4
Conceitos Chaves

Este capítulo abordará os conceitos fundamentais para a execução do estudo. Os seguintes tópicos serão discutidos:

- UML.
- XMI.
- Caminhos Mais Curtos.
- Clusterização.

4.1.
UML

Os diagramas UML são feitos no momento de arquitetura do sistema e mantidos na sua manutenção. Esta etapa, por ser inicial, tem um foco mais na análise do problema (Object Management Group, 2012).

A especificação do UML é definida por uma abordagem de metamodelo que se adapta às técnicas de especificação formal. Sua arquitetura obedece os princípios de modularidade, camadas, particionamento, extensibilidade e reúso (Object Management Group, 2007).

A ênfase deste trabalho no UML é o diagrama de classe.

4.1.1.
Diagramas de classe

Idealmente, este tipo de diagrama foi feito para projetos de software orientado a objetos, mas sua abstração pode atingir diversas áreas. O diagrama de classe, basicamente, é composto por classes com os seus atributos, métodos, funções, estereótipos e por relacionamentos que ligam diferentes classes.

Já que o diagrama de classes faz parte da UML, seus modelos estendem de um metamodelo denominado *Constructs*, ilustrado na Figura 3:

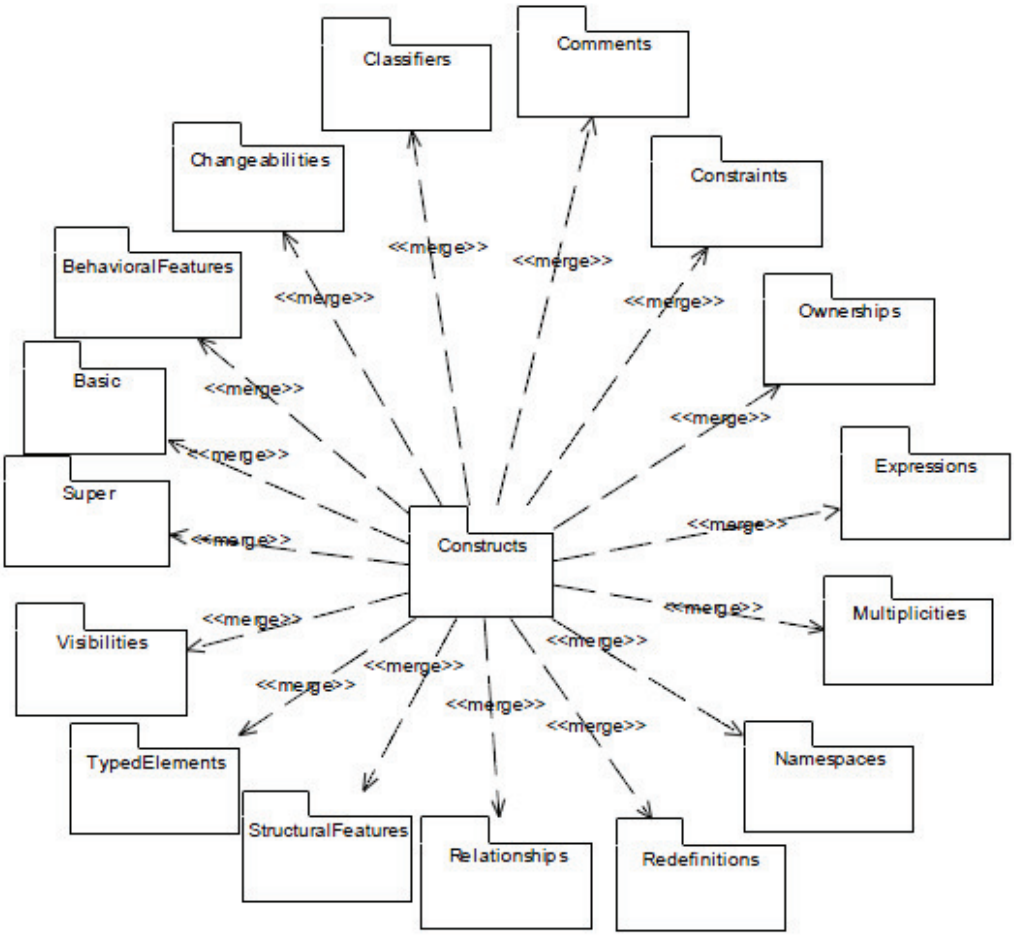


Figura 3 - Metamodelo dos diagramas de classe.

Dentro do contexto do pacote *Relationships*, segundo IBM (2012), os principais tipos de relacionamento UML entre classes são:

- *Generalização*: Capacidade de identificar características comuns entre várias classes. Uma superclasse contém as características comuns de subclasses para evitar a replicação de código e melhorar o reúso.
- *Dependência:* Relacionamento que denota um objeto que, para existir, depende do outro.
- *Realização:* Relacionamento onde uma classe implementa funções ou comportamentos especificados por outra classe. Um bom exemplo são interfaces e suas classes de implementação.
- *Abstração:* Relacionamento misto de generalização e realização. As partes comuns entre as classes são implementadas pela classe

abstrata e funções e comportamentos específicos de cada classe são implementadas pelas classes herdeiras da classe abstrata.

- *Associação:* Representa uma conexão simples entre duas classes. Pode ser recursiva (classe destino e origem da conexão são a mesma classe);
- *Associação n-ária:* É o mesmo conceito que a associação, porém, a conexão entre as duas classes é representada por classes denominadas classes de associação, o que qualifica a conexão.
- *Agregação:* É um tipo de associação onde o todo está relacionado com as suas partes.
- *Composição:* É muito semelhante à agregação. O objeto "Parte" só pode pertencer ao objeto "Todo". A diferença é que, se o objeto "Todo" morrer, todos os seus objetos "Parte" também morrem.

A ferramenta *Astah Professional 6.5*, escolhida como editor de diagramas de classe deste estudo, tem suporte para todos estes tipos de relacionamentos. Vale salientar que a abstração, nesta ferramenta, é uma generalização onde a classe pai é abstrata.

Os grafos orientados gerados para o estudo dos conceitos centrais, para a componentização e para a obtenção das métricas, terão arestas cujos pesos serão definidos para cada tipo de relacionamento descrito acima.

4.2. XMI

O XMI[8] é o formato de intercâmbio mais utilizado para compartilhamento de modelos UML. Ele descreve aspectos importantes envolvidos na descrição de objetos por XML.

O XML[9] foi criado pelo *W3C* para permitir a interoperabilidade entre aplicações diferentes, no papel de um protocolo de comunicação. Esta linguagem prega a simplicidade para seu funcionamento, onde o *parser* de leitura de documentos XML deve ser fácil de escrever, o documento deve ser legível por

[8] XML Metadata Interchange
[9] Extensible Markup Language

humanos e não só por máquinas, o seu design deve ser formal e conciso e devem ser fáceis de criar (W3C, 2012).

De acordo com o Object Management Group (2011), um dos aspectos do XMI é que, como os objetos são interconectados, inclui-se um mecanismo padrão de ligação entre os elementos dentro do mesmo arquivo ou em arquivos diferentes. Os identificadores possibilitam objetos a referenciar outros objetos por IDs ou UUIDs. E, por fim, a validação de documentos XMI é feita por *XML Schemas*.

Não existe um padrão dos elementos XMI, então, para o estudo ser abrangente o bastante, é necessário criar um mecanismo, onde o usuário entrará com o nome dos elementos XML e dos seus atributos, para que o *parser*, automaticamente, importe as informações contidas nos modelos UML.

4.2.1. XML Schema e XML Namespace

O *XML Schema* fornece um meio pelo qual um processador valida a sintaxe e a semântica de um XML, no caso, um XMI (Object Management Group, 2011). Este documento fornece regras que funcionam como um dicionário de metadados.

O uso de esquemas, no entanto, não é obrigatório, não há necessidade da referência a um esquema no arquivo XMI.

O *XML Namespace* foi adotado pelo *W3C* para permitir um mesmo arquivo XMI utilizar vários modelos simultaneamente. A validação do *XML Schema* funciona através de *XML Namespaces,* assim uma ferramenta escolhe seus próprios prefixos de namespace dentro de um arquivo XMI e, utiliza o *XML Schema* para validá-lo. A URI[10] do namespace, não o seu prefixo, é utilizada para identificar quais schemas validam o documento XML.

Todos os elementos XML utilizados na especificação do UML estão contidos no namespace *"http://www.omg.org/spec/XMI/versionnamespace"*, onde *versionnamespace* é a versão da especificação UML. Este *XML Namespace* pode ser utilizado para evitar conflitos de nomes entre os elementos do XMI.

[10] Uniform Resource Identifier

4.2.2.
Estrutura do Documento

Cada XMI deveria conter as seguintes declarações sobre o seu esquema:

- Versão do XML para instrução de processamento. Exemplo: <? XML version="1.0" ?>.
- Declarações de encoding que especificam o charset, que segue a norma ISO-10646. Exemplo: <? Versão XML = "1.0" encoding = "UCS-2" ?>.
- Todas as instruções de processamento de validação
- Um elemento XML para o *Schema*.
- Um elemento de importação XML para o *XMI Namespace.*

4.2.3.
Identificadores dos elementos

Todos os elementos dentro do XMI têm identificadores associados que podem ser o *id*, o *label* e o *uuid.*

A semântica do UML exige que os valores associados aos atributos XML sejam únicos dentro do contexto de um documento, entretanto, não requer que o valor seja globalmente único. O atributo *id* é utilizado como valor do atributo *xmi:id* para identificação única do elemento, e da cláusula *idref* para ser utilizada como referência em outros elementos dentro do documento.

O atributo *label* serve para fornecer um texto identificador de um elemento XML em particular. Estes valores são ignorados na importação, e funcionam apenas como forma de apresentação do título do elemento ou como uma referência, não funcionam como identificador.

Finalmente, o atributo *uuid* fornece um identificador global e único. Os valores destes atributos são textos pré-fixados pelo tipo do gerador de identificação e são utilizados no atributo *xmi:uuid*.

4.2.4.
Representação do Modelo de Diagrama de Classes

O nome do elemento XML para cada classe e tipo de relacionamento dentro do documento é o seu nome curto, que corresponde ao nome das *tags* XML, que representam as propriedades do modelo. Obviamente, a descrição deste nome

curto está envolvida dentro de um contexto, que está embutido no *namespace* e na árvore de *tags* XML.

A seguir, um fragmento de XMI, que descreve o *namespace* e a declaração de uma classe:

```
<xmi:XMI xmlns:uml="http://www.omg.org/spec/UML/20110701"
         xmlns:xmi="http://www.omg.org/spec/XMI/20110701">
  <uml:Class name="C1" xmi:type="uml:Class" xmi:id="_1">
    <ownedAttribute xmi:type="uml:Property" xmi:id="_2" name="a1"
      visibility="private"/>
  </uml:Class>
</xmi:XMI>
```

Figura 4 - XMI de declaração do namespace e da classe.

O código definido na Figura 4, primeiramente, define *http://www.omg.org/spec/UML/20110701* como *namespace* do documento XML. Depois, é declarada a classe de nome *C1*, que contém a propriedade privada *a1*.

Segundo o Object Management Group (2011), trabalho que resultou no documento de especificação do XMI, quaisquer relacionamentos UML entre classes são tratados como *links*.

Cabe às ferramentas de edição de diagramas UML, a definição mais detalhada das *tags* dos relacionamentos. Isto causa uma quebra de padronização, porque possibilita a falta de suporte a determinados relacionamentos ou o uso de esquemas diferentes por ferramenta.

A declaração do *link* dentro da especificação do XMI segue a mesma lógica da declaração de classes. Ao invés de ter *tags* de atributos da classe dentro da *tag class*, outras *tags*, denominadas *Linking Attributes*, são criadas para se determinar as origens e os destinos da *tag link*. Esta determinação é feita por referência via *idref* com *xmi:id* ou *xmi:uuid*.

No capítulo *Abordagem* será exibido um exemplo de arquivo XMI com detalhamentos das definições de classes e seus relacionamentos para o *Astah Professional 6.5*.

4.3. Caminhos Mais Curtos

Existem alguns algoritmos que determinam os caminhos mais curtos entre todos os nós do grafo. Porém, quando é necessário guardar a sequência de nós existente no caminho mais curto de um nó para o outro, a dificuldade aumenta

consideravelmente. Um dos algoritmos mais utilizados na área é o Floyd-Warshall.

O algoritmo de Floyd-Warshall determina as distâncias dos menores caminhos entre todos os pares de vértices de um grafo, trabalha com arestas com pesos negativos, mas não funciona quando existem ciclos negativos no grafo (Medeiros, 2012). Isto não é um problema para o contexto do trabalho, pois todas as distâncias entre nós são valores reais maiores que zero; não existe aresta de peso negativo, ou seja, é impossível a detecção de um ciclo negativo.

Este algoritmo usa programação dinâmica, que consiste em resolver um problema simples para iterativamente se resolver um problema mais complexo.

Os caminhos mais curtos $u \to w_1 \to \cdots \to w_l \to v$ entre *u* e *v* utilizam alguns nós intermediários, possivelmente nenhum. Suponha que se retire todos os nós intermediários de uma só vez. Assim, todos os caminhos de *u* para *v* seriam simplesmente a aresta *(u,v)*, se ela existir. Que tal agora gradualmente expandir o conjunto de nós intermediários permitidos? Isto pode ser feito um nó de cada vez, para atualizar o tamanho do caminho mais curto a cada estágio. Eventualmente, este conjunto cresce até todos os vértices do grafo *V*, e os verdadeiros caminhos mais curtos do grafo são descobertos (Dasgupta & Papadimitriou, 2006).

Segundo Cormem et al. (2009), o algoritmo de Floyd-Warshall confia na seguinte observação. Considere todos os vértices do grafo *G* como $V = \{1,2,3,\dots,n\}$ e o subconjunto de vértices $\{1,2,3,\dots,k\}$ para algum *k*. Para quaisquer par de vértices $i,j \in V$, considere todos os caminhos de *i* a *j* cujos vértices intermediários estão contidos em $\{1,2,3,\dots,k\}$ e *p*, o caminho mais curto entre eles. O algoritmo explora a relação entre *p* e os caminhos mais curtos de *i* a *j* no conjunto $\{1,2,3,\dots,k-1\}$. A relação depende se *k* é um vértice que participa de *p* ou não.

- Se *k* não for um vértice intermediário do caminho *p*, então todos os vértices intermediários do caminho *p* estão no conjunto $\{1,2,3,\dots,k-1\}$. Assim, o caminho mais curto de *i* a *j* no conjunto $\{1,2,3,\dots.k-1\}$ também é o caminho mais curto no conjunto $\{1,2,3,..,k\}$.
- Se *k* for um vértice intermediário do caminho *p*, então *p* é decomposto em $i \overset{p1}{\to} k \overset{p2}{\to} j$. Por construção, *p1* é o caminho mais

curto de i a k com todos os vértices intermediários $\{1,2,3,\ldots,k-1\}$. Similarmente, *p2* é o caminho mais curto de k a j com os vértices intermediários $\{1,2,3,\ldots,k-1\}$.

A Figura 5 ilustra a divisão de *p* em *p1* e *p2*:

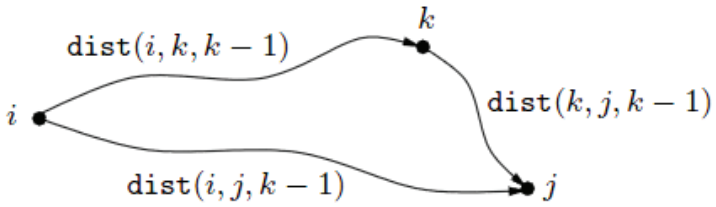


Figura 5 - Divisão do caminho *p* em dois caminhos distintos.

Baseado nestas informações, já se pode definir a fórmula recursiva da determinação do caminho mais curto de todos os nós do grafo, conforme esperado pela programação dinâmica. Considere $d(ij)^k$, o peso do caminho mais curto entre i e j para um conjunto de vértices intermediários $\{1,2,3,\ldots,k\}$. Quando $k = 0$, o conjunto de vértices intermediários não existe. O peso do caminho será $d(ij)^0 = w_{ij}$, onde w_{ij} é o peso da aresta *(i,j)*. Recursivamente, $d(ij)^k$ é definido como:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0\,, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1\,. \end{cases}$$

Figura 6 - Visão recursiva Floyd-Warshall.

Onde a resposta final é $D^{(n)} = d(ij)^n$. O pseudo-algoritmo do Floyd Warshall é demonstrado na Figura 7:

```
FLOYD-WARSHALL(W)
1  n ← rows[W]
2  D(0) ← W
3  for k ← 1 to n
4       do for i ← 1 to n
5            do for j ← 1 to n
6                 do d_ij^(k) ← min(d_ij^(k−1), d_ik^(k−1) + d_kj^(k−1))
7  return D(n)
```

Figura 7 - Pseudo-algoritmo Floyd Warshall.

Segundo Medeiros (2012), é fácil perceber, através dos três enlaces *for*, que percorrem todos os vértices do grafo, que a complexidade do algoritmo é $\theta(n^3)$.

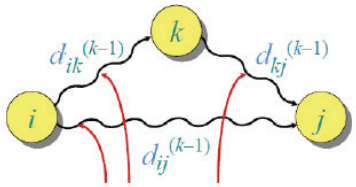
4.3.1. Guardar o Caminho

O algoritmo do Floyd Warshall introduzido até aqui só guarda os pesos dos caminhos mais curtos. Para o trabalho proposto isto não é suficiente, pois para o cálculo da metodologia *Caminho*, descrita na seção 3.4, é necessário realizar a contagem de quantos caminhos mais curtos um determinado nó pertence. Assim, é necessário guardar também os vértices intermediários dos caminhos mais curtos entre todos os nós do grafo *V*.

De acordo com Cormem et al. (2009), para resolver este problema é computada a sequência de matrizes $\pi_0, \pi_1, \ldots, \pi_n$, onde π_{ij}^k é o predecessor do vértice *j* dentro de um caminho mais curto, com origem no vértice *i* com todos os vértices intermediários $\{1,2,3,\ldots,k\}$.

É possível chegar a uma fórmula recursiva também para π_{ij}^k. Quando $k = 0$, o caminho mais curto entre *i* e *j* não tem nenhum vértice intermediário, sendo nulo se $i = j$. A Figura 8 explicita a recursão:

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty, \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty. \end{cases}$$



$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases}$$

Figura 8 - Visão recursiva Floyd Warshall guardando caminho.

Para $k \geq 1$, no caminho $i \to k \to j$, onde $k \neq j$, o predecessor de *j* pode ser o mesmo predecessor do caminho mais curto de *j* vindo de *k* com todos os vértices intermediários $\{1,2,3,\ldots,k-1\}$. Senão, é o mesmo predecessor do caminho mais curto de *j*, vindo de *i*, com todos os vértices intermediários $\{1,2,3,\ldots,k-1\}$, as distâncias dos caminhos, que já foram calculadas, irão dizer qual dos caminhos tomar.

4.4. Clusterização

O algoritmo escolhido para realizar a clusterização das classes do grafo orientado, que representa o modelo UML, é o *k-means*.

A ideia do algoritmo *k-means* (também chamado de k-médias) é fornecer uma classificação de informações de acordo com os próprios dados. Esta classificação, como será vista a seguir, é baseada em análise e comparações entre os valores numéricos dos dados. Desta maneira, o algoritmo fornecerá uma classificação automática sem a necessidade de nenhuma supervisão humana, ou seja, sem nenhuma pré-classificação existente. Por causa desta característica, o *k-means* é considerado um algoritmo de mineração de dados não supervisionado de classificação, que particiona *n* registros em *k* agrupamentos (*clusters*), onde $k < n$ (Pichiliani, 2012).

Seu funcionamento pode ser descrito da seguinte forma: dado um valor inicial de *k* médias, os registros são separados em agrupamentos, onde os *k* pontos representam o centro de cada agrupamento. Normalmente, as coordenadas iniciais desses centróides são determinadas de forma aleatória. Em seguida, cada registro é associado ao cluster cujo centro está mais próximo, através de uma métrica de distância. Existem diversas métricas de cálculo de distâncias, uma destas métricas é a distância Euclidiana[11].

A partir deste ponto, uma iteração é executada. Quando todos os registros estiverem classificados, os *k* centros são recalculados como a soma das distâncias dos registros aos centros em cada *cluster*. A próxima etapa é a associação dos registros a um novo agrupamento, segundo sua distância aos novos *k* calculados. A iteração se repete até que as médias dos clusters não se desloquem consideravelmente.

Para simplificar a explicação do algoritmo, os seguintes passos são definidos:

- *Fornecer valores para os centróides:* Escolhe-se aleatoriamente *k* elementos para representar o centróide de cada grupo.

[11] $d_{p,q} = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$, onde q e p são vetores euclidianos.

- *Gerar matriz de distância dos elementos aos centróides:* A distância entre todos os elementos não-centróides para os centróides é calculada. Como existem $(N - k)$ elementos e k centróides, este passo realiza $(N - k) * k$ cálculos.
- *Classificar os elementos:* Os elementos são classificados de acordo com sua distância do centróide de cada grupo. O centróide que está mais perto deste elemento vai incorporá-lo. Vale salientar que o algoritmo termina se nenhum elemento trocar de centróide durante este passo.
- *Calcular novos centróides:* Neste passo, os valores das coordenadas dos centróides são refinados. Para cada grupo ou *cluster*, o novo valor dos centróides é calculado através da média das distâncias entre todos os elementos do grupo. O novo elemento centróide será o que tiver a menor média de distâncias.
- *Repetir até convergir:* O algoritmo volta para a classificação dos centróides para, iterativamente, refinar o cálculo das coordenadas do centróide, até que nenhuma modificação substancial seja encontrada.

Note que a classificação do *k-means* coloca cada elemento em apenas um grupo. Assim, o algoritmo faz uma classificação *hard*. Outros algoritmos trabalham com o conceito de classificação *soft*, onde existe uma métrica que diz o quão um determinado elemento pertence a cada grupo (Pichiliani, 2012).

O algoritmo de clusterização *k-means* tem complexidade *θ(kni)*, onde *k* é o número de grupos, *n* é o número de classes e *i* é o numero de iterações até o algoritmo parar (critério de parada). A Figura 9 representa o pseudo-algoritmo do *k-means*:

```
Especifique k;
Selecione os k objetos que serão os centróides dos agrupamentos;
para todos os objetos restantes faça
        Calcule a distância entre o elemento e os centróides;
        Adicione o elemento ao agrupamento que possuir a menor distância;
        Recalcule o centróide do agrupamento;
fim para
para todos os k agrupamentos faça
        Calcule a Soma de Quadrados Residual;
fim para
repita
        para todos os n elementos faça
        Mova o elemento para os outros agrupamentos;
        Recalcule a Soma de Quadrados Residual;
                se soma dos Quadrados Residual diminuiu então
                O objeto passa a fazer parte do agrupamento que produzir maior ganho;
                Recalcule a Soma de Quadrados Residual dos agrupamentos alterados;
                fim se
        fim para
até Número de interações = i ou Não ocorra mudança de objetos
```

Figura 9 - Pseudo-algoritmo k-means.