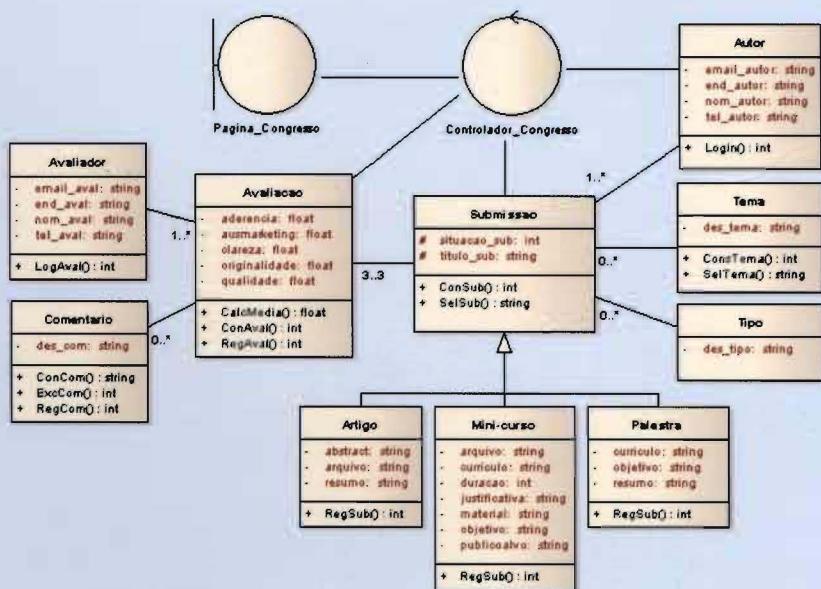


UML 2

GUIA PRÁTICO



UML 2

Guia Prático

Gilleanes T.A. Guedes

Obra revisada e ampliada a partir do título Guia de Consulta Rápida UML 2

Novatec

Copyright © 2007 da Novatec Editora Ltda.

Todos os direitos reservados e protegidos pela Lei 9.610 de 19/02/1998.
É proibida a reprodução desta obra, mesmo parcial, por qualquer processo,
sem prévia autorização, por escrito, do autor e da Editora.

Editor: Rubens Prates

Assistente editorial: Camila Araújo

Revisão gramatical: Gabriela de Andrade Fazioni

Capa: Camila Araújo

ISBN: 978-85-7522-145-7

Obra originalmente publicada sob o título Guia de Consulta Rápida UML 2.
ISBN 85-7522-065-9

NOVATEC EDITORA LTDA.

Rua Luís Antônio dos Santos 110
02460-000 – São Paulo, SP – Brasil
Tel.: +55 11 6959-6529
Fax: +55 11 6950-8869
E-mail: novatec@novatec.com.br
Site: www.novatec.com.br

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Guedes, Gilleanes T. A.
UML 2 : guia prático / Gilleanes T. A. Guedes. --
São Paulo : Novatec Editora, 2007.

ISBN 978-85-7522-145-7

1. Software - Desenvolvimento 2. UML (Ciência
da computação) I. Título.

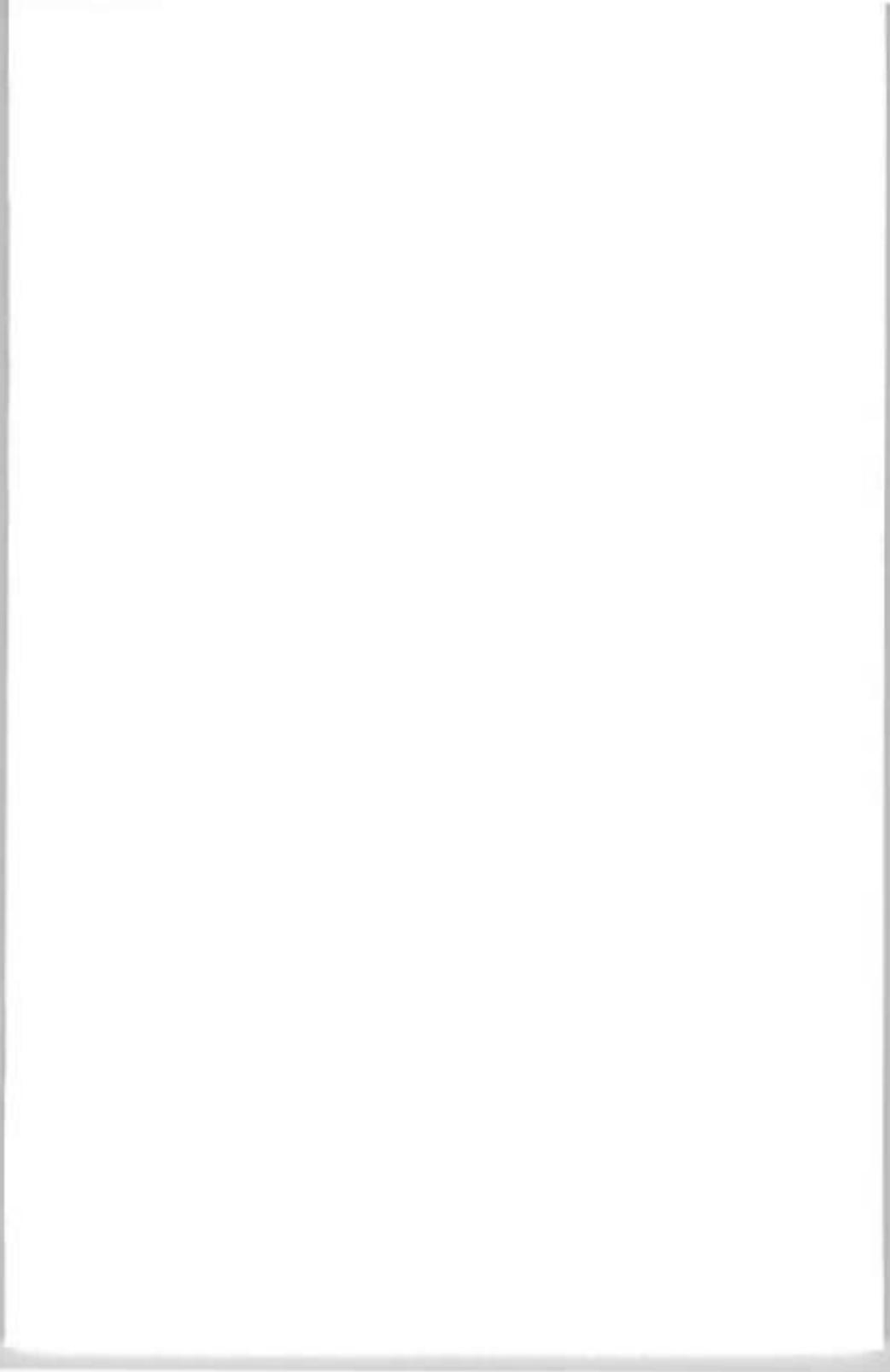
07-8935

CDD-005.369

Índices para catálogo sistemático:

1. UML 2.0 : Desenvolvimento : Ciência da computação
005.369

*Dedico este livro à minha esposa Sílvia,
ao meu filho Ulisses e ao novo bebê*



Sumário

Sobre o autor	11
Estrutura deste Guia.....	12
Capítulo 1 ■ Introdução à UML	13
1.1 Breve Histórico da UML.....	13
1.2 Por Que Tantos Diagramas?	14
1.3 Resumo dos Diagramas da UML.....	15
1.3.1 Diagrama de Casos de Uso	15
1.3.2 Diagrama de Classes	17
1.3.3 Diagrama de Objetos	18
1.3.4 Diagrama de Estrutura Composta	19
1.3.5 Diagrama de Seqüência	20
1.3.6 Diagrama de Comunicação	21
1.3.7 Diagrama de Máquina de Estados	22
1.3.8 Diagrama de Atividade.....	22
1.3.9 Diagrama de Interação Geral.....	23
1.3.10 Diagrama de Componentes.....	24
1.3.11 Diagrama de Implantação	25
1.3.12 Diagrama de Pacotes.....	26
1.3.13 Diagrama de Tempo.....	27
1.3.14 Síntese Geral dos Diagramas	27
1.4 Ferramentas CASE Baseadas na Linguagem UML.....	28
Capítulo 2 ■ Orientação a Objetos	30
2.1 Classificação, Abstração e Instanciação	30
2.2 Classes de Objetos	32
2.3 Atributos.....	32
2.4 Métodos	33
2.5 Visibilidade	34
2.6 Herança	35
2.7 Polimorfismo	36
Capítulo 3 ■ Diagrama de Casos de Uso	38
3.1 Atores	38
3.2 Casos de Uso	39
3.3 Associações	40

3.4 Especialização/Generalização	40
3.5 Inclusão	42
3.6 Extensão	43
3.7 Restrições em Associações de Extensão.....	44
3.8 Pontos de Extensão	45
3.9 Multiplicidade no Diagrama de Casos de Uso.....	45
3.10 Fronteira de Sistema.....	46
3.11 Documentação de Casos de Uso	46
3.12 Estudo de Caso	48
Capítulo 4 ■ Diagrama de Classes	52
4.1 Relacionamentos ou Associações	53
4.1.1 Associação Unária ou Reflexiva	54
4.1.2 Associação Binária	56
4.1.3 Associação Ternária ou N-ária	56
4.1.4 Agregação	57
4.1.5 Composição	58
4.1.6 Especialização/Generalização.....	59
4.1.7 Dependência.....	59
4.1.8 Realização	60
4.2 Classe Associativa	60
4.3 Interfaces	62
4.3.1 Interfaces Fornecidas	62
4.3.2 Interfaces Requeridas	63
4.4 Restrição	64
4.5 Estereótipos	67
4.5.1 Estereótipo <<entity>>	67
4.5.2 Estereótipos <<boundary>> e <<control>>	68
4.6 Estudo de Caso	69
Capítulo 5 ■ Diagrama de Objetos	73
5.1 Objeto	73
5.2 Vínculos.....	74
Capítulo 6 ■ Diagrama de Estrutura Composta.....	76
6.1 Colaborações.....	76
6.2 Ocorrência de Colaboração	79
6.3 Portas	80
6.4 Propriedades	81
6.5 Estudo de Caso	83
Capítulo 7 ■ Diagrama de Seqüência	84
7.1 Atores	84
7.2 Objetos	85
7.3 Linha de Vida	86
7.4 Foco de Controle ou Ativação.....	87
7.5 Mensagens ou Estímulos	87

Sumário

Sumário	9
7.6 Mensagens de retorno	90
7.7 Auto-chamadas ou Auto-delegações	91
7.8 Condições ou Condições de Guarda	92
7.9 Fragmentos de Interação e Ocorrências de Interação	92
7.9.1 Portões (Gates)	94
7.9.2 Fragmentos Combinados e Operadores de Interação	95
7.10 Estudo de Caso	103
7.10.1 Realizar Submissão	103
7.10.2 Realizar Login Submissor	104
7.10.3 Verificar Submissões	105
7.10.4 Verificar Comentários	106
7.10.5 Manter Avaliações	106
7.10.6 Manter Comentários	108
7.10.7 Relatório de Avaliações	110
Capítulo 8 ■ Diagrama de Comunicação.....	111
8.1 Mensagens	111
8.2 Estudo de Caso	112
Capítulo 9 ■ Diagrama de Máquina de Estados	118
9.1 Estado	118
9.2 Transições	119
9.3 Estados Inicial e Final	119
9.4 Atividades internas	120
9.5 Transições internas	121
9.6 Auto-Transições	121
9.7 Pseudo-Estado de Escolha	122
9.8 Barra de Sincronização	124
9.9 Pseudo-Estado de Junção	125
9.10 Estado Composto	126
9.11 Estado de História	126
9.12 Estado de Submáquina	127
9.13 Estados de Entrada e Saída (Entry e Exit States)	128
9.14 Pseudo-Estado de Término	128
9.15 Estado de Sincronismo	129
9.16 Estudo de Caso	130
9.16.1 Verificar Comentários	130
9.16.2 Manter Avaliações	131
9.16.3 Relatório de Avaliações	132
Capítulo 10 ■ Diagrama de Atividade	133
10.1 Nô de Ação	134
10.2 Controle de Fluxo	134
10.3 Nô Inicial	135
10.4 Nô Final	135
10.5 Nô de Decisão	135

10.6 Exemplo de Diagrama de Atividade	136
10.7 Conectores.....	138
10.8 Subatividade.....	139
10.9 Nó de Bifurcação/União	140
10.10 Final de Fluxo	140
10.11 Fluxo de Objetos.....	141
10.12 Nó de Objeto	141
10.13 Alfinetes (Pins)	141
10.14 Nó de Parâmetro de Atividade	142
10.15 Exceções	142
10.16 Ação de Objeto de Envio	143
10.17 Ação de Evento de Aceitação	143
10.18 Ação de Evento de Tempo de Aceitação.....	144
10.19 Nó de Repositório de Dados (Data Store Node)	145
10.20 Partição de Atividade	145
10.21 Região de Atividade Interrompível.....	146
10.22 Região de Expansão	147
10.23 Estudo de Caso.....	149
Capítulo 11 ■ Diagrama de Interação Geral.....	154
Capítulo 12 ■ Diagrama de Componentes.....	156
12.1 Componente	156
12.2 Interfaces Fornecidas e Requeridas	157
12.3 Classes e Componentes Internos	158
12.4 Exemplo de Diagrama de Componentes	160
Capítulo 13 ■ Diagrama de Implantação.....	161
13.1 Nós.....	161
13.2 Associação entre Nós	162
13.3 Exemplo de Diagrama de Implantação.....	162
13.4 Artefatos.....	163
13.5 Especificação de Implantação.....	165
Capítulo 14 ■ Diagrama de Pacotes	166
14.1 Pacote	166
14.2 Dependência	167
14.3 Pacotes Contendo Pacotes	167
Capítulo 15 ■ Diagrama de Tempo.....	169
Índice remissivo	171

Sobre o autor

Gilleanes Thorwald Araujo Guedes é Bacharel em Informática pela Universidade da Região da Campanha (URCAMP) e mestre em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (UFRGS). É professor de Engenharia de Software no curso de Licenciatura Plena em Informática na Universidade Federal de Mato Grosso (UFMT). Ministrou diversas palestras e cursos sobre UML em eventos científicos e pós-graduações “Lato Sensu”, e é autor dos livros “UML – Uma Abordagem Prática” e “UML 2 – Guia de Consulta Rápida” publicados pela Novatec.

Estrutura deste Guia

O Capítulo 1 deste guia fornece uma visão geral da UML, apresentando um breve histórico da linguagem e destacando seus objetivos, além de fornecer uma breve explanação sobre cada um de seus diagramas. Ao fim deste capítulo são apresentadas algumas das ferramentas CASE que suportam a UML disponíveis no mercado.

O Capítulo 2 apresenta o Paradigma de Orientação a Objetos, cujo conhecimento é imprescindível para quem deseja modelar sistemas orientados a objeto. Neste capítulo serão detalhados os conceitos de abstração, classes, objetos, herança e polimorfismo, entre outros.

Nos capítulos seguintes o guia passa a discorrer sobre cada um dos 13 diagramas da linguagem UML, detalhando seus objetivos, apresentando e exemplificando seus componentes, além de detalhar como utilizar cada diagrama na modelagem de sistemas.

Ao longo deste guia iremos modelar um pequeno sistema como estudo de caso. O sistema modelado tem por objetivo controlar a submissão de trabalhos em um congresso científico e será modelado por meio de praticamente todos os diagramas fornecidos pela linguagem, de acordo com a visão e com os objetivos de cada diagrama.

A ferramenta utilizada para a modelagem de todos os diagramas deste guia foi a Enterprise Architect. No final do Capítulo 1 faremos um breve comentário sobre essa ferramenta.

CAPÍTULO 1

Introdução à UML

A UML (Unified Modeling Language ou Linguagem de Modelagem Unificada) é uma linguagem visual utilizada para modelar sistemas computacionais por meio do paradigma de Orientação a Objetos. Essa linguagem se tornou, nos últimos anos, a linguagem-padrão de modelagem de software adotada internacionalmente pela indústria de Engenharia de Software.

Deve ficar bem claro, no entanto, que a UML não é uma linguagem de programação, mas uma linguagem de modelagem, cujo objetivo é auxiliar os engenheiros de software a definir as características do software, tais como seus requisitos, seu comportamento, sua estrutura lógica, a dinâmica de seus processos e até mesmo suas necessidades físicas em relação ao equipamento sobre o qual o sistema deverá ser implantado. Todas essas características são definidas por meio da UML antes de o software começar a ser realmente desenvolvido.

1.1 Breve Histórico da UML

A UML surgiu da união de três metodologias de modelagem: o método de Booch, o método OMT (Object Modeling Technique) de Jacobson e o método OOSE (Object-Oriented Software Engineering) de Rumbaugh. Essas eram, até meados da década de 1990, as três metodologias de modelagem orientada a objetos mais populares entre os profissionais da área de engenharia de software. A união dessas metodologias contou com o amplo apoio da Rational Software, que incentivou e financiou tal união.

O esforço inicial do projeto começou com a união do método de Booch com o método OMT de Jacobson, o que resultou no lançamento do Método Unificado no final de 1995. Logo em seguida, Rumbaugh juntou-se a Booch e Jacobson na Rational Software e seu método OOSE começou também a ser incorporado à nova metodologia. O trabalho de Booch, Jacobson e Rumbaugh, conhecidos popularmente como “Os Três Amigos”, resultou no lançamento, em 1996, da primeira versão da UML propriamente dita.

Tão logo a primeira versão foi lançada, diversas grandes empresas atuantes na área de engenharia e desenvolvimento de software passaram a contribuir com o projeto, fornecendo sugestões para melhorar e ampliar a linguagem. Finalmente a UML foi adotada pela OMG (Object Management Group ou Grupo de Gerenciamento de Objetos) em 1997, como uma linguagem-padrão de modelagem. A UML em sua versão 2.0 trouxe grandes novidades em relação à estrutura geral da linguagem principalmente com relação à abordagem de quatro camadas e à possibilidade de se desenvolver “perfis” particulares a partir da UML, cuja documentação oficial pode ser consultada no site da OMG em www.omg.com.

1.2 Por Que Tantos Diagramas?

O objetivo disso é fornecer múltiplas visões do sistema a ser modelado, analisando-o e modelando-o sob diversos aspectos, procurando-se assim atingir a completude da modelagem, permitindo que cada diagrama complemente os outros. Cada diagrama da UML analisa o sistema, ou parte dele, sob uma determinada ótica; é como se o sistema fosse modelado em camadas. Alguns diagramas enfocam o sistema de forma mais geral, apresentando uma visão externa do sistema, como é o objetivo do Diagrama de Casos de Uso, ao passo que outros oferecem uma visão de uma camada mais profunda do software, apresentando um enfoque mais técnico ou ainda visualizando apenas uma característica específica do sistema ou um determinado processo.

A utilização de diversos diagramas permite que falhas possam ser descobertas nos diagramas anteriores, diminuindo a possibilidade da ocorrência de erros durante a fase de desenvolvimento do software. É importante destacar que, embora cada diagrama tenha sua utilidade,

nem sempre é necessário modelar um sistema utilizando-se de todos os diagramas, pois alguns deles possuem funções muito específicas, como é o caso do Diagrama de Tempo, por exemplo.

1.3 Resumo dos Diagramas da UML

A seguir descreveremos rapidamente cada um dos diagramas oferecidos pela UML, destacando suas principais características e objetivos. O leitor irá notar que cada diagrama possui uma aba em seu canto superior esquerdo contendo um operador e a descrição do diagrama. O operador serve para determinar qual é o tipo de diagrama, assim **uc** refere-se a Use Case Diagram (Diagrama de Caso de Uso); **class** a Class Diagram (Diagrama de Classes); **object**, a Object Diagram (Diagrama de Objetos); **sd**, a Sequence Diagram (Diagrama de Seqüência); **stm**, a State-Machine Diagram (Diagrama de Máquina de Estados) e assim por diante.

1.3.1 Diagrama de Casos de Uso

Este é o diagrama mais geral e informal da UML, sendo utilizado principalmente para auxiliar no levantamento e análise dos requisitos, em que são determinadas as necessidades do usuário, e na compreensão do sistema como um todo, embora venha a ser consultado durante todo o processo de modelagem e sirva de base para todos os outros diagramas.

O Diagrama de Casos de Uso apresenta uma linguagem simples e de fácil compreensão para que os usuários possam ter uma idéia geral de como o sistema irá se comportar. Ele procura identificar os atores (usuários, outros softwares que interajam com o sistema ou até mesmo algum hardware especial), que utilizarão de alguma forma o software, bem como os serviços, ou seja, as opções que o sistema disponibilizará aos atores, conhecidas neste diagrama como Casos de Uso. A Figura 1.1 apresenta um exemplo desse diagrama.

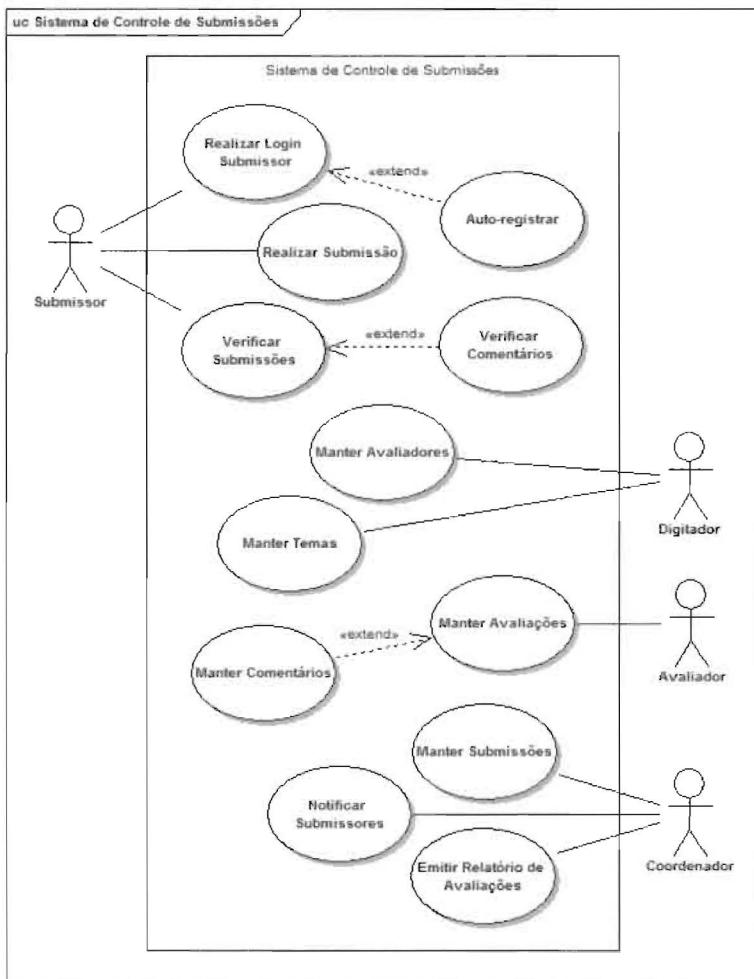


Figura 1.1 – Exemplo de Diagrama de Casos de Uso.

1.3.2 Diagrama de Classes

Este é o diagrama mais utilizado e o mais importante da UML, servindo de apoio para a maioria dos outros diagramas. Como o próprio nome diz, esse diagrama define a estrutura das classes utilizadas pelo sistema, determinando os atributos e métodos possuídos por cada classe, além de estabelecer como as classes se relacionam e trocam informações entre si. A Figura 1.2 demonstra um exemplo desse diagrama.

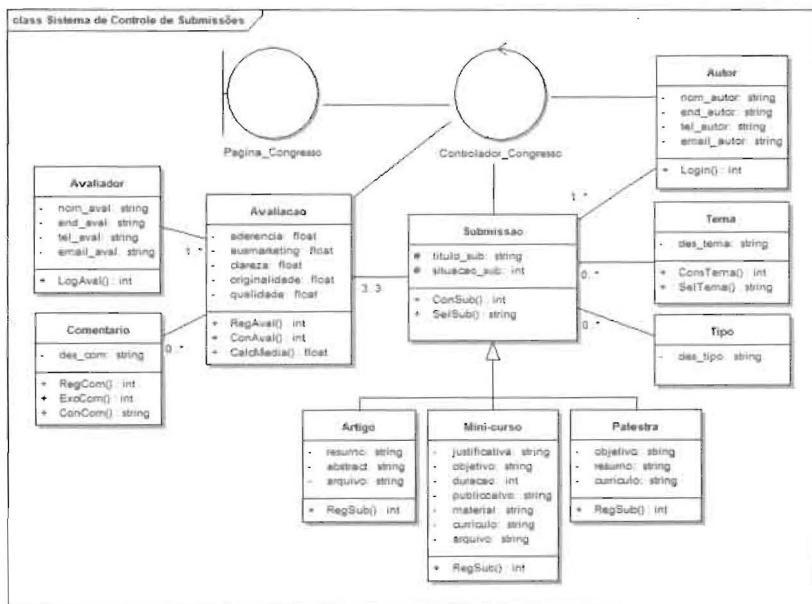


Figura 1.2 – Exemplo de Diagrama de Classes.

1.3.3 Diagrama de Objetos

Este diagrama está amplamente associado ao Diagrama de Classes. Na verdade, o Diagrama de Objetos é praticamente um complemento do Diagrama de Classes, sendo bastante dependente deste. O Diagrama de Objetos fornece uma visão dos valores armazenados pelos objetos de um Diagrama de Classes em um determinado momento da execução de um processo. A Figura 1.3 apresenta um exemplo de Diagrama de Objetos.

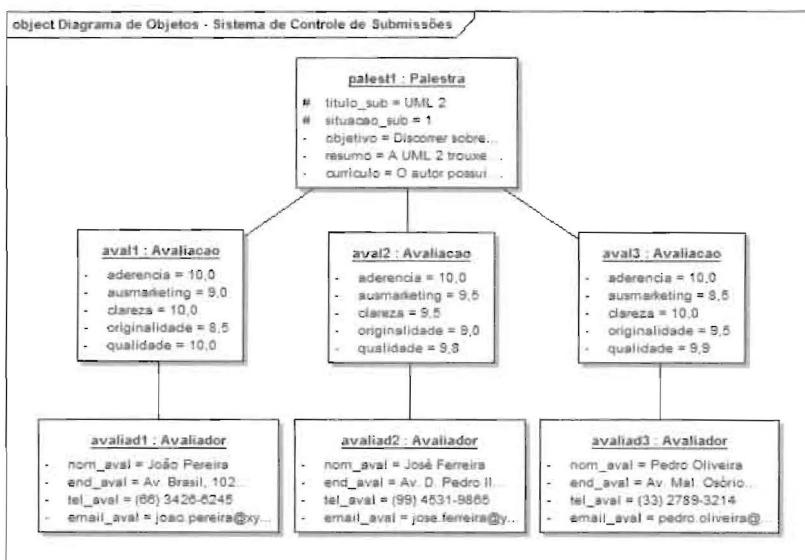


Figura 1.3 – Exemplo de Diagrama de Objetos.

1.3.4 Diagrama de Estrutura Composta

O Diagrama de Estrutura Composta é utilizado para modelar Colaborações. Uma colaboração descreve uma visão de um conjunto de entidades cooperativas interpretadas por instâncias que cooperam entre si para executar uma função específica. O termo estrutura desse diagrama refere-se a uma composição de elementos interconectados, representando instâncias de tempo de execução colaboram, por meio de vínculos de comunicação, para atingir algum objetivo comum. Esse diagrama também pode ser utilizado para definir a estrutura interna de um classificador. A Figura 1.4 apresenta um exemplo de Diagrama de Estrutura Composta.

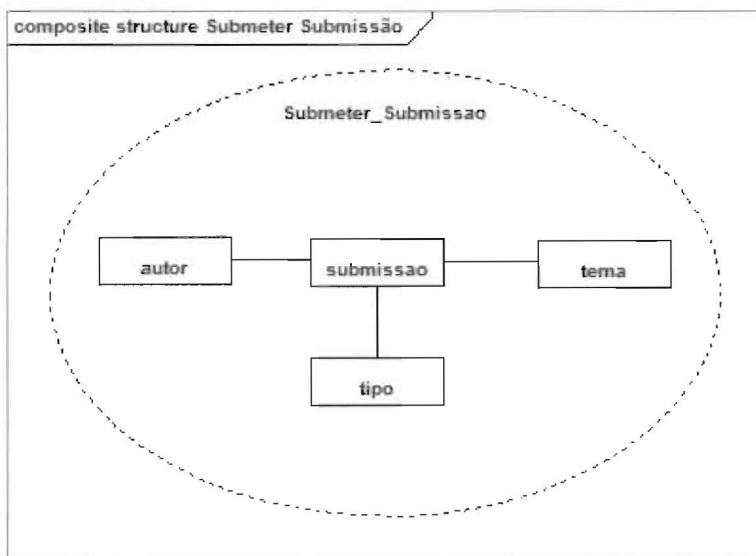


Figura 1.4 – Exemplo de Diagrama de Estrutura Composta.

1.3.5 Diagrama de Seqüência

O Diagrama de Seqüência preocupa-se com a ordem temporal em que as mensagens são trocadas entre os objetos envolvidos em um determinado processo. Em geral, baseia-se em um Caso de Uso definido pelo diagrama de mesmo nome e apóia-se no Diagrama de Classes para determinar os objetos das classes envolvidas em um processo, bem como os métodos disparados entre os mesmos. Um Diagrama de Seqüência costuma identificar o evento gerador do processo modelado, bem como o ator responsável por este evento, e determina como o processo deve se desenrolar e ser concluído por meio do envio de mensagens, que em geral disparam métodos entre os objetos. A Figura 1.5 apresenta um exemplo desse diagrama.

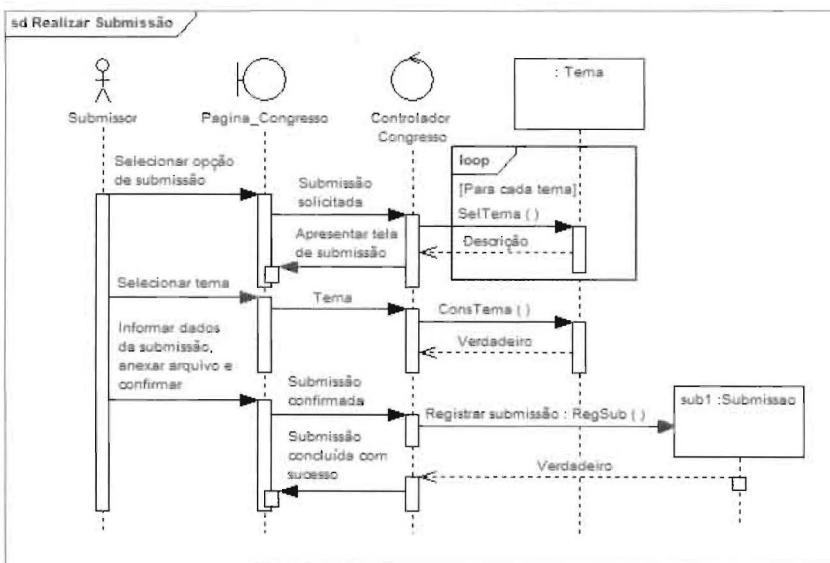


Figura 1.5 – Exemplo de Diagrama de Seqüência.

1.3.6 Diagrama de Comunicação

O Diagrama de Comunicação era conhecido como Diagrama de Colaboração até a versão 1.5 da UML, tendo seu nome modificado para Diagrama de Comunicação a partir da versão 2.0. Esse diagrama está amplamente associado ao Diagrama de Seqüência; na verdade, um complementa o outro. As informações mostradas no Diagrama de Comunicação são, com freqüência, praticamente as mesmas apresentadas no Diagrama de Seqüência, porém com um enfoque diferente, visto que este diagrama não se preocupa com a temporalidade do processo, concentrando-se em como os objetos estão vinculados e quais mensagens trocam entre si durante o processo. A Figura 1.6 apresenta um exemplo de Diagrama de Comunicação.

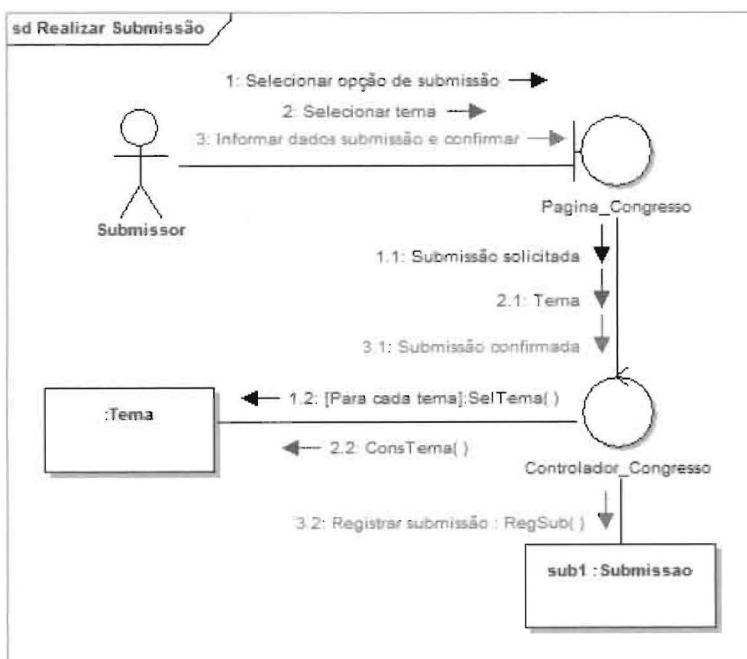


Figura 1.6 – Exemplo de Diagrama de Comunicação.

1.3.7 Diagrama de Máquina de Estados

O Diagrama de Máquina de Estados era conhecido nas versões anteriores da linguagem como Diagrama de Gráfico de Estados ou simplesmente como Diagrama de Estados, tendo assumido nova nomenclatura a partir da versão 2. Esse diagrama procura acompanhar as mudanças sofridas nos estados de uma instância de uma classe, de um Caso de Uso ou mesmo de um subsistema ou sistema completo. Como o Diagrama de Seqüência, o Diagrama de Máquina de Estados muitas vezes se baseia em um Caso de Uso e se apóia no Diagrama de Classes. A Figura 1.7 apresenta um exemplo de Diagrama de Máquina de Estados.

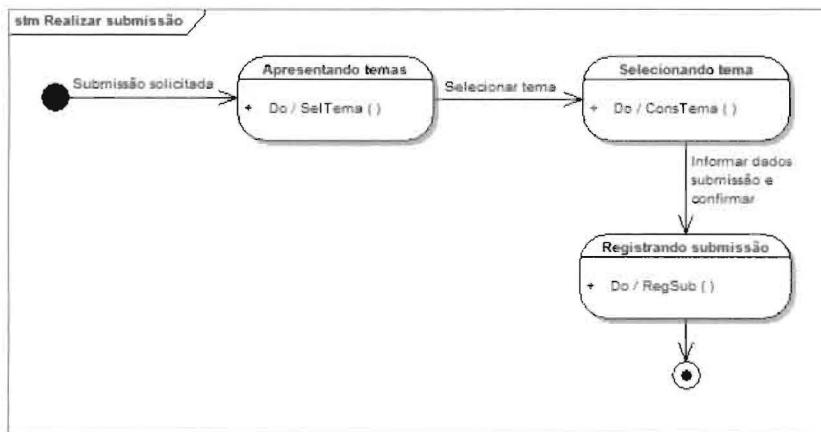


Figura 1.7 – Exemplo de Diagrama de Máquina de Estados.

1.3.8 Diagrama de Atividade

O Diagrama de Atividade era considerado um caso especial do antigo Diagrama de Gráfico de Estados, mas, a partir da UML 2.0, esse diagrama se tornou independente, deixando inclusive de se basear em máquinas de estados e passando a se basear em Redes de Petri. O Diagrama de Atividade se preocupa em descrever os passos a serem percorridos para a conclusão de uma atividade específica, muitas vezes representada por um método com um certo grau de complexidade, podendo, no entanto, modelar um processo completo. Concentra-se na representação do fluxo de controle e no fluxo de objeto de uma atividade. A Figura 1.8 apresenta um exemplo desse diagrama.

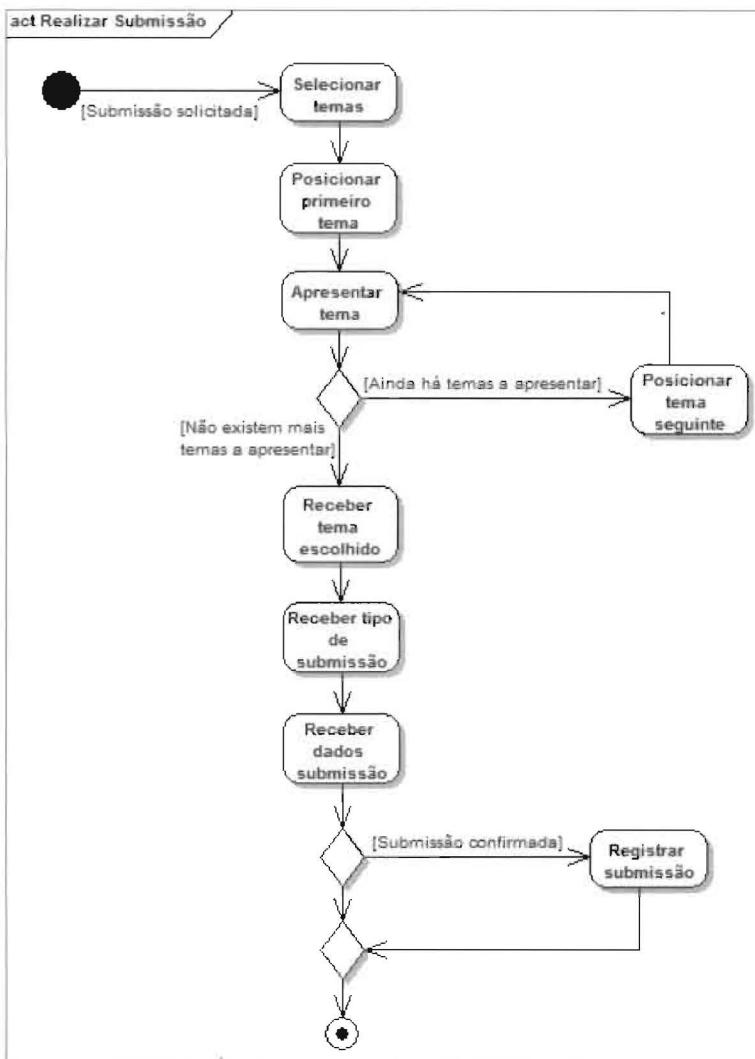


Figura 1.8 – Exemplo de Diagrama de Atividade.

1.3.9 Diagrama de Interação Geral

O Diagrama de Interação Geral é uma variação do Diagrama de Atividade que fornece uma visão ampla dentro de um sistema ou processo de negócio. Esse diagrama passou a existir somente a partir da UML 2. O Diagrama de Interação Geral costuma englobar diversos tipos de diagramas de

interação para demonstrar um processo geral. A Figura 1.9 apresenta um exemplo desse diagrama.

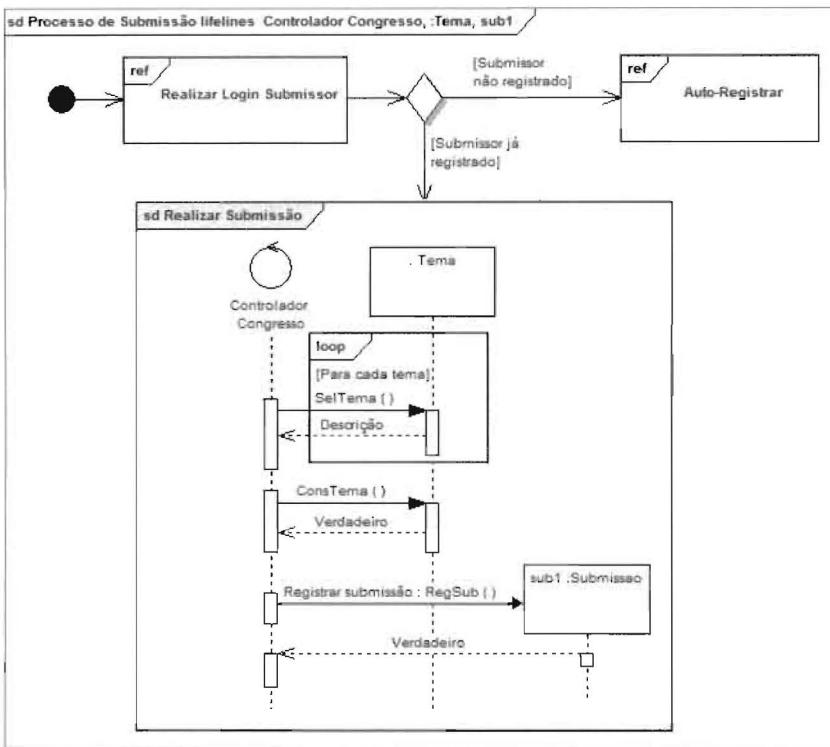


Figura 1.9 – Exemplo de Diagrama de Interação Geral.

1.3.10 Diagrama de Componentes

O Diagrama de Componentes está amplamente associado à linguagem de programação que será utilizada para desenvolver o sistema modelado. Esse diagrama representa as componentes do sistema quando este for ser implementado em termos de módulos de código-fonte, bibliotecas, formulários, arquivos de ajuda, módulos executáveis etc. e determina como esses componentes estarão estruturados e interagirão para que o sistema funcione de maneira adequada. O Diagrama de Componentes pode ser utilizado para modelar o código-fonte, os módulos executáveis de um sistema, a estrutura física de um banco de dados ou mesmo os componentes

necessários para a construção de interfaces. A Figura 1.10 apresenta um exemplo desse diagrama.

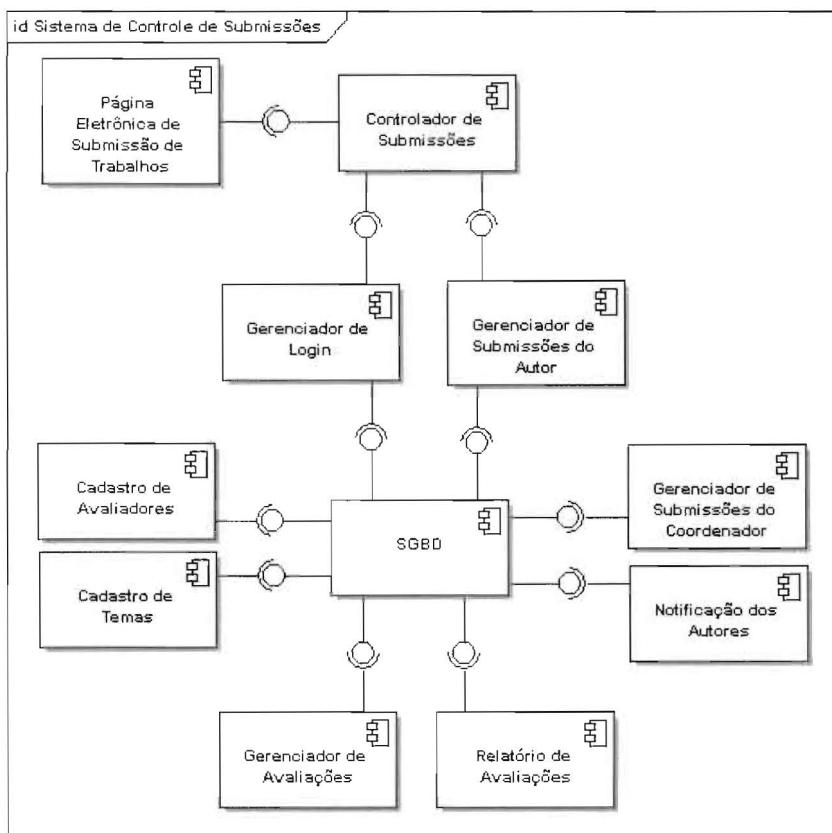


Figura 1.10 – Exemplo de Diagrama de Componentes.

1.3.11 Diagrama de Implantação

O Diagrama de Implantação determina as necessidades de hardware do sistema, as características físicas como servidores, estações, topologias e protocolos de comunicação, ou seja, todo o aparato físico sobre o qual o sistema deverá ser executado. A Figura 1.11 apresenta um exemplo desse diagrama.

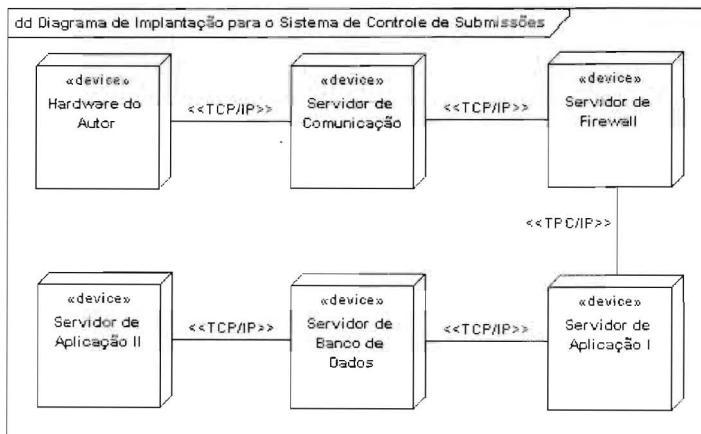


Figura 1.11 – Exemplo de Diagrama de Implantação.

1.3.12 Diagrama de Pacotes

O Diagrama de Pacotes tem por objetivo representar os subsistemas ou submódulos englobados por um sistema de forma a determinar as partes que o compõem. Pode ser utilizado de maneira independente ou associado com outros diagramas. Este diagrama pode ser utilizado também para ajudar a demonstrar a arquitetura de uma linguagem, como ocorre com a própria UML. A Figura 1.12 apresenta um exemplo desse diagrama.

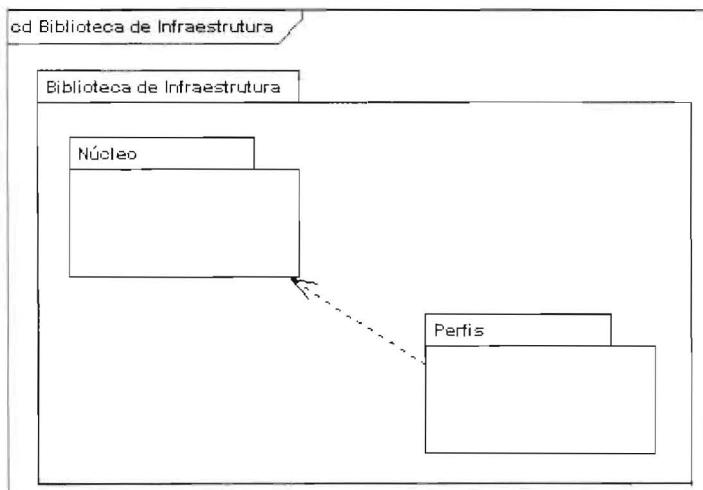


Figura 1.12 – Exemplo de Diagrama de Pacotes.

1.3.13 Diagrama de Tempo

O Diagrama de Tempo descreve a mudança no estado ou condição de uma instância de uma classe ou seu papel durante um tempo. É tipicamente utilizado para demonstrar a mudança no estado de um objeto no tempo em resposta a eventos externos. Esse diagrama somente passou a existir a partir da versão 2.0 da linguagem. A Figura 1.13 apresenta um exemplo desse diagrama.

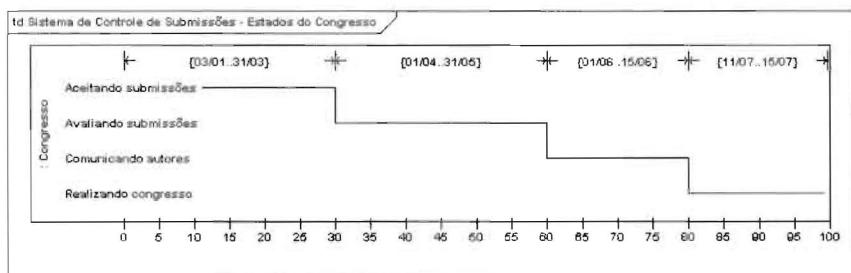


Figura 1.13 – Exemplo de Diagrama de Tempo.

1.3.14 Síntese Geral dos Diagramas

Os diagramas da UML 2.0 dividem-se em Diagramas Estruturais e Diagramas Comportamentais, sendo que estes últimos possuem ainda uma subdivisão representada pelos Diagramas de Interação, conforme pode ser verificado na Figura 1.14.

Como podemos observar, os Diagramas Estruturais abrangem os Diagramas de Classes, de Estrutura Composta, de Objetos, de Componentes, de Implantação e de Pacotes, ao passo que os Diagramas Comportamentais englobam os Diagramas de Casos de Uso, Atividade, Máquina de Estados, Seqüência, Comunicação, de Interação Geral e de Tempo, sendo que estes últimos quatro correspondem aos diagramas da subdivisão de Diagramas de Interação.

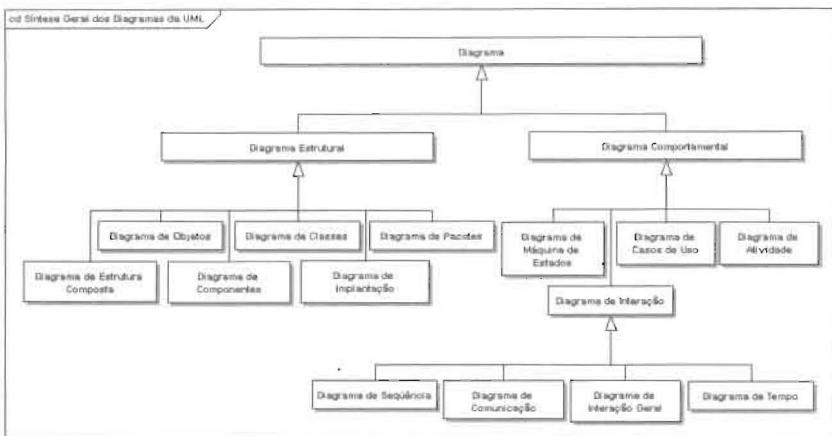


Figura 1.14 – Diagramas da UML.

1.4 Ferramentas CASE Baseadas na Linguagem UML

Ferramentas CASE (Computer-Aided Software Engineering ou Engenharia de Software Auxiliada por Computador) são softwares que, de alguma maneira, colaboram para a execução de uma ou mais atividades realizadas durante o processo de Engenharia de Software. A maioria das ferramentas CASE atuais suportam a UML, sendo esta, em geral, sua principal característica. Entre as diversas ferramentas existentes no mercado podemos destacar:

- **Rational Rose** – esta foi a primeira ferramenta CASE baseada na linguagem UML. A Rational Rose foi desenvolvida pela Rational Software, empresa que incentivou a criação da UML. Por ter sido a primeira, é provavelmente a ferramenta mais conhecida. Ela é totalmente orientada à UML, no entanto, suporta igualmente os métodos de Booch e OMT. Uma cópia da ferramenta Rational Rose pode ser adquirida por meio do link www.rational.com.
- **Visual Paradigm for UML ou VP-UML** – esta ferramenta pode ser encontrada no site www.visual-paradigm.com e oferece inclusive uma edição para a comunidade, ou seja, uma versão da ferramenta que pode ser baixada gratuitamente de sua página. Logicamente, a edição para a comunidade não suporta todos os serviços e opções disponíveis nas versões Standard ou Professional da ferramenta, no entanto, para quem deseja praticar a UML, a edição para a comuni-

dade é uma boa alternativa, apresentando um ambiente amigável e de fácil compreensão. Além disso, a Visual-Paradigm oferece ainda uma cópia acadêmica da versão Standard para instituições de ensino superior, que podem conseguí-la por meio de uma solicitação na própria página da empresa. Tão logo a Visual-Paradigm comprove a veracidade das informações fornecidas pela instituição, ela enviará uma licença de um ano para uso pelos professores e seus alunos. A licença precisa ser renovada anualmente.

- **Poseidon for UML** – esta ferramenta também possui uma edição para a comunidade, apresentando bem menos restrições que a edição para a comunidade da Visual-Paradigm. A interface da Poseidon, porém é sensivelmente inferior à VP-UML, além de apresentar um desempenho um pouco inferior também, embora ambas as ferramentas tenham sido desenvolvidas em Java. Uma cópia da Poseidon for UML pode ser adquirida no site www.gentleware.com.
- **ArgoUML** – é uma ferramenta um tanto limitada e sua interface não é das mais amigáveis e intuitivas. Porém, ela apresenta uma característica bastante interessante e atrativa: é totalmente livre. O desenvolvimento da ferramenta ArgoUML constitui-se em um projeto acadêmico, em que os códigos-fonte dessa ferramenta podem até mesmo ser baixados e utilizados para o desenvolvimento de ferramentas comerciais, como foi o caso da Poseidon for UML. Os usuários dessa ferramenta podem perceber muitas semelhanças entre as duas ferramentas, mas a Poseidon possui uma interface muito melhor e é, em geral, superior à ArgoUML. No entanto, o projeto de código aberto ArgoUML pede que quaisquer empresas que utilizarem seus códigos como base para uma nova ferramenta disponibilizem uma edição para a comunidade gratuitamente. Uma cópia da ArgoUML pode ser encontrada no site www.argouml.tigris.org.
- **Enterprise Architect** – embora a ferramenta Enterprise Architect não ofereça uma edição para a comunidade como as anteriores, ela é uma das ferramentas que mais oferecem recursos compatíveis com a UML 2. Apesar de não dispor de uma edição para a comunidade, a Sparx Systems, empresa que produz a Enterprise Architect, disponibiliza no site www.sparxsystems.com.au uma versão trial, que pode ser utilizada por cerca de 60 dias.

CAPÍTULO 2

Orientação a Objetos

A UML está totalmente inserida dentro do Paradigma de Orientação a Objetos. Por esse motivo, para comprehendê-la corretamente, precisamos antes compreender o conceito de Orientação a Objetos.

2.1 Classificação, Abstração e Instanciação

O ser humano, no início de sua infância, aprende e pensa de uma maneira orientada a objetos, representando seu conhecimento por meio de classificações e abstrações (na verdade continuamos fazendo isso mesmo quando adultos, mas desenvolvemos outras técnicas que utilizamos paralelamente). As crianças aprendem conceitos simples como pessoa, carro e casa e, ao fazerem isso, definem classes, ou seja, grupos de objetos, sendo que cada objeto é um exemplo de um determinado grupo, possuindo as mesmas características e comportamentos de qualquer objeto do grupo em questão. A partir desse momento, qualquer coisa que possuir cabeça, tronco e membros torna-se uma pessoa; qualquer construção em que as pessoas possam entrar passa a ser uma casa; e qualquer peça de metal com quatro rodas que se locomova de um lugar para outro transportando pessoas recebe a denominação de carro.

Na verdade, esse processo por si só já envolve um grande esforço de abstração, em razão de os carros, por exemplo, aos carros apresentarem diferentes formatos, cores e estilos. Uma criança deve se sentir um pouco confusa no começo ao descobrir que o objeto amarelo e o objeto vermelho possuem a mesma classificação: Carro. A partir desse momento, a criança

precisa abstrair o conceito de carro para chegar à conclusão de que “carro” é um termo geral que se refere a muitos objetos, no entanto cada um dos objetos-carro possui características semelhantes entre si – todos, por exemplo, todos têm quatro rodas e, no mínimo duas portas, além de luzes de farol e de freio, bem como vidros frontais e laterais. Além disso, os objetos-carro podem realizar determinadas tarefas, sendo a principal delas transportar pessoas de um lugar para outro.

No momento em que a criança comprehende esse conceito, ela percebe que “carro” é a denominação de um grupo, ou seja, ela abstraiu uma classe: a classe carro. Assim, sempre que ela perceber a presença de um objeto com as características já determinadas, ela concluirá que aquele objeto é um exemplo do grupo carro, ou seja, uma instância da classe carro. Dessa maneira, nosso modo de aprender é, ao menos no início, orientado a objetos, ou seja, aprendemos por meio de classificações. Aprendemos a classificar praticamente tudo, criando grupos de objetos com características iguais, sendo que cada grupo de objetos é equivalente a uma classe. Sempre que precisamos compreender um conceito novo, criamos uma nova classe para esse conceito e determinamos que todo objeto com as características dessa classe é um exemplo, uma instância dela. Dessa forma, instanciação constitui-se simplesmente em criar um exemplo de um tipo, um grupo, uma classe.

Quando instanciamos um objeto de uma classe, estamos criando um novo item do conjunto representado por essa classe, com as mesmas características e comportamentos de todos os outros objetos já instanciados. Além disso, após abstrair um conceito inicial, costumamos criar subdivisões dentro das classes, isto é, grupos dentro de grupos, o que já caracteriza um novo nível de abstração. Podemos criar subgrupos dentro da classe Carro, classificando-os por marca ou modelo, por exemplo, ou dentro da classe Pessoa, classificando os novos grupos por profissão, grau de parentesco ou status social.

Deve-se ter em mente, no entanto, que, apesar de possuírem os mesmos atributos, os objetos de uma classe não são exatamente iguais, pois cada objeto armazena valores diferentes em seus atributos. Por exemplo, todos os objetos da classe Carro possuem o atributo cor, mas um dos objetos pode ter uma cor azul ao passo que outro objeto pode possuir uma cor

amarela. Da mesma forma, todos os objetos da classe Pessoa possuem o atributo altura, porém, o valor armazenado no atributo altura varia de pessoa para pessoa.

2.2 Classes de Objetos

Uma classe é representada por um retângulo que pode possuir até três divisões. A primeira divisão armazena o nome da classe; a segunda, os atributos da classe; e a terceira, os métodos. Geralmente uma classe possui atributos e métodos, porém, é possível encontrar classes que contenham apenas uma dessas características ou mesmo nenhuma quando se tratar de classes abstratas. A Figura 2.1 apresenta um exemplo de classe.

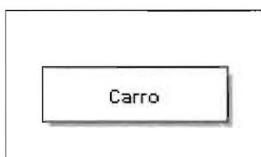


Figura 2.1 – Exemplo de uma Classe.

A Figura 2.1 ilustra um exemplo de uma classe, em que identificamos a classe Carro. Note que essa classe possui apenas uma divisão que contém seu nome, já que não é obrigatório representar uma classe totalmente expandida, embora seja mais comum encontrar exemplos assim. As outras divisões de uma classe serão explicadas nas seções a seguir.

2.3 Atributos

Classes costumam definir atributos que representam as características de uma classe, que costumam variar de objeto para objeto – como a altura em um objeto da classe Pessoa ou a cor em um objeto da classe Carro – e que permitem diferenciar um objeto de outro da mesma classe.

Os atributos são apresentados na segunda divisão da classe e contêm, normalmente, duas informações: o nome que identifica o atributo e eventualmente o tipo de dado que o atributo armazena, como, por exemplo, integer, float ou character. Na realidade, não é exatamente a classe que possui os atributos, mas sim as instâncias, os objetos dessa classe. Não é realmente possível trabalhar com uma classe, apenas com suas instâncias.

A classe Pessoa, por exemplo, não existe realmente; ela é uma abstração, uma forma de classificar e identificar um grupo de objetos semelhantes. Nunca poderemos trabalhar com a classe Pessoa propriamente dita, mas somente com suas instâncias, como, por exemplo, João, Pedro ou Paulo, que são nomes que identificam três objetos da classe Pessoa. A Figura 2.2 demonstra um exemplo de classe com atributos.



Figura 2.2 – Exemplo de Classe com Atributos.

O exemplo aqui apresentado toma a mesma classe ilustrada na Figura 2.1, desta vez contendo uma segunda divisão que armazena os atributos da classe carro, que, neste caso, são cor e número de portas. Não foram definidos os tipos dos atributos por ainda não serem necessários.

2.4 Métodos

Classes costumam possuir métodos, que representam as atividades que um objeto de uma classe pode executar. Um método pode receber ou não parâmetros (valores que são utilizados durante a execução do método) e pode retornar valores. Esses valores podem representar o resultado da operação executada ou simplesmente indicar se o processo foi concluído com sucesso ou não. Assim, um método representa um conjunto de instruções que são executadas quando o método é chamado. Um objeto da classe Carro, por exemplo, pode executar a atividade de transportar pessoas. Os métodos são armazenados na terceira divisão de uma classe. A Figura 2.3 apresenta um exemplo de uma classe com métodos.

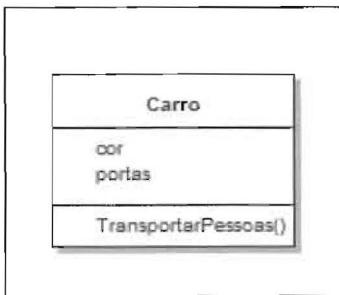


Figura 2.3 – Exemplo de Classe com Métodos.

Novamente tomamos a mesma classe Carro, ilustrada na Figura 2.1, e acrescentamos uma terceira divisão para armazenar o método TransportarPessoas().

2.5 Visibilidade

A visibilidade é um símbolo que precede um atributo ou método e é utilizada para indicar seu nível de acessibilidade. Existem basicamente três modos de visibilidade: público, protegido e privado.

- A visibilidade pública é representada por um símbolo de mais (+) e significa que o atributo ou método pode ser utilizado por objetos de qualquer classe.
- A visibilidade protegida é representada pelo símbolo de sustentido (#) e determina que apenas objetos da classe possuidora do atributo ou método ou de suas subclasses podem acessá-lo.
- O atributo privado é representado por um símbolo de menos (-) e significa que somente os objetos da classe possuidora do atributo ou método poderão utilizá-lo.

A Figura 2.4 apresenta um exemplo de atributos e métodos com sua visibilidade representada à esquerda de seus nomes, em que podemos perceber que os atributos cor e portas possuem visibilidade privada, já que apresentam o símbolo de menos à esquerda de sua descrição. O método TransportarPessoas(), por sua vez, possui visibilidade pública, como indica o símbolo de mais acrescentado a sua descrição.

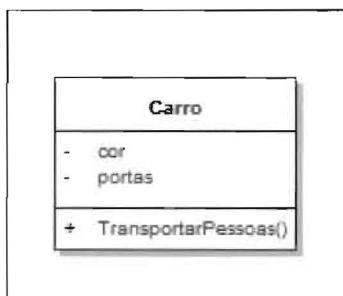


Figura 2.4 – Exemplo de Visibilidade.

2.6 Herança

Juntamente com o Polimorfismo, a Herança é uma das características mais poderosas e importantes da Orientação a Objetos, em razão do fato de esta permitir o reaproveitamento de atributos e de métodos, otimizando o tempo de desenvolvimento, além de permitir a diminuição de linhas de código, bem como facilitar futuras manutenções.

O conceito de Herança trabalha com os conceitos de superclasses e subclasses. Uma superclasse é uma classe que possui classes derivadas dela, chamadas subclasses. As subclasses, ao serem derivadas de uma superclasse, herdam seus atributos e métodos.

A vantagem do uso da herança é óbvia: ao declararmos uma classe com atributos e métodos específicos e após isso derivarmos uma subclass, não precisamos redeclarar os atributos e métodos já definidos; a subclass herda automaticamente, permitindo uma reutilização do código já pronto. Assim, só precisamos nos preocupar em declarar os atributos ou métodos exclusivos da subclass, o que torna muito mais ágil o processo de desenvolvimento, além de facilitar igualmente futuras manutenções, sendo necessário somente alterar o método da superclasse para que todas as subclasses estejam também atualizadas imediatamente. A Figura 2.5 apresenta um exemplo de herança.

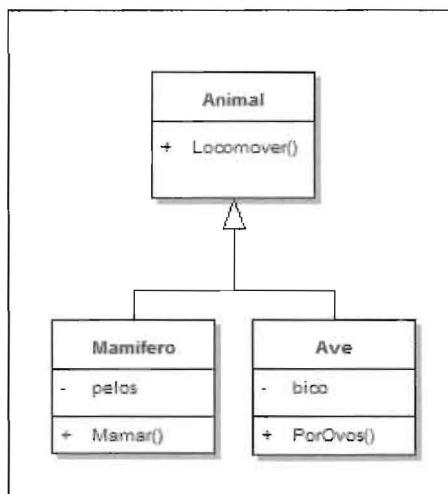


Figura 2.5 – Exemplo de Herança.

Aqui percebemos a existência de uma classe geral chamada `Animal`, e essa classe possui um método chamado `Locomover()`, assim qualquer instância da classe `Animal` pode se locomover de um lugar para outro. A partir da classe `Animal`, derivamos duas subclasses, `Mamífero` e `Ave`. A classe `Mamífero` tem como atributo a existência de pêlos e como método a capacidade de mamar; já a classe `Ave` tem como atributo a existência de um bico e como método a capacidade de por ovos. Ambas, porém, classes possuem a capacidade de se locomover, herdada da superclasse `Animal`.

2.7 Polimorfismo

O conceito de Polimorfismo está associado à Herança. O Polimorfismo trabalha com a redeclaração de métodos previamente herdados por uma classe. Esses métodos, embora semelhantes, diferem de alguma forma da implementação utilizada na superclasse, sendo necessário, portanto, reimplementá-los na subclasse. Porém, para evitar ter de modificar o código-fonte, inserindo uma chamada a um método com um nome diferente, redeclara-se o método com o mesmo nome declarado na superclasse. Dessa maneira, podem existir dois ou mais métodos com a mesma nomenclatura, diferenciando-se na maneira como foram implementados, sendo o sistema responsável por verificar se a classe da instância em questão possui o

método declarado nela própria ou se o herda de uma superclasse. A Figura 2.6 ilustra um exemplo de polimorfismo.

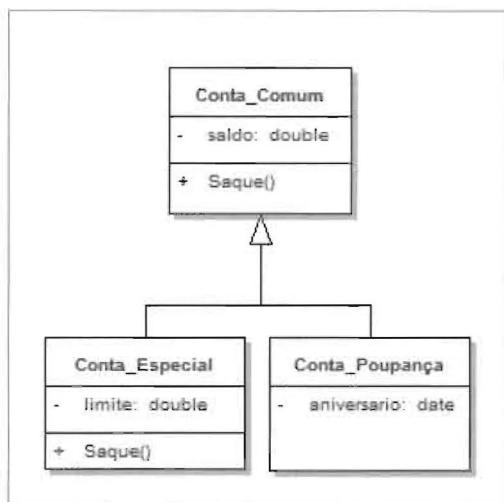


Figura 2.6 – Exemplo de Polimorfismo.

No exemplo apresentado na Figura 2.6, utilizamos uma classe geral chamada `Conta_Comum`. Essa classe possui um atributo chamado `Saldo`, que contém o valor total depositado em uma determinada instância da classe e um método chamado `Saque`. O método `Saque` da classe `Conta_Comum` simplesmente diminui o valor a ser debitado do saldo da conta e, se este não possuir o valor suficiente, a operação deverá ser recusada.

A partir da classe `Conta_Comum` derivamos uma nova classe chamada `Conta_Especial` que possui um atributo próprio, além dos herdados, chamado `Limite`. Esse atributo define o valor extra que pode ser sacado além do valor contido no saldo da conta. Por esse motivo, a classe `Conta_Especial` apresenta uma redefinição do método `Saque`, porque a rotina do método `Saque` da classe `Conta_Especial` não é idêntica à do método `Saque` declarado na classe `Conta_Comum`, já que é necessário incluir o limite da conta no teste para determinar se o cliente pode retirar ou não o valor solicitado. Todavia, o nome do método permanece o mesmo; apenas no momento de executar o método, o sistema deverá verificar se a instância que chamou método pertence à classe `Conta_Comum` ou à classe `Conta_Especial`, executando o método definido na classe em questão.

CAPÍTULO 3

Diagrama de Casos de Uso

Este diagrama procura, por meio de uma linguagem simples, demonstrar o comportamento externo do sistema, buscando apresentar o sistema por uma perspectiva do usuário, demonstrando as funções e os serviços oferecidos e quais usuários poderão utilizar cada serviço. Esse diagrama é, dentre todos os diagramas da UML, o mais abstrato, flexível e informal, sendo utilizado principalmente no início da modelagem do sistema, embora venha a ser consultado e possivelmente modificado durante todo o processo de engenharia e sirva de base para a modelagem de outros diagramas.

3.1 Atores

Os atores representam os papéis desempenhados pelos diversos usuários que poderão utilizar ou interagir com os serviços e funções do sistema. Eventualmente um ator pode representar algum hardware especial ou mesmo um outro software que interaja com o sistema. Assim, um ator pode ser qualquer elemento externo que interaja com o software. Os atores são representados por símbolos de “bonecos magros”, contendo uma breve descrição logo abaixo do seu símbolo que identifica qual é o papel assumido pelo ator dentro do diagrama. A Figura 3.1 apresenta alguns exemplos de atores.

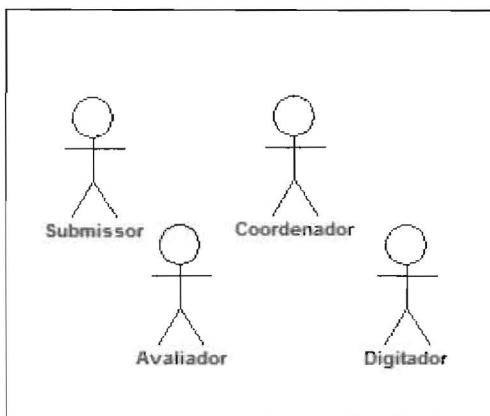


Figura 3.1 – Exemplos de Atores.

3.2 Casos de Uso

Os casos de uso referem-se a serviços, tarefas ou funções oferecidas pelo sistema, como emitir um relatório ou cadastrar a venda de algum produto. São utilizados para expressar e documentar os comportamentos pretendidos para as funções do sistema. A Figura 3.2 apresenta um exemplo de caso de uso, representando o processo de submissão de um trabalho em um congresso.



Figura 3.2 – Exemplo de Caso de Uso.

Os casos de uso costumam ser documentados, demonstrando qual o comportamento pretendido para o caso de uso em questão e quais funções ele executará quando for solicitado.

3.3 Associações

As associações representam os relacionamentos entre os atores que interagem com o sistema, entre os atores e os casos de uso ou os relacionamentos entre os casos de uso e outros casos de uso – estes últimos recebem nomes especiais, como inclusão, extensão e generalização.

Uma associação entre um ator e um caso de uso demonstra que o ator se utiliza, de alguma maneira, da função representada pelo caso de uso. A associação entre um ator e um caso de uso é representada por uma reta ligando o ator ao caso de uso, podendo ocorrer que as extremidades da reta contenham setas indicando a navegabilidade da associação, ou seja, se as informações são fornecidas pelo ator ao caso de uso, se são transmitidas pelo caso de uso ao ator ou ambos (neste último caso a reta não possui setas). Além disso, uma associação pode possuir uma descrição própria quando há necessidade de esclarecer a natureza da informação que está sendo transmitida ou para dar um nome à associação. A Figura 3.3 representa uma associação entre um ator e um caso de uso.

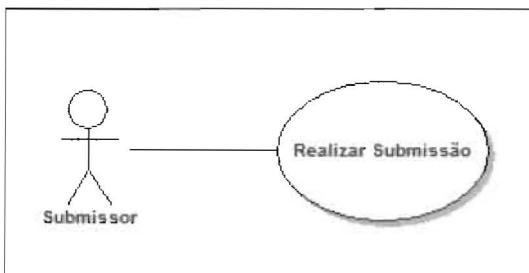


Figura 3.3 – Associação entre um Ator e um Caso de Uso.

Ao examinarmos o exemplo ilustrado pela Figura 3.3, percebemos que o ator denominado Submissor utiliza, de alguma forma, o serviço de Realizar Submissão e que a informação referente a esse processo trafega nas duas direções.

3.4 Especialização/Generalização

Este relacionamento é uma forma de associação entre casos de uso na qual existem dois ou mais casos de uso com características semelhantes, apresentando pequenas diferenças entre si. Quando tal situação ocorre,

costuma-se definir um caso de uso geral, que descreve as características compartilhadas por todos os casos de uso em questão, e então relacioná-lo com estes, cuja documentação conterá somente as características específicas de cada um. Assim, não é necessário colocar a mesma documentação para todos os casos de uso envolvidos, porque toda a estrutura de um caso de uso generalizado é herdada pelos casos de uso especializados, incluindo quaisquer possíveis associações. A associação é representada por uma reta com uma seta mais grossa, partindo dos casos de uso especializados e atingindo o caso de uso geral. A Figura 3.4 apresenta um exemplo de especialização/generalização.

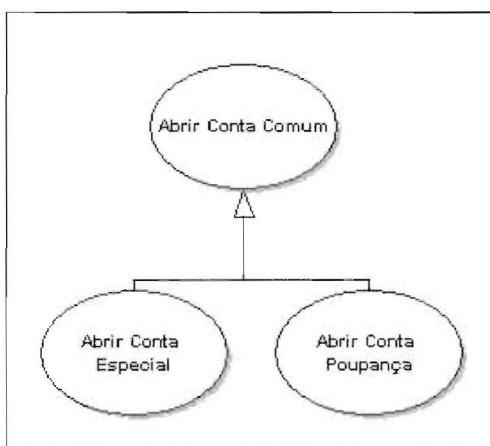


Figura 3.4 – Especialização/Generalização.

Nesse exemplo existem três opções de abertura de conta muito semelhantes entre si: abertura de conta comum, de conta especial e de conta poupança, cada uma representada por um caso de uso diferente. As aberturas de conta especial e de conta poupança possuem todas as características e requisitos da abertura de conta comum, porém, possuem também algumas características e requisitos próprios, o que justifica colocá-las como especializações do caso de uso Abertura de Conta Comum, detalhando-se as particularidades de cada caso de uso especializado em sua própria documentação. Esse relacionamento também pode ser aplicado sobre atores. A Figura 3.5 apresenta um exemplo, em que existe um ator geral chamado Pessoa e dois atores especializados chamados Pessoa Física e Pessoa Jurídica.

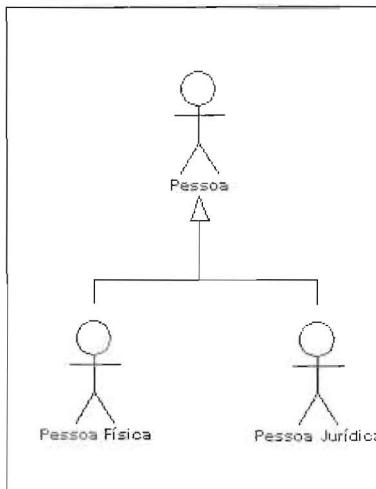


Figura 3.5 – Especialização/Generalização com Atores.

3.5 Inclusão

Esta associação é utilizada quando existe uma situação ou rotina comum a mais de um caso de uso. Quando isso ocorre, a documentação dessa rotina é colocada em um caso de uso específico para que outros casos de uso se utilizem desse serviço, evitando-se descrever uma mesma seqüência de passos em vários casos de uso. Os relacionamentos de inclusão indicam uma obrigatoriedade, ou seja, quando um determinado caso de uso possui um relacionamento de inclusão com outro, a execução do primeiro obriga também a execução do segundo. Um relacionamento de inclusão pode ser comparado à chamada de uma sub-rotina.

Uma associação de inclusão é representada por uma reta tracejada contendo uma seta em uma de suas extremidades que aponta para o caso de uso incluído pelo caso de uso que o incluiu, posicionado na outra extremidade. As associações de inclusão apresentam também um estereótipo contendo o texto “<<include>>”. Estereótipos são amplamente utilizados pela UML; o objetivo é dar um destaque especial a um componente ou relacionamento, atribuindo a este características extras ou diferentes de seus iguais, podendo inclusive modificar seu desenho-padrão. Um exemplo de inclusão pode ser visto na Figura 3.6.

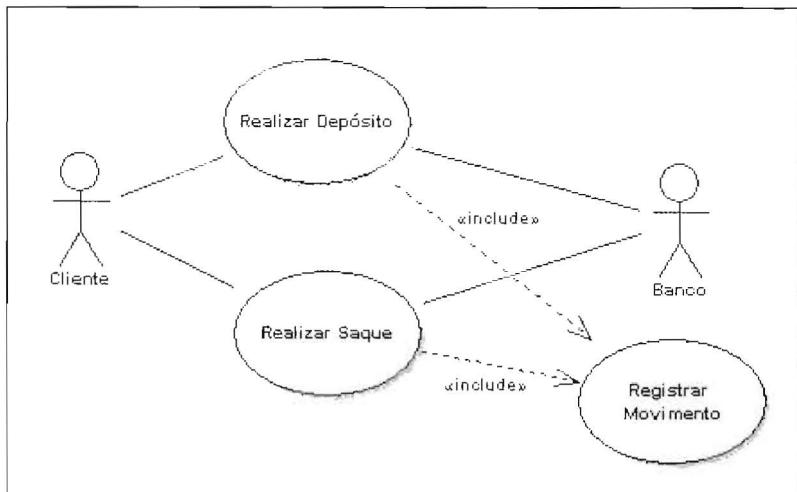


Figura 3.6 – Inclusão.

Ao analisarmos esse exemplo, percebemos que, sempre que um saque ou depósito ocorrer, ele deve ser registrado para fins de histórico bancário. Como as rotinas para registro de um saque ou um depósito são extremamente semelhantes, colocou-se a rotina de registro em um caso de uso à parte, chamado Registrar Movimento, que será executado obrigatoriamente sempre que os casos de uso Depósito ou Saldo forem utilizados. Assim, só é preciso descrever os passos para registrar um movimento no caso de uso incluído.

3.6 Extensão

Esta associação é utilizada para descrever cenários opcionais de um caso de uso, que somente ocorrerão se uma determinada condição for satisfeita. As associações de extensão possuem uma representação muito semelhante às associações de inclusão, sendo também representadas por uma reta tracejada, diferenciando-se por possuir um estereótipo contendo o texto “<<extend>>” e pela seta apontar para o caso de uso que estende. A Figura 3.7 apresenta um exemplo de extensão.

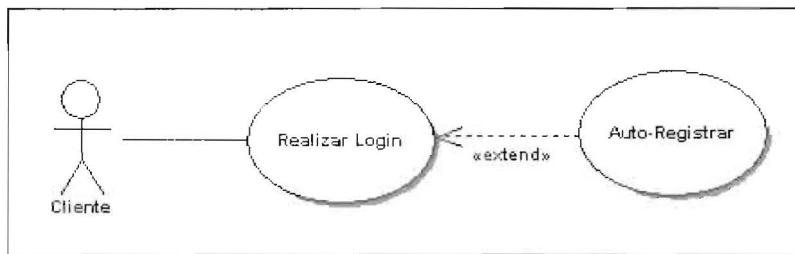


Figura 3.7 – Extensão.

Aqui enfocamos um caso comum em sistemas para Internet, em que, para utilizar os serviços, o cliente deve se logar no sistema e, caso não esteja logado, deverá se registrar. Nesse exemplo, o caso de uso Realizar Login pode estender o caso de uso Auto-Registrar se o ator Cliente ainda não estiver registrado.

3.7 Restrições em Associações de Extensão

Restrições são compostas por um texto entre chaves e são utilizadas para definir validações, consistências, condições etc., que devem ser aplicadas a um determinado componente ou situação. Às vezes, não fica claro qual é a condição para que um caso de uso estendido seja executado, assim, pode-se acrescentar uma restrição à associação de extensão por meio de uma nota explicativa, determinando a condição para que o caso de uso seja executado, conforme apresentado na Figura 3.8.



Figura 3.8 – Associação de Extensão com Restrição.

Nesse exemplo, o caso de uso Auto-Registrar só será executado se o cliente não estiver registrado. O componente nota explicativa é utilizado para fornecer comentários ou restrições sobre um componente ou relacionamento. A seta tracejada que o une ao componente é chamada âncora.

3.8 Pontos de Extensão

Podem ocorrer situações em que somente parte do comportamento de um caso de uso estendido seja utilizado pelo caso de uso que o estendeu. Nesses casos, são utilizados pontos de extensão que especificam qual comportamento está sendo estendido, conforme mostra a Figura 3.9.

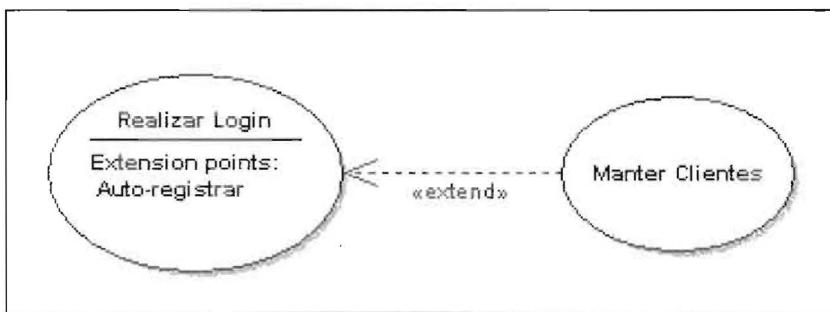


Figura 3.9 – Exemplo de Ponto de Extensão.

Aqui modificamos o exemplo apresentado na Figura 3.7, de modo que o caso de uso Realizar Login possui uma associação de extensão com o caso de uso Manter Clientes. Esse caso de uso permite a consulta, o auto-registro, a alteração e a exclusão de clientes, no entanto, o caso de uso Realizar Login poderá somente utilizar a função referente ao auto-registro do cliente, conforme destaca o ponto de extensão do caso de uso Realizar Login.

3.9 Multiplicidade no Diagrama de Casos de Uso

A Multiplicidade em uma associação entre um ator e um caso de uso basicamente especifica o número de vezes que um ator pode utilizar um determinado caso de uso. A Figura 3.10 apresenta um exemplo de multiplicidade em associações entre atores e casos de uso.

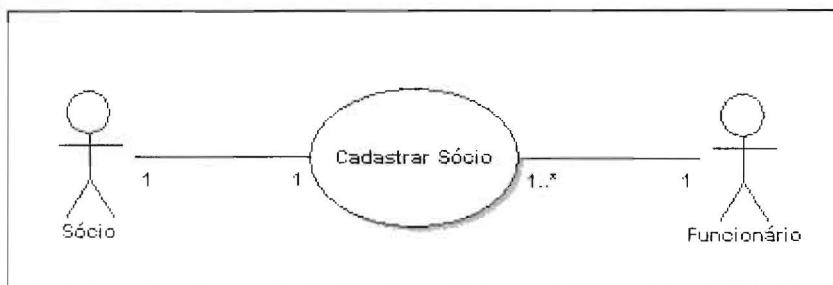


Figura 3.10 – Exemplos de Multiplicidade em Associações entre Atores e Casos de Uso.

Ao examinarmos esse exemplo percebemos que um Sócio utiliza o caso de uso Cadastrar Sócio somente uma vez, ao passo que o ator Funcionário pode utilizá-lo muitas vezes. Da mesma forma, percebemos que esse caso de uso só pode ser utilizado por um sócio e um funcionário por vez.

3.10 Fronteira de Sistema

Uma fronteira de sistema identifica um classificador que contém um conjunto de casos de uso. Uma fronteira de sistema permite identificar um subsistema ou mesmo um sistema completo, além de destacar o que está contido no sistema e o que não está. A fronteira do sistema será apresentada na Seção 3.11, referente ao estudo de caso trabalhado neste livro.

3.11 Documentação de Casos de Uso

A documentação de um caso de uso costuma descrever, por meio de uma linguagem bastante simples, a função em linhas gerais do caso de uso, destacando quais atores interagem com ele, quais etapas devem ser executadas pelo ator e pelo sistema para que o caso de uso execute sua função, quais parâmetros devem ser fornecidos e quais restrições e validações o caso de uso deve possuir.

Não existe um formato específico de documentação para casos de uso definido pela UML, o que está de acordo com a característica do próprio diagrama, ou seja, o formato de documentação de um caso de uso é bastante flexível, permitindo que se documente o caso de uso da forma que se considerar melhor.

Os casos de uso podem também ser documentados por meio de outros diagramas, como o Diagrama de Seqüência ou o Diagrama de Atividade. A Tabela 3.1 fornece uma sugestão de como documentar o caso de uso representado na Figura 3.2.

Tabela 3.1 – Documentação do Caso de Uso Realizar Submissão

Nome do caso de Uso	Realizar submissão
Caso de uso geral	
Ator Principal	Submissor
Atores secundários	
Resumos	Esse caso de uso descreve as etapas percorridas por um autor para submeter um trabalho ao congresso
Pré-condições	O Submissor precisa estar logado
Pós-condições	
Ações do ator	Ações do sistema
1. Solicitar opção de submissão de trabalhos	
	2. Selecionar temas aceitos pelo congresso
	3. Apresentar tela de submissão contendo os temas aceitos e os tipos de submissão (artigos, minicursos e palestras) válidos
4. Selecionar tema	
	5. Consultar tema selecionado
6. Selecionar tipo de submissão	
7. Informar dados de submissão	
8. Anexar arquivo com o trabalho a ser submetido	
9. Confirmar	
	10. Registrar submissão
	11. Apresentar mensagem de trabalho submetido com sucesso
Restrições/validações	1. Todos os campos são obrigatórios 2. É obrigatório anexar o arquivo do trabalho submetido

Nessa documentação o campo caso de uso geral foi deixado em branco simplesmente porque o caso de uso documentado não o possui, uma vez que não é um caso de uso especializado a partir de outro. O campo ator secundário contém a descrição de outros atores que podem participar do processo, inexistentes nesse exemplo. Os campos pré-condições e pós-condições informam possíveis condições que devem ser satisfeitas antes e depois de o caso de uso ser executado e o campo restrições contém as consistências que devem ser validadas durante o processo.

Alguns autores sugerem a documentação de cenários alternativos em que seriam descritas situações de exceção ou situações em que o ator tivesse de escolher entre mais de uma alternativa, por exemplo. As restrições são muitas vezes chamadas de regras do negócio.

3.12 Estudo de Caso

Procurando fornecer um exemplo mais concreto, iremos enfocar a modelagem de um Sistema para Controle de Submissões de Congresso, que deverá satisfazer as seguintes condições (Detalhamos aqui apenas as informações necessárias para a modelagem do Diagrama de Casos de Uso. Outras informações sobre os requisitos do sistema serão detalhadas no tópico referente ao Diagrama de Classes):

- O sistema aceita submissões sobre diversos temas como Engenharia de Software, Banco de Dados e Hipermídia, sendo necessário, portanto, manter um cadastro de todos os temas aceitos.
- Um autor pode realizar muitas submissões. Uma submissão pode se constituir em um artigo, um minicurso ou uma palestra. As submissões só podem ser realizadas pela Internet. Ao acessar a página de submissão o autor pode se logar, realizar uma submissão ou verificar a situação de trabalhos porventura já submetidos; no entanto, para poder utilizar os dois últimos serviços, ele deverá antes executar o primeiro.
- Um submissor (autor) deve se registrar no sistema antes de poder se logar. Se já estiver registrado deverá então logar-se, informando seu nome-login e senha.

- Toda submissão precisa ser avaliada por uma comissão de três avaliadores, responsável por analisá-la e fornecer notas. Um avaliador pode avaliar muitas submissões, as quais são aprovadas de acordo com as maiores notas gerais. A nota geral de uma submissão será o resultado da média de todas as notas das avaliações de cada submissão. As n melhores notas de cada tema e tipo serão consideradas aprovadas. É necessário manter um cadastro de todos os avaliadores do congresso.
- É responsabilidade do coordenador do evento definir quais serão os avaliadores das diferentes submissões. É também responsabilidade do coordenador notificar os submissores sobre a aceitação ou não de suas submissões no evento.
- O coordenador pode emitir o relatório das avaliações sempre que quiser; no entanto, a partir do momento em que selecionar a opção notificar submissores, estes serão avisados se suas submissões foram ou não aprovadas.
- Sendo ou não aprovada, uma submissão pode ou não receber comentários dos avaliadores referentes a possíveis alterações necessárias antes de a submissão ser publicada/disponibilizada no congresso ou conter informações relativas ao porquê da não aprovação do trabalho pelo avaliador.
- Um submissor pode consultar o estado de suas submissões, ou seja, se elas estão ainda sob avaliação, foram aprovadas ou reprovadas. Um submissor pode também, se assim o desejar, verificar os possíveis comentários dos avaliadores a respeito de uma submissão específica.

A Figura 3.11 apresenta uma solução para o problema descrito anteriormente. Nesse exemplo utilizamos uma fronteira de sistema, separando os atores, que, na verdade, não fazem realmente parte do sistema, apenas interagem com ele. Os atores identificados foram: Submissor, Digitador, Avaliador e Coordenador.

O ator denominado Submissor pode utilizar os serviços de Realizar Login Submissor, Auto-registrar, Realizar Submissão, Verificar Submissões e Verificar Comentários. Observe que o Submissor somente poderá utilizar o caso de uso Auto-registrar a partir do caso de uso Realizar Login, e assim mesmo, se ainda não estiver registrado, conforme demonstra o relaciona-

CAPÍTULO 4

Diagrama de Classes

O diagrama de classes é o mais utilizado da UML. Seu objetivo é permitir a visualização das classes utilizadas pelo sistema e como estas se relacionam. Esse diagrama apresenta uma visão estática de como as classes estão organizadas, preocupando-se em definir sua estrutura lógica.

Nesse diagrama, a abstração de cada classe com seus atributos e métodos individualmente corresponde à modelagem de vocabulário, em que são definidas as classes que farão parte do diagrama, ao passo que a definição de como as classes se relacionam e colaboram para a execução de um determinado processo corresponde à modelagem de colaboração. Um modelo de colaboração normalmente não enfoca os relacionamentos entre todas as classes que são abstraídas no projeto, apenas as que colaboram de alguma maneira para a execução de um processo específico.

Um diagrama de classes pode ser utilizado para definir o modelo lógico de um banco de dados, quando se assemelha aos antigos modelos entidade-relacionamento. Na verdade, o diagrama de classes foi propositalmente projetado para ser uma evolução do modelo E-R. No entanto, deve ficar bem claro que o diagrama de classes não é utilizado unicamente para o projeto de modelos lógicos de bancos de dados e mais importante, que uma classe não corresponde a uma tabela em um banco de dados.

Eventualmente uma classe pode representar o repositório lógico dos atributos de uma tabela, no entanto, a classe não é a tabela, uma vez que os atributos de seus objetos são armazenados em memória , ao passo que uma tabela armazena seus registros fisicamente em disco. Além disso, uma

classe possui métodos, que não existem em uma tabela. Em um modelo lógico de banco de dados, os métodos de uma classe podem corresponder às operações que serão realizadas sobre a tabela.

É importante destacar a existência de classes persistentes e não-persistentes, também chamadas de transientes. Uma classe persistente é uma classe cujos objetos precisam ser preservados fisicamente de alguma maneira, o que não ocorre em classes transientes, cujos objetos são destruídos durante a execução do sistema ou quando este é encerrado. Além disso, é possível encontrar diagramas contendo tanto classes persistentes como transientes. Nessas situações, é preciso destacar, por meio de estereótipos ou restrições, quais classes são persistentes e quais não são.

Normalmente uma classe dita persistente em um modelo lógico terá um “correspondente” na forma de uma tabela que preservará os atributos de seus objetos, de maneira que estes possam ser recuperados. Isto, porém, não é uma regra; a forma como os objetos de uma classe persistente serão preservados irá depender da forma como o software for desenvolvido. Alguns autores recomendam o uso de uma camada de persistência, ou seja, a derivação de uma subclasse a partir da classe cujas informações necessitam ser mantidas, em que seria definida a maneira como as instâncias de cada classe seriam preservadas.

4.1 Relacionamentos ou Associações

As classes costumam possuir relacionamentos entre si, chamados associações, que as permitem compartilhar informações e colaborar para a execução dos processos pelo sistema. Uma associação descreve um vínculo que ocorre normalmente entre os objetos de uma ou mais classes.

As associações são representadas por retas ligando as classes envolvidas, podendo também possuir setas em suas extremidades para indicar a navegabilidade da associação, o que representa o sentido em que as informações são transmitidas entre os objetos das classes envolvidas. As associações podem também possuir títulos para determinar o tipo de vínculo estabelecido entre os objetos das classes; isto é útil quando é necessária alguma forma de esclarecimento – nesse caso é recomendável determinar também a navegabilidade da associação para facilitar o sentido da leitura.

4.1.1 Associação Unária ou Reflexiva

Este tipo de associação ocorre quando existe um relacionamento de um objeto de uma classe com objetos da mesma classe. Um exemplo de associação unária pode ser observado na Figura 4.1.

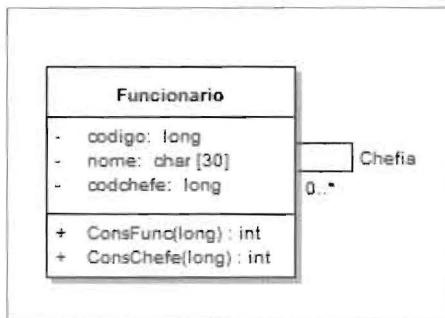


Figura 4.1 – Associação Unária.

Podemos perceber nesse exemplo a existência da classe Funcionário, cujos atributos são o código do funcionário, seu nome e o código do possível chefe do funcionário. Esse chefe também é um funcionário, assim a associação chamada “Chefia” indica uma possível relação entre uma ou mais instâncias da classe Funcionário com outras instâncias da mesma classe, ou seja, tal associação determina que um funcionário pode ou não chefiar outros funcionários. Essa associação faz que a classe possua o atributo codchefe para armazenar o código do funcionário que é responsável pela instância do funcionário em questão. Desse modo, após consultar uma instância da classe Funcionário, pode-se utilizar o atributo codchefe da instância consultada para pesquisar por outra instância da classe.

Note que existe outra informação na associação, além de seu próprio nome, representada pelo valor “0..*”, conhecida como multiplicidade. Esta procura determinar o número mínimo e máximo de objetos envolvidos em cada extremidade da associação, além de permitir especificar o nível de dependência de um objeto para com os outros.

No caso apresentado na Figura 4.1, a multiplicidade 0..* indica que um determinado funcionário pode chefiar nenhum (0) ou muitos (*) funcionários. Pode-se perceber que só existe multiplicidade em uma das extremidades, por default. Quando não existe multiplicidade explícita,

entende-se que a multiplicidade é “1..1”, significando que um e somente um objeto dessa extremidade da associação se relaciona com os objetos da outra extremidade. Nesse exemplo significa que um funcionário pode ou não chefiar outros funcionários, mas um funcionário possui um e apenas um funcionário como chefe imediato. A Tabela 4.1 demonstra alguns dos diversos valores de multiplicidade que podem ser utilizados em uma associação.

Tabela 4.1 – Exemplos de multiplicidade

Multiplicidade	Significado
0..1	No mínimo zero (nenhum) e no máximo um. Indica que os objetos das classes associadas não precisam obrigatoriamente estar relacionados, mas, se houver relacionamento indica, que apenas uma instância da classe se relaciona com as instâncias da outra classe
1..1	Um e somente um. Indica que apenas um objeto da classe se relaciona com os objetos de outra classe
0..*	No mínimo nenhum e no máximo muitos. Indica que pode ou não haver instâncias da classe participando do relacionamento
*	Muitos. Indica que muitos objetos da classe estão envolvidos no relacionamento
1..*	No mínimo 1 e no máximo muitos. Indica que há pelo menos um objeto envolvido no relacionamento, podendo haver muitos objetos envolvidos
3..5	No mínimo 3 e no máximo 5. Indica que existem pelo menos 3 instâncias envolvidas no relacionamento e que podem ser 4 ou 5, mas não mais do que isso

Na verdade, não é realmente necessário definir o atributo *código*, que armazena o código de cada funcionário e serve como uma chave primária das instâncias da classe. Como regra geral, cada classe ao ser declarada na UML já possui um código interno implícito, não sendo necessário declarar um atributo exclusivamente para isso.

Nem mesmo o atributo *codchefe* seria realmente necessário, já que associações que possuem multiplicidade do tipo muitos (*) em qualquer de suas extremidades forçam a transmissão do atributo-chave contido na classe da outra extremidade da associação. Como o atributo a ser transmitido é um atributo implícito, ele também não precisa ser apresentado na classe para a qual foi transmitido. Assim, não é obrigatório definir um novo atributo se este não possuir outra função que não seja a de identificar de forma exclusiva uma instância de uma classe.

4.1.2 Associação Binária

Associações binárias ocorrem quando são identificados relacionamentos entre objetos de duas classes. A Figura 4.2 demonstra um exemplo de associação binária.

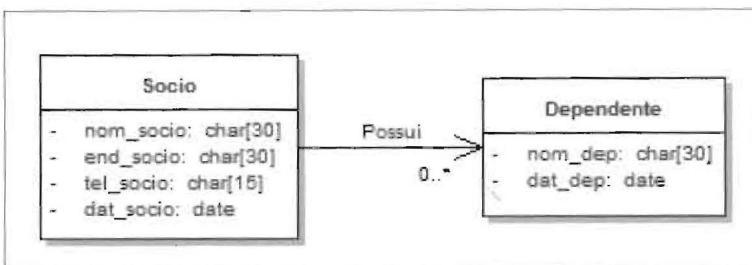


Figura 4.2 – Associação binária.

Nesse exemplo, um objeto da classe Sócio pode se relacionar ou não com instâncias da classe Dependente, conforme demonstra a multiplicidade 0..*, ao passo que, se existir um objeto da classe Dependente, ele terá de se relacionar com um objeto da classe Sócio, pois, como não foi definida a multiplicidade na extremidade da classe Sócio, significa que ela é 1..1. Observe que foi também acrescentada uma descrição e navegabilidade para essa associação, o que nos permite ler a associação da seguinte maneira: Uma instância da classe Sócio possui, no mínimo, nenhuma instância e, no máximo, muitas instâncias da classe Dependente, e uma instância da classe Dependente é possuída por uma e somente uma instância da classe Sócio.

4.1.3 Associação Ternária ou N-ária

Associações Ternárias ou N-árias são associações que conectam objetos de mais de duas classes. São representadas por um losango para o qual convergem todas as ligações da Associação. A Figura 4.3 apresenta um exemplo de Associação N-ária.

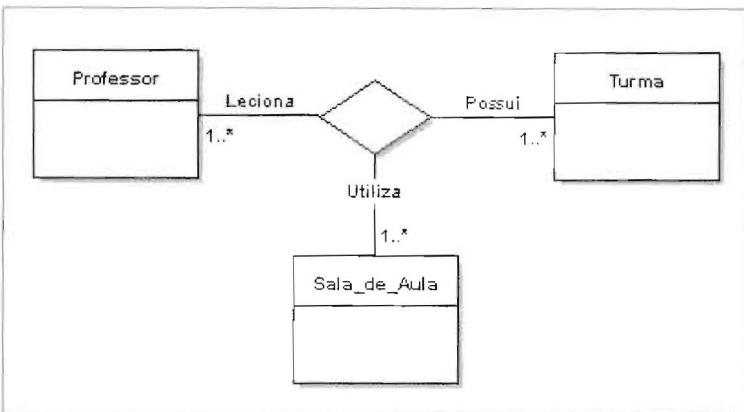


Figura 4.3 – Associação N-ária.

4.1.4 Agregação

Agregação é um tipo especial de associação em que se tenta demonstrar que as informações de um objeto (chamado objeto-todo) precisam ser complementadas pelas informações contidas em um ou mais objetos de outra classe (chamados objeto-parte). Esse tipo de associação tenta demonstrar uma relação todo/parte entre os objetos associados. Objetos-parte não podem ser destruídos por um objeto diferente do objeto-todo. O símbolo de agregação difere do de associação por conter um losango na extremidade da classe que contém os objetos-todo. A Figura 4.4 demonstra um exemplo de agregação, em que existe uma classe Pedido, a qual armazena os objetos-todo e uma classe Item_Pedido, em que são armazenados os objetos-parte.

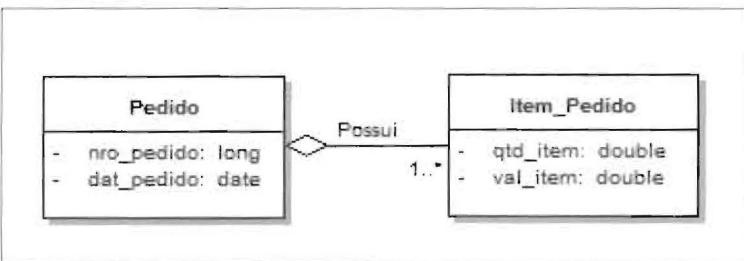


Figura 4.4 – Agregação.

Nesse exemplo, um pedido pode tanto possuir apenas um item como vários e é muito difícil determinar o número máximo de itens que um pedido pode ter. Mesmo que isso fosse possível, na imensa maioria das vezes, o número máximo não seria atingido, o que faria com que diversos atributos fossem deixados em branco, ocupando um espaço desnecessário.

Por isso, as informações da classe Pedido estão incompletas, possuindo apenas atributos que não se repetem. Os atributos que podem se repetir devem ser armazenados em uma classe dependente da classe Pedido. Dessa forma, sempre que uma instância da classe Pedido for pesquisada, todas as instâncias da classe Item_Pedido relacionadas à instância da classe Pedido pesquisada deverão também ser apresentadas. Da mesma forma, sempre que um objeto da classe Pedido for destruído, todos os objetos da classe Item_Pedido a ele relacionados devem também ser destruídos.

4.1.5 Composição

Esta associação é uma variação da agregação, em que é apresentado um vínculo mais forte entre os objetos-todo e os objetos-parte, procurando demonstrar que os objetos-parte têm de estar associados a um único objeto-todo. O símbolo de composição se diferencia graficamente do símbolo de agregação por utilizar um losango preenchido. Da mesma forma que na agregação, o losango deve ficar ao lado do objeto-todo. A Figura 4.5 apresenta um exemplo de composição.

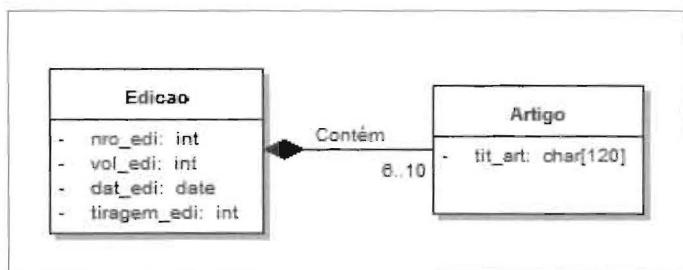


Figura 4.5 – Composição.

Observando-se a Figura 4.5 percebe-se que um objeto da classe **Edição** deve relacionar-se a no mínimo seis objetos da classe **Artigo**, podendo relacionar-se com até dez, no entanto, um objeto da classe **Artigo** refere

unicamente a um objeto da classe Edição, já que uma edição de uma revista científica só deve publicar trabalhos inéditos.

4.1.6 Especialização/Generalização

Esta associação identifica classes-mãe, chamadas gerais, e classes-filhas, chamadas especializadas, demonstrando a ocorrência de herança e possivelmente de métodos polimórficos nas classes especializadas. A Figura 4.6 apresenta um exemplo de especialização/generalização.

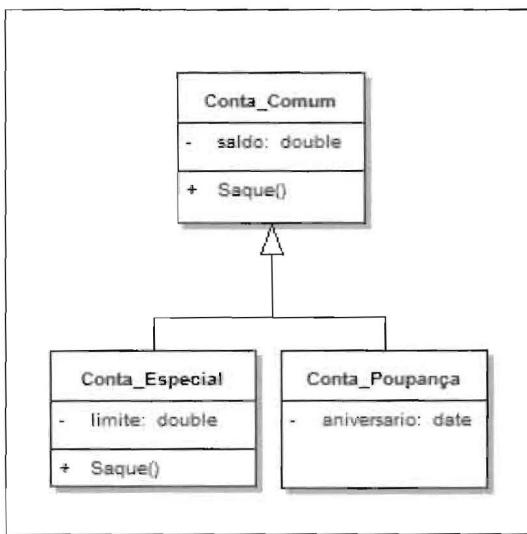


Figura 4.6 – Especialização/Generalização.

Aqui utilizamos um exemplo idêntico ao do capítulo Orientação a Objetos, sendo desnecessário explicá-lo novamente.

4.1.7 Dependência

Este relacionamento, como o próprio nome diz, identifica um certo grau de dependência de uma classe em relação à outra. Tal relacionamento de dependência é representado por uma reta tracejada entre duas classes, contendo uma seta apontando a classe da qual a classe posicionada na outra extremidade do relacionamento é dependente. Ele é utilizado também em diversos outros diagramas, possuindo algumas variações definidas por estereótipos.

4.1.8 Realização

Uma realização é um tipo de relacionamento especial que mistura características dos relacionamentos de generalização e dependência, sendo usada para identificar classes responsáveis por executar funções para classes que representam interfaces. Esse tipo de relacionamento herda o comportamento de uma classe, mas não sua estrutura. O relacionamento de realização é representado por uma seta tracejada contendo uma seta vazia que aponta para a classe de interface, ao passo que na outra extremidade é definida a classe que realiza um comportamento pretendido pela classe de interface. A Figura 4.7 apresenta um exemplo de relacionamento de dependência e realização.

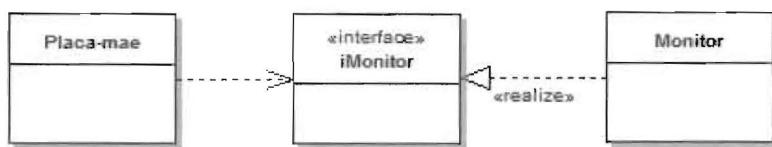


Figura 4.7 – Relacionamento de Dependência e Realização.

Como podemos observar nessa figura, a classe Placa-Mãe possui um relacionamento de Dependência com a classe iMonitor, determinando que a classe Placa-Mãe necessita de alguma forma dessa interface; já a classe Monitor, por sua vez, possui um relacionamento de realização com a classe iMonitor, o que determina que a classe Monitor implementa algum dos serviços oferecidos pela classe iMonitor. Observe que para destacar sua função, a classe iMonitor possui um estereótipo <<interface>>. Podermos ter colocado o estereótipo boundary, mas preferimos utilizar o estereótipo interface para deixar mais clara a função da classe.

4.2 Classe Associativa

Classes associativas são produzidas quando ocorrem associações que possuem multiplicidade muitos (*) em todas as suas extremidades. As classes associativas são necessárias nesses casos porque não existe um repositório que possa armazenar as informações produzidas pelas associações, já que todos os objetos envolvidos apresentam multiplicidade muitos e isto obriga que seu atributo-chave seja transmitido aos outros objetos. Como todos

possuem a mesma multiplicidade, nenhum deles pode receber os atributos dos outros, assim, é preciso criar uma classe associativa para armazenar os atributos transmitidos pela associação, o que não impede que a classe associativa possua atributos próprios. Uma classe associativa é representada por uma seta tracejada partindo do meio da associação e atingindo uma classe. A Figura 4.8 apresenta um exemplo de classe associativa.

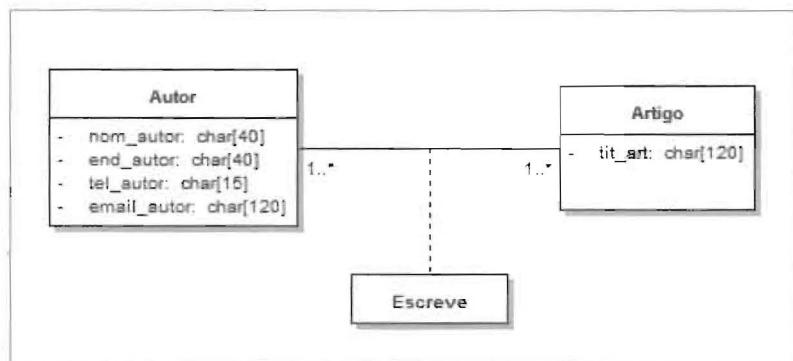


Figura 4.8 – Classe Associativa.

Nesse exemplo, uma instância da classe Autor pode se relacionar com muitas instâncias da classe Artigo, e uma instância da classe Artigo pode se relacionar com muitas instâncias da classe autor. Como existe a multiplicidade muitos nas extremidades de ambas as classes da associação, não há como reservar atributos para armazenar as informações decorrentes da associação, além do que seria um desperdício de espaço fazê-lo, já que não há como determinar um limite para a quantidade de autores que podem escrever um artigo nem há como saber quantos artigos um autor ainda vai escrever. Poderia-se reservar um certo número de atributos em uma das classes para armazenar essa informação, mas, na maioria das vezes, diversos deles seriam deixados em branco e poderia acontecer de, em alguns casos, eles não serem suficientes.

Assim, é preciso criar uma classe para guardar essa informação. Note que a classe associativa desse exemplo não possui atributos, ela simplesmente recebe os atributos-chave da classe Autor e da classe Artigo transmitidos de forma transparente pela associação, não sendo necessário apresentá-los. Pode-se perceber também que a associação não possui uma descrição; essa informação já está contida na própria classe associativa.

Classes associativas podem perfeitamente ser substituídas por classes normais, chamadas, neste caso, classes intermediárias, mas que desempenham exatamente a mesma função das classes associativas. A Figura 4.9 apresenta um exemplo de classe intermediária.

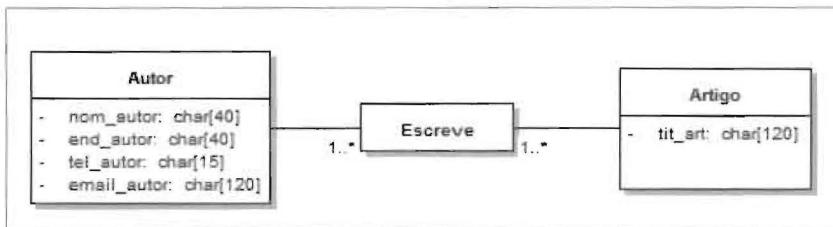


Figura 4.9 – Classe Intermediária.

4.3 Interfaces

Muitas linguagens de programação têm adotado interfaces como um meio para descrever os serviços disponíveis de classes específicas. Na versão 2.0 da UML, as interfaces podem ser de dois tipos: Fornecidas ou Requeridas.

4.3.1 Interfaces Fornecidas

Uma interface fornecida descreve um serviço implementado por uma classe. O conjunto de interfaces implementadas por uma classe são suas interfaces fornecidas e representam o conjunto de serviços que a classe oferece aos seus clientes. Uma classe ao implementar uma interface suporta o conjunto de características possuídas por esta e obedece a suas restrições. As interfaces fornecidas são representadas por um círculo fechado ligado à classe por uma reta sólida. A Figura 4.10 apresenta um exemplo de interface fornecida.

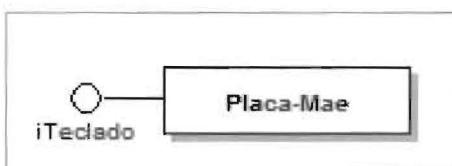


Figura 4.10 – Exemplo de Interface Fornecida.

Nesse exemplo utilizamos uma classe chamada Placa-Mãe, que representa a placa principal de um computador. Essa classe possui uma interface fornecida chamada iTeclado, que representa a interface física entre uma placa-mãe e um teclado, em que a placa-mãe (na verdade um de seus componentes, o processador) é responsável por interpretar as teclas pressionadas no teclado.

4.3.2 Interfaces Requeridas

Descrevem os serviços que outras classes devem fornecer a uma determinada classe, que não precisa ter conhecimento de quais classes irão implementar esses serviços. As interfaces requeridas são representadas por um semicírculo ligado a uma classe por uma linha sólida. A Figura 4.11 apresenta um exemplo de interface requerida.

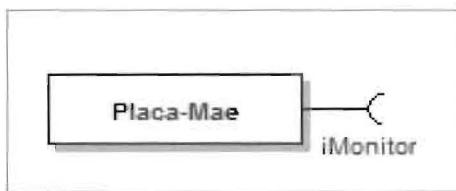


Figura 4.11 – Exemplo de Interface Requerida.

Aqui utilizamos novamente a classe Placa-Mãe, desta vez possuindo uma interface requerida chamada iMonitor, que representa a interface física entre uma placa-mãe e um monitor, em que o monitor deve apresentar no vídeo as imagens e símbolos solicitados pelo processador. É comum uma interface fornecida em uma classe ser uma interface requerida em outra, podendo facilmente ocorrer de ambas surgirem juntas, como demonstra a Figura 4.12.

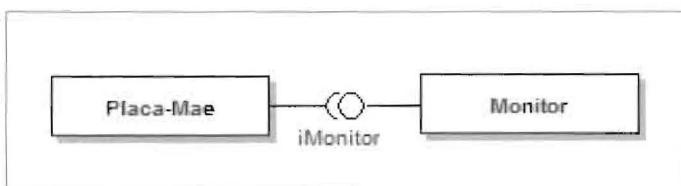


Figura 4.12 – Exemplo de Interface Fornecida e Requerida.

No exemplo da Figura 4.12, tanto a classe Placa-Mãe como a classe Monitor possuem a mesma interface, no entanto, esta é requerida pela primeira e fornecida pela última. É importante notar que as interfaces requeridas e fornecidas podem ser substituídas pelos relacionamentos de dependência e realização já existentes nas versões anteriores, conforme demonstrado na Figura 4.7.

4.4 Restrição

Restrições constituem-se em informações extras que definem condições a serem validadas durante a implementação dos métodos de uma classe, das associações entre as classes ou mesmo de seus atributos. As restrições são representadas por textos limitados por chaves. A Figura 4.13 apresenta um exemplo de restrição.

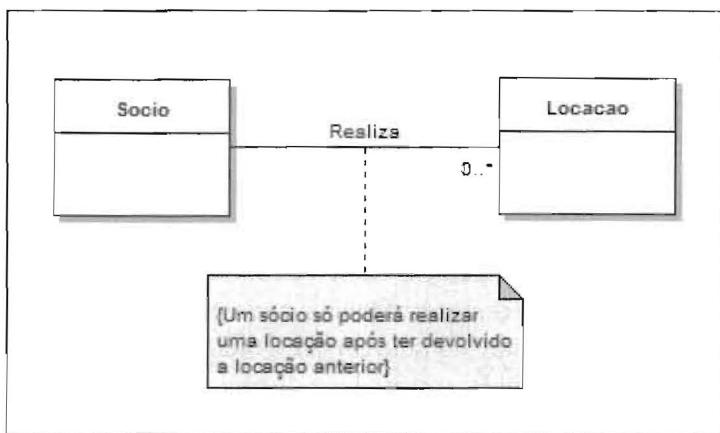


Figura 4.13 – Exemplo de Restrição em uma Associação.

Nesse exemplo, utiliza-se uma restrição para determinar que um sócio somente poderá realizar uma locação se não possuir nenhuma locação anterior pendente. No exemplo utilizamos uma notação informal e de fácil compreensão, porém, existe uma linguagem utilizada para a definição de restrições mais complexas, semelhante a uma linguagem de programação, chamada OCL (Object Constraint Language – Linguagem de Restrição de Objeto).

Restrições podem também ser utilizadas para representar o “ou” exclusivo (xor), quando instâncias de duas ou mais classes podem se relacionar com instâncias de uma outra classe específica, mas somente uma instância de uma das classes pode se relacionar com uma instância da classe específica, em detrimento das outras. Um exemplo desse tipo de restrição é apresentado na Figura 4.14.

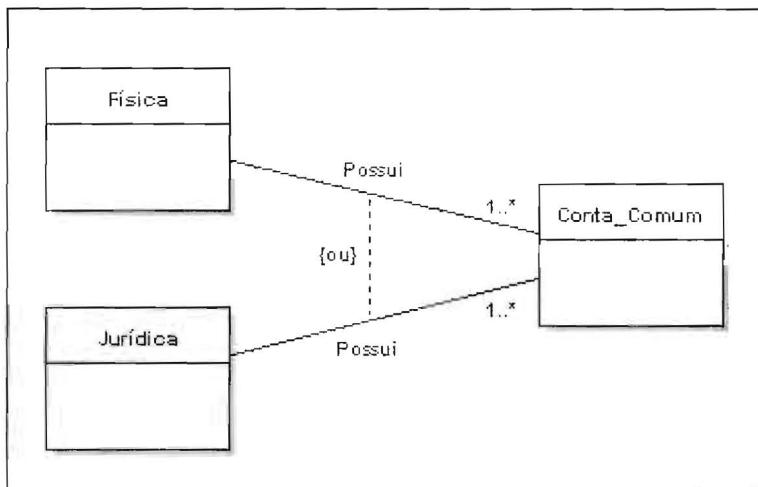


Figura 4.14 – Exemplo de Restrição com Ou Exclusivo.

As restrições podem ainda ser utilizadas para definir melhor a semântica de classes especializadas derivadas de classes gerais. As restrições predefinidas para classes especializadas são:

- **Completa** – quando todas as subclasses possíveis foram derivadas da classe geral.
- **Incompleta** – quando ainda é possível derivar novas subclasses.
- **Separada ou disjunta** – quando as subclasses são mutuamente exclusivas, ou seja, no momento que uma instância pertence a uma subclass, não poderá pertencer a qualquer das outras subclasses. A Figura 4.15 fornece um exemplo de classes especializadas disjuntas e completas.

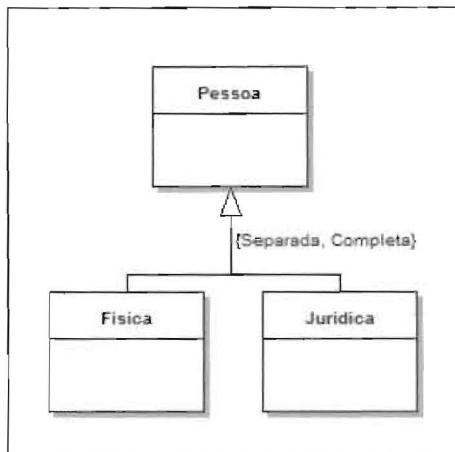


Figura 4.15 – Exemplo de Restrição Separada e Completa.

Nesse exemplo, foram derivadas duas subclasses a partir da classe Pessoa: as classes Física e Jurídica. Dentro do escopo em que estão inseridas, não existem mais classes que possam ser derivadas a partir da classe Pessoa, por isso a restrição entre as classes generalizadas é completa. Além disso, no momento em que uma pessoa for física, ela não pode ser jurídica e vice-versa, portanto a especialização também é separada.

- **Sobreposta** – quando o fato de pertencer a uma subclasse não impede que pertença a outras, a Figura 4.16 apresenta um exemplo de restrição sobreposta e incompleta para classes especializadas.

A Figura 4.16 apresenta um exemplo em que, a partir da classe Veículo, derivou-se duas subclasses Aéreo e Aquático. Como poderia-se ainda derivar uma classe para veículos terrestres, colocou-se uma restrição incompleta e, como pode ocorrer de um veículo ser tanto aéreo como aquático, como é o caso de um hidroavião, acrescentou-se ainda a restrição de sobreposta à especialização.

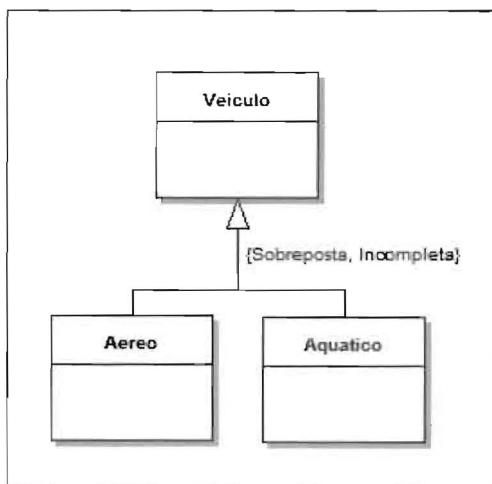


Figura 4.16 – Exemplo de Restrição Sobreposta e Incompleta.

4.5 Estereótipos

Estereótipos são uma maneira de destacar ou diferenciar um componente ou relacionamento de outros componentes ou relacionamentos iguais, atribuindo-lhe características especiais ou modificando-as de alguma forma. Estereótipos podem ser de texto, quando não modificam o desenho-padrão do componente, ou gráficos quando modificam a forma-padrão do componente. No diagrama de classes existem três estereótipos que merecem ser destacados neste tópico.

4.5.1 Estereótipo <<entity>>

O estereótipo <<entity>> tem por objetivo tornar explícito que uma classe é uma entidade, ou seja, a classe contém informações recebidas ou geradas por meio do sistema. Classes com o estereótipo <<entity>> também fornecem a informação de que normalmente possuirão muitos objetos e que estes possivelmente terão um período de vida longo, existindo a possibilidade de que os objetos dessas classes precisem ser preservados. Porém, como isto não é uma regra, é melhor deixar explícito quando uma classe é persistente por meio de um estereótipo próprio ou uma restrição. Esse é um estereótipo gráfico, que modifica a forma-padrão da classe, conforme pode ser observado na Figura 4.17.

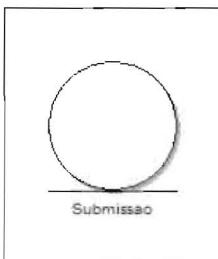


Figura 4.17 – Classe com estereótipo <<entity>>.

4.5.2 Estereótipos <<boundary>> e <<control>>

O estereótipo <<boundary>>, ou estereótipo de fronteira, identifica uma classe que serve de comunicação entre os atores externos e o sistema. Muitas vezes uma classe <<boundary>> é associada à própria interface do sistema. É comum que uma classe do tipo <<boundary>> precise interagir com outra classe do tipo <<control>>. A utilização de classes <<boundary>> é importante quando é preciso definir a existência de uma interface para o sistema, sendo desnecessária em sistemas muito simples cujas interfaces não apresentam nenhuma característica especial.

O estereótipo <<control>> identifica classes que servem como intermediárias entre as classes <<boundary>> e as outras classes do sistema. Os objetos <<control>> são responsáveis por interpretar os eventos ocorridos sobre os objetos <<boundary>>, como os movimentos do mouse ou o pressionamento de um botão, e retransmiti-los para os objetos das classes de entidade que compõem o sistema. A Figura 4.18 apresenta um exemplo de classes com estereótipos <<boundary>> e <<control>>.

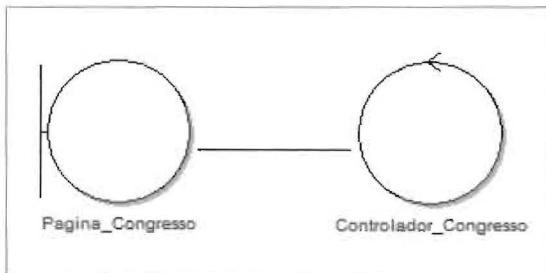


Figura 4.18 – Classes com estereótipos <<boundary>> e <<control>>.

Nesse exemplo, a classe <<boundary>> representa uma página para submissão de trabalhos em um congresso, ao passo que a classe <<control>> assume o papel de controlador do congresso, sendo seus objetos responsáveis por interpretar os eventos gerados pelos usuários da página e repassá-los aos outros objetos que participam do sistema. Note que esses estereótipos também são gráficos e modificam o formato-padrão do componente.

4.6 Estudo de Caso

Aqui daremos continuidade à modelagem do Sistema para Controle de Submissões de Congresso, desta vez modelando o problema por meio do Diagrama de Classes. Para modelar esse diagrama, além das informações já fornecidas no tópico 1.3.1, devemos levar em consideração também as seguintes características do sistema:

- Existem diversos temas. Um tema pode possuir muitas submissões, mas uma submissão refere-se a um tema único.
- Existem três tipos válidos de submissão: artigos, minicursos ou palestras. Pode haver muitas submissões de um determinado tipo, porém uma submissão deve-se referir a um único tipo.
- Um autor (submissor) pode realizar muitas submissões, no entanto, apesar de uma submissão poder ser elaborada por mais de um autor, somente um deles deverá ficar responsável pela submissão, e somente este será notificado da aceitação ou não de sua submissão.
- Existem diversos avaliadores, responsáveis por analisar e fornecer notas para as submissões que lhes forem atribuídas. Um avaliador pode avaliar muitas submissões e cada submissão deve ser analisada por três avaliadores. Cada avaliação deve pontuar os seguintes quesitos: aderência aos temas do evento, ausência de marketing, clareza de apresentação, originalidade e qualidade técnica.
- Cada avaliação pode conter comentários fornecidos pelo avaliador, que podem representar possíveis alterações necessárias antes que a submissão seja disponibilizada no congresso ou esclarecimentos do porquê da recusa da submissão. Uma avaliação pode conter muitos comentários, mas um comentário refere-se unicamente a uma avaliação.

- Cada um dos tipos de submissão possui atributos próprios, tendo como atributos comuns somente o título da submissão e sua situação (se está sendo avaliada, foi aprovada ou reprovada). Submissões do tipo artigo possuem como atributos próprios o resumo e o abstract do artigo, além do arquivo contendo o artigo propriamente dito. Já submissões do tipo minicurso possuem os seguintes atributos particulares: justificativa, objetivo, duração, público-alvo, materiais necessários, o currículo do autor proponente e o arquivo contendo um resumo do que será visto no minicurso. Finalmente as submissões do tipo palestra têm como atributos próprios o objetivo, o resumo e o currículo do autor.

A Figura 4.19 apresenta um Diagrama de Classes em que foram modeladas as classes necessárias para solucionar, no contexto estrutural, o problema apresentado anteriormente. Destacamos apenas os métodos e atributos que julgamos importantes para não deixar o diagrama poluído demais.

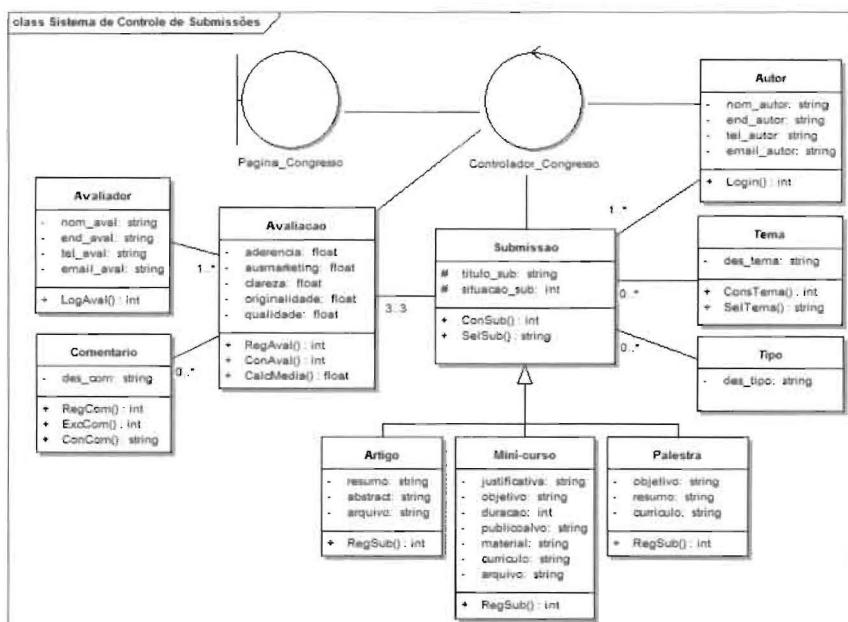


Figura 4.19 – Diagrama de Classes do Sistema de Controle de Submissões.

Nesse diagrama primeiramente identificamos duas classes que não são do tipo entidade: a primeira representa a página de submissão, possuindo um estereótipo do tipo <>boundary><, uma vez que representa a interface entre o sistema e os atores; a segunda classe representa a classe controladora, conforme demonstra seu estereótipo <>control><, responsável por interpretar os eventos ocorridos na página e executar os procedimentos e métodos adequados. Esta classe está associada às classes Submissão, Autor e Avaliações, que são as principais classes do modelo e que terão maior interação com a classe controladora. Poderia haver mais ligações, mas como isso deixaria o diagrama muito poluído, optamos por associar a classe controladora somente com as classes mais importantes.

Em seguida passaremos para a principal classe do sistema, a classe Submissão, cuja descrição é auto-explicativa e cujos atributos são o título da submissão e sua situação (em avaliação, aprovada ou reprovada), além dos métodos para consultar uma submissão e para selecionar várias delas. Observe que Submissão é uma superclasse possuindo três subclasses chamadas Artigo, Minicurso e Palestra. Optamos por essa abordagem porque cada tipo de submissão possui atributos diferentes, conforme foi descrito no enunciado deste estudo de caso. Por este motivo o método para registrar a submissão foi declarado em cada classe.

A classe Autor contém as informações pessoais de cada autor (submissor) que submeteu trabalhos no congresso. O único método que achamos importante destacar aqui é o método Login. Conforme demonstra a multiplicidade da associação desta classe com a classe Submissão, um objeto da classe Autor pode se associar a muitos objetos da classe Submissão (no mínimo um), porém um objeto da classe Submissão só pode estar associado a um objeto da classe Autor (pode haver vários autores para uma submissão, mas somente um será notificado).

A classe Tema possui como atributo somente a descrição do tema e os métodos para consultar um tema e selecionar vários temas, que serão demonstrados nos próximos tópicos. Um tema pode estar associado a nenhuma ou muitas submissões, mas uma submissão só pode se associar a um tema.

Criamos também a classe Tipo, no entanto, declaramos apenas o atributo para conter sua descrição; não detalhamos qualquer método por acreditarmos serem óbvios e, uma vez que existem somente três tipos, essa classe talvez nem seja realmente necessária.

A classe Avaliação possui como atributos os tópicos a serem avaliados em cada submissão. Conforme demonstra a multiplicidade de suas associações, uma avaliação se associa a somente uma submissão, porém uma ~~associação~~ deve se associar a três avaliações. Além disso, uma avaliação se associa a somente um avaliador e pode se associar a muitos comentários. Identificamos aqui os métodos para registrar uma avaliação, consultá-la e para calcular a nota geral de uma submissão.

A classe Avaliador identifica os atributos pessoais de cada avaliador, além do método que permite um avaliador logar-se no sistema. Finalmente, a classe Comentário armazena a descrição de cada comentário de uma avaliação, além dos métodos para registrá-lo, excluí-lo ou consultá-lo.

Todas as classes aqui descritas são classes do tipo entidade (entity), com exceção da classe que representa a página do congresso, que é do tipo boundary, e da controladora do sistema, que pertence ao tipo control. Preferimos, entretanto, não atribuir-lhes o estereótipo entity, por que este, por ser gráfico, modifica o desenho-padrão do componente e suprime seus atributos e métodos.

CAPÍTULO 5

Diagrama de Objetos

O diagrama de objetos tem como objetivo fornecer uma “visão” dos valores armazenados pelos objetos das classes definidas no diagrama de classes em um determinado momento da execução de um determinado processo do sistema. Assim, embora o diagrama de classes seja estático, pode-se criar diagramas de objetos, em que as possíveis situações pelas quais os objetos das classes passarão possam ser simuladas.

5.1 Objeto

Um componente objeto é bastante semelhante a um componente classe, porém os objetos não apresentam métodos, somente atributos e estes armazenam os valores possuídos pelos objetos em uma determinada situação. O nome dos objetos está contido, como nas classes, na primeira divisão do retângulo que representa os objetos e pode ser apresentado de três formas:

- O nome do objeto, com todas as letras minúsculas, seguido do símbolo de dois pontos (:) e o nome da classe a qual o objeto pertence, com as letras iniciais maiúsculas. Este é o formato mais completo.
- O nome do objeto omitido, mas mantendo o símbolo de dois pontos e o nome da classe.
- Somente o nome do objeto, sem dois pontos.

A Figura 5.1 apresenta um exemplo de objeto chamado sub1, pertencente à classe Submissão.

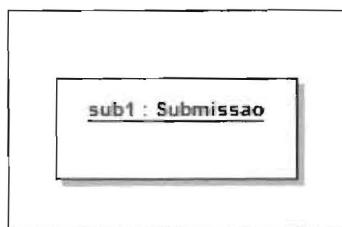


Figura 5.1 – Exemplo de Objeto.

5.2 Vínculos

Os objetos de um diagrama de objetos possuem vínculos entre si. Esses vínculos nada mais são do que instâncias das associações entre as classes representadas no diagrama de classes, assim como os objetos são instâncias das próprias classes. Um vínculo possui exatamente o mesmo símbolo utilizado pelas associações do diagrama de classes, não possuindo multiplicidade, porque esta especifica justamente o número de instâncias de uma determinada classe que podem estar envolvidas em uma associação. Assim, um vínculo em um diagrama de objetos liga apenas um único objeto em cada extremidade. A Figura 5.2 apresenta um exemplo de vínculo entre objetos.

Nesse exemplo, criamos uma visão por meio do diagrama de objetos referente aos vínculos entre um objeto da classe Submissão e os objetos da classe Avaliação a ele relacionados, assim como os vínculos existentes entre um objeto da classe Avaliação com um objeto da classe Avaliador.

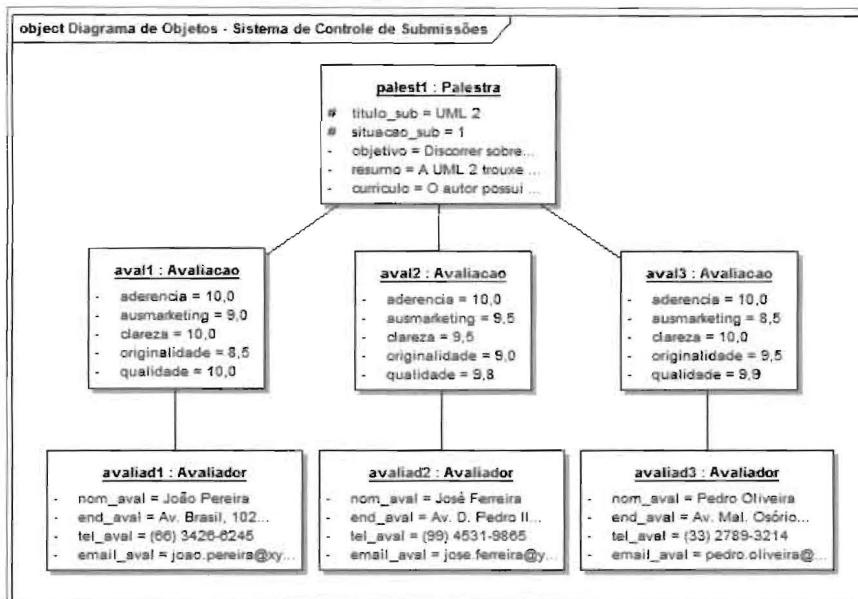


Figura 5.2 – Vínculo entre Objetos.

CAPÍTULO 6

Diagrama de Estrutura Composta

Este é um dos três novos diagramas propostos na UML 2. O diagrama de estrutura composta é utilizado para modelar colaborações. Uma colaboração descreve uma visão de um conjunto de entidades cooperativas interpretadas por instâncias que cooperam entre si para executar uma função específica. O termo estrutura desse diagrama refere-se a uma composição de elementos interconectados, representando instâncias de tempo de execução colaborando por meio de vínculos de comunicação para atingir algum objetivo comum. Esse diagrama também pode ser utilizado para descrever a estrutura interna de um classificador.

O diagrama de estrutura composta é semelhante ao diagrama de classes, porém este último apresenta uma visão estática da estrutura de classes, ao passo que o primeiro tenta expressar arquiteturas de tempo de execução, padrões de uso e os relacionamentos dos elementos participantes, o que nem sempre pode ser representado por diagramas estáticos.

6.1 Colaborações

Uma colaboração é representada como um tipo de classificador e define um conjunto de entidades cooperativas que serão interpretadas por instâncias, que representarão o papel das entidades, bem como um conjunto de conectores que definem caminhos de comunicação entre as instâncias participantes. As entidades cooperativas são as propriedades da colaboração.

Colaborações são geralmente utilizadas para explicar como um conjunto de instâncias cooperando entre si realiza uma tarefa conjunta ou um conjunto de tarefas. O propósito primário de uma colaboração é explicar como um sistema funciona, portanto, apresenta somente os aspectos extremamente necessários à explicação em questão, suprimindo quaisquer outros detalhes. Um mesmo objeto pode estar interpretando simultaneamente diversos papéis em diferentes colaborações, mas cada colaboração deverá representar somente os aspectos do objeto relevantes ao seu propósito. Uma colaboração é representada por uma elipse tracejada contendo uma descrição, conforme pode ser observado na Figura 6.1.

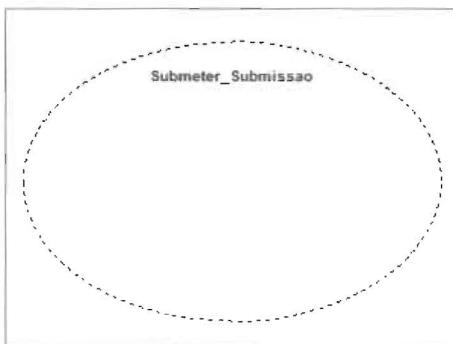


Figura 6.1 – Exemplo de Colaboração.

Os “papéis” de uma colaboração são interpretados por instâncias que cooperam entre si para concluir uma tarefa. Os relacionamentos entre as instâncias considerados relevantes para a tarefa em questão são representados por meio da utilização de retas ligando uma instância a outra, chamadas conectores.

Os papéis de colaborações definem um uso de instâncias, ao passo que os classificadores que definem esses papéis especificam todas as propriedades requeridas dessas instâncias. Assim, uma colaboração específica quais propriedades uma instância deve ter habilitadas. Nem todas as características, nem todo o conteúdo das instâncias participantes e nem todas as ligações dessas instâncias são sempre requeridas em uma colaboração particular. Por este motivo, uma colaboração é muitas vezes descrita em termos de papéis definidos por interfaces, uma vez que estas são descrições de um conjunto de propriedades (características observáveis

externamente) fornecidas ou requeridas por uma instância. A Figura 6.2 demonstra um exemplo de colaboração com sua estrutura interna, ou seja, papéis e conectores.

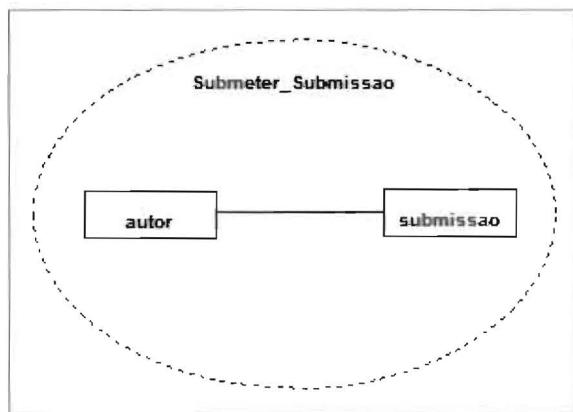


Figura 6.2 – Colaboração Contendo Papéis e Conectores

No exemplo da Figura 6.2 um autor colabora com uma submissão para submetê-la a avaliação em um congresso. Uma colaboração pode conter outras colaborações dentro de si, conforme é demonstrado pela Figura 6.3.

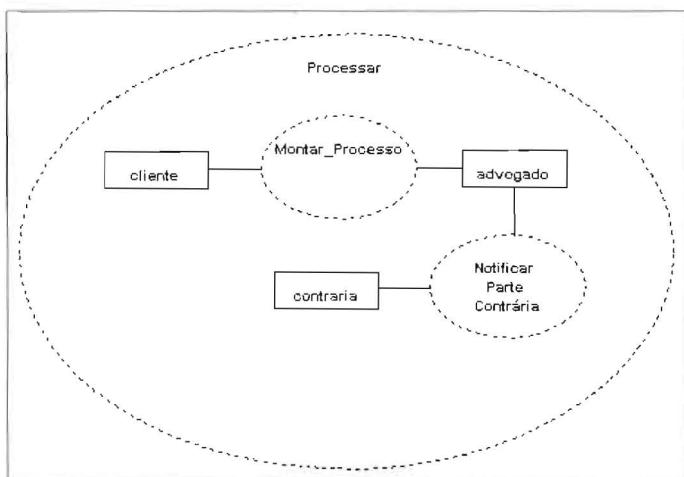


Figura 6.3 – Colaboração Contendo Colaborações.

Nesse exemplo, a colaboração Processar contém duas colaborações internas, as colaborações Montar_Processo e Notificar_Parte_Contrária. Na primeira, um cliente colabora com um advogado para montar um processo, e, na segunda, um advogado interage com uma parte contrária para notificá-la do processo.

6.2 Ocorrência de Colaboração

Uma ocorrência de colaboração representa a aplicação do padrão descrito por uma colaboração a uma situação específica envolvendo classes ou instâncias executando papéis específicos da colaboração. A denominação de uma ocorrência de colaboração possui a mesma notação utilizada na denominação de um objeto (uma vez que uma ocorrência de colaboração é uma instância de uma colaboração), ou seja, o nome da ocorrência seguida de dois pontos (:) e o nome da colaboração, ou caso não se queira dar um nome para a ocorrência, simplesmente dois pontos (:) e o nome da colaboração. Uma ocorrência de colaboração pode ser relacionada a uma classe por meio de uma seta tracejada contendo o estereótipo <<represents>>, conforme demonstra a Figura 6.4.

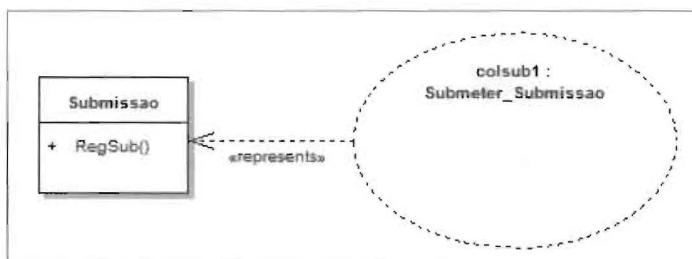


Figura 6.4 – Ocorrência de Colaboração Relacionada a uma Classe.

Aqui modificamos um pouco a classe Submissão, inserindo nela o método RegSub, que na realidade foi colocado em suas subclasses derivadas, para facilitar a exemplificação da ocorrência de colaboração. Assim, esse exemplo determina que a ocorrência de colaboração está relacionada a um classificador, neste caso a classe Submissão.

Pode-se representar uma ocorrência de colaboração a partir de uma colaboração por meio da mesma seta tracejada contendo o estereótipo <<occurrence>>, conforme demonstra a Figura 6.5.

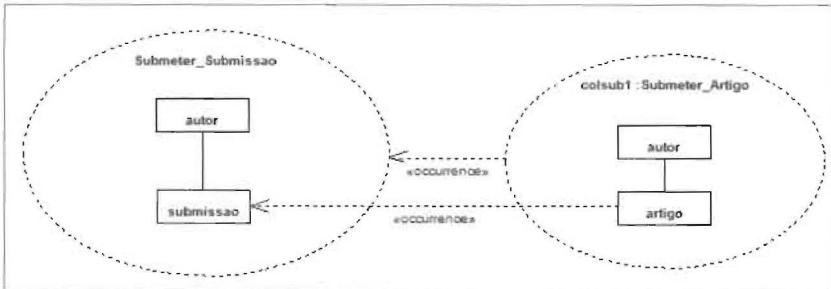


Figura 6.5 – Ocorrência de Colaboração.

Nesse exemplo percebemos que Submeter_Artigo é uma ocorrência de colaboração da colaboração Submeter_Submissão e que a instância artigo da ocorrência colsub1 interpreta o papel de submissão do elemento da colaboração Submeter_Submissão.

6.3 Portas

Portas são características estruturais de um classificador que representam pontos de interação distintos entre um classificador e seu ambiente ou entre o classificador e suas partes internas. Portas são conectadas às propriedades de um classificador por meio dos quais requisições podem ser feitas para invocar as características comportamentais do classificador. Uma porta é utilizada para representar os serviços que um classificador fornece ao seu ambiente ou os serviços que o classificador requer dele. Portas são representadas por quadrados sobrepostos à borda de um classificador ou de suas partes internas. Uma porta pode possuir um nome ou não. A Figura 6.6 apresenta um exemplo de porta.

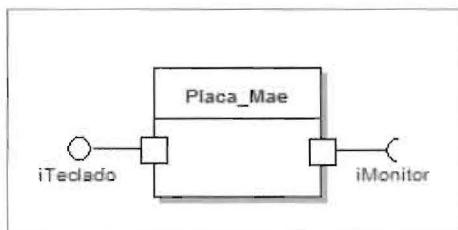


Figura 6.6 – Exemplo de Portas.

Nesse exemplo, a classe Placa-Mãe possui duas portas, uma para cada interface com seu ambiente externo. A classe está totalmente encapsulada, ou seja, não possuímos nenhum conhecimento a respeito de sua estrutura interna.

6.4 Propriedades

Uma propriedade representa um conjunto de instâncias internas que possuídas por uma instância de um classificador container. Quando uma instância de um classificador é criada, um conjunto de instâncias correspondente às suas propriedades podem ser criadas. Uma propriedade especifica que um conjunto de instâncias pode existir. Este conjunto é um subconjunto do conjunto total de instâncias especificado pelo classificador que define a propriedade.

Uma parte declara que uma instância desse classificador pode conter um conjunto de instâncias por composição, ou seja, essas instâncias não podem se relacionar com outro objeto, pois pertencem exclusivamente à instância da classe container e serão destruídas quando a instância da classe container for destruída. A Figura 6.7 apresenta um exemplo de partes.

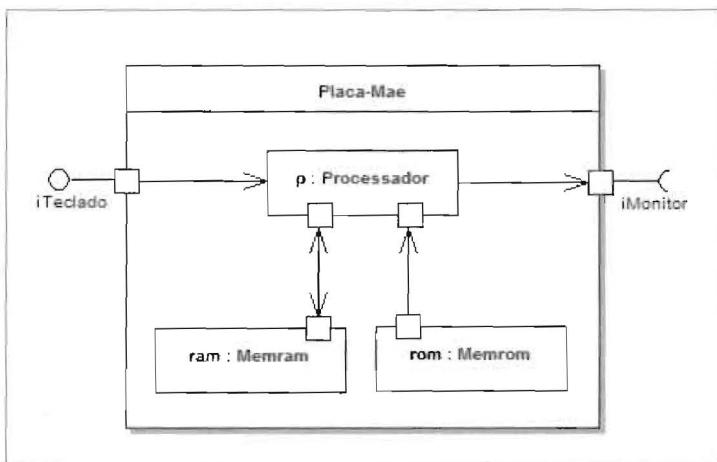


Figura 6.7 – Exemplo de Propriedade.

Nesse exemplo apresentamos a estrutura interna de uma placa-mãe. As retas que ligam as partes entre si e ao ambiente externo por meio das portas

são chamadas conectores e podem ser unidirecionais, se a seta aponta somente em uma direção, ou bidirecionais, se apontam para as duas direções, significando que podem receber ou enviar informações ou ambos.

Uma propriedade que especifique uma instância que não pertença por composição à instância do classificador container é apresentada como um retângulo tracejado e dita referenciada. Partes podem conter a definição de sua multiplicidade, podendo esta ser fixa, quando se sabe o número exato de instâncias contidas na instância do classificador container, ou determinando o número mínimo e máximo da multiplicidade, conforme demonstra o exemplo da Figura 6.8.

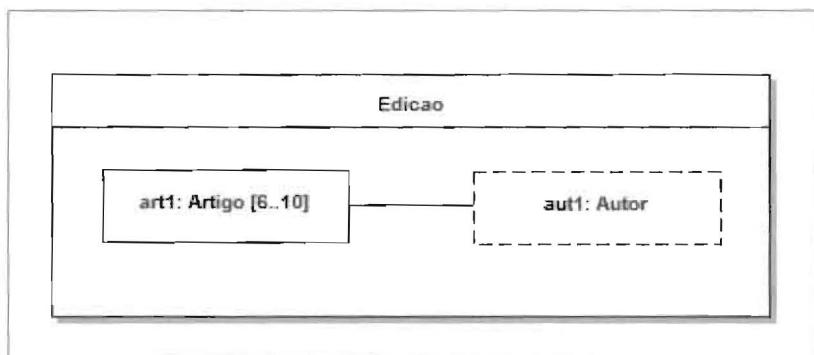


Figura 6.8 – Exemplo de Parte com Multiplicidade e Propriedade Referenciada.

Aqui tomamos o exemplo da Figura 4.5 do capítulo Diagrama de classes, em que uma edição se relaciona por composição a diversos artigos, significando que uma instância da classe Artigo se relaciona única e exclusivamente com uma instância da classe Edição. Como pode-se observar, a parte que representa os artigos da edição possui uma multiplicidade que determina que devem existir no mínimo seis instâncias da classe artigo dentro da instância da classe Edição e no máximo dez. Ao mesmo tempo, existe uma propriedade referenciada, como podemos observar pelo tracejado de seu retângulo, uma vez que instâncias da classe Autor devem estar associadas a um artigo mas não se relacionam à edição por composição.

6.5 Estudo de Caso

Nesta seção exemplificaremos uma colaboração relacionada ao sistema que estamos modelando. A colaboração enfoca o processo de realizar uma submissão por um autor, a qual foi parcialmente explicada neste tópico, mas que aqui é apresentada de uma forma um pouco mais completa.

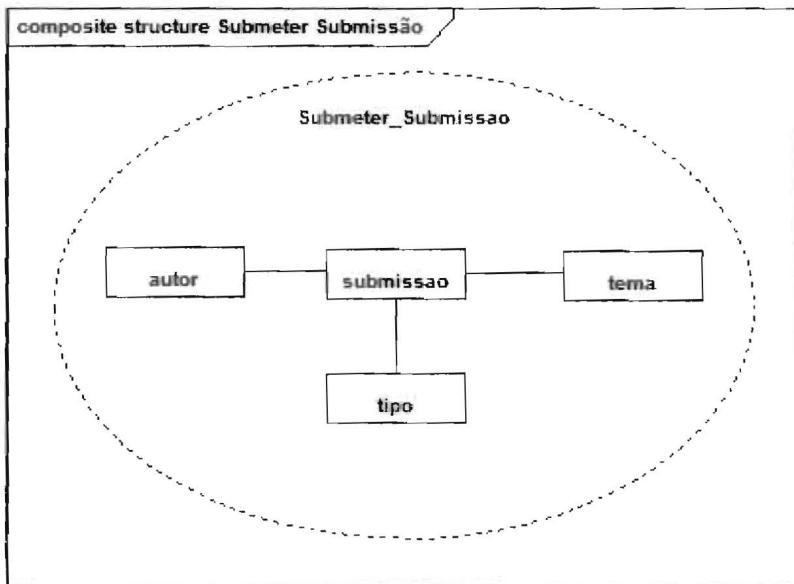


Figura 6.9 – Colaboração Referente ao Processo de Submissão.

CAPÍTULO 7

Diagrama de Seqüência

Este diagrama procura determinar a seqüência de eventos que ocorrem em um determinado processo, identificando quais métodos devem ser disparados entre os atores e objetos envolvidos e em que ordem. O diagrama de seqüência baseia-se no diagrama de casos de uso, havendo normalmente um diagrama de seqüência para cada caso de uso, uma vez que um caso de uso, em geral, refere-se a um processo disparado por um ator. Assim, um diagrama de seqüência também permite documentar um caso de uso.

Obviamente, o diagrama de seqüência depende também do diagrama de classes, já que as classes dos objetos declarados no diagrama estão descritas nele, bem como os métodos disparados entre os objetos. No entanto, o diagrama de seqüência é uma excelente forma de validar o diagrama de classes, pois ao modelá-lo muitas vezes percebem-se falhas e a necessidade de se declarar novos métodos que não haviam sido imaginados antes.

7.1 Atores

Os atores são os mesmos do diagrama de casos de uso e possuem a mesma representação diferenciando-se por possuírem uma linha de vida. A Figura 7.1 ilustra um exemplo de ator.

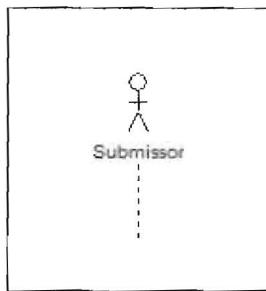


Figura 7.1 – Exemplo de Ator.

7.2 Objetos

Objetos representam as instâncias das classes envolvidas no processo ilustrado pelo diagrama de seqüência. Os objetos do diagrama de seqüência possuem a mesma notação utilizada no diagrama de objetos, diferenciando-se por possuírem uma linha de vida, representada por uma linha vertical tracejada abaixo do objeto. A Figura 7.2 apresenta um exemplo de objeto no diagrama de seqüência.

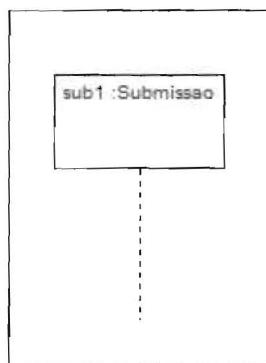


Figura 7.2 – Exemplo de Objeto.

Um objeto pode existir desde o início do processo ou ser criado durante a execução do mesmo. No primeiro caso, o objeto aparecerá na parte superior do diagrama; já no segundo, o objeto surgirá na mesma altura em que a mensagem que o criar for chamada. A Figura 7.3 apresenta um exemplo contendo objetos ativos desde o início do processo e objetos criados no decorrer do mesmo.

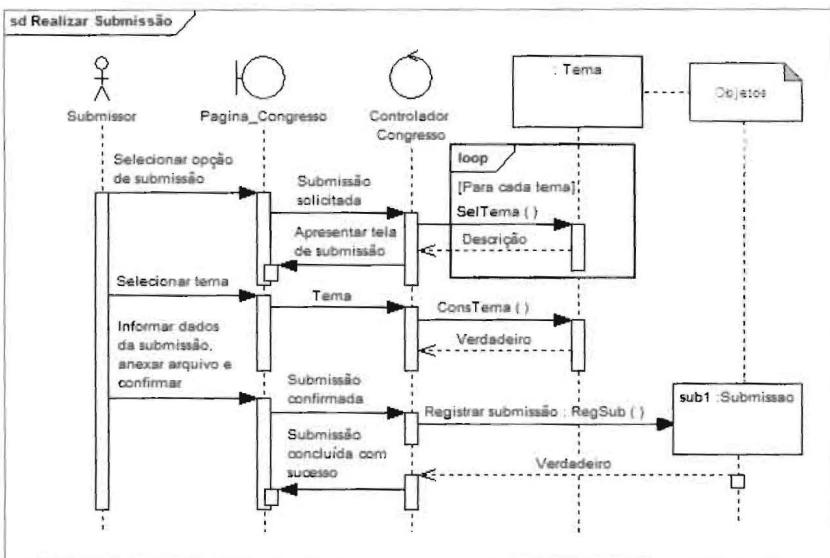


Figura 7.3 – Objetos em um Diagrama de Seqüência.

Na Figura 7.3 utilizamos o componente notas para destacar a existência de dois objetos no diagrama. Ao estudarmos a figura, verificamos que o objeto da classe Tema esteve ativo desde o início do processo, no entanto, o objeto sub1 da classe Submissão foi instanciado no decorrer do processo. O objeto da classe Tema não possui uma descrição, porque nesse caso o processo pode se referir a muitos objetos dessa classe. É importante observar também que existem dois outros objetos no diagrama, pertencentes às classes Página_Congresso e Controlador_Congresso, cujos símbolos foram modificados por possuírem estereótipos <<boundary>> e <<control>>. O exemplo aqui apresentado se refere ao processo de realizar submissão do sistema que temos estado modelando ao longo do guia.

7.3 Linha de Vida

Representa o tempo em que um objeto existe durante um processo. As linhas de vida são representadas por linhas finas verticais tracejadas partindo do objeto. A linha de vida é interrompida com um "X" quando o objeto é destruído.

7.4 Foco de Controle ou Ativação

Indica os períodos em que um objeto está participando ativamente do processo. Os focos de controle são representados dentro da linha de vida de um objeto, porém as linhas de vida são representadas por tracejados finos, ao passo que o foco de controle é representado por uma linha mais grossa.

A Figura 7.4 apresenta um exemplo de linha de vida e foco de controle.

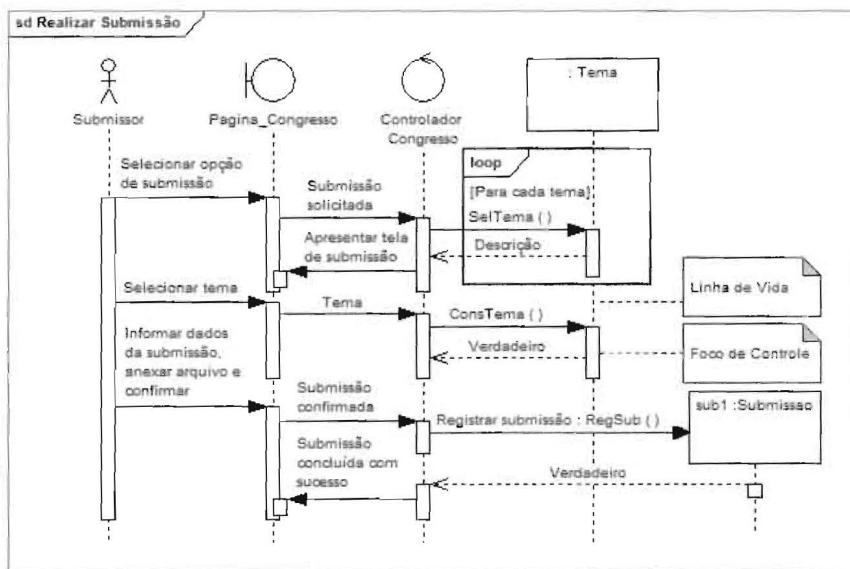


Figura 7.4 – Exemplo de Linha de Vida e Foco de Controle.

Ao observarmos a Figura 7.4, podemos perceber, por meio da linha de vida, que o objeto da classe Tema esteve presente durante todo o processo, no entanto, ele só teve participação ativa quando do disparo dos métodos de seleção e consulta de tema, quando a linha de vida se tornou mais grossa, indicando que o foco de controle do processo estava sobre o objeto da classe Tema.

7.5 Mensagens ou Estímulos

As mensagens são utilizadas para demonstrar a ocorrência de eventos, que normalmente forçam a chamada de um método em algum dos objetos envolvidos no processo. Pode ocorrer, no entanto, de uma mensagem

representar a comunicação entre dois atores, neste caso, não disparando métodos. Um diagrama de seqüência em geral é iniciado por um evento externo, causado por algum ator, o que acarreta o disparo de um método em um dos objetos. As mensagens podem ser disparadas entre:

- Um ator e outro ator.
- Um ator e um objeto, em que um ator produz um evento que dispara um método em um objeto.
- Um objeto e um objeto, o que constitui a ocorrência mais comum de mensagens, em que um objeto transmite uma mensagem para outro objeto em geral solicitando a execução de um método. Um objeto pode inclusive enviar uma mensagem para si mesmo, disparando um método em si próprio, o que é conhecido como auto-chamada.
- Um objeto e um ator; em geral, isso ocorre somente quando um objeto envia uma mensagem de retorno em resposta à chamada de um método solicitado, contendo seus resultados.

As mensagens são representadas por uma seta entre dois componentes, indicando qual componente enviou a mensagem e qual a recebeu. As mensagens são apresentadas na posição horizontal entre as linhas de vida dos componentes e sua ordem seqüencial é demonstrada de cima para baixo.

Os textos contidos nas mensagens primeiramente identificam qual evento ocorreu e forçou o envio da mensagem e qual método foi chamado. As duas informações são separadas por um símbolo de dois pontos (:). Podem ocorrer eventos que não disparam métodos; neste caso, a mensagem descreve apenas o evento que ocorreu, sem o símbolo de dois pontos e nenhum texto após os mesmos. Também pode acontecer de somente o método chamado ser descrito, sem detalhar qual evento o causou. A Figura 7.5 apresenta um exemplo de mensagem.

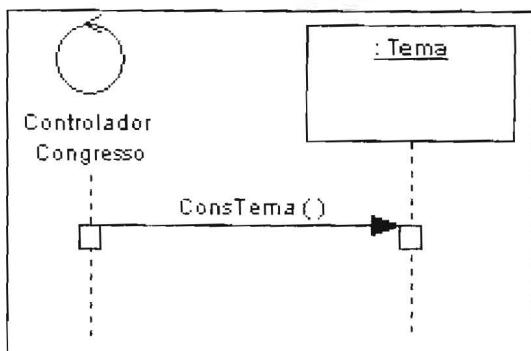


Figura 7.5 – Mensagem no Diagrama de Seqüência.

Quando a mensagem é dirigida a um objeto que já existe, a seta da mensagem atinge a linha de vida do objeto, engrossando-a, indicando que o foco de controle está sobre o objeto em questão. Quando a mensagem cria um novo objeto, no entanto, a seta atinge o retângulo do objeto, indicando que a mensagem representa um método construtor e que o objeto passa a existir somente a partir daquele momento. A Figura 7.6 apresenta um exemplo de mensagem que causa a criação de um novo objeto.

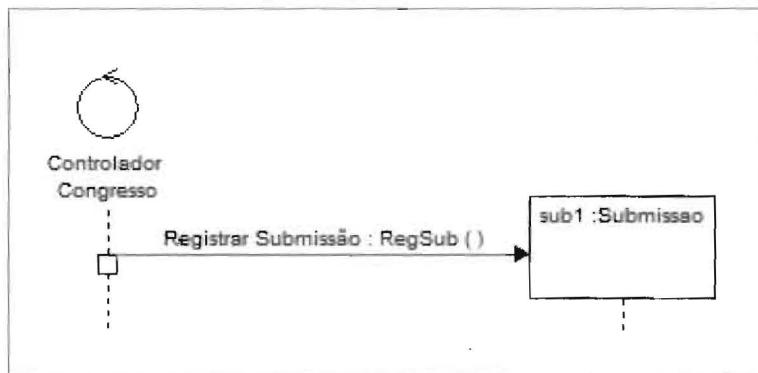


Figura 7.6 – Mensagem que instancia um novo objeto.

A Figura 7.6 representa a criação de um novo objeto da classe Submissão, causada pela confirmação de submissão por um submissor de um trabalho a ser analisado em um congresso.

Uma mensagem pode também representar um método destrutor, ou seja, um método que elimina um objeto não mais necessário. Nesse caso,

a mensagem atinge a linha de vida de um objeto e a interrompe com um X. A Figura 7.7 apresenta um exemplo de chamada de método destrutor.

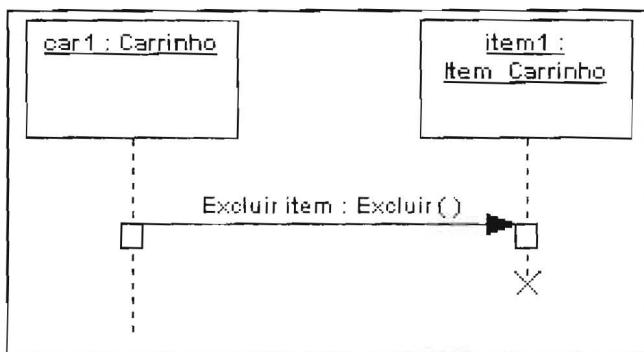


Figura 7.7 – Mensagem que dispara um método destrutor.

Nesse exemplo, existe um objeto car1 pertencente a uma classe Carrinho, que representa um carrinho de compras, como os encontrados nas livrarias digitais da Internet e que pode possuir muitos itens, representados pelos objetos da classe Item_Carrinho. Se em algum momento o cliente resolver cancelar a compra de algum dos itens do carrinho, o objeto da classe Carrinho deverá disparar um método destrutor no objeto da classe Item_Carrinho, aqui representado pelo método Excluir.

7.6 Mensagens de retorno

Este tipo de mensagem identifica a resposta a uma mensagem para o objeto que a chamou. Uma mensagem de retorno pode retornar informações específicas do método chamado ou apenas um valor indicando se o método foi executado com sucesso ou não. As mensagens de retorno são representadas por uma seta tracejada contendo uma seta fina que aponta para o objeto ou ator que recebe o resultado do método chamado. A Figura 7.8 apresenta um exemplo de mensagem de retorno.

Nesse exemplo, o método RegSub disparado sobre o objeto sub1 retorna um valor considerado verdadeiro, significando que o método foi concluído com sucesso. Obviamente um método dificilmente retornaria uma string com o texto “Verdadeiro”, mas um valor numérico cuja interpretação significaria verdadeiro ou falso. Alguns autores só modelam as

mensagens de retorno consideradas realmente importantes para evitar deixar o diagrama muito poluído.

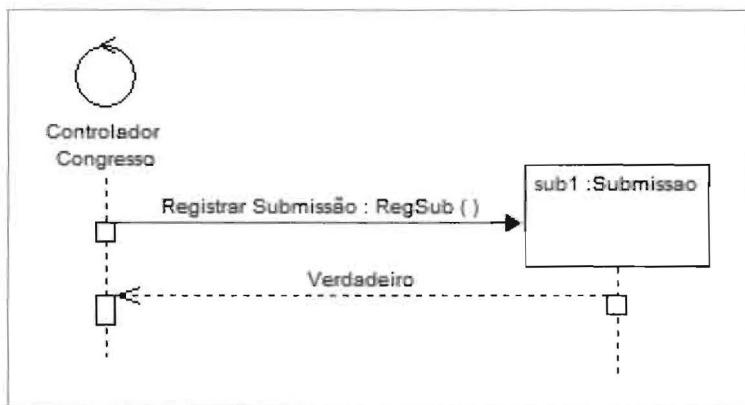


Figura 7.8 – Mensagem de Retorno.

7.7 Auto-chamadas ou Auto-delegações

São mensagens que partem da linha de vida objeto e atingem a linha de vida do próprio objeto. A Figura 7.9 demonstra um exemplo de auto-chamada.

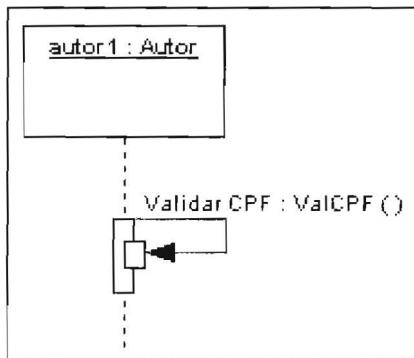


Figura 7.9 – Auto-chamada.

Neste exemplo, o objeto autor1 dispara em si mesmo um método para validação de CPF.

7.8 Condições ou Condições de Guarda

Estabelecem uma regra ou condição para que uma mensagem possa ser disparada. As condições são descritas entre colchetes na mensagem e serão exemplificadas na próxima seção.

7.9 Fragmentos de Interação e Ocorrências de Interação

Os fragmentos de interação são noções abstratas de unidades de interação geral. Um fragmento de interação é uma parte de uma interação, no entanto, cada fragmento de interação é considerado como uma interação independente. Um fragmento é representado como um retângulo que envolve toda a interação, além de conter uma aba no canto superior esquerdo, contendo um operador que determina a qual tipo de diagrama de interação ele se refere. O texto seguinte ao operador sd, por exemplo, indica que o fragmento é um diagrama de seqüência. O texto seguinte ao operador contém a descrição da interação que está sendo modelada, normalmente contendo apenas o nome da interação. A Figura 7.10 apresenta um exemplo de fragmento de interação.

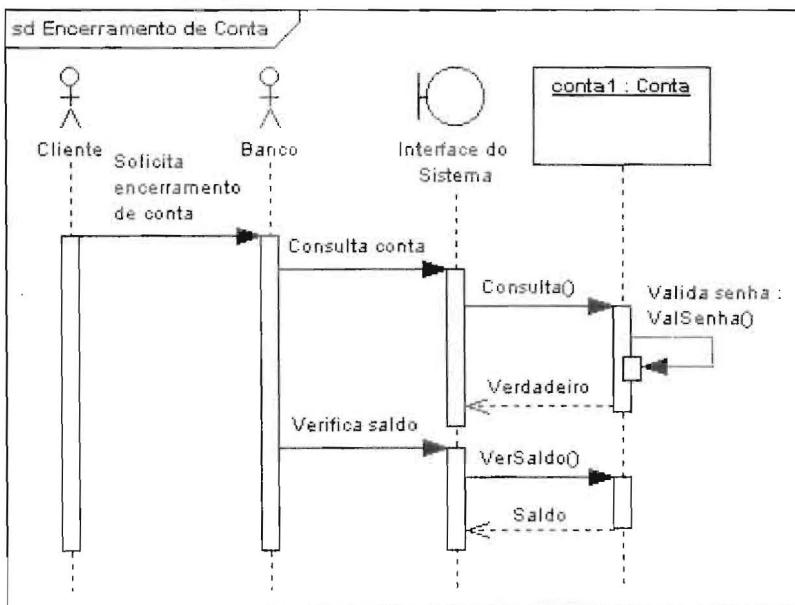


Figura 7.10 – Exemplo de Fragmento de Interação.

Essa figura representa o início do processo de encerramento de uma conta bancária especial, em que o cliente solicita ao funcionário do banco (aqui representado pelo ator Banco) que sua conta seja encerrada, informando sua conta e senha. Caso estas sejam válidas, o funcionário disparará o método Saldo para obter o saldo da conta. Esse exemplo está incompleto, posto que seu objetivo é, neste caso, apenas ilustrativo; esse processo será continuado nas seções seguintes, nas quais será abordado o conceito de fragmentos combinados.

Uma das principais vantagens do uso de fragmentos de interação caracteriza-se pela possibilidade de se poder referenciá-los por meio do operador Ref, que é a abreviatura de Referred (referido) e significa que deve-se procurar por um diagrama cujo nome é o mesmo do nome apresentado após o operador Ref, ou seja, o fragmento faz referência a outro diagrama, não detalhado no diagrama em questão e que deve ser inserido neste. A isso se chama ocorrência de interação, e esta inovação permite que se montem diagramas mais complexos que fazem referência a outros diagramas como se fossem sub-rotinas, detalhadas em separado. A Figura 7.11 apresenta um exemplo do uso de ocorrências de interação em um fragmento de interação.

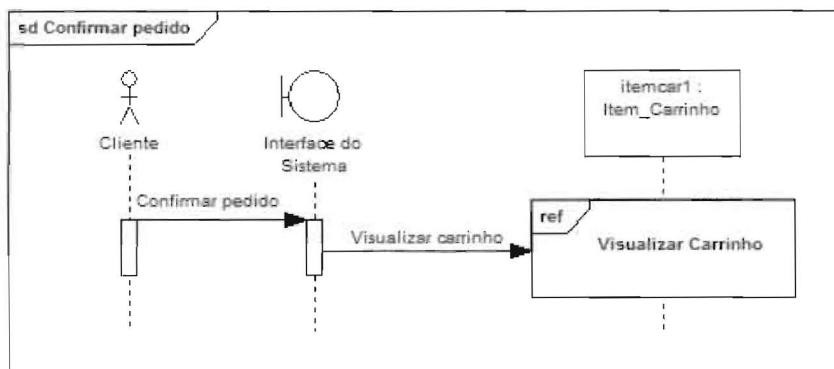


Figura 7.11 – Exemplo de Ocorrência de Interação.

Nesse exemplo, enfocamos o processo de confirmação de pedidos pertencente a um sistema de vendas pela Internet, em que o cliente pode consultar o carrinho de compras a qualquer momento, no entanto, caso ele resolva confirmar o pedido, o sistema obrigatoriamente deverá chamar

a rotina de Visualização de Carrinho. Assim, como o processo de Visualização de Carrinho pode ser chamado de várias partes do sistema, é mais prático modelá-lo em separado e referenciá-lo sempre que for necessário, como é demonstrado nesse caso.

É possível encontrar ocorrências de interação simplesmente sobrepostas às linhas de vida dos objetos que fazem parte do processo, sem nem ao menos chamá-las por meio de uma mensagem, como se as instruções contidas nas ocorrências de interação fossem adicionadas automaticamente ao diagrama, como pode ser visto na Figura 7.12.

Como pode-se observar, a Figura 7.12 apresenta o mesmo exemplo da figura anterior; nesse caso, porém, a ocorrência de interação está simplesmente sobre a linha de vida dos objetos envolvidos. As ocorrências de interação podem se constituir em uma simples chamada a outro fragmento de interação ou podem passar parâmetros para o mesmo, receber o retorno da chamada deste ou ambos.

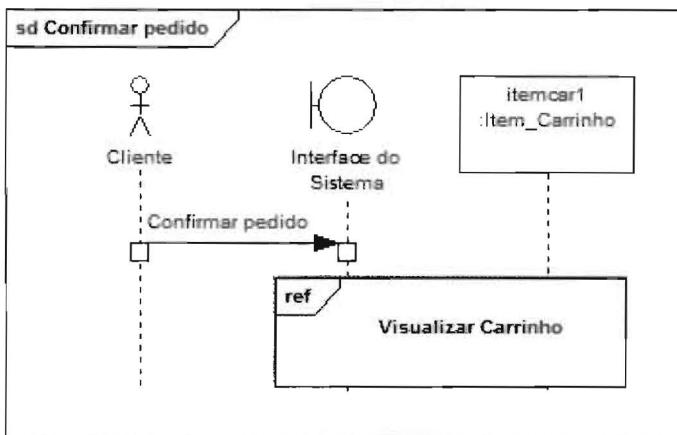


Figura 7.12 – Exemplo de Ocorrência de Interação.

7.9.1 Portões (Gates)

Um portão é uma interface entre fragmentos, um ponto de conexão para relacionar uma mensagem fora de uma ocorrência de interação com uma mensagem dentro da ocorrência de interação. Portões são representados simplesmente pelo encontro da seta da mensagem no retângulo da ocorrência.

rênciam de interação ou, em algumas ferramentas CASE, por pequenos quadrados posicionados na linha vertical do retângulo do fragmento, na altura em que a mensagem deverá atingir a linha de vida do objeto a que ela se refere. Quando se tratar de uma ocorrência de interação referenciada, na qual não é possível determinar em que local o objeto está, o portão é colocado no centro da linha vertical. O propósito de um portão é simplesmente estabelecer quem está enviando e quem está recebendo uma mensagem, quando esta mensagem não está contida em um único fragmento de interação. A Figura 7.13 apresenta um exemplo de uso de um portão.

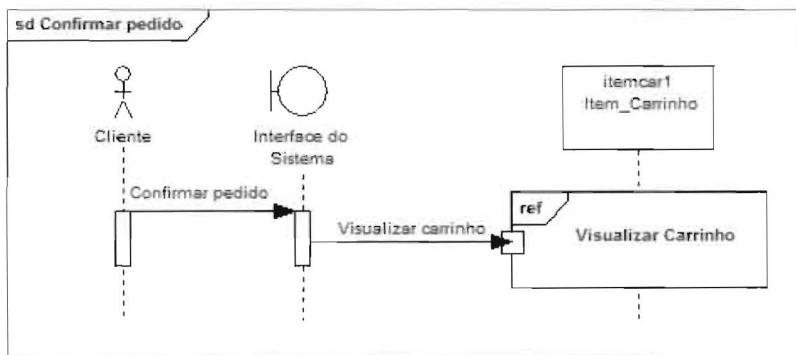


Figura 7.13 – Exemplo de Portão.

7.9.2 Fragmentos Combinados e Operadores de Interação

Nas versões anteriores à versão 2.0 da UML, os diagramas de seqüência tinham dificuldade em trabalhar questões como testes se-senão, laços ou processamentos paralelos. Essas questões foram abordadas na versão 2 por meio do uso de fragmentos combinados que permitem uma modelagem semi-independente da parte do diagrama em que se deve enfocar problemas como os enunciados.

Os fragmentos combinados são representados por um retângulo que determina a área de abrangência do fragmento no diagrama, além de conterem ainda uma subdivisão em sua extremidade superior esquerda, para identificar a descrição do fragmento combinado e seu operador de interação, que define o tipo de fragmento que está sendo modelado. Alguns dos operadores de interação mais comuns são listados e exemplificados a seguir:

- **Alt – Abreviatura de Alternativas (Alternativas).** Este operador de interação define que o fragmento combinado representa uma escolha entre dois ou mais comportamentos. Esse tipo de fragmento combinado costuma utilizar condições de guarda, também conhecidas como restrições de interação, para definir o teste a ser considerado na escolha de um dos comportamentos. A Figura 7.14 apresenta um exemplo de fragmento combinado com o operador Alt.

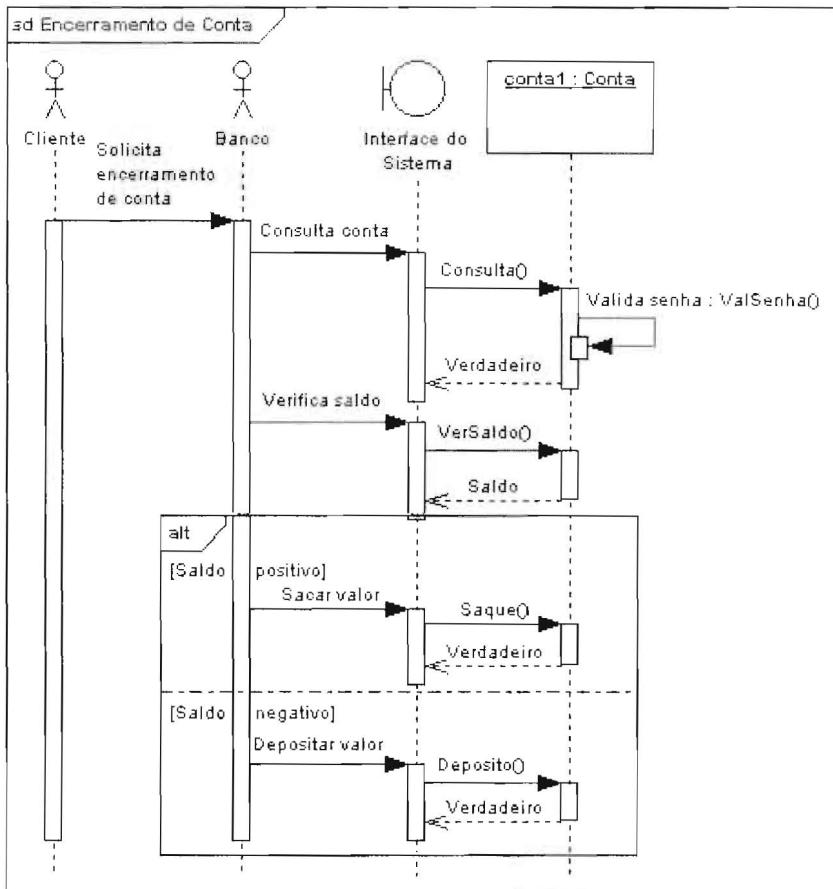


Figura 7.14 – Exemplo de Fragmento Combinado com Operador Alt.

Aqui damos continuidade ao processo de encerramento de conta especial, em que, após verificar o saldo da conta, deverá ser feita uma escolha entre duas operações: se o saldo da conta for positivo, então ele executará um saque e entregará ao cliente o valor depositado; se o saldo estiver negativo, o cliente deverá depositar o valor necessário para cobrir o saldo negativo da conta antes de encerrá-la.

Observe que fragmentos combinados que utilizam o operador de interação Alt possuem ao menos uma divisão, representada por uma linha tracejada, separando as ações executadas em cada opção. Cada uma dessas divisões é chamada separador de operando de interação, e o conteúdo representado em cada divisão é conhecido como operando de interação, ou seja, uma área de atuação de um fragmento combinado. Este possui ao menos um operando de interação, sendo que em alguns casos ele deve possuir ao menos dois, como neste exemplo, e em outros, não poderá ter mais do que um, quando não existem fluxos alternativos ou paralelos. Neste último caso, o operando de interação representa todo o conteúdo do fragmento combinado.

- **Opt – Abreviatura de Option (Opção).** Este operador de interação determina que o fragmento combinado representa uma escolha de comportamento em que este será ou não executado, não havendo uma escolha entre mais de um comportamento possível. A Figura 7.15 apresenta um exemplo de fragmento combinado utilizando o operador de interação Opt.

A Figura 7.15 dá continuidade ao exemplo da Figura 7.11, em que o cliente, após visualizar o carrinho de compras, deverá se logar, caso ainda não o tenha feito. Uma vez que o cliente pode se logar antes de confirmar o pedido (para verificar a situação de seus pedidos anteriores, por exemplo), é necessário testar se o cliente já se encontra logado. Por este motivo utilizamos um fragmento combinado com operador Opt, significando que os passos nele contidos serão ou não executados dependendo de sua condição, determinada pela restrição de interação “Ainda não logado”.

O leitor notará que o processo Logar é um processo referenciado, ou seja, uma ocorrência de interação, da mesma forma que Visualizar carrinho. Posto que o cliente tem a possibilidade de executar o processo Logar em mais de um local do sistema, é melhor modelá-lo em separado e referenciá-lo sempre que for necessário incluí-lo em um outro processo.

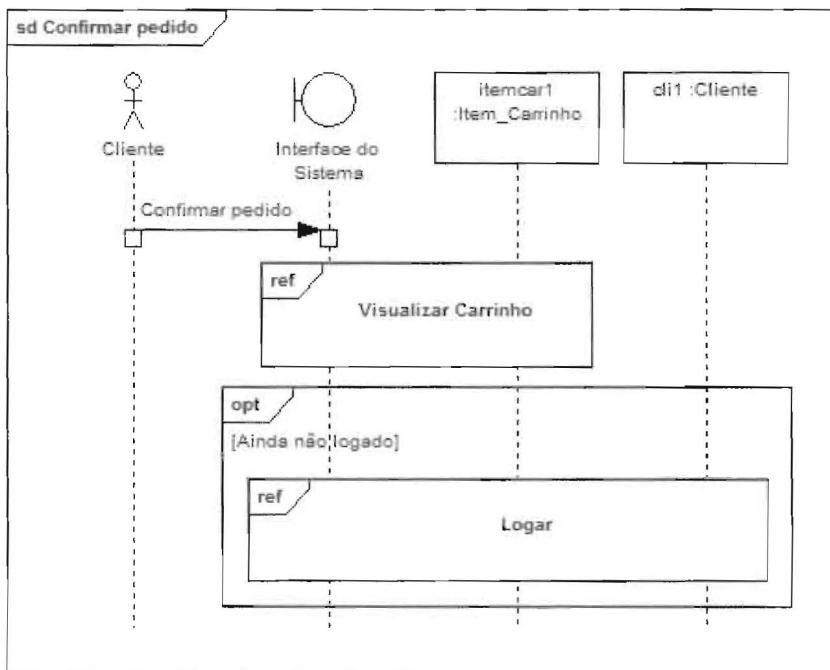


Figura 7.15 – Exemplo de Fragmento Combinado com Operador Opt.

- **Par – Abreviatura de Parallel (Paralelo).** Este operador de interação determina que o fragmento combinado representa uma execução paralela de dois ou mais comportamentos. A Figura 7.16 apresenta um exemplo fragmento combinado utilizando o operador de interação Par.

Identificamos aqui uma situação em que o ator Motorista deve realizar duas operações simultâneas sobre o objeto car1 da classe Carro, para poder dirigí-lo: soltar a embreagem e pressionar o acelerador. Note que uma linha tracejada divide os operandos de interação representando cada operação paralela.

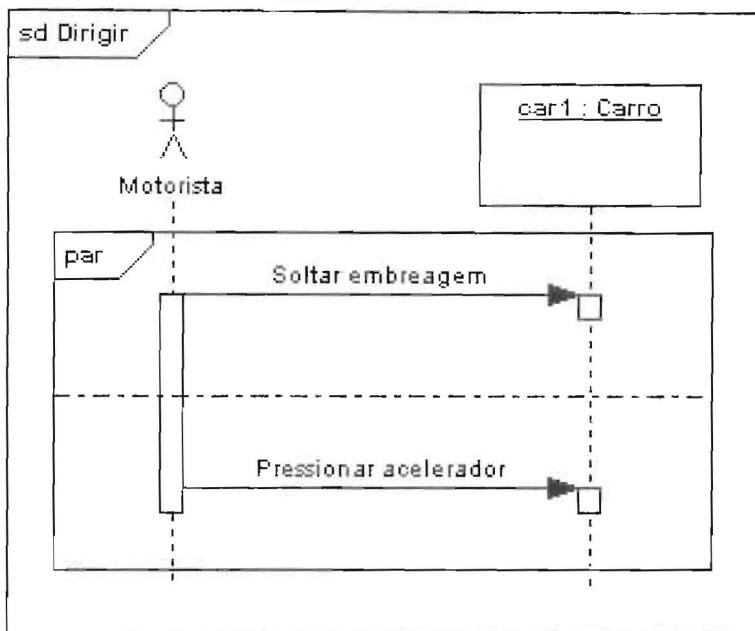


Figura 7.16 – Exemplo de Fragmento Combinado com Operador Par.

- **Loop – Abreviatura de Looping (Laço).** Este operador de interação determina que o fragmento combinado representa um laço que poderá ser repetido diversas vezes. A Figura 7.17 demonstra um exemplo de fragmento combinado utilizando o operador de interação Loop.

Nesse exemplo identificamos um processo utilizado para aumentar os produtos de uma categoria. Ao observarmos a figura, percebemos que primeiramente o funcionário seleciona uma categoria, informa o percentual de aumento e manda aumentar o valor de todos os produtos pertencentes à categoria selecionada. Percebemos ainda que o mesmo método é aplicado a cada produto pertencente à categoria, conforme demonstra a restrição de interação “[Para cada produto]”.

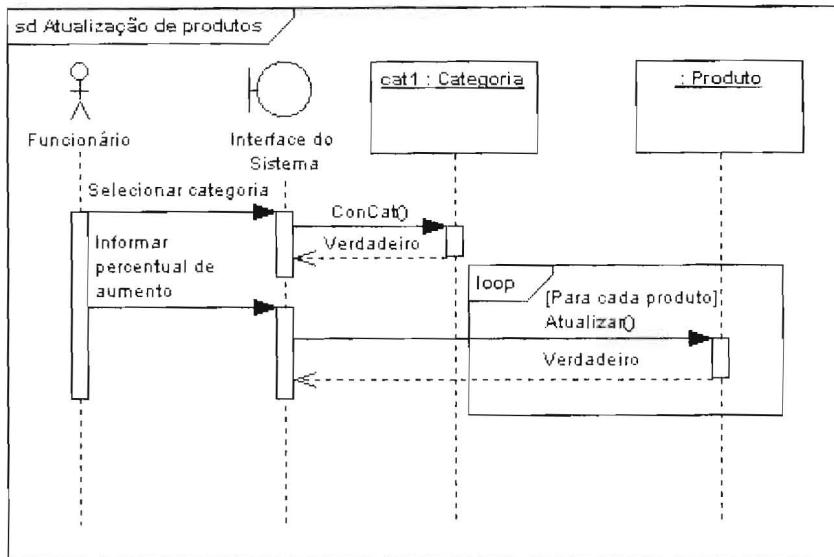


Figura 7.17 – Exemplo de Fragmento Combinado com Operador Loop.

- **Break (Quebra).** Este operador de interação indica uma “quebra” na execução normal do processo. É usado principalmente para modelar o tratamento de exceções. A Figura 7.18 apresenta um exemplo de fragmento combinado utilizando o operador de interação Break.

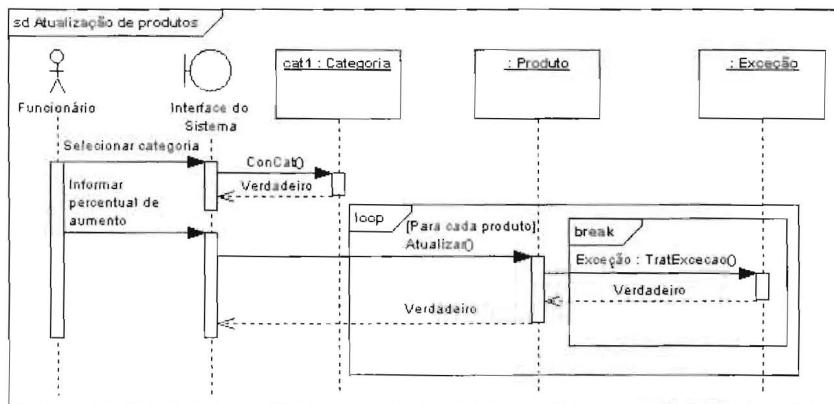


Figura 7.18 – Exemplo de Fragmento Combinado com Operador Break.

Aqui demos continuidade ao processo do exemplo anterior, no qual identificamos uma situação em que pode ocorrer uma exceção, em razão de algum registro estar corrompido ou algum campo possuir valores inválidos. Assim criamos um método TratExcecao (Tratamento de Exceção), pertencente à classe Exceção, para tratar possíveis exceções que venham ocorrer durante a atualização. Observe ainda que, como uma exceção interrompe o desenrolar normal do laço, além de não ocorrer normalmente, ela está contida em um fragmento combinado do tipo Break. Esse exemplo ilustra também a possibilidade de um fragmento combinado poder conter outro fragmento combinado.

- **Critical Region (Região Crítica).** Este operador de interação identifica uma operação atômica que não pode ser interrompida por outro processo até ser totalmente concluída. A Figura 7.19 apresenta um exemplo de fragmento combinado utilizando esse operador.

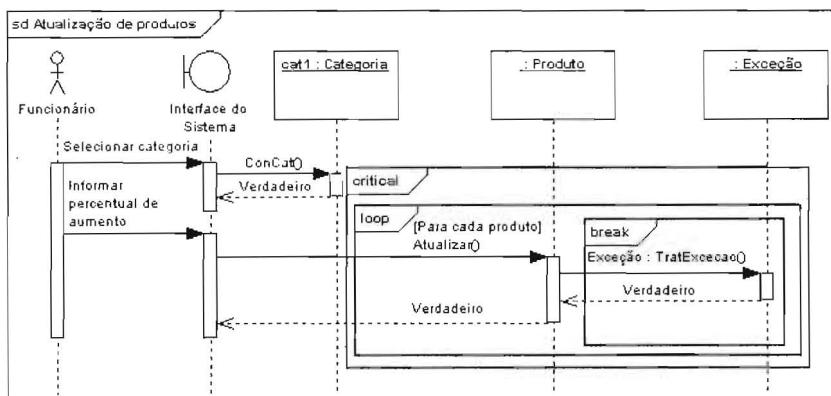


Figura 7.19 – Exemplo de Fragmento Combinado com Operador Critical Region.

Na Figura 7.19 melhoramos o exemplo utilizado para ilustrar os dois últimos operadores de interação, envolvendo o laço de atualização de produtos com um fragmento combinado do tipo Critical, indicando que a atualização dos produtos não deverá ser interrompida até que o processo seja totalmente concluído.

Existem ainda alguns operadores de interação menos utilizados, apresentados a seguir:

- **Neg – Abreviatura de Negative (Negativo)** – este operador de interação representa eventos considerados inválidos, que não devem ocorrer. Todos os eventos não-contidos em um fragmento combinado do tipo neg (quando existir um) são considerados positivos.
- **Assertion (Afirmação)** – este operador de interação é o oposto do anterior, representando eventos considerados como válidos. Todos os eventos não contidos em um fragmento combinado do tipo Assertion são automaticamente considerados negativos.
- **Ignore (Ignorar)** – o operador de interação Ignore determina que as mensagens contidas no fragmento devem ser ignoradas. Essas mensagens podem ser consideradas insignificantes e são intuitivamente ignoradas se elas aparecerem em uma execução correspondente. Alternativamente pode-se entender Ignore como significando que as mensagens que são ignoradas podem aparecer em qualquer lugar nos eventos.
- **Consider (Considerar)** – este operador de interação é o oposto do anterior e determina que as mensagens devem obrigatoriamente ser consideradas e que todas as outras mensagens não contidas no fragmento devem ser automaticamente desconsideradas. Tanto o operador de interação Ignore como Consider são freqüentemente utilizados juntamente com os operadores Neg e Assertion, de maneira que um fragmento pode conter o outro.
- **Seq – Abreviatura de Weak Sequencing (Seqüência Fraca)** – este operador de interação identifica uma situação em que as ocorrências de evento devem atender estas propriedades:
 1. As ordenações das ocorrências de evento dentro de cada um dos operandos são mantidas no resultado.
 2. Ocorrências de evento em linhas de vida diferentes de operandos diferentes podem vir em qualquer ordem.
 3. Ocorrências de evento na mesma linha de vida de operandos diferentes são ordenadas de tal forma que uma ocorrência de evento do primeiro operando venha antes do segundo operando.

- **Strict – Abreviatura de Strict Sequencing (Seqüência Estrita)** – o operador de interação strict apresenta um refinamento do operador Weak Sequencing e garante que todas as mensagens no fragmento combinado são ordenadas do início ao fim.

7.10 Estudo de Caso

Aqui iremos identificar os principais diagramas de seqüência referentes ao sistema que vem sendo modelado. O diagrama referente ao processo de Realizar Submissão não se encontra aqui, uma vez que já foi apresentado no início deste capítulo, nas figuras 7.3 e 7.4.

7.10.1 Realizar Submissão

Ao observarmos a Figura 7.3, percebemos que, no momento em que o Submissor seleciona a opção de submissão na página, esta repassa o evento ao controlador, que, por sua vez busca todos os temas considerados válidos por meio do método SelTema (Seleciona Temas), e manda apresentá-los na página.

Uma vez que os tipos de submissão considerados válidos são fixos (palestra, artigo ou minicurso), não foi considerado necessário buscá-los em sua classe, colocando-os direto na página.

Após visualizar a página de submissão, o Submissor escolhe o tema do trabalho que deseja submeter, e o controlador em resposta o deixa carregado.

Em seguida, o Submissor informa os dados de seu trabalho, incluindo a escolha do tipo desta submissão, anexa o arquivo (que poderá conter o artigo a ser submetido ou o resumo da palestra ou minicurso que ele deseja ministrar).

Após isso, caso o Submissor confirme, o controlador criará, por meio do método RegSub, um novo objeto da classe Submissão, ordenando em seguida que a interface apresente uma mensagem de confirmação.

7.10.2 Realizar Login Submissor

Neste processo, ao receber a solicitação de login, o controlador apresenta a tela respectiva. Caso o Submissor não esteja cadastrado, ele deverá escolher a opção Auto-Registrar, que se refere a outro processo detalhado em um diagrama à parte, como podemos perceber pelo fragmento combinado do tipo Opt (já que se trata de algo opcional, uma vez que o Submissor pode já estar cadastrado) e a ocorrência de interação que faz referência ao processo de Auto-Registrar.

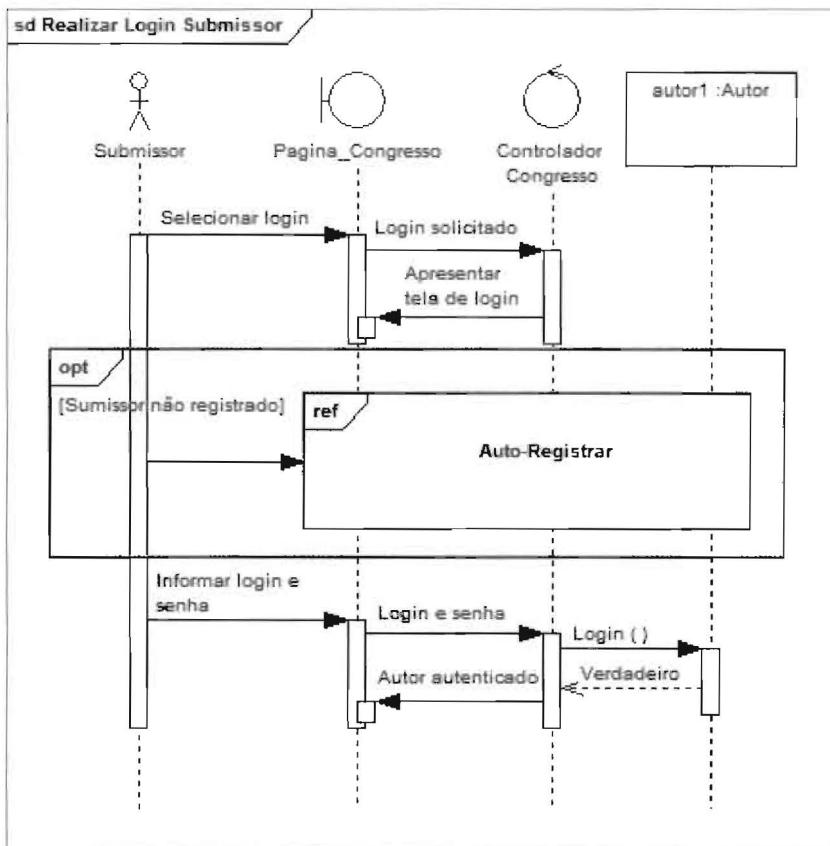


Figura 7.20 – Realizar Login Submissor.

Caso o Submissor já possua registro, então este informará seu nome login e senha, que serão validados pelo método login(), o qual autenticará o Submissor, permitindo que este utilize outras opções do sistema.

7.10.3 Verificar Submissões

Neste processo, a classe Controlador consulta todas as submissões por meio de um laço e as manda apresentar na página, detalhando sua situação, ou seja, se foram aprovadas, reprovadas ou ainda estão sob avaliação.

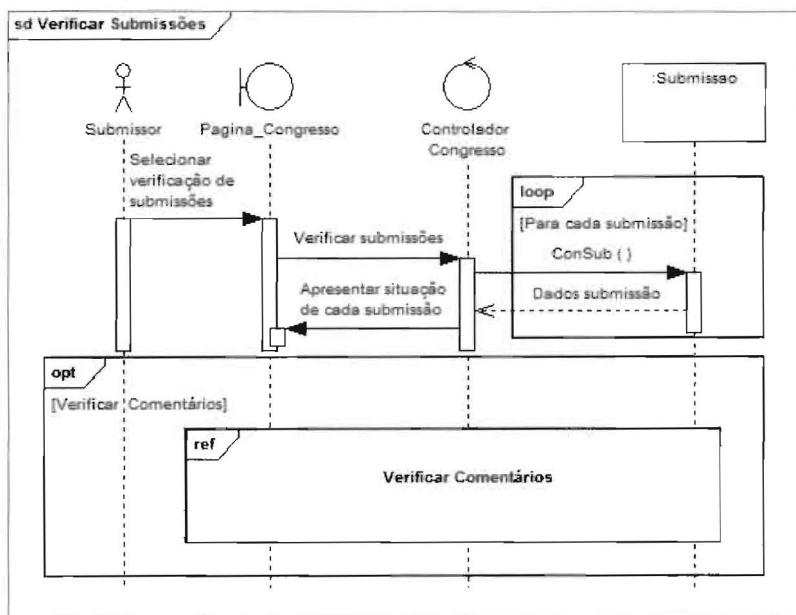


Figura 7.21 – Verificar Submissões.

A partir dessa listagem, o Submissor tem a opção de verificar os comentários de uma submissão específica, mas, como isso se trata de outro processo, esse está apenas referenciado, estando detalhado em um diagrama separado.

7.10.4 Verificar Comentários

Este processo é dependente do processo anterior, em que o Submissor seleciona uma submissão a partir da listagem apresentada e, em resposta, o controlador carrega a submissão selecionada.

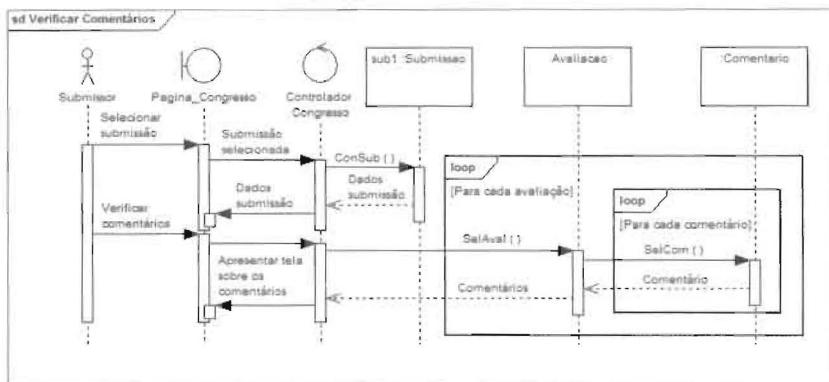


Figura 7.22 – Verificar Comentários.

Após isso, o Submissor escolhe a opção Verificar Comentários, o que faz que o controlador execute um laço para selecionar todas as avaliações da submissão em questão (cada submissão deve ter três avaliações realizadas por avaliadores diferentes).

A partir do primeiro laço é executado um segundo laço, em que serão selecionados todos os comentários de cada avaliação referente à submissão selecionada. Após o término da execução desses laços, o controlador apresenta uma listagem com todos os comentários encontrados.

7.10.5 Manter Avaliações

Este processo se refere à manutenção das avaliações sob responsabilidade de um Avaliador.

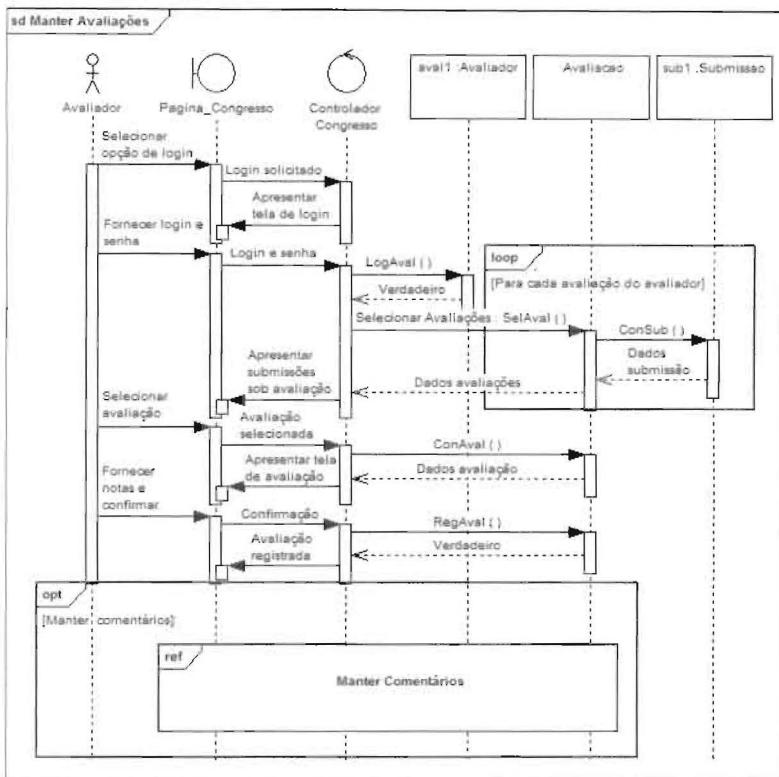


Figura 7.23 – Manter Avaliações.

Primeiramente o Avaliador loga-se no sistema e, em resposta, o controlador apresenta todas as avaliações, juntamente com as submissões às quais elas se referem, sob a responsabilidade do Avaliador.

Em seguida, o Avaliador, a partir da listagem apresentada, escolhe uma das avaliações, o que faz que o controlador carregue a avaliação em questão. Após isso, o Avaliador fornece as notas e confirma, fazendo que o controlador registre essas notas em um objeto da classe Avaliação.

Os objetos da classe Avaliação já haviam sido criados antes desse processo, quando da atribuição da avaliação de uma submissão a um avaliador.

Opcionalmente o Avaliador pode querer inserir comentários a respeito da avaliação, para esclarecer o porquê das notas ou sugerir modificações no trabalho, por exemplo.

Como isso é opcional, utilizamos um fragmento combinado do tipo Opt e, como o processo de inserir comentários é um processo diferente, embora dependente dele, apenas o referenciamos por meio de uma ocorrência de interação.

7.10.6 Manter Comentários

Este processo apresenta a manutenção dos comentários relativos a uma submissão. Quando o Avaliador seleciona esta opção, o controlador seleciona todos os comentários da avaliação em questão e os manda apresentar.

A partir daí o Avaliador pode inserir um novo comentário, alterar um já existente ou excluir um comentário.

Caso ele queira inserir um novo comentário, deve simplesmente digitar o comentário e confirmar, o que levará o controlador a criar um novo objeto da classe Comentário por meio do método RegCom.

Se o Avaliador, porém, quiser alterar ou excluir, primeiramente ele deve selecionar o comentário em questão para que o controlador possa carregá-lo. Após isso, o Avaliador pode alterar o comentário e confirmar, fazendo que o controlador registre as alterações por meio do mesmo método RegCom. Ele pode também escolher excluir o comentário o que fará que o controlador destrua o objeto pelo método ExcCom.

Observe que utilizamos um fragmento combinado do tipo Alt para separar cada uma das opções do Avaliador.

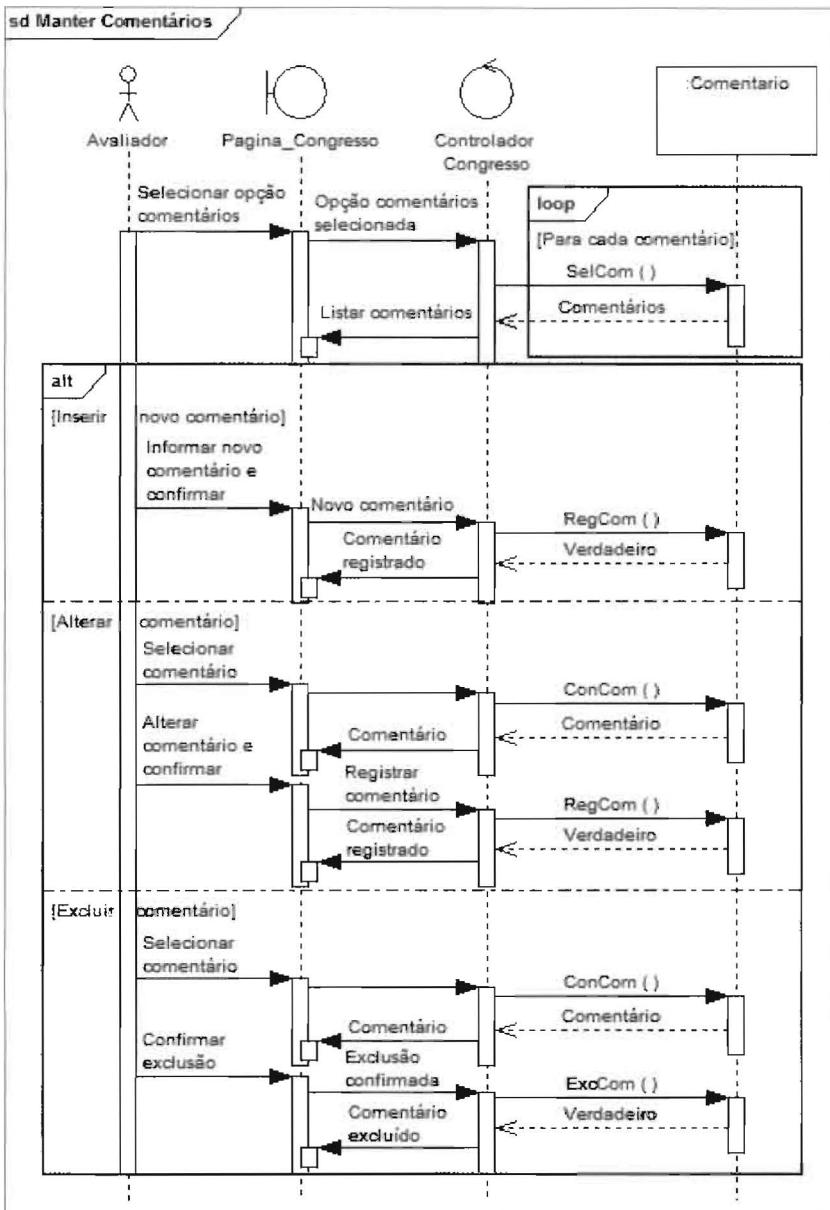


Figura 7.24 – Manter Comentários.

7.10.7 Relatório de Avaliações

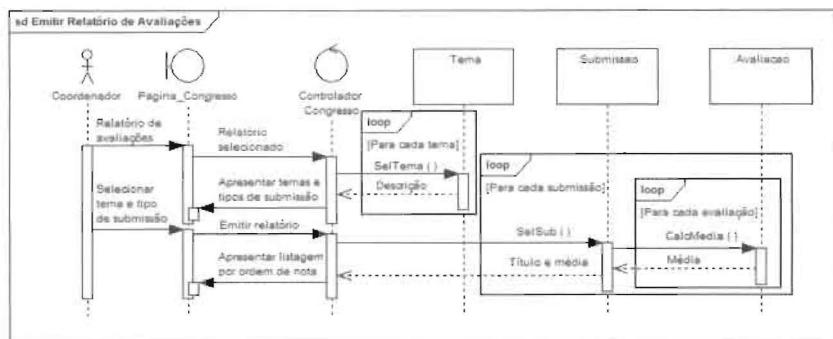


Figura 7.25 – Relatório de Avaliações.

Este diagrama representa o processo para gerar o relatório de avaliações, o qual só pode ser gerado pelo Coordenador do evento.

O objetivo dessa listagem é apresentar as submissões de um determinado tema e tipo por ordem da média geral das notas fornecidas pelos avaliadores, de forma que o Coordenador possa determinar quais serão aprovadas e quais não. Além de informar quais submissões ainda não foram avaliadas.

Dessa forma, ao saber que essa opção foi selecionada, o controlador seleciona e apresenta todos os temas aceitos pelo evento. O coordenador por sua vez seleciona o tema e o tipo de submissão que ele deseja visualizar e ordena que o relatório seja emitido.

O controlador em resposta seleciona todas as submissões referentes ao tema e tipo selecionados por meio de um laço. Dentro deste laço ocorre um segundo laço, por meio do qual é calculada a nota geral de cada submissão, somando as notas de cada uma das três avaliações e calculando a média dessas. Finalmente as submissões são apresentadas por ordem da maior média.

CAPÍTULO 8

Diagrama de Comunicação

O diagrama de comunicação era conhecido como diagrama de colaboração até a versão 1.5 da UML, tendo seu nome modificado para diagrama de comunicação a partir da versão 2.0. Esse diagrama está amplamente associado ao diagrama de seqüência – na verdade, um complementa o outro. As informações mostradas no diagrama de comunicação são, com freqüência, praticamente as mesmas apresentadas no diagrama de seqüência, porém com um enfoque diferente, visto que este diagrama não se preocupa com a temporalidade do processo, concentrando-se em como os objetos estão vinculados e quais mensagens trocam entre si durante o processo.

Por ser muito semelhante ao diagrama de seqüência, o diagrama de comunicação utiliza muitos de seus componentes, como atores e objetos, incluindo seus estereótipos de fronteira e controle. No entanto, os objetos no diagrama de comunicação não possuem linhas de vida. Além disso, esse diagrama não suporta ocorrências de interação ou fragmentos combinados como o diagrama de seqüência.

8.1 Mensagens

Como já foi dito, este diagrama se preocupa com o inter-relacionamento entre os objetos envolvidos em um processo, e isto é feito principalmente por meio de mensagens. Uma mensagem é causada por um evento e pode conter uma descrição, uma chamada de um método ou ambos. Mensagens podem ainda conter condições de guarda, bastante úteis neste diagrama.

Para que possa ser enviada uma mensagem de um componente é necessário haver uma associação entre os componentes, conforme demonstra a Figura 8.1.

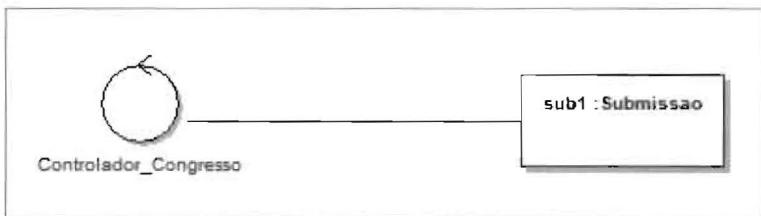


Figura 8.1 – Associação.

Após existir a associação pode-se então acrescentar mensagens a ela. Uma mensagem se caracteriza por conter uma seta apontando para o objeto para o qual está sendo enviada, conforme demonstra a Figura 8.2.

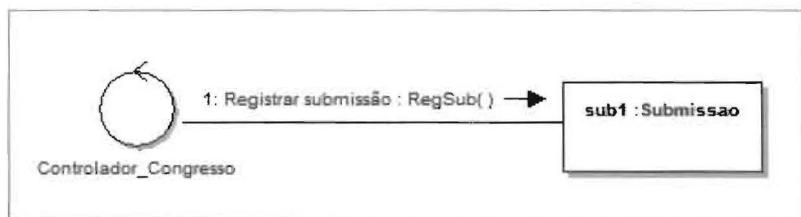


Figura 8.2 – Mensagem.

8.2 Estudo de Caso

Aqui daremos seguimento à modelagem do Sistema de Controle de Submissões. Os diagramas seguintes correspondem aos mesmos processos descritos no Capítulo 7, sobre o diagrama de seqüência e as mensagens e métodos são os mesmos, porém apresentados sob um enfoque diferente.

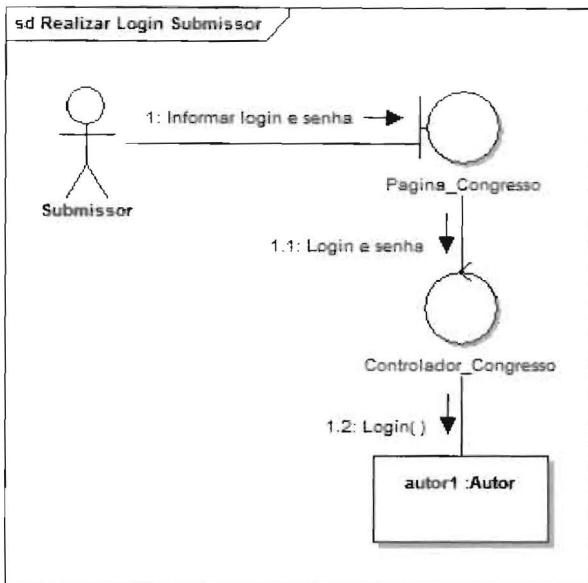


Figura 8.3 – Realizar Login Submissor.

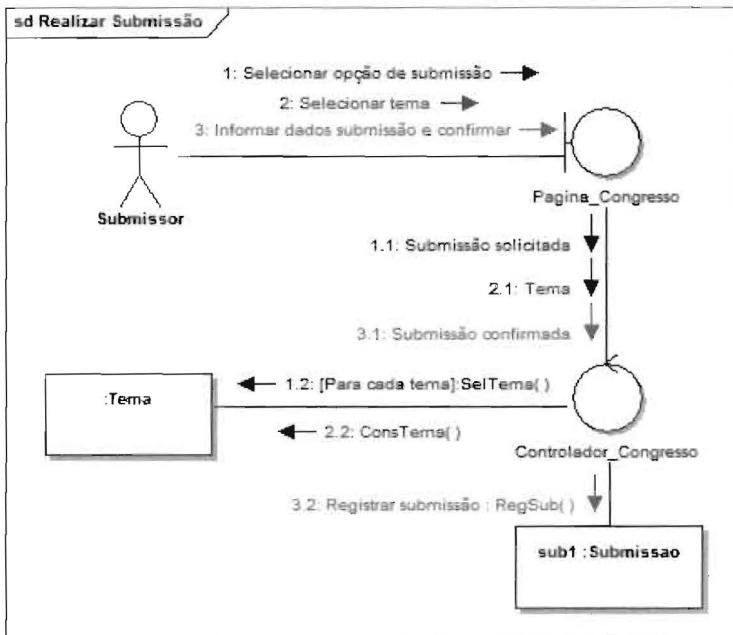


Figura 8.4 – Realizar Submissão.

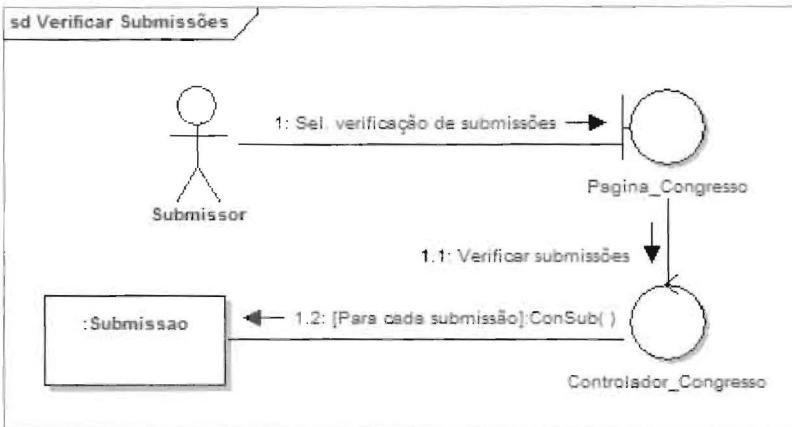


Figura 8.5 – Verificar Submissões.

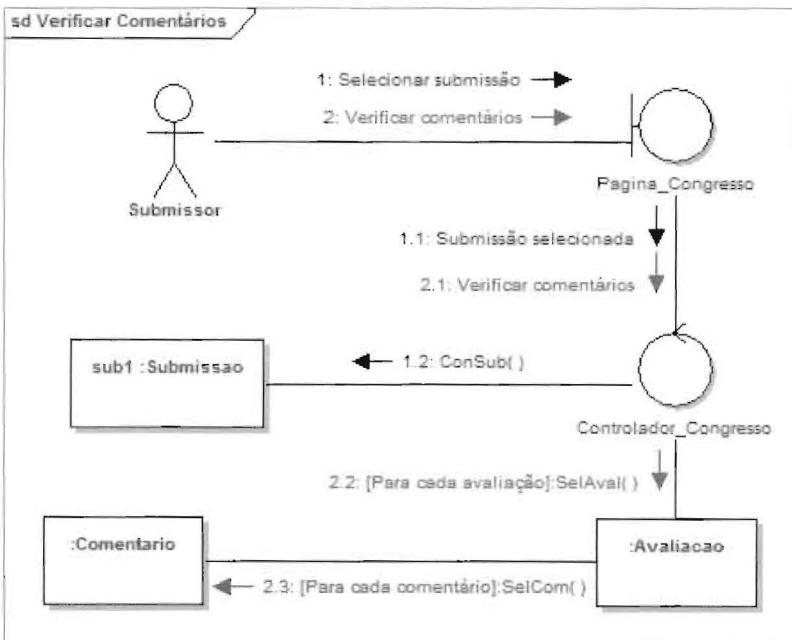


Figura 8.6 – Verificar Comentários.

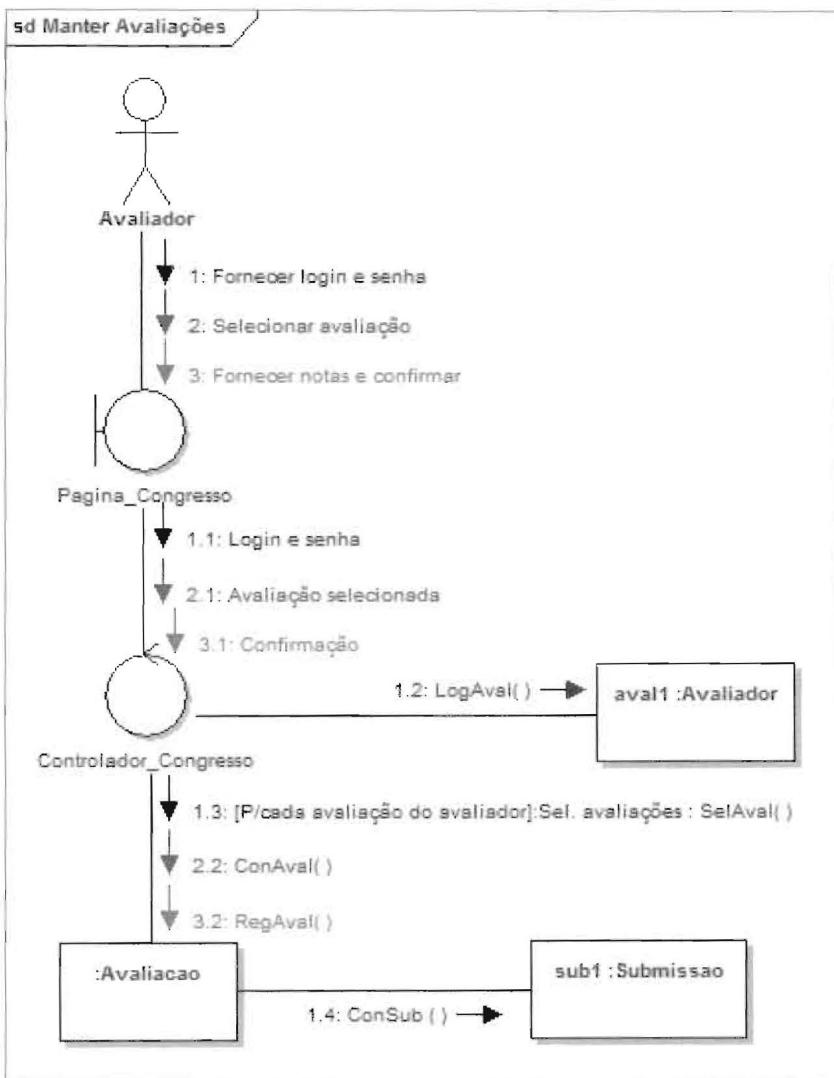


Figura 8.7 – Manter Avaliações.

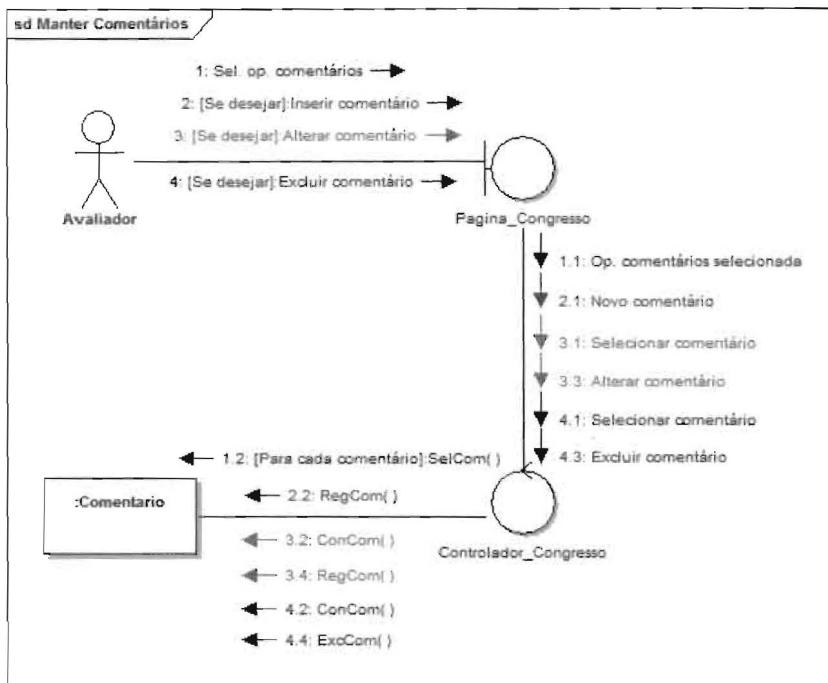


Figura 8.8 – Manter Comentários.

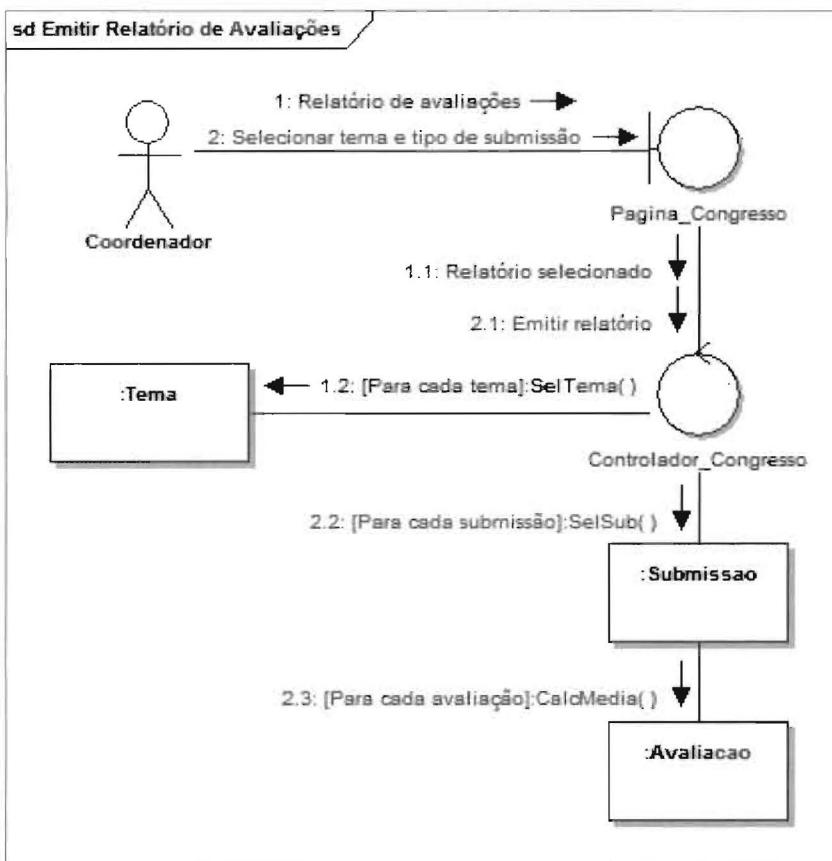


Figura 8.9 – Relatório de Avaliações.

CAPÍTULO 9

Diagrama de Máquina de Estados

Este diagrama demonstra o comportamento de um elemento por meio de um conjunto de transições de estado. O elemento modelado muitas vezes é uma instância de uma classe, no entanto, pode-se usar esse diagrama para modelar o comportamento de um caso de uso ou mesmo o comportamento de um sistema completo – neste caso estaremos considerando o caso de uso ou o sistema como objetos.

9.1 Estado

Um estado representa a situação em que um objeto se encontra em um determinado momento durante o período em que este participa de um processo. Um objeto pode passar por diversos estados dentro de um mesmo processo. Um estado pode demonstrar a espera pela ocorrência um evento, a reação a um estímulo, a execução de alguma atividade ou a satisfação de alguma condição. A Figura 9.1 apresenta um exemplo em que é representado um estado de um objeto da classe Submissão. Nesse estado, a submissão está sendo registrada. Muitas vezes o estado de um objeto é descrito no gerúndio, uma vez que, em geral, ele está executando uma atividade.



Figura 9.1 – Estado.

9.2 Transições

Uma transição representa um evento que causa uma mudança no estado de um objeto, gerando um novo estado. Uma transição é representada por uma reta ligando dois estados, contendo uma seta em uma de suas extremidades, apontando para o novo estado gerado. Transições podem possuir condições de guarda e descrições, se isto for considerado necessário. A Figura 9.2 apresenta um exemplo de transição entre estados.



Figura 9.2 – Transição.

9.3 Estados Inicial e Final

Um estado inicial é utilizado para representar o início da modelagem dos estados de um objeto. Da mesma forma, um estado final é utilizado para representar o fim dos estados modelados. A partir de um estado inicial sempre existe uma transição que gera o primeiro estado do diagrama, ao passo que o último estado gerará uma transição para o estado final. O estado inicial é representado por um círculo preenchido, e o estado final por um círculo preenchido envolvido por outro círculo não-preenchido.

A Figura 9.3 apresenta um exemplo de estados iniciais e finais. O exemplo aqui apresentado se refere aos estados do caso de uso Realizar Submissão, do sistema que temos estado a modelar neste guia.

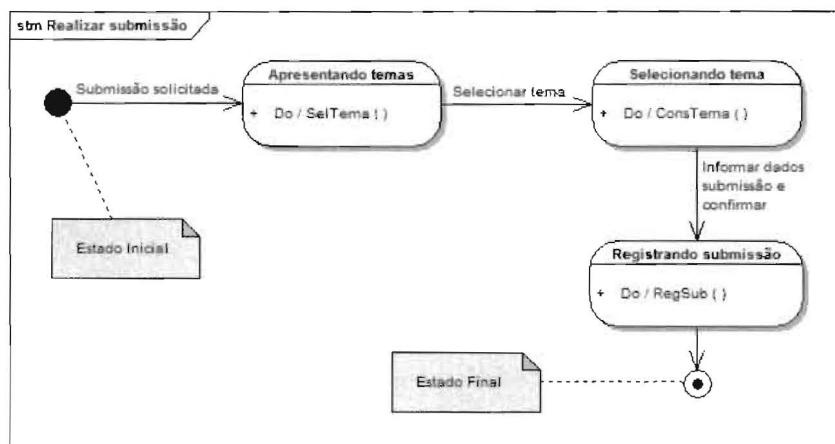


Figura 9.3 – Estados Inicial e Final.

9.4 Atividades internas

Quando em um estado, um objeto pode executar uma ou mais atividades, que são conhecidas como atividades internas. Essas atividades podem ser detalhadas por meio das seguintes cláusulas:

- **Entry** – identifica uma atividade que é executada quando o objeto assume (entra em) um estado.
- **Exit** – identifica uma atividade que é executada quando o objeto sai de um estado
- **Do** – identifica uma atividade realizada durante o tempo em que o objeto se encontra em um estado. Atividades internas do tipo Do também são chamadas de atividades de estado.

As atividades internas são representadas em uma segunda divisão do estado, conforme demonstra a Figura 94, em que o objeto executa o método RegSub, enquanto se encontra no estado Registrando submissão.

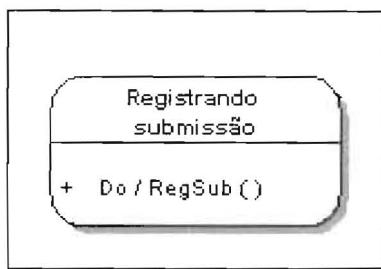


Figura 9.4 – Estado com Atividades Internas.

9.5 Transições internas

Transições internas são transições que não produzem modificações no estado de um objeto. A Figura 9.5 apresenta um exemplo de transição interna, em que, durante o registro de um cliente, é disparado um método para validar o CPF do cliente. Como este é um evento interno referente ao registro do cliente, ele é descrito como uma transição interna.

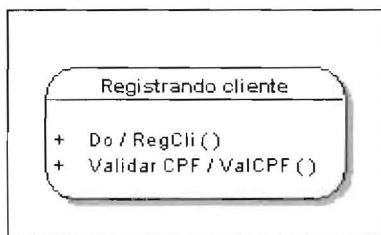


Figura 9.5 – Transição Interna.

9.6 Auto-Transições

Auto-transições saem do estado atual do objeto, podendo executar alguma ação quando dessa saída e retornam ao mesmo estado. A Figura 9.6 apresenta um exemplo de auto-transição, em que um pedido está sendo atendido e nem todos os itens do pedido se encontram disponíveis, sendo, portanto, necessário adquiri-los. Assim, sempre que um novo item for adquirido, ocorre uma auto-transição; no entanto, se algum item ainda estiver indisponível, o objeto volta para o estado Atendendo pedido. Somente quando todos os itens do pedido estiverem disponíveis será possível passar ao estado seguinte.

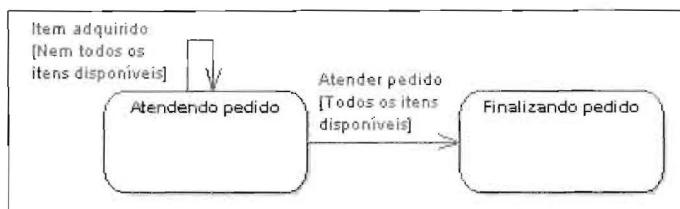


Figura 9.6 – Auto-Transição.

9.7 Pseudo-Estado de Escolha

Conhecido nas versões anteriores como estado de ponto de escolha dinâmico, representa um ponto na transição de estados de um objeto em que deve ser tomada uma decisão, a partir da qual um determinado estado será ou não gerado, normalmente em detrimento de diversos outros possíveis estados. Assim, um pseudo-estado de escolha representa uma decisão, apoiada por condições de guarda, em que se decidirá qual o próximo estado do objeto a ser gerado. Um pseudo-estado de escolha pode ser representado por um losango ou por um círculo vazio. A Figura 9.7 apresenta um exemplo de pseudo-estado de escolha.

O exemplo aqui utilizado corresponde aos estados do caso de uso Manter Comentários, referente ao sistema que estamos modelando. O leitor notará que são realizados dois testes após a listagem dos possíveis comentários existentes: o primeiro verifica se o Avaliador deseja inserir um novo comentário ou consultar um comentário já existente. Se o usuário resolver consultar um comentário ele pode, após isso, alterar o comentário, excluí-lo ou simplesmente finalizar a consulta.

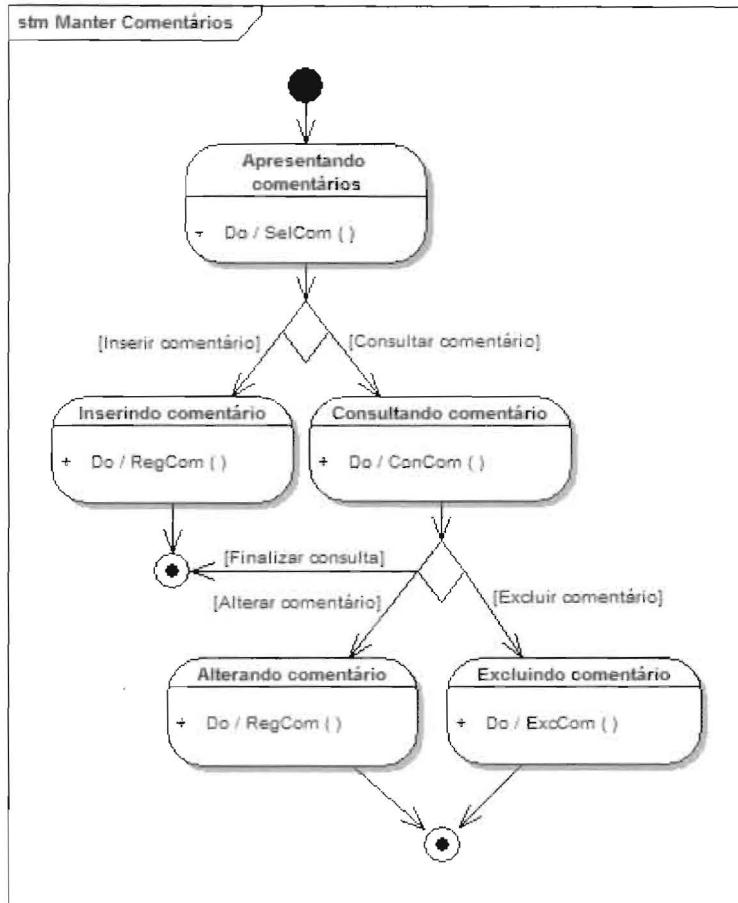


Figura 9.7 – Pseudo-estado de Escolha.

9.8 Barra de Sincronização

É utilizada quando da ocorrência de estados paralelos, causados por transições concorrentes. Sua função é determinar o momento em que o processo passou a ser executado em paralelo e em quantos subprocessos se dividiu (bifurcação) ou determinar o momento em que dois ou mais subprocessos se uniram em um único processo (união). A Figura 9.8 apresenta um exemplo de barra de sincronização, em que são modelados dois estados paralelos por que passa um objeto da classe carro no momento em que um ator necessita dirigir-lo.

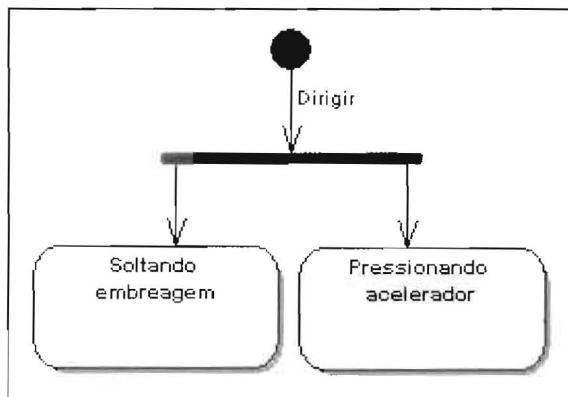


Figura 9.8 – Barra de Sincronização

9.9 Pseudo-Estado de Junção

É utilizado para projetar caminhos transacionais complexos, podendo unir múltiplos fluxos em um único ou dividir um fluxo em diversos; pode ainda utilizar condições de guarda como auxílio. A Figura 99 fornece um exemplo de pseudo-estado de junção, em que novamente modelamos os estados de um caso de uso do sistema de controle de submissões que estamos modelando – desta vez o caso de uso Realizar Login Submissor. Nesse exemplo, existem duas possibilidades: o submissor pode já estar registrado e irá fornecer seu login e senha ou o submissor ainda não está registrado e terá de se auto-registrar. Após se auto-registrar, o submissor fornecerá seu login e senha de qualquer modo, e o fluxo volta a ser unificado pelo pseudo-estado de junção que é extremamente semelhante a um estado inicial.

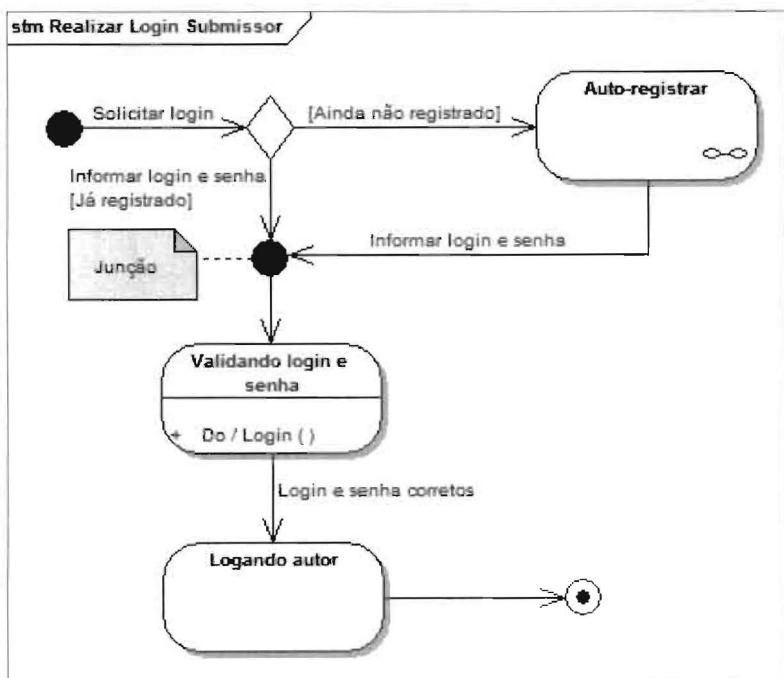


Figura 9.9 – Pseudo-Estado de Junção.

9.10 Estado Composto

É um estado que contém internamente dois ou mais estados, chamados de subestados. São utilizados para “dissecar” um Estado individual, ou seja, um estado composto é um estado que foi “explorado”, de maneira a apresentar detalhadamente todas as etapas por que passa o objeto quando no estado em questão. A Figura 9.10 apresenta um exemplo de estado composto, no qual detalhamos os estados internos do estado “Registrando cliente” já apresentado anteriormente.

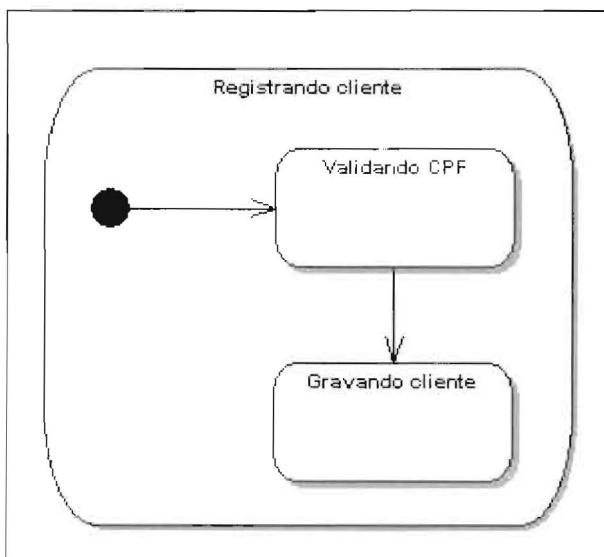


Figura 9.10 – Estado Composto.

9.11 Estado de História

Um estado de história representa o registro do último subestado em que um objeto se encontrava, quando, por algum motivo, o processo foi interrompido. Assim, por meio do estado de história, pode-se retornar exatamente ao último subestado em que o objeto se encontrava quando da interrupção do processo. A Figura 9.11 apresenta um exemplo de estado de história aplicado a um estado composto em que é calculado o novo valor de todos os produtos de uma categoria. Caso haja a necessidade de suspender o recálculo, o estado de história servirá para guardar o último

estado em que se encontrava o processo antes da suspensão. O fato de o estado de história apontar para um subestado específico não significa que seja este o estado que ele está guardando.

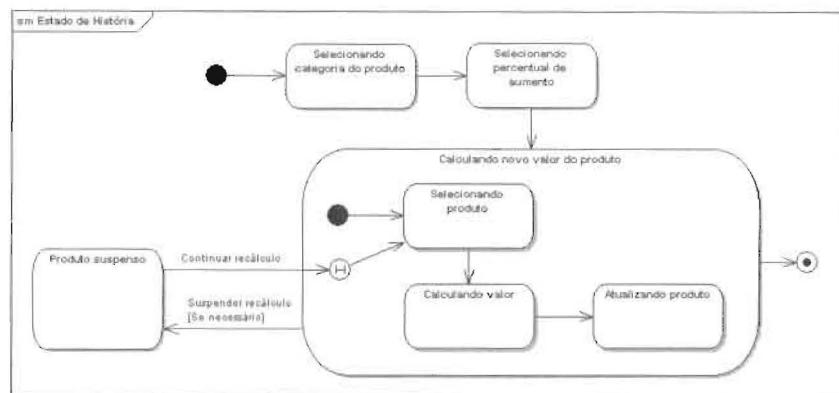


Figura 9.11 – Estado de História.

Existe também o estado de história profunda, representado pelo mesmo símbolo seguido de um asterisco, utilizado quando da ocorrência de estados compostos dentro de estados compostos. Assim, o estado de história profunda guarda e recupera o último subestado em qualquer dos estados compostos em que ele possa estar.

9.12 Estado de Submáquina

Um estado de submáquina é equivalente a um estado composto, no entanto não apresenta seus subestados. Ele determina que existem estados internos, mas que estes estão detalhados em um outro diagrama, normalmente com o nome do próprio estado de submáquina. Isso é vantajoso por permitir fazer referência a outros diagramas já modelados.

A Figura 9.12 apresenta um estado de submáquina em que modelamos os estados de outro caso de uso do sistema que vimos modelando, referente ao processo de Verificar Submissões, no qual, após visualizar a situação de suas submissões, o autor pode, se quiser, visualizar os comentários de uma submissão específica. Como esse processo refere-se a outro caso de uso, nós o modelamos em separado e o referenciamos por meio de um estado de submáquina.

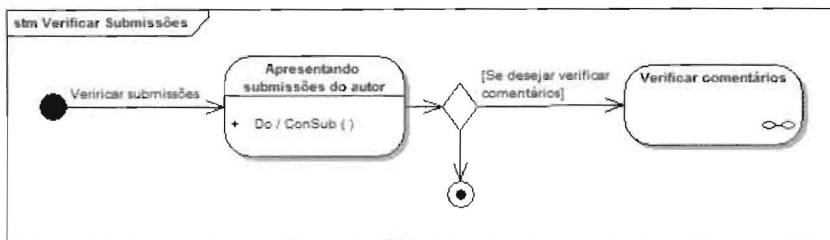


Figura 9.12 – Estado de submáquina.

9.13 Estados de Entrada e Saída (Entry e Exit States)

Normalmente utilizados juntamente com estados de submáquina, demonstram pontos de entrada e saída que serão somente usados em casos excepcionais. A Figura 9.13 apresenta um exemplo de estados de entrada e saída, no qual o estado de entrada é representado por um círculo vazio na borda do estado de submáquina e o estado de saída é representado por um círculo com um X. Esses estados normalmente representam exceções, por este motivo o exemplo apresenta transições normais que não necessitam desses estados e exceções que os utilizam.

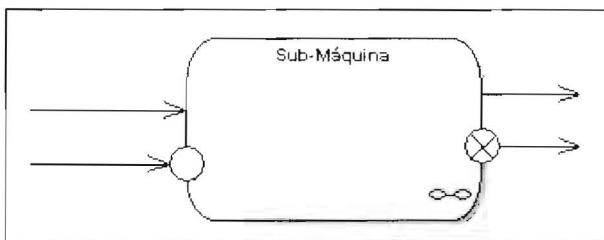


Figura 9.13 – Estados de Entrada e Saída.

9.14 Pseudo-Estado de Término

Força o término da execução de uma máquina de estados, em razão, por exemplo, da ocorrência de uma exceção. É representado por um "X", conforme demonstra a Figura 9.14.

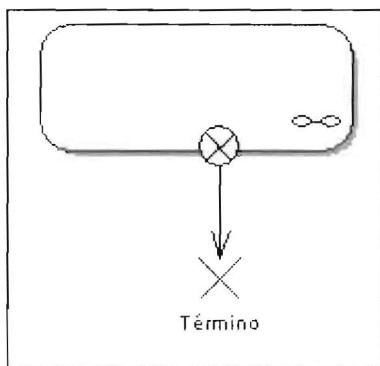


Figura 9.14 – Pseudo-Estado de Término.

9.15 Estado de Sincronismo

Em alguns processos pode eventualmente haver a necessidade de que estes estejam de alguma forma sincronizados, por vezes sendo necessário que um processo paralelo espere por outro. Assim, é preciso existir um estado de sincronismo, cuja função é permitir que os relógios de dois ou mais processos paralelos estejam sincronizados em um determinado momento do processo, conforme demonstra a Figura 9.15, que ilustra uma situação em que, sempre que um semáforo troca de sinal, força a troca de sinal do outro semáforo.

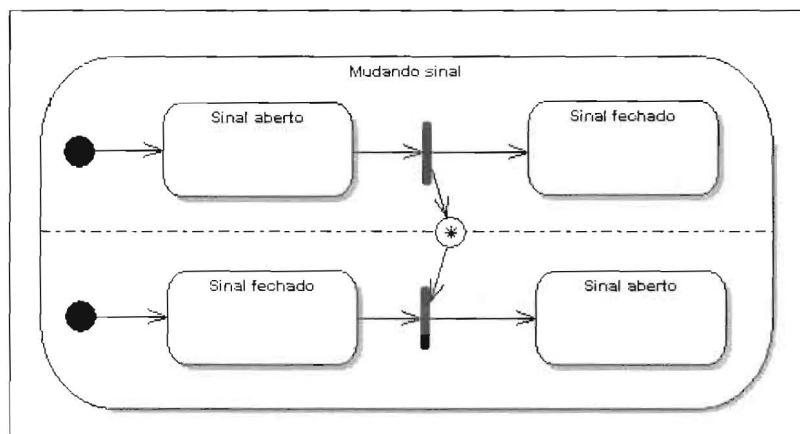


Figura 9.15 – Estado de Sincronismo.

9.16 Estudo de Caso

Aqui iremos identificar os principais diagramas de máquina de estados referentes ao sistema que vem sendo modelado. Os diagramas que não se encontram nesta seção já foram exemplificados nas seções anteriores. É importante destacar que estamos modelando os estados dos principais casos de uso do sistema, em razão de os objetos deste sistema apresentarem poucos estados individuais.

9.16.1 Verificar Comentários

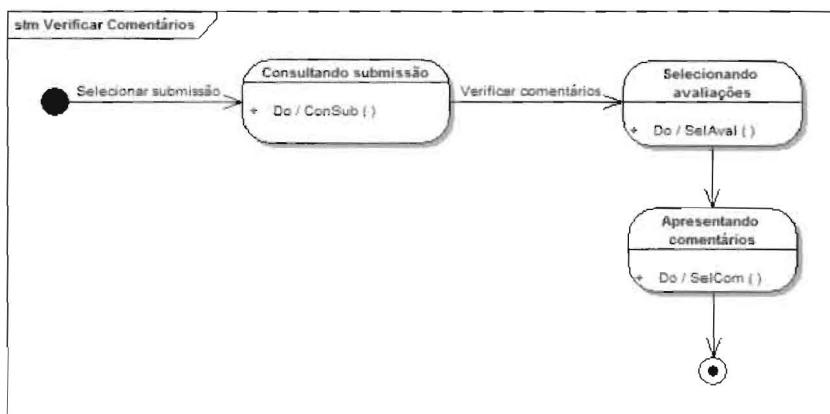


Figura 9.16 – Verificar Comentários.

Este diagrama (Figura 9.16) apresenta os estados do caso de uso Verificar Comentários, que se inicia com o estado Consultando a submissão, no qual será executado o método ConSub e a submissão selecionada pelo submissor será carregada. Em seguida, passa-se ao estado Seleccionando avaliações, no qual é executado o método SelAval, que carregará todas as avaliações da submissão selecionada. Finalmente, passa-se para o estado Apresentando comentários, em que todos os possíveis comentários de cada avaliação serão carregados pelo método SelCom.

9.16.2 Manter Avaliações

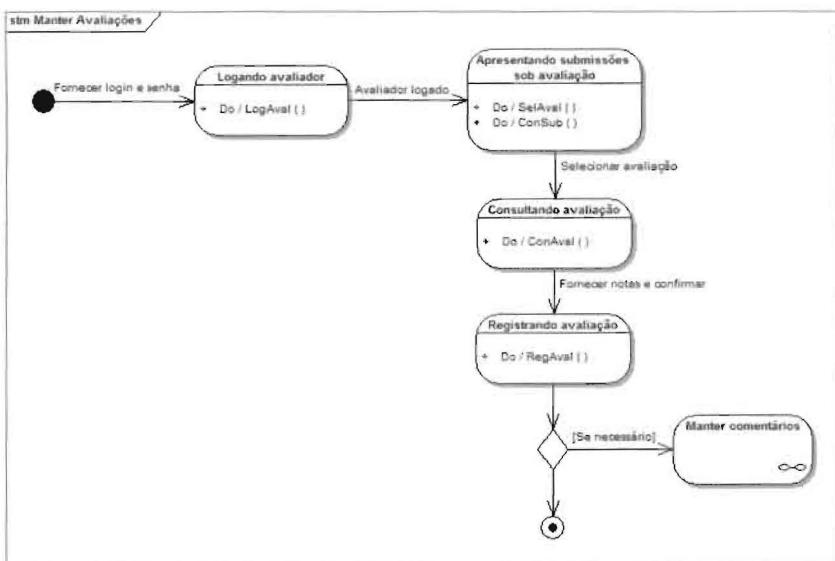


Figura 9.17 – Manter Avaliações.

Este diagrama representa os estados por que passa o caso de uso Manter Avaliações. O primeiro estado identificado representa o momento em que o Avaliador se loga no sistema, passando-se para o estado em que são apresentadas as avaliações sob responsabilidade do Avaliador. Após isso, é representado o estado em que a avaliação escolhida é consultada e finalmente o estado em que a avaliação é registrada.

Observe que ao final do diagrama existe uma situação, demonstrada por um pseudo-estado de escolha, a partir do qual pode-se fazer referência a um estado de submáquina, representando o processo de Manter Comentários, caso o avaliador ache necessário.

9.16.3 Relatório de Avaliações

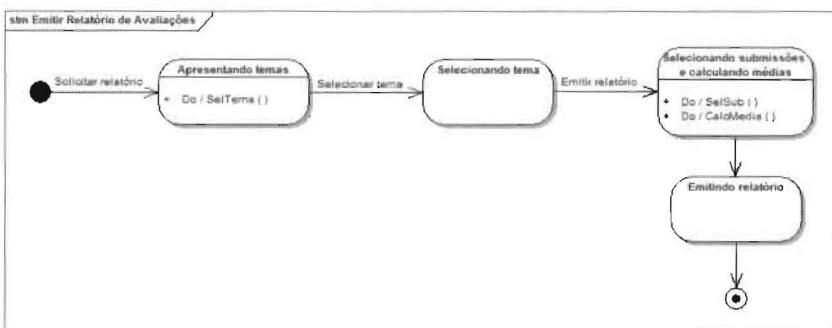


Figura 9.18 – Relatório de Avaliações.

Este último diagrama representa os estados do caso de uso Relatório de Avaliações. Este processo inicia com o estado Apresentando temas, em que o método SelTema retorna todos os temas aceitos, passando pelo estado Selecionando tema, no qual o Coordenador escolherá um dos temas aceitos pelo evento. Por fim, o processo atinge o estado em que as submissões do tipo escolhido são selecionados e a média de suas avaliações é calculada, de modo que são executados os métodos SelSub e CalcMedia. O último estado representa simplesmente a emissão do relatório propriamente dito.

CAPÍTULO 10

Diagrama de Atividade

O diagrama de atividade era considerado um caso especial do diagrama de gráficos de estados, chamado diagrama de máquina de estados na versão 2.0. A partir desta versão passou a ser considerado um diagrama totalmente independente, deixando inclusive de se basear em máquinas de estados e passando a se basear em redes de Petri.

O diagrama de atividade é o diagrama com maior ênfase ao nível de algoritmo da UML e provavelmente um dos mais detalhistas. Este diagrama apresenta muitas semelhanças com os antigos fluxogramas utilizados para desenvolver a lógica de programação e determinar o fluxo de controle de um algoritmo, sendo possível inclusive encontrar diagramas de atividade utilizando pseudocódigo ou até mesmo uma linguagem de programação real, como Java, C ou Pascal.

Esse diagrama é utilizado, como o próprio nome diz, para modelar atividades, que podem ser um método ou um algoritmo, ou mesmo um processo completo. Atividades podem descrever computação procedural; neste contexto elas são os métodos correspondentes às operações sobre classes. Atividades também podem ser aplicadas à modelagem organizacional para engenharia de processos de negócios e workflow. Finalmente atividades podem também ser usadas para modelagem de sistemas de informação para especificar processos ao nível de sistema. Uma atividade é composta por um conjunto de ações, ou seja, os passos necessários para que a atividade seja concluída.

Um diagrama de atividade pode modelar mais de uma atividade. Uma atividade sempre conterá ações, as quais, no entanto, não necessariamente estarão representadas dentro da atividade, como quando for necessário fazer referência a uma atividade já modelada ou para poupar espaço no diagrama. Além disso, o diagrama de atividade pode ser usado para representar dois tipos de fluxo: de controle e de objetos, conforme será visto a seguir.

10.1 Nô de Ação

Nós de ação são os elementos mais básicos de uma atividade. Um nó de ação representa um passo, uma etapa que deve ser executada em uma atividade. Um nó de ação é atômico, não podendo ser decomposto. A Figura 10.1 apresenta um exemplo de nó de ação.

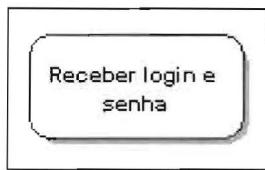


Figura 10.1 – Nô de Ação.

10.2 Controle de Fluxo

O controle de fluxo é um conector que liga dois nós, enviando sinais de controle. É representado por uma reta contendo uma seta apontando para o novo nó e partindo do antigo, podendo conter uma descrição, uma condição de guarda ou uma restrição, chamada neste diagrama de peso (weight), que determina, por exemplo, o número mínimo de sinais que devem ser transmitidos pelo fluxo. Um sinal (token) pode conter valores de controle, objetos ou dados, sendo que estes dois últimos somente podem ser transmitidos por meio de um fluxo de objetos que será explicado neste capítulo. A Figura 10.2 apresenta um exemplo de controle de fluxo.

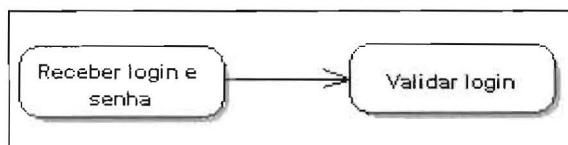


Figura 10.2 – Controle de Fluxo.

10.3 Nô Inicial

Este componente pertence ao grupo de nós de controle, utilizados para o controle de fluxo. É usado para representar o início da atividade e é representado por um círculo preenchido. A Figura 10.3 apresenta um exemplo de nó inicial.

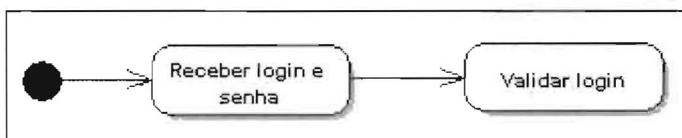


Figura 10.3 – Nô Inicial.

10.4 Nô Final

Este componente é um nó de controle usado para representar o fim de uma atividade; é representado por um círculo preenchido dentro de um círculo vazio. A Figura 10.4 apresenta um exemplo de nó final.

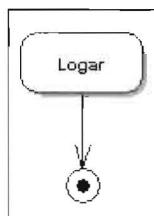


Figura 10.4 – Nô Final.

10.5 Nô de Decisão

Um nó de decisão é também um nó de controle, utilizado para representar uma escolha entre dois ou mais fluxos, dos quais um será escolhido em detrimento dos outros. Em geral, um nó de decisão é acompanhado por condições de guarda, que determinam a condição para que um fluxo possa ser escolhido. Um nó de decisão pode ser utilizado também para unir um fluxo dividido por um nó de decisão anterior, quando então passa a chamar-se nó de união. A Figura 10.5 apresenta um exemplo de nó de decisão.

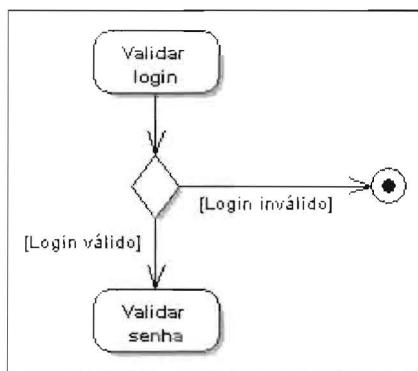


Figura 10.5 – Nό de Decisão.

10.6 Exemplo de Diagrama de Atividade

Nesta seção demonstraremos um exemplo de diagrama de atividade, no qual modelamos o processo de login de um submissor, referente ao sistema que temos modelado ao longo deste guia, conforme demonstra a Figura 10.6.

Como o leitor poderá observar, existem duas atividades aqui representadas: a atividade de Auto-registrar e a de Login. A primeira está apenas sendo referenciada, uma vez que seus nós de ação não estão sendo representados, e deverá ser detalhada em um diagrama em separado. Já a segunda apresenta seus nós de ação internos, ou seja, os passos necessários para que o autor possa se logar. Ao observarmos a figura percebemos que primeiro é realizado um teste para determinar se o autor já está registrado ou não, para decidir qual atividade deverá ser executada. Caso o autor já esteja registrado, ele informará seu login e senha, que serão validados e, se ambos estiverem corretos, o sistema o logará.

Nesse exemplo existem duas atividades, mas é mais comum modelar-se apenas uma. Optamos por esta abordagem para tornar o exemplo mais completo, uma vez que o submissor pode não estar registrado e, neste caso, precisará registrar-se antes de poder se logar. Poderíamos, no entanto, ter modelado somente a atividade de Login e, se assim procedêssemos, não seria obrigatório representar a atividade envolvendo os nós de ação.

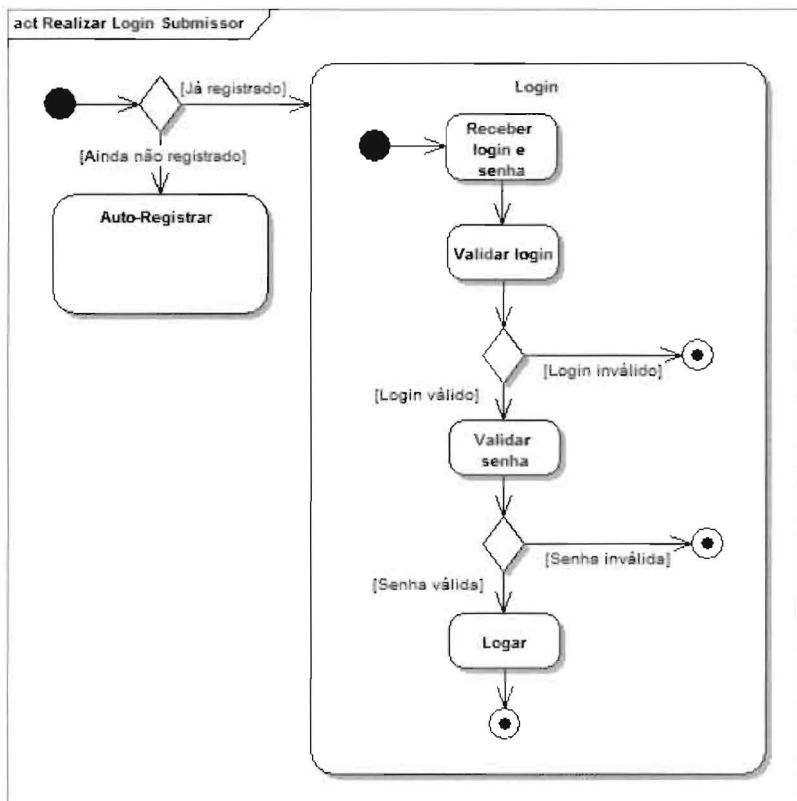


Figura 10.6 – Exemplo de Diagrama de Atividade – Login do Submissor.

10.7 Conectores

Conectores são basicamente atalhos para o fluxo, utilizados quando existe uma distância relativamente grande entre os nós que o fluxo precisa ligar. A Figura 10.7 apresenta um exemplo de conectores, no qual modelamos o processo de Relatório de Avaliações referente ao sistema de submissões que estamos modelando. O leitor notará que existe um conector denominado “A” que liga os nós de ação “Posicionar próxima submissão” e “Selecionar avaliações da submissão”.

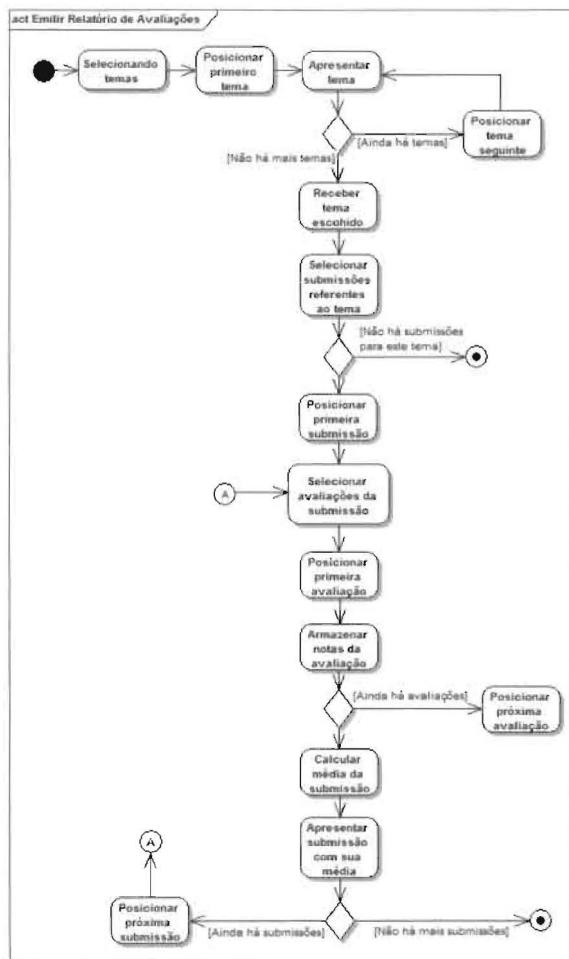


Figura 10.7 – Exemplo de Conectores – Relatório de Avaliações.

10.8 Subatividade

Uma subatividade representa a execução de uma seqüência não-atômica de etapas que possuem alguma duração. Pode ser comparada a uma sub-rotina que já foi ou será explorada em outro diagrama de atividade e, portanto, não há necessidade de explorá-la no diagrama atual. Uma subatividade pode ainda representar uma rotina contida em uma biblioteca cujas etapas de execução são desconhecidas. Este componente possui a mesma representação de uma atividade, exceto por possuir em seu canto inferior direito um símbolo que representa um diagrama de atividade. Uma subatividade difere-se de uma atividade por não ser independente, só podendo ser executada a partir de uma atividade específica.

A Figura 10.8 apresenta um exemplo de subatividade, em que um arquivo criptografado é recebido e repassado para uma rotina de decodificação. Como as etapas para decodificar o arquivo são desconhecidas ou já foram modeladas em outro diagrama, não há necessidade de defini-las novamente.

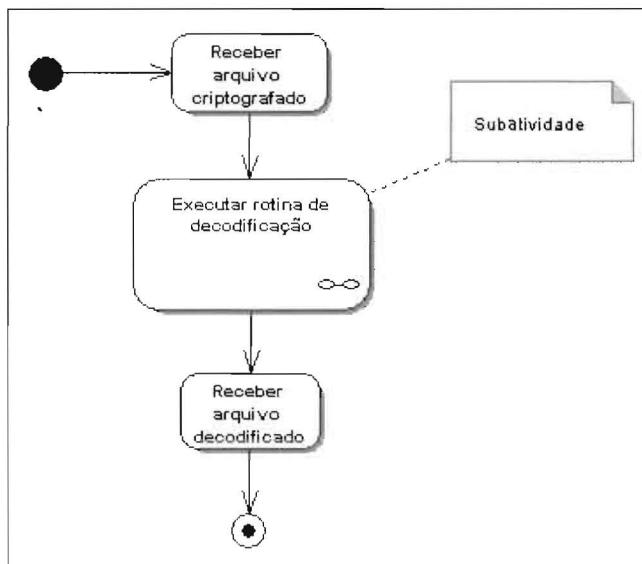


Figura 10.8 – Exemplo de Estado de Subatividade.

10.9 Nó de Bifurcação/União

Um nó de bifurcação/união é um nó de controle que pode tanto dividir um fluxo em dois ou mais fluxos concorrentes, como mesclar dois ou mais fluxos concorrentes em um único fluxo de controle. Este nó é representado por uma barra que pode estar tanto na horizontal como na vertical. A Figura 10.9 apresenta um exemplo de nó de bifurcação/união.

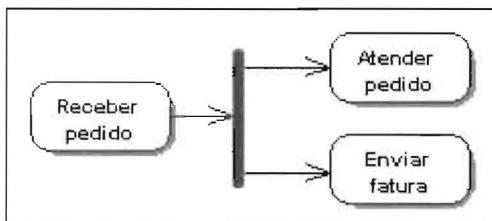


Figura 10.9 – Exemplo de Nó de Bifurcação/União.

10.10 Final de Fluxo

Representa o encerramento de uma rotina representada pelo fluxo, mas não de toda a atividade. O símbolo de final de fluxo é representado por um círculo com um X, conforme é demonstrado na Figura 10.10, na qual assumimos uma situação em que muitos componentes podem ser construídos, e instalados. Quando o último componente for construído esta parte do fluxo estará concluída como demonstra o símbolo de final de fluxo; no entanto, outras etapas deverão ser concluídas durante a atividade.

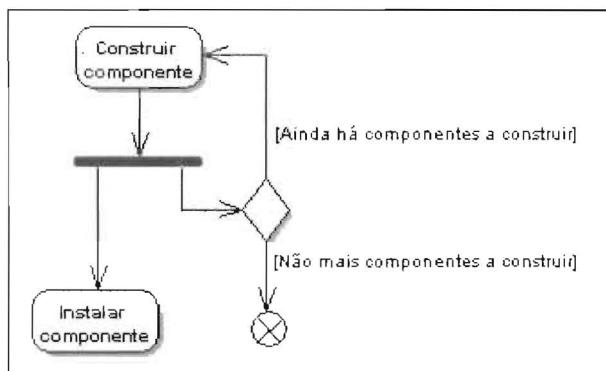


Figura 10.10 – Exemplo de Final de Fluxo.

10.11 Fluxo de Objetos

Um fluxo de objetos é um conector que pode possuir objetos ou dados passando através dele. Representa o fluxo de valores (objetos ou dados) que são enviados a partir de um nó de objeto ou para um nó de objeto.

10.12 Nô de Objeto

Um nó de objeto representa uma instância de uma classe, que pode estar disponível em um determinado ponto da atividade. Nós de objeto podem ser utilizados de diversas formas, em sua forma mais tradicional, são representados como um retângulo, como demonstra a Figura 10.11. O fluxo de objetos pode ser utilizado para modificar o estado de um objeto, definindo um valor para um de seus atributos ou mesmo instanciando o objeto. Um objeto pode apresentar multiplicidade, que, neste caso, determina o número mínimo e máximo de valores que o objeto aceita. Se existir um valor mínimo determinado, a ação só inicia quando ele for atingido. Um objeto pode apresentar também um “Upperbound” entre colchetes que determina o valor máximo de valores que o objeto pode conter. Em tempo de execução, quando este valor é ultrapassado o fluxo é interrompido.

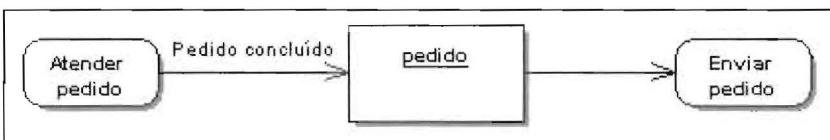


Figura 10.11 – Exemplo de Objeto e Fluxo de Objeto.

10.13 Alfinetes (Pins)

O fluxo de objetos muitas vezes exige a utilização de alfinetes (pins). Alfinetes são nós de objeto que representam uma entrada para uma ação ou uma saída de uma ação. Eles fornecem valores para as ações e recebem os valores resultantes delas. O próprio objeto apresentado anteriormente na Figura 10.11 é um alfinete. Os alfinetes podem ser representados como pequenos retângulos e, em geral, são colocados ao lado das ações às quais estão ligados, conforme demonstra a Figura 10.12. Quando o tipo de entrada e saída é o mesmo, pode-se utilizar um único retângulo no centro

do fluxo de dois nós de ação, conforme apresentado anteriormente na Figura 10.11.

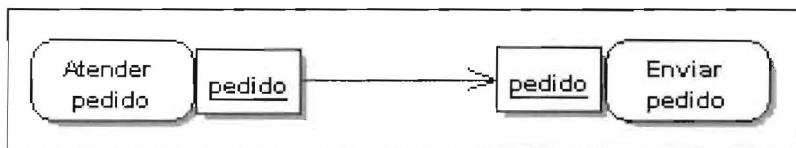


Figura 10.12 – Exemplo de Alfinetes.

10.14 Nô de Parâmetro de Atividade

Um nó de parâmetro de atividade é um nó de objeto utilizado para representar a entrada ou saída de um fluxo de objetos em uma atividade. A Figura 10.13 apresenta um exemplo de nó de parâmetro de atividade. O nó de objeto “Materiais de Produção” é um nó de parâmetro de entrada na atividade, ao passo que os nós de objeto “Computadores Aceitos” e “Computadores Rejeitados” são nós de parâmetro de saída na atividade.

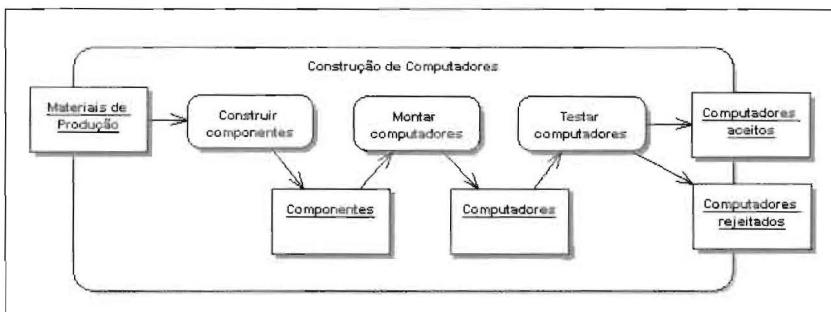


Figura 10.13 – Exemplo de Parâmetro de Atividade.

10.15 Exceções

É possível, no diagrama de atividade, detalhar a ocorrência de exceções, bastante comuns na maioria das linguagens de programação atuais. Para descrever a manipulação de uma exceção, o diagrama de atividade fornece uma seta em forma de raio que aponta para a rotina de tratamento da interrupção ou exceção, chamada de fluxo de interrupção, conforme é demonstrado na Figura 10.14.

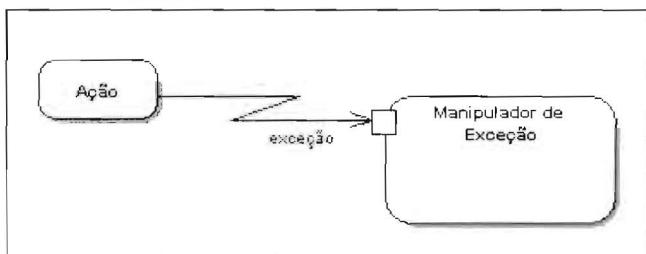


Figura 10.14 – Tratamento de Exceção.

Observe que a seta de exceção atinge um quadrado no manipulador de exceção. Esse quadrado é também um nó de objeto, chamado nó de entrada de exceção.

10.16 Ação de Objeto de Envio

É uma ação que representa o envio de um objeto para seu objeto de destino. Uma ação de objeto de envio é representada por um retângulo com uma protuberância triangular em seu lado direito.

10.17 Ação de Evento de Aceitação

É uma ação que representa a espera de ocorrência de um evento de acordo com determinadas condições. Uma ação de evento de aceitação é representada por um retângulo com uma reentrância triangular em seu lado direito. A Figura 10.15 apresenta um exemplo de ações de objeto de envio e de evento de aceitação.

Nesse exemplo, utilizamos um fluxo de controle referente ao envio de um texto para impressão. Após preparar o texto, é enviado um sinal à impressora para verificar se esta está pronta para receber o material a ser impresso. Em seguida, a impressora envia um sinal em resposta informando que ela pode imprimir o documento. Observe que a impressora é representada como um objeto e que os sinais são enviados e recebidos por meio de um fluxo de objeto.

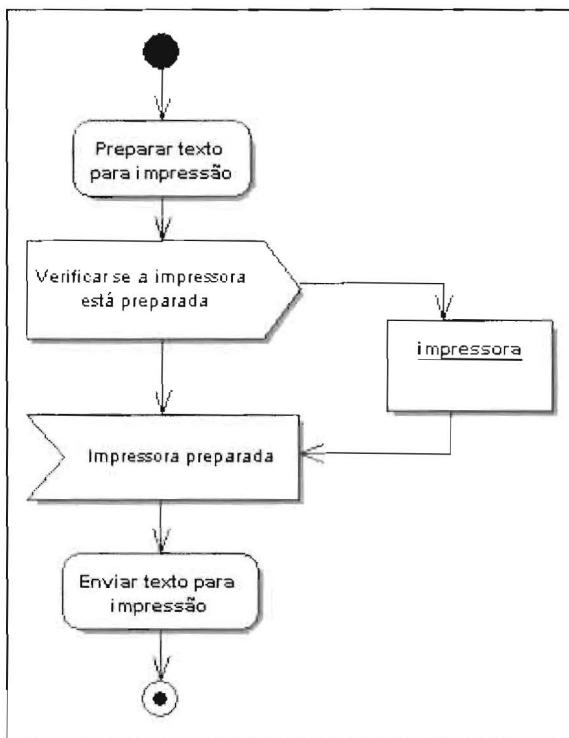


Figura 10.15 – Ações de Objeto de Envio e de Evento de Aceitação.

10.18 Ação de Evento de Tempo de Aceitação

É uma variação da ação de evento de aceitação que leva em consideração o tempo para que o evento possa ser disparado. Pode ser comparada com uma trigger. A Figura 10.16 apresenta um exemplo de ação de evento de tempo de aceitação.

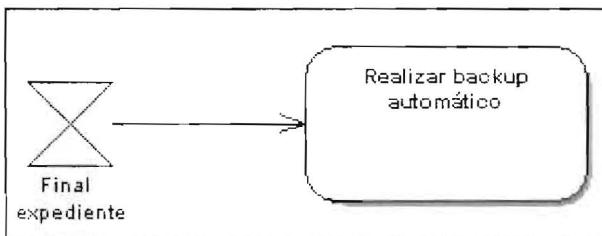


Figura 10.16 – Ação de Evento de Tempo de Aceitação.

10.19 Nô de Repositório de Dados (Data Store Node)

Um nó de repositório de dados é um nó de objeto representado com o estereótipo de <<datastore>>. Um nó de repositório de dados é usado para armazenar dados ou objetos permanentemente. É um tipo especial de nó de buffer central, cuja função é gerenciar fluxos de múltiplas fontes e destinos. Os dados que entram em um nó de repositório de dados são armazenados permanentemente, atualizando os dados que já existiam. Os dados que vêm de um nó de repositório de dados são uma cópia dos dados originais. A Figura 10.17 apresenta um exemplo de nó de repositório de dados.

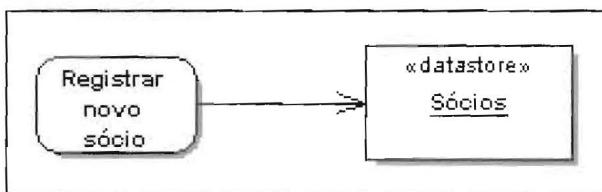


Figura 10.17 – Nô de Repositório de Dados.

10.20 Partição de Atividade

As partições de atividade permitem representar o fluxo de um processo que passa por diversos setores ou departamentos de uma empresa, ou mesmo um processo que é manipulado por diversos atores. As partições de atividade são formadas por retângulos representando divisões que identificam as zonas de influência de um determinado setor sobre um determinado processo. A Figura 10.18 apresenta um exemplo de partições de atividade.

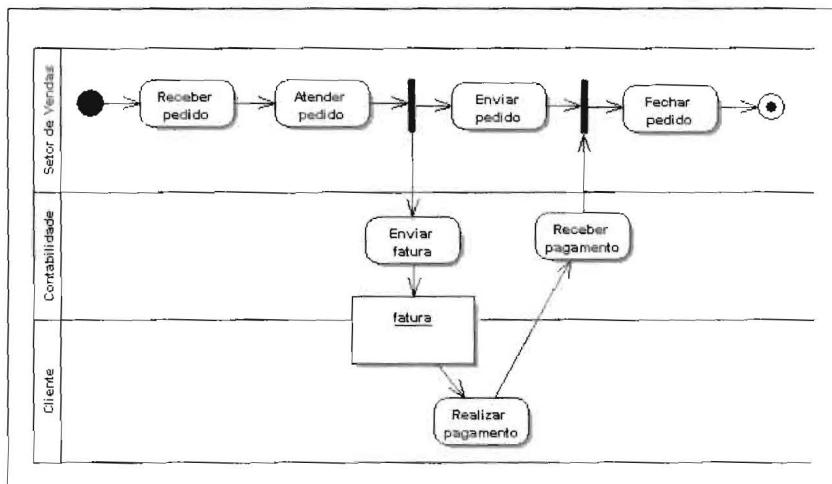


Figura 10.18 – Partições de Atividade.

10.21 Região de Atividade Interrompível

Uma região de atividade interrompível engloba um certo número de elementos da atividade que podem sofrer uma interrupção, assim todo o processo envolvido pela região poderá ser interrompido, não importando em que elemento da atividade a interrupção ocorreu. A região é delimitada por um retângulo tracejado com as bordas arredondadas. É necessário utilizar uma ação de objeto de envio para indicar a interrupção, conforme demonstra o exemplo da Figura 10.19, em que a atividade representada na Figura 10.18 foi melhorada para permitir a opção de cancelamento do pedido.

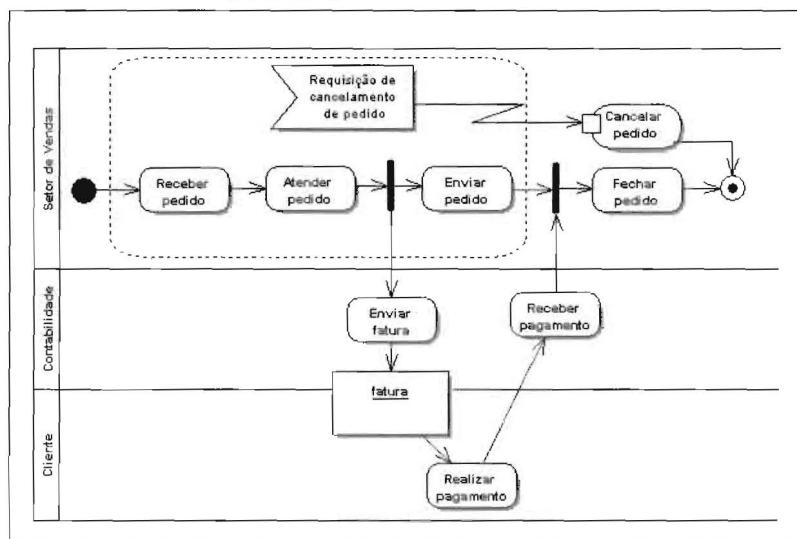


Figura 10.19 – Exemplo de Região de Atividade Interrompível.

10.22 Região de Expansão

Uma região de expansão é uma região de atividade estruturada que é executada múltiplas vezes de acordo com os elementos de uma coleção de entrada. Uma região de expansão engloba uma parte da atividade e representa uma região aninhada na qual cada entrada é uma coleção de valores. A região de expansão é executada uma vez para cada elemento na coleção de entrada. A cada execução da região um valor de saída é inserido na coleção de saída na mesma posição que os elementos de entrada. Os elementos de entrada e saída são nós de objeto, cada um representado por três pequenos quadrados juntos, colocados nas extremidades da região de expansão, aqui chamados de nós de expansão.

Uma região de expansão pode ser iterativa, em que as interações ocorrem na ordem dos elementos; paralela, na qual todas as interações são independentes; ou de fluxo (stream), na qual há uma única execução da região em que os valores na coleção de entrada são extraídos e colocados na execução da região de expansão como um fluxo. A Figura 10.20 apresenta um exemplo de região de expansão, no qual foi modelado um cálculo recursivo de factorial.

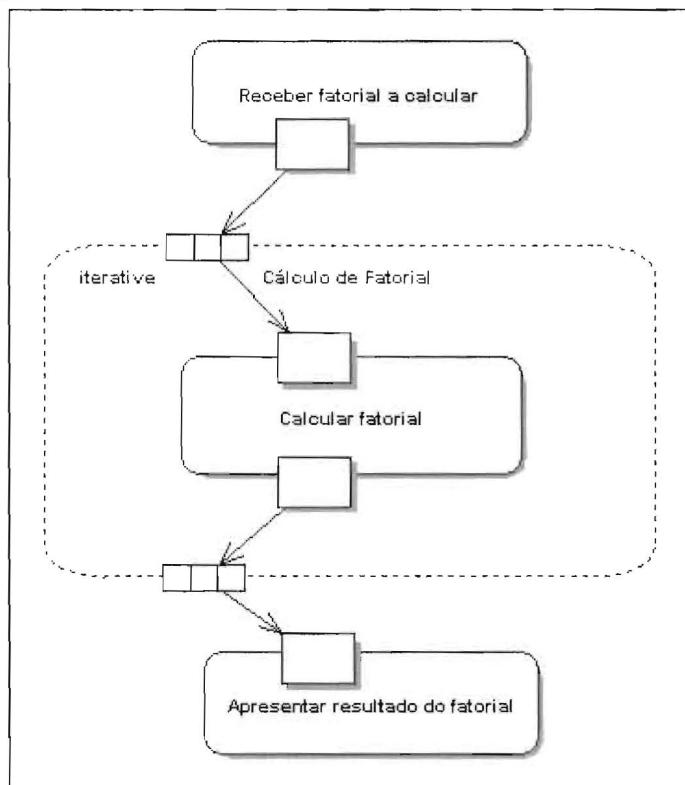


Figura 10.20 – Exemplo de Região de Expansão.

10.23 Estudo de Caso

Aqui iremos identificar os principais diagramas de atividade referentes ao sistema que vem sendo modelado. Optamos por modelar cada um dos casos de uso mais importantes novamente, desta vez modelando as ações necessárias para que cada um desses processos possa ser executado. Neste estudo de caso modelamos apenas o fluxo de controle das atividades. Os diagramas de atividade referentes ao processo de Login do Submissor e Relatório de Avaliações não se encontram nesta seção, uma vez que já foram exemplificados anteriormente. Por representarem passo a passo as ações dos processos que já modelamos anteriormente, por meio de outros diagramas, acreditamos que estes diagramas de atividade sejam suficientemente auto-explicativos.

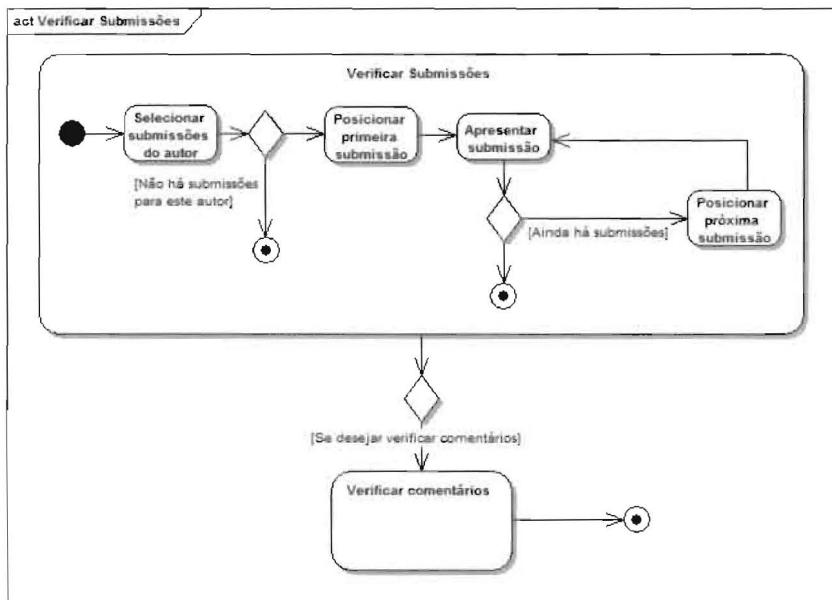


Figura 10.21 – Verificar Submissões.

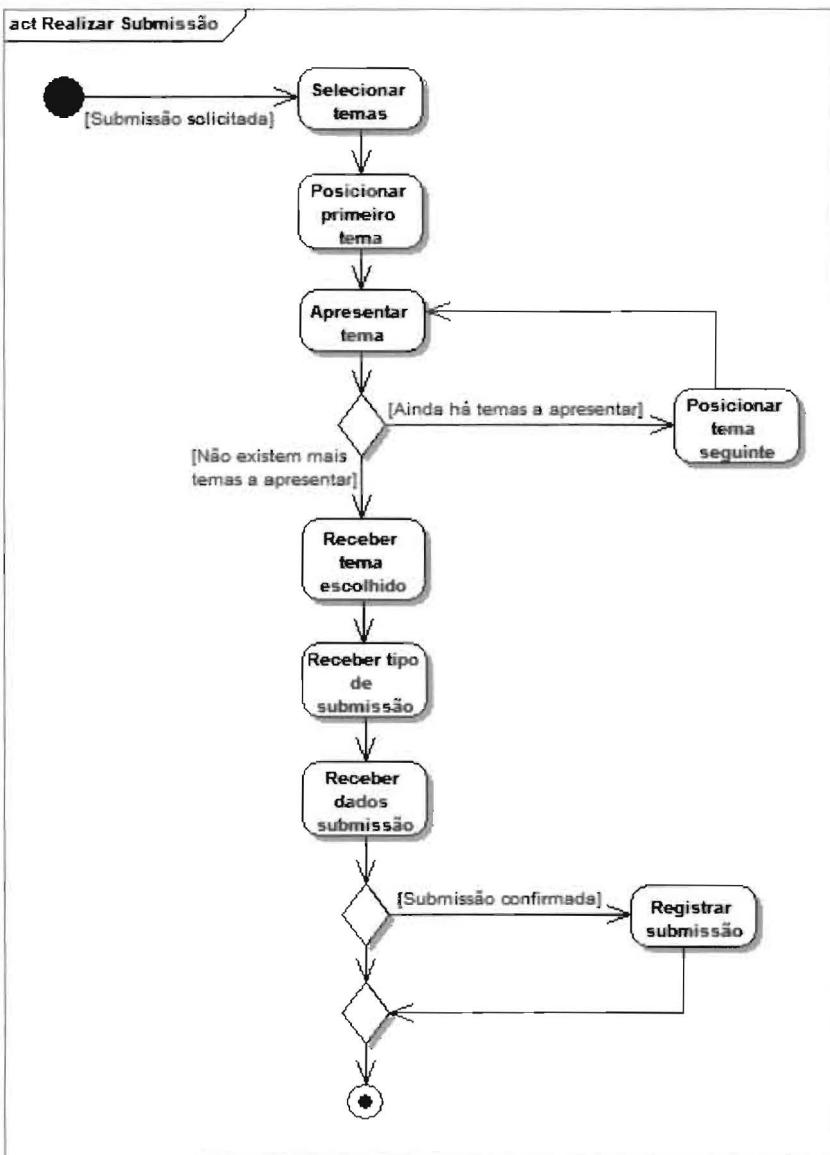


Figura 10.22 – Realizar Submissão.

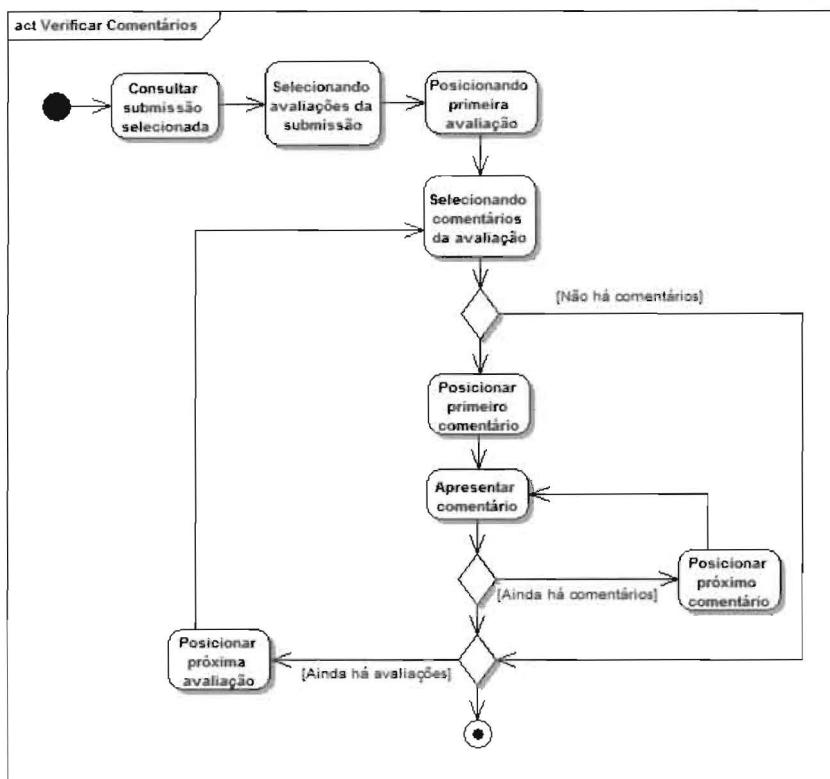


Figura 10.23 – Verificar Comentários.

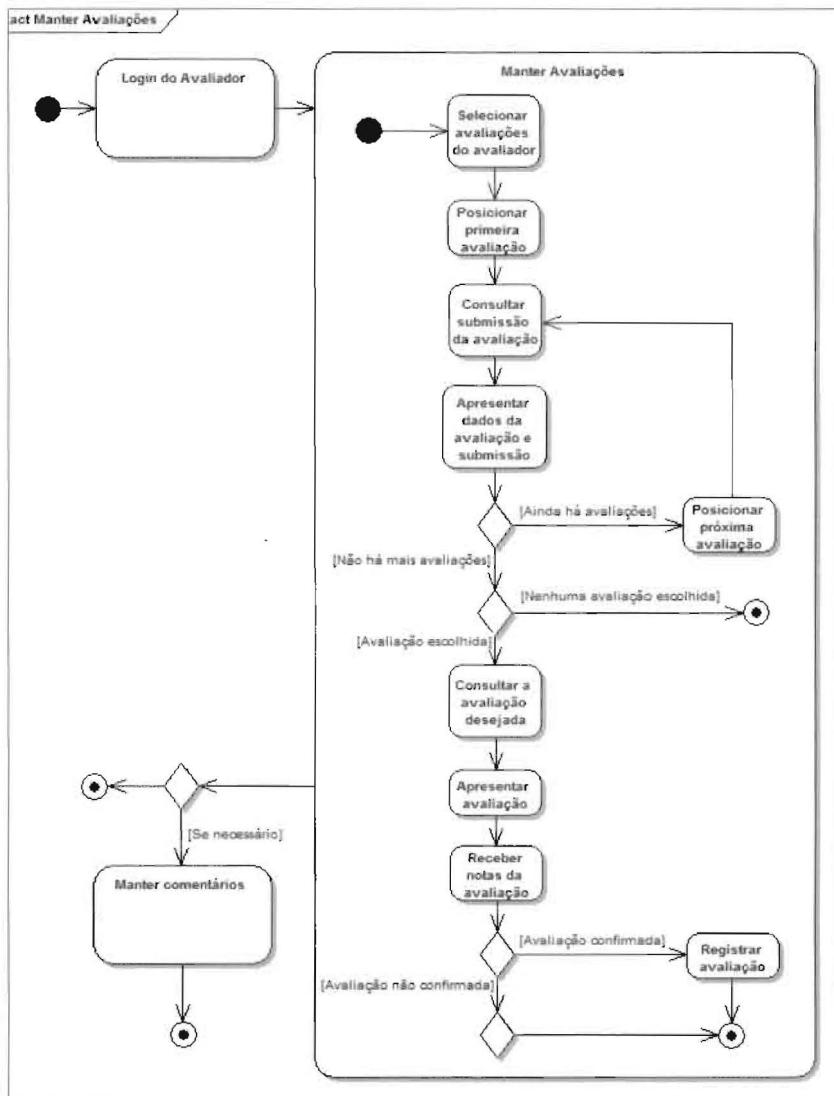


Figura 10.24 – Manter Avaliações.

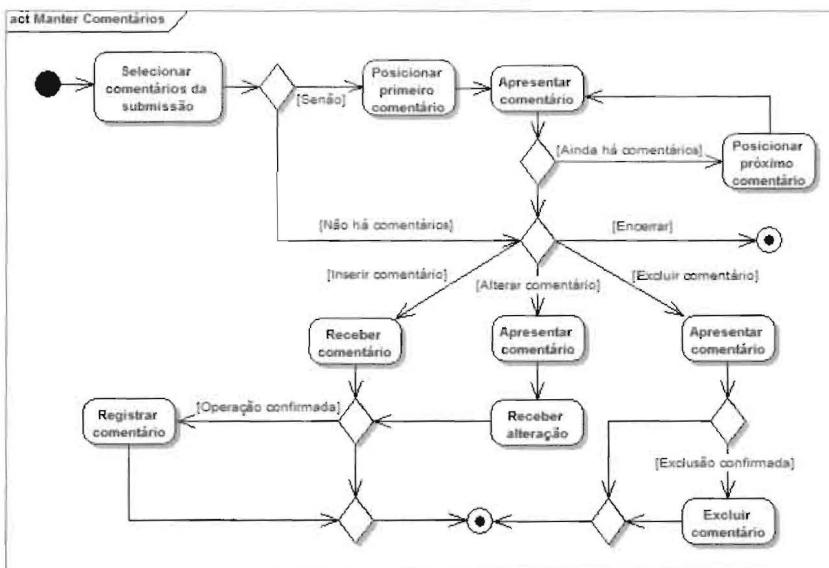


Figura 10.25 – Manter Comentários.

CAPÍTULO 11

Diagrama de Interação Geral

Este diagrama é uma variação do diagrama de atividade. Seu objetivo é fornecer uma visão geral do controle de fluxo. Nele são utilizados quadros no lugar dos nós de ação, embora símbolos como o nó de decisão e o nó inicial sejam também utilizados. Existem basicamente dois tipos de quadros: de interação, que contêm qualquer tipo de diagrama de interação da UML, e de ocorrência de interação, que normalmente fazem uma referência a um diagrama de interação, mas não apresentam seu detalhamento. A Figura 11.1 apresenta um pequeno exemplo de diagrama de interação geral representando o processo geral de submissão de trabalhos referente ao sistema que temos estado modelando neste guia.

Na Figura 11.1 estão representados dois quadros de ocorrência de interação, referentes aos processos de Login do Submissor e Auto-Registro do mesmo, e um quadro de interação referente ao processo de Realizar Submissão, que contém um diagrama de seqüência. Na verdade, os quadros de ocorrência de interação também referem-se a diagramas de seqüência, no entanto optamos por apenas referenciá-los para não tornar o diagrama grande demais, o que é muito comum nesse tipo de diagrama.

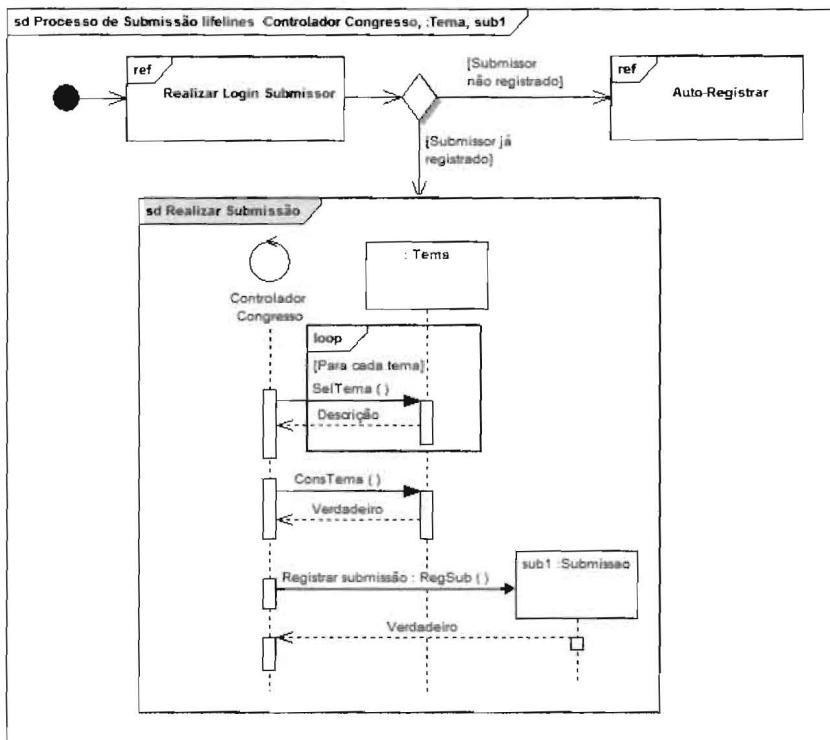


Figura 11.1 – Diagrama de Interação Geral – Processo Geral de Submissão.

Nesse exemplo o fluxo se inicia quando da solicitação de login pelo submissor, que caso não esteja cadastrado deverá se auto-registrar. Caso já esteja registrado o autor poderá então realizar a submissão. Deve-se destacar ainda a palavra “lifelines” (linhas de vida) contida após o título do diagrama. Esta palavra apenas indica os objetos cujas linhas de vida são apresentadas no diagrama – neste caso o objeto Controlador_Congresso, os objetos da classe Tema e o objeto sub1 da classe Submissão. Devemos destacar ainda que apresentamos este diagrama de forma “resumida”, demonstrando apenas os objetos efetivamente necessários à realização da submissão, retirando o ator e a interface contidos no diagrama completo, desnecessários neste diagrama.

CAPÍTULO 12

Diagrama de Componentes

O diagrama de componentes, como seu próprio nome diz, identifica os componentes que fazem parte de um sistema, um subsistema ou mesmo os componentes ou classes internas de um componente individual. Um componente pode representar tanto um componente lógico (um componente de negócio ou de processo) ou um componente físico, como arquivos contendo código-fonte, arquivos de ajuda (help), bibliotecas, arquivos executáveis etc.

O diagrama de componentes pode ser utilizado como uma forma de documentar como estão estruturados os arquivos físicos de um sistema, permitindo assim uma melhor compreensão do mesmo, além de facilitar a reutilização de código. Esse diagrama também pode identificar os componentes utilizados no desenvolvimento de sistemas baseados em componentes.

12.1 Componente

Um componente pode sempre ser considerado uma unidade autônoma dentro de um sistema ou subsistema. Ele tem uma ou mais interfaces fornecidas e requeridas (potencialmente expostas via portas), e seus interiores são transparentes e inacessíveis por outro meio que não seja fornecido por suas interfaces. Embora ele possa ser dependente de outros elementos em termos de interfaces que são requeridas, um componente é encapsulado e suas dependências são projetadas de tal forma que ele possa ser tratado tão independentemente quanto possível.

Na UML 1.x, um componente era representado por um retângulo com dois retângulos menores sobressaindo-se à sua esquerda. A partir da UML 2 esse símbolo foi substituído por um retângulo contendo internamente o antigo símbolo, conforme demonstra a Figura 12.1.

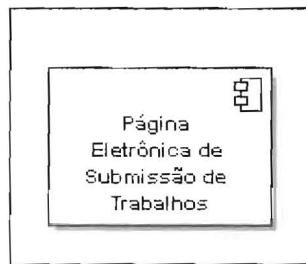


Figura 12.1 – Exemplo de Componente.

12.2 Interfaces Fornecidas e Requeridas

Na UML 1.x, um dos principais relacionamentos entre os componentes era realizado por meio do relacionamento de dependência. No entanto, a partir da UML 2, embora esse tipo de relacionamento continue sendo válido e utilizado, passou-se a utilizar com muito mais freqüência interfaces fornecidas e requeridas, descritas no diagrama de classes, conforme demonstra a Figura 12.2.

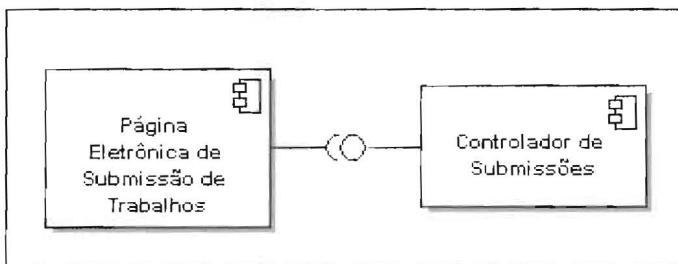


Figura 12.2 – Relacionamento entre Componentes por meio de Interfaces Fornecidas e Requeridas.

12.3 Classes e Componentes Internos

Um componente pode conter ou implementar uma ou mais classes internas, podendo possuir também componentes internos. A representação de um componente sem apresentar seus componentes ou classes internas é chamada de visão de caixa preta; já a representação de um componente com suas classes ou componentes internos é chamada visão de caixa branca. A contenção de classes por um componente significa que este as implementa de alguma forma.

A Figura 12.3 demonstra um exemplo de componente com classes internas. O leitor notará que o componente implementa as classes necessárias para a realização de uma submissão por um autor.

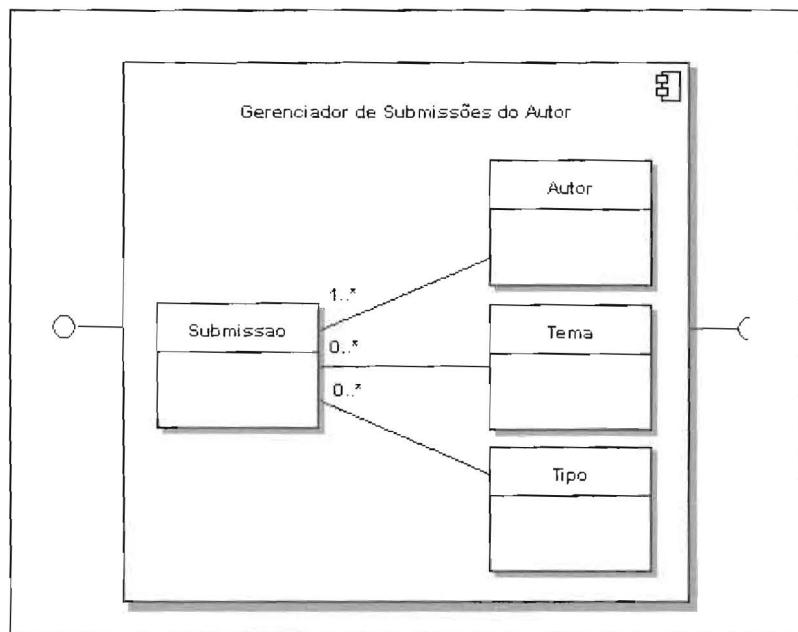


Figura 12.3 – Componente com Classes Internas.

Uma outra forma de representar as classes internas de um componente é por meio do relacionamento de dependência, conforme demonstra a Figura 12.4.

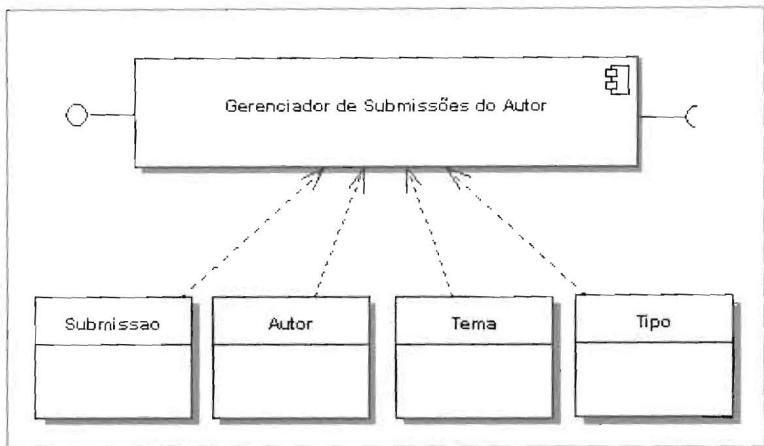


Figura 12.4 – Classes Internas relacionadas a um componente por meio de Dependências.

É comum o uso de portas para comunicar os elementos internos de um componente com o ambiente externo, quando esses solicitam ou executam algum serviço, conforme demonstra a Figura 12.5.

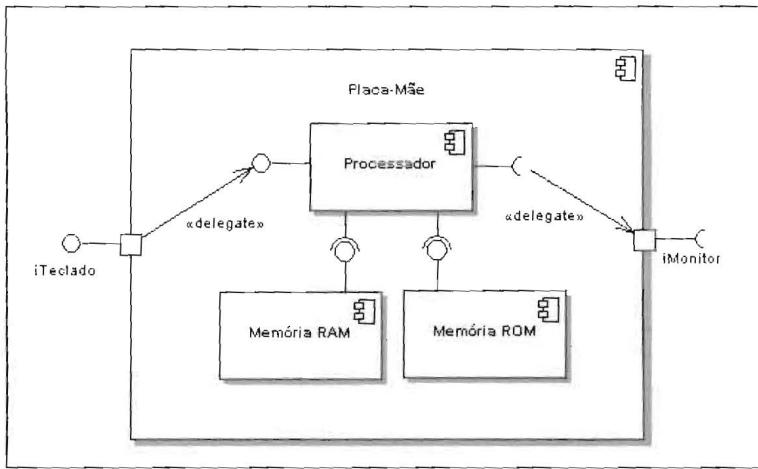


Figura 12.5 – Visão de Caixa Branca de um Componente.

Nesse exemplo representamos uma placa-mãe com seus componentes internos. Observe que o componente Processador solicita serviços dos componentes Memória RAM e Memória ROM, além de se comunicar com o ambiente externo por meio de Portas. Observe o relacionamento de

dependência com o estereótipo <<delegate>>, significando que o serviço atribuído ao componente Placa-Mãe está sendo delegado ao seu componente interno Processador. Essa mesma abordagem pode ser também utilizada com as classes internas de um componente.

12.4 Exemplo de Diagrama de Componentes

A seguir apresentaremos o diagrama de componentes equivalentes aos módulos executáveis do sistema de controle de submissões que estamos modelando ao longo deste guia. Alguns módulos não são exatamente executáveis, como o componente que representa a página de submissão de trabalhos, ou pertencem exclusivamente ao sistema, como o componente que representa o Sistema Gerenciador de Banco de Dados, mas são indispensáveis para o funcionamento do mesmo.

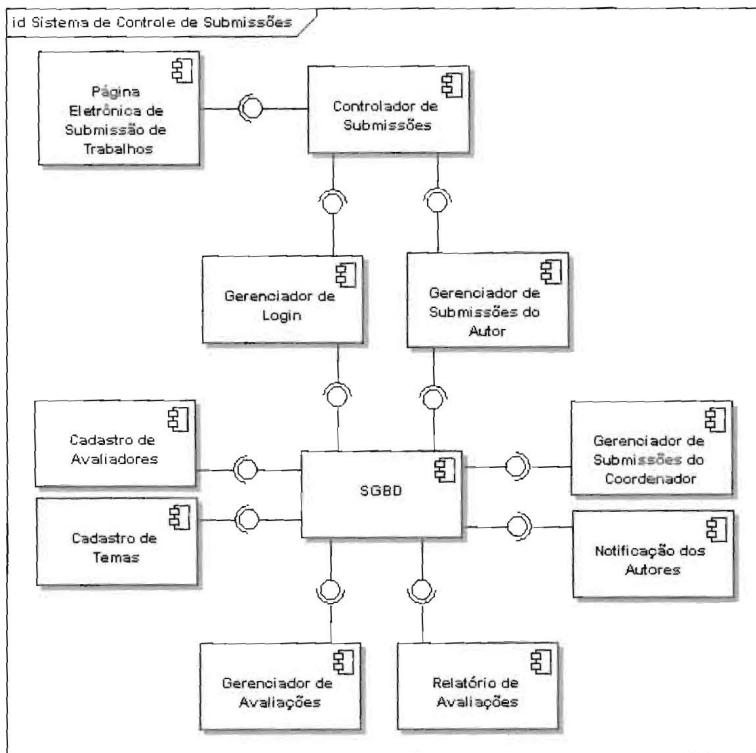


Figura 12.6 – Diagrama de Componentes do Sistema de Controle de Submissões.

CAPÍTULO 13

Diagrama de Implantação

O diagrama de implantação é o diagrama com a visão mais física da UML. Ele enfoca a questão da organização da arquitetura física sobre a qual o software irá ser implantado e executado em termos de hardware, ou seja, as máquinas (computadores pessoais, servidores etc.) que suportarão o sistema, além de definir como essas máquinas estarão conectadas e por meio de quais protocolos se comunicarão e transmitirão informações.

13.1 Nós

Nós são os componentes básicos de um diagrama de implantação. Um nó pode representar um item de hardware, como um servidor em que um ou mais módulos do software são executados ou que armazene arquivos consultados pelos módulos do sistema, ou pode representar um ambiente de execução, ou seja, um ambiente que suporta o sistema de alguma forma. Nós podem conter outros nós, sendo comum encontrar um nó que representa um item de hardware contendo outro nó que representa um ambiente de execução, embora um nó que represente um item de hardware possa conter outros nós representando itens de hardware, e um nó que represente um ambiente de execução possa conter outros ambientes de execução.

Quando um nó representa um hardware, deve possuir o estereótipo <<device>>; quando, porém, um nó representa um ambiente de execução, pode utilizar o estereótipo <<ExecutionEnvironment>>. Exemplos de

ambientes de execução são sistemas operacionais ou sistemas de banco de dados. A Figura 13.1 apresenta um exemplo de nó, representando um item de hardware.



Figura 13.1 – Exemplo de Nô.

13.2 Associação entre Nós

Os nós possuem ligações físicas entre si de forma que possam se comunicar e trocar informações. Essas ligações são chamadas associações e são representadas por retas ligando um nó a outro. Uma associação pode conter estereótipos utilizados para determinar, por exemplo, o tipo de protocolo e comunicação utilizado entre os nós. A Figura 13.2 apresenta um exemplo de associação entre nós.

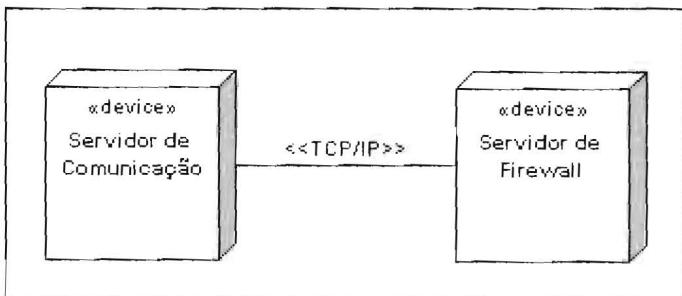


Figura 13.2 – Associação entre Nô.

13.3 Exemplo de Diagrama de Implantação

A seguir apresentaremos o diagrama de implantação referente à arquitetura física necessária para suportar o sistema de controle de submissões modelado neste guia.

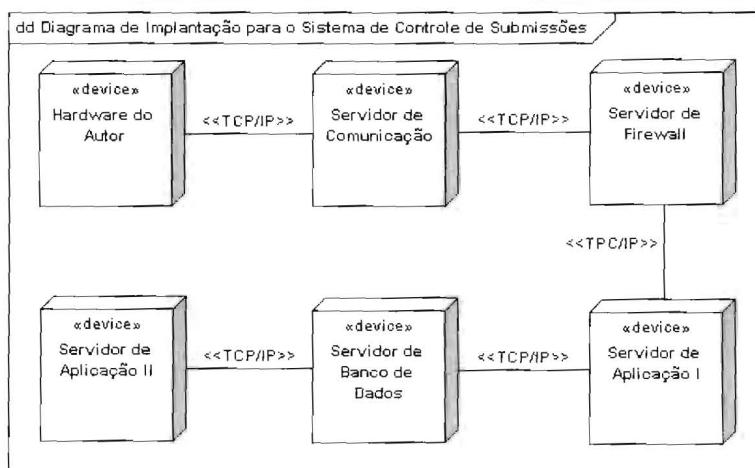


Figura 13.3 – Exemplo de Diagrama de Implantação.

13.4 Artefatos

Um artefato é uma entidade física, um elemento concreto que existe realmente no mundo real, assim como os nós que o suportam. Um artefato pode ser um arquivo fonte, um arquivo executável, um arquivo de ajuda, um documento de texto etc. Um artefato deve estar implementado em um nó. A Figura 13.4 apresenta um exemplo de artefato implementado em um nó.

O leitor notará que o artefato denominado “Módulo Gerenciador de Login” possui a mesma denominação que um dos componentes apresentados no Capítulo 12. Na verdade, um artefato é muitas vezes uma “manifestação” no mundo real de um componente. No entanto, não necessariamente existirá um artefato para cada componente, sendo possível existirem diversos artefatos manifestados a partir de um único componente. A Figura 13.5 apresenta um exemplo de artefato instanciado a partir de um componente. Observe que existe um relacionamento de dependência entre o componente e o artefato, contendo o estereótipo <<manifest>>, significando que o artefato é uma representação do componente no mundo real.



Figura 13.4 – Artefato implementado em um Nó.

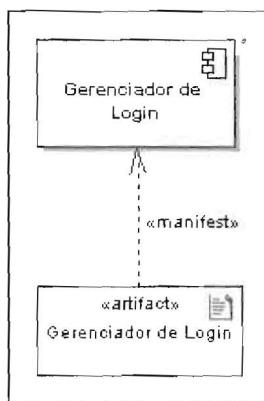


Figura 13.5 – Artefato manifestado a partir de um Componente.

Uma outra forma de demonstrar que um artefato está contido em um nó é também por meio de um relacionamento de dependência, contendo o estereótipo <<deploy>>, entre o nó e os artefatos, conforme demonstra a Figura 13.6.

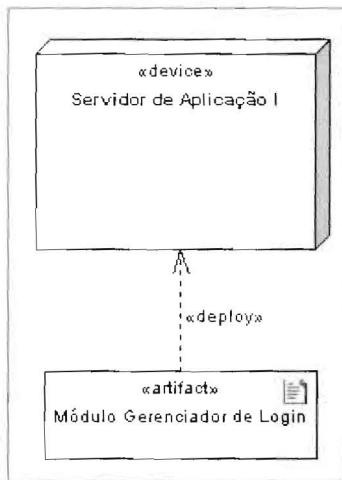


Figura 13.6 – Artefato implementado em um Nô.

Um nô pode conter componentes da mesma forma que artefatos, como uma maneira de demonstrar em que lugar os componentes poderão ser localizados no hardware que suportará o sistema.

13.5 Especificação de Implantação

Especifica um conjunto de propriedades que determinam parâmetros de execução de um artefato implementado em um nô, conforme demonstra a Figura 13.7.

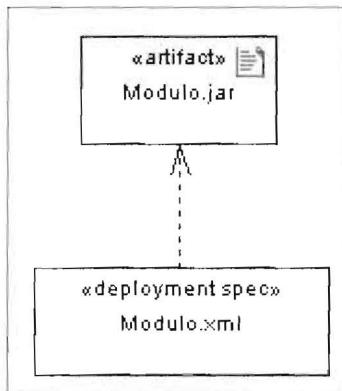


Figura 13.7 – Especificação de Implantação.

CAPÍTULO 14

Diagrama de Pacotes

O diagrama de pacotes tem por objetivo representar os subsistemas ou submódulos englobados por um sistema de forma a determinar as partes que o compõem. Demonstra como os elementos estão organizados nos pacotes e as dependências que existem entre os elementos e os próprios pacotes. Pode ser utilizado de maneira independente ou associado com outros diagramas. Esse diagrama pode ser utilizado também para auxiliar a demonstrar a arquitetura de uma linguagem, como ocorre com a própria UML.

14.1 Pacote

Pacotes são utilizados para agrupar elementos e fornecer denominações para esses grupos. São muito úteis para a modelagem de subsistemas e para a modelagem de subdivisões da arquitetura de uma linguagem. A Figura 14.1 apresenta um exemplo em que é representado o Pacote Núcleo da Biblioteca de Infra-estrutura da UML.

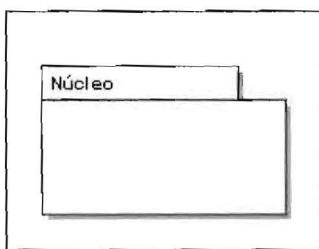


Figura 14.1 – Exemplo de Pacote.

14.2 Dependência

Pacotes normalmente possuem dependências entre si. Um relacionamento de dependência informa que o elemento dependente necessita de alguma forma do elemento do qual depende. A Figura 14.2 apresenta um exemplo de relacionamento de dependência, em que é demonstrada a dependência que o pacote *Perfis* tem em relação ao pacote *Núcleo*.

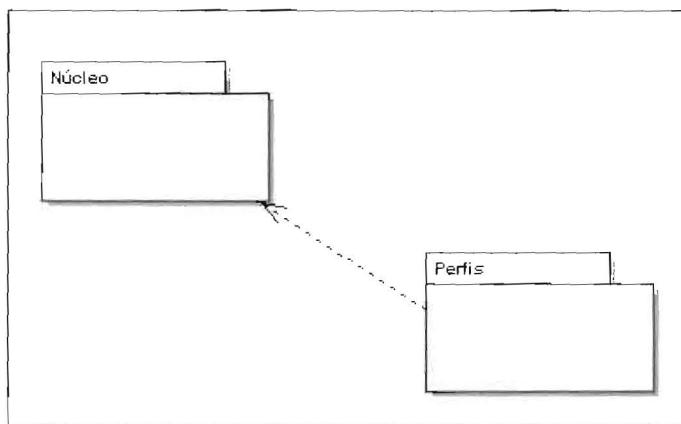


Figura 14.2 – Relacionamento de Dependência entre Pacotes.

O relacionamento de dependência no diagrama de pacotes pode possuir dois estereótipos, <<merge>>, significando que os elementos do pacote que utiliza tal dependência serão unidos aos elementos do outro pacote, e <<import>>, significando que o pacote que utiliza essa dependência está importando alguma característica ou elemento do outro pacote.

14.3 Pacotes Contendo Pacotes

É bastante comum, principalmente na definição de arquiteturas de linguagens de modelagem, que um pacote se subdivida em diversos outros pacotes. Isso ocorre, por exemplo, com a Biblioteca de Infra-estrutura da UML, conforme demonstra a Figura 14.3.

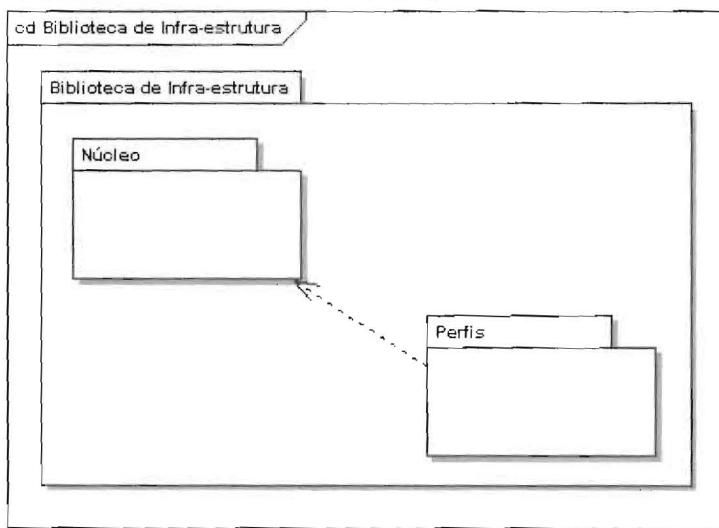


Figura 14.3 – Pacotes Contendo Pacotes.

CAPÍTULO 15

Diagrama de Tempo

Este diagrama apresenta algumas semelhanças com o diagrama de máquina de estados, no entanto, ele enfoca as mudanças de estado de um objeto ao longo do tempo. Esse diagrama terá pouca utilidade para modelar aplicações comerciais, contudo poderá ser utilizado na modelagem de sistemas de tempo real ou sistemas que utilizem recursos de multimídia/hipermídia, nos quais o tempo em que um objeto executa algo é muitas vezes importante. A Figura 15.1 apresenta um exemplo de diagrama de tempo.

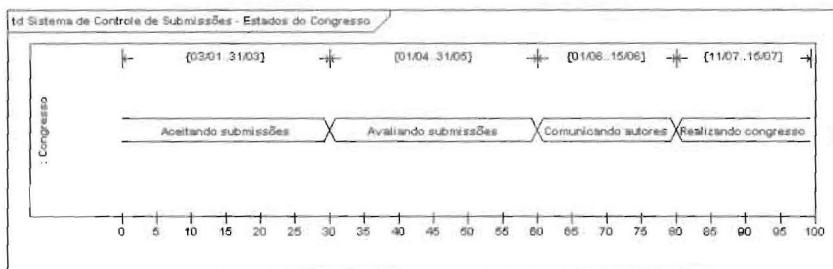


Figura 15.1– Diagrama de Tempo – Forma Concisa.

É importante destacar que o diagrama de tempo possui duas notações ou formas de representação, uma notação chamada concisa, mais simples adotada no exemplo da Figura 15.1 e uma notação chamada robusta, em que as etapas são apresentadas em uma forma semelhante a um gráfico.

No exemplo da Figura 15.1 utilizamos a forma concisa. Nele, descrevemos as etapas por que passa o congresso, para o qual temos estados a modelar o Sistema de Controle de Submissões, desde a etapa de aceitação

de submissões até a realização do congresso. Cada etapa ou estado do objeto da classe Concurso é apresentada por meio de um hexágono, sendo que o primeiro e o último estados se encontram abertos. Acima de cada etapa entre barras verticais se encontram as restrições de duração que determinam o tempo em que transcorrem as etapas. No caso do estado “Aceitando Submissões”, o período vai de 3 de janeiro a 31 de março. A Figura 15.2 apresenta o mesmo diagrama, desta vez, utilizando sua forma robusta, em que as transições de estado são determinadas por mudanças em um gráfico, podendo estas possuir descrições que determinam o evento que causou a mudança, se isso for considerado necessário.

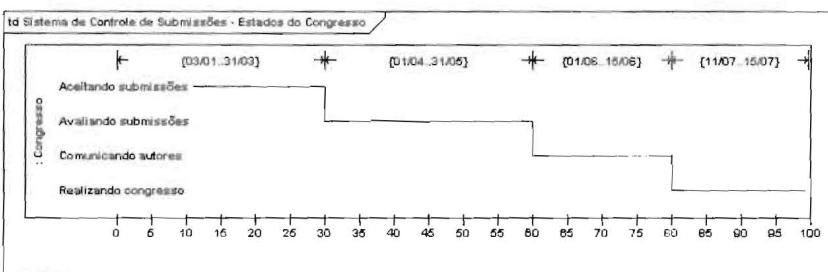


Figura 15.2 – Diagrama de Tempo – Forma Robusta.

A

Abstração 30, 31, 33
Ação
de Evento de Aceitação 143
de Evento de Tempo de Aceitação 144
de Objeto de Envio 143
Agregação 57
Alfinetes 141
ArgoUML 29
Artefatos 163
Assertion 102
Associação 40, 53, 55
Binária 56
Ternária ou N-ária 56
Unária ou Reflexiva 54
Atividades
de estado 120
internas 120
Ator 20, 88
secundário 48
Atores 15, 38, 84
Atributos 32, 33
privado 34
Auto-chamadas 88, 91
Auto-Transições 121

B

Barra de Sincronização 124
Bifurcação 124
Booch 13

C

Camada de Persistência 53
Casos de Uso 15, 20, 22, 39, 41, 84
geral 41, 48
Classes 20, 30, 31, 32, 33, 52, 53, 54
-filhas 59
-mãe 59
Associativas 60, 62
de Objetos 32
e Componentes Internos 158
especializadas 65
gerais 65
Intermediárias 62
Internas 76, 79, 156
Classificações 30, 31
Colaboração 76, 77, 79, 111
Componentes 24, 156
Internos 158, 159
Composição 58

Condições de Guarda 92, 96, 122
Conectores 77, 138

D

Dependência 59, 64, 167
Diagrama
Comportamentais 27
de Interação 27
Estruturais 27
de Atividade 22, 133
de Casos de Uso 14, 15, 38
de Classes 17, 20, 52
de Componentes 24, 25, 156
de Comunicação 21, 111
de Estados 22
de Estrutura Composta 19, 76
de Gráfico de Estados 22
de Implantação 25, 161
de Interação Geral 23, 24, 154
de Máquina de Estados 22, 118
de Objetos 18, 73, 74
de Pacotes 26, 166
de Seqüência 20, 84
de Tempo 27, 169
Documentação de Casos de Uso 46

E

Edição para a comunidade 28
Enterprise Architect 12, 29
Especialização/Generalização 40, 41, 59
Especificação de Implantação 165
Estado 118
Composto 126
de Entrada e Saída 128
de História 126
Profunda 127
de Ponto de Escolha Dinâmico 122
de Sincronismo 129
de Submáquina 127
Inicial e Final 119
paralelos 124
Estereótipos 67
Esteriotípos
<<boundary>> 68, 69, 71, 86
<<control>> 68, 69, 71, 86
<<datastore>> 145
<<delegate>> 160
<<deploy>> 164
<<device>> 161

<<entity>> 67
 <<ExecutionEnvironment>> 161
 <<extend>> 43
 <<import>> 167
 <<include>> 42
 <<interface>> 60
 <<manifest>> 163
 <<merge>> 167
 <<occurrence>> 79
 <<represents>> 79
 Estrutura interna 76, 78, 81
 Estudo de Caso 12, 48, 69, 83, 103, 112, 130,
 149
 Exceções 142
 Exemplo
 de Diagrama
 de Atividades 136
 de Componentes 160
 de Implantação 162
 Extensão 43

F

Ferramentas CASE 28
 Final de Fluxo 140
 Fluxo
 concorrentes 140
 de controle 22, 133, 134, 140, 143, 149
 de interrupção 142
 de objeto 22, 143
 de objetos 134, 141
 de valores 141
 Fluxo de Objetos 141
 Foco de Controle ou Ativação 87
 Fragmento de Interação 92, 93, 95
 Fronteira de Sistema 46

G

Generalização 40

H

Herança 35, 36

I

Inclusão 42
 Instância 31, 32, 77, 79, 81, 82
 Instanciação 30, 31
 Interface Fornecida e Requerida 63
 Interfaces 62
 Fornecidas 62, 63, 157
 Requeridas 63, 157

J

Jacobson 13

L

Levantamento e análise dos requisitos 15
 Linha de Vida 84, 86, 89

M

Máquina de Estado 22
 Mensagens 20, 87, 88, 111
 de retorno 90
 ou Estímulos 87
 Método 32, 33, 36, 37
 construtor 89
 destrutor 89
 Unificado 14
 Modelagem
 de Colaboração 52
 de Vocabulário 52
 Modelo de Colaboração 52
 Multiplicidade 54, 56
 no Diagrama de Casos de Uso 45

N

Navegabilidade 40, 53, 56
 Nó 161
 de ação 134
 de Bifurcação/União 140
 de buffer central 145
 de contrrole 135
 de Decisão 135
 de entrada de exceção 143
 de expansão 147
 de objeto 141, 142, 143
 de Parâmetro de Atividade 142
 de Repositório de Dados 145
 Final 135
 Inicial 135
 Nota explicativa 44

O

Objeto 18, 20, 30, 31, 32, 33, 73, 74, 85, 86,
 87, 89
 -parte 57, 58
 -todo 57, 58
 Ocorrência
 de colaboração 79
 de interação 93, 94, 154
 Ocorrências
 de interação 92

OMG 14
OMT 13
OOSE 13
Operador de interação 98
Alt 96
Assertion 102
Break 100
Consider 102
Critical Region 101
Ignore 102
Loop 99
Neg 102
Opt 97
Par 98
Seq 102
Strict 103
Orientação a Objetos 30
Ou Exclusivo 65

P

Pacote 166
contendo Pacotes 167
Partição de Atividade 145
Polimorfismo 36
Pontos de Extensão 45
Portas 80
Portões 94
Poseidon for UML 28, 29
Propriedades 81
Pseudo-Estado
de Escolha 122
de Junção 125
de Término 128

Q

Quadro
de interação 154
de ocorrência 154

R

Rational Rose 28
Realização 60, 64
Redes de Petri 22, 133
Região
de Atividade Interrompível 146
de Expansão 147
Regras do negócio 48
Relacionamentos ou Associações 53
Restrições 64, 65
Completa 65

disjunta 65
em Associações de Extensão 44
Incompleta 65
Separada 65
Sobreposta 66
Rumbaugh 13

S

Separador de operando de interação 97
Sinal 134
Síntese Geral dos Diagramas 27
Subatividade 139
Subclasse 35, 36
Subestados 126
Superclasse 35, 36

T

Transições 119
concorrentes 124
de Estados 122
internas 121

U

União 124
Upperbound 141

V

Vínculos 74
entre Objetos 75
Visão
caixa preta 158
de caixa branca 158, 159
Visibilidade 34
privada 34
protégida 34
pública 34
Visões do sistema 14
Visual Paradigm for UML 28
VP-UML 28

X

Xor 65

