de dependência entre os diversos pacotes do sistema: setas tracejadas conectam pacotes entre si, de forma que o pacote na origem acessa classes do pacote do destino.
Pacotes são uma ferramenta vital a projetos grandes. Devem ser utilizados quando um diagrama de classes que englobe todo o sistema não for visível em uma folha A3. Podem ser mostrados em diagramas de classes, quando deseja-se evidenciar que classes acessam pacotes contidos na aplicação.
Na Figura 3.23, os pacotes Controle, Interface e Banco de Dados contêm classes que implementam as camadas respectivas da arquitetura 3 camadas. O pacote Controle visualiza as classes do pacote Interface e o Pacote Banco de Dados é visualizado pelos outros dois pacotes.

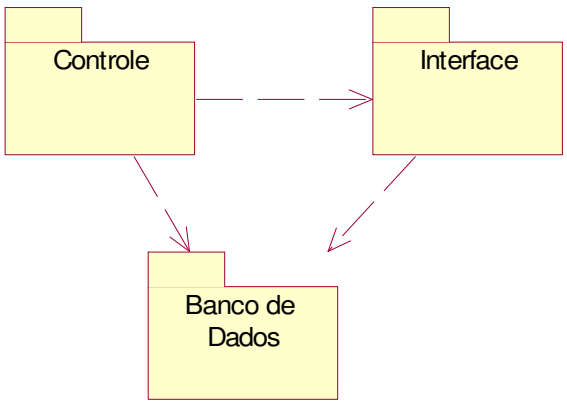


Figura 3.23 – Exemplo de Diagrama de Pacotes

3.3-UML-Diagramas de Transição de Estados(DTE)

Os Diagramas de Transição de Estados descrevem os estados possíveis que um objeto em particular pode assumir e como estes estados são modificados em função de eventos que afetam o objeto. Diagramas de Estados são desenhados para uma única classe para mostrar o comportamento de um único objeto.

A Figura 3.24 exemplifica um DTE para o objeto Documento da Revista Digital. Nele vemos as notações para estado, eventos, ações e condições. O nome do estado é colocado no topo do retângulo de bordas arredondadas. Sob este nome pode-se colocar, separada dele por uma linha horizontal, uma ação que esteja sendo realizada enquanto o objeto estiver naquele estado. No exemplo, o objeto está no estado "Submetido esperando definição de revisores" e, enquanto nele, a ação "Determinar revisores" está sendo executada.Repare que a ação é precedida por "do/". Temos uma transição deste último estado para "Aguardando Revisão", que ocorre quando a condição "Revisores definidos" (colocada entre colchetes) é verificada. Esta condição dispara a ação "Enviar para revisão" , colocada sob a barra ("/"). Por convenção, este tipo de ação ocorre instantaneamente. Entre estados podem ocorrer eventos que provocam transição, como o evento "Revisor Iniciar Revisão" mostrado na transição entre "Aguardando Revisão" e "Em revisão". No evento "Em Revisão" temos um exemplo de estrutura de decisão: a execução da ação "Efetuar Revisão" pode causar a transição para o estado "Revisão Concluída Reprovado", caso a condição "Documento Reprovado" ocorra, ou para "Revisão Concluída Aprovado", caso a condição "Documento aprovado"' seja satisfeita. Repare também nos ícones utilizados para determinar o início e o fim do diagrama.
DTEs podem ser multinivelados, ou seja, um estado pode dar origem a um subdiagrama contendo subestados pelos quais um objeto passa (denominam-se tais estados "aninhados").

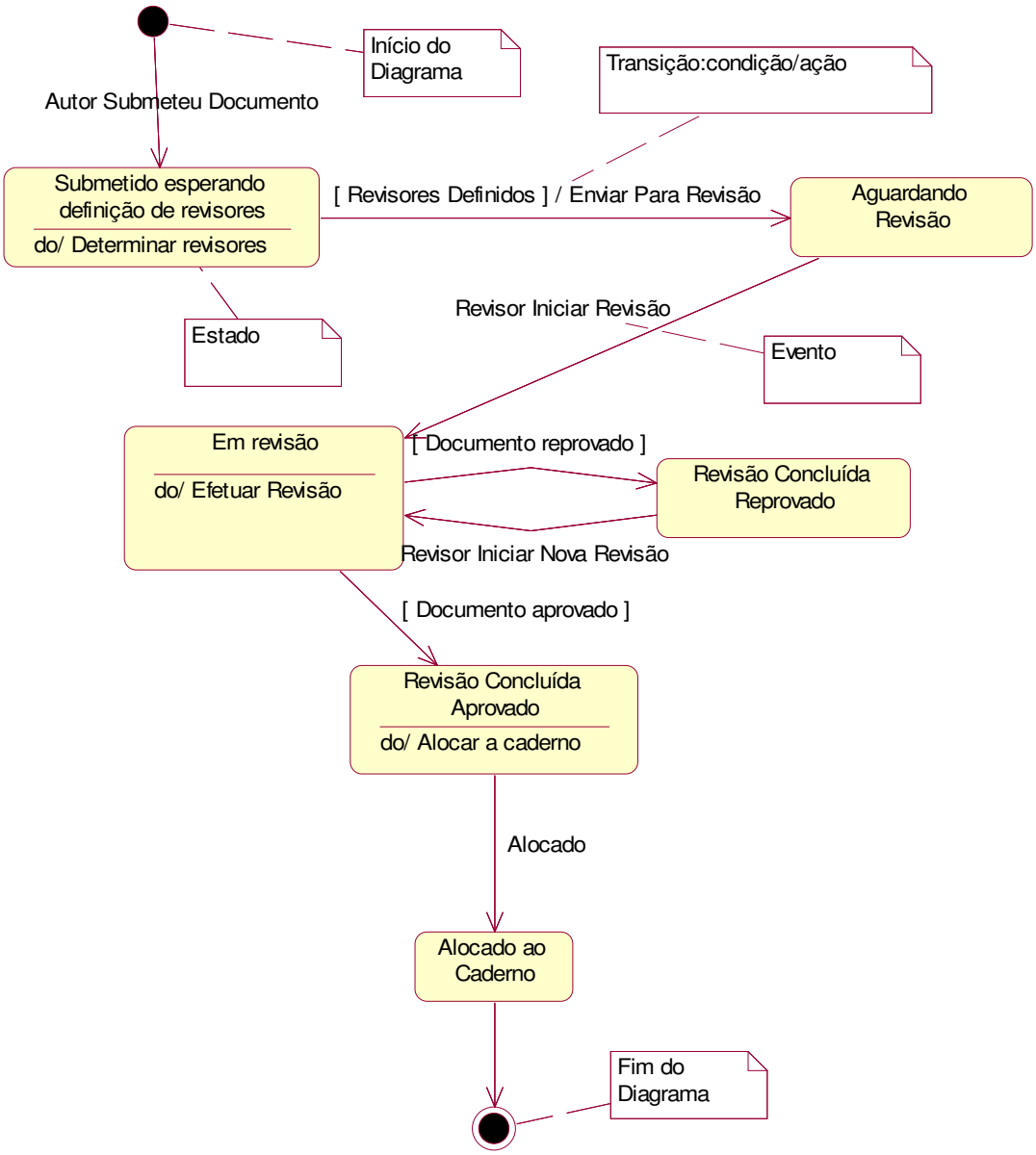


Figura 3.24-DTE que mostra o ciclo do objeto Documento

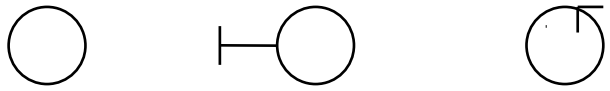
3.3 –Conceito: Estereótipos de Jacobson

A arquitetura de classes utilizada neste trabalho embasa-se em *estereótipos* de classes propostos por Ivar Jacobson [JAC94] que permite uma organização criteriosa das classes, de forma que a atividade de desenvolvimento possa ser particionada e distribuída entre equipes distintas. Os estereótipos são os seguintes:

-*Classes entidade* – são classes persistentes, normalmente pertencentes ao domínio da aplicação (ou seja, ao contexto da aplicação) e, na maioria das situações, mapeadas a tabelas do banco de dados.
-*Classes de interface*- implementam a interação entre o sistema(casos de uso) e os atores do mesmo.
-*Classes de controle* – contêm métodos que implementam o fluxo de eventos de um caso de uso, ou seja, possuem encapsuladas todas as regras de negócio da aplicação e, ainda, podem despachar requisições do usuário para ações a serem realizadas pelo sistema.

Parece bastante óbvio que estes estereótipos estão relacionados, respectivamente, às camadas de persistência, de apresentação e de negócio do modelo 3 camadas descrito no primeiro capítulo.

A notação da UML aceita a colocação de ícones para representação de estereótipos. Os propostos por Jacobson são apresentados na Figura 3.25.



Objeto Entidade Objeto de Interface Objeto de Controle

Figura 3.25 – Ícones utilizados para estereótipos de Jacobson

3.4-Atividade definir classes de domínio do problema

Esta atividade, conforme visto na Figura 2.1, pode ser realizada logo após a definição dos casos de uso expandidos. As classes de domínio de problema são *conceitos* pertencentes ao mundo real, que participam ativamente na consecução de tarefas neste domínio.

3.3.1-Obtendo classes de domínio do problema por categorias

Classes de domínio de problema podem ser obtidas através de um "brainstorm" inicial, em que um grande número de candidatas são capturadas dentro das categorias mostradas a seguir. A ordem apresentada sugere inicialmente as categorias onde estão normalmente o maior número de classes prováveis de se tornarem classes de domínio de problema:

Papéis realizados dentro do contexto da aplicação: autor, revisor, administrador
Coisas físicas ou tangíveis: documentos, caderno
Coisas intangíveis: revisão
Lugares: loja, secretaria

Organizações: Departamento de vendas
Relatórios: administrativos, gerenciais, etc.

As classes obtidas são avaliadas para:

-Verificar se são realmente relevantes no domínio do problema.
-Verificar se não são atributos de outros conceitos. [LAR00] sugere que "se um conceito no mundo real não for pensado como um texto ou um número, então muito provavelmente é uma classe".

Como exemplo, seja a descrição do caso de uso Fazer Revisões de Documentos:

Ações do ator	**Resposta do Sistema**
1-O revisor seleciona de uma lista (que contém somente os documentos alocados ao mesmo) o documento a revisar.	2-Abre tela para registro de revisão do documento.
3-O revisor pode:	
3.1-Solicitar download do documento.	3.2-Iniciar *Fazer Download Documentos.*
3.3-Preencher tabela, atribuindo nota a quesitos de avaliação.	
3.4-Registrar resultado da avaliação: *aceito, não aceito, aceito com nova revisão, não aceito com nova revisão.*	
3.5-Preencher campo contendo comentários e correções.	
3.6-Preencher data da revisão.	
4-Finalizada a revisão, o usuário pode:	
3.a.1-Submeter a revisão ao sistema.	3.a.2-Se revisão tiver status de *aceita,* registra no documento sua data de aprovação. 3.a.3-Grava revisão.
3.b.1-Criar uma nova revisão do documento.	3.b.2-Se revisão tiver status de *aceita,* registra no documento sua data de aprovação. 3.b.3-Grava revisão. 3.b.4-Abre tela para registro da nova revisão e inicia novamente este caso de uso.

Identificamos as seguintes categorias de substantivos e as respectivas classes candidatas:

Papéis: revisor

Coisas físicas: documento,tela
Coisas intangíveis: download, revisão, quesitos de avaliação, resultado da avaliação, campo comentários e correções, data da revisão, sistema.
Revisor é um ator, mas certamente será uma classe do sistema, pois deverá estar cadastrado no sistema e estará ligado a revisões de documento.
Documento será uma classe do sistema pois é um dos principais conceitos do mesmo, a razão de ser da revista.
Tela não será uma classe do sistema porque é uma apenas um dispositivo de interface de interação, como também o é um mouse, um teclado, etc.
Download: nesta primeira abordagem é uma classe do sistema, pois representa um conceito importante do domínio do problema.
Revisão é um conceito importante do sistema. É classe do mesmo.
Quesitos de avaliação, resultado da avaliação, campo comentários e correções e data da revisão não têm muita razão para que sejam considerados classes: são melhor alocados como atributos de outra classe, muito provavelmente da classe *revisão.*
Sistema obviamente deve ser descartado, pois não pode ser uma classe de si mesmo.

Portanto, da análise deste caso de uso obtivemos as classes *Revisor, Documento, Download* e *Revisão.*

3.3.2-Obtendo classes de domínio de problema por substantivos

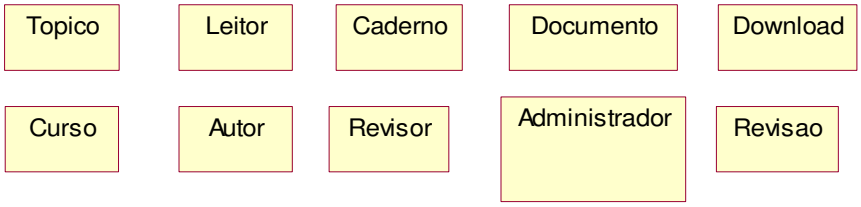
Substantivos presentes nas descrições de casos de uso podem ser capturados e utilizados como abordagem inicial para obtenção de classes de domínio de problema. Dois cuidados devem ser tomados em se utilizando esta abordagem:

-Palavras em linguagem natural podem ser ambíguas;
-Substantivos podem ser também atributos de classes.

3.3.3-Definindo as classes de domínio que serão persistentes(classes *entidade)*

As classes de domínio de problema cujo estado dos objetos deva ser preservado são classificadas como persistentes (entidade).A maior parte das classes de domínio de problema certamente será persistente. Estas classes encapsulam apenas métodos para armazenamento e recuperação de informações.

No nosso exemplo temos que, a partir da descrição estendida dos casos de uso, identificamos as classes de domínio de problema mostradas na abaixo .Utilizamos o processo descrito no item 3.3.1.



3.5 –Atividade Produzir diagrama de classes de domínio do problema

O diagrama de classes de domínio de problema é um excelente artefato para a representação de conceitos com os quais o sistema vai trabalhar, representando seus atributos e o relacionamento entre eles. Nos subitens a seguir detalhamos os passos necessários à consecução desta atividade.

3.5.1-Adicionar associações

Criam-se associações para determinar dependências entre classes, ou seja, a necessidade que determinada classe tem de *conhecimento* acerca de instâncias de outras classes do modelo. Por exemplo, a classe *Revisão* necessita de conhecimento acerca de instância da classe *Documento,* pois uma *Revisão* somente tem sentido quando vinculada a um *Documento.* Um *Documento,* por sua vez, pode estar relacionado a mais de uma instância de *Revisão.* Portanto, vislumbra-se desta forma uma *associação de dependência* entre *Revisão* e *Documento.* Em nível de projeto conceitual, adicionamos as seguintes categorias de associações(inspirado em [LAR00]):

Categoria	Descrição
Conhecimento necessário	Uma classe necessita conhecer a ocorrência de outra.Ex: Autor e Documento
Todo e parte	Uma classe é uma parte lógica ou física de outra, está contida em outra ou ainda, é possuída por outra. Ex: Caderno e Documento
Uso	Uma classe utiliza algo de outra. Ex: Item de Pedido e Produto (Item de Pedido usa o preço de Produto)
Dependência	Uma classe depende de outra para que seu conceito faça sentido.Ex: Revisão e Documento
Comunicação	Uma classe se comunica com a outra.

	Ex: Cliente e Caixa
Objeto-Transação	Um objeto se relaciona com uma determinada transação. Ex: Cliente (objeto) com Pagamento (transação)
Inter-transações	Uma transação se relaciona com outra. Ex:Pagamento com Venda
Classe associativa	Uma classe define a associação entre duas ou mais outras. Ex: Revisão é uma classe associativa entre Revisor e Documento

Após definirem-se as associações, deve-se explicitar nos extremos o número de instâncias da outra classe com as quais a classe em consideração se relaciona (multiplicidade ou cardinalidade). Pode-se também nomear uma associação, adotando-se como formato padrão uma frase verbal ou um verbo. Na Figura 3.26 temos a associação entre *Revisor* e *Documento.* Repare na cardinalidade entre as classes: Revisor revisa vários Documentos e vice-versa. A associação foi nomeada de forma a incrementar o nível de compreensão do contexto da mesma.

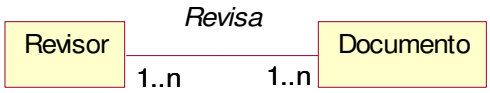
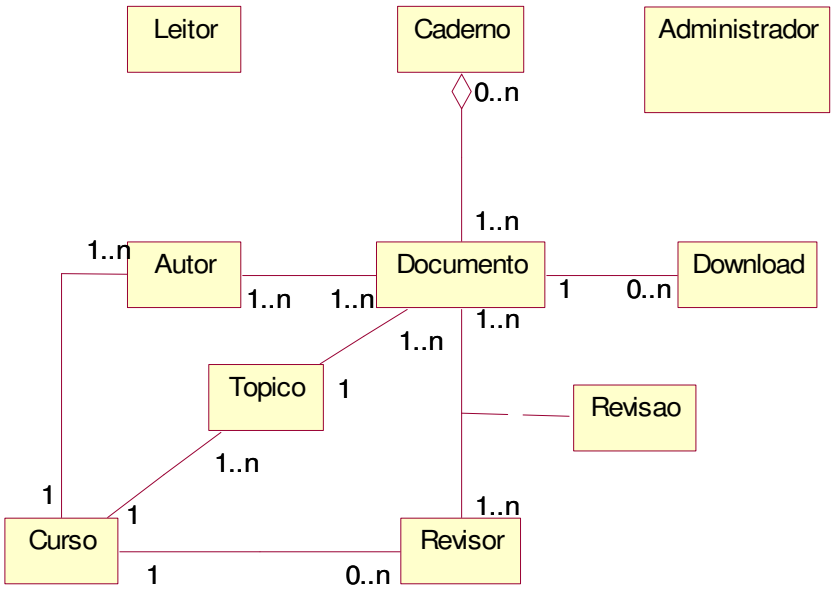


Figura 3.26- Uma associação nomeada e com cardinalidade definida

No exemplo Revista Digital, identificamos as seguintes associações entre as classes:

Categoria	**Associações**
Conhecimento necessário	Autor e Documento, Documento e Download, Autor e Curso, Curso e Tópico, Curso e Revisor,Tópico e Documento
Todo e parte	Caderno e Documento
Classe Associativa	Revisão associa Revisor e Documento

A figura abaixo mostra o diagrama de classes de domínio de problema obtido após a adição das associações acima definidas:



3.5.2-Adicionar atributos

Atributos são características cujos valores determinam o estado atual de um objeto, ou seja, identificam uma instância de uma classe. Em um grande número de situações, classes possuem um número considerável de atributos, e a tarefa do analista consiste em determinar aqueles que sejam relevantes no domínio do problema.
Atributos devem ser preferencialmente tipos de dados simples e é o que ocorre na maioria esmagadora das vezes. Quando forem compostos, ou seja, se dividem em dois ou mais atributos, é melhor considerá-los como classes à parte.
Atributos não devem ser usados como chave de associação entre classes, ou seja, como chave estrangeira, tal como acontece em um modelo de dados relacional, em um diagrama de classes. No próximo item, quando descrevermos o processo de mapeamento para um modelo de dados, veremos o aparecimento de tais chaves estrangeiras.
Toda classe deve possuir um atributo que designe a identidade de suas instâncias. Normalmente, tais atributos são códigos ou nomes. As boas práticas de modelagem determinam que uma classe deve possuir um único atributo identificador, de forma a se diminuir a probabilidade de ocorrência de inconsistências e ambigüidades.

Na figura abaixo temos a versão do diagrama de classes do exemplo,acrescido dos atributos. Repare que não existem chaves estrangeiras no diagrama de classes. Elas aparecem após o mapeamento do mesmo para banco de dados.

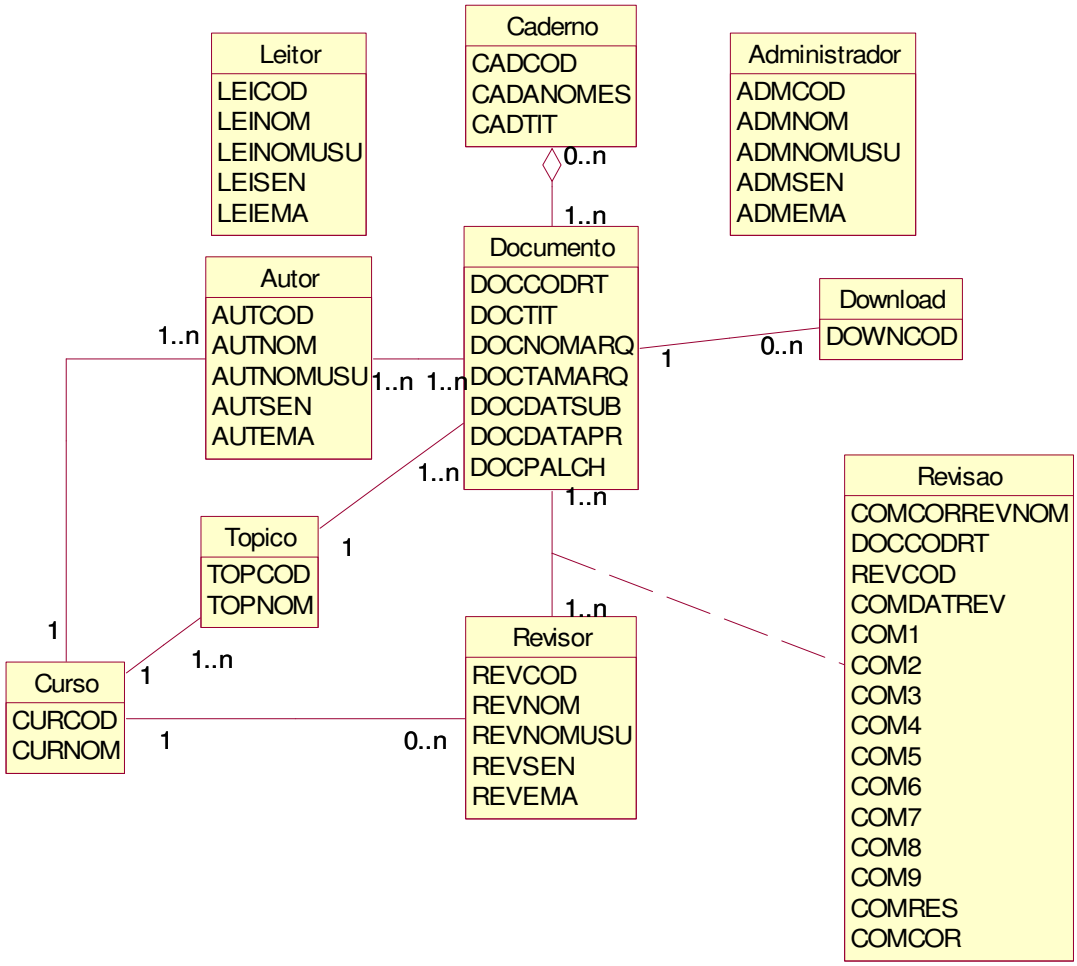


Diagrama de Classes de Domínio de Problema com atributos

3.6 –Atividade: Produzir DER a partir do diagrama de classes de domínio do problema

Neste ponto, os objetos de domínio de problema persistentes já estão definidos e relacionados entre si. Podemos então, criar o Diagrama de Entidade-Relacionamento do sistema, utilizando técnicas simples de mapeamento entre os diagramas, aqui descritas.
A necessidade de armazenamento persistente na forma de um banco de dados relacional é indicada pelos seguintes fatos[JAC94] :

-Informação contida nos objetos necessita ser persistente
-Mais de uma aplicação compartilha (parte dos) dados
-Estruturas de informação possuem grande número de instâncias
-Existem buscas complexas na estrutura de informação
-Necessidade de geração avançada de relatórios a partir da informação armazenada

-Manipulação freqüente de transações do usuário

Deve-se decidir a partir do diagrama de classes de domínio do problema quais objetos devem ser persistentes, ou seja, que devem ter seu conteúdo de informação armazenado. Os objetos entidade são os candidatos naturais ao armazenamento persistente.

3.6.1-Bancos de dados relacionais

3.6.1.1-Problemas

Quando armazena-se informação na forma de tabela do modelo relacional, alguns problemas surgem, quais sejam:

-Somente armazenam-se dados, nunca comportamento
-Estruturas complexas têm que ser decompostas em tipos de dados primitivos (problema da impedância)
-Cria-se um forte acoplamento entre a aplicação e o banco de dados
-Como representar a herança

3.6.1.2- Representando objetos na forma de tabelas

Uma classe é mapeada em tabelas da seguinte maneira:

-Alocar uma tabela à classe
-Cada atributo primitivo tornar-se-á uma coluna na tabela. Se o atributo for complexo, adiciona-se uma tabela para o mesmo, ou pode-se parti-lo em muitas colunas na tabela da classe.
-A chave primária será o único identificador da instância, ou seja, será o identificador pelo qual a instância será unicamente reconhecida. O identificador deve ser preferencialmente invisível ao usuário, pois mudanças das chaves por razões administrativas não devem afetar o mesmo; chaves geradas automaticamente devem ser preferencialmente utilizadas.
-Cada instância da classe será agora representada por uma chave nesta tabela.
-Relações 1 para muitos serão representadas por uma coluna de chave estrangeira na classe de maior cardinalidade.
-Relações muitos para muitos serão representadas por uma tabela contendo as chaves primárias importadas das tabelas que a geraram, como chaves primárias e/ou estrangeiras.
Com relação à normalização, tem-se o seguinte: um projeto de bancos de dados baseado em um modelo de objetos está na terceira forma normal. Isto ocorre porque na terceira forma normal, cada linha consiste de um identificador com um número de atributos mutuamente independentes. Como o que normalmente se tem em um modelo de objetos é um objeto contendo um identificador único e um certo número de atributos não dependentes uns dos outros, concluímos a afirmação anterior.A normalização, entretanto, pode

ocasionar em muitas situações, problemas de performance. Quando não for possível utilizarem-se tabelas de indexação especiais, deve-se desnormalizar a base de dados para incrementar a performance. Isto vai, naturalmente , causar os clássicos problemas de redundância na base de dados, porém este problema é minimizado no modelo de objetos, já que esta redundância vai estar encapsulada no interior do objeto, portanto, livre do conhecimento do programador.

3.6.1.1-Mapeamento de Herança

Para resolver o problema da herança, duas soluções são possíveis:

(1) Os atributos herdados são copiados para as tabelas que representam as classes descendentes. Não há tabela para a classe abstrata.
(2) A classe abstrata é representada por uma tabela, para a qual as tabelas das classes descendentes fazem referência através de chave primária.

3.7-Atividade Produzir Protótipo da Interface

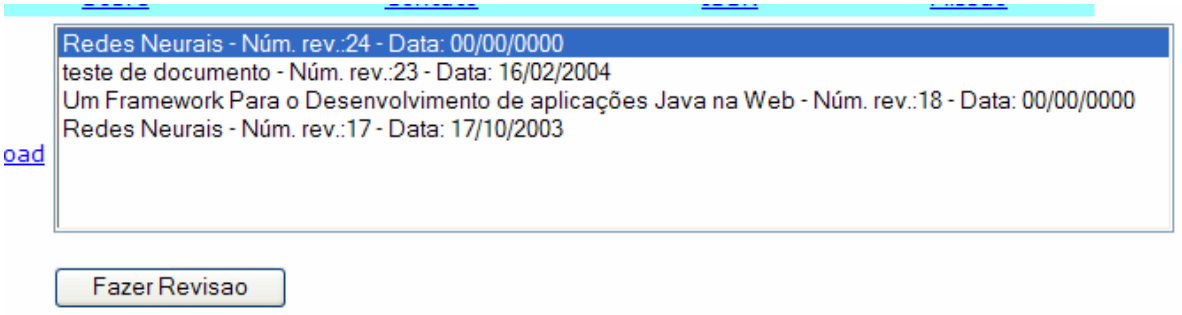
Quando da descrição dos casos de uso e da comunicação dos mesmos aos usuários em potencial, é importante descrever as interfaces em maior detalhe. Se for uma interface homem-máquina pode-se mostrar o que o usuário visualizará na tela quando executando o caso de uso ou prover simulações mais sofisticadas utilizando-se uma ferramenta específica para isso. Desta forma podemos simular os casos de uso da forma como ele aparecerá aos usuários antes mesmo de pensar em como realizá-los.Pode-se portanto, alocar às descrições dos casos de uso a interação real dos usuários com o sistema, o que certamente eliminará vários mal-entendidos. Deve-se frisar que quando da definição de interfaces com usuários estes devem estar totalmente envolvidos. Esta interface deve refletir a visão lógica do usuário com relação ao sistema. Em nível de protótipo executável, podem ser documentos html ou jsp, formulários Delphi , Java ou VB, etc.
As interfaces produzidas nesta atividade serão utilizadas na descrição em nível real dos casos de uso, apresentada no próximo item.
No processo de codificação da interface proposto no Capítulo 6, veremos que a produção do protótipo em HTML ou JSP é mais produtiva e, por conseguinte, fortemente recomendada.
Deve-se ressaltar que é nesta atividade que o papel do designer de interface é mais relevante: o protótipo deve refletir o aspecto final da aplicação.

3.8-Atividade Refinar Casos de Uso: produzir casos de uso reais

Os casos de uso reais possuem descrições mais minuciosas de sua lógica, incluindo nas descrições expandidas, interações com os atores por meio de interfaces e soluções comprometidas com tecnologias . Assim sendo, tais

descrições envolvem interações por meio de objetos de interface, como botões, links, tabelas, displays, etc. Abaixo temos a descrição real do caso de uso Registrar Revisões de Documentos , com o qual o ator Revisor interage através da interface mostrada na figura abaixo.

Ações do ator	Resposta do Sistema
1-O revisor seleciona de uma lista (que contém somente os documentos alocados ao mesmo) o documento a revisar.	2-Abre tela para registro de revisão do documento.
3-O usuário pode:	
3.1-Solicitar download do documento.	3.2-Iniciar *Fazer Download Documentos.*
3.3-Preencher tabela, atribuindo nota a quesitos de avaliação.	
3.4-Registrar resultado da avaliação: *aceito, não aceito, aceito com nova revisão, não aceito com nova revisão.*	
3.5-Preencher campo contendo comentários e correções.	
3.6-Preencher data da revisão.	
4-Finalizada a revisão, o usuário pode:	
3.a.1-Submeter a revisão ao sistema.	3.a.2-Se revisão tiver status de *aceita,* registra no documento sua data de aprovação. 3.a.3-Grava revisão.
3.b.1-Criar uma nova revisão do documento.	3.b.2-Se revisão tiver status de *aceita,* registra no documento sua data de aprovação. 3.b.3-Grava revisão. 3.b.4-Abre tela para registro da nova revisão e inicia novamente este caso de uso.



1-*Um Framework Para o Desenvolvimento de aplicações Java na Web* Download : linha.doc

Resumo:
Ambientes de Engenharia de software e suas aplicações.

Apreciação Geral:	**Notas**
O Tema possui interesse científico-tecnológico	3
O Resumo é suficientemente descritivo	3
O Assunto é atrativo para publicação na Revista Digital	3
Originalidade:	
O trabalho apresenta um avanço importante	3
Há novos resultados, novos métodos ou novos dados	3
Há aplicações relevantes de teorias e métodos	3
Clareza:	
O título descreve adequadamente o conteúdo	3
Existem erros aparentes de fatos ou lógica	3
O texto é ortograficamente e gramaticalmente correto	3

Resultado da revisão:

- ○ Relatório pode ser aceito
- ○ Pode ser aceito mas com revisão sugerida abaixo, não requerendo nova revisão
- ○ Não aceitável como se apresenta, necessita de maior revisão abaixo, requerendo revisão posterior
- ○ Não aceitável

Comentários e correções:

Data desta revisão: 00/00/0000

Gravar revisão | Gerar Nova Revisão

Figura – Interface HTML do caso de uso Registrar Revisão de Documentos

3.9-Atividade Definir Classes de Interface

Toda funcionalidade especificada nas descrições do caso de utilização que seja diretamente dependente do ambiente do sistema é abstraída como classe de interface. É através destes objetos que os atores comunicam-se com o sistema.
A tarefa de uma classe de interface é traduzir as entradas dos atores ao sistema em eventos no sistema e traduzir os eventos do sistema nos quais o usuário esteja interessado em algo que é apresentado ao ator.
As linguagens visuais (Delphi, C++ Builder por exemplo), retiram do desenvolvedor a responsabilidade pela criação de classes de interface. Entretanto, o desenvolvimento para internet, pressupõe na parte das vezes, a utilização de formulários HTML, de JSP ou de ASP, por exemplo, que se não utilizados convenientemente, tornam a estrutura do projeto difícil de ser mantida.
As classes de interface serão definidas segundo uma arquitetura que encapsule métodos para estruturação e apresentação de interfaces, segundo os seguintes princípios:
-Detalhes de implementação são completamente omitidos do desenvolvedor, que tem ciência apenas de métodos e atributos para exibição e coleta de dados da interface.
-As classes de interface devem ser hierarquicamente estruturadas por meio de relações de herança. Em aplicações web especificamente, diversas páginas têm partes em comum (cabeçalho e rodapé, por exemplo), variando em outras (as partes centrais e laterais). Estas partes em comum são alocadas a classes mães e os detalhes específicos a classes derivadas.
As classes de interface, neste ponto, podem ser melhor definidas e estruturadas, pois as descrições reais já foram produzidas, detalhando a forma de interação com as interfaces.

3.10-Atividade Definir Classes de Controle

As classes de controle encapsulam todas as regras de negócio da aplicação, ou seja, implementam o fluxo de eventos da aplicação e, ainda, controlam as requisições do usuário, alocando-as a ações específicas do sistema. Para cada caso de uso identificamos uma classe de controle que encapsule estas atribuições.

3.12-Atividade Produzir Diagramas de Classes de Análise

O Diagrama de Classes de Análise é um refinamento realizado sobre o diagrama de classes de domínio do problema de forma a incorporarem-se níveis de abstração ditados pelas classes de controle e interface já vistos anteriormente. A visão proporcionada pelo diagrama de classes de análise é a da arquitetura preliminar proposta para o sistema, livre ainda de aspectos de

implementação. Os subitens apresentam os passos sugeridos para geração deste diagrama.

3.12.1-Refinando a estrutura de classes entidade

As classes entidade têm em comum o fato de serem responsáveis pelo armazenamento e recuperação de informações de banco de dados. Portanto seria útil em termos de reutilização, a definição no diagrama de classes de análise de uma classe base que encapsule métodos para armazenamento e recuperação de informações, quais sejam, *incluir, alterar,excluir e pesquisar* (no mínimo, mas outros podem ser criados, desde que colaborem no intuito de armazenar e recuperar informações). No framework mostrado na Figura 3.27, a classe *BasePersistencia* implementa os métodos supra citados e as classes entidade *Persistente1, Persistente2* e *PersistenteN* herdam tais métodos da mesma.

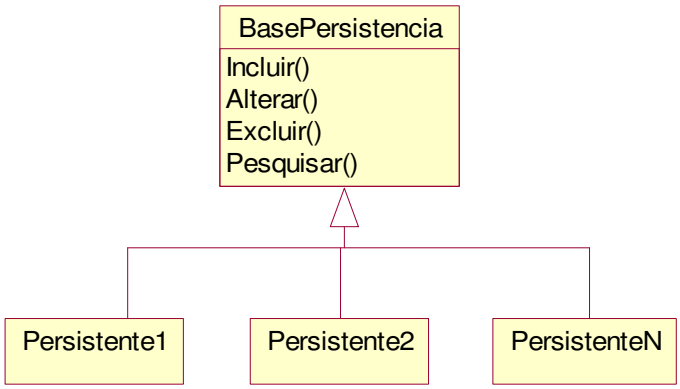


Figura 3.27 –Framework para classes entidade (persistentes)

3.12.2-Adicionando classes de controle

No diagrama de classes de análise, propomos a estruturação das classes de controle já definidas segundo o framework exibido na Figura 3.28.

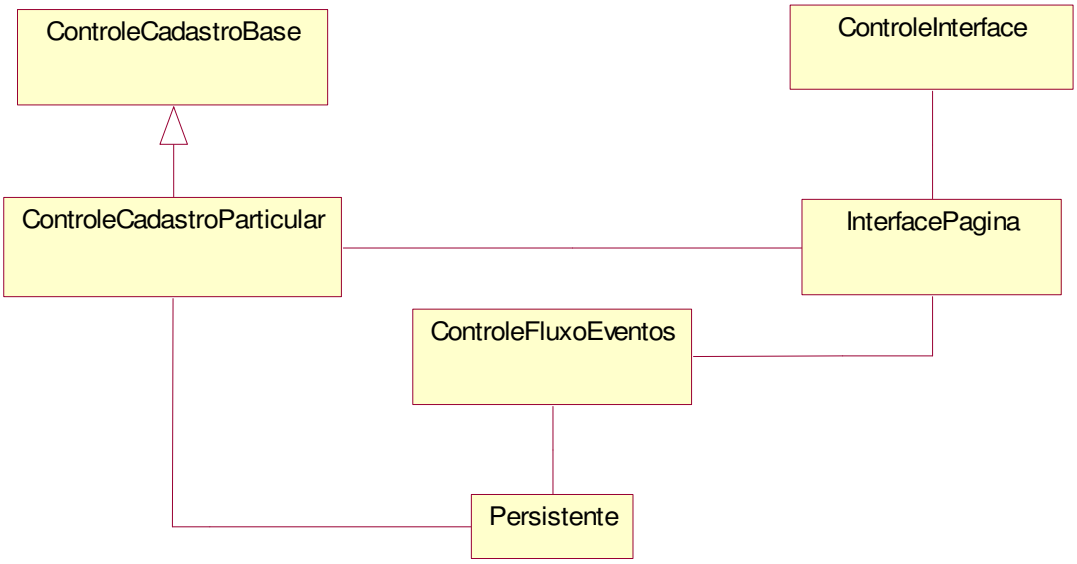


Figura 3.29 – Framework proposto para classes de controle.

A classe *ControleCadastroBase* é utilizada para encapsular aspectos comuns de classes de controle que controlem fluxos de eventos de casos de uso de cadastros de objetos do tipo *Persistente* já vistos. *ControleCadastroParticular* particulariza a classe *ControleCadastroBase* para uma classe *Persistente* específica. Por sua vez, *ControleFluxoEventos* possui métodos que implementam fluxos de eventos da aplicação e também se relaciona com objetos do tipo *Persistente.* A classe *ControleInterface* serve apenas para invocar classes de interface (*InterfacePagina*), que por sua vez invocam classes do tipo *ControleFluxoEventos.*

3.12.3-Adicionando classes de interface

Propomos a alocação de classes de interface segundo o framework mostrado na Figura 3.30, estruturado segundo os princípios citados no item 3.9. A classe *InterfaceBase* encapsula todos os métodos e atributos para exibição e coleta de dados. A classe *InterfaceEstruturaComum* particulariza *InterfaceBase* contendo a estrutura da página comum a todas as páginas da aplicação web. As classes *InterfacePagina1, InterfacePagina2* e *InterfacePagina3* acrescentam detalhes a *InterfaceEstruturaComum.* A classe *InterfacePagina21* herda detalhes da classe *InterfacePagina2*, particularizando alguma parte da mesma. No próximo capítulo descreveremos a implementação desta estrutura para Java-JSP.

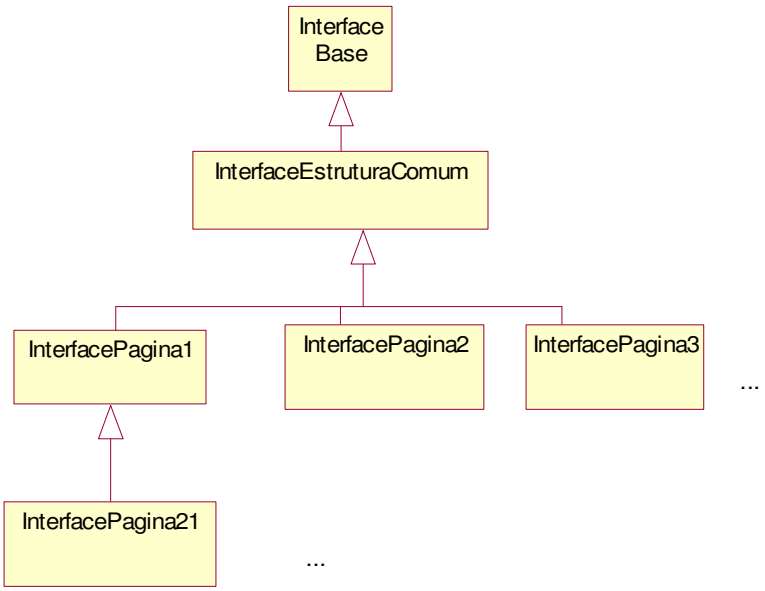


Figura 3.30 – Hierarquia genérica de classes de interface

3.12.4-Adicionando relações de herança.

Nos item 3.12.1, 412.2 e 3.12.3 propusemos frameworks que incorporam relações de herança, de forma a agregar funcionalidades e características comuns em classes mães, privilegiando o reutilização das mesmas.
No diagrama de classes de análise, devemos ainda identificar conceitos que possuam pontos comuns, de forma que possamos criar uma relação de herança dos mesmos, em relação a uma classe mãe que agregue tais pontos comuns (conceitos mais gerais) . Este tipo de relação é denominado *generalização-especialização(gen-espec)*. A classe mãe, mais geral e abrangente, é também denominada *supertipo* ou *classe base,* e as classes filhas, particularizadas, *subtipos* ou ainda *classes derivadas.*Vamos utilizar com mais freqüência os termos classe base e classe derivada. Por exemplo as classes *CompraaPrazo e CompraaVista* contêm conceitos comuns, que podem ser generalizados na classe base *Compra*. Assim, objetos das classes *CompraaPrazo e CompraaVista* são também objetos do classe *Compra.*
Uma classe derivada sempre vai conter todos os atributos, associações e operações da classe base. Se isto não ocorrer, a relação de herança não vai existir. Entretanto, muitas vezes, a divisão de uma classe base em classes derivadas apesar de semanticamente correta, pode não ser necessária.

Portanto, antes de criarmos um subtipo, devemos considerar o seguinte[MO95]:

-Se a classe derivada possui atributos, associações e/ou operações adicionais em relação à classe base. No exemplo citado, *CompraaPrazo* contém o atributo *NumeroParcelas* que lhe é particular;
-Se o conceito relacionado à classe derivada é tratado de maneira diferente ou se comporta de maneira diferente da classe base ou de outras classes derivadas. No exemplo citado, *CompraaPrazo* é um tratamento particular para uma *Compra,* diferente do tratamento dado a *CompraaVista.*

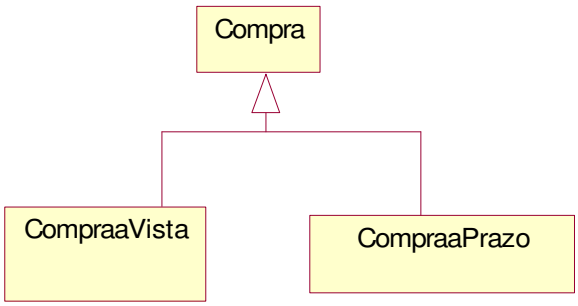


Figura 3.31-Relação de herança

Por outro lado, se temos várias classes derivadas, a generalização deve ser motivada pelo seguinte:

-As classes derivadas têm atributos, associações e/ou operações comuns que devem ser evidenciados em uma classe base;
-As classes derivadas são variações de um conceito comum expresso na classe base.

Os seguintes aspectos devem ser também considerados:

-*Classes abstratas*- Este conceito, definido no item 3.1.2.6, deve ser utilizado quando a classe base não possui instâncias típicas (ou seja *que não sejam instâncias das classes derivadas)*, existindo apenas para agregar os pontos comuns das classes derivadas. No exemplo citado, *Compra* certamente será classificada como abstrata, pois não existem objetos que não sejam instâncias de uma das duas classes derivadas (*CompraaVista* e *CompraaPrazo).*
-*Herança múltipla* – Classes derivadas podem herdar características de mais de uma classe base. Quando isto ocorrer, teremos uma situação de herança múltipla. Neste caso, instâncias das classes derivadas vão ser instâncias de todas as classes bases e todos os atributos, associações e operações das classes bases também pertencerão às classes derivadas.

3.12.4-Determinar visibilidade dos atributos (encapsulamento)

Um aspecto relevante de aplicações orientadas a objeto é a possibilidade de se ocultar ao desenvolvedor, informações que não lhe sejam importantes, disponibilizando apenas o relevante na consecução das responsabilidades de um determinado objeto. Neste passo, o analista determina o escopo de visibilidade de atributos. Os critérios sugeridos para classificar quanto à visibilidade, não somente atributos, mas também métodos (cuja obtenção será descrita no próximo capítulo), são os seguintes:

-*Públicos:* atributos e operações devem ser classificados como públicos quando serão utilizados em nível de instância de objetos, ou seja, a informação ou funcionalidade deve ser útil a quem quer que as utilize ou manipule.

-*Protegidos :* atributos e operações são protegidos quando seu escopo de visibilidade deve ser estendido a classes derivadas, mas não são relevantes a instâncias das classes bases e derivadas. É o caso em que um atributo ou operação são utilizados por outras operações na classe base, e devem ser igualmente utilizados nas classes derivadas.

-*Privados:* atributos e operações privados não são importantes nem a instâncias da classe nem a suas classes derivadas. São de escopo local à classe, funcionando como auxílio à realização de funcionalidades de outros métodos da mesma.

3.12.5-Definir pacotes

Uma vez identificadas as classes de análise, o sistema pode conter um grande número de classes. Para um projeto de tamanho médio, tipicamente de 30 a 100 classes serão especificadas. Para obter-se uma visão mais clara destes objetos, pode-se agrupá-los nos já apresentados *pacotes* (ver item 3.2). Este agrupamento pode ser feito em diversos níveis, ou seja, pacotes podem conter outros pacotes. Eles empacotam os objetos de forma a reduzir a complexidade.
O nível mais baixo de um pacote é denominado *pacote de serviço* e é considerado uma unidade de modificações atômica.
A divisão de um sistema em pacotes deve ser a priori baseada na funcionalidade do mesmo. Todas as classes que têm um acoplamento funcional mútuo forte devem ser colocados no mesmo pacotes. Deseja-se também um mínimo de comunicação entre os pacotes, de forma a privilegiar o baixo acoplamento.
Começamos procurando por pacotes opcionais.Outros pacotes são definidos a partir da funcionalidade do sistema: todas as classes envolvidas em uma parte específica da funcionalidade serão colocadas no mesmo pacote. Para identificar partes de funcionalidade um bom procedimento é remover uma determinada classe do modelo e verificar se ela e classes relacionadas são ou não supérfluos. Se forem, constituirão um mesmo pacote. Outros critérios que podem auxiliar na identificação de pacotes:
-Modificações em uma classe levam modificações em outra classe?
-Elas se comunicam com o mesmo ator?
-São dependentes de uma terceira classe, de interface ou entidade?

-Uma classe executa muitas operações de outra?
Aqui prevalece a idéia da alta coesão no pacote e baixo acoplamento entre pacotes. Um bom início seria colocar a classe de controle em um pacote e então colocar classes entidade e de interface fortemente acopladas no mesmo pacote.
Uma classe pode também não pertencer a nenhum pacote por não de encaixar a nenhuma das funcionalidades ou por pertencer igualmente a duas ou mais.