Projeto de Software

UNIDADE 3

FASE DE ELABORAÇÃO
Disciplina de Requisitos: Modelos Conceituais

Autores:

Profa. Dra Rosana Terezinha Vaccare Braga
Prof. Dr. Paulo Cesar Masiero

ICMC/USP

Professor da Disciplina:
Prof. Dr. Valter Camargo
Prof. Esp. Alexandre Pedroso Fernandes

Projeto de Software

1 - Modelo Conceitual

1.1 – Conceitos e atributos

Conforme visto na Unidade 1, o objetivo da disciplina de requisitos é produzir modelos que representem o sistema a ser desenvolvido. O Modelo Conceitual têm por objetivo mostrar todos os **CONCEITOS** importantes no domínio do sistema, bem como as associações entre esses conceitos. A idéia é fazer com que o usuário que tem acesso a esse modelo ENTENDA OS PRINCIPAIS ELEMENTOS DO DOMÍNIO QUE ESTÃO ENVOLVIDOS NO SISTEMA A SER DESENVOLVIDO.

O modelo conceitual fornece um panorama geral do sistema, podendo servir como um ponto inicial para começar a entender o escopo do sistema e seu funcionamento global. Assim, alguém que não conhece o domínio do sistema pode utilizar o modelo conceitual para ter uma idéia dos principais conceitos e seus relacionamentos. 1.1.1 – Como identificar CONCEITOS A identificação de conceitos do domínio do sistema não é uma tarefa trivial, principalmente para novatos em análise orientada a objetos. Alguns conceitos são bastante óbvios quando se descreve o sistema, como os conceitos Leitor e Livro quando se trata de um sistema de Biblioteca. Entretanto, muitos conceitos estão geralmente implícitos e requerem uma maio maturidade do analista para descobrí-los. Por exemplo, o conceito de Empréstimo, no mesmo sistema de Biblioteca, poderia não ser modelado como um conceito, mas sim como uma associação entre Leitor e Livro. Isso não é totalmente errado, mas as informações sobre o empréstimo em si ficam difíceis de representar por meio de uma simples associação. Por isso, existem algumas sugestões básicas para a correta identificação dos conceitos do sistema.

Passo 1: Comece isolando, no documento de requisitos ou na descrição dos casos de uso elaborados na fase anterior, **todos os substantivos** presentes no texto. Eles são candidatos a conceitos. Por exemplo, analisando o cenário de sucesso principal, pode-se isolar diversos substantivos, conforme mostrado na Figura 1.

1. O Leitor chega ao balcão de atendimento da biblioteca e diz ao atendente que deseja emprestar um ou mais livros da biblioteca.
2. O Atendente seleciona a opção para adicionar um novo empréstimo.
3. O Atendente solicita ao leitor sua carteirinha, seja de estudante ou professor.
4. O Atendente informa ao sistema a identificação do leitor.
5. O Sistema exibe o nome do leitor e sua situação.
6. O Atendente solicita os livros a serem emprestados.
7. Para cada um deles, informa ao sistema o código de identificação do livro.
8. O Sistema informa a data de devolução de cada livro.
9. O Atendente desbloqueia os livros para que possam sair da biblioteca.
10. O Leitor sai com os livros.

Figura 1 – Trecho de um caso de uso com os substantivos sublinhados.

Passo 2: Considere **cada um dos substantivos isoladamente** e verifique se são relacionados a assuntos importantes no domínio do sistema. Muitos deles podem ser descartados simplesmente porque fogem do escopo do sistema, porque são similares a outros conceitos já identificados, ou porque são meramente propriedades de outros conceitos. Por exemplo, na Figura 1, podemos descartar balcão e opção pelo primeiro motivo e identificação do leitor, nome do leitor, situação, código de identificação do livro e data de devolução pelo terceiro motivo. Carteirinha pode ser descartada por ser um mero instrumento para identificar o leitor, portanto o interesse na carteirinha se deve ao código que ela possui e que identifica o leitor.

Passo 3: Isole agora os **verbos** que poderiam ser transformados em substantivos (possivelmente com a ajuda de outras palavras). Concentre-se nos verbos que representam ações de interesse para o sistema, ou seja, aqueles relacionados a eventos e transações que possuem informações importantes e que devem ser lembrados pelo sistema. Por exemplo, na Figura 2 há vários verbos sublinhados, mas somente um deles, “emprestar”, é candidato a conceito no sistema de Biblioteca. Como a palavra empréstimo já havia sido sublinhada no Passo 1, a adição desse verbo não representa nenhum novo conceito nesse exemplo.

1. O Leitor chega ao balcão de atendimento da biblioteca e diz ao atendente que deseja emprestar um ou mais livros da biblioteca.
2. O Atendente seleciona a opção para adicionar um novo empréstimo.
3. O Atendente solicita ao leitor sua carteirinha, seja de estudante ou professor.
4. O Atendente informa ao sistema a identificação do leitor.
5. O Sistema exibe o nome do leitor e sua situação.
6. O Atendente solicita os livros a serem emprestados.
7. Para cada um deles, informa ao sistema o código de identificação do livro.
8. O Sistema informa a data de devolução de cada livro.
9. O Atendente desbloqueia os livros para que possam sair da biblioteca.
10. O Leitor sai com os livros.

Figura 2 – Trecho de um caso de uso com os verbos sublinhados.

Passo 4: Para cada candidato a conceito, verifique se ele é composto de outras partes que sejam de interesse para o sistema, mesmo que elas não apareçam explicitamente no texto. Ou seja, nesse passo deve-se fazer um **refinamento dos conceitos**. Por exemplo, um empréstimo normalmente refere-se a vários livros emprestados em uma mesma ocasião para um mesmo leitor. Assim, pode-se dizer que o empréstimo é dividido em vários itens de empréstimo, um para cada livro emprestado. Portanto, ItemDoEmpréstimo é um outro conceito. Outro exemplo: Um conserto de carro engloba tanto os serviços de mão-de-obra executados quanto a venda das peças substituídas para realizar o conserto. Portanto ServiçoExecutado e PeçaVendida também são conceitos. Mesmo seguindo os passos acima, alguns conceitos poderão vir a ser descobertos só mais à frente, na etapa de projeto. Isso ocorre porque muitos conceitos são difíceis de serem descobertos usando os documentos existentes até então, mas que são mais facilmente identificados durante o projeto da colaboração entre os objetos para cumprir as responsabilidades necessárias ao funcionamento do sistema.

Algo muito importante de ser observado é que um diagrama CONCEITUAL é independente do paradigma de programação. Isso quer dizer que, embora um diagrama de classes da UML esteja sendo usado para a modelagem do diagrama conceitual, cada retângulo que se vê no diagrama (classe) é um CONCEITO e não uma classe no sentido de "classe" de uma linguagem de programação. Esta é uma diferença muito importante e deve ser muito bem entendida. O que estamos fazendo aqui é usar o diagrama de classes da UML como FERRAMENTA para a criação de um MODELO CONCEITUAL. Isto é, poderíamos usar, na verdade, qualquer outra forma de representação para o modelo conceitual, já que o objetivo é representar apenas os conceitos do domínio. Na próxima unidade, veremos que esse mesmo diagrama de classes será usado para representar um MODELO DE CLASSES DE PROJETO, em que cada classe realmente representa uma classe no sentido clássico de uma linguagem de programação.

1.1.2 – Como identificar Atributos

Conforme mencionado no Passo 1, vários dos substantivos sublinhados a partir do texto do caso de uso podem ser candidatos a atributos dos conceitos já identificados. É necessária certa cautela ao incluir os atributos, para não tornar o modelo conceitual muito complexo desnecessariamente. Portanto, limite-se a adicionar os atributos importantes para compreender o real sentido de cada conceito, ou atributos que serão importantes para o futuro projeto do sistema, por se referirem às propriedades do conceito que tem papel definido no cumprimento das funcionalidades.

Por exemplo, os atributos identificação do leitor, nome do leitor, situação, código de identificação do livro e data de devolução devem ser adicionados aos respectivos conceitos, pois serão utilizados de alguma forma no projeto da parte lógica do sistema.

1.1.3 – Representação na UML

Conceitos são também chamados de classes conceituais do sistema e, na UML, utiliza-se a mesma notação do diagrama de classes, que será visto mais adiante neste curso. Na Figura 3, é ilustrada a notação para Conceitos em UML, referente aos exemplos mencionados nas sub-seções anteriores: um retângulo com uma divisória, na qual a parte superior contém o nome do conceito e a parte inferior os atributos do conceito.

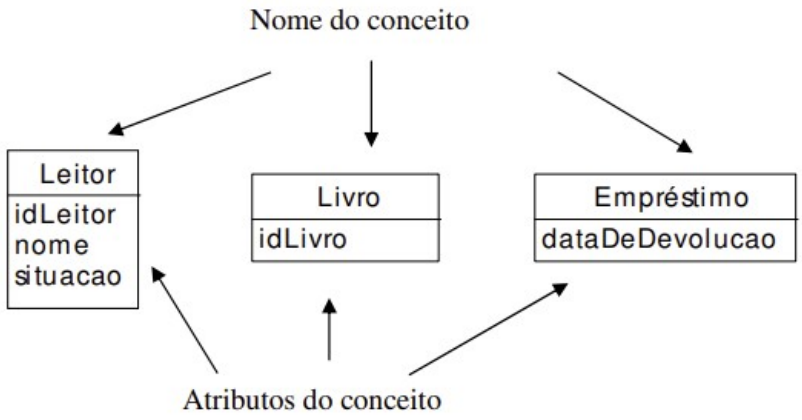

Figura 3 – Conceitos em UML.

1.2 – Associações

Conceitos são importantes para entender o domínio do sistema. Analisar os relacionamentos entre conceitos do sistema é de grande ajuda para melhor compreender as conexões existentes entre as partes que compõem o sistema. Uma associação pode ser definida como um relacionamento entre conceitos, que precisa ser lembrado pelo sistema durante seu funcionamento. Por exemplo, existe uma associação entre Empréstimo e Leitor, que precisa ser lembrada pelo sistema, pois quando o Empréstimo termina, o Leitor e o Livro devem ter sua situação regularizada.

1.2.1 – Como identificar associações

Assim como os conceitos, existem associações que são bastante simples de serem identificadas, bastando aplicar certas regras básicas. Outras associações são implícitas e podem requerer mais experiência do analista para identificá-las logo na disciplina de requisitos, ou podem surgir mais adiante, na disciplina de projeto. Da mesma forma que introduzir muitos atributos pode ser prejudicial para o entendimento do modelo conceitual, incluir associações em demasia também causa um efeito indesejado, levando a um modelo confuso e conseqüentemente com pouca legibilidade. Portanto, uma regra básica é evitar incluir associações redundantes ou que podem ser derivadas a partir de outras.

Três regras que podem ajudar na identificação de associações são:

Regra 1: Um conceito que, fisicamente ou logicamente, faz parte de outro. Por exemplo, um livro que está fisicamente armazenado em uma estante ou um ItemDeEmpréstimo que logicamente faz parte do Empréstimo.

Regra 2: Um conceito que serve para descrever ou qualificar outro conceito. Por exemplo, um Livro pode ser classificado em diversas Categorias ou por Autor; um Item de Estoque que é descrito por uma Especificação de Produto.

Regra 3: Um conceito que é responsável por registrar ou manter informações sobre outro. Por exemplo, o Atendente é quem registra e atende o Leitor; a Bibliotecária é responsável pelos Livros.

Regra 4: Os verbos sublinhados nos casos de uso (Passo 3) podem indicar associações entre conceitos (por exemplo, o Atendente é quem empresta o Livro ao Leitor).

1.2.2 – Representação na UML

Nas Figuras 4 e 5 é ilustrada a notação UML para associação entre dois conceitos. Uma linha contínua liga os dois conceitos. Uma etiqueta com o nome da associação é colocada sobre ou sob a linha (de preferência de maneira centralizada). Nos extremos de cada lado da associação é colocada a multiplicidade da associação. A direção de leitura de uma associação é da esquerda para a direita e de cima para baixo. Por exemplo, na Figura 4 lêse: Atendente registra Leitor. Caso o nome da associação tenha sido escrito pensando em um sentido diferente do tradicional, deve-se colocar um pequeno triângulo para denotar que a leitura é num sentido não convencional, conforme mostrado na Figura 5, onde lê-se: Leitor é registrado por Atendente.

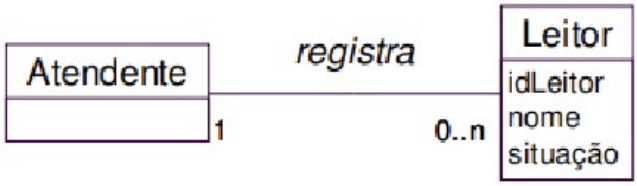

Figura 4 – Associação em UML.

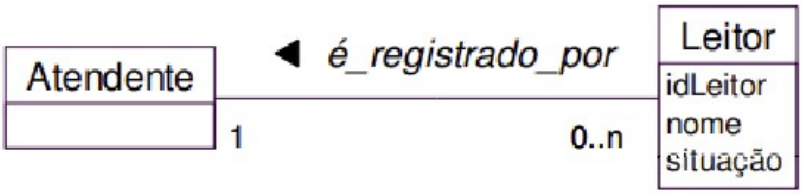

Figura 5 – Associação em UML (leitura no sentido contrário).

1.2.3 – Multiplicidade

Ao definir uma associação, é importante saber quantos objetos de uma classe C1 estão (ou podem estar) associados a quantos objetos de uma classe C2. Esse conceito é conhecido como **multiplicidade** da associação, e é denotado por meio de uma anotação em cada um dos lados da associação.

Por exemplo, nas Figuras 4 e 5, próximo ao conceito Atendente encontra-se o valor 1 e próximo ao conceito Leitor encontra-se o valor 0..n. O significado desses valores pode ser encontrado na Figura 6. A leitura da multiplicidade é feita sempre considerando um (1) objeto do lado oposto à multiplicidade analisada e contrapondo esse objeto ao número de objetos que podem existir relacionados a ele em um dado momento. Por exemplo, na Figura 4 lendo-se a multiplicidade da esquerda para a direita tem-se “Um Atendente registra 0 (zero) ou mais leitores” e lendo-se da direita para a esquerda tem-se “um Leitor é registrado por um Atendente”. Pode-se também fazer a leitura do ponto de vista de objetos, ou seja, um objeto Atendente registra vários objetos Leitor e um objeto Leitor é registrado por um objeto Atendente. Com isso, consegue-se restringir os tipos de associações que podem ser estabelecidas entre os objetos de cada conceito.

É importante ressaltar que algumas ferramentas CASE, como é o caso da versão mais antiga da IBM Rational Rose, não utilizam o “n” para denotar muitos objetos, mas o asterisco “*”. A UML é flexível neste ponto, deixando para o usuário as duas alternativas.

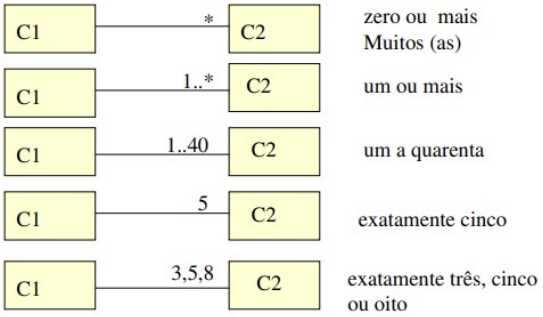

Figura 6 – Multiplicidades em UML.

1.2.4 – Associação Reflexiva

É comum precisarmos denotar a associação entre objetos de uma mesma classe, o que é chamado de associação reflexiva. Isso pode ser feito com uma linha ligando o objeto a ele mesmo, como mostrado na Figura 7, que mostra que um objeto da classe Pessoa, pode ter uma associação de paternidade com zero ou mais pessoas dessa mesma classe. Uma anotação especial é colocada juntamente com a multiplicidade, chamada de **papel**, que denota o papel desempenhado pelo objeto de cada lado da associação. No exemplo da Figura 7, um objeto da classe Pessoa, com o papel de “pai”, é pai de zero ou mais objetos dessa mesma classe, que desempenham o papel de “filhos”.

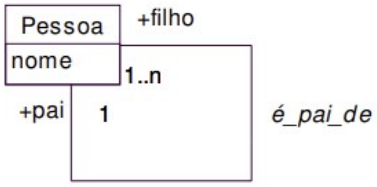

Figura 7 – Associação Reflexiva.

Uma última observação é que o conceito de papéis pode ser utilizado normalmente em associações não reflexivas, se for importante estabelecer o papel de cada objeto na classe origem e destino da associação, como no exemplo da Figura 8.

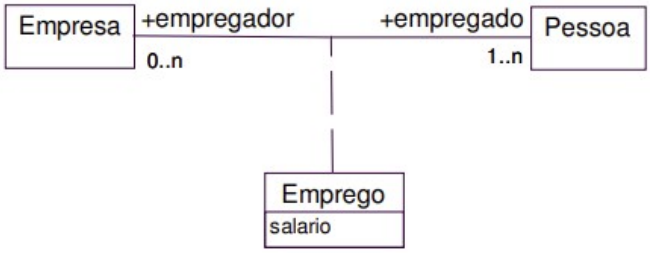

Figura 8 – Tipo Associativo.

1.3 – Tipo Associativo ou Associação

Um tipo associativo é uma associação que também possui propriedades de classe (ou uma classe que tem propriedades de uma associação). É mostrada como uma classe, ligada por uma linha tracejada a uma associação, mostrado também na Figura 8, na qual o emprego que uma pessoa possui em uma dada empresa é modelado como um tipo associativo. O emprego poderia ter sido modelado simplesmente pela associação entre empresa e Pessoa, que é do tipo “muitos para muitos”, já que uma empresa emprega várias pessoas e uma pessoa pode ter empregos em diversas empresas.

Alguns indícios de que um tipo associativo pode ser útil em um modelo conceitual são:

- um atributo está relacionado com uma associação, ou seja, o atributo não é inerente a nenhum dos conceitos nas extremidades da associação, mas parece pertencer à associação em si. Por exemplo, se não houvesse o tipo associativo Emprego na Figura 8, seria difícil decidir onde colocar o atributo “salário”, pois como uma pessoa tem vários empregos, o salário é diferente em cada um deles. Além disso, também não se pode colocá-lo na empresa, pois o salário é diferente para cada funcionário.

- as instâncias do tipo associativo têm um tempo de vida dependente do tempo de vida da associação. Por exemplo, na Figura 8, um emprego só faz sentido entre Empresa e Pessoa associadas.

- existe uma associação muitos-para-muitos entre dois conceitos, bem como informações relacionadas à associação propriamente dita, que é o caso do exemplo da Figura 8. Ao invés de usar o tipo associativo, poderia ter sido criado o conceito Emprego, removendo-se a associação entre Empresa e Pessoa e criando duas associações um-para-muitos, uma entre Empresa e Emprego e outra entre Pessoa e Emprego, conforme mostrado na Figura 9.

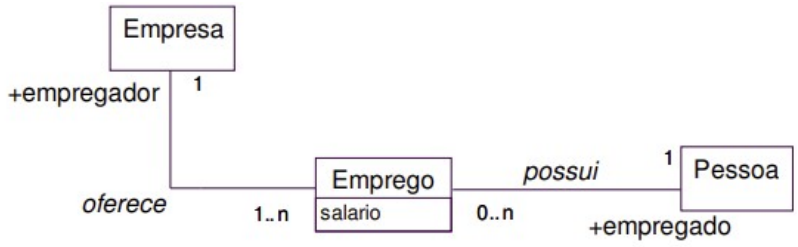

Figura 9 – Alternativa ao Tipo Associativo.

1.4 – Herança

Herança é um mecanismo que permite que características comuns a diversas classes sejam colocadas em uma classe base, ou superclasse. A partir de uma classe base, outras classes podem ser especificadas (as subclasses). Cada subclasse apresenta as características (estrutura e métodos) da superclasse e acrescenta a elas novas características ou comportamento. Dizemos que uma subclasse **herda** todas as propriedades da superclasse e acrescenta suas características próprias e exclusivas. As propriedades da superclasse não precisam ser repetidas em cada subclasse. Por exemplo, quando pensamos em um forno, logo nos vêm à mente sua utilidade geral, que é cozinhar alimentos. Várias subclasses de Forno adicionam características próprias aos diversos tipos de forno, como microondas, a gás, elétrico e a lenha, conforme ilustrado na Figura 10, já utilizando a notação da UML, que é de um triângulo não preenchido ligando a superclasse às suas subclasses. Podemos criar diversos níveis de hierarquia de herança. Por exemplo, fornos a gás podem ser de diversos tipos, como com acendimento elétrico, com fósforo e para camping.

Em termos de Modelo Conceitual, chamamos as superclasses de **tipos** e as subclasses de **subtipos**. É importante identificar a herança durante a análise do sistema, embora, durante o projeto, seja possível identificar outros casos que podem ter passado despercebidos durante a análise.

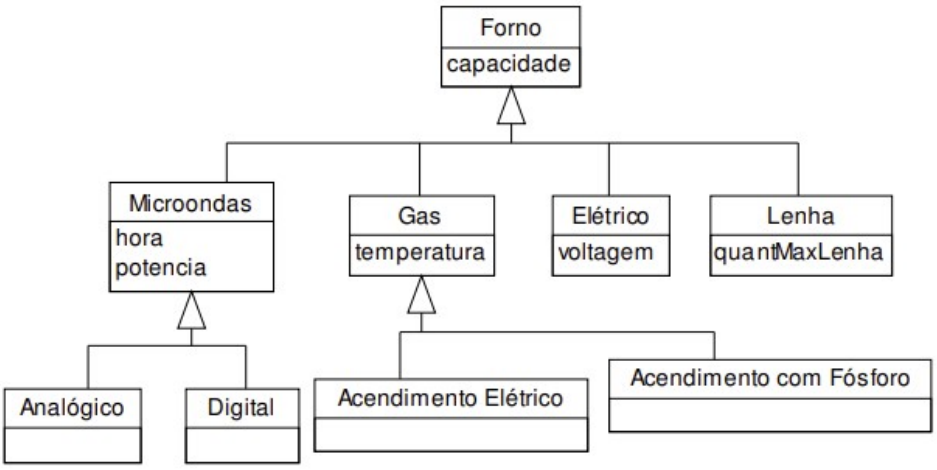

Figura 10 – Exemplo de Herança.

Para ter certeza de que um conceito ou classe pode ser representado usando herança, deve-se ter em mente duas regras importantes: a regra “é-um” e a regra dos 100%.

A regra “é-um” estabelece que todos os membros do conjunto de um subtipo devem ser membros do conjunto do supertipo, ou seja, o subtipo é um supertipo. Por exemplo, podemos dizer que um microondas digital **É UM** forno, e por isso possui todos os atributos e comportamentos esperados de um forno.

A regra dos 100% estabelece que 100% da definição do supertipo dever ser aplicado ao subtipo, ou seja, o subtipo deve estar conforme com 100% dos elementos do supertipo, que são seus atributos e associações. Assim, se a superclasse possui um dado atributo, é necessário garantir que

todas as subclasses também o possuam, ou seja, ele deve fazer sentido para elas e deve ser importante conhecer seu conteúdo. O mesmo vale para as associações da superclasse, que devem ser aplicáveis às subclasses. Por exemplo, se incluirmos a classe Cozinheiro no modelo da Figura 10 e associarmos essa classe à classe forno, significando que o Cozinheiro utiliza o Forno, isso deve ser verdadeiro para todas as subclasses, ou seja, **TODOS (100%)** os tipos de forno devem poder ser utilizáveis por um cozinheiro.

1.5. – Agregação

Agregação é um mecanismo pelo qual um objeto inclui atributos e comportamento de outros objetos a ele agregados, indicando a existência de um todo, composto por partes. Um exemplo de agregação é um carro: consiste em 4 rodas, um motor, chassis, caixa de câmbio, e assim por diante. Quando uma agregação é usada, ela geralmente descreve vários níveis de abstração. As perguntas básicas para identificar a agregação seriam: o objeto "é composto de" outros objetos? O objeto "consiste em" outros objetos? O objeto "contém" outros objetos? O objeto "é parte de" outro objeto? A Figura 11 ilustra a agregação no caso da cozinha: o fogão é o todo, e poderia ser composto de partes como os queimadores (de um a seis), o forno e a estufa. A representação em UML é de um losango ligando a classe que representa o TODO às classes que representam as PARTES.

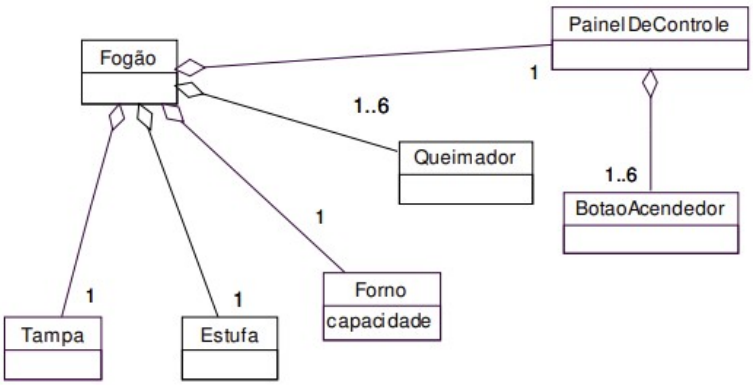

Figura 11 - Exemplo de Agregação.

Há dois tipos possíveis de agregação: a agregação composta (composição) e a agregação compartilhada. A agregação composta ou composição ocorre quando a multiplicidade na extremidade do composto pode ser no máximo 1. A notação para isso em UML é um losango negro. Por exemplo, na Figura 12 um automóvel é composto de quatro rodas. O losango negro significa que a roda pertence à no máximo um automóvel.

Figura 12 – Exemplo de Agregação composta.

Já a **agregação compartilhada**, denotada em UML por um losango vazio, denota que a multiplicidade na extremidade do composto pode ser maior do que um. Por exemplo, a Figura 13 ilustra o caso de pacotes na UML, que são estruturas que agregam vários outros elementos da UML. No entanto, um dado elemento que pertence a um pacote específico pode simultaneamente pertencer a outros pacotes diferentes. Este é um exemplo de agregação lógica. Já na Figura 14 mostra-se um exemplo de agregação física, entre CD e Música.

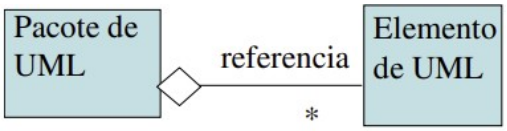

Figura 13 – Exemplo de Agregação compartilhada.

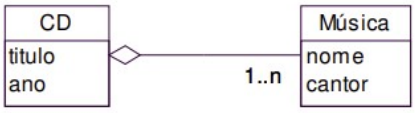

Figura 14 – Exemplo de Agregação compartilhada.

1.6 – Um exemplo

Para melhor ilustrar os modelos conceituais, a Figura 15 contém um Modelo Conceitual de uma biblioteca (versão simplificada).

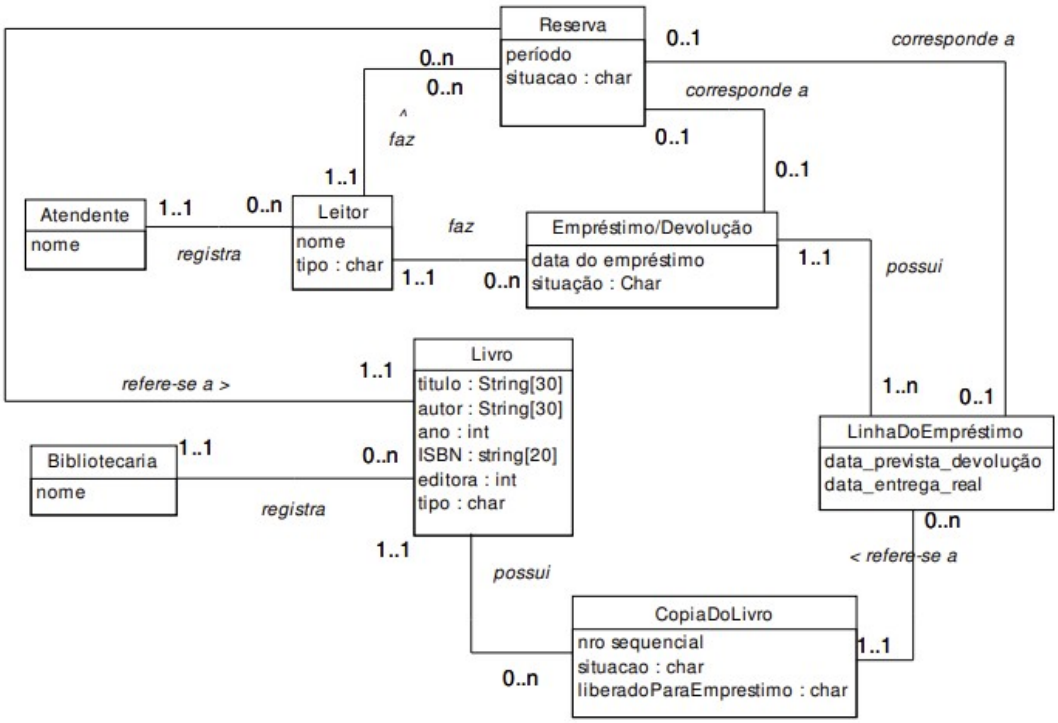

Figura 15 – Modelo Conceitual para Biblioteca.

Referências

[Alexander 77] Christopher Alexander et. al., A Pattern Language, Oxford University Press, New York, 1977.
[Alexander 79] Christopher Alexander, The Timeless Way of Building, Oxford University Press, New York, 1979.
[Appleton 97] Appleton, Brad. Patterns and Software: Essential Concepts and Terminology, disponível na WWW na URL: http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html
[Bauer, 2004] Bauer, Christian; King, Gavin. Hibernate in Action, Manning Publications.
[Beck 87] Beck, Kent; Cunningham, Ward. Using Pattern Languages for Object-Oriented Programs, Technical Report nº CR-87-43, 1987, disponível na WWW na URL: http://c2.com/doc/oopsla87.html
[Booch, 1995] Object Solutions : Managing the Object-Oriented Project (Addison-Wesley Object Technology Series by Grady Booch , Pearson Education; 1st edition (October 12, 1995)
[Buschmann 96] Buschmann, F. et at. A System of Patterns, Wiley, 1996. [Coad 92] Coad, Peter. Object-Oriented Patterns. Communications of the ACM, V. 35, nº9, p. 152-159, setembro 1992.
[Coad 95] Coad, P.; North, D.; Mayfield, M. Object Models: Strategies, Patterns and Applications, Yourdon Press, 1995.
[Coleman et al, 1994] Coleman, Derek et al. Object Oriented Development - the Fusion Method, Prentice Hall, 1994.
[Coplien 92] Coplien, J.O. Advanced C++ Programming Styles and Idioms. Reading-MA, Addison-Wesley, 1992.
[Coplien 95] Coplien, J.; Schmidt, D. (eds.) Pattern Languages of Program Design, Reading-MA, Addison-Wesley, 1995.
[Deitel, 2002] Deitel, Harvey M; Deitel Paul J. JAVA – como programar, Editora Bookman, 4a Edição. 16
[Gamma 95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. Design Patterns - Elements of Reusable Object-Oriented Software. Reading-MA, Addison-Wesley, 1995.
[Goodwill, 2002] Goodwill,J. Martering Jakarta Struts, Wiley Publishing, Inc.
[Krutchen, 2000] Krutchen, Philippe. The Rational Unified Process, An Introduction, Second Edition, Addison Wesley, 2000.
[Larman, 2004] Larman, Craig. Utilizando UML e Padrões, 2a edição, Bookman, 2004. [Rational, 2000] RATIONAL, C. Unified Modeling Language. Disponível na URL: http://www.rational.com/uml/references, 2000.
[Rumbaugh, 1990] Object-Oriented Modeling and Design by James R Rumbaugh, Michael R. Blaha, William Lorensen, Frederick Eddy, William Premerlani, Prentice Hall; 1st edition (October 1, 1990)
[Waslawick, 2004] Waslawick, Raul. Análise e Projeto de sistemas de Informação Orientados a Objetos, Campus, 2004.