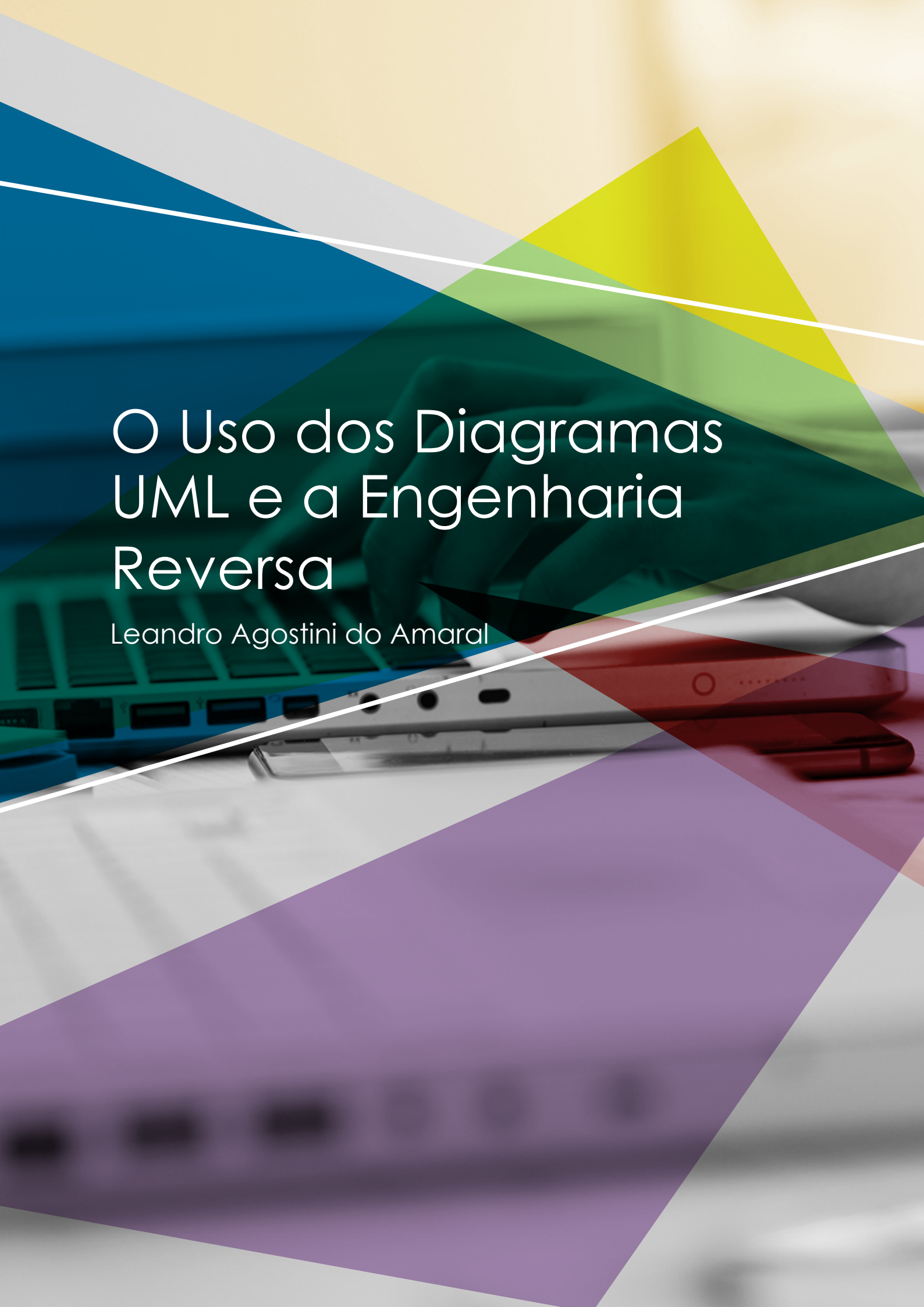

O Uso dos Diagramas UML e a Engenharia Reversa

Leandro Agostini do Amaral

Objetivos da Aprendizagem

Ao final do conteúdo, esperamos que você seja capaz de:

- Apresentar os diagramas de pacotes, perfil e de estados abordando suas principais características e aplicações;
- Abordar sobre os diagramas de sequências, tempo e caso de uso expondo suas atribuições e funcionalidades;
- Observar os aspectos da engenharia reversa com UML, verificando os seus principais aspectos.

Package Diagram

Como é feita a documentação de um sistema? Quais os requisitos básicos para que isto seja realizado? Segundo o livro *UML*: *guia do usuário*, publicado em 2012 e escrito por Grady Booch, James Rumbaugh e Ivar Jacobson, os atos de observar, especificar, desenvolver e documentar sistemas considerados de grande porte requerem uma série de manipulações elevadas de atributos, aos quais se incluem as classes, diagramas e outros elementos. O uso dos sistemas indica a necessidade de organização dos itens em agrupamentos maiores. Dentro de tal contexto em uma UML, o conceito de pacote é uma ferramenta empregada na organização de itens pertencentes a uma modelagem realizada por grupos.

Além disso, o usuário é capaz de estabelecer um controle de visibilidade desses itens, permitindo que alguns deles sejam percebidos do lado externo do pacote, enquanto outros componentes se encontram ocultos. Os pacotes são aproveitados nas visões distintas da arquitetura dentro do seu sistema. Quando têm estrutura qualificada, reúnem elementos que estão aproximados e que possivelmente irão se alterar em conjunto. Diante disso, a conclusão a que se chega é de que os pacotes bem estruturados são coesos e com baixo acoplamento, além de permitir um acesso com alto nível de controle ao conteúdo do pacote.

Para explicar o conceito de pacotes de maneira mais didática, Booch, Rumbaugh e Jacobson fazem uma analogia à construção de uma casa. Casas de cachorro, por exemplo, demandam um menor nível de complexidade. A partir do momento em que há um nível mais sofisticado de construção, como nas casas e prédios, se pode notar que tais estruturas estão inseridas em componentes maiores, como áreas públicas, em que servem para organizar os planos, mesmo não tendo qualquer relação aparente entre si.

Qualquer sistema segue este padrão de desenvolvimento. Ou seja, a única forma de decodificar um sistema complexo é agrupar as abstrações em conjuntos cada vez maiores, mas boa parte deles não são instâncias reais e só existem com o intuito de interpretar o próprio sistema.

Dentro de uma UML, o conceito de pacotes é uma série de agrupamentos que organizam um modelo para facilitar o seu entendimento, além de proporcionar o controle ao acesso dos conteúdos e às emendas dentro da arquitetura do sistema.

A Figura 1 demonstra a representação gráfica dos pacotes, na qual sua notação permite a observação dos grupos de itens que são manipulados de maneira total ou de uma forma que controle sua visibilidade e acessibilidade dos componentes individuais.

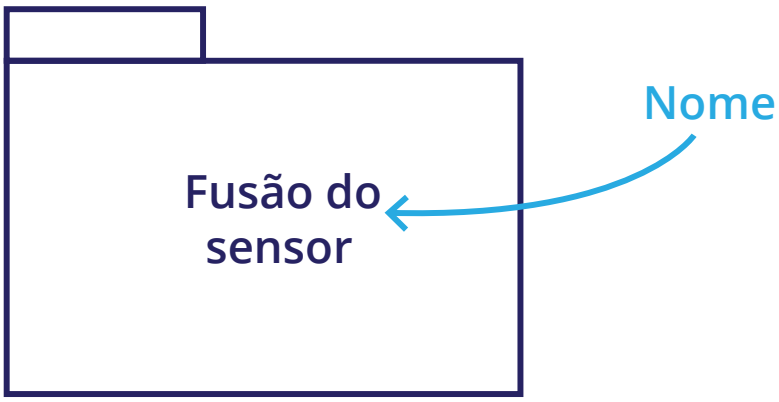

Figura 1 – Pacotes

Fonte: BOOCH; RUMBAUGH; JACOBSON, 2012. (Adaptado).

Booch, Rumbaugh e Jacobson sintetizam que um pacote nada mais é do que um mecanismo para organizar o próprio modelo dentro de uma hierarquia. Uma pequena advertência a ser feita é relativa ao pacote, simbolizado na condição de uma pasta se o conteúdo da pasta não for exibido ou, senão, como um guia.

Todo e qualquer pacote tem um **nome** que o distingue e que é uma sequência de caracteres de texto, sendo classificado como simples, se estiver sozinho, ou como qualificado, quando tiver como prefixo o nome do pacote que o contém.

Atenção

O nome de um pacote é constituído por um texto formado por letras, números e sinais diversificados, chegando a se estender por diversas linhas. De modo prático, os nomes são vistos como expressões, ou pequenos substantivos de grupos, partindo do vocabulário do modelo.

Como já mencionado, um pacote tem diversos componentes, como classes e interfaces. A propriedade de elementos, ou componentes, é considerada como uma relação composta, o que implica afirmar que os elementos estão dispostos no pacote. Se o pacote for destruído automaticamente, seus elementos também serão. Logo, cada elemento pertence apenas a um pacote.

Ainda com base nos autores citados, o pacote fornece um espaço de nome, o que significa dizer que os componentes do mesmo tipo estão de uma única maneira dentro do pacote que os contêm e, por isso, os elementos de modelos distintos têm o mesmo nome dentro de um pacote. Na prática, porém, é mais adequado dar nomes únicos para os elementos em relação a todos os tipos de pacote.

Sobre os pacotes que armazenam outros pacotes, há a facilidade de decompor os modelos de maneira hierarquizada, porém é mais adequado evitar pacotes com diversos níveis de aninhamento, usando três camadas, no máximo, a fim de limitar o que será melhor. Além disso, no intuito de organizar os pacotes, se usa a importação.

Há uma espécie de semântica de propriedade que coloca os pacotes na condição de um mecanismo essencial para lidar com escala. Sem ele, isto é, sem a presença dos pacotes, os principais modelos planos seriam extintos, o que obrigaria a todos os elementos à condição de receber nomes únicos, sendo uma situação impossível de ser gerida, em especial quando classes e outros elementos são criados por diversas equipes. Os pacotes têm um papel preponderante, pois controlam elementos que constituem o sistema. Com a evolução em taxas diferentes durante um período, se pode revelar explicitamente o conteúdo de um pacote na forma textual ou gráfica.

Em relação aos pacotes, é possível controlar a visibilidade dos elementos de um pacote de forma similar à visibilidade dos atributos e operações que fazem parte de uma classe. Um elemento de um pacote é público, indicando que o elemento é visível em relação ao conteúdo de qualquer pacote que realizar a importação daquele que contém o elemento. De maneira contrária, os elementos em proteção só são vistos através dos chamados “elementos-filho”, além dos elementos-privados, invisíveis ao exterior do pacote em que são declarados.

Para designar a visibilidade de um elemento que faz parte de um pacote, é recomendável empregar o nome do elemento como prefixo de símbolo de visibilidade. Dentro do contexto, os elementos públicos utilizam (+) como um prefixo que representa seus respectivos nomes. Sendo assim, e de forma coletiva, as partes públicas de um pacote formam a sua interface. Para Booch, Rumbaugh e Jacobson, assim como acontece com as classes, se pode classificar um elemento como protegido ou privado através do prefixo do nome do elemento. Os elementos são visualizados apenas pelos pacotes herdados, ou seja, eles se tornam ocultos fora do pacote, sob qualquer hipótese. A visibilidade do pacote aponta que uma classe se torna aparente em um mesmo pacote, mas é invisível para classes em pacotes distintos.

Para assimilar melhor os processos de importação e exportação, imagine duas classes, posicionadas lado a lado, sendo que a primeira vê aspectos da segunda. Não há uma necessidade específica de qualquer tipo de pacote quando há a presença de duas classes formando um sistema trivial.

Agora, imagine haver algumas centenas de classes, posicionadas lado a lado. Não há limites para desenvolver uma complexa rede de relacionamentos nem há chance alguma de se entender um grupo de classes extenso e sem organização. Como um acesso mais simplificado não cresce em escala, isto acaba se tornando um problema muito sério para os sistemas considerados de grande porte. Logo, visando a organização das abstrações, se segue algum tipo de pacote controlado.

Em uma segunda situação, suponha que foram usados dois grupos, A e B, e que eles sejam declarados como partes públicas dos seus pacotes relacionados, uma situação bastante distinta, pois, apesar de ambas serem públicas, uma não consegue acessar a outra. Entretanto, se o pacote de A realizar o processo de importação do pacote B, será possível vê-lo. A importação garante uma permissão de maneira unilateral, tendo em vista que os elementos de um pacote obtenham acesso aos elementos de pacotes distintos.

Na UML, a modelagem de um relacionamento de importação controla o acesso de um número extenso de abstrações, em que as partes públicas de um pacote são classificadas como exportações. As partes exportadas através de um pacote são visíveis para o seu conteúdo, apesar das dependências de importação e de acesso não serem transitivas.

Profile Diagram

Para falar do uso do Profile diagram, ou diagrama de perfil, é preciso interpretar algumas de suas propriedades. Gilleanes Guedes, no livro *UML 2 – Guia Prático*, de 2014, o considera como um diagrama mais abstrato, uma vez que adapta uma UML a uma plataforma, como J2EE, ou até mesmo a um domínio.

É possível explicar o J2EE como a plataforma Java para a criação e realização de aplicações servidoras com capacidade de auxílio ao desenvolvimento de aplicações fortes, com uma série de serviços que disponibilizam uma funcionalidade para a criação de aplicações das multicamadas, que têm a web como referência. O objetivo é tornar simples a criação de soluções no âmbito enterprise por meio de padrões, serviços e elementos modulares. Tais componentes são configuráveis ao longo do desenvolvimento e adotam um modelo de programação em paralelo com seu contêiner.

Originalmente, a linguagem UML não foi desenvolvida para essas plataformas ou domínios, portanto não se pode exigir que a UML modele as suas características. Diante disto, através do desenvolvimento de perfis, há a oportunidade de estender a linguagem ao criar novas metaclasses e estereótipos capazes de modelar novos domínios.

Ao desenvolver um perfil, se cria uma extensão da UML em um nível mais conservador, sem grandes mudanças no metamodelo original, o que torna o diagrama de perfil um mecanismo mais leve de extensão da UML, conforme afirma Guedes. Para maior aprofundamento sobre o assunto, serão estudados os conceitos básicos de um diagrama de perfil, como modelos, metamodelos e metaclasses.

Um modelo captura uma visão do sistema físico, sendo uma abstração, ou amostra, do sistema, cujo objetivo é dar alguns aspectos estruturais ou de desempenho do software. A partir dos modelos, se determina o que é importante ou não para ser incluído.

Já um metamodelo é um modelo que fornece uma linguagem para apresentar modelos. Seu objetivo é estabelecer uma semântica para modelar elementos em um modelo em que está sendo instanciado, ou seja, o modelo é uma instância de um metamodelo, como registrado por Guedes.

Figura 2 – Metaclasse Actor

Fonte: GUEDES, 2014. (Adaptado).

A Figura 2 retrata a metaclasse Actor, que, se inserida no modelo, simboliza a instância da metaclasse. A seguir, estão outras metaclasses que são inseridas nos metamodelos. São elas:

Metaclasse Classifier

É considerada como uma metaclasse abstrata que designa uma classificação de instâncias. Ela tem uma série de instâncias em comum e, além disso, é um espaço de nomes em que os componentes acrescentam aspectos. Um classificador é um modelo com uma série de generalizações, o que torna possível decidir relacionamentos de generalização para outros classificadores.

Um classificador determina uma hierarquia de generalização através dos classificadores gerais, sendo possível redefinir classificadores aninhados por meio deste tipo de classe. Ainda que um classificador seja um mecanismo que descreva aspectos ligados ao comportamento e à estrutura, as instâncias de um classificador não são lidas como objetos, e sim como elementos UML concretos.

Behaviored Classifier

É considerado como um classificador com especificações de comportamento, como indicado no seu nome. A partir dele é que outras metaclasses, como o Actor e UseCase, são derivadas.

NameSpace

É considerada como uma metaclasse abstrata que resume um componente dentro de um modelo com uma série de elementos que são reconhecidos pelo nome.

NamedElement

Representa os componentes nomeados para reconhecer o elemento identificado dentro do espaço de nome onde está demarcado. A partir desta metaclasse foram especializadas duas metaclasses especificas: Extend e Include.

DirectedRelationship

Representa uma relação entre uma coleção de componentes de um modelo-fonte e de um modelo-destino. Ela especializa as metaclasses Extend e Include, proporcionando uma herança diversificada.

State Machine Diagram

Os diagramas de estados estão entre os disponíveis em uma UML e na modelagem dos aspectos dinâmicos de sistemas, como afirmam Booch, Rumbaugh e Jacobson. Este tipo de diagrama mostra uma máquina de estados, como os diagramas de atividades, que representam um caso específico em que a maioria dos estados se encontra em atividade e as transições são ativadas através da finalização das atividades em estado de origem.

Os diagramas de atividades e os diagramas de estados são valiosos para uma modelagem que estipula o tempo de vida de um objeto. A distinção básica está no diagrama de atividades, cujo fluxo de controle decorre de uma atividade para outra, enquanto o diagrama de estados sujeita o fluxo de controle de um estado para outro. Portanto, os diagramas de estados são aproveitados na modelagem dos aspectos dinâmicos de um sistema.

Na maioria das vezes, isso está ligado aos modelos alusivos ao comportamento dos chamados “objetos reativos” que, para Booch, Rumbaugh e Jacobson, são objetos cujo comportamento responde aos eventos ativados de forma externa. Um objeto reativo tem tempo de vida e desempenho atual influenciado por seu comportamento passado. Os diagramas de estados são anexados a classes, casos de uso ou sistemas inteiros com o propósito de imaginar, especificar, construir e documentar a dinâmica de um objeto individual.

Os diagramas de estados não são relevantes apenas para determinar os modelos dos aspectos dinâmicos de um sistema, pois estão presentes no desenvolvimento dos sistemas do tipo “executáveis” mediante o uso da engenharia direta e reversa.

Para entender os diagramas de maneira mais simples, pense na figura de um investidor responsável pelo financiamento de um novo prédio. Ele não se ocupará em verificar detalhes do processo de construção, que são tarefas pertinentes ao executor do projeto.

Cabe ao investidor proteger o investimento contra possíveis riscos, sendo possível conceber dois perfis de investidor. O primeiro, confiante no executor do construtor, e o outro, mais pragmático e que desejará acompanhar o andamento do projeto antes da liberação do dinheiro, deliberando as fases do projeto. Ao longo do caminho, outras atividades referentes à construção serão seguidas. Apesar disso, é mais relevante para o investidor vislumbrar a fase de mudança do prédio do que as atividades executadas, cuja função pertence ao construtor.

Trazendo tal raciocínio para a modelagem de sistemas complexos de software, se percebe que a maneira mais comum para observar, definir, desenvolver e documentar o desempenho de alguns modelos de objetos consiste em realizar o fluxo de controle, transferindo de um estado para outro, a partir de um fluxograma.

Frente a isso, pense em uma modelagem do desempenho do sistema de segurança embutido doméstico, em que ele é executado de maneira contínua e reage a eventos externos. Outro aspecto trata da ordem dos eventos que altera a forma como o sistema se comporta de estado. Dentro de uma UML, de acordo com Booch, Rumbaugh e Jacobson, a modelagem de desempenhos ordenados através de eventos de um objeto é realizada pelo uso de diagramas de estados.

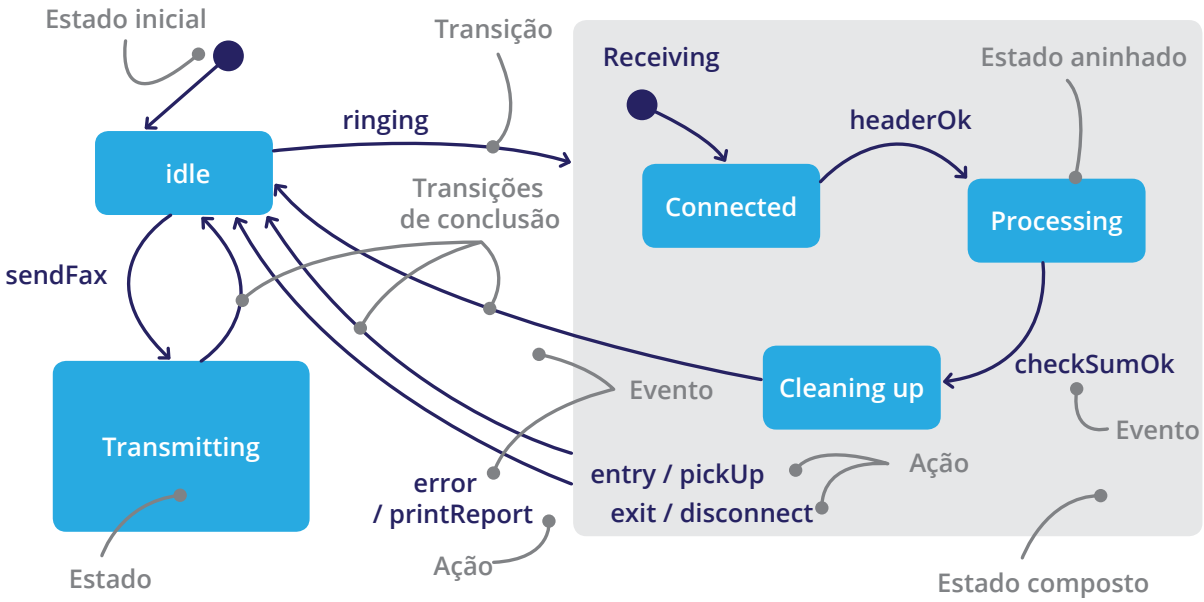

Figura 3 – Diagrama de estados

Fonte: BOOCH; RUMBAUGH; JACOBSON, 2012. (Adaptado).

Na Figura 3, se pode enxergar que um diagrama de estados consiste na exposição de uma máquina de estados, em que se destaca o fluxo de controle de um estado para outro, como já citado, sempre lembrando que uma máquina de estados é um perfil que sequencia os estados por onde um objeto passa ao longo do seu tempo de vida através das suas respostas direcionadas a esses eventos.

O estado é uma condição ou momento presente na vida de um objeto que atende a alguma condição, executa atividades ou espera algumevento – lembrando que um evento é uma especialização significativa de uma ocorrência que se localiza no tempo e no espaço. Trazendo tal conceito para uma máquina de estados, um evento é um estímulo que consegue ativar uma transição de estado, como salientado por Booch, Rumbaugh e Jacobson.

Uma transição é uma relação entre dois estados, apontando que um objeto no primeiro estado realiza atividades e entra no segundo estado assim que um evento específico for realizado e as condições específicas forem atendidas. Desta forma, é possível conceituar que uma atividade é uma realização não atômica em andamento dentro de uma máquina de estados. Por sua vez, uma ação é uma computação atômica executável que muda o estado do modelo ou o faz regredir a um certo valor.

Este diagrama é um modelo específico que compartilha propriedades similares nos outros diagramas. Um diagrama de estados se caracteriza por conta de seu conteúdo, sendo usado para realizar a modelagem dos aspectos dinâmicos de um sistema envolvendo o desempenho ordenado por eventos relacionados a qualquer modelo de objeto e visão da arquitetura do sistema, incluindo classes e interfaces.

O uso de um diagrama de estados é direcionado para criar modelos de aspectos dinâmicos de um sistema. É possível criar modelos dentro de uma conjuntura virtualizada de qualquer componente da modelagem, porém os diagramas de estados serão utilizados em um sistema, subsistema ou até mesmo dentro de uma classe. O usuário vincula diagramas de estados aos casos de uso e, ao realizar a modelagem dos aspectos dinâmicos, de sistema, classe ou caso de uso, pode valer-se dos diagramas de estados de maneira única.

Um objeto reativo é aquele cujo desempenho responde a eventos ativados de maneira externa, se tornando ocioso de forma típica até um evento em que sua resposta dependa de eventos anteriores. Depois de oferecer uma resposta ao evento, ele retorna ao estado ocioso, à espera da ocorrência de um próximo evento. Para tanto, é necessário focar nos estados estáveis do objeto, verificando aqueles que realizam a transição entre os estados e as ações realizadas em cada mudança de estado.

Nas modelagens de objetos reativos, se observam alguns aspectos: o uso dos diagramas de estados está relacionado à modelagem do perfil de objetos reativos em relação às instâncias de classes, casos de uso e do sistema. Se, por um lado, as interações definem o modelo de comportamento de uma sociedade de objetos atuando em conjunto, o diagrama de estados institui modelos de comportamento durante o seu período de vida. O diagrama de atividades modela o fluxo de controle na transição de uma atividade para outra, enquanto o diagrama de estados modela o fluxo de controle de um evento para outro.

Ao desenvolver modelos de comportamento de um objeto reativo, são ditados três aspectos básicos: primeiro, se especificam os **estados estáveis** em que o objeto atuará; depois, vêm os eventos que acionam a transição de um estado para outro; e, por fim, as atividades que acontecem nas alterações de estado. Esta modelagem envolve a chamada “modelagem do tempo de vida de um objeto”, iniciada junto com o objeto e seguindo até sua extinção, evidenciando os estados estáveis em que o objeto será visto.

Atenção

Um estado estável simboliza a condição em que o objeto será visto durante certo período. Na existência de um evento, o objeto executará a transição de um estado para outro. Há a possibilidade de tais eventos acionarem as chamadas “autotransições” e as “transições internas”, em que a origem e a destinação da transição acontecem dentro do mesmo estado. O objeto responde acionando uma atividade, reagindo a um evento ou a uma mudança de estado.,

Em concordância com o que está no livro de Booch, Rumbaugh e Jacobson, algumas ações são realizadas para modelar um objeto reativo, como:

- Escolher o contexto para a máquina de estados, seja uma classe, um caso de uso ou o sistema todo;
- Escolher os estados inicial e final do objeto. Para orientar o restante de seu modelo, estabeleça as pré e pós-condições dos estados inicial e final, respectivamente;
- Decidir os estados estáveis do objeto, levando em conta as condições que o objeto se submeterá por algum período identificável de tempo, começando com estados de nível mais alto e depois considere seus subestados;
- Decidir a ordenação parcial significativa dos estados estáveis ao longo do tempo de vida do objeto;
- Decidir os eventos que ativam a transição de um estado para outro. Faça a modelagem como a ativação de transições que passam de uma ordenação legal de estados para outra ordenação;
- Anexar ações às transições, como em uma máquina de Mealy, e/ou a esses estados, como em uma máquina de Moore;
- Considerar formas para simplificar sua máquina, a partir de subestados, ramificações, bifurcações, uniões e estados de histórico;
- Verifique se todos os estados são alcançados sob alguma combinação de eventos;
- Verifique se nenhum estado é um “buraco negro” no qual nenhuma combinação de eventos fará a transição do objeto para outro estado;
- Examine a máquina de estados manualmente ou com ferramentas para verificá-la em relação a sequências esperadas de eventos e respectivas respostas.

Sequence Diagram

Quando se trata de Sequence diagram, ou diagrama de sequências, sua principal funcionalidade é a ordenação temporal das mensagens. Na Figura 5, há alguns aspectos relevantes sobre o diagrama de sequências, que é composto da inserção de objetos na interação pertencentes a um nível mais elevado do diagrama, como no chamado “eixo X”. De maneira típica, o objeto responsável pela inicialização da interação é inserido à esquerda, enquanto os objetos mais subordinados vão evoluindo à direita.

Em seguida, as mensagens enviadas e recebidas pelos objetos são dispostas no eixo Y, em uma ordem crescente ao longo do tempo, se deslocando do topo para a base, formando uma indicação visual do fluxo de controle durante um período.

Os diagramas de sequências e os de comunicação têm similaridades e diferenças entre si. Diante de tal contexto, é possível versar sobre dois aspectos que diferenciam os diagramas de sequência dos de comunicação. O primeiro alude à linha de vida do objeto, que é uma linha esboçada verticalmente traduzindo a existência de um objeto em um período. Os objetos dentro de um diagrama de interação, ao qual os diagramas de sequências fazem parte, existirão durante o mesmo período de duração da interação. Portanto, para Booch, Rumbaugh e Jacobson, tais objetos estarão dispostos de maneira alinhada na parte superior do diagrama e suas linhas de vida serão esboçadas da parte superior para a parte inferior do diagrama.

Os objetos são desenvolvidos ao longo de uma interação. Um ponto a se observar é que as linhas de vida começam com o destinatário da mensagem, cujo estereótipo é estipulado como Create. Da mesma forma, eles são extintos ao longo da interação através de um destinatário da mensagem descrito como Destroy.

Outro aspecto aborda a interação, isto é, a história de objetos individuais. Os símbolos de objetos cujos nomes estejam sublinhados serão inseridos no começo da linha da vida. Em boa parte do tempo serão apontadas as interações prototípicas, no entanto são os papéis prototípicos que expressam objetos distintos em cada etapa da interação.

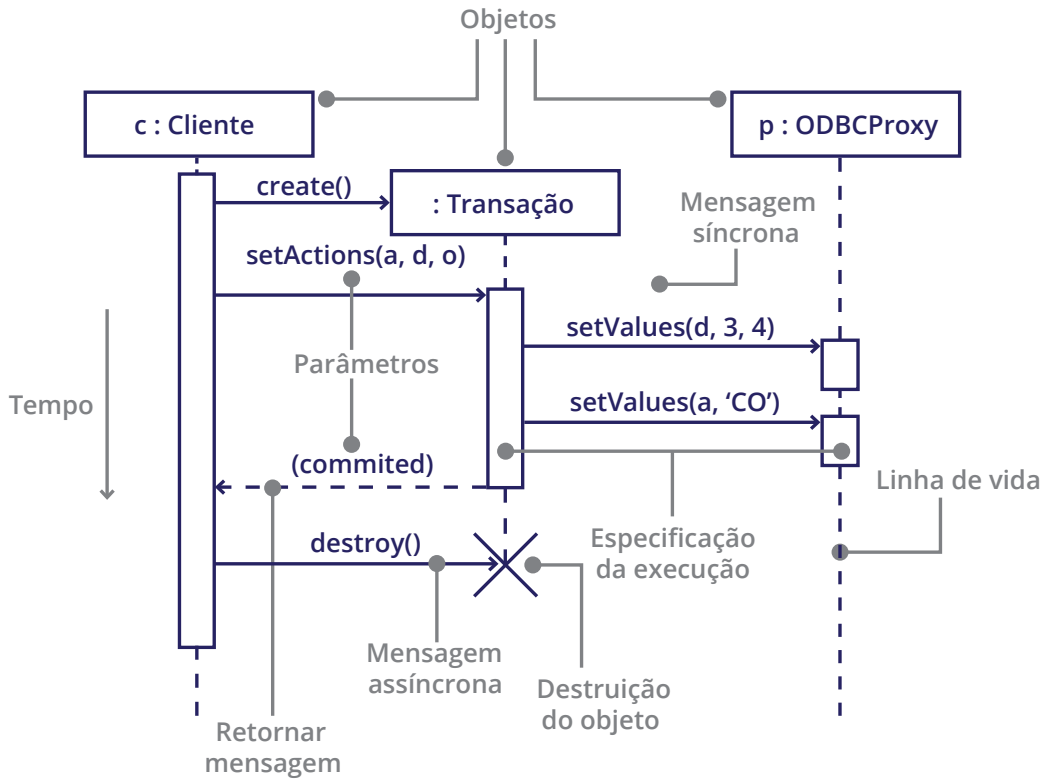

Figura 4 – Diagrama de sequências

Fonte: BOOCH; RUMBAUGH; JACOBSON, 2012. (Adaptado).

A segunda diferença entre os diagramas de sequências e de comunicação está no foco de controle. Dentro de um diagrama de sequência, o foco de controle consiste em um retângulo, de comprimento elevado e largura estreita, que apresenta o momento em que um objeto executa uma ação, de maneira direta ou através de um processo subordinado, de modo que a área superior do retângulo esteja alinhada com o começo da ação e a parte inferior com a conclusão.

O conteúdo mais relevante dentro de um diagrama de sequências é o conjunto de mensagens. Uma mensagem é exibida através de uma seta de uma linha da vida que pode ser transferida para outra, responsável por mostrar o que as mensagens receberão. Se houver uma mensagem assíncrona, a linha tem uma seta com menor espessura. Do contrário, a configuração da linha será uma seta triangular cheia. Uma mensagem síncrona, ou retorno de chamada, tem uma resposta através de uma linha esboçada por meio de uma seta mais fina. Por seu turno, a mensagem de retorno é excluída, visto que há um retorno subentendido depois de qualquer chamada.

No que diz respeito à ordenação temporal dentro de uma linha de vida única, não há um grande interesse sobre a distância exata, pois as linhas da vida se limitam a sequências relativas, o que significa afirmar que a linha da vida não é um diagrama de escala de tempo. Com base no que foi dito por Booch, Rumbaugh e Jacobson, as

posições das mensagens dispostas em pares separados de linhas da vida não resultam em qualquer tipo de sequenciamento de informações. Em suma, as mensagens se sucedem em uma ordem qualquer, fazendo com que a ordenação parcial seja formada por um conjunto inteiro de mensagens dentro das linhas da vida que estão separadas.

Uma sequência de mensagens possui uma sequência linear unificada, mas é preciso demonstrar as condicionais e loops com frequência e, esporadicamente, a execução concorrente de diversas sequências. O modelo de controle de alto nível se dá por operadores de controle estruturados, presentes em diagramas de sequências.

O operador de controle é exposto como uma região retangular dentro de um diagrama de sequências. Reparando com mais detalhamento, ele possui uma tag, que nada mais é do que um rótulo de texto que informa qual o tipo de operador de controle. Ele é aplicado nas linhas da vida que cruza, sendo que isso será considerado o corpo do operador. Se uma linha da vida não estiver adequada a ele, a mesma é obstruída no topo do operador de controle e volta para a base. Os tipos de controle mais empregados são:

- **Execução opcional**: a tag que a compõe é denominada de opt. Neste tipo de controle, o corpo do operador de controle é realizado se uma condição de guarda for considerada como verdadeira em uma entrada do operador. A condição de guarda é uma expressão booleana que é vista entre colchetes na camada superior de qualquer linha da vida presente no corpo, além de fazer citação aos requisitos do objeto;
- **Execução condicional**: representada pela tag alt, divide o corpo do operador em diversas partes, chamadas de **sub-regiões**, através de linhas esboçadas horizontalmente. Cada sub-região é um ramo de uma condicional, com sua respectiva condição de guarda. Se for verdadeira, haverá uma execução da sub-região, porém vale se atentar que, se houver várias condições de guarda verdadeiras, a seleção de uma sub-região não será o ponto determinante. Não havendo nenhuma condição de guarda verdadeira, o controle se mantém com o operador;

Atenção

Uma sub-região tem uma condição de guarda especial ou else. Nesta situação, a sub-região será realizada desde que não existam outras condições de proteção verdadeiras.

- **Execução paralela**: marcada pelo uso da tag par, faz com que cada sub-região traduza um modelo de computação denominada de paralela, ou concorrente. Em boa parte dos casos, cada sub-região tem linhas da vida diferentes e, no momento em que o operador de controle entra, as sub-regiões passam a ser realizadas de forma paralela. A execução das mensagens sucedidas dentro de cada sub-região é sequencial, mas a relativa ordem das mensagens dentro das sub-regiões paralelas é impositiva. Entretanto, tal conceito não deve ser usado na hipótese de uma interação entre computações distintas;
- **Execução de loop (iterativa)**: representada pela tag loop, o corpo do loop é executado de maneira repetitiva quando a condição de guarda for considerada como verdadeira, antes de cada iteração. Do contrário, o controle não passa pelo operador.

Na Figura 5, há um exemplo simples de operadores de controle.

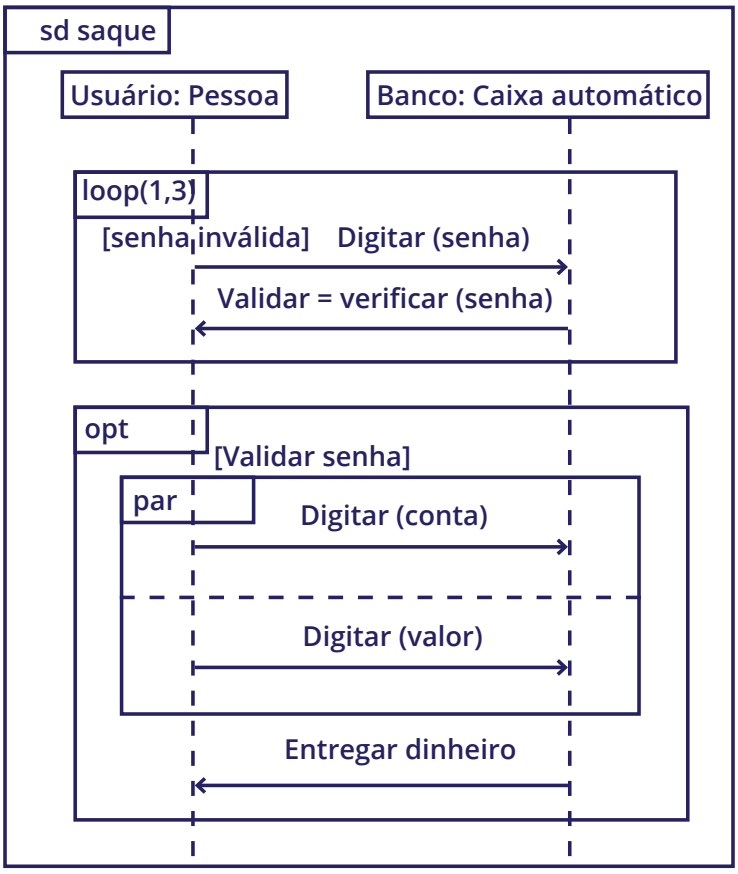

Figura 5 – Operadores de controle estruturado

Fonte: BOOCH; RUMBAUGH; JACOBSON, 2012. (Adaptado).

O usuário é responsável por iniciar a sequência, na qual o operador inicial é um loop. Dentro do loop, o usuário digita a senha para o sistema verificar. Enquanto a senha estiver incorreta, o loop continuará, mas, depois de três tentativas, o loop termina de qualquer maneira. Diante disso, o próximo operador será um opcional, cujo corpo será executado com a validade da senha. Não havendo validade, o resto do diagrama de sequências passa a ser desconsiderado.

O corpo do operador opcional possui um operador paralelo, com duas sub-regiões. A primeira informa ao usuário sobre a conta, e a outra sobre a informação do montante. Por mais que se apresentem de maneira paralela, elas são executadas em qualquer ordem.

Esse procedimento destaca a existência da concorrência, não assinalando a execução física de forma simultânea, significando que duas ações realizadas sem coordenação se dão em uma ordem qualquer. Havendo independência nas ações, elas se sobrepõem. Nas ações sequenciais, isto acontece de forma aleatória. O operador paralelo será finalizado depois de executar duas ações.

Timing Diagram

A respeito do Timing diagram, ou diagrama de tempo, se percebe que ele contém similaridades com o diagrama de máquinas de estado. Entretanto, sua principal diferença está no fato de que o diagrama de tempo muda o estado de um objeto ao longo do tempo, o que reflete em pouco uso para modelar aplicações comerciais, sendo mais empregado na modelagem de sistemas com recursos de multimídia/hipermídia.

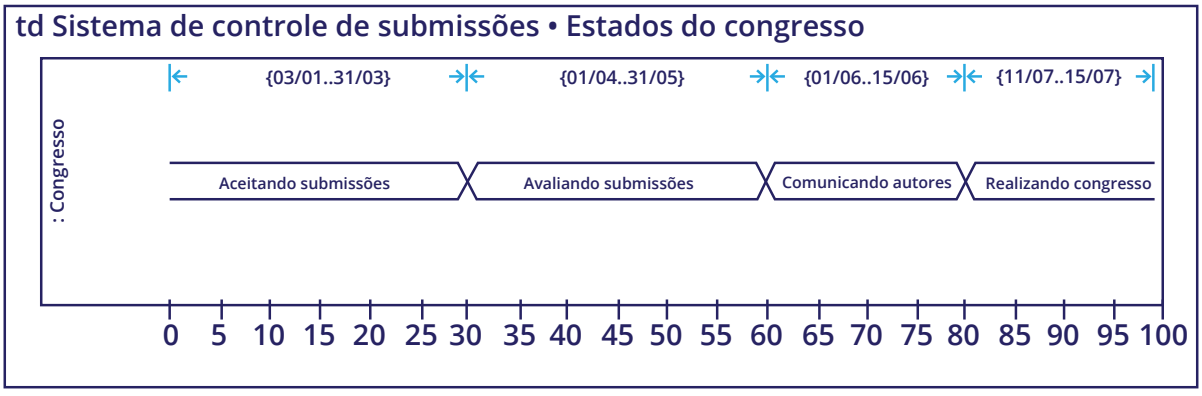

Figura 6 – Diagrama de tempo (concisa)

Fonte: GUEDES, 2014. (Adaptado).

O diagrama de tempo tem duas maneiras de exibição, mediante a representação concisa, como na Figura 6, ou através de uma notação robusta, em que as etapas estão dispostas como um gráfico.

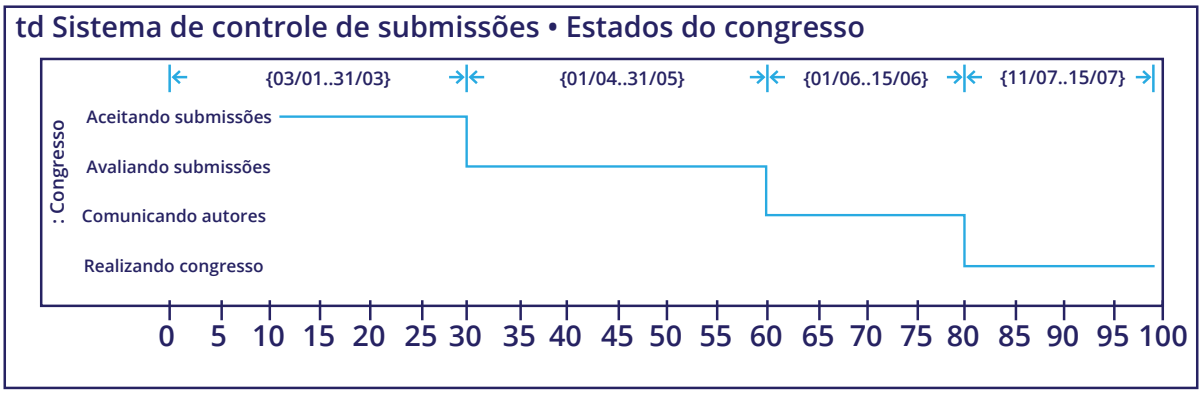

Figura 7 – Diagrama de tempo (robusta)

Fonte: GUEDES, 2014. (Adaptado).

Um aspecto diz respeito aos formatos aplicados nos diagramas de linha de tempo, visto que o ato de aplicar um formato consegue, de imediato, reorganizar o diagrama de maneira que os padrões estejam dispostos ao longo de um período de tempo.

O primeiro formato é o ordenado, que se caracteriza por ser mais útil quando o usuário tem um diagrama junto com uma sequência de eventos. A sua utilidade é notada:

- Ao adquirir uma quantidade de dados e aplicando em um formato inicial;
- Ao analisar os dados de volume elevado;
- Para avaliar a exibição e impressão.

Com um formato ordenado, se pode aproveitar o layout em um diagrama inteiro. Além disso, é possível restringir o layout para os componentes escolhidos ou até mesmo limitar o layout para os componentes que estão dentro de um intervalo de data e hora. Outra possibilidade é a configuração da distância entre os componentes sucessivos, sendo possível selecionar a configuração da distância entre itens.

Outro tipo de formato é o proporcional, considerado essencial quando há um diagrama associado a uma sequência de eventos e que ajuda a adquirir uma visão da maneira como os eventos são realizados em tempo real, mas pode gerar diagramas mais ampliados. Na importação dos dados, o formato proporcional dá uma observação inicial da maneira que os eventos são difundidos no tempo antes da aplicação em formatos, liberando a inserção no diagrama inteiro ou apenas uma escolha de componentes de diagrama, sendo possível determinar como a largura do diagrama é especificada.

O formato linha de tema é útil quando o usuário contém diversas linhas de tema dentro do seu diagrama, por ser possível categorizá-las através de uma propriedade para alocar a linha de tema com boa parte das interligações surgidas na parte superior do diagrama. Além disso, se pode especificar a separação vertical e alterar a ordem das

linhas de tema. Por fim, se alinham os ícones da linha de tema e caixas de evento, sem esquecer de organizar caixas de evento desviadas. O formato de linha de tema não é adequado para diagramas sem itens de controle.

Ao se referir ao layout linha de tema agrupada, se observa que sua utilidade ocorre quando existem diversas linhas de tema dentro de um diagrama e se deseja reduzir o número de vínculos que ultrapassam as linhas de tema, mas lembrando que ele é tratado também com um formato proporcional. Com base nas informações do IBM Knowledge, com este layout é possível:

- Organizar as linhas de tema com o objetivo de reunir o seu diagrama em grupos de linhas de tema com alto nível de conexão;
- Reduzir a quantidade de vínculos que atravessam as linhas de tema;
- Limitar as extensões na linha de tema;
- Alocar os ícones da linha de tema no começo das linhas de tema, antes do vínculo inicial. No momento em que uma linha de tema tem só um vínculo, o ícone da linha de tema será alocado no momento do vínculo.

O formato agrupado temporal organiza componentes da esquerda para a direita dentro do seu diagrama, e os componentes que acontecem em tempos similares são exibidos juntos. O formato agrupado temporal é usado para:

- Coletar dados de alto volume para buscar bursts de atividade que possam estar listados;
- Ser aceito em um diagrama de linha de tempo com diversos eventos que acontecem em intervalos irregulares;
- Limitar as larguras de diagramas que são extensas à medida que são desenhadas de maneira proporcional, dado que o formato mantém uma indicação de períodos de baixa e alta atividade.

Um formato agrupado temporal origina um diagrama no qual a distância entre cada componente é fixada pelo usuário, compondo um formato não proporcional, ou seja, a diferença entre os componentes é diferente em relação ao tempo entre eles. Com um formato agrupado temporal, se pode implementar o layout no diagrama inteiro, restringi-los para os componentes selecionados ou para os itens em um intervalo de data e hora. Além disso, se pode configurar a diferença de tempo máxima entre itens consecutivos, a distância entre os grupos e entre itens sucessivos em um grupo.

Use Case Diagram

Os diagramas de casos de uso são apenas um dos diagramas disponíveis dentro da UML direcionados para a modelagem de aspectos dinâmicos de sistemas, de acordo com Booch, Rumbaugh e Jacobson. Além da modelagem do comportamento do sistema, eles são essenciais para um subsistema ou até mesmo uma classe, onde cada um fornece uma série de casos de uso.

É evidente que os diagramas de casos de uso são essenciais na visualização, especificação e documentação do perfil de um elemento. Tais diagramas tornam o conjunto composto por sistemas, subsistemas e classes com elevado nível de acessibilidade e percepção, já que disponibilizam uma visão exterior abordando como os elementos são usados. Os diagramas de casos de uso são relevantes para avaliar sistemas executáveis por meio de engenharia direta e entendê-los através da engenharia reversa.

A partir da UML, é possível inserir os diagramas de casos de uso para examinar o desempenho de um sistema, subsistema ou classe. Isto faz com que os usuários compreendam a forma de uso dos componentes e como os desenvolvedores irão implementá-los. Na Figura 8, o usuário disponibiliza um diagrama de caso de uso ao realizar a modelagem do comportamento da caixa.

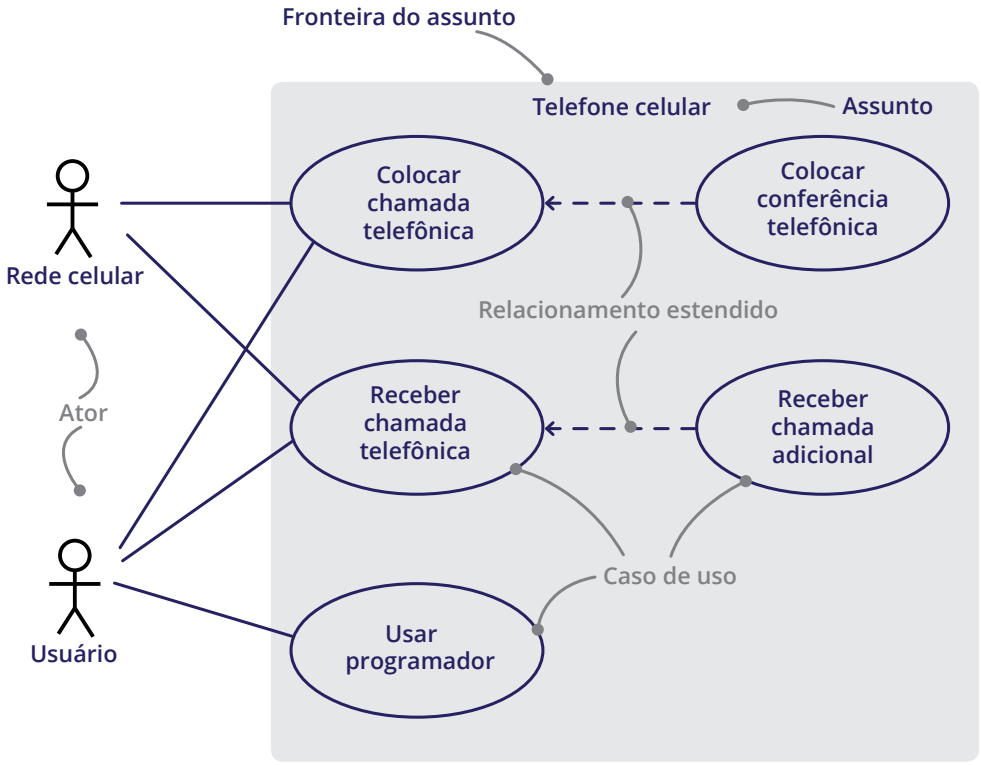

Figura 8 – Diagrama de caso de uso

Fonte: BOOCH; RUMBAUGH; JACOBSON, 2012. (Adaptado).

Portanto, um diagrama de caso de uso consiste em um modelo especial de diagrama, posto que ele se destaca por compartilhar propriedades similares a todos os diagramas, que se resumem ao nome e conteúdo gráfico da projeção dentro de um modelo.

Os diagramas têm uma série de notas e restrições, assim como em outros diagramas. Eles contêm pacotes usados para reunir componentes do modelo em grupos maiores. De maneira ocasional, se incluem as instâncias de casos de uso dentro dos seus diagramas no momento em que se quer analisar um sistema especial em execução. O usuário assume os diagramas de casos de uso para a realização da modelagem da visão do caso de uso dentro de um panorama, como um sistema, permitindo um suporte para o desempenho.

Na modelagem da visão de caso de uso de um cenário, se usam os diagramas de casos de uso de duas formas distintas: primeiro, se pode realizar a modelagem de um cenário. Ela é responsável pelo desenho de uma linha em torno do sistema, mostrando os atores excluídos do cenário e a maneira como eles interagem; a segunda forma está relacionada à modelagem dos requisitos de um sistema, determinando o que o cenário realizará de **maneira externa**, independentemente de como o cenário será executado.

Atenção

Em um sistema, alguns elementos estarão dentro ou fora do sistema, a depender da funcionalidade. Um sistema de validação de cartão de crédito, por exemplo, estará dentro do sistema, enquanto os clientes e as instituições atuam de maneira externa.

As situações ocorridas em um sistema são as executoras do comportamento; já as que se encontram fora aguardam que seja disponibilizado por meio do sistema. As situações externas que realizam a interação com o sistema definem o contexto do sistema que ordena o cenário em que sistema irá atuar, já que é possível, dentro de uma UML, realizar a modelagem de um sistema através de um diagrama de caso de uso, enfatizando os atores no sistema. Definir o que será incluído como ator auxilia na especificação da classe de coisas que são interagidas com o sistema e decidir o que não será incluído é relevante, pois limita o ambiente do sistema para inserir os atores imprescindíveis na vida do sistema. De acordo com Booch, Rumbaugh e Jacobson, para a realização da modelagem do contexto dentro um sistema é preciso:

- Reconhecer as limitações do sistema, fixando quais comportamentos fazem parte dele e quais são executados por entidades externas, demarcando o cenário;
- Reconhecer os atores em torno do sistema, levando em consideração quais grupos carecem deste sistema para a execução de suas atividades e suas funções, vislumbrando o grupo responsável pela interação com algum hardware externo ou outros sistemas de software e aqueles que executam funções auxiliares de gerenciamento e de manutenção;
- Organizar os atores similares dentro de uma hierarquia de especialização;
- Disponibilizar um estereótipo a cada ator, para auxiliar no entendimento.

É necessário completar um diagrama de caso de uso através desses atores e ditar os caminhos de comunicação até os casos de uso do sistema pertencentes a cada ator. A Figura 9 dá o contexto de um sistema de validação de cartões de crédito, em que é possível ver clientes, individual e jurídico, quando interagem com o sistema, ressaltando o desempenho das instituições do sistema, como a instituição de venda e a financeira patrocinadora.

Tal metodologia é aplicada à modelagem do contexto dentro de um subsistema. Se considerarmos um sistema dentro de um nível de abstração, ele é um subsistema que faz parte de um sistema mais extenso em um nível maior de abstração. Portanto, a modelagem do contexto de um subsistema é considerada quando se está desenvolvendo a partir de sistemas interligados.

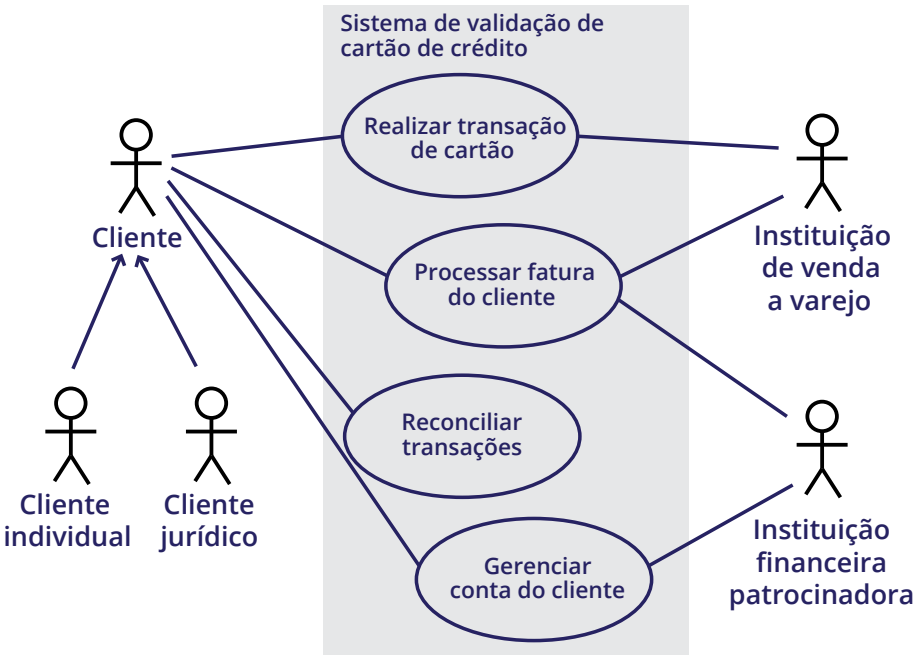

Figura 9 – Contexto de um sistema

Fonte: BOOCH; RUMBAUGH; JACOBSON, 2012. (Adaptado).

No que diz respeito à modelagem dos requisitos de um sistema, um requisito é um aspecto do projeto, uma propriedade ou um desempenho de um sistema. Ao definir os requisitos, ou atributos, do sistema, é apresentado um contrato, definido entre os elementos externos ao sistema e o sistema em si. Assim, o usuário não se preocupa com as atividades desenvolvidas do sistema, e sim com seu funcionamento.

Um sistema bem delimitado realiza todos os seus atributos de maneira fidedigna, pragmática e confiável. À medida que se desenvolve um sistema, se cria um consenso do que ele realizará. De modo parecido, ao receber um sistema a ser usado, é primordial entender seu comportamento para usá-lo de maneira adequada.

Os requisitos são das mais variadas formas, pois boa parte dos requisitos funcionais de um sistema aparecem como diagramas de casos de uso da UML, que são importantes para a administração dos requisitos. Para executar a modelagem dos requisitos de um sistema, se deve:

- Determinar o contexto do sistema, reconhecendo os atores ao seu redor e levando em consideração, para cada ator, o desempenho que cada um espera que o sistema proporcione;
- Selecionar comportamentos comuns, como casos de uso;
- Realizar a fatoração do desempenho comum em novos casos de uso usados pelos outros;
- Realizar a fatoração do desempenho variante em novos casos de uso que aumentam os fluxos da linha principal;
- Realizar a modelagem que envolve, dentre outros aspectos, os atores e seus relacionamentos dentro de um diagrama de uso;
- Incluir adornos nos casos de uso com notas exibindo requisitos não funcionais.

A Figura 10 amplia o diagrama de caso de uso anterior. Mesmo que se escondam as relações entre os atores e os casos de uso, são aproveitados casos de uso adicionais que não são visíveis ao cliente médio, por não serem comportamentos fundamentais ao sistema. Esse diagrama é considerado como essencial, pois disponibiliza um ponto inicial comum de suas decisões relativas aos requisitos funcionais do sistema para os agentes pertencentes ao mesmo, como usuários finais, especialistas de domínio e desenvolvedores para visualização, especificação, construção e documentação. A descoberta de uma fraude de cartão, por exemplo, é um procedimento considerável tanto para o setor de venda quanto para a instituição financeira que patrocina. De maneira similar, apresentar o status da conta é outro comportamento requerido pelo sistema através das diversas instituições em seu contexto.

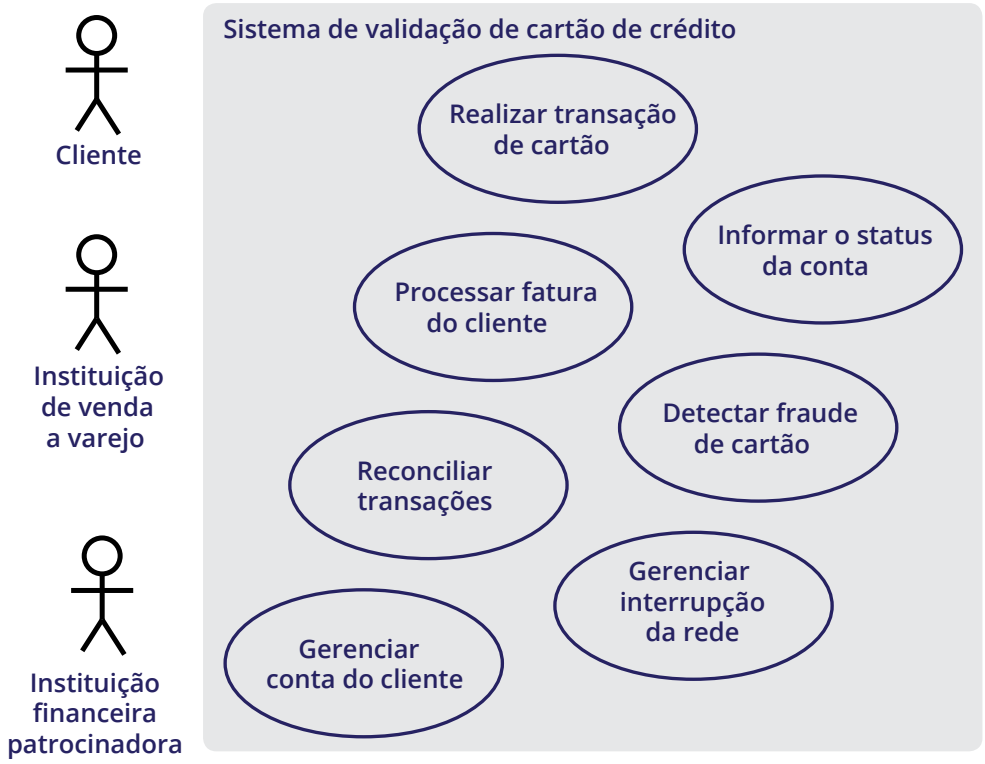

Figura 10 – Requisitos de um sistema

Fonte: BOOCH; RUMBAUGH; JACOBSON, 2012. (Adaptado).

O requisito modelado pelo caso de uso denominado “Gerenciar interrupção da rede” é distinto, pois exprime um comportamento acessório do sistema, essencial para sua operação contínua e confiável, como relatado no livro de Booch, Rumbaugh e Jacobson. A partir do momento em que a estrutura do caso de uso é constituída, se descreve o comportamento de cada caso de uso.

É necessário escrever um ou mais diagramas de sequências em cada caso da linha principal. Depois, se escrevem os diagramas de sequências para os chamados “casos variantes”. Por fim, se escreve ao menos um diagrama de sequências para exibir cada modelo de erro ou condição de exceção. O tratamento de erros faz parte do caso de uso e é planejado junto com o desempenho normal.

Ao desenvolver os diagramas de casos de uso dentro da UML, é preciso lembrar que todo diagrama de caso de uso simboliza uma apresentação gráfica da visão estática do caso de uso dentro de um sistema. Booch, Rumbaugh e Jacobson afirmam que um diagrama de caso de uso bem estruturado tem como objetivo comunicar um aspecto da visão estática de caso de uso do sistema, além de conter apenas os casos de uso e atores fundamentais. Além disso, o diagrama visa fornecer detalhes consistentes com seu nível de abstração.

Engenharia Reversa com UML

A engenharia reversa, no processo de desenvolvimento de software, tem o objetivo de criar informações com ponto de partida nos códigos, a fim de desenvolver novos itens que colaborem com o projeto. Tal metodologia para códigos-fonte auxilia na construção de método e códigos novos a partir dos modelos.

Ideias sobre os sistemas de software e ferramentas realizam a execução de trechos dos códigos para a criação de documentos através da engenharia reversa. Esta engenharia é um componente de software que cria arquiteturas novas de software de maneira automatizada ou semiautomatizada.

Arquiteturas ou projetos menores de software não desenvolvem documentos sobre o desenvolvimento feito. Sendo assim, cabe à engenharia reversa desenvolver o diagrama de classes partindo de sistemas já adotados, além de servir como método para criar novos modelos de projeto que ajudam na compreensão do sistema. Entretanto, não existem mecanismos que desenvolvam e atualizem os diagramas, o que representa um aspecto negativo das ferramentas de engenharia reversa, além do fato de que ela não é tão usada na abstração de informações de produtos concorrentes.

A engenharia reversa tem a função de reverter um código-fonte de software por meio das determinações com elevado nível de abstração.

Sendo assim, se pode reconhecer informações das mudanças de cada código. A Figura 11 apresenta o uso da técnica por meio do código-fonte.

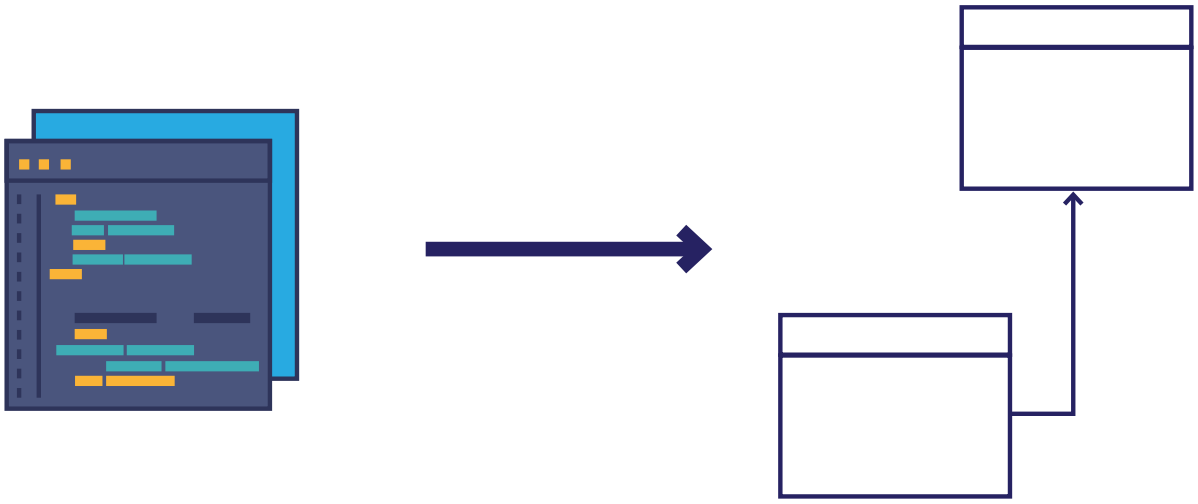

Figura 11 – O uso da técnica da engenharia reversa por meio do código-fonte

Fonte: SANTOS, 2018. (Adaptado).

A Figura 11 traz um conceito generalista da engenharia reversa com o código-fonte fazendo a mudança para um diagrama de classes. Em situações eficientes, ela executa mudanças em partes do modelo ao invés do modelo total. A engenharia reversa realiza um processo de mudança, ou seja, o código-fonte é usado para que seja criado um componente novo a partir dele, capturando informações, melhorando as operações e diminuindo o risco e o custo envolvidos no desenvolvimento de um software. No momento atual, a engenharia reversa é útil na busca de definição da linha de produtos de software.

Conclusão

Ao longo desse estudo, se concluiu que o ato de observar, especificar, desenvolver e documentar sistemas considerados de grande porte exige um conjunto de manipulações de requisitos elevados e que a utilização de sistemas aponta a necessidade de organização dos itens em grupos maiores. Neste contexto, foi possível entender o conceito de pacote, ferramenta usada na organização de itens pertencentes a uma modelagem realizada nos grupos.

Um diagrama de perfil é mais abstrato em comparação com os outros devido à adaptabilidade de uma UML a uma plataforma ou domínio, mas há a possibilidade de aumentar a linguagem ao desenvolver novas metaclasses e estereótipos que permitam a modelagem de novos domínios.

Os diagramas de estados mostram, dentre outras coisas, uma máquina de estados. Os diagramas de atividades, por seu turno, representam um caso específico de diagramas de estados em que a maioria dos estados se encontrarem em atividade.

Também se notou que o diagrama de tempo contém similaridades com o diagrama de máquinas de estado. A diferença está no diagrama de tempo, que altera o estado de um objeto ao longo do tempo.

Se analisou ainda que os diagramas de casos de uso são estabelecidos como um dos diagramas disponíveis dentro da UML focada para a modelagem de aspectos dinâmicos de sistemas. Além da modelagem do comportamento do sistema, eles são fundamentais em um subsistema ou até mesmo em uma classe na qual cada um se apresenta.

Quanto ao diagrama de sequências, sua principal funcionalidade é desenvolver uma ordenação temporal das mensagens, sendo composto, primeiramente, inserindo os objetos na interação pertencentes a um nível mais elevado do diagrama.

Referências

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML**: guia do usuário. Trad. Fábio Freitas da Silva e Cristina de Amorim Machado. 12 reimp. Rio de Janeiro: Elsevier, 2012.

GUEDES, G. T. A. **UML 2** – Guia Prático. 2 ed. São Paulo: Novatec Editora, 2014.

IBM KNOWLEDGE CENTER. **Aplicando formatos a diagramas de linha de tempo**. Disponível em: https://docs.i2group.com/anb/10.0.1/apply_layout_timeline_charts.html. Acesso em: 15 fev. 2024.

SANTOS, M. **TechREF**: uma técnica de engenharia reversa orientada a features. 2018. 106 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação Aplicada, Universidade do Vale do Rio dos Sinos, São Leopoldo, RS, 2018. Disponível em: http://www.repositorio.jesuita.org.br/bitstream/handle/UNISINOS/7024/Maicon%20dos%20Santos_.pdf?sequence=1&isAllowed=y. Acesso em: 15 fev. 2024.