UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Avaliação de metodologias de análise e projeto orientadas a objetos voltadas ao desenvolvimento de aplicações, sob a ótica de sua utilização no desenvolvimento de frameworks orientados a objetos

por

RICARDO PEREIRA E SILVA

T.I. n. 556 CPGCC-UFRGS
Trabalho Individual I

Prof. Roberto Tom Price
Orientador

Porto Alegre, julho de 1996

Sumário

Lista de abreviaturas

- ACO - análise do comportamento do objeto
- AEO - análise da estrutura do objeto
- CASE - computer aided Software Engineering (Engenharia de Software auxiliada por computador)
- DFD - diagrama de fluxo de dados
- ER - entidade-relacionamento
- LOTOS - Language of Temporal Ordering Specifications (Linguagem de Especificações Ordenadas no Tempo)
- OMT - Object Modelling Technique (Técnica de Modelagem de Objetos)
- OOA - object-oriented analysis (análise orientada a objetos)
- OOAD - object-oriented analysis and design (análise e projeto orientados a objetos)
- OOD - object-oriented design (projeto orientado a objetos)
- OOP - object-oriented programation (programação orientada a objetos)
- OOSE - Object-Oriented Software Engineering (Engenharia de Software Orientada a Objetos)
- PCO - projeto do comportamento do objeto
- PEO - projeto da estrutura do objeto
- SDL - Specification and Description Language (Linguagem de Especificação e Descrição)
- SGBD - sistema de gerenciamento de base de dados
- VDM - Viena Development Method (Método de Desenvolvimento de Viena)

Lista de figuras

Lista de tabelas

Resumo

Este trabalho apresenta uma visão geral de cinco metodologias de desenvolvimento de aplicações orientadas a objetos. Descreve a ênfase adotada nas fases de análise e projeto, os modelos usados e os passos para a construção de uma descrição de sistema. Um exemplo ilustra a aplicação de cada metodologia. A partir da descrição e dos exemplos desenvolvidos, é procedida uma avaliação comparativa das metodologias. São apresentadas ainda três metodologias de desenvolvimento de frameworks.

A partir da análise das metodologias de desenvolvimento de frameworks, observa-se a ausência de técnicas de modelagem e de um processo de desenvolvimento detalhado. Tais características fazem parte das metodologias de desenvolvimento de aplicações orientadas a objetos. Assim, a partir da análise destas metodologias, discute-se como adaptar suas técnicas de modelagem para uso no desenvolvimento de frameworks. Percebe-se a necesssidade de estender as técnicas de modelagem estática com a classificação de classes e métodos, e ressaltar a ocorrência de colaborações na modelagem dinâmica.

Palavras chave:

Engenharia de Software;
metodologias de desenvolvimento de software;
framework orientado a objetos;
análise orientada a objetos;
projeto orientado a objetos.

Abstract

This work presents an overview of five object-oriented application development methods. It describe the emphsis adopted in analysis and design phases, the models used, and the steps to construct a system description. An example ilustrates the application of each method. From descriptions and developed examples, is made a comparative avaliation of the methods. It's presented other three methods of frameworks development.

The analysis of some framework development methodologies, shows the lack of modelling techniques and a detailed development process. In contrast, this features are contained in most object-oriented application development methodologies. Starting with analysis of these methodologies, is discussed how to adapt their modelling techniques to be used in frameworks development. The perceived need is to extend the static modelling techniques with class and method categorization, and to stress the use of colaborations for the dynamic modelling.

Keywords:

Software Engineering;
software development methodologies;
object-oriented framework;
object-oriented analysis;
object-oriented design.

1 Introdução

A reutilização de componentes de software em larga escala é um dos argumentos a favor da abordagem de orientação a objetos. Em muitos casos, constitui uma perspectiva frustrada, pois a reutilização não é característica inerente da orientação a objetos, mas deve ser obtida a partir do uso de técnicas que produzam software reutilizável [JOH 88] [PRE 95]. Os frameworks orientados a objetos[1] são estruturas de classes interrelacionadas, que permitem não apenas reutilização de classes, mas minimizam o esforço para o desenvolvimento de aplicações - por conterem o protocolo de controle da aplicação (a definição da arquitetura), liberando o desenvolvedor de software desta preocupação. Os frameworks invertem a ótica do reuso de classes, da abordagem bottom-up[2] para a abordagem top-down. A lógica de desenvolvimento parte da visão global da aplicação, já definida no framework, em direção aos detalhes da aplicação específica, que são definidos pelo usuário do framework. Assim, a implementação de uma aplicação a partir do framework é feita pela adaptação de sua estrutura de classes, fazendo com que esta que inclua as particularidades da aplicação [TAL 95].

Frameworks promovem reutilização de software em larga escala. Em contrapartida, produzir um framework é uma tarefa muito mais complexa, que produzir aplicações específicas. O início do desenvolvimento de um framework demanda profundo conhecimento do domínio de aplicações tratado. Isto requer a existência prévia de aplicações. O desenvolvimento de um framework demanda a generalização das características de um domínio de aplicações, para capacitá-lo a cobrir diferentes aplicações deste domínio. Além disso, o processo de desenvolvimento de frameworks precisa ser iterativo e incremental: precisa ser refinado ao longo de ciclos de desenvolvimento, e deve ter a capacidade de assimilar novas informações do domínio, capturadas ao longo da utilização do framework. A aceitação de um framework requer o seu teste no desenvolvimento de diferentes aplicações. Todos estes aspectos exigem um esforço de desenvolvimento bastante superior ao necessário, para o desenvolvimento de uma aplicação específica [JOH 93].

Metodologias de desenvolvimento de frameworks ora propostas, descrevem como produzir frameworks, sem estabelecer um processo detalhado de construção de modelos que dirija o desenvolvimento [JOH 93] [TAL 94] [PRE 95]. Também não estabelecem um conjunto de técnicas de modelagem para a descrição do projeto do framework - a modelagem do domínio de aplicações. A documentação proposta pelos autores de metodologias abrange fundamentalmente a questão de como utilizar os frameworks. Mecanismos como cookbooks, contratos e patterns [PRE 95] são usados para descrever os aspectos de implementação do framework, cujo entendimento é necessário para o desenvolvimento de aplicações.

Várias metodologias de desenvolvimento de aplicações orientadas a objetos[3] vem sendo propostas, tendo como base a abordagem de orientação a objetos. Caracterizam-se por um conjunto de técnicas de modelagem e um processo de desenvolvimento. Admitem o uso de ferramentas para auxílio ao processo de

---

1 Também chamados frameworks de aplicação orientados a objetos, ou simplesmente, frameworks.

2 Desenvolvimento bottom-up é o que ocorre quando classes de objetos de uma biblioteca são interligadas, para a contrução de uma aplicação.

3 Metodologias de desenvolvimento de aplicações orientadas a objetos, que abrangem as etapas de análise e projeto serão tratadas daqui por diante coma metodologias OOAD.

desenvolvimento de aplicações, que podem ser integradas em ambientes de desenvolvimento. Por outro lado, as metodologias OOAD em geral, se atêm à produção de uma aplicação específica, sem se preocupar com o desenvolvimento de classes de objetos reutilizáveis, ou em buscar subsídios em bibliotecas de classes. Usam o encapsulamento de dados da orientação a objetos basicamente como um mecanismo de estruturação da especificação[4] em uma abordagem de desenvolvimento top-down.

No presente trabalho é desenvolvida uma análise de como aspectos de metodologias OOAD podem ser aproveitados, para o preenchimento das lacunas presentes nas metodologias de desenvolvimento de frameworks.

Cinco metodologias de desenvolvimento de software baseadas em orientação a objetos são apresentadas e avaliadas: de Coad e Yourdon [COA 92], [COA 93], OMT [RUM 94], OOSE [JAC 92], de Martin e Odell [MAR 95] e Fusion [COL 94]. Constituem uma amostra significativa no contexto das ramificações da orientação a objetos - abordagens dirigidas a dado, a evento e a cenário [FOW 94]. As duas primeiras e a última são dirigidas a dado; a terceira, a cenário e a quarta, a evento. Sua seleção foi feita com base em publicações que ressaltam seus méritos em relação a outras existentes [FOW 94], [MON 92], [PAS 94], [PER 95] e os resumos construídos com base nos livros que descrevem as metodologias[5].

No capítulo 2 é definido o contexto de trabalho, situando as metodologias de desenvolvimento de software orientadas a objetos no universo da Engenharia de Software; é feita a justificativa da escolha das quatro metodologias citadas, e é apresentado um enunciado de problema a ser modelado, com base nestas metodologias.

A descrição das metodologias OOAD é objeto dos cinco capítulos seguintes. As cinco metodologias são descritas usando um mesmo roteiro de apresentação. Inicialmente é feita uma descrição geral: são delineadas as etapas de análise e projeto, descrevendo sucintamente seus objetivos, em que informações se baseiam, que descrições produzem. A partir desta visão geral, são descritos de forma detalhada as técnicas de modelagem usadas pela metodologia: seus elementos sintáticos e os conceitos por eles representados. A fase seguinte é a apresentação das etapas de construção dos modelos, ou seja, os passos da análise e do projeto. Para ilustrar a descrição, é desenvolvida uma especificação de sistema a partir da metodologia. O mesmo enunciado de sistema (descrição informal) é usado para as cinco metodologias

No capítulo 8, com base nas descrições das metodologias e nos exemplos desenvolvidos, é feita uma avaliação comparativa das quatro metodologias OOAD, no contexto do desenvolvimento de aplicações.

No capítulo 9 são apresentados os conceitos associados a frameworks e três metodologias voltadas ao seu desenvolvimento - Projeto Dirigido por Exemplo [JOH 93], Projeto Dirigido por Hot Spot [PRE 95] e a metodologia da empresa Taligent [TAL 94]. É desenvolvida uma análise comparativa destas metodologias, destacando aspectos não cobertos pelas propostas, que estão presentes em metodologias OOAD.

No capítulo 10, é proposta uma aglutinação de características julgadas positivas das metodologias OOAD, para serem usadas no desenvolvimento de frameworks.

---

4 Em função das vantagens do tipo abstrato de dados, em relação aos módulos da abordagem funcional. Esta comparação é tratada em [MEY 88].

5 Algumas das publicações que fazem avaliações apresentam uma descrição sumária das metodologias, mas que não capacitam a seu uso. A descrição apresentada neste trabalho, complementada por exemplo de aplicação, tem a pretensão de capacitar ao uso das metodologias.

2 Definição de contexto - metodologias de desenvolvimento de software orientadas a objetos

2.1 Ciclo de vida

A existência de software passa por um conjunto de etapas denominado ciclo de vida. Cada etapa tem recebido denominações e subdivisões diferentes de diferentes autores, mas que correspondem às seguintes etapas [MON 92]:
Análise - envolve a definição e modelagem do problema;
Projeto - especificação e modelagem da solução;
Implementação - construção do sistema, a partir do projeto (incluída a etapa de testes);
Manutenção - alteração do software original.

Algumas abordagens tem sido adotadas ao longo das últimas décadas, para nortear o desenvolvimento de software [HOD 94]. O desenvolvimento orientado a função é uma abordagem em que um sistema é visto como um conjunto de funções. O desenvolvimento do sistema consiste numa decomposição progressiva de sua funcionalidade (em funções cada vez mais simples, que são implementadas). No desenvolvimento orientado a evento o sistema é particionado de acordo com os eventos que ocorrem no seu ambiente, e aos quais deve responder. O desenvolvimento orientado a dado volta sua atenção à definição de uma estrutura de dados - as suas funções são construídas em torno das abstrações de dados. Na abordagem de orientação a objetos um sistema é construído como um conjunto de entidades interativas, que encapsulam dados e operações que atuam sobre estes dados. A orientação a objetos se baseia nos conceitos de tipo abstrato de dados e ocultação de informação. O presente trabalho se aterá a metodologias de desenvolvimento de software baseadas na abordagem de orientação a objetos.

2.2 A filosofia da orientação a objetos

A proposta da abordagem de orientação a objetos é atacar a questão da produção de software a partir de uma ótica menos funcionalista, isto é, gerar descrições mais próximas da realidade do domínio do problema - em contraste com uma visão próxima de funções que devam atuar sobre este domínio. As justificativas para isto são [COA 92]:
- a menor mutabilidade do domínio em relação a funções facilitará o processo de manutenção;
- esta constância do domínio gera descrições reutilizáveis para outras aplicações no mesmo domínio;
- requisitos neste tipo de descrição são mais claros às pessoas que atuam no domínio do problema, facilitando sua validação;
- transição natural de uma fase para outra, na medida em que a visão de conjunto de objetos se propaga desde a análise até a implementação.

Esta abordagem propõe formas mais próximas dos mecanismos humanos de gerenciar a complexidade inerente ao software - complexidade que se acentua nos sistemas de grande porte. A orientação a objetos se utiliza de abstração, assim como o ser humano - incapaz de dominar completamente o entendimento de algo, o ser humano procura ignorar detalhes não essenciais, se atendo a uma visão geral, por ele elaborada.

Neste esforço de busca de entendimento, o ser humano precisa classificar elementos (objetos) em grupos (classes), em alguma espécie de estrutura (agregação, herança, associações). Estes mecanismos são adotados pela abordagem de orientação a objetos [BOO 91].

2.3 Conceitos preliminares de orientação a objetos

Um **objeto** é "uma abstração de alguma coisa no domínio de um problema ou em sua implementação, refletindo a capacidade de um sistema para manter informações sobre ela, interagir com ela, ou ambos: um encapsulamento de valores de **atributos** e seus **métodos** (serviços, operações) exclusivos". Uma **classe de objetos** consiste na "descrição de um ou mais objetos através de um conjunto uniforme de atributos e métodos" [COA 93]. Um objeto é uma **instância** de uma classe de objetos.

Herança é uma relação entre classes em que uma classe herda a estrutura de dados e o comportamento definidos em uma ou mais classes (herança múltipla neste caso, e simples naquele). A herança estabelece uma estrutura hierárquica entre classes. Geralmente ocorre a alteração ou ampliação da estrutura de dados e do comportamento (métodos) herdados das **superclasses** (pela **subclasse**).

Uma **classe abstrata** é uma classe que não possui instâncias (objetos). É definida com a perspectiva de gerar subclasses, em que haverá alteração ou ampliação da estrutura de dados e do comportamento. Uma classe que não é abstrata é uma **classe concreta**.

A interação entre objetos se dá através da prestação de serviços, em que objetos **clientes** requisitam execução de métodos a objetos **servidores**. Este relacionamento se procede através de troca de **mensagens**, através das **interfaces** dos objetos, que constituem a sua única parte externamente visível. O princípio do **encapsulamento de dados** estabelece que nada além da interface de um objeto pode ser acessível ao meio externo.

A propriedade do **polimorfismo** presente em linguagens orientadas a objetos, permite que objetos de diferentes classes possam receber um mesmo formato de mensagem. Isto permite a padronização de interfaces quando da criação das classes de um domínio, o que facilita a reutilização de software, na abordagem de orientação a objetos.

2.4 O paradigma de orientação a objetos aplicado ao ciclo de vida do software

A análise orientada a objetos (OOA) modela o domínio do problema. Sua ênfase é a representação do problema. Uma de suas atividades é a identificação das classes de objetos do domínio. Dentre as metodologias que procedem análise orientada a objetos, há as que preconizam a identificação de atributos e métodos (por exemplo [COA 92]), algumas, apenas atributos (por exemplo [COL 94]) e outras ainda não tratam atributos e métodos, se atendo a operações e responsabilidades do sistema como um todo (por exemplo [MAR 95]).

O projeto orientado a objetos (OOD) foca sua atenção na definição de uma solução para o problema levantado na análise, ou seja definir uma estrutura de software.

O projeto modela o domínio de solução, o que inclui as classes de objetos do domínio do problema (com possíveis adições) e outras classes de objetos pertencentes ao domínio de solução, estas identificadas na fase de projeto. Uma implementação orientada a objetos consiste na definição de um conjunto de classes de objetos, com atributos e métodos. Assim, o projeto orientado a objetos deve definir completamente os atributos e métodos das classes, deixando para a etapa de implementação a responsabilidade de apenas traduzir as descrições de classe, para a linguagem alvo adotada.

"A programação orientada a objetos (OOP) é um enfoque de implementação em que programas são organizados como coleções de objetos que cooperam entre si, em que cada um representa uma instância de alguma classe, e cujas classes são todas membros de uma hierarquia de classes unidas através de relações de herança" [BOO 91]. A OOP pode ser suportada por linguagens orientadas a objetos, como Smalltalk ou C++, ou por linguagens não orientadas a objetos, como C ou Ada - sendo neste caso uma tarefa mais complexa manter o princípio da OOP, pela falta de mecanismos adequados na linguagem.

2.5 Modelos e técnicas de modelagem

As informações obtidas nos procedimentos de análise e projeto são registradas em modelos. Um modelo é uma descrição de alguma coisa, segundo uma linguagem. Uma técnica de modelagem usada para a construção de modelos, apresenta uma visão particular de um sistema - ênfase à descrição dos dados, ou comportamento, ou arquitetura etc. Uma linguagem formal para a construção de modelos é baseada em formalismo matemático, e permite que propriedades do modelo sejam verificadas a partir de procedimentos algébricos.

Há dois aspectos básicos a considerar na escolha de uma técnica: formalismo e compreensibilidade. Uma técnica de modelagem formal como VDM ou LOTOS, permite produzir modelos absolutamente sem ambigüidades e que podem ser validados formalmente. A questão da compreensibilidade se reflete na maior ou menor facilidade de entendimento dos modelos produzidos. Os dois aspectos são importantes, porém em geral são antagônicos [FOW 94]. Por exemplo, uma descrição de sistema em LOTOS pode ser de difícil compreensão para o usuário do software em desenvolvimento, enquanto um diagrama ER (que é formal, porém de expressividade limitada) pode ser mais fácil de entender, mas demanda um dicionário de dados, que pode conter ambigüidades (por não ser formal).

2.6 Metodologias de desenvolvimento de software

Metodologia é um procedimento sistemático de geração de modelos, que descrevem aspectos de um sistema sob desenvolvimento. Uma metodologia de desenvolvimento de software inclui um conjunto de técnicas de modelagem e a seqüência de passos para gerar uma descrição de sistema (processo de desenvolvimento) [FOW 94]. Uma vez que uma técnica de modelagem é voltada a enfatizar um aspecto da realidade descrita (estrutura de dados, comportamento etc.), uma metodologia necessita de mais de uma técnica para descrever um sistema. É variável nas metodologias a quantidade de técnicas de modelagem usadas e a finalidade de cada uma. Uma técnica

de modelagem não é exclusiva de uma metodologia - pode ser usada por metodologias diferentes. Outra característica que diferencia as metodologias é o uso de técnicas de modelagem diferentes - mais ou menos expressivas; mais ou menos formais - para a descrição de um mesmo aspecto de um sistema (seu comportamento dinâmico, por exemplo).

Além do conjunto de técnicas de modelagem adotado, uma metodologia é caracterizada pelo seu processo de desenvolvimento (procedimentos de construção de modelos). As metodologias orientadas a objetos, em função da ênfase adotada na construção de modelos, se classificam em dirigidas a dado, dirigidas a evento e dirigidas a cenário. As metodologias orientadas a objetos, dirigidas a dado, adotam como primeiro procedimento a identificação de classes de objetos. A ênfase das metodologias dirigidas a evento é identificar e descrever em primeiro lugar, os eventos que podem ocorrer, envolvendo o sistema - as demais informações, como as classes, são obtidas em função dos eventos. Nas metodologias dirigidas a cenário são identificadas inicialmente todas as situações de processamento a que o sistema pode ser submetido, e descritos os possíveis diálogos entre sistema e meio externo (cenários). O restante da descrição do sistema é feito em função do conjunto de cenários.

Independente da ênfase adotada, uma metodologia de análise ou projeto orientada a objetos descreverá um sistema como um conjunto de objetos (pertencentes a classes) que interagem. Assim, o conjunto de técnicas de modelagem deve suprir dois requisitos: descrição completa de cada classe de objetos e descrição da relação entre as classes (como são combinadas para gerar o sistema).

Em [HOD 94] alguns requisitos são estabelecidos para uma descrição de sistema. Primeiramente é estabelecido que uma descrição (baseada em orientação a objetos ou não) deve abordar três pontos de vista: visão da estrutura de dados, visão comportamental e visão funcional. Para uma descrição baseada em orientação a objetos estabelece um conjunto de modelos que julga desejáveis:

- Modelo essencial: uma descrição não ambígua do sistema[6] que satisfaz os requisitos estabelecidos;
- Cenários de uso: descrição da interação entre objetos, em resposta a um evento;
- Modelo de objetos: relação entre objetos;
- Modelo de interação: expressa a totalidade das comunicações entre objetos, sem seqüenciamento;
- Modelo de seqüência de eventos: expressa de forma ordenada (no tempo) a seqüência de interações entre objetos - um diagrama formaliza total ou parcialmente um cenário de uso;
- Modelo dinâmico: diagramas de transição de estados descrevendo o comportamento dinâmico dos objetos - seus estados e as mudanças de estados em função dos eventos;
- Hierarquia de uso: expressa a hierarquia de controle entre objetos;
- Hierarquia de criação: um grafo expressando a hierarquia de criação de instâncias;
- Grafo de classes: relação entre classes;
- Definição de classe: os detalhes de implementação de uma classe;
- Especificação de interface: visão externa das classes - o que é visível ao meio externo ao objeto.

Esta relação de modelos é passível de críticas, assim como o conjunto de técnicas de modelagem das várias metodologias existentes. Na proposta de Hodgson

---

6 Como um todo, e não como conjunto de classes.

não está definido por exemplo, em que modelo é descrita a interface do sistema, ou ainda, que não apresenta justificativa para o uso de dois modelos para a descrição de cenários (senão o fato disto ser feito pela metodologia OOSE).

Em termos das possíveis visões de um sistema, será considerado no presente trabalho que uma técnica de modelagem pode ser voltada ao aspecto estático ou dinâmico (principalmente a um aspecto, e não exclusivamente). A modelagem estática é responsável pela descrição da estrutura de componentes de um sistema (descrição dos componentes e da sua organização). A modelagem dinâmica se refere a dois aspectos: descrição comportamental, que se preocupa com o fluxo de controle do sistema (seqüências de procedimentos), e descrição funcional, que se preocupa em descrever os fluxos de dados e suas transformações.

2.7 Seleção de metodologias de desenvolvimento de software, para avaliação

No presente trabalho serão descritas as metodologias de análise projeto de Coad e Yourdon [COA 92], [COA 93], OMT [RUM 94], OOSE [JAC 92], Fusion [COL 94] e de Martin e Odell [MAR 95] - sendo esta última apenas de análise. Um mesmo requisito de sistema será usado para geração de uma especificação, segundo cada metodologia. A partir da descrição das metodologias e dos exemplos gerados, será procedida uma avaliação de cada uma.

A escolha destas metodologias foi feita com base em um conjunto de publicações que procedem comparações entre metodologias e destacam estas como estando entre as que apresentam mais características desejáveis em relação a outras [FOW 94], [MON 92], [PAS 94], [PER 95].

Em [FOW 94] são descritas as características das metodologias citadas (exceto Fusion). Esta publicação motivou a inclusão da metodologia de Martin e Odell no presente trabalho, mais por ser um exemplo de abordagem dirigida a evento que por seus méritos - a metodologia não é citada por nenhuma das outras publicações. Em [MON 92] destacam-se as metodologias OMT, de Coad e Yourdon e a de Booch [BOO 91] - esta última não tratada neste trabalho. Em [PER 95] destacam-se as metodologias OMT, OOSE e Fusion [COL 94]. Em [PAS 94] destacam-se entre as metodologias orientadas a objetos, OMT, OOSE e de Coad e Yourdon.

2.8 Definição de um sistema a ser especificado a partir das metodologias

2.8.1 Enunciado do problema Jogo "Corrida" - descrição das regras

O jogo denominado "Corrida" consiste num dos mais simples jogos envolvendo dado, peões e tabuleiro. O jogo de que poderão participar dois ou mais jogadores, consiste em deslocar um peão ao longo das casas de um tabuleiro, da posição inicial, à posição final. O jogador que chegar primeiro à posição final é o vencedor. O número de casas do tabuleiro que um jogador desloca o seu peão é definido a partir do lançamento de um dado.

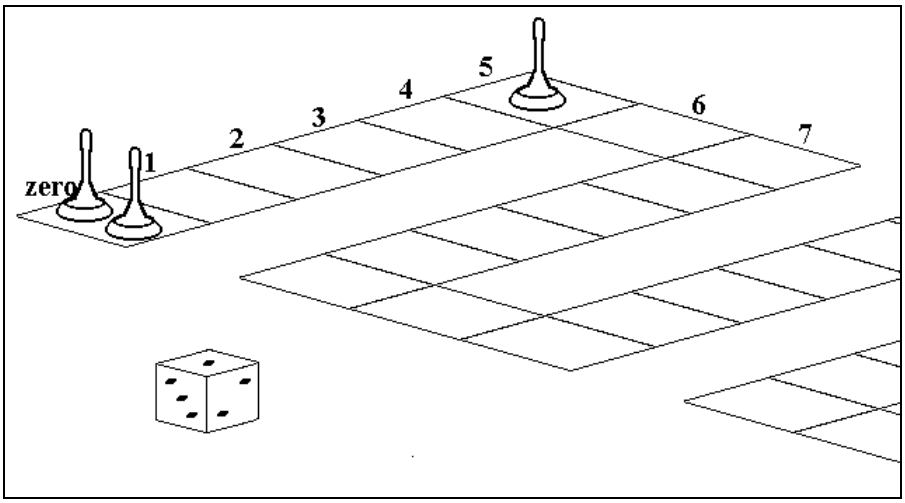


Figura 2.1 - ilustração do jogo "Corrida"

O tabuleiro define um percurso linear (sem bifurcações) e tem um determinado número de "casas" numeradas. A posição inicial corresponde ao número zero e para sair dela, o jogador deve obter no dado o valor 1 ou o valor 6. Com qualquer dos dois valores o jogador passa para a posição 1 (casa 1); com qualquer outro valor, permanece na casa zero. Os demais movimentos ao longo da partida consistem em deslocar o peão de um número de casas igual ao número obtido no dado. Quando o jogador obtiver no dado um valor que deslocaria seu peão para além da última casa, não é procedido movimento (o peão permanece na casa em que estava antes do lançamento do dado). Sempre que o jogador obtiver o valor 6 no dado, terá direito a realizar mais um lance (além daquele que esta sendo realizado, e em que foi obtido o valor 6).

O jogador que realiza o primeiro lance é aquele que em um sorteio prévio, tirar o número mais elevado num lançamento de dado. A ordem dos demais acompanha a ordem decrescente dos valores obtidos no dado. Empates devem ser resolvidos por um novo lançamento de dados (envolvendo apenas os jogadores empatados, para resolver o que fica na posição de ordenamento disputada - o perdedor, no caso de dois, fica na posição de ordenamento seguinte).

2.8.2 Requisitos para um sistema que implemente o jogo corrida.

- Apresentação no vídeo dos elementos do jogo: dado e tabuleiro, com peões posicionados;
- Diferenciação visual dos peões, que os identifique;
- Interferência do usuário através de teclado ou mouse;
- O sistema deve antes de uma partida, perguntar ao usuário o número de jogadores e a dimensão do tabuleiro;
- O sistema deverá proceder a determinação de quem inicia a partida;
- O sistema deverá determinar o jogador que deve realizar um lance e solicitar que jogue;
- O sistema deverá informar quando um jogador for vencedor e indicar o final da partida;
- O valor obtido no lançamento de um dado deve ser apresentado no vídeo;
- Uma partida deve poder ser interrompida a qualquer instante.

Observação: nas especificações geradas a partir das cinco metodologias é subentendido o uso de um ambiente de desenvolvimento que possua uma biblioteca de classes para a construção da interface com o usuário (como Visual Works ou Visual Age, para Smalltalk). Nas especificações há uma classe de interface que fará a conexão das classes desta biblioteca com as demais classes. Para efeito de simplificação, o detalhamento dos algoritmos desta classe e as classes da biblioteca não estão incluídos nas especificações.

3 Metodologia de Coad e Yourdon

3.1 Visão geral da metodologia

A metodologia de Coad e Yourdon [COA 92] [COA 93] é classificada como uma metodologia orientada a objetos, dirigida a dado. A partir da necessidade estabelecida e do conhecimento do domínio de aplicação, é construído um modelo do domínio (análise). A evolução da descrição da análise para o projeto se dá a partir da inclusão ao modelo, de classes pertencentes ao domínio da solução computacional. A mesma notação é utilizada na análise e no projeto. Com isto, os autores pretendem evitar o gap semântico entre notações de análise e projeto, presente em algumas metodologias (na análise e projeto estruturados, por exemplo[7]).

A metodologia utiliza como principal ferramenta de descrição, um modelo de objetos. Como mecanismo auxiliar, utiliza "especificações de classe-&-objeto" para detalhes não representados no modelo de objetos, como modelagem dinâmica (incluindo descrição dos algoritmos dos métodos). Estas "especificações de classe-&-objeto" misturam descrição textual, com técnicas de modelagem gráficas.

3.1.1 Análise e projeto

A metodologia de Coad e Yourdon estabelece uma seqüência de atividades para análise e projeto, cuja ordem pode ser trocada, e que prevê iteratividade, ou seja, é esperado que se volte a etapas anteriores para refinar o então eleborado.

As atividades da análise e do projeto são parecidas, no sentido de quais os passos compõem cada etapa. O que diferencia análise e projeto é a matéria tratada. Na análise buscam-se as classes exclusivamente do domínio do problema, e a partir delas são compostos os modelos da metodologia. A análise se concentra em definir os objetos do domínio do problema, que são elementos ou conceitos que representam a descrição do problema (independente de preocupação com sua solução) - por isto estes objetos são chamados objetos semânticos.

No projeto buscam-se as classes do domínio da solução computacional. As classes do domínio da solução buscadas durante o projeto, se agrupam em três componentes: componente interação humana, que são as classes responsáveis pela interface do sistema; componente administração de dados, responsável pelo gerenciamento, acesso, atualização dos elementos de armazenamento de dados, e por tornar estes procedimentos transparentes às outras partes; componente administração de tarefas, elemento do sistema responsável por monitorar e cadenciar as atividades dos demais componentes em tempo de execução[8]. A união destes três componentes com o

---

7 A análise estruturada trabalha com diagramas de fluxo de dados, e o projeto estruturado gera uma estrutura de módulos (a implementar) e o fluxo de controle entre eles. A passagem da especificação da análise para a especificação do projeto é complexa, pois não há uma correspondência (em termos de semântica) entre os diferentes modelos de cada etapa. Coad e Yourdon pretendem evitar a existência de gap semântico entre análise e projeto usando as mesmas técnicas de modelagem na análise e projeto. Dentre as metodologias OOAD observa-se que algumas adotam esta prática, e que outras usam técnicas de modelagem em projeto diferentes das técnicas usadas na análise.

8 Também fazem parte deste componente as considerações de alocação de hardware e protocolos de dispositivos e sistemas externos.

componente domínio do problema - que é o produto da análise, com ou sem alterações ao longo do projeto - constituirá o projeto do sistema.

3.1.2 Técnicas de modelagem para a descrição de uma aplicação

A metodologia utiliza duas técnicas de modelagem para a descrição de um sistema, o modelo de objetos e a "especificação de classe-&-objeto".

As atividades da metodologia iniciam com a construção do modelo de objetos. As especificações de classe-&-objeto são compostas ao longo da construção do modelo de objetos. Esta técnica de modelagem se constitui de um mecanismo gráfico para a descrição das classes, objetos e seus relacionamentos. Os elementos que compõem o modelo de objetos são as classes ("classes-&-objetos" para representação de classes concretas, e classes, para representação de classes abstratas) os mecanismos de estrutura, que representam agregação (todo-parte) e herança (generalização-especialização), e os mecanismos de representação da interdependência entre classes - conexão de ocorrência e conexão de mensagem (conexões correspondem a associações entre classes).

A metodologia de Coad e Yourdon estabelece o não particionamento do modelo de objetos. Para tornar legível um diagrama que pode ter um grande número de classes, os autores utilizam a representação de assuntos. Representar um assunto no diagrama de objetos consiste em agrupar no interior de um polígono, as classes relacionadas por propriedades comuns, como funcionalidade, localização física. O modelo com assuntos consiste no conjunto de classes divididas em cercados.

As especificação de classe-&-objeto mistura descrição textual, com modelos gráficos. Ao contrário do modelo de objetos que descreve o sistema global, esta especificação descreve cada classe separadamente. Seus principais componentes são um dicionário de dados para uma descrição geral da classe e de seus atributos, descrição textual da interface da classe, diagrama de estado de objeto - que é um autômato finito que modela dinamicamente o comportamento de um objeto da classe - e descrição dos métodos (serviços) da classe - através de descrição textual e de um diagrama de serviço, que é um mecanismo gráfico semelhante a um fluxograma e que é responsável pela descrição do algoritmo associado ao método. Os autores sugerem que sejam acrescidas às especificações de classe-&-objeto, tantas informações adicionais (em forma textual) quanto se julgar necessário.

3.2 Elementos sintáticos das técnicas de modelagem da metodologia de Coad e Yourdon

3.2.1 Modelo de objetos

A figura abaixo reúne o conjunto de elementos sintáticos da metodologia de Coad e Yourdon, para a construção do modelo de objetos.

Uma classe abstrata é representada como um retângulo de cantos arredondados com três partes: nome da classe, atributos e métodos (serviços). Uma classe concreta tem uma representação semelhante, porém um segundo retângulo circunda o retângulo subdividido - esta representação é denominada "classe-&-objeto". O retângulo interno

representa a classe, e o externo, os objetos (instâncias da classe). As linhas que conectam os símbolos de classes diferenciam se a associação envolve classe ou instância de classe.

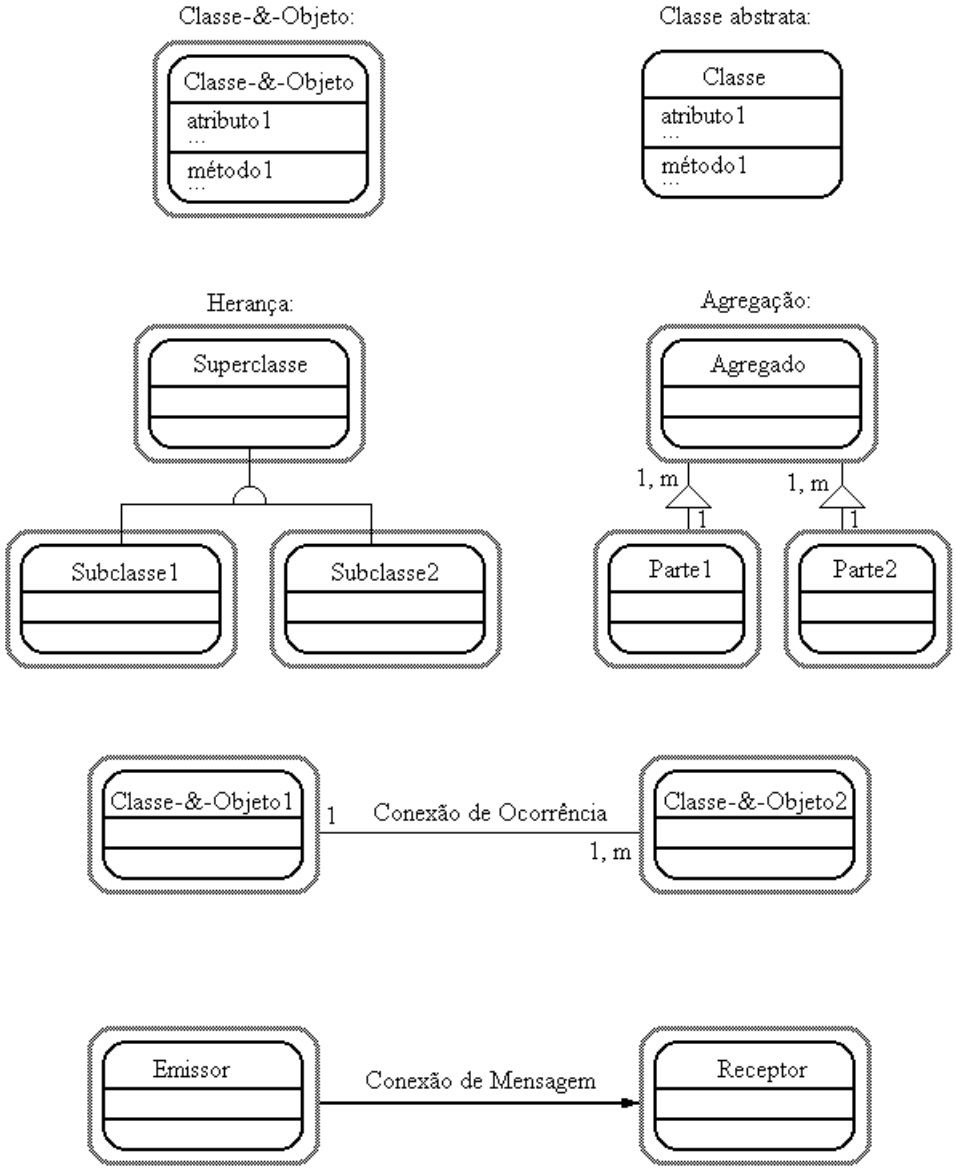


Figura 3.1 - elementos sintáticos do modelo de objetos

A representação de herança conecta classes e é direcional, isto é diferencia a superclasse das subclasses. A representação de agregação conecta duas instâncias, possui cardinalidade e é direcional, isto é, diferencia o agregado (todo) de sua parte (a seta aponta para para o agregado). Herança e agregação constituem o que a metodologia chama de estrutura do modelo de objetos, ou seja, a organização das classes.

O associação entre classes - ou seja, a manutenção de referência e a interação (cooperação) entre as instâncias da classe - é representada pelas conexões. A conexão de ocorrência, representada como uma linha com cardinalidade entre objetos, expressa a necessidade da existência de outros objetos para que um objeto possa cumprir suas responsabilidades, ou seja, expressa o conhecimento da existência de outro objeto, o que se reflete em atributos (que armazenam este conhecimento). Por exemplo, para que um objeto carro possa executar o método correr, é preciso que haja um objeto pista.

A conexão de mensagem, representada por uma seta ligando objetos (eventualmente uma mensagem pode ser enviada a uma classe, para a criação de uma instância, como ocorre em Smalltalk), expressa a comunicação cliente-servidor, para a execução de um método (prestação de um serviço). A conexão de mensagem modela a dependência de processamento. Por exemplo, segundo o paradigma de orientação a objetos, para um objeto conhecer o valor do atributo de outro objeto, ele solicita a este objeto, através de uma mensagem, a execução de um método que emita esta informação. Na sintaxe adotada pela metodologia as setas podem ser rotuladas ou não. Os autores recomendam a não-rotulação e que o significado da conexão seja buscado nas especificações de classe-&-objeto - o receptor possuirá método em resposta à mensagem e o emissor enviará mensagem. A não-rotulação permite uma especificação em um nível de abstração mais elevado que com a inclusão de um nome de método, explicitando que há a necessidade de comunicação entre os elementos, mas não detalhando as mensagens em si. Por outro lado, o significado da associação deve ser buscado no restante da especificação. Rotular as setas com nomes de mensagens deixa a especificação muito próxima da implementação, num nível de abstração mais baixo - o que pode ser conveniente para aplicações simples, porém, impraticável para aplicações complexas, com muitas mensagens. Uma rotulação que expresse a finalidade da associação sem se ater aos métodos, parece uma solução mais adequada a sistemas complexos.

3.2.2 Especificação de classe-&-objeto

O diagrama abaixo apresenta a estrutura da especificação de classe-&-objeto. As figuras 3.3 e 3.4 apresentam o conjunto de elementos sintáticos da metodologia de Coad e Yourdon, para a construção do diagrama de estado de objeto e do diagrama de serviço, respectivamente. Estes diagramas são parte da especificação de classe-&-objeto.

especificação

atributo
atributo
atributo ...

Entrada externa
Saída externa

Diagrama de estado de objeto

Especificações adicionais

notas

método <nome & Diagrama de serviço>
método <nome & Diagrama de serviço>
método <nome & Diagrama de serviço> ...

e, na medida do necessário,

Códigos de acompanhamento
Códigos de estado aplicáveis
Requisitos de tempo
Requisitos de memória

Figura 3.2 - modelo de especificação de Classe-&-Objeto

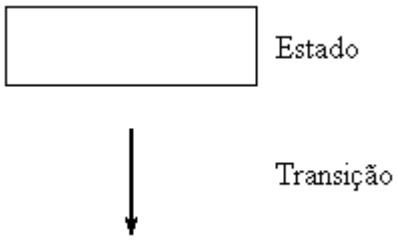


Figura 3.3 - elementos sintáticos do diagrama de estado do objeto

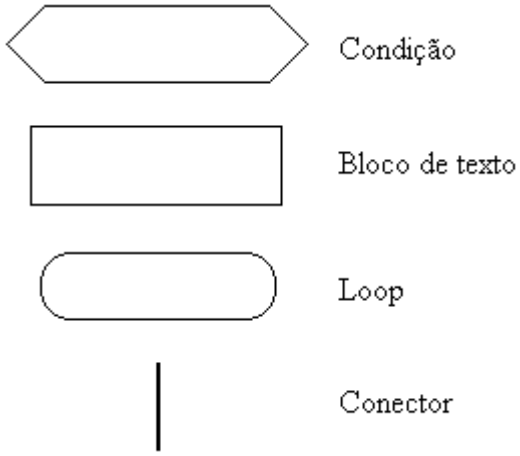


Figura 3.4 - elementos sintáticos do diagrama de serviço

O diagrama de estado de objeto é um grafo sintaticamente simples (é um autômato finito), que possui representação de estados e transições - ambas representações podendo ser rotuladas. Cada estado de um objeto é caracterizado pelo valor de seus atributos. Uma transição representa algum acontecimento (evento) que acarreta uma mudança de estado.

O diagrama de serviço é uma descrição semelhante a um fluxograma. possui elementos para descrição de atividade, execução condicional, repetição[9] (looping) de processamento e conector (que interliga os outros elementos). Se presta ao detalhamento do algoritmo do método. A metodologia admite o uso de descrição textual, ao invés do diagrama de serviço para métodos algoritmicamente simples - como por exemplo, a execução seqüencial de um conjunto de atividades.

---

9 Observa-se uma deficiência na notação de looping adotada na metodologia, que é a ausência de notação explícita, para o delimitador do bloco contido no looping. Para a representação desta informação no exemplo que será apresentado, um conector ligará o limite do bloco à lateral do símbolo de looping (que contém a condição).

3.3 As etapas de construção de uma especificação na metodologia de Coad e Yourdon

3.3.1 Passos da análise

- Determinar classes-&-objetos: a partir do conhecimento do domínio do problema, identificar suas classes (a princípio representadas num diagrama pelo símbolo de classe-&-objeto);

- Identificar estruturas: identificar relações de herança e agregação (nesta etapa podem surgir classes abstratas no diagrama);

- Identificar assuntos: separar as classes em grupos com propriedades comuns. Esta preocupação pode inexistir em aplicações de pequeno porte;

- Definir atributos: identificar os atributos associados aos objetos das classes e os valores que podem ser assumidos pelos atributos (e restrições). São identificadas as conexões de ocorrência. Nesta etapa começam a ser construídas as especificações de classe-&-objeto, a partir do dicionário de dados para a descrição dos atributos[10];

- Definir métodos. A partir desta etapa a especificação do sistema deixa de ser uma descrição exclusivamente estática e passa a ter incorporado o aspecto dinâmico. O modelo de objetos e as especificações de classe-&-objeto são concluídos. São tarefas relacionadas à etapa de definição de métodos:
  - Identificar os estados que o objeto de uma classe pode assumir e construir o respectivo diagrama de estado de objeto;
  - Identificar os métodos;
  - Identificar as conexões de mensagens e a interface das classes;
  - Especificar os métodos através de diagramas de serviço (ou através de descrição textual, para métodos simples).

Documento da análise: modelo de objetos e especificações de classe-&-objeto (uma para cada classe).

3.3.2 Passos do projeto

A análise trabalha e produz uma descrição sobre classes exclusivamente pertencentes ao domínio do problema. O projeto identifica e define classes adicionais, pertencentes ao domínio da solução. O conjunto das classes identificadas e descritas ao longo da análise e projeto comporá uma documentação formada de quatro componentes: componente domínio do problema, componente interação humana, componente gerenciamento de tarefas e componente gerenciamento de dados.

As classes do componente domínio do problema são basicamente o produto da análise, porém como haverá a necessidade de interação com classes que até então não existiam na descrição, poderão ser acrescentadas novas classes ao domínio do problema, ou as existentes poderão sofrer alterações.

---

10 Até este ponto a descrição do sistema é exclusivamente estática.

A elaboração do projeto consiste na construção dos quatro componentes, o que corresponde a quatro atividades distintas, que não são passos seqüenciais:

- Projetar o componente domínio do problema;
- Projetar o componente interação humana;
- Projetar o componente gerenciamento de tarefas;
- Projetar o componente gerenciamento de dados.

Estas atividades podem se desenvolver em paralelo, desde que respeitadas as restrições estabelecidas pelas dependências entre elas. Projetar cada componente consiste em percorrer os passos descritos na análise, ou seja[11]:

- Determinar classes-&-objetos.
- Identificar estruturas.
- Identificar assuntos.
- Definir atributos.
- Definir métodos.

3.4 Exemplo de uso da metodologia de Coad e Yourdon[12]

O exemplo será desenvolvido conduzindo as atividades de análise e projeto em paralelo (uma seqüência admitida pela metodologia).

1 - Identificar classes:
Componente domínio do problema (análise): Dado, Peão, Tabuleiro;
Componente gerenciamento de tarefas: CoordCorrida (gerencia a seqüência de lances);
Componente interação humana: InterfaceCorrida (classe citada no enunciado do problema);
Componente gerenciamento de dados: a aplicação não demanda gerenciamento de estruturas de dados.

2 - Identificar Estruturas:
Analisando o problema sob uma ótica de maximização de reutilização, Tabuleiro pode ser visto como uma classes abstrata (muitos jogos diferentes usam peões, dados e diferentes tabuleiros) e uma subclasse TabuleiroCorrida descreveria o tabuleiro da aplicação particular[13]. Não há classes originadas a partir de agregação.

3 - Identificar assuntos: como o número de classes envolvidas no exemplo é pequeno, optou-se por não proceder sua divisão em "assuntos"[14].

---

11 De fato, na descrição do projeto orientado a objetos, os autores colocam a identificação de assuntos como a primeira atividade - porém, como os próprios autores ressaltam, este ordenamento não deve ser tomado como regra absoluta.

12 Na especificação a seguir, para efeito de simplificação, não serão incluídos métodos referentes a criação de instância, informe de valor de atributo, atribuição de valor de atributo (exceto quando for relevante para dar significado à conexão de mensagem).

13 Algo semelhante poderia ser feito com Peão (peças de xadrez, por exemplo, são "peões" diferenciados) ou com Dado (dados podem ser tetraédricos, dodecaédricos ou outras formas, além de cúbicos). Para simplicidade da apresentação, não será feito.

14 Eventualmente, poderia ser feita a separação em função dos três componentes de projeto desta aplicação (interação humana, domínio do problema e gerenciamento de tarefas).

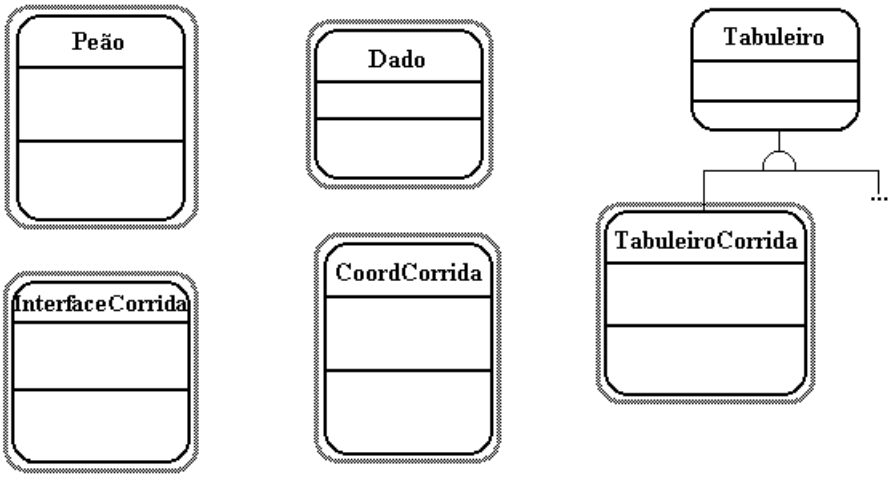


Figura 3. 5 - modelo de objetos inicial a partir da metodologia de Coad e Yourdon

4 - Definir atributos:

CoordCorrida:
- tabuleiro: referência ao tabuleiro do jogo (a instância de TabuleiroCorrida é criada por solicitação do coordenador, que possui o tabuleiro como atributo);
- dado: referência ao dado do jogo;
- p1, ..., pn: referência aos peões que participam do jogo (lista ordenada segundo a ordem de ocorrência de lances);
- Interface: referência à interface;
- numJogadores: contém o número de jogadores de uma partida (inteiro);
- quemJoga: referência ao peão que deve proceder o lance;

Tabuleiro:
- topologia: descrição da topologia do tabuleiro (no caso do jogo corrida, é um percurso linear sem características adicionais; no caso de tabuleiros mais complexos deve conter a descrição de suas características);

TabuleiroCorrida:
- numCasas: contém o valor associado à última casa, informação que identifica o número de casas do tabuleiro: numCasas + 1 (devido à casa zero);

Dado: não possui atributos;

Peão:
- identificador: identifica o jogador a que está associado o peão (string);
- posição: posição ocupada pelo peão no tabuleiro (inteiro);

InterfaceCorrida: em função da observação feita no enunciado do problema, nesta especificação não serão associados atributos a esta classe; ela simplesmente responderá às solicitações de métodos;

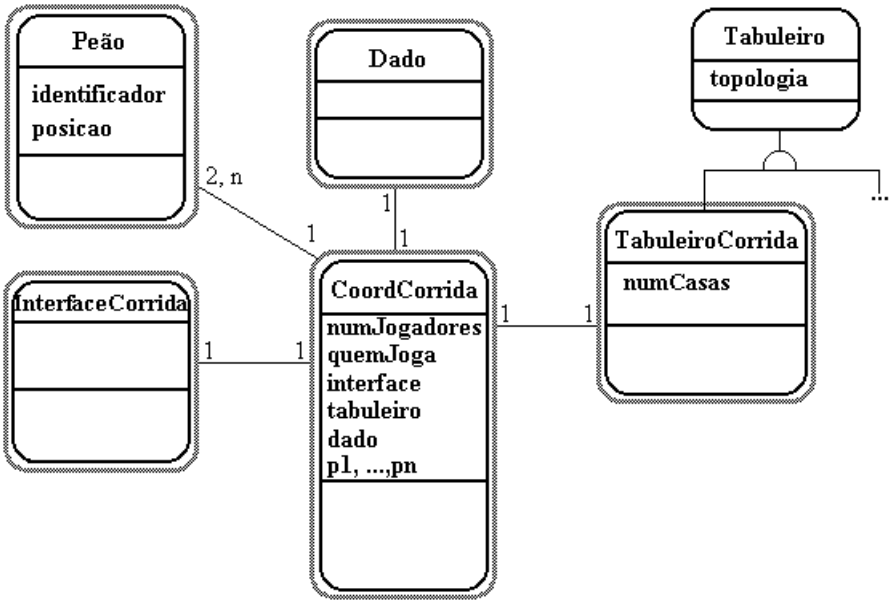


Figura 3. 6 - modelo de objetos anterior à definição de métodos

5 - Definir métodos: construir as especificações de classe-&-objeto para cada classe, procedendo a modelagem do comportamento dos objetos - o que produz a identificação dos métodos e conexões de mensagem, que completarão o modelo de objetos. A seguir será apresentado o produto desta etapa, que constitui o documento de projeto: modelo de objetos completo e conjunto de especificações de classe-&-objeto.

Modelo de objetos:

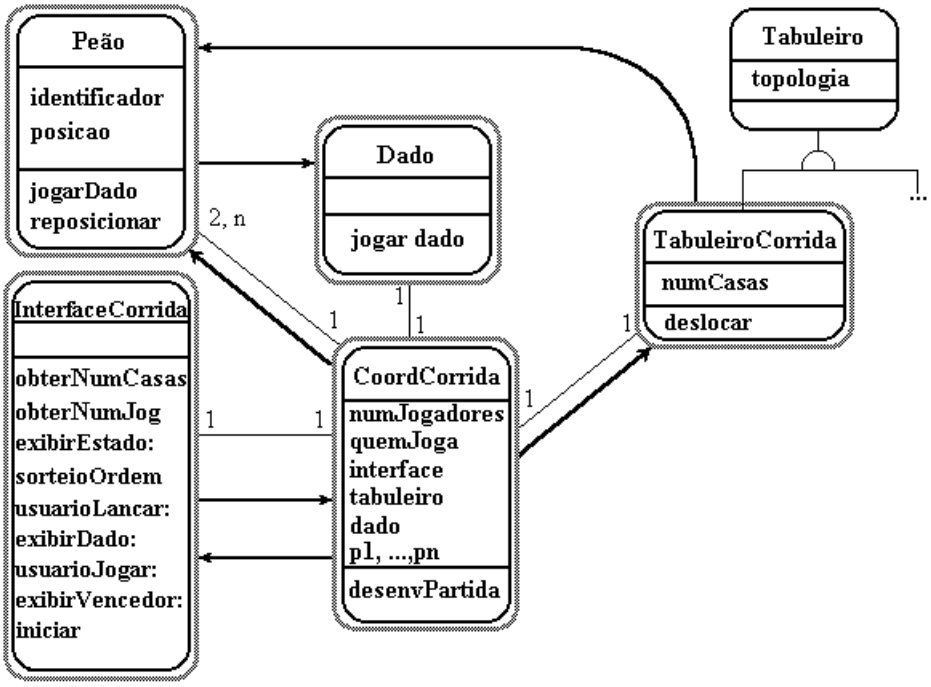


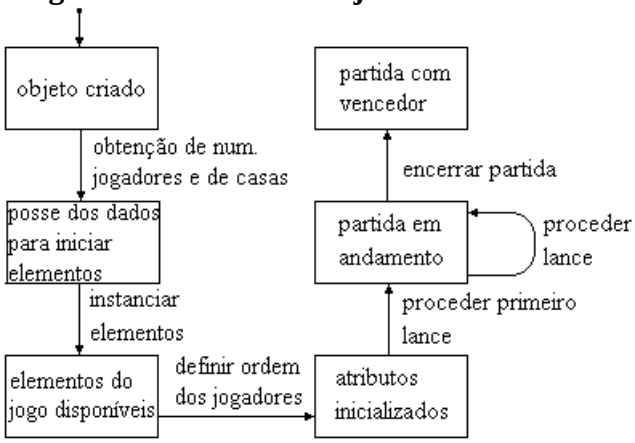
Figura 3. 7 - modelo de objetos final

especificação CoordCorrida

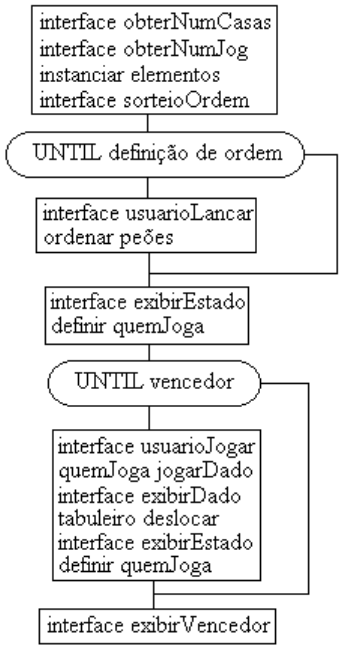
atributo tabuleiro: referência ao tabuleiro do jogo (a instância de TabuleiroCorrida é criada por solicitação do coordenador, que possui o tabuleiro como atributo)
atributo dado: referência ao dado do jogo
atributo p1, ..., pn: referência aos peões que participam do jogo
atributo interface: referência à interface
atributo numJogadores: contém o número de jogadores de uma partida (inteiro)
atributo quemJoga: contém o identificador do peão que deve proceder o lance

Entrada externa: solicitações e respostas a solicitações do usuário, respostas a solicitações feitas às demais classes
Saída externa: solicitações de métodos às classes

Diagrama de estado de objeto:



método desenvPartida:



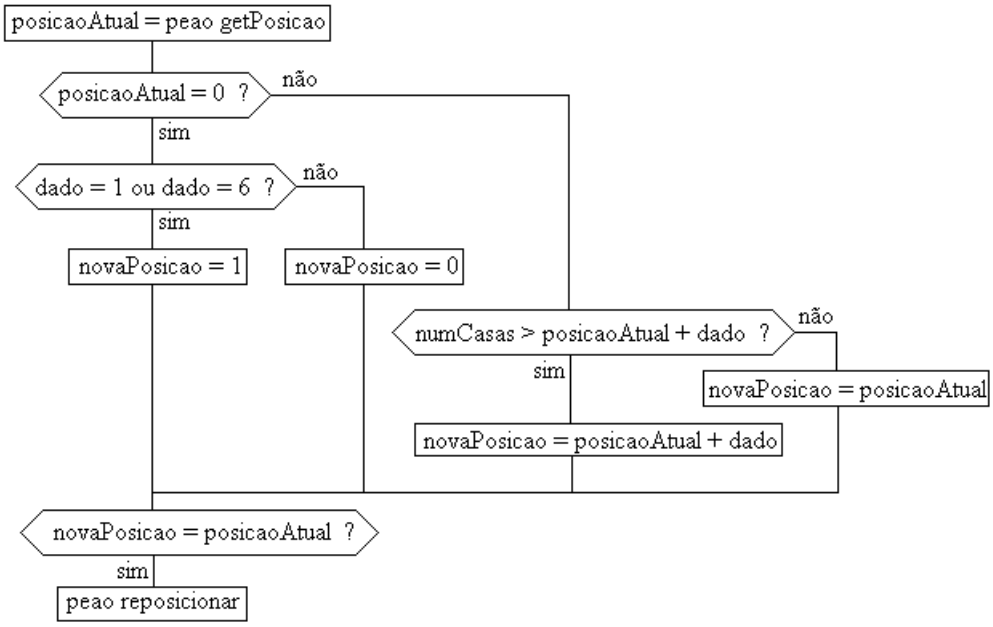
especificação TabuleiroCorrida

atributo numCasas: contém o valor associado à última casa, informação que identifica o número de casas do tabuleiro - numCasas + 1 (devido à casa zero)
atributo topologia: descrição da topologia do tabuleiro (no caso do jogo corrida é a constante *'linear'*)

Entrada externa: comando de reposicionamento de peão
Saída externa: peão reposicionado

Diagrama de estado de objeto: não há transição de estado (não há alteração de atributo, pois, são os peões que armazenam suas posições)

método deslocar:



especificação Dado

atributo: não possui atributos

Entrada externa: solicitação de lançamento de dado
Saída externa: número inteiro no intervalo 1 a 6

Diagrama de estado de objeto: não há transição de estado

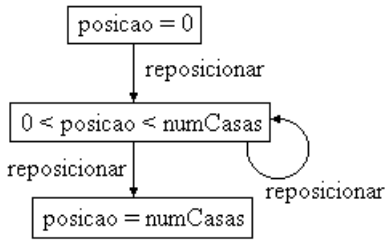
método jogarDado:
gerar valor aleatório entre 1 e 6;
retornar valor.

especificação Peão

atributo identificador: identifica o jogador a que está associado o peão (string)
atributo posição: posição ocupada pelo peão no tabuleiro (inteiro); inicializado com o valor zero

Entrada externa: comando de reposicionamento
Saída externa: valores de atributos

Diagrama de estado de objeto



método jogarDado:
dado jogarDado;
retornar valor.
método reposicionar:
atualizar posição, com o valor do argumento.

especificação InterfaceCorrida

atributos: não definidos[15].

Entrada externa: comandos e dados do usuário, comandos do coordenador
Saída externa: exibição da partida

Diagrama de estado de objeto: não definido.

método obterNumCasas: solicita ao usuário o número de casas do tabuleiro; retorna valor informado.
método obterNumJog: solicita ao usuário o número de jogadores da partida; retorna valor informado.
método sorteioOrdem: informa ao usuário que será procedido o sorteio para a definição da ordem dos jogadores (emite mensagem).
método usuarioLancar: solicita ao usuário autorização para proceder lançamento de dado; retorna autorização.

15 Como estabelecido no enunciado do problema, a especificação desta classe não será completa - apenas especificará genericamente os seus métodos.

método exibirDado: exibe o valor do argumento, na janela de representação do dado.
método usuarioJogar: solicita ao usuário autorização para proceder lance; retorna autorização.
método exibirEstado: exibe o tabuleiro e os peões posicionados (posição corrente).
método exibirVencedor: informa o identificador do peão vencedor.
método iniciar: ativado pelo usuário, solicita método desenvPartida do coordenador (instância de CoordCorrida).

4 Metodologia OMT

4.1 Visão geral da metodologia

A metodologia OMT (Técnica de Modelagem de Objetos) [RUM 94] se propõe a gerar um projeto de sistema a ser implementado em linguagem orientada a objeto ou não. Para tanto, utiliza diferentes modelos de representação, tendo cada um maior ou menor importância, em função de aspectos como tipo de aplicação e linguagem de programação. Preconiza a construção de um modelo do domínio de aplicação tratado (análise) e na posterior adição a este modelo, dos detalhes de implementação (projeto). A mesma descrição do domínio do problema será usada ao longo da análise e projeto, sendo enriquecida em informações ao longo desta evolução.

A metodologia OMT utiliza três tipos de modelos para descrever um sistema: o modelo de objetos, que descreve a estrutura estática de dados do sistema, ou seja, os objetos e seus relacionamentos; o modelo dinâmico[16], que descreve a seqüência de interações entre os objetos do sistema e o modelo funcional, que descreve as transformações de valores dos dados procedidas pelo sistema.

4.1.1 Análise e projeto

A metodologia estabelece uma seqüência de etapas para elaborar uma descrição de sistema - análise, projeto e implementação - mas enfatiza que o processo é por essência iterativo, isto é, informações levantadas em determinadas etapas podem complementar ou alterar os produtos de etapas anteriores.

A análise parte de um enunciado do problema e gera uma descrição para o sistema - composta pelos três modelos da metodologia. Neste primeiro esforço o objetivo é entender e descrever o problema estabelecido. A ênfase da descrição produzida durante a análise é o que o sistema deve fazer e não como o sistema deve fazer. Os objetos presentes na descrição da análise são elementos e conceitos do domínio da aplicação e não, conceitos da solução computacional. Assim, o modelo de análise deve ser uma descrição acessível ao usuário, sem detalhes da solução computacional.

A etapa de projeto é dividida em duas subetapas, o projeto do sistema e o projeto dos objetos - desenvolvidas cronologicamente, nesta ordem. O projeto do sistema consiste numa descrição de alto nível (acima do nível dos objetos) da solução do problema. Nesta etapa o sistema é organizado em subsistemas (numa ótica basicamente funcional). Cada subsistema engloba aspectos do sistema que compartilham algumas propriedades comuns - funcionalidade, localização física etc. Em termos do produto da análise, um subsistema da etapa de projeto do sistema, consiste num pacote de classes interrelacionados, com uma interface bem definida e preferencialmente pequena, com outros subsistemas. É procedida a alocação dos subsistemas a elementos de software e hardware (a presente descrição não se aterá à questão de funções do sistema serem

---

16 No contexto de OMT (pela nomenclatura que usa), a modelagem dinâmica se atém à descrição comportamental (controle). No restante do texto, modelagem dinâmica se refere a dois aspectos: descrição comportamental, que se preocupa com o fluxo de controle do sistema, e descrição funcional, que se preocupa em descrever os fluxos de dados e suas transformações - conforme já estabelecido.

executadas por elementos de hardware, como por exemplo, leitura e tratamento de sinais).

É definida uma organização geral dos subsistemas a partir de modelos arquitetônicos puros ou híbridos. Os autores listam os modelos arquitetônicos em função do tipo de sistema. No modelo arquitetônico de transformação em lote, a transformação de dados é efetuada de uma vez sobre um conjunto de dados de entrada - os compiladores constituem um exemplo deste modelo. No modelo de transformação contínua a transformação de dados ocorre à medida em que valores de entrada se modificam - como exemplo, programas de monitoramento de processos. A ênfase do modelo de interface interativa é a interação externa, como ocorre em aplicações com interfaces gráficas, com opções de botões e menus a serem ativados. O modelo de simulação dinâmica engloba os sistemas que reproduzem o comportamento de elementos do mundo real, como simuladores e jogos. Os sistemas em tempo real englobam aplicações em que o fator tempo é preponderante, como em aplicações de controle de processos, comunicações. O modelo gerenciador de transações inclui os bancos de dados, em que a principal finalidade é o armazenamento, acesso e atualização de dados.

A partir da classificação da aplicação em um destes modelos - ou como uma combinação dos modelos - é definido o conjunto de subsistemas e o relacionamento entre eles (camadas horizontais, partições verticais ou combinações destas). Além disto, são tomadas decisões que dependem dos ambientes de desenvolvimento, de execução e da(s) linguagem(s) alvo, como identificação e alocação de elementos concorrentes, gerenciamento de depósitos de dados (uso de arquivos e bancos de dados), forma de manipulação de recursos globais (unidades físicas, arquivos, bancos de dados). O documento gerado nesta etapa é basicamente uma descrição da arquitetura do sistema, nos termos estabelecidos acima. Não se enquadra nos três modelos usados em OMT, mas um documento à parte, para orientar a evolução dos modelos gerados na análise, ao longo da etapa de projeto de objetos. A organização em subsistemas é representada em um diagrama de blocos, como na figura abaixo [RUM 94]. Para sistemas de pequeno porte, em que a questão da definição de subsistemas não tenha grande relevância, o procedimento desta etapa pode ser bastante trivial e rápido.

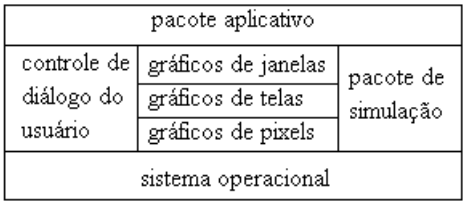


Figura 4.1 - representação dos subsistemas a partir de diagrama de blocos

O projeto dos objetos consiste na evolução dos modelos gerados na análise. Complementa a descrição daquilo que o sistema deve fazer (definida na análise) com o modo como o sistema deve fazer. Os detalhes são acrescidos em conformidade com a estratégia definida no projeto do sistema. As classes provenientes da análise são complementadas com estruturas de dados e algoritmos do domínio da solução computacional; novas classes pertencentes ao domínio da solução são acrescentadas aos modelos.

4.1.2 Técnicas de modelagem para a descrição de uma aplicação

Conforme já citado, a metodologia OMT descreve um sistema a partir de três modelos: o modelo de objetos, o modelo dinâmico e o modelo funcional.

O modelo de objetos estabelecido pela metodologia OMT é definido pelos autores, como uma tentativa de aperfeiçoamento da técnica de modelagem, diagrama de entidade-relacionamento (ER) [CHE 76]. Seus principais elementos[17] são as classes, com sua organização estrutural - representadas como retângulos contendo o nome da classe, seus atributos e métodos - e os relacionamentos entre as classes - representados como traços interligando classes; cada um dos dois elementos apresentando uma correspondência com os elementos do diagrama ER. O modelo de objetos, que descreve a estrutura estática do sistema, é o primeiro dos três modelos a ser construído. Os demais modelos porém, geram informações que o complementam.

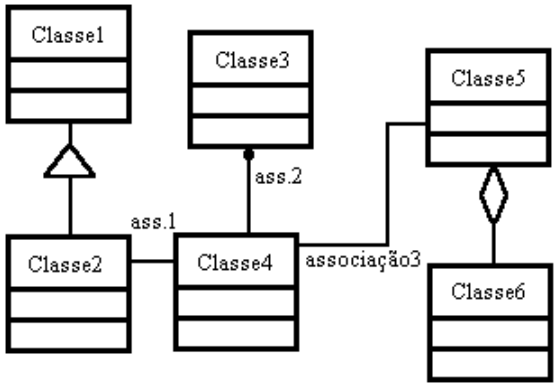


Figura 4.2 - exemplo de modelo de objetos

O modelo dinâmico descreve aspectos de controle e utiliza statechart [HAR 87], diagrama de eventos e diagrama de fluxo de eventos. Uma especificação de sistema contém um conjunto de diagramas de estados (statecharts) - um para cada classe, que apresente comportamento dinâmico relevante. Um diagrama de estados é composto por estados e transições, e modela o comportamento de uma classe como uma evolução de estados a partir da ocorrência de eventos. Os estados descrevem os valores dos atributos do objeto; as transições, a modificação do estado por causa de um evento envolvendo diferentes objetos - onde um objeto passa informação a outro.

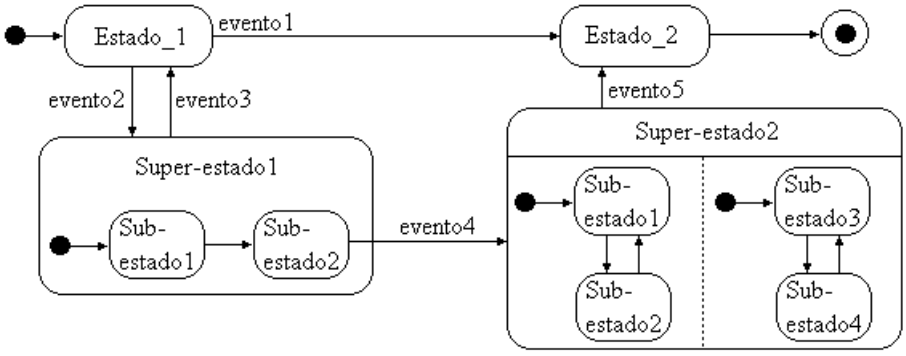


Figura 4.3 - exemplo de statechart

17 Uma descrição detalhada do conjunto de elementos sintáticos do modelo de objetos será feita no item a seguir.

Além dos diagramas de estados, que se referem a classes individuais, a construção do modelo dinâmico envolve diagramas de eventos, que descrevem cenários. Um cenário é uma descrição de situação a que o sistema pode ser submetido, como inicialização de dados ou resposta a um comando do usuário, e que explicita a interação entre classes - a descrição inclui os eventos que ocorrem e a ordem em que estes ocorrem. Na geração do modelo dinâmico, o primeiro passo consiste em gerar os diagramas de eventos e a partir deles, os diagramas de estados. Os diagramas de eventos são em OMT um mecanismo auxiliar para obtenção dos diagramas de estados e não, parte do modelo dinâmico.

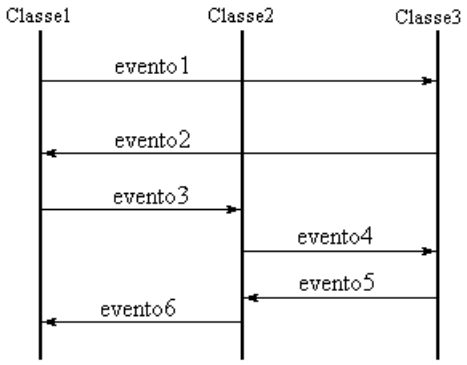


Figura 4.4 - exemplo de diagrama de eventos

O diagrama de fluxo de eventos é a outra técnica de modelagem que compõe o modelo dinâmico de OMT. Este diagrama fornece uma visão global do sistema e concentra em um diagrama, o conjunto de eventos enviados entre as classes. Assim, o modelo dinâmico é composto por duas técnicas de modelagem: os diagramas de estados, que descrevem cada classe individualmente, e o diagrama de fluxo de eventos, que descreve o sistema como um todo. O diagrama de eventos é usado como um mecanismo auxiliar para a busca de informações do sistema modelado e não é incluído pelos autores na relação de modelos que compõem a especificação.

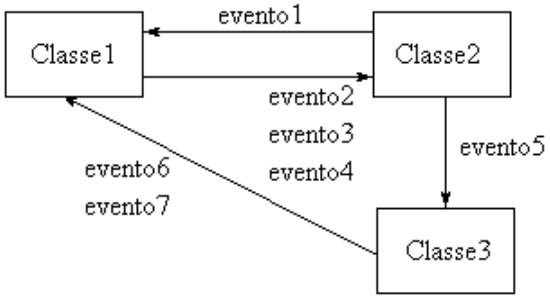


Figura 4.5 - exemplo de diagrama de fluxo de eventos

O modelo funcional se utiliza do diagrama de fluxo de dados (DFD), técnica de modelagem usada desde as metodologias baseados nas abordagens funcional e estruturada [DEM 79], [GAN 78], [YOU 89]. Enfatiza a descrição de cálculos e transformações de dados, sem a preocupação de como ocorrem (algoritmos), se ocorrem (procedimentos de decisão), em que ordem ocorrem (seqüenciamento) ou quem realiza (alocação de tarefas aos elementos do sistema). Um DFD é composto de processos, atores externos, depósitos de dados e fluxos de dados interligando estes elementos. Um processo do DFD constituirá um método de classe ou um fragmento de método. O DFD

constitui uma descrição da aplicação como um todo (semelhante aos diagramas de eventos) e não de cada classe, separadamente.

4.2 Elementos sintáticos das técnicas de modelagem da metodologia OMT

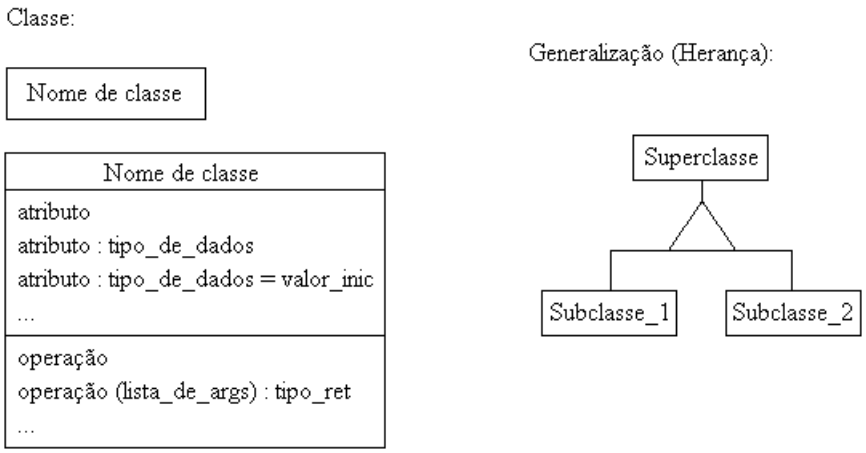
4.2.1 Modelo de objetos

As figuras 4.6 e 4.7 reúnem o conjunto de elementos sintáticos de OMT, para a construção do modelo de objetos.

Uma classe é representada como um retângulo com três partes: nome da classe, atributos e métodos (operações, serviços).

A representação de herança conecta classes e é direcional, isto é diferencia a superclasse das subclasses. De forma semelhante, a representação de agregação conecta classes e é direcional, isto é, diferencia o agregado (todo) de sua parte (o losângulo fica junto ao agregado).

Quanto às associações, constituem o único elemento do diagrama que estabelece um relacionamento entre classes. Às associações incluem-se rótulo - que descreve o significado da associação - e cardinalidade. Opcionalmente podem ser acrescidas as demais informações constantes nas figuras. A semântica das associações do modelo de objetos de OMT se assemelha à semântica dos relacionamentos dos diagramas ER - que constituem uma informação útil na construção de bancos de dados relacionais, mas que não encontram estrutura correspondente nas linguagens de programação baseadas em objetos. OMT, como salientam seus autores, dá grande ênfase às associações[18] - o que fica claro, comparando a carga de informação que podem conter as associações de OMT, com a das conexões de ocorrência e de mensagem, o equivalente às associações na metodologia de Coad/Yourdon.



18 Esta importância das associações de OMT podem constituir um obstáculo à implementação em linguagens de programação baseadas em objetos, na medida em que a inexistência de um mapeamento direto exige mais esforço durante a programação.

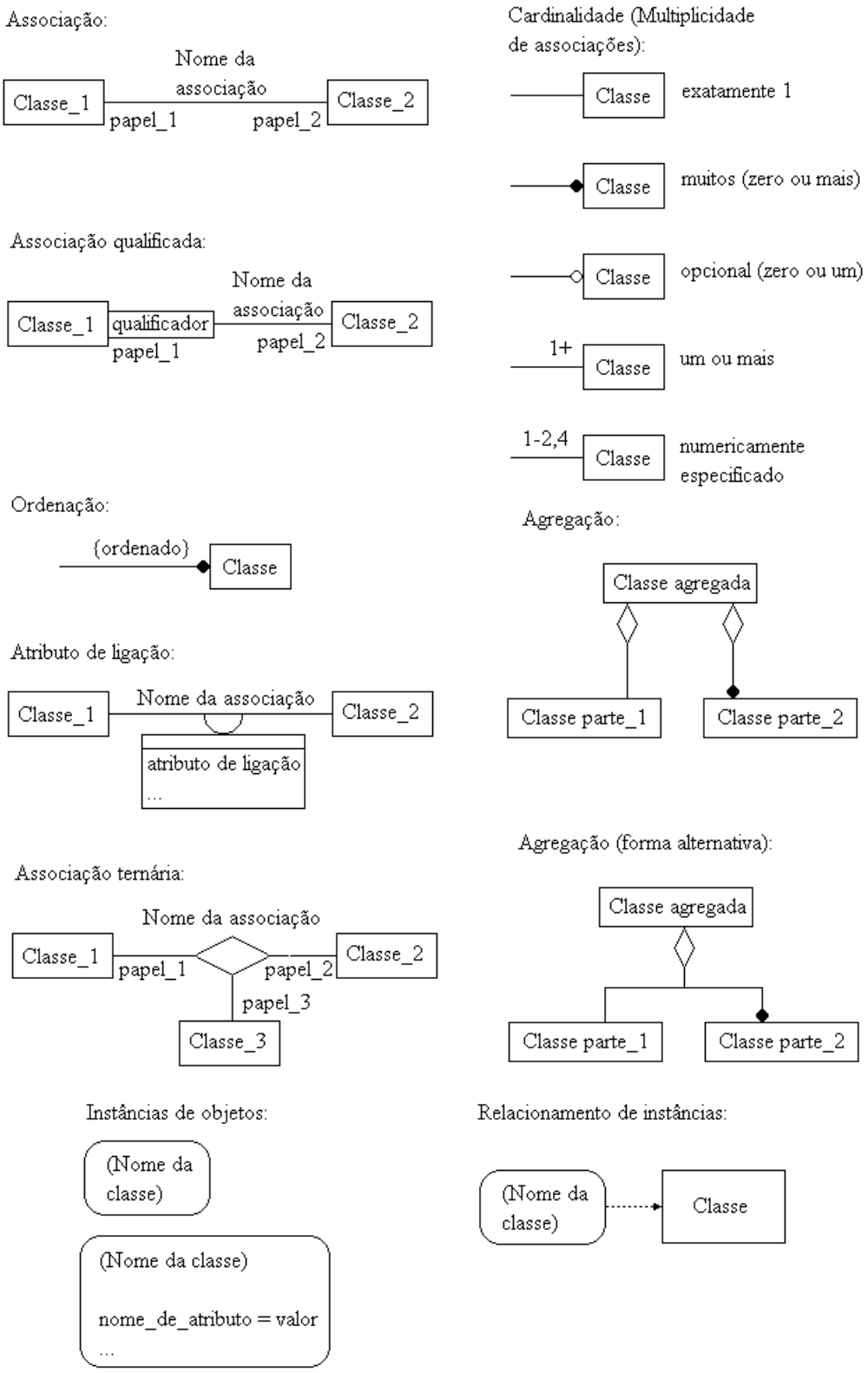


Figura 4.6 - elementos sintáticos do modelo de objetos

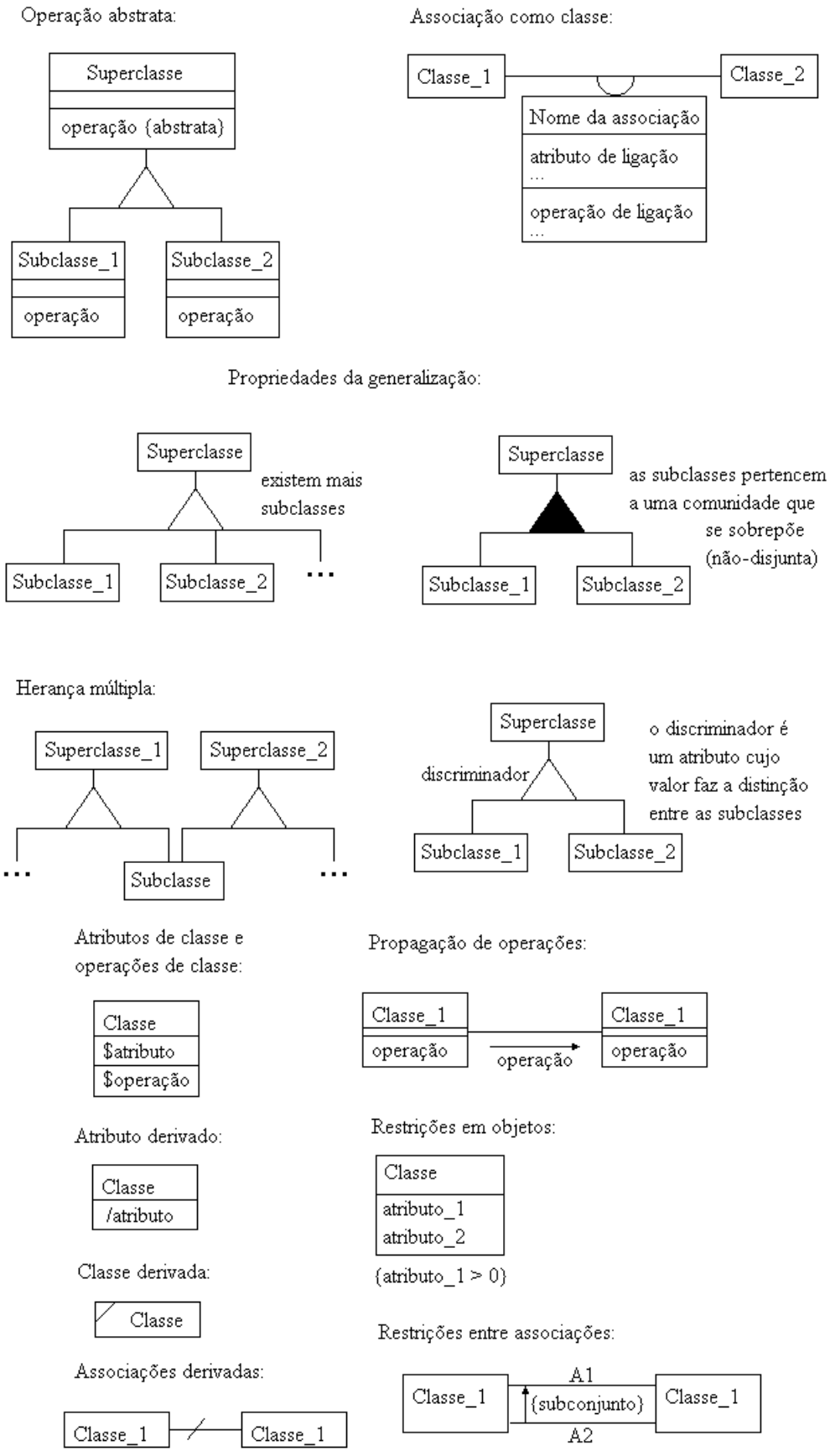


Figura 4.7 - elementos sintáticos do modelo de objetos - continuação

As demais informações representáveis através das associações serão descritas a seguir, a partir de um exemplo. A preocupação da descrição será usar os mecanismos definidos na metodologia para a construção de um modelo conceitual de um sistema, deixando de lado num primeiro momento, a relevância destes mecanismos para gerar implementações em um ou outro tipo de linguagem de programação.

Sejam duas classes, empresa e pessoa, e uma relação denominada trabalha_para, ligando as duas, com cardinalidade um ao lado de empresa e cardinalidade muitos ao lado de pessoa, conforme definido na figura abaixo. Uma primeira interpretação possível é o significado da relação: pessoa trabalha para empresa. Como uma associação é bidirecional (característica dos diagramas ER), a relação pode também ser interpretada como, empresa emprega pessoa. A informação da cardinalidade especifica quantas instâncias de uma classe podem relacionar-se com cada instância da outra classe. Segundo a convenção adotada, a cardinalidade do exemplo expressa que uma pessoa trabalha para uma empresa (exatamente) e que uma empresa emprega zero ou mais pessoas.

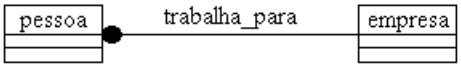


Figura 4.8 - um exemplo de relação envolvendo duas classes

Um papel pode ser associado às classes envolvidas na associação. No exemplo, o papel de empregador pode ser associado à empresa, e de empregado, à pessoa. Os papéis tem maior importância quando descrevem associações envolvendo a mesma classe, por exemplo a relação trabalha_para com as duas extremidades ligadas a pessoa: pessoa trabalha para pessoa. Neste caso, um dos objetos (da classe pessoa) assumirá o papel de empregador e o outro, de empregado.

Um qualificador pode ser acrescido a um dos lados da associação. Por exemplo, a inclusão do qualificador código_empregado à classe empresa, significa que um identificador de empresa acrescido a um dado código_empregado, se refere a uma instância específica da classe pessoa (e não a muitas).

Atributos podem ser definidos para as associações. O atributo cargo vinculado à associação trabalha_para dos exemplos anteriores permite descrever que na associação uma instância de pessoa trabalha para uma instância de empresa, assumindo um determinado cargo. Os autores preconizam que a vantagem do uso de atributos nas associações é evitar a alocação de atributos impróprios às classes - neste caso, o atributo cargo estaria inadequado na classe empresa ou na classe pessoa, estando adequado na associação que une as duas classes[19].

Uma associação ternária envolve três classes ao invés de duas. Tomando o exemplo anterior, seja a seguinte alteração: cargo passa a ser uma classe (com atributos salário, cargos superiores, cargos subalternos etc.). Uma associação ternária pode ser estabelecida entre as classes empresa, pessoa e cargo: pessoa trabalha para empresa em um cargo (que apresenta o mesmo significado do exemplo anterior). Associações ternárias podem ser excluídas de diagramas (a partir de uma modelagem diferente) caso se deseje somente associações binárias.

---

19 Isto é lógico, mas cabe considerar que modelagens alternativas são possíveis, caso se julgue inadequado incluir atributos a associações. Aliás, o que é algo sem correspondência com o paradigma de orientação a objetos, independente da preocupação com linguagens de programação.

OMT permite que um modelo de objetos seja repartido em módulos - evitando a construção de diagramas com grande número de classes, o que prejudica sua legibilidade. O modelo de objetos seria então, um conjunto de módulos. O módulo é uma estrutura lógica de agrupamento de classes relacionadas (por exemplo, um modelo de objetos dividido em módulo de interface, módulo de armazenamento de dados e módulo de processamento).

Outros elementos sintáticos presentes na figura 4.7, menos comuns (segundo os autores), não serão abordados nesta descrição.

4.2.2 Modelo dinâmico

O modelo dinâmico se utiliza de três técnicas de modelagem: o diagrama de estados, o diagrama de fluxo de eventos e o diagrama de eventos (sendo este último um instrumento auxiliar para a construção do primeiro, que os autores não incluem como parte do modelo dinâmico)[20]. Estes diagramas, como a visão do sistema expressa pela modelagem dinâmica, estão baseados nos conceitos de evento e estado.

Evento é uma indicação de que algo acontece em certo momento. Esta definição descreve a ótica de um observador externo. Assim, um evento que ocorre em um sistema é necessariamente uma manifestação externamente observável. Ocorrências internas não-observáveis de um sistema não constituem eventos. Num sistema descrito segundo a abordagem de orientação a objetos, as unidades de descrição são os objetos. Neste contexto, descrever eventos consiste em descrever as ocorrências externamente visíveis aos objetos[21].

Um evento é definido como uma ocorrência instantânea, isto é, não consome tempo. Dois eventos ocorrem de forma causal quando um evento ocorre em função de uma ocorrência anterior de outro evento. Eventos não relacionados de forma causal são chamados concorrentes.

Uma vez que um evento é uma manifestação externamente visível, ele envolve necessariamente mais de um objeto[22], o que obriga a ocorrência de comunicação entre objetos. Esta comunicação é admitida como unidirecional, ou seja, um dos objetos tem a iniciativa de procedê-la - assume uma atitude ativa, em contraste com a atitude passiva de quem recebe a mensagem. A esta iniciativa é adotada pelos autores, a noção de envio de evento: se um objeto é o responsável pela ocorrência de um evento que envolva um outro objeto, diz-se que o primeiro envia um evento ao segundo. Um evento pode transportar informações de um objeto a outro, na forma de atributos. Envio de eventos sem atributos corresponde a uma sincronização.

Eventos com características comuns são agrupados em classes de eventos[23]. Eventos podem descrever ocorrências normais de um sistema, ou condições de erro.

Um cenário é uma seqüência de eventos que ocorre durante uma determinada execução do sistema. Todas as funções que se espera que um sistema desempenhe

---

20 Além destas técnicas , a metodologia admite que o modelo dinâmico seja complementado por descrição textual para detalhar aspectos ausentes das demais descrições, por limitação de expressividade destas.

21 Que é diferente de descrever as ocorrências externamente visíveis ao sistema como um todo.

22 Ocorrências que não envolvam interação entre objetos são ocorrências internas e não caracterizam eventos.

23 Assim como quando se fala em descrever o comportamento de objetos, o que realmente se descreve são classes, a identificação e descrição de eventos se refere, de fato, a classes de eventos.

podem ser descritas como seqüências de eventos. O diagrama de eventos é utilizado para a descrição de cenários. É composto por traços verticais representando os objetos (classes) e setas, representando eventos (indicando o envio do evento). Identificar as situações a que o sistema deve responder e construir um diagrama de eventos para cada uma, é a forma adotada pela metodologia para a identificação de eventos.

O estado é uma visão instantânea de um objeto. É definido como a união de todos os valores que descrevem uma situação [JAC 92]. Coad e Yourdon utilizam uma definição mais direta: o estado de um objeto é definido pelos valores de seus atributos [COA 92]. Em [RUM 94] é assumido que o estado de um objeto é definido pelos valores de atributos e ligações[24]. [MOR 94] estabelece que além dos valores dos atributos o estado de um objeto é caracterizado também pelos serviços ora disponíveis[25].

Ao contrário dos eventos, um estado tem duração (tempo) e caracteriza o intervalo entre dois eventos.

O diagrama de estados relaciona estados e eventos: descreve a seqüência de estados percorridos por um objeto por causa de uma seqüência de eventos. Na definição dos estados que comporão o diagrama, atributos que não afetam o comportamento do objeto são ignorados. Por exemplo para descrever que um carro pode estar parado ou em movimento (comportamento dinâmico), não é relevante a cor do carro (atributo).

A figura 4.9 reúne o conjunto de elementos sintáticos de OMT, para a construção do diagrama de estados, parte do modelo dinâmico.

Os elementos básicos do diagrama de estados são os estados e as transições. O evento é o rótulo da transição correspondente. Aos eventos podem ser associados atributos, que são valores de dados transportados pelos eventos.

Guardas (funções booleanas) podem ser associadas às transições, condicionando sua ocorrência. Uma ação (operação instantânea em resposta a um evento) pode ser vinculada a uma transição - o envio de um evento a um objeto pode constituir uma ação. A estados podem ser associadas ações de entrada (executada quando o objeto atinge o estado) e de saída (executada imediatamente antes do objeto sair do estado). Ao estado também pode ser associada uma atividade (seqüência de ações que tomam tempo para se completarem). Ao estado ainda pode ser associado um evento. Neste caso, a ocorrência do evento não produz mudança de estado. Além disto, não são executadas as ações de entrada e saída (o que é semanticamente diferente de uma seta saindo e entrando no mesmo estado).

Estados e eventos podem ser expandidos em diagramas de estados multinivelados e podem ser organizados em hierarquias de herança. Os subestados herdam as transições de seus superestados. Os subeventos disparam as mesmas transições de seus supereventos. Concorrência também pode ser representada.

---

24 O uso de associações com atributos admitido em OMT faz com que parte das informações que descrevem um objeto estejam contidas nestas associações - associações sem atributos se refletem em atributos em objetos, como a metodologia de projeto de OMT deixa claro.

25 As várias definições não se contradizem e nem uma é incompleta em comparação a outras, pois, tanto as informações contidas em atributos de associações, como sobre a disponibilidade de um método são representáveis através de atributos de objeto.

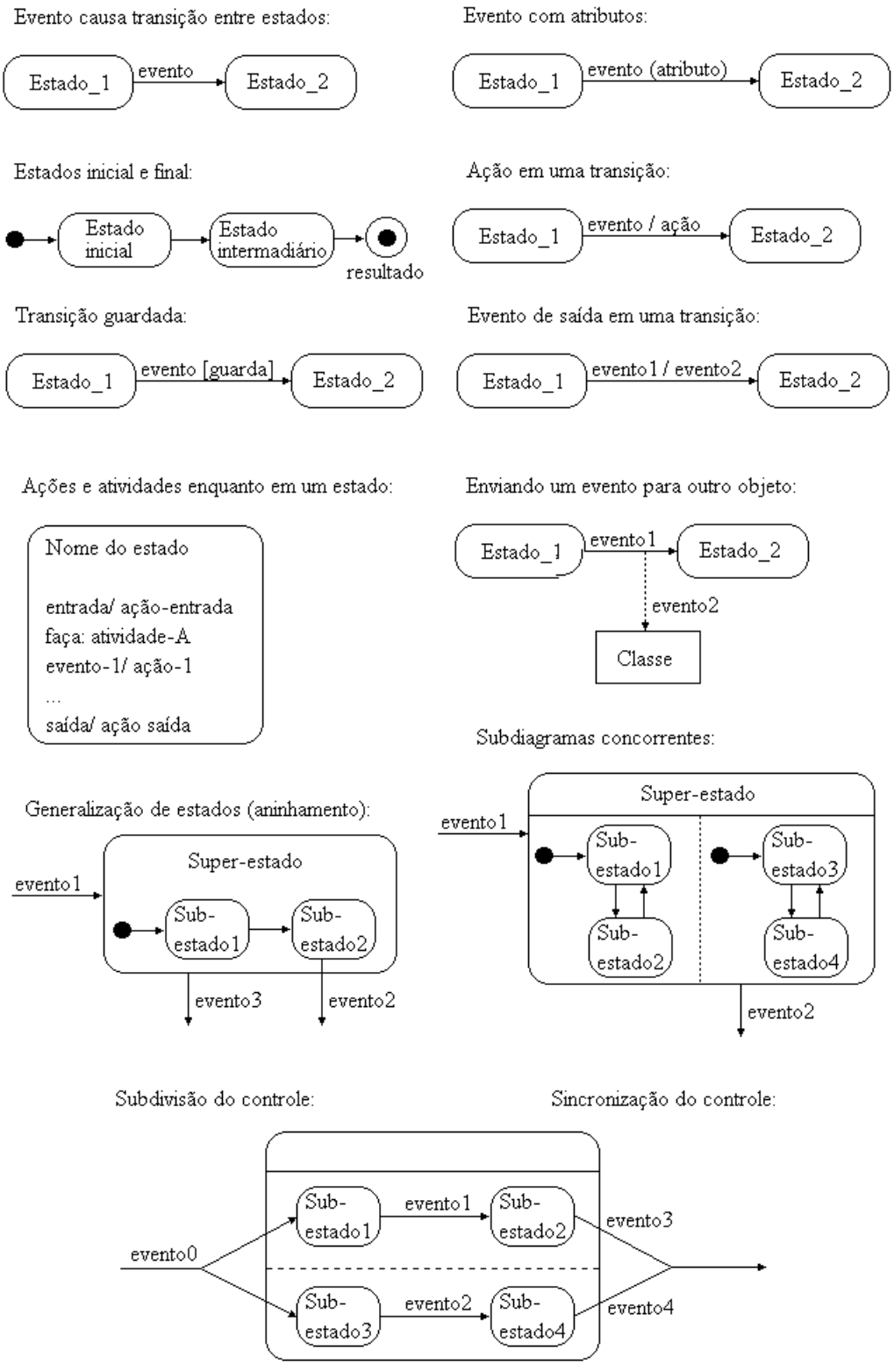


Figura 4.9 - elementos sintáticos do diagrama de estados (statechart)

O diagrama de fluxo de eventos consiste de retângulos representando as classes e setas as interligando. Sejam duas classes A e B interligadas por uma seta apontando para a classe A. Sobre esta seta são agrupados todos os eventos que B envia para A. O diagrama de fluxo de eventos, ao contrário dos diagramas de estados, fornece uma visão global do sistema e informa o conjunto de eventos enviados de uma classe para outra.

4.2.3 Modelo funcional

A figura abaixo reúne o conjunto de elementos sintáticos de OMT, para a construção do modelo funcional (DFD). Como os elementos são claros a partir da descrição da figura, não serão tecidos comentários a respeito deles.

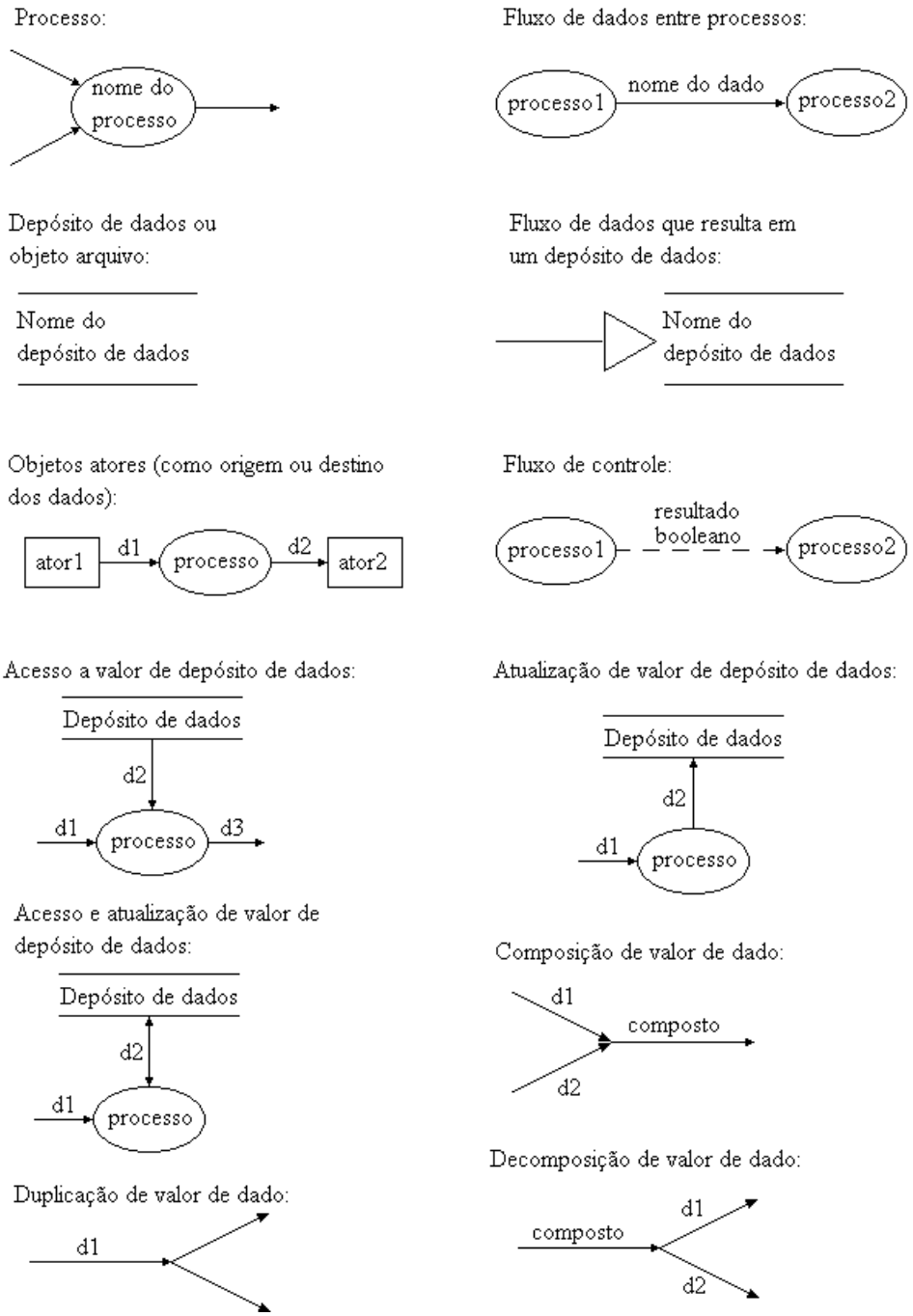


Figura 4.10 - elementos sintáticos do modelo funcional (DFD)

4.3 As etapas de construção de uma especificação na metodologia OMT

OMT propõe uma seqüência de passos para, a partir de um problema, chegar a uma proposta de solução (o projeto de uma aplicação). Os autores ressaltam que a metodologia é iterativa e que o seqüenciamento de passos não deve ser tomado de forma rígida, mas como uma linha de orientação. A ênfase e a complexidade de cada modelo e de cada etapa varia em função da aplicação em desenvolvimento, por exemplo, um compilador tem um modelo dinâmico trivial e um bancos de dados, um modelo funcional trivial.

Os passos listados a seguir reproduzem o exposto em [RUM 94], com alguns comentários, acrescidos para complementar ou discutir alguma afirmação - parte deles como nota de rodapé. O desenvolvimento de um exemplo (item seguinte) ilustrará estes passos.

4.3.1 Passos da análise

- Descrever ou obter uma descrição inicial do problema (enunciado do problema);

- Construir um modelo de objetos:
  - Identificar as classes de objetos;
  - Iniciar um dicionário de dados[26] com as descrições das classes, atributos e associações;
  - Acrescentar as associações entre classes;
  - Acrescentar os atributos para objetos e para ligações;
  - Organizar e simplificar as classes de objetos utilizando herança;
  - Testar os caminhos de acesso às informações[27];
  - Agrupar as classes em módulos, com base em estreito acoplamento e função relacionada;

Produto desta etapa (modelo de objetos): diagrama do modelo de objetos[28] e dicionário de dados;

- Desenvolver um modelo dinâmico:
  - Preparar cenários das seqüências típicas de interação;
  - Identificar eventos entre objetos e preparar uma seqüência de eventos para cada cenário;

---

26 OMT associa uma descrição textual a cada modelo para complementar informações - os autores reconhecem as limitações dos modelos em conter todas as informações da aplicação. O problema das descrições textuais é que introduzem informalidade nos modelos (com o risco de introduzir ambigüidades).

27 A noção de caminho de acesso de OMT é a mesma dos diagramas ER: uma classe tem acesso a informações de outra classe se existe um percurso de associações que as interliga (passando ou não por outras classes).

28 Até esta etapa não há, durante a construção do modelo de objetos, grande preocupação com a identificação dos métodos associados às classes. A identificação e alocação de métodos às classes é adiada para após a construção dos demais modelos, com base nas informações neles contida.

- Preparar um diagrama de fluxo de eventos para o sistema (descrição do sistema como um todo);
- Desenvolver um diagrama de estados para cada classe que tenha comportamento dinâmico relevante (descrição de cada classe, separadamente);
- Verificar a consistência e completeza dos eventos comuns a mais de um diagrama de estados;

Produto desta etapa (modelo dinâmico): diagramas de estados e diagrama global do fluxo de eventos;

- Construir um modelo funcional:
  - Identificar os valores de entrada e saída[29] (do sistema);
  - Utilizar diagramas de fluxo de dados quando necessário, para mostrar dependências funcionais;
  - Descrever o que cada função faz[30];
  - Identificar restrições[31];
  - Especificar os critérios de otimização[32];

Produto desta etapa (modelo funcional): diagrama de fluxo de dados e descrição das restrições;

- Verificar, repetir e refinar os três modelos:
  - Acrescentar ao modelo de objetos as operações-chave (métodos) que foram descobertas durante a preparação dos modelos dinâmico e funcional. OMT, como descrevem os autores, dá menos ênfase a métodos durante a análise, que outras metodologias baseadas em objetos. Assim, sob a justificativa de tornar o modelo da análise menos confuso, é recomendado que se acrescente ao modelo de objetos apenas os métodos mais importantes.
  - Verificar se as classes, associações atributos e operações estão consistentes e completos no nível escolhido de abstração. Comparar os três modelos com o enunciado do problema e conhecimento do domínio, e testar a consistência dos modelos utilizando cenários.
  - Desenvolver cenários mais detalhados (incluindo condições de erro) como variações dos cenários básicos. Utilizar estes cenários hipotéticos para verificar mais detalhadamente os três modelos.
  - Repetir os passos anteriores tantas vezes quantas forem necessárias para completar a análise - uma vez que OMT é essencialmente iterativo.

Documento da análise: enunciado do problema, modelo de objetos, modelo dinâmico e modelo funcional.

---

29 Consiste em construir um DFD de mais alto nível, contendo apenas um processo, que é o próprio sistema em desenvolvimento.

30 Esta descrição pode ser em linguagem natural, equações matemáticas, pseudocódigo ou alguma outra forma. OMT deixa em aberto a forma desta descrição.

31 Consiste em registrar condições a serem respeitadas ao longo das transformações de valores - que não é permitido aplicar o procedimento de cálculo de raiz quadrada se o valor de entrada for um número negativo, por exemplo.

32 Identificar pontos críticos em termos de eficiência - por exemplo, minimizar as mensagens entre elementos fisicamente separados, otimizar acessos a dados feitos com maior freqüência.

4.3.2 Passos do projeto

Projeto do sistema

- Organizar o sistema em subsistemas
- Identificar concorrências inerentes ao problema
- Alocar os subsistemas
- Escolher a estratégia básica para implementação dos depósitos de dados em termos de estrutura de dados, arquivos e bancos de dados
- Identificar os recursos globais e determinar mecanismos para controlar o acesso a eles
- Definir uma abordagem para a implementação do controle de software
- Considerar condições externas
- Estabelecer prioridades (desempenho, reusabilidade etc.)

Documento do projeto do sistema: estrutura da arquitetura básica do sistema, bem como decisões estratégicas de alto nível.

Projeto dos objetos

- Obter os métodos para o modelo de objetos a partir dos outros modelos:
  - Definir um método para cada processo do modelo funcional;
  - Definir um método para cada evento (recebido) do modelo dinâmico;

- Projetar algoritmos para implementar os métodos:
  - Escolher algoritmos que minimizem o custo da implementação dos métodos;
  - Selecionar as estruturas de dados adequadas aos algoritmos;
  - Definir novas classes internas e métodos, quando necessário;
  - Atribuir para métodos as responsabilidades que não estejam claramente associadas a uma única classe;

- Otimizar as vias de acesso aos dados:
  - Acrescentar associações redundantes para diminuir o custo de acesso[33];
  - Reorganizar a ordem de execução para melhorar a eficiência;
  - Salvar valores derivados para evitar que expressões complicadas sejam recalculadas;

- Implementar controle de software pelo refinamento da abordagem escolhida durante o projeto do sistema;

- Ajustar a estrutura de classes para aumentar a herança:
  - Rearranjar e ajustar classes e métodos para aperfeiçoar a herança;
  - Abstrair os comportamentos comuns dos grupos de classes;
  - Utilizar a delegação para compartilhar comportamentos em que a herança é semanticamente inválida;

---

33 A eficácia deste passo dependerá da interpretação dada às associações durante a implementação.

- Projetar a implementação de associações:
  - Analisar a travessia[34] das associações;
  - Implementar cada associação como um objeto ou acrescentando atributos a uma ou ambas as classes da associação;

- Determinar a representação exata dos atributos de objetos (tipo, formato etc.);

- Agrupar as classes e associações em módulos;

Documento de projeto: modelo de objetos detalhado, modelo dinâmico detalhado e modelo funcional detalhado.

4.4 Exemplo de uso de OMT

Análise

Modelo de objetos

Identificar classes associações e atributos; iniciar dicionário de dados[35]:

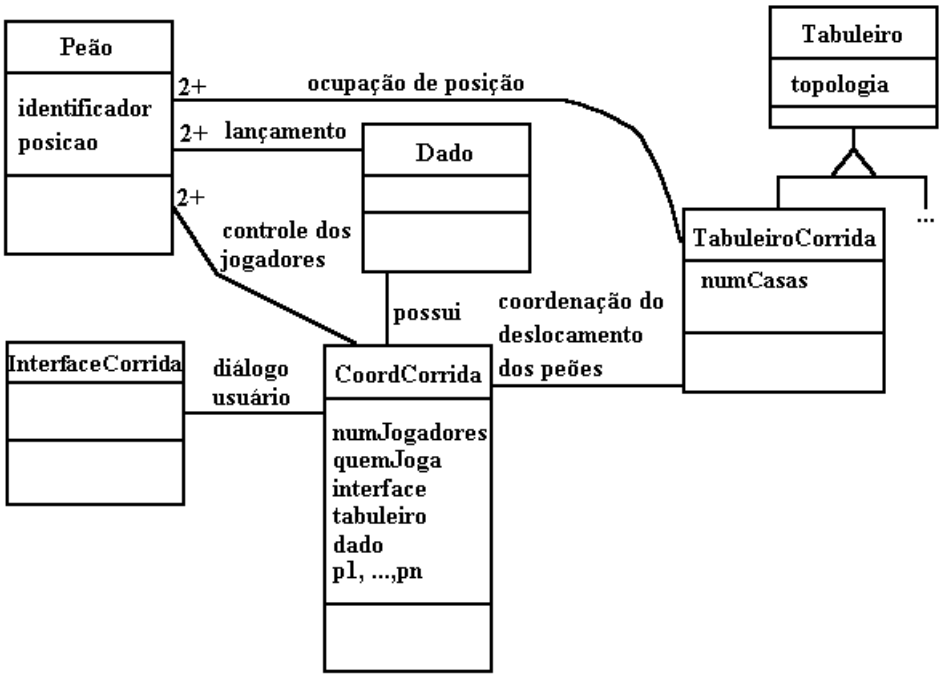


Figura 4.11 - modelo de objetos inicial a partir de OMT

---

34 Avaliar se a associação envolverá um fluxo bidirecional de informações (uma classe acessando informações da outra) ou unidirecional (apenas uma das classes tendo acesso a informações da outra). A implementação será diferente para cada caso.

35 Está sendo considerada a identificação destes elementos feita no capítulo anterior.

Classes (descrição de classes e atributos):

Peão: corresponde ao elemento peão, componente do jogo e representa a figura de um jogador (por isso não é necessária uma classe jogador).
Atributo:

- Posição: a posição do tabuleiro ocupada pelo peão. É inicializado na casa zero;
- Identificador: identifica o jogador a que corresponde o peão (string).

Dado: corresponde ao elemento dado, componente do jogo e sua finalidade é produzir um número aleatório no intervalo 1 a 6.
Atributo: o dado não precisa manter informação; apenas responde a uma chamada de método (solicitação de serviço).

TabuleiroCorrida: corresponde ao elemento tabuleiro do jogo corrida (um tabuleiro com topologia "linear"), componente do jogo, e sua finalidade é conter as normas que definem o deslocamento dos peões.
Atributo:

- numCasas: corresponde ao número associado à última casa (o que define o número de casas do tabuleiro). Este valor é solicitado ao usuário na inicialização de uma partida.

CoordCorrida: é o elemento que centraliza o controle de uma partida - inicializa a partida, solicita informações ao usuário, procede o sorteio da ordem de jogadores, estabelece a vez de cada jogador proceder seu lance ao longo de uma partida, verifica se houve um vencedor ao final de um lance.
Atributos:

- numJogadores: corresponde ao número de jogadores de uma partida. Este valor é solicitado ao usuário na inicialização de uma partida.
- quemJoga: referência ao peão que realiza o lance.

InterfaceCorrida: executa os métodos responsáveis pela comunicação entre o usuário e o coordenador.
Atributos: não definidos[36].

Associações[37]

Ocupação de posição (relacionando Peão e TabuleiroCorrida): estabelece que um peão[38] ocupa uma posição do tabuleiro. Cardinalidade: um peão está associado a exatamente um tabuleiro e um tabuleiro, a dois ou mais peões - o peão precisa conhecer sua posição no tabuleiro, mas o tabuleiro não precisa armazenar informações sobre os peões como atributos, pois, precisa apenas saber definir o novo posicionamento do peão a cada lance.

---

36 Como estabelecido no enunciado do problema, a especificação desta classe não será completa - apenas especificará genericamente os seus métodos.

37 Como estilo de modelagem, optou-se por não incluir atributos ou métodos às associações.

38 Letra minúscula indica referência a uma instância - e não à classe.

Lançamento (relacionando Peão e Dado): estabelece que o peão lança o dado (solicita o método correspondente). Cardinalidade: para uma partida existe apenas um dado e dois ou mais peões.

Controle dos jogadores (relacionando Peão e Coordenador): esta associação é mais que uma chamada de método - corresponde ao controle de toda a existência de um peão em uma partida, procedida pelo coordenador: inicialização, deslocamentos (lances), verificação de vencedor. Cardinalidade: para uma partida existe apenas um coordenador e dois ou mais peões.

Coordenação do deslocamento dos peões (relacionando TabuleiroCorrida e Coordenador): A relação entre coordenador e tabuleiro é o controle do deslocamento dos peões sobre o tabuleiro - o coordenador controla a atividade do tabuleiro de deslocar peões (a definição de novas posições para os peões). Cardinalidade: para uma partida existe apenas um coordenador e um tabuleiro.

Possui (relacionando Dado e Coordenador): é basicamente uma relação de ocorrência - para uma partida existe apenas um coordenador e um dado. Dado e coordenador não precisam se comunicar ao longo do desenvolvimento de uma partida - apenas a instância dado é criada por iniciativa do coordenador (instância da classe CoordCorrida).

Diálogo usuário: abstrai todo o relacionamento entre o usuário e o jogo, que se processa entre instâncias das classes InterfaceCorrida e CoordCorrida.

Modelo dinâmico

Primeiro passo: preparar cenários - dois cenários caracterizam o sistema sob descrição, que são a inicialização de uma partida e seu desenvolvimento (que corresponde a um conjunto de procedimentos de lance.

Segundo passo: identificar eventos e construir um diagrama de eventos para cada cenário.

Diagramas de eventos

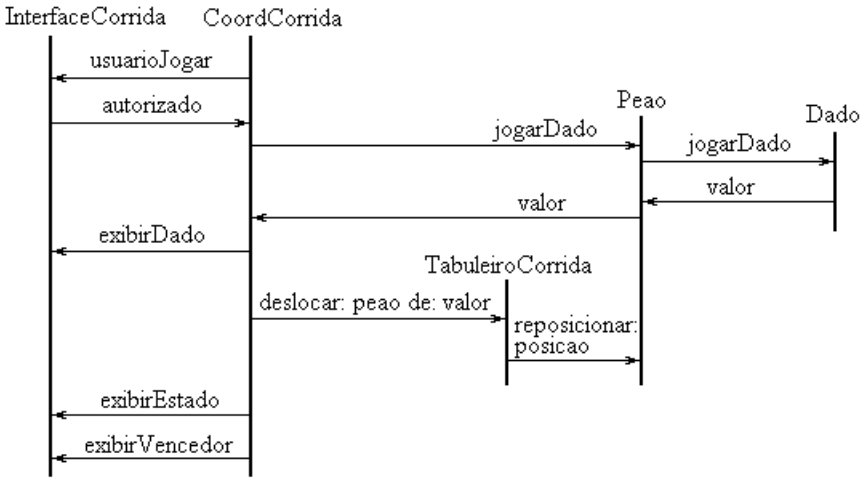


Figura 4.12 - cenário para o desenvolvimento de um lance em uma partida

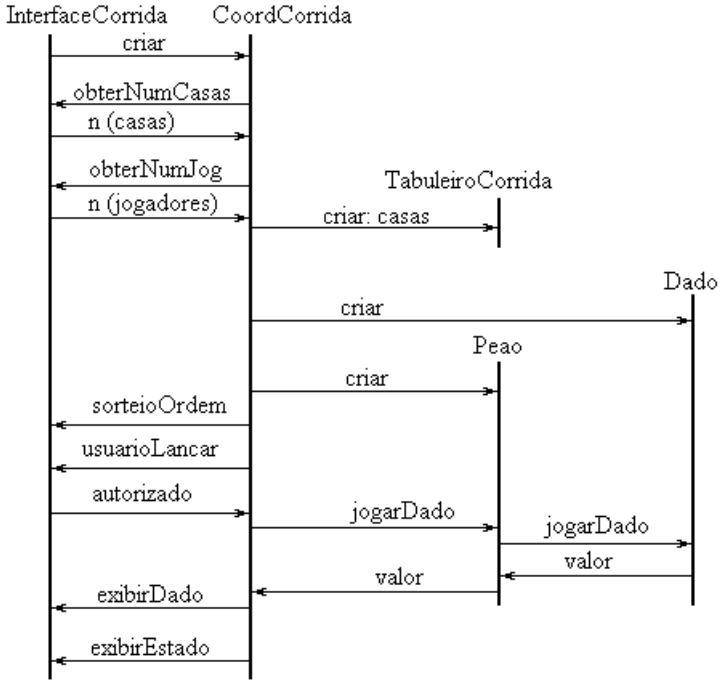


Figura 4.13 - cenário para a inicialização

Terceiro passo: preparar um diagrama de fluxo de eventos para o sistema.

Diagrama de fluxo de eventos

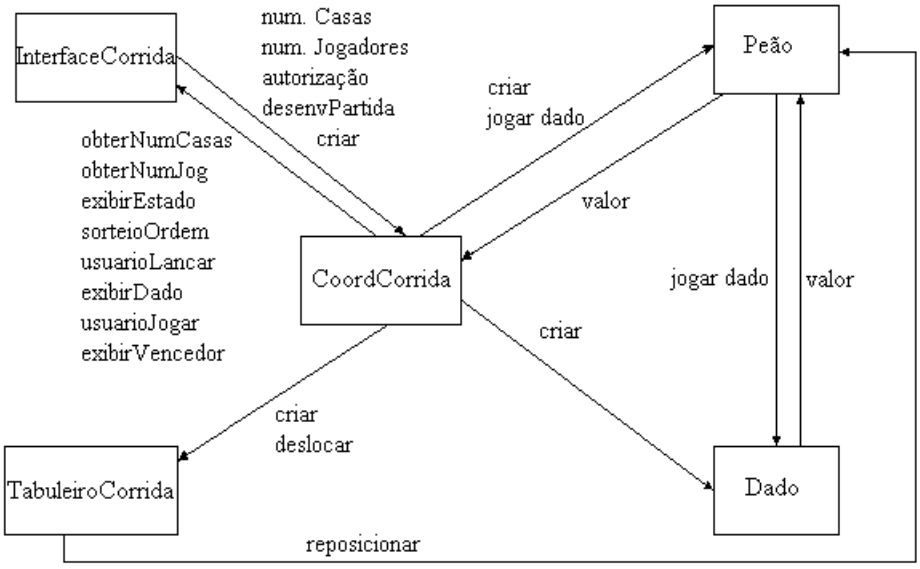


Figura 4.14 - diagrama de fluxo de eventos

Quarto passo: desenvolver um diagrama de estados para cada classe que tenha comportamento dinâmico relevante.

Será apresentado a seguir apenas o diagrama de estados referente à classe coordenador, porque é a classe cujo comportamento dinâmico pode ser considerado de fato, relevante (e também para não estender demais esta especificação).

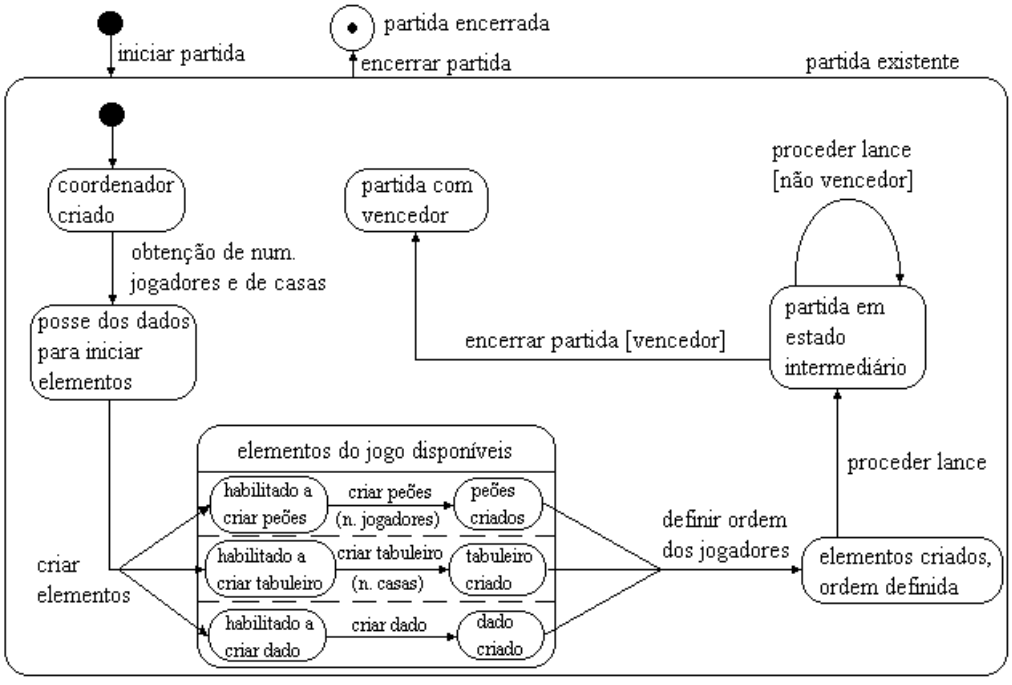


Figura 4.15 - diagrama de estados para a classe CoordCorrida

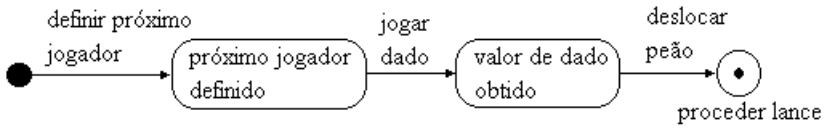


Figura 4.16 - refinamento da transição "proceder lance"

Modelo funcional (diagramas de fluxo de dados do sistema)

Figura 4.17 - DFD de nível mais elevado: o único processo corresponde ao sistema

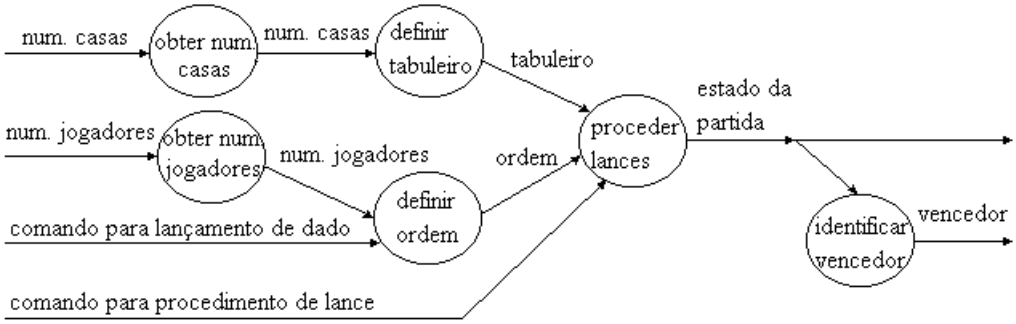


Figura 4.18 - refinamento do processo "jogo corrida"

Projeto

Projeto do sistema

Para a presente aplicação, os procedimentos desta etapa são elementares. Primeiro, como considerado no capítulo anterior, não se fará uma divisão das classes em subsistemas, devido ao número reduzido de classes. Segundo, quanto a gerenciamento de depósitos de dados, não há necessidade, pela simplicidade dos dados manipulados. Terceiro, conforme estabelecido no enunciado do problema, será usada uma biblioteca de classes para a construção da interface gráfica, que não será detalhada na presente especificação.

Diagrama de blocos do sistema:

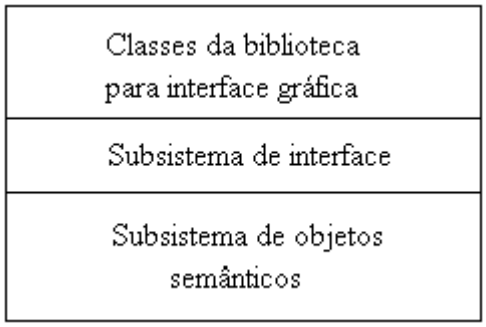


Figura 4.19 - diagrama de blocos do sistema

Subsistema de interface: contém a classe InterfaceCorrida;
Subsistema de objetos semânticos: agrupa as classes tabuleiro, Peao e Dado.

Projeto dos objetos

Obter os métodos para o modelo de objetos a partir dos outros modelos[39].

Dicionário de dados (continuação)

Classes (métodos)

Peão
Métodos:
- jogarDado: a solicitação deste método leva o peão a proceder um lançamento de dado e retornar o valor obtido;
- reposicionar: faz com que o peão passe a ocupar uma nova posição - que é o argumento.

Dado
Método:
- jogarDado: a solicitação deste método produz um número aleatório no intervalo 1 a 6, que é retornado.

TabuleiroCorrida
Método:
- deslocar: tendo como argumentos o peão a ser deslocado e o valor obtido por este em um lançamento de dado, retorna a nova posição a ser ocupada pelo peão. No algoritmo deste método está embutido o conjunto de regras de deslocamento (descrito no enunciado do problema).

---

39 OMT determina que os algoritmos dos métodos sejam projetados, porém não determina um formato para esta informação (fluxograma, pseudocódigo, descrição textual) e nem em que modelo ela deve ser incluída. Assim, para não executar um procedimento não previsto, não será feita outra descrição além da que está incluída no dicionário de dados. Para maiores detalhes dos algoritmos, vide as especificações de classe-&-objeto do capítulo anterior.

Coordenador
Método:

- desenvPartida: desenvolve uma partida completa: Procede a inicialização de uma partida - solicita ao usuário o número de jogadores e a dimensão do tabuleiro; procede a determinação da ordem dos jogadores; instancia e inicializa (atributos) os objetos necessários ao desenvolvimento da partida (peões, dado e tabuleiro) - procede seqüências de lances na ordem estabelecida na inicialização, em looping, até haver um vencedor. O desenvolvimento de um lance corresponde a solicitar a um usuário jogador (que corresponde a um dos peões) proceder um lançamento de dados; passar ao tabuleiro o valor obtido, juntamente com o peão correspondente (que, por sua vez, providenciará o reposicionamento do peão); verificar se o peão é o vencedor (se ocupa a última casa do tabuleiro).

Quanto às associações, já estão traduzidas nos atributos e métodos definidos, não demandando a definição de novas classes.

Documento de projeto:

Modelo de objetos (diagrama e dicionário de dados): o dicionário de dados corresponde à união das duas partes apresentadas (não será repetido) e o modelo de objetos completo está apresentado abaixo.

Modelos dinâmico e funcional: correspondem aos apresentados na análise (como o sistema é simples, não houve a necessidade de alterações - o que não é regra geral).

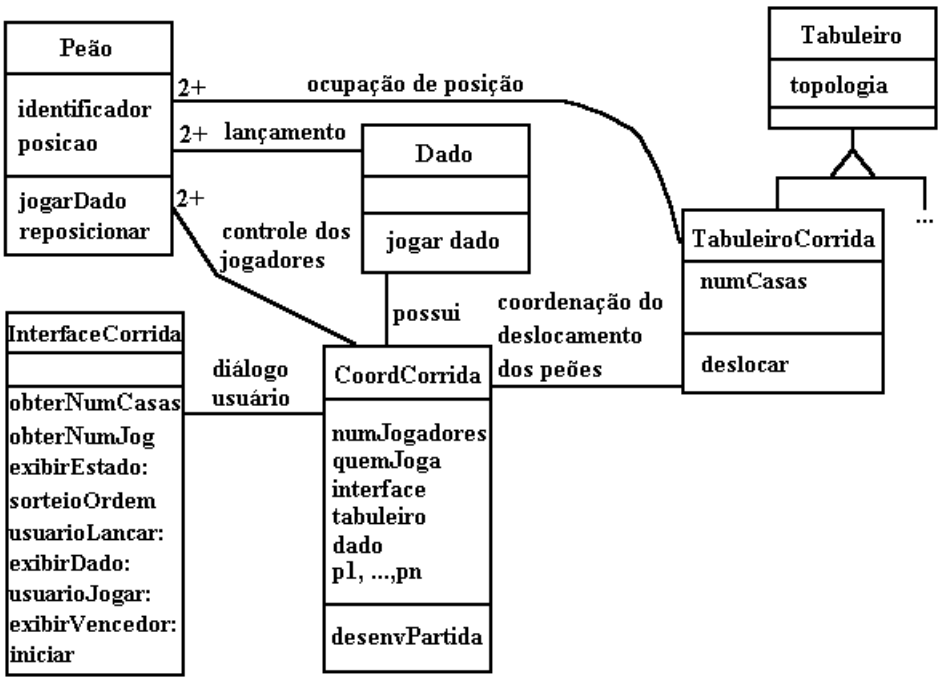


Figura 4.20 - modelo de objetos completo

5 Metodologia OOSE

5.1 Visão geral da metodologia

A metodologia OOSE [JAC 92] é uma metodologia orientada a objetos, cuja abordagem é classificada como dirigida a cenário[40]. O desenvolvimento de uma aplicação passa pelas etapas de análise e construção. Na análise é construído um modelo de requisitos, que apresenta uma especificação do sistema sob a ótica de um observador externo, e o modelo de análise, que faz uma primeira aproximação da descrição interna do sistema. Os autores estabelecem que a preocupação da análise é produzir uma descrição de sistema independente da realidade de implementação. Não há grande preocupação nesta etapa em definir atributos e serviços, o que é protelado para o projeto - os autores preferem nesta etapa definir papéis e responsabilidades dos elementos do sistema.

A etapa chamada de construção se divide em projeto e implementação. A etapa de projeto, onde é construído o modelo de projeto, visa "formalizar"[41] o modelo de análise e completá-lo com detalhes de implementação.

O principal elemento de descrição da metodologia são os *use cases*[42]. *Use cases* são as situações a que o sistema pode ser submetido, ou seja, as situações de processamento a que o sistema deve responder. As várias etapas de descrição de um sistema em OOSE se baseiam na definição de *use cases*.

5.1.1 Análise e projeto

Na análise são construídos os modelos de requisitos e de análise. A construção do modelo de requisitos é iniciada com a identificação dos atores, ou seja, os elementos que interagem com o sistema - pessoas, equipamentos, outros sistemas. Identificados os atores, são definidos os *use cases* a partir da busca de todas as situações de interação entre atores e sistema. Atores e *use cases* são agrupados em um diagrama sintaticamente simples e é procedida uma descrição textual de cada *use case*, sob a ótica de um observador externo ao sistema. O modelo de requisitos inclui uma descrição informal da interface e um primeiro modelo de objetos, com as classes do domínio do problema identificadas[43] - com herança e associações.

O principal elemento do modelo de análise é um conjunto de modelos de objetos. A partir dos *use cases* são identificados as classes e sua participação em cada *use case* - classes podem estar presentes em uma ou mais descrições de *use cases*. O modelo de análise contém uma descrição textual de cada classe, onde a ênfase não é descrever seus atributos e métodos, mas seus papéis e responsabilidades nos *use cases*.

---

40 Em contraste com a abordagem dirigida a dado, das metodologias OMT, Fusion e de Coad e Yourdon, e com a abordagem dirigida a evento de Martin e Odell.

41 Os modelos de requisitos e de análise estão muito escorados em descrição textual, sendo assim bastante informais, como se verificará a seguir.

42 A expressão *use cases* pode ser traduzida como casos de uso ou como situações de utilização. No presente texto será mantida a expressão em Inglês, por ser uma expressão que caracteriza a metodologia - assim como em OMT se manteve a sigla da denominação da metodologia em Inglês.

43 Na construção de modelo de objetos, que constitui o modelo de análise, OOSE não busca uma descrição completa das classes, mas as descreve em função de sua participação nos *use cases*.

Também faz parte do modelo de análise, uma descrição textual de cada *use case* - é descrito como os objetos interagem para a ocorrência do *use case*, o que é diferente da descrição do modelo de requisitos.

Na etapa de análise o sistema é dividido em subsistemas, que agrupam as classes identificados - o critério da divisão é basicamente funcionalidade, ou seja, um subsistema é uma parte do sistema que desempenha uma determinada função.

O modelo de projetos é composto pelos blocos que constituem o sistema. Um bloco é basicamente uma classe, em termos de implementação. Os blocos correspondem às classes identificadas na análise. Em função de fatores referentes à implementação, novos blocos podem ser adicionados (novas classes). Inicialmente é feita a identificação dos blocos; a seguir são construídos diagramas de interação, que formalizam a descrição de *use cases* desenvolvida na análise (descrição de cenários). Estas informações que se referem à descrição do sistema como um todo são a base para o detalhamento de cada bloco. O projeto dos blocos consiste em inicialmente definir a interface de cada bloco, com base nas interações que participa (informação retirada dos diagramas de interação), e a seguir, definir o comportamento do bloco, que corresponde a uma modelagem dinâmica usando diagrama SDL [CCI 88]. A implementação consiste na tradução do projeto dos blocos, na linguagem de programação escolhida.

5.1.2 Técnicas de modelagem para a descrição de uma aplicação

A metodologia OOSE utiliza um conjunto de modelos para descrever um sistema: o modelo de requisitos, o modelo de análise, o modelo de projeto e o modelo de implementação - sendo este último a própria implementação do sistema.

O modelo de requisitos possui como principal elemento, o diagrama de *use cases*, que é bastante simples, onde são representados atores, *use cases* e a ligações entre eles - sem detalhamento dos elementos, apenas seus nomes. O restante da descrição de *use cases* é feito de forma textual. Sua finalidade é descrever as situações de processamento a que o sistema pode ser submetido, sem descrevê-lo internamente, mas apenas numa visão "caixa preta", do diálogo entre ator e sistema. A descrição da interface, parte do modelo de requisitos, não tem formato definido em OOSE: pode conter descrição textual, lay-outs de telas gráficas e painéis de controle, ou mesmo incluir protótipos. O modelo de objetos, também parte do modelo de requisitos, inclui classes do domínio do problema e suas associações - bem como estrutura hierárquica de herança e agregação.

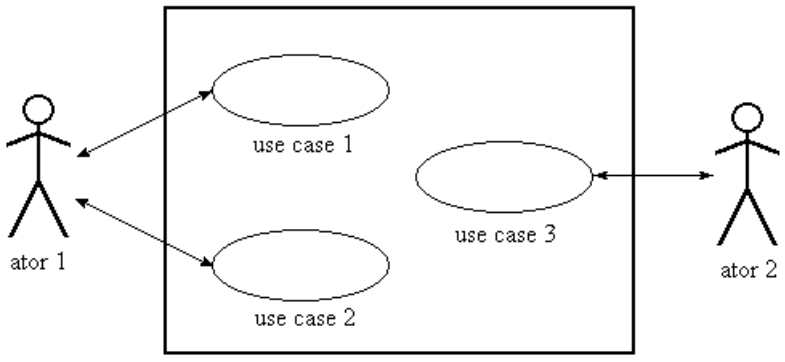


Figura 5.1 - exemplo de diagrama de use cases

O elemento central do modelo de análise é um conjunto de modelos de objetos compostos por três tipos de classes: as classes entidade, que são as classes do domínio do problema já identificadas; as classes de interface, responsáveis pela comunicação entre o sistema e o meio externo; e as classes de controle, que concentram funcionalidades específicas de determinados *use cases*, que não se deseje[44] alocar nos outros tipos de classes. O modelo de análise é completado por descrição textual associada aos diagramas (descrição das classes e dos *use cases*). A construção do modelo de objetos do sistema é feita a partir de vários diagramas de objetos, cada um descrevendo um *use case*. A funcionalidade de cada classe está distribuída por vários diagramas. A construção de cada diagrama consiste em identificar que classes participam do *use case* (classes entidade, de interface e de controle) e como estão associadas. A associação entre classes é diferente para os vários *use cases* (duas classes associadas em um *use case* podem não estar associadas em outro, por exemplo). A descrição de classes do modelo de análise difere da descrição do modelo de requisitos pela inclusão de tipos de classes de interface e de controle, e pela maior especificidade das associações, que refletem as particularidades dos *use cases* (no modelo de requisitos as associações são mais genéricas). A parte diagramática do modelo de análise consiste basicamente em uma descrição estática (modelo de objetos). A modelagem dinâmica fica a cargo da descrição textual, sendo portanto, informal - exceto pela presença de associações de comunicação no modelo de objetos. Cabe salientar que OOSE concentra menos informações no modelo de análise que outras metodologias orientados a objetos: não há grande preocupação em definir atributos e serviços[45], o que é deixado para a etapa de projeto.

O modelo de projeto é mais formal que os anteriores. Inclui descrição estática e dinâmica tanto do sistema como um todo, quanto das classes em particular. A estrutura de blocos reflete a mesma estrutura estática definida na análise - exceto pelas alterações que ocorram no projeto. Os diagramas de interação produzem uma descrição dinâmica do sistema (composto por um conjunto de blocos). A descrição da interface dos blocos consiste em relacionar seus atributos e a assinatura de seus métodos, na sintaxe da linguagem de programação adotada para a implementação. A última atividade de projeto consiste em produzir a descrição do corpo dos blocos, ou seja, definir os algoritmos dos métodos. Isto é feito no projeto do comportamento dos módulos, que utiliza diagrama SDL.

5.2 Elementos sintáticos das técnicas de modelagem da metodologia OOSE

5.2.1 Modelo de requisitos

A figura abaixo apresenta a representação de *use cases* do modelo de requisitos de OOSE. Ícones representam os atores, e elipses, os *use cases*. Os *use cases* estão

---

44 A expressão *desejo* quer significar estilo de modelagem, ou seja, qualquer sistema descrito usando classes de controle pode também ser descrito sem que seja usado este tipo de classe. Isto é semelhante à opção de concentrar ou não muitas informações nas associações de OMT.

45 Atributos necessariamente aparecem na descrição textual, mas sem a preocupação de identificar todos os atributos de uma classe. Quanto a métodos, ao invés de sua identificação, OOSE recomenda definir papéis e responsabilidades associados às classes.

contidos em um retângulo, que representa o limite do sistema. Os atores recebem identificadores que descrevem o papel por eles representado no sistema - uma mesma pessoa que interaja com um sistema, por exemplo, pode corresponder a mais de um ator, se desempenhar diferentes papéis. As elipses que representam *use cases* são rotuladas com seus identificadores. Uma representação alternativa, útil quando a quantidade de *use cases* é elevada, é representar em um diagrama os atores identificados e o sistema, sem os *use cases* (apenas o retângulo). Neste caso o conjunto de *use cases* é representado em um diagrama à parte - o relacionamento dos atores com os *use cases* fica registrado na descrição textual.

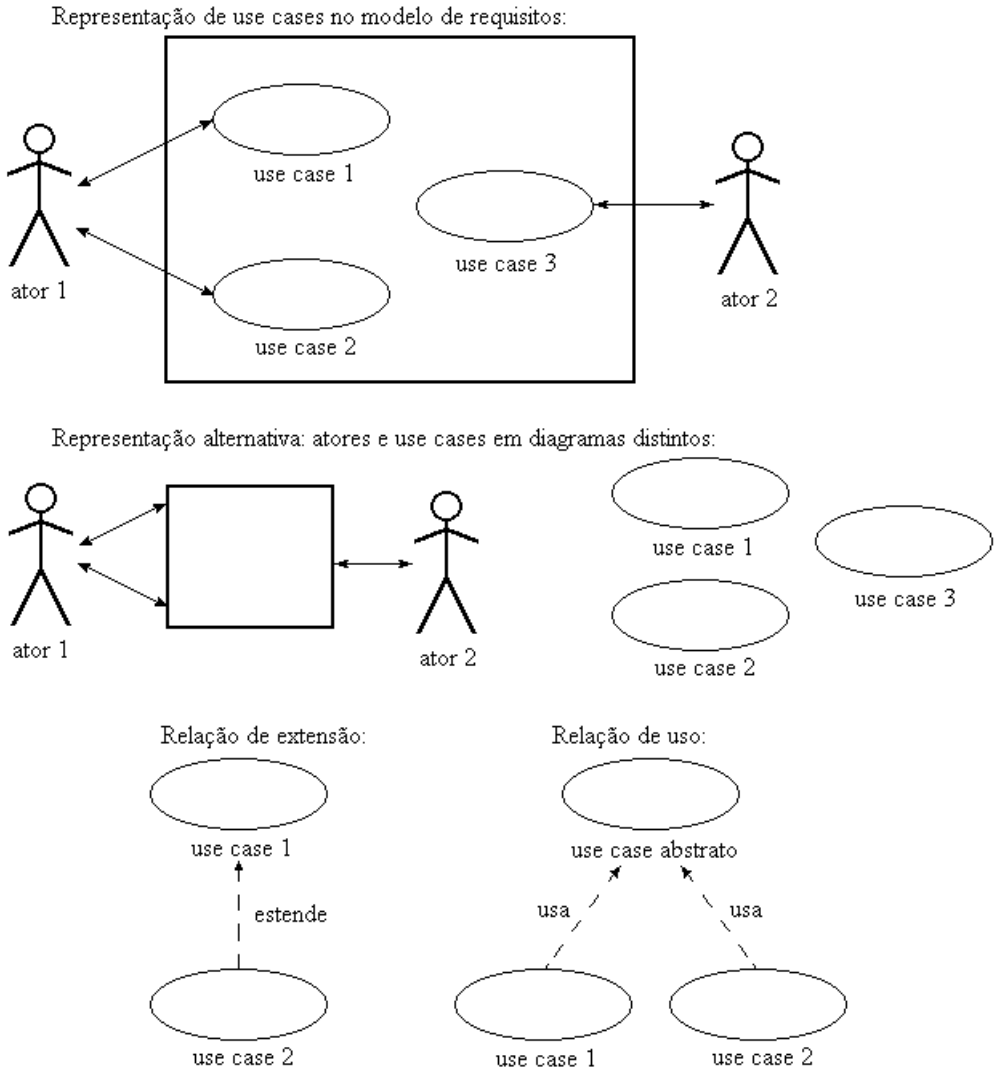


Figura 5.2 - elementos sintáticos para representação de use cases

Associações de extensão são usadas para conectar *use cases*, em que um corresponde a uma extensão do outro (um processamento adicional). Por exemplo, um *use case* corresponde ao curso normal de um processamento e a extensão, a um tratamento de falta. A extensão só será executada em caso de problema no processamento.

Associações de uso fazem a conexão de *use cases* concretos a *use cases* abstratos. Um *use case* abstrato é um procedimento que por fazer parte de mais de um

use case, é representado em separado - como um procedimento de impressão, por exemplo.

A modelagem de objetos do modelo de requisitos utiliza um subconjunto dos elementos sintáticos do modelo de análise (não utiliza classes de interface e de controle), que será descrito a seguir.

5.2.2 Modelo de análise

Os elementos sintáticos para a construção dos diagramas do modelo de análise estão apresentados na figura abaixo.

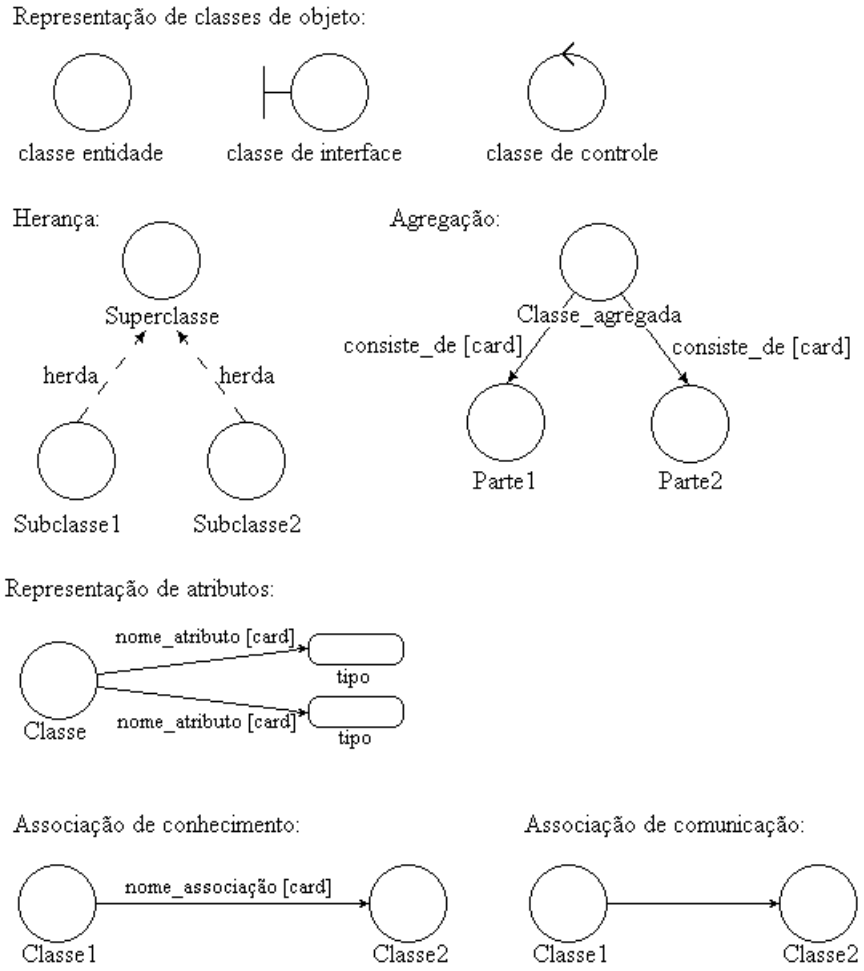


Figura 5.3 -elementos sintáticos do modelo de análise

As classes de objetos tem formato circular e são diferenciadas, conforme os tipos definidos em OOSE. Os atributos são representados por retângulos de cantos arredondados conectados ao símbolo de classe. A representação de atributo inclui nome do atributo, tipo e cardinalidade (quantos elementos de um tipo de atributo uma classe pode comportar). A técnica de modelagem não comporta a inclusão de métodos.

OOSE classifica associações em dois tipos: estáticas e dinâmicas. No grupo das associações estáticas inclui herança, agregação e associação de conhecimento - sendo esta uma associação entre instâncias em que uma instância conhece a existência de outra instância (manutenção de referência). A associação de conhecimento é unidirecional -

uma seta aponta para a classe conhecido. Sua representação inclui cardinalidade. Esta associação se refletirá em atributos na implementação, semelhante ao que pode ocorrer com as associações de OMT. Associação bidirecional demanda duas associações de conhecimento. A associação de conhecimento não capacita o objeto a trocar informações com outro. Para isto são necessárias as associações dinâmicas.

Associações dinâmicas são as associações de comunicação. Através desta associação um objeto envia e recebe estímulos - o que corresponde a ocorrência de eventos. A representação da associação de comunicação é feita a partir de uma seta indicando quem procede o envio. A seta não é rotulada, assim, o significado da comunicação representada no diagrama, deve ser expresso na descrição textual (a ausência de rótulo diferencia sintaticamente esta associação da associação de conhecimento). A associação de comunicação pode representar o envio de uma mensagem para a execução de um método ou abstrair uma interação mais complexa, que envolva várias trocas de mensagens (identificando pela seta, quem toma a iniciativa da interação). A diferenciação entre associação de conhecimento (estática) e associação de comunicação (dinâmica) não é adotada pelas metodologias OMT, Fusion e de Martin e Odell.

5.2.3 Modelo de projeto

A estrutura estática do modelo de projeto consiste em um modelo de objetos de sintaxe idêntica à do modelo de análise, exceto que as classes (agora chamadas blocos) são representadas por retângulos (não diferenciados). OOSE admite o uso de apenas um diagrama ou de vários, específicos por *use case*, como na análise.

O diagrama de interação de OOSE é bastante semelhante ao diagrama de eventos de OMT. Como neste, as classes de objetos são representadas por traços verticais, os eventos (estímulos) por setas e ordem de eventos se dá de cima para baixo. Adicionalmente ao diagrama de OMT, uma barra vertical diferenciada representa a borda do sistema (destacando os estímulos que entram e saem); os estímulos podem ser diferenciados entre mensagens (estímulos intra-processos) e sinais[46] (estímulos inter-processos, que podem ser síncronos ou assíncronos); mais de um traço vertical pode ser associado a uma classe, representando suas instâncias; a duração da execução de uma operação é destacada por um retângulo sobre o traço do respectivo objeto; e pseudocódigo (ou texto estruturado) colocado à esquerda do diagrama permite a representação de execução condicional, execução em looping e não-determinismo[47]. A figura abaixo apresenta um exemplo de diagrama de interação.

---

46 Esta diferenciação se baseia na noção de processos concorrentes, como apresentado na linguagem ADA. Em linguagens não concorrentes como Smalltalk e C++, só se aplica o conceito de mensagem.

47 A possibilidade de representar execução condicional, looping e não-determinismo representa a principal diferença - e a principal vantagem - da técnica de modelagem de OOSE em relação à de OMT.

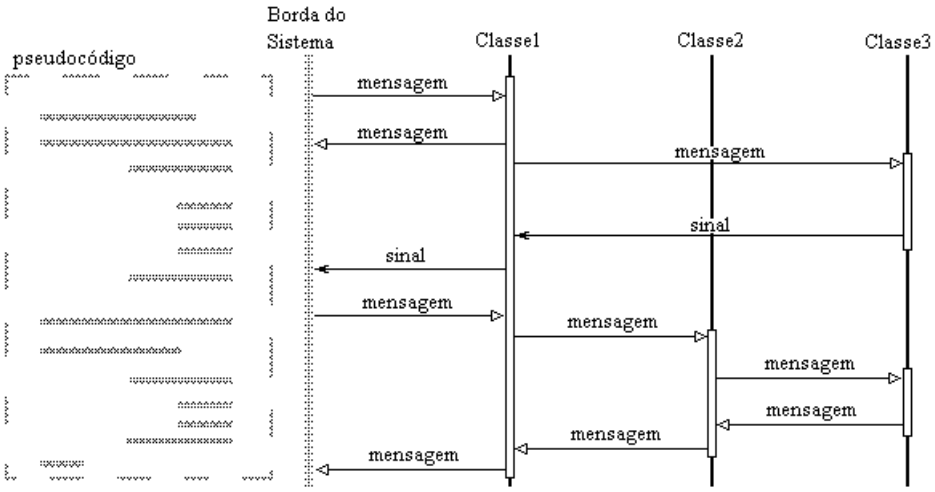


Figura 5.4 -elementos sintáticos do diagrama de interação

Para a descrição do comportamento interno dos blocos (informação necessária para o desenvolvimento dos algoritmos de seus métodos) é utilizada a notação SDL (Specification and Description Language - Linguagem de Especificação e Descrição), um padrão CCITT [CCI 88]. Esta técnica de modelagem descreve o comportamento dinâmico associado à classe de objetos, como uma máquina de estados, semelhante ao que é feito com statechart em OMT. Possui representação de estado, de comunicação (enviada e recebida), de execução de procedimento, de tomada de decisão, de criação (símbolo de início) e destruição de instância e label, que permite estender o diagrama em outra folha. Este conjunto de elementos sintáticos, possibilita a descrição do algoritmo dos métodos, algo que não é possível com statecharts. O diagrama SDL não admite estruturação. O diagrama associado a uma classe deve ser detalhado o suficiente para que a implementação dos métodos seja uma tradução trivial (a elaboração do diagrama SDL é o último passo antes da implementação). A figura abaixo apresenta os elementos sintáticos do diagrama SDL.

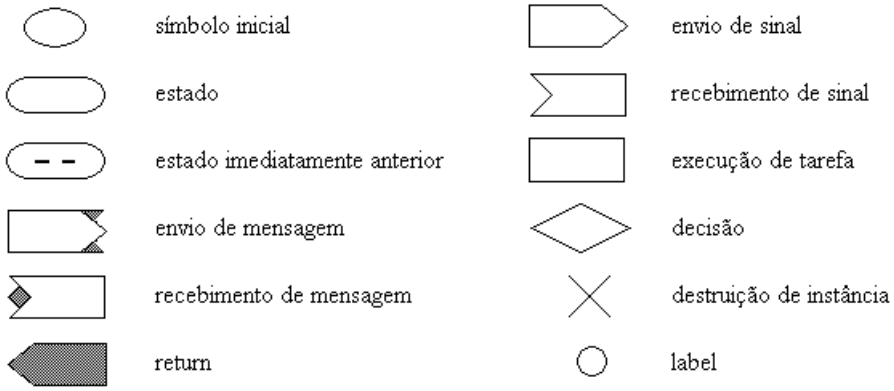


Figura 5.5 -elementos sintáticos do diagrama SDL

5.3 As etapas de construção de uma especificação na metodologia OOSE

5.3.1 Passos da análise

OOSE propõe uma seqüência de passos para a construção de uma especificação, mas como é comum a outras metodologias orientados a objetos, ressalta a iteratividade inerente à atividade de modelagem, ou seja, passos já procedidos podem ser retomados, com base em informações obtidas em passos posteriores.

Construção do modelo de requisitos:

- Identificação de atores: os elementos que interagem com o sistema - eles constituem uma ferramenta para identificação dos *use cases*;
- Identificação dos *use cases*: obtido a partir de uma busca exaustiva de todas as formas possíveis de cada ator interagir com o sistema, inclusive as situações que envolvem simultaneamente mais de um ator. Para identificação de *use cases* a partir dos atores, OOSE propõe o seguinte roteiro de questões:
  - Quais as principais tarefas de cada ator?
  - O ator terá que ler, escrever ou alterar alguma informação do sistema?
  - O ator terá que informar ao sistema mudanças ocorridas externamente?
  - O ator deseja ser informado sobre mudanças inesperadas?
- Construção de um diagrama relacionando atores e *use cases*;
- Descrição textual dos *use cases*, apresentando a seqüência de transações procedidas no diálogo entre ator e sistema;
- Identificação de extensões de *use cases*: identificação de fluxos de processamento alternativo aos identificados, ou transformação de *use cases* identificados em extensões de outros;
- Descrição da interface do sistema: pode conter descrição textual, lay-outs de telas gráficas e painéis de controle, ou mesmo incluir protótipos;
- Identificação das classes de objetos do domínio do problema - identificação de atributos está restrita às necessidades de informação dos *use cases*, no nível de abstração ora disponível, e pode ser deixada para o modelo de análise (os atributos não precisam estar contidos no primeiro modelo de objetos); identificação de métodos não é procedida;
- Construção de um modelo de objetos relacionando as classes identificados (com herança, agregação e associações);
- Refinamento do modelo de requisitos:
  - Identificação de novos *use cases*;
  - Identificação de extensões de *use cases*;
  - Identificação de *use cases* abstratos: identificação de procedimentos comuns a mais de um *use case*;

Construção do modelo de análise:

- Identificação das classes de interface: três estratégias podem ser adotadas para a identificação das classes de interface: a partir dos atores (OOSE propõe que deve

haver pelo menos uma classe de interface para cada ator identificado[48]); a partir da descrição dos *use cases*, identificando as necessidades de interação; a partir da descrição da interface do sistema desenvolvida;

- Descrição textual das classes de interface identificadas, salientando papel e responsabilidades;
- Identificação das classes entidade: consiste em um refinamento da identificação já procedida na construção do modelo de requisitos;
- Descrição textual das classes entidade identificados, salientando papel e responsabilidades;
- Primeiro esboço de modelo de análise: agrupamento em modelos de objetos (separados por *use cases*), das classes entidade e de interface identificadas - classes podem estar presentes em um ou mais diagramas;
- Inclusão de associações;
- Identificação das classes de controle: funcionalidades necessárias a um *use case* que não se julgue adequadas às classes de interface e entidade podem ser alocadas a classes de controle. As classes de controle são determinadas a partir dos *use cases* e não tem uma correspondência direta com os elementos e conceitos do domínio (como ocorre com as classes determinados por outras metodologias); sua finalidade é suprir necessidades funcionais identificadas nos *use cases*[49]. A metodologia propõe como primeira identificação, associar uma classe de controle a cada *use case* - e depois refinar esta estrutura acrescentando ou excluinda classes de controle;
- Descrição textual das classes de controle identificadas, salientando papel e responsabilidades;
- Construir os diagramas de objetos: um por *use case*;
- Descrição textual dos *use cases*, salientando a interação entre objetos (ótica diferente da descrição do modelo de requisitos);
- Agrupar classes em subsistemas - classes com forte acoplamento funcional são colocadas no mesmo subsistema;

5.3.2 Passos do projeto

- Construir o primeiro esboço do modelo de projeto (estrutura estática de blocos), o que consiste em repetir o modelo de análise, substituído o símbolo das classes;
- Refinar o modelo de projeto a partir da realidade de implementação:
    - Identificar o ambiente de implementação (uso de SGBD, disponibilidade de biblioteca de classes etc.);
    - Incorporar conseqüências das condições do ambiente ao modelo de projeto :
        - Introduzir no modelo de projeto novos blocos (classes) que não tenham representação no modelo de análise;
        - Eliminar blocos do modelo de projeto;

---

48 Cabe aqui uma ressalva: na medida em que mais de um ator pode usar um mesmo dispositivo de interface, parece mais adequado pensar em pelo menos uma classe de interface para cada dispositivo de interface (painel de controle, impressora etc.), e não para cada ator.

49 As classes de controle constituem uma excentricidade em relação ao paradigma de orientação a objetos, na medida que não concentram informação, mas funcionalidade - e funcionalidade particular de um *use case*, que é algo muito mais sujeito a alterações que as classes de objeto do domínio de aplicação [COA 92]. Usar ou não classes de controle - bem como usar de forma intensiva ou moderada - é uma questão de estilo de análise. Pode-se optar por qualquer das hipóteses.

- Alterar blocos do modelo de projeto (dividir ou fundir blocos existentes);
- Alterar associações entre blocos do modelo de projeto (tendo em vista a forma como as associações serão implementadas);

- Construir os diagramas de interação: a princípio um para cada *use case* identificado[50] - consiste em uma formalização, bem como um refinamento da interação entre objetos descrita textualmente no modelo de análise;
- Descrição da interface dos blocos: consiste em definir os atributos e a assinatura dos métodos (sem o algoritmo), na sintaxe da linguagem de programação adotada para a implementação;
- Descrição do comportamento dinâmico dos blocos: construção de um diagrama SDL para cada módulo (descrição individual da classe de objetos), num nível de detalhamento tal que explicite os algoritmos dos métodos.

5.4 Exemplo de uso da metodologia OOSE

Análise

Construção do modelo de requisitos

Identificação de atores
- inicializador: procede a inicialização de uma partida;
- jogador: procede lances em uma partida.

Identificação de *use cases*:

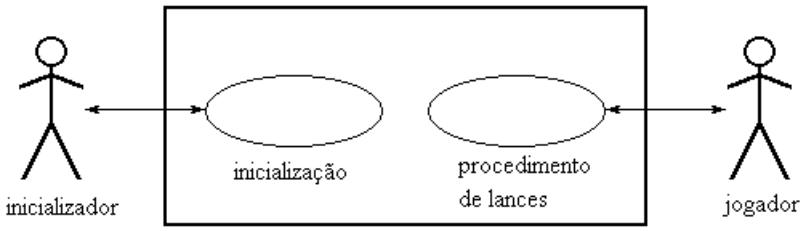


Figura 5.6 -diagrama de use cases

Descrição textual dos *use cases*

Inicialização:
- O sistema pergunta ao usuário (inicializador) o número de jogadores e a dimensão do tabuleiro;
- O inicializador fornece estes dados através do teclado;
- O sistema informa que irá proceder o sorteio para a determinação de quem inicia a partida;

---

50 O refinamento da descrição dos *use cases* aqui procedido, pode levar a identificação cursos de processamento alternativos (para tratamento de faltas, por exemplo), que podem originar extensões de *use cases*.

- O sistema determina o jogador que deve realizar um lance de dado e solicita que jogue;
- O jogador procede um lançamento de dados (um "click" de mouse);
- O valor obtido no lançamento do dado é apresentado no vídeo;
- O sistema determina, a partir dos valores obtidos nos dados, a ordem dos jogadores - em caso de empate o sistema pode solicitar que dois ou mais jogadores procedam novo lançamento de dado;
- O sistema apresenta no vídeo o estado inicial da partida: tabuleiro com peões posicionados na casa zero, ordem de jogadores estabelecida e definido o jogador que deve fazer o primeiro lance - neste momento o sistema está apto a que seja procedido o primeiro lance.

Procedimento de lances:

- O sistema determina o jogador que deve realizar um lance de dado e solicita que jogue;
- O jogador procede um lançamento de dados (um "click" de mouse);
- O valor obtido no lançamento do dado é apresentado no vídeo;
- O sistema determina a nova posição do peão do jogador atualiza a imagem do tabuleiro no vídeo (reposicionando este peão):
    - se posição atual for zero e obtiver um ou seis no dado, a nova posição é a casa um;
    - se posição atual for zero e obtiver um valor diferente de um ou seis, o peão permanece na casa zero;
    - se posição atual não for zero e a soma do valor da posição atual e do obtido no dado for maior que o valor da última casa, o peão permanece na mesma casa;
    - se posição atual não for zero e a soma do valor da posição atual e do obtido no dado não for maior que o valor da última casa, a nova posição é a casa correspondente a esta soma;
- O sistema informa quando, ao final do lance, indentificando um jogador como vencedor, o final da partida;
- Se o lance não implicar no final da partida, o sistema determina o jogador que deve proceder o próximo lance (pode ser o mesmo jogador, caso tenha obtido o valor seis no dado);
- Este procedimento se repete, até haver um vencedor.

Especificação da interface:

Elementos de interface: vídeo, teclado (para entrada de dados de inicialização) e mouse (para intervenção do usuário ao longo da partida). A figura abaixo apresenta um lay-out de tela para a situação particular de quatro jogadores (identificados por números, o que não é obrigatório) e um tabuleiro de vinte e duas casas.

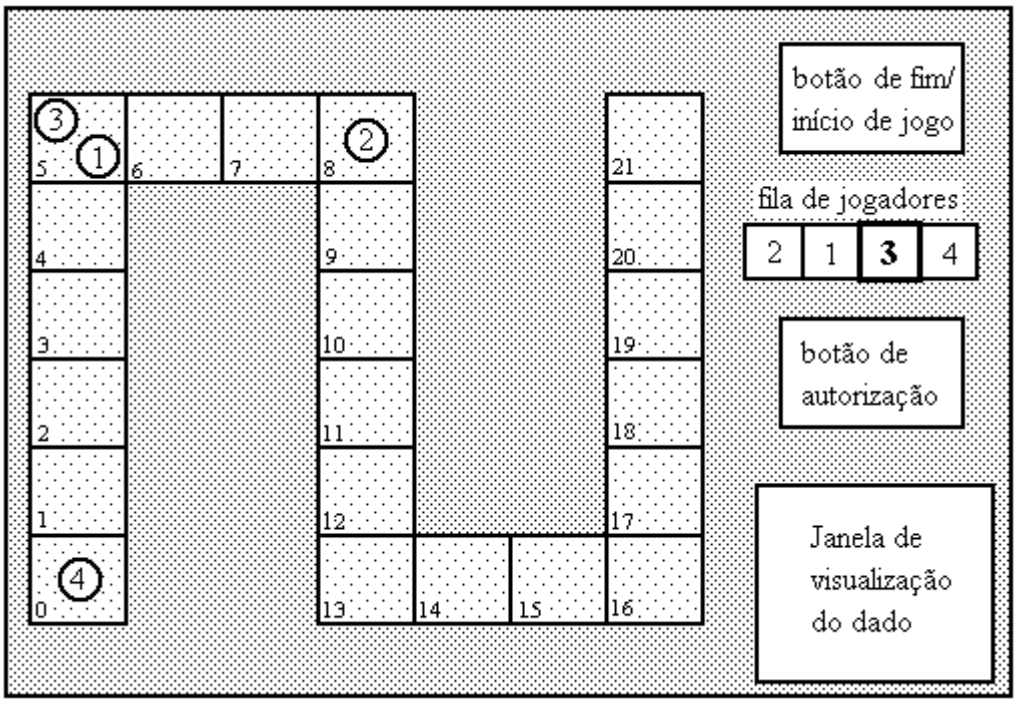


Figura 5.7 -lay-out de tela para o jogo Corrida

Modelo de objetos[51]:

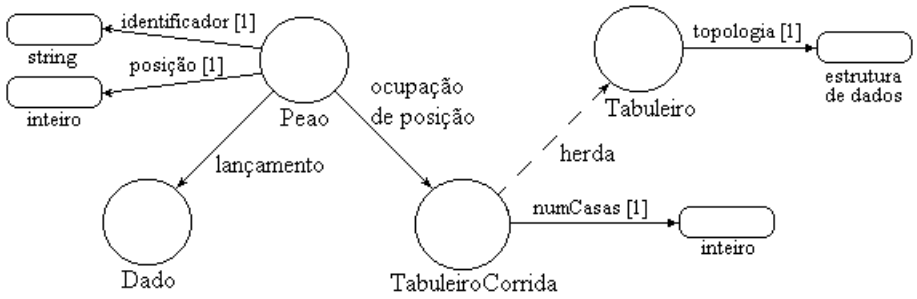


Figura 5.8 -modelo de objetos, parte do modelo de requisitos

Construção do modelo de análise

Descrição das classes identificadas:

classes entidade:

- Peão: existe um peão para cada jogador. Um peão possui um identificador e uma posição (que casa do tabuleiro ocupa, num determinado instante da partida).
- Dado: o jogo possui um dado, que é lançado por um jogador (peão). Gera valores aleatórios na faixa de 1 a 6 (valores inteiros).

---

51 Este primeiro modelo de objetos abrange somente as classes do domínio, com associações bastante genéricas. Em comparação com os modelos das metodologias anteriores, as classes que aqui faltam serão incluídas como dos tipos de interface e de controle, na fase de construção do modelo de análise - bem como as associações serão mais significativas. Este primeiro modelo se presta de fato, segundo proposto em OOSE, a ser um mecanismo de identificação e refinamento de *use cases*.

- Tabuleiro: apresenta uma topologia (número de casas) e regras de deslocamento - o número de casas no deslocamento do peão sobre o tabuleiro pode ou não ser igual ao valor obtido no dado.

classe de interface:

- InterfaceCorrida: definido a princípio como apenas uma classe[52], que executa os métodos responsáveis pela comunicação entre os usuários e o jogo;

classe de controle:

- CoordCorrida: definido como apenas uma classe, instancia os objetos necessários ao desenvolvimento de uma partida, procede as inicializações e coordena o desenvolvimento da partida; se utiliza dos serviços da classe InterfaceCorrida para fornecer e obter informações do usuário.

Use case "Inicialização":

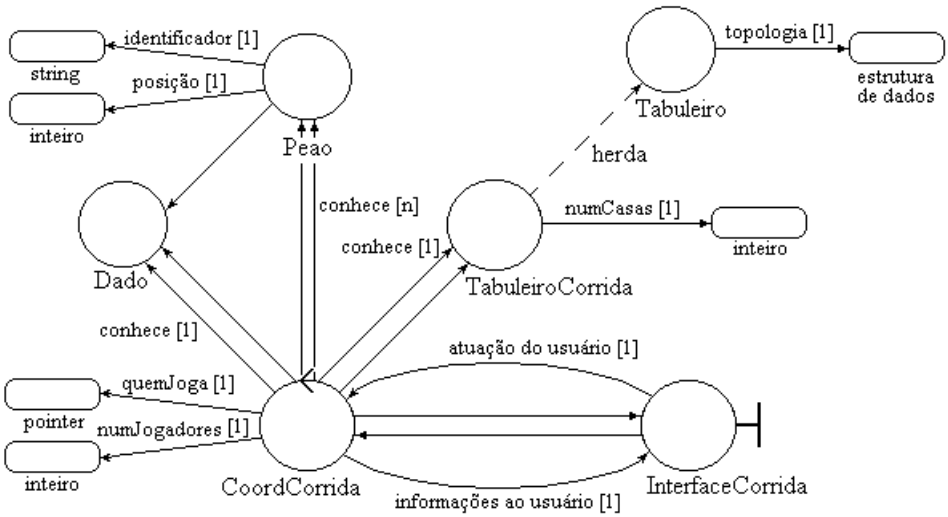


Figura 5.9 -modelo de objetos referente ao use case "inicialização"

Descrição textual do *use case* "Inicialização"

- instância de InterfaceCorrida (que será chamada interface daqui por diante) cria instância de CoordCorrida (que será chamada coordenador daqui por diante) - solicita método de criação de instância da classe;
- coordenador silicita à interface o número de casas do tabuleiro e o número de jogadores;
- interface solicita ao usuário estes dados;
- usuário fornece dados à interface;
- interface repassa estes dados ao coordenador;
- coordenador cria uma instância de Dado (que será chamada dado), uma instância de TabuleiroCorrida com o número de casas informado (que será chamada tabuleiro) e um número de instâncias de Peao igual ao número de jogadores (peão);

---

52 Como estabelecido no enunciado do problema, a especificação incluirá apenas uma classe responsável pela interface, abstraindo as classes extraídas de biblioteca, responsáveis pela interface gráfica. Quanto à possibilidade de estabelecer uma classe de interface por ator, levantada por OOSE, observa-se neste caso que não é preciso, pois os dois tipos de atores comunicam-se com o sistema através dos mesmos dispositivos.

- coordenador, através da interface, informa ao usuário que será procedido o sorteio da ordem dos jogadores;
- o procedimento a seguir será repetido para cada jogador:
    - coordenador, através da interface, solicita que o jogador proceda um lançamento de dado;
    - jogador procede lançamento (autoriza através do mouse);
    - coordenador solicita ao respectivo peão o lançamento de dado;
    - peão solicita método de lançamento ao dado;
    - dado retorna valor a peão;
    - peão retorna valor a coordenador;
    - com base no valor decrescente obtido nos dados, é estabelecida a ordem dos jogadores, que é apresentada através da interface - em caso de empate, o coordenador solicitará novos lançamentos por parte dos jogadores empatados;
- o estado inicial da partida (peões na casa zero) é mostrado pelo coordenador ao usuário, através da interface;
- coordenador, através da interface, solicita que o jogador determinado pelo sorteio, proceda o primeiro lance.

Use case "Procedimento de Lances":

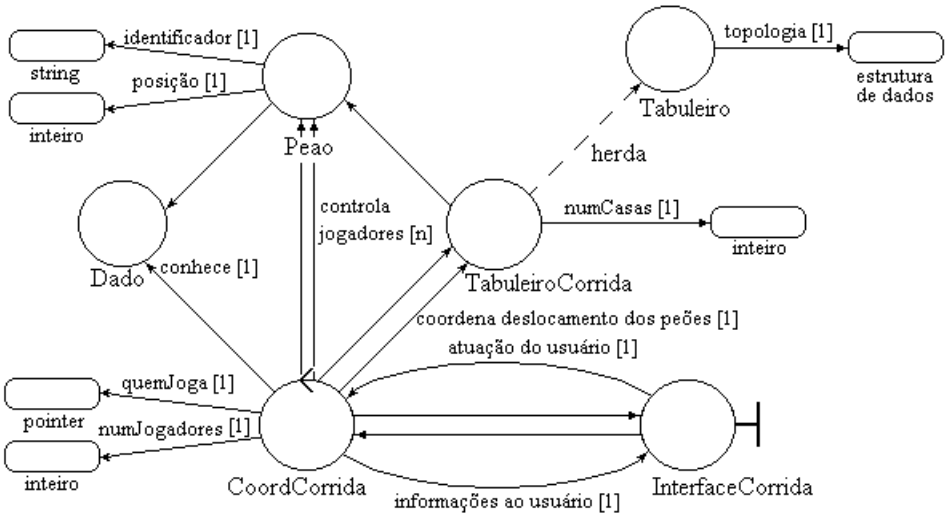


Figura 5.10 -modelo de objetos referente ao use case "procedimento de lances"

Descrição textual do *use case* "Procedimento de Lances"

- o coordenador solicita através da interface a autorização do jogador escalado para proceder o lance;
- o coordenador recebe autorização do jogador, através da interface (um "click" de mouse);
- coordenador solicita ao respectivo peão o lançamento de dado;
- peão solicita método de lançamento ao dado;
- dado retorna valor a peão;
- peão retorna valor a coordenador;

- o coordenador apresenta no vídeo o valor obtido no lançamento do dado (através da interface);
- o coordenador solicita ao tabuleiro o deslocamento do peão, passando o valor obtido no lançamento do dado;
- o tabuleiro determina a nova posição do peão:
    - se posição atual for zero e o valor obtido no dado for um ou seis, a nova posição é a casa um;
    - se posição atual for zero e o valor obtido no dado for um valor diferente de um ou seis, o peão permanece na casa zero;
    - se posição atual não for zero e a soma do valor da posição atual e do obtido no dado for maior que o valor da última casa, o peão permanece na mesma casa (nova posição igual à posição atual);
    - se posição atual não for zero e a soma do valor da posição atual e do obtido no dado não for maior que o valor da última casa, a nova posição é a casa correspondente a esta soma;
- se a nova posição for diferente da atual, o tabuleiro solicita o reposicionamento do peão;
- o coordenador verifica ao final do lance, se o peão é vencedor - caso isto ocorra, informa através da interface, o vencedor e o final da partida;
- Se o lance não implicar no final da partida, o coordenador determina o jogador que deve proceder o próximo lance (pode ser o mesmo, caso o valor obtido no dado tenha sido seis);
- Este procedimento se repete, até haver um vencedor.

Projeto

Construção do modelo de blocos[53]

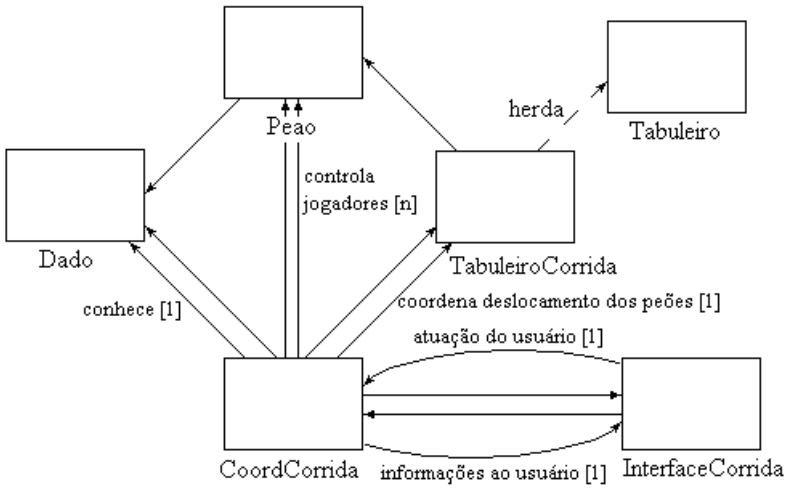


Figura 5.11 -modelo de blocos

[53] Optou-se por apenas um diagrama para o sistema, ao invés de um por *use case*, como procedido na análise. Com isto, em termos de relações de conhecimento, prevaleceram as mais significativas - por exemplo, na inicialização só é relevante estabelecer que o coordenador conhece a existência do tabuleiro, mas no procedimento de lances, a relação é mais significativa, prevalecendo sobre a outra. Um outro aspecto a ressaltar é que o bloco InterfaceCorrida poderia ser refinado para conter as classes extraídas de biblioteca, que serão usadas para a construção da interface gráfica.

Construção dos diagramas de interação

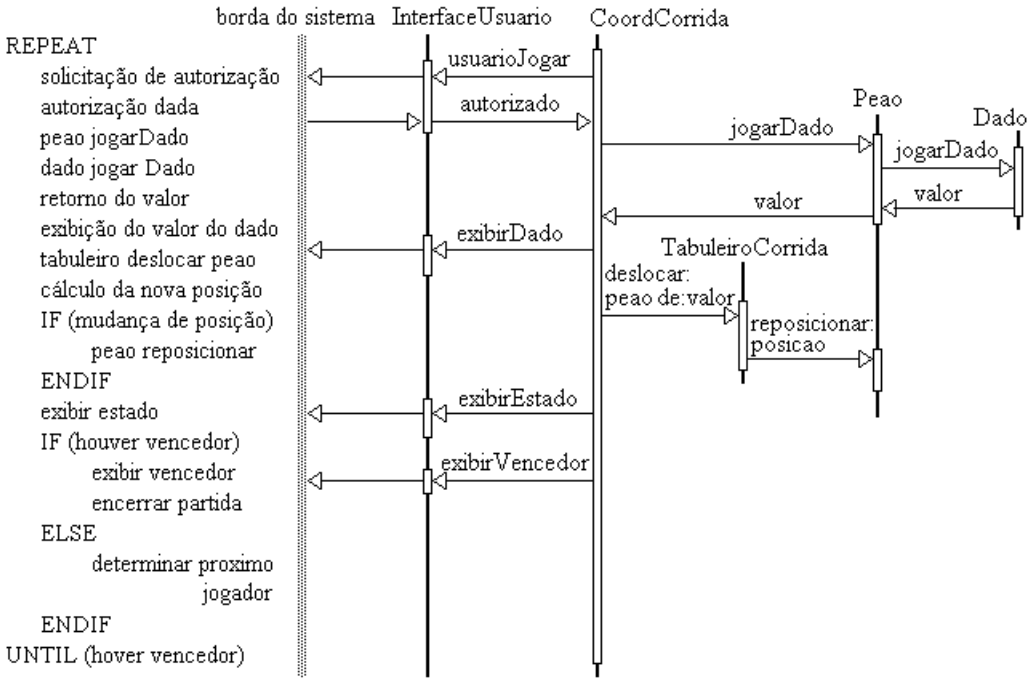


Figura 5.12 -diagrama de interação para o use case "Procedimento de Lances"

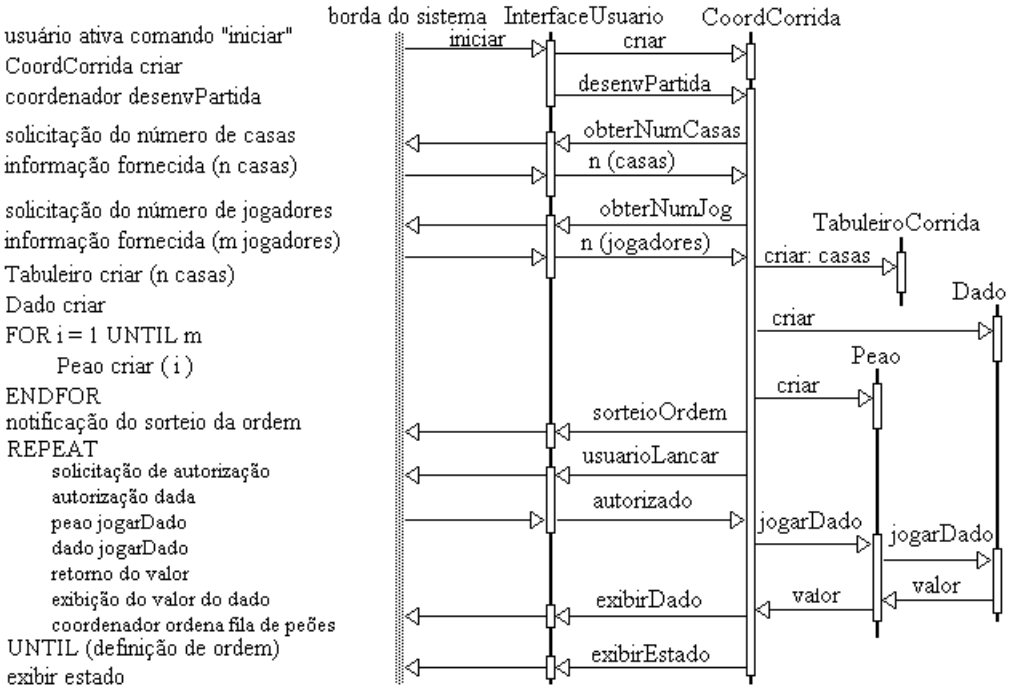


Figura 5.13 -diagrama de interação para o use case "Inicializar"

Especificação da interface dos blocos (sintaxe de Smalltalk)

class name **CoordCorrida**
superclass Object
instance variables
    quemJoga
    numJogadores
    interface
    tabuleiro
    dado
    p1 ... pn[54]
instance methods
    desenvPartida

class name Peao
superclass Object
instance variables
    identificador
    posicao
class methods
    new: identificador
instance methods
    setIdent: identificador
    getIdent
    reposicionar: posicao
    getPosic
    jogarDado: dado

class name Tabuleiro
superclass Object
instance variables
    numCasas
class methods
    newWith: casas
instance methods
    setDimens: casas
    getDimens
    deslocar: peao de: casas

class name Dado
superclass Object
instance methods
    jogarDado

class name InterfaceCorrida
superclass Object
instance methods

---

54 Isto de fato, se refere a uma lista de peões.

obterNumCasas
obterNumJog
exibirEstado: estado
sorteioOrdem
usuarioLancar
exibirDado: valor
usuarioJogar
exibirVencedor: vencedor
iniciar

Construção dos diagramas SDL (descrição do comportamento dinâmico dos blocos)

Classe CoordCorrida:

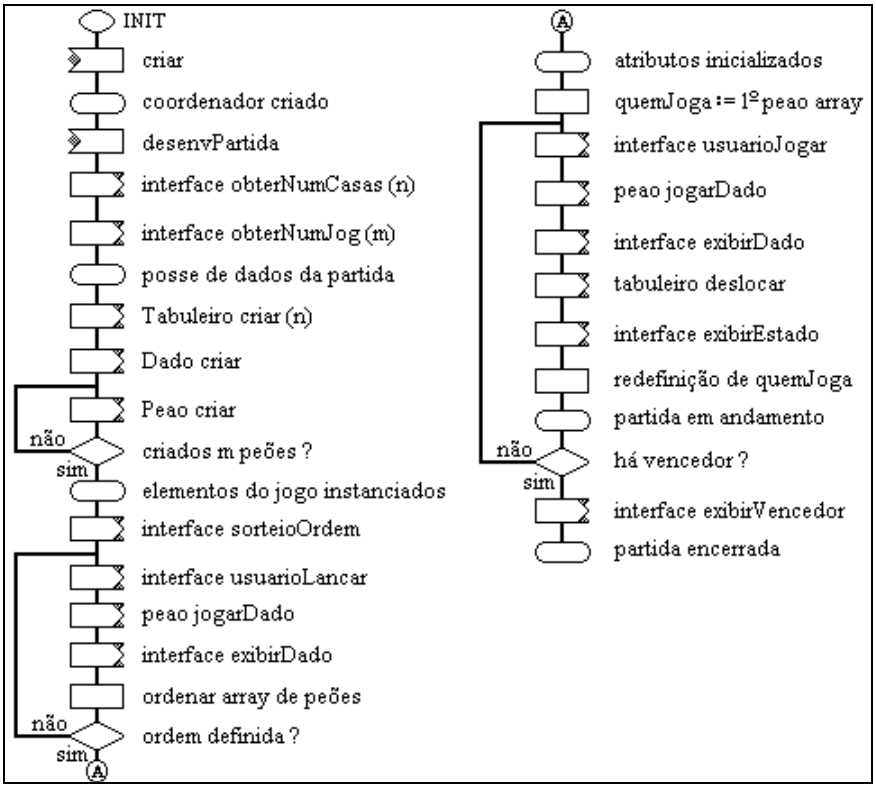


Figura 5.14 -diagrama SDL da classe CoordCorrida

Classe Peao:

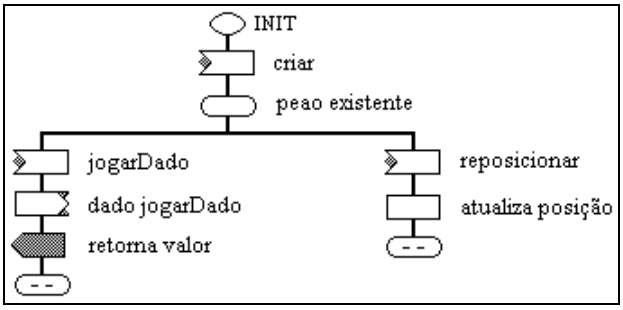


Figura 5.15 -diagrama SDL da classe Peao

Classe Tabuleiro:

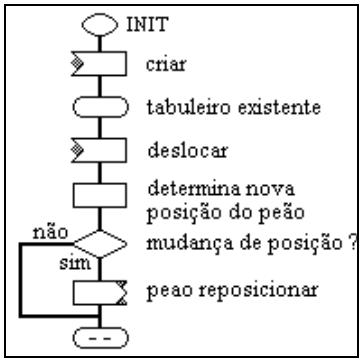


Figura 5.16 -diagrama SDL da classe Tabuleiro

Classe Dado:

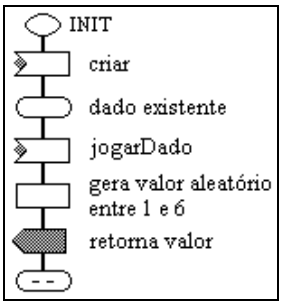


Figura 5.17 -diagrama SDL da classe Dado

Classe InterfaceCorrida:

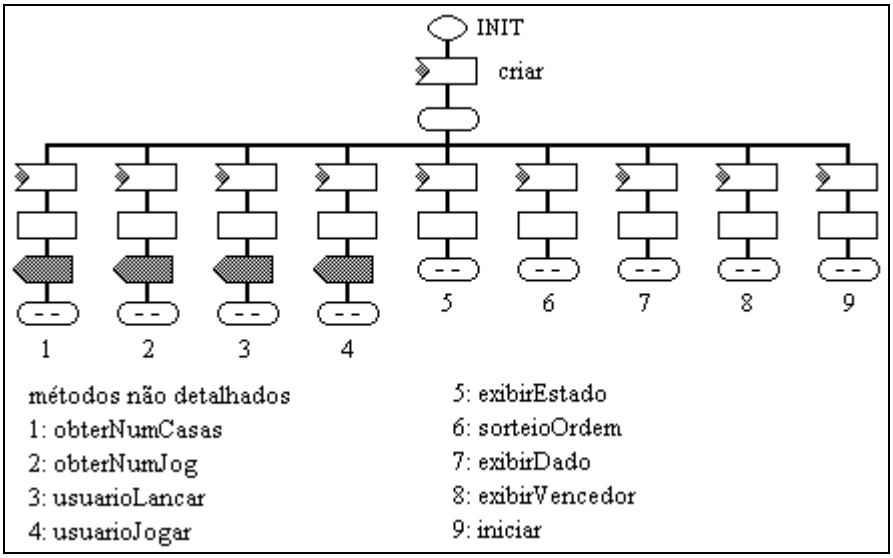


Figura 5.18 -diagrama SDL da classe InterfaceCorrida

6 Metodologia de Martin e Odell

6.1 Visão geral da metodologia

As metodologias de Coad e Yourdon e OMT, anteriormente descritos buscam uma descrição da aplicação como um conjunto de tipos abstratos de dados (que constitui um pilar de sustentação da orientação a objetos). Ou seja, descrevem objetos (de fato, classes), que são elementos encapsuladores de conhecimento, que contêm dados (atributos) e métodos, que manipulam estes dados.

A metodologia de Martin e Odell [MAR 95], a nível de análise, trabalha com uma visão em que os atributos de um objeto sob descrição correspondem a outros objetos associados a ele. Conceitualmente, não há em princípio, afronta ao paradigma de orientação a objetos - haja vista Smalltalk, em que todos os dados são objetos. Porém, a descrição gerada na análise tende a diferir bastante daquela gerada nas metodologias citadas, extrapolando como se demonstrará no próximo item, os princípios estabelecidos pelo paradigma de orientação a objetos.

A metodologia de Martin e Odell caracteriza-se como uma metodologia propriamente dita, apenas para análise[55] - neste caso são propostas técnicas de modelagem, como também um processo de desenvolvimento. A nível de projeto os autores sugerem o refinamento dos modelos de análise, porém não propõem uma seqüência de passos - apenas estabelecem diretrizes a serem seguidas - bem como não estabelecem técnicas de modelagem adicionais, por exemplo, para a descrição de algoritmos.

A descrição da aplicação gerada apresenta duas linhas distintas: uma descrição estrutural e uma descrição dinâmica. A visão estrutural se refere a uma descrição estática de como os objetos se situam, dentro da realidade sob modelagem. Em contrapartida, a visão dinâmica se refere à maneira segundo a qual os objetos mudam ao longo do tempo, dentro de suas estruturas definidas. O principal modelo da descrição estrutural é o esquema de objetos e o principal modelo da descrição dinâmica é o esquema de eventos.

Segundo Martin e Odell, uma metodologia deve se apoiar numa base dinâmica que incorpore a estrutural. Assim, a ênfase da metodologia de análise é a construção do esquema de eventos; paralelamente aos passos desta construção, a descrição estática vai sendo composta. No projeto é proposto um refinamento dos modelos da análise visando chegar a operações básicas, e um mapeamento[56] dos elementos dos modelos gerados para a estrutura de uma linguagem de programação orientada a objetos[57].

---

55 Apesar disto, o presente texto utilizará a expressão "metodologia de Martin e Odell" para referir-se tanto a análise quanto a projeto, para manter a uniformidade com os textos que descrevem os demais metodologias. Fica registrada a ressalva que a expressãa *metodologia*, para referir-se a projeto, a rigor, é inadequada.

56 Não existe na obra de Martin e Odell qualquer técnica de modelagem que registre este mapeamento. Não é também citada a necessidade ou não de seu registro. Isto pode dar a entender que este mapeamento deve ser executado pelo elemento responsável pela implementação, porém, isto não seria uma ótica adequada, na medida em que não é um processo de tradução trivial - o porquê disto será melhor explorado no item referente a projeto.

57 A metodologia inclui sugestões para o mapeamento para linguagens não orientadas a objetos e sistemas de bancos de dados, de forma resumida. Estas sugestões são baseadas, como referenciam os autores, em [RUM 94].

6.1.1 A visão conceitual de objetos e atributos de objetos, presente na metodologia de Martin e Odell

Para ilustrar a visão conceitual de objetos e atributos de objetos da metodologia, recorrer-se-á a um exemplo, que pode ser visualizado na ilustração abaixo. A questão colocada é descrever uma *empregada*, de uma forma orientada a objeto, e supondo a existência de uma classe Pessoa[58].

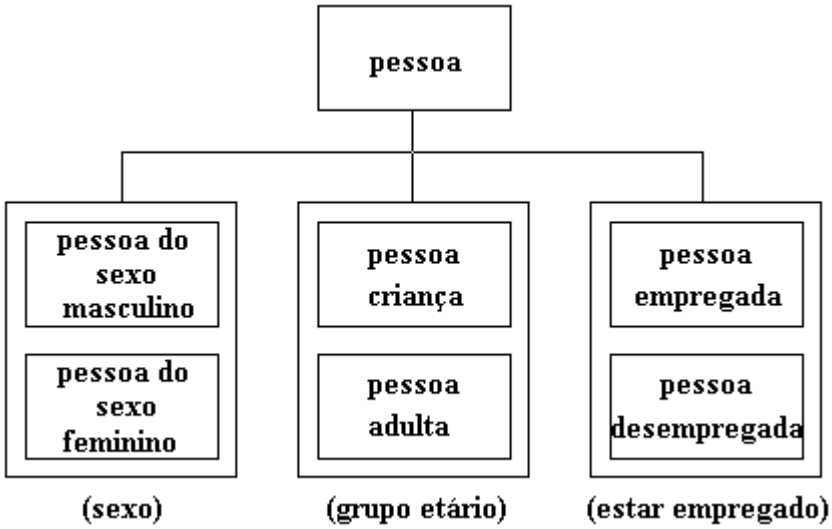


Figura 6.1 - partições de uma classe

Segundo definido por várias metodologias (como por exemplo OMT e OOSE, apresentadas nos capítulos anteriores), a classe Pessoa poderia ter os seguintes atributos (não exclusivamente): sexo, grupoEtario e estarEmpregado. Uma empregada (uma instância de pessoa, que possuiria um identificador) teria respectivamente, os valores mulher (pessoa do sexo feminino), adulto (pessoa adulta) e empregado (pessoa empregada), para os três atributos citados. Esta visão é mapeada diretamente para uma linguagem de programação orientada a objetos[59]. A alteração de um objeto, por exemplo, o objeto da situação descrita passar à condição de desempregado, consiste na troca de um valor de atributo.

Martin e Odell apresentam o conceito de partição de tipo como uma divisão de uma classe em subclasses disjuntas. Uma classe pode ser particionada de formas diferentes. Assim, partições múltiplas de uma classe são admitidas e um objeto particular pode se localizar em várias partições. Dentro de uma partição, porém, os objetos das subclasses devem ser exclusivos e não-sobrepostos. A figura acima apresenta a divisão da classe pessoa em três partições: sexo, grupo etário e estar empregado. Um objeto pessoa seria uma instância de uma sub-classe de cada partição. Uma empregada qualquer, seria então simultaneamente, uma instância de pessoa do sexo feminino, pessoa adulta e pessoa empregada. Isto define um estado de objeto, que

---

58 A possibilidade de criar uma classe empregada como sub-classe de Pessoa ou como sub-classe de uma ou mais sub-classes de Pessoa, não será tratada. A intenção é que tanto na modelagem segundo os autores, como na modelagem baseada em outras metodologias, uma empregada (uma instância particular) seja uma instância da classe Pessoa.

59 Ou para um estilo de implementação orientado a objetos, com uma linguagem de programação não-orientada a objetos.

segundo os autores, é a coleção de todas as classes de objeto que se aplicam a um objeto[60].

A alteração do objeto da situação descrita, por exemplo, passar à condição de desempregado, consiste na remoção (desclassificação) do objeto da classe pessoa empregada e inclusão (classificação) em pessoa desempregada - mantidas as demais classificações. Processo de classificação ou desclassificação que necessite ocorrer em tempo de execução, é denominado classificação dinâmica. Os autores defendem que um suporte para classificação dinâmica e múltipla, ausente atualmente das linguagens de programação orientadas a objetos, é importante, bem como que estes mecanismos "representam uma extensão desejável do paradigma de orientação a objetos"[61].

Em termos de implementação, a visão de Martin e Odell pode levar a um resultado equivalente ao de outras metodologias, porém não através de uma tradução trivial.

Quanto ao paradigma de orientação a objetos, é estranha a possibilidade de um objeto ser uma instância de mais de uma classe. Aplicando o conceito de herança, definido nas metodologias anteriores, poderiam ser criadas seis sub-classes abstratas de Pessoa (as seis sub-classes da figura) e poderia ser criada uma classe Empregada, que por herança múltipla seria subclasse de PessoaDoSexoFeminino, de PessoaAdulta e de PessoaEmpregada. Uma instância de Empregada poderia ser vista como um elemento das superclasses, porém ligada diretamente (pela capacidade de gerar instâncias) apenas à classe Empregada.

Um último aspecto a salientar é que a ausência de atributos tende a tornar o esquema de objetos (correspondente ao modelo de objetos de outras metodologias) mais complexo que em outras metodologias: por exemplo, ao invés de representar como um elemento do diagrama, um objeto com três atributos, o estilo estabelecido demanda a representação de quatro objetos associados. Com isto, também se corre o risco de tornar os objetos do domínio menos destacados no diagrama (em função da presença de outros objetos que representam seus atributos).

6.1.2 Análise e projeto

A análise e o projeto tem dois aspectos. O primeiro preocupa-se com as classes de objetos, relações entre objetos e herança, e é chamado análise da estrutura do objeto (AEO) e projeto da estrutura do objeto (PEO). O outro aspecto preocupa-se com o comportamento dos objetos e com aquilo que acontece a eles no decorrer do tempo e é denominado análise do comportamento do objeto (ACO) e projeto do comportamento do objeto (PCO).

A análise da estrutura do objeto e a análise do comportamento do objeto ocorrem em paralelo, segundo a metodologia. De fato, a ênfase é dentro de uma abordagem top-down, identificar os eventos gerados pelo sistema; a partir deles, as operações que produzem estes eventos e os fatores que levam à realização da operação (inclusive eventos). A identificação de novos eventos leva à aplicação cíclica deste procedimento

---

60 Diferente da definição tradicional que estabelece que o estado de um objeto é definido pelos valores de seus atributos. Na análise de Martin e Odell os atributos não são utilizados, diferente do que ocorre em outras metodologias.

61 Uma visão alternativa à de Martim e Odell é o uso de atributos na análise, como comentado no exemplo - o que também evita um "gap" conceitual entre a visão da análise e a visão do projeto (sendo que esta deve estar próxima da realidade de implementação).

até que restem apenas eventos externos - que não precisam ser modelados. Isto representa a construção do modelo dinâmico. O modelo estrutural vai sendo construído à medida em que junto com a identificação de eventos, identificam-se estados de objetos - o que reflete as classes existentes e as associações entre elas. Uma visão inicial de objetos presentes no domínio modelado (visão estrutural) é necessária para a construção do modelo dinâmico. Assim, não é possível afirmar que a construção de um modelo seja precedida pela construção de outro.

Na análise da estrutura do objeto as seguintes informações são identificadas:

- As classes e como elas estão associadas (o que compõe o esquema do objeto). Essa informação servirá como base para definir no projeto, que classes serão implementadas[62] e suas estruturas de dados.
- Organização das classes em uma hierarquia de generalização, com base em herança.
- Organização das classes em uma hierarquia de composição, explicitando objetos compostos pela agregação de outros objetos.

Na análise do comportamento do objeto as seguintes informações são identificadas:

- O conjunto de estados que cada objeto pode assumir.
- As transições de estado que ocorrem. Estas transições são representadas em diagramas de transição de estado.
- Os eventos - um evento representa uma mudança de estado.
- Quais operações se desenvolvem. Um esquema de eventos mostra a seqüência de operações e eventos.
- Quais interações ocorrem entre objetos. Um diagrama pode mostrar mensagens entre classes.
- As regras de gatilho usadas para reagir aos eventos.

No projeto da estrutura e do comportamento do objeto as seguintes informações são identificadas:

- Quais classes serão implementadas.
- Que estruturas de dados cada classe empregará.
- Os métodos que cada classes oferecerá.
- A forma de implementação da herança.
- Adaptação de classes pré-existentes, para reutilização.

6.1.3 Técnicas de modelagem para a descrição de uma aplicação

Para a modelagem estrutural são definidos quatro modelos: diagrama de objeto-relacionamento, diagrama de generalização, diagrama de composição e esquema de objetos. O esquema de objetos contém o conjunto de informação dos três primeiros diagramas e é usado alternativamente a estes.

---

62 Os autores utilizam duas expressões diferentes para se referirem ao conceito de classe de objetos (definido no paradigmade orientação a objetos). Na análise utilizam a expressão tipo de objeto, e no projeto, classe. Optou-se neste texto por utilizar apenas a expressão classe para as duas etapas para manter uniformidade com as outras descrições de metodologias - dado que também não há distorção de conceito. As expressões diferentes usadas pelos autores refletem que nem todas as classes definidas na análise se refletirão em classes na etapa de projeto (as classes definidas no projeto mantêm uma relação de um para um com as classes implementadas) - é esperado que algumas classes da análise se tornem atributos de classe no projeto, e conseqüentemente, na implementação.

O diagrama de objeto-relacionamento é exatamente o diagrama ER [CHE 76], onde são representados os objetos (sem métodos) e suas associações (relacionamentos). O diagrama de generalização é uma árvore de classes que explicita a estrutura de subclasses e superclasses. O diagrama de composição é uma representação de agregação. Cada relacionamento de agregação conecta duas classes: a do objeto composto e a do objeto parte, explicitando a cardinalidade.

O esquema de objetos une num único diagrama as informações dos demais: as classes, sua estrutura hierárquica, composição e associações. A orientação dos autores é que em princípio a modelagem estrutural se proceda através de um esquema de objetos, porém se este procedimento resultar em um diagrama confuso, pode-se usar alternativamente os outros três, separando as informações em diagramas distintos.

Para a modelagem dinâmica são definidos quatro técnicas de modelagem: esquema de eventos, diagrama de transição de estado (diagrama fence), diagrama de mensagens e diagrama de fluxo de objetos.

O esquema de eventos é o principal mecanismo de modelagem dinâmica. Seus componentes são eventos, operações que mudam o estado de objetos e levam à ocorrência de eventos, regras de gatilho e condições de controle, que regem a realização de operações.

O diagrama de transição de estado (diagrama fence) é composto pelos estados de um objeto e setas indicando todas as possíveis transições de estado. O diagrama de mensagens apresenta as classes e as mensagens trocadas entre elas - centraliza em um único diagrama a interação entre classes.

O diagrama de fluxo de objetos expressa o comportamento funcional do sistema. Baseia-se no diagrama de fluxo de dados (DFD). Ao invés de depósitos de dados utiliza objetos. Seus elementos são objetos (do sistema), atividades que consomem e produzem objetos, agentes externos e fluxos, que interligam estes elementos. Segundo os autores, quando o domínio do problema é "grande ou intricado demais para ser entendido usando-se técnicas comportamentais (pelo menos inicialmente)", uma visão funcional é o caminho para o início da modelagem do sistema[63].

6.2 Elementos sintáticos das técnicas de modelagem da metodologia de Martin e Odell

6.2.1 Diagrama de objeto-relacionamento

O diagrama de objeto-relacionamento como já citado, corresponde ao diagrama ER. A figura abaixo contém os elementos sintáticos do diagrama de objeto-relacionamento. Seus elementos são as classes (correspondentes às entidades do diagrama ER) e as associações (relacionamentos) entre objetos. As classes são representadas por retângulos contendo o nome da classe; as associações (tipos de associações[64]), por traços interligando as classes, rotulados com nomes de associação.

---

63 Esta abordagem de partir inicialmente para a modelagem funcional, para promover um melhor entendimento do domínio, no caso de sistemas complexos, é semelhante ao que é feito em OMT, com o uso de DFD.

64 Uma associação representada num diagrama particular corresponde a um conjunto de associações entre instâncias de classes. Daí a denominação tipo de associação usada pelos autores (tipo como sinônimo de classe). No presente texto a expressão tipo será suprimida, ficando implícita a semântica acima descrita.

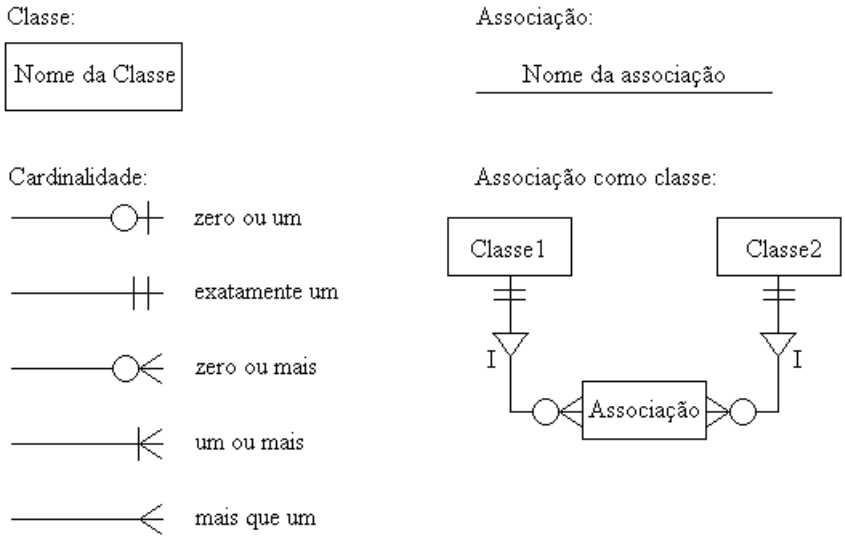


Figura 6.2 - elementos sintáticos do diagrama de objeto-relacionamento

A cardinalidade é representada na associação. Os autores apresentam três possibilidades de representação: representação numérica da cardinalidade máxima e mínima, representação simbólica idêntica à apresentada na metodologia OMT [RUM 94] e a notação de pés de corvo, ilustrada na figura 6.2. A semântica da cardinalidade é idêntica à apresentada na metodologia OMT.

Associações podem ser representadas como classes. Neste caso a conexão da classe associação com as classes associadas se dá utilizando o símbolo de composição, com a letra I. Associações podem ter atributos. A representação disto - coerente com a visão de atributo dos autores - corresponde a uma associação que liga a associação com atributo, à classe que representa o atributo. Um tipo de associação pode ser representado como um subtipo de outra associação. Isto é representado com o símbolo de generalização interligando as associações.

6.2.2 Diagrama de composição

A composição (agregação) expressa que um objeto é constituído a partir do agrupamento de outros objetos. O diagrama de composição reúne em um diagrama todas as composições verificadas no sistema descrito. A figura abaixo contém os elementos sintáticos da metodologia, para a construção do diagrama de composição.

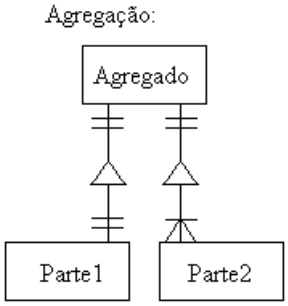


Figura 6.3 - elementos sintáticos do diagrama de composição

As representações de classes e de cardinalidade de composição são idênticas às do diagrama de objeto-relacionamento. A representação de composição é feita com uma seta apontando para a classe composta.

6.2.3 Diagrama de generalização

O diagrama de generalização expressa a estrutura hierárquica de herança das classes do sistema descrito. A metodologia apresenta três formatos possíveis para a construção do diagrama de generalização, como ilustrado nas figuras abaixo.

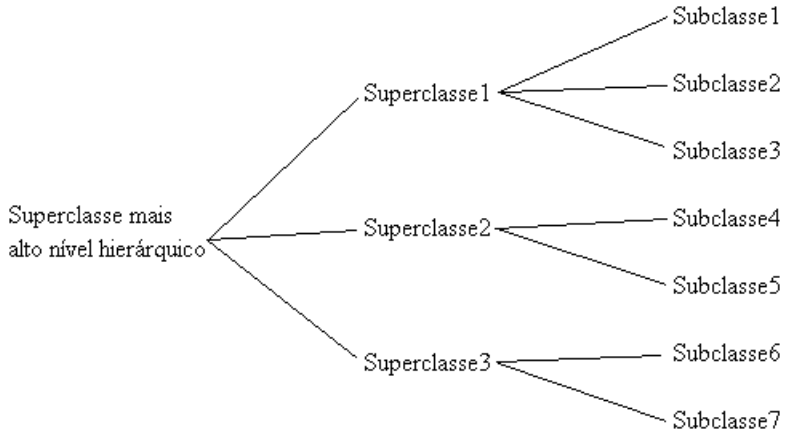


Figura 6.4 - diagrama fern

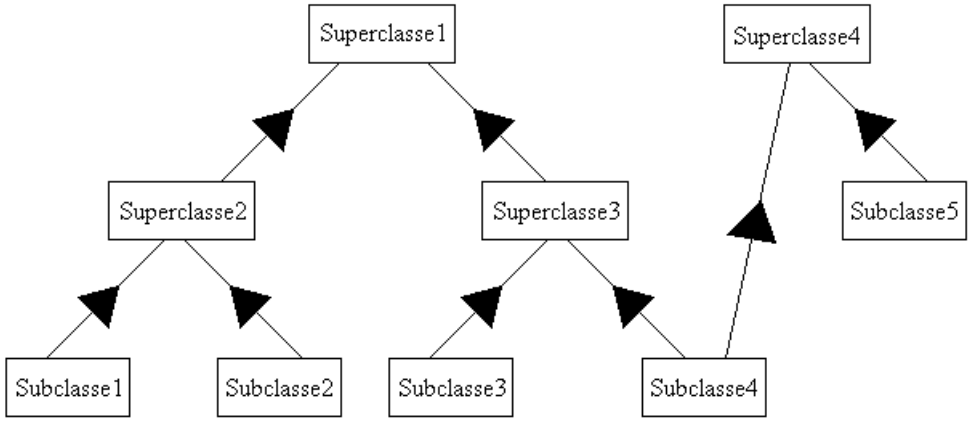


Figura 6.5 - estrutura hierárquica vertical, para representação de herança

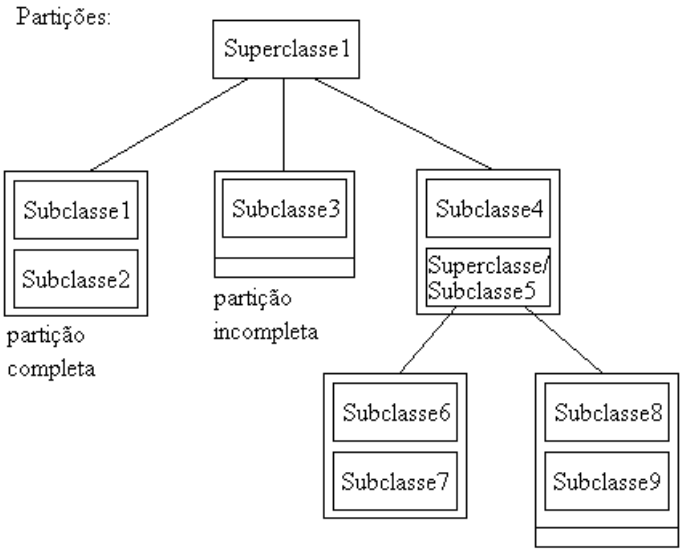


Figura 6.6 - diagrama de partições, para representação de herança

Na primeira representação, o diagrama fern, a estrutura hierarquica é descrita de forma horizontal - as classes hierarquicamente superiores são representadas mais à esquerda. Superclasses e subclases são interligadas por traços (que explicitam a estrutura hierárquica). É possível representar herança múltipla, porém não há explicitação se as subclasses de uma classe são ou não disjuntas. Instâncias de classes podem ser incluídas no diagrama fern, unindo o identificador da instância à classe correspondente, por uma linha descontínua.

Numa segunda forma de representação, a hierarquia se desenvolve de forma vertical, e a conexão entre superclasses e subclasses é feita por um traço com uma seta cheia (para diferenciar da seta que representa composição).

A terceira forma é o uso de partições, que já foram descritas. Cada partição agrupa subclasses disjuntas. Uma classe pode ser várias partições. Uma partição pode ser completa (se as classes da partição contêm todas as instâncias possíveis da superclasse) ou incompleta (se existem instâncias da superclasse que não pertencem a nenhuma das classes incluidas na partição). A representação de partição é feita por um retângulo que contém subclasses. Esta representação, como as demais, admite qualquer quantidade de níveis hierárquicos. O retângulo que representa uma partição incompleta contém um traço na parte inferior.

6.2.4 Esquema de objetos

O esquema de objetos reúne as informações da modelagem estrutural, conforme proposto pelos autores[65]. Um esquema de objetos consiste na união das informações - e,

---

65 A metodologia não propõe o uso de dicionário de dados ou outro mecanismo de descrição mais refinada das classes, porém, numa rápida comparação com as metodologias anteriores, fica clara a maior expressividade destas em relação à metodologia de Martin e Odell. Isto evidencia a necessidade de mecanismos adicionais para a modelagem estrutural. A omissão disto por parte dos autores ou transfere este refinamento para a etapa de projeto (em que a metodologia não propõe técnicas de modelagem), onde os autores afirmam que serão definidas as estruturas de dados, ou transfere implicitamente a responsabilidade de acessar informações mais refinadas ao ambiente CASE, onde supõem que a metodologia tenha sido implantada.

conseqüentemente da simbologia - dos três diagramas anteriores. No caso da representação de herança, a notação do diagrama fern não é adequada ao esquema de objetos. A figura abaixo apresenta um exemplo de esquema de objetos.

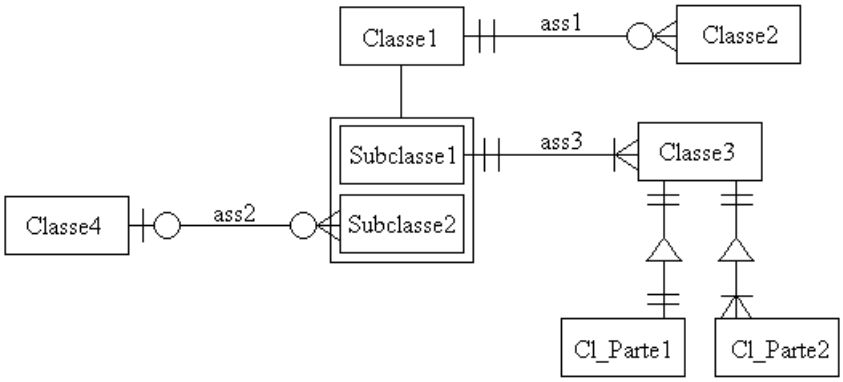


Figura 6.7 - exemplo de esquema de objetos

6.2.5 Esquema de eventos

O núcleo da modelagem dinâmica é o esquema de eventos. A figura abaixo apresenta a representação dos seus elementos sintáticos. É construído a partir de eventos, operações, regras de gatilho e condições de controle. Representa a evolução de estados de um objeto (semelhante a uma máquina de estado finito), admite estruturação e representação de concorrência.

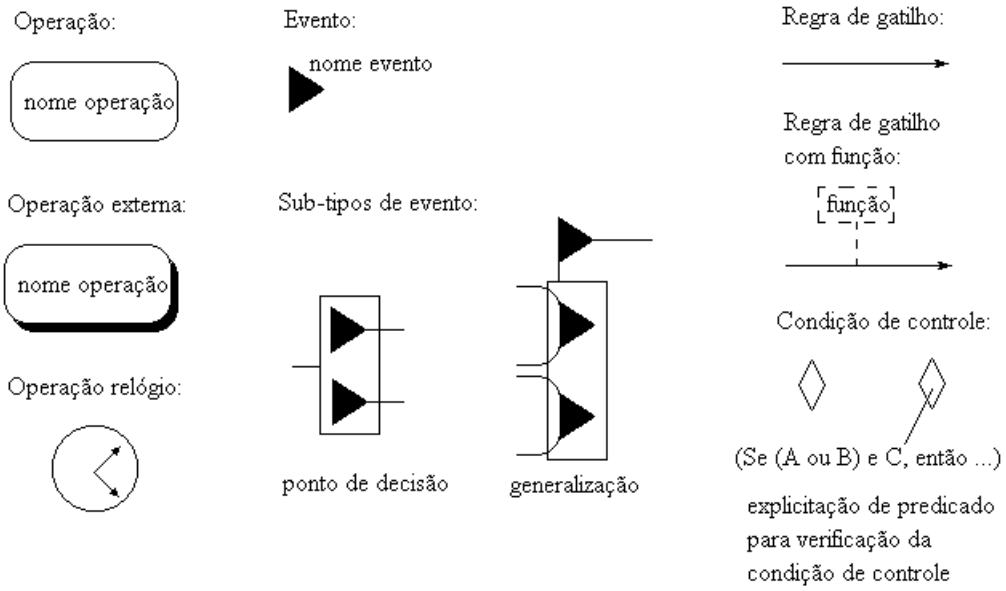


Figura 6.8 - elementos sintáticos do esquema de eventos

Um evento indica que alguma coisa ocorreu e que o sistema (alguma parte dele) deve reagir a isto. Assim, os eventos são nesta metodologia, o centro da especificação do comportamento dos objetos. O evento registra uma mudança de estado de um objeto

e sempre resulta do término de uma operação[66]. A metodologia utiliza a noção de subtipos de evento, permitindo compor uma estrutura hierárquica. A representação de subtipos de evento pode ser usada para expressar a generalização de eventos (qualquer dos subtipos de evento que ocorra corresponde ao superevento) ou pontos de decisão em processamento (a ocorrência de um superevento corresponderá à ocorrência mutuamente exclusiva de um de seus subtipos). A representação de eventos no diagrama é feita através de triângulos cheios, conectados às operações, e a de subtipos de eventos, consiste em agrupá-los em um retângulo.

Enquanto os eventos representam ocorrências de mudanças de estado, as operações correspondem às unidades de processamento que fazem a mudança. Operações são representadas através de retângulos de cantos arredondados. Operações externas (ao escopo descrito no esquema de eventos) são representadas com sombreamento. Uma operação relógio é um mecanismo para representar a ação do tempo no comportamento do objeto (que uma informação deve ser coletada de hora em hora, por exemplo). O mecanismo de refinamento dos esquemas de eventos consiste em descrever uma operação a partir de um novo esquema de eventos - o que viabiliza a estrututação. Operações, numa ótica mais funcional, também podem ser descritas (refinadas) em diagramas de fluxo de objetos[67].

Condição de controle é um elemento que pode ser associado a uma operação, condicionando o seu início a um conjunto de requisitos. O processamento continua além desse ponto somente quando o conjunto de requisitos é satisfeito. A condição de controle é uma espécie de guarda. É representada por um losângulo - ou mais de um - na "entrada" da operação (na "saída" da operação é representado um ou mais eventos).

Regras de gatilho, representadas no esquema de objetos por setas que conectam um evento a uma ou mais operações, representam a invocação de operações em função da ocorrência de eventos. Uma regra de gatilho tem associada uma função responsável por coletar e fornecer à operação a que está conectada, os argumentos necessários à sua execução. As finalidades básicas de uma regra de gatilho são então, detectar um evento e invocar uma ou mais operações - adicionalmente, pode ter que preencher argumentos da operação e avaliar condição de controle. Como salientam os autores, o uso de regra de gatilho representa uma abordagem conceitual sem correspondência direta com mecanismos de linguagens de programação orientadas a objetos.

6.2.6 Diagrama de transição de estado (diagrama fence)

O diagrama de transição de estado é composto pelos estados de um objeto e todas as possíveis transições de estado. Os estados são representados por traços horizontais com o nome do estado; transições, por setas interligando estados, como

---

66 Nesta metodologia uma operação se realiza e o seu término produz um evento - que leva à execução de outra operação, e assim por diante. A metodologia OMT também utiliza a noção de evento, porém, lá a ocorrência de um evento produz uma transição de estado. Nos dois casos há uma ligeira diferença no conceito de evento: em OMT o evento é o agente que leva o objeto de um estado a outro; na metodologia de Martin e Odell o evento é a constatação de que a mudança de estado ocorreu - e o resultado de uma mudança de estado é um novo estado. Assim, se um mesmo comportamento dinâmico for representado pelas duas formas, os nomes atribuídos aos eventos de Martin e Odell corresponderão aos nomes atribuídos aos estados de OMT.

67 De fato, a metodologia admite que uma operação seja refinada a partir de um esquema de eventos e de um diagrama de fluxo de objetos, se sua complexidade demandar tal nível de detalhamento.

ilustrado na figura abaixo. Um sinal "+" associado a um estado indica a existência de um diagrama onde o estado é expandido em subestados. A metodologia admite que mais de um diagrama seja associado a um objeto, para representar conjuntos de subestados complementares. O diagrama de transição de estado representa de uma forma diferente o mesmo tipo de informação do esquema de eventos, porém descrevendo cada classe separadamente. Os autores propõem seu uso como intrumento auxiliar para identificação de estados e transições de estado[68] de objetos.

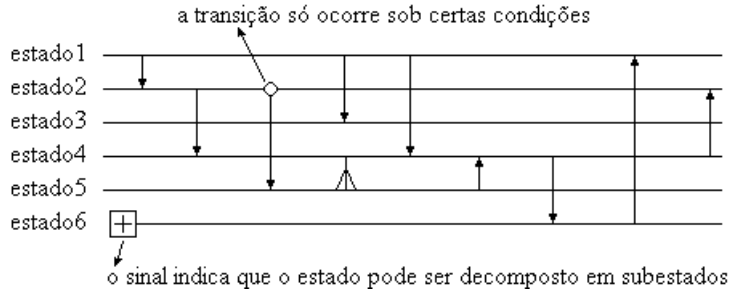


Figura 6.9 - exemplo de diagrama de transição de estado

6.2.7 Diagrama de mensagens

A figura abaixo apresenta um exemplo de diagrama de mensagens. Basicamente é composto de retângulos, que representam as classes e setas interligando classes - rotuladas pelas mensagens trocadas entre elas. É uma informação do comportamento das classes complementar ao esquema de objetos.

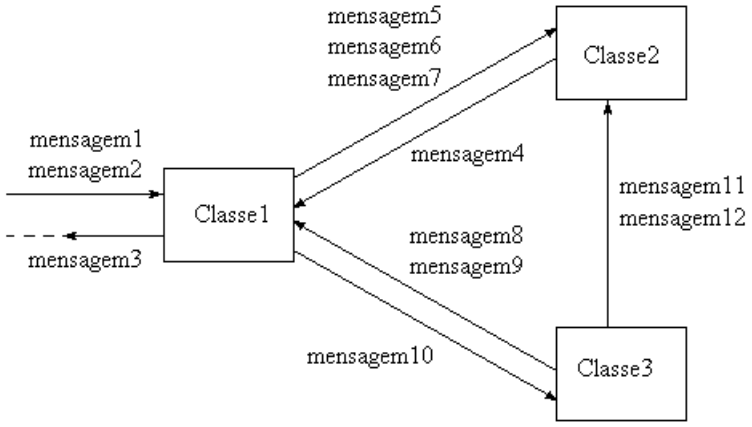


Figura 6.10 - exemplo de diagrama de mensagens

6.2.8 Diagrama de fluxo de objetos

O diagrama de fluxo do objeto, cujos elementos estão apresentados na figura abaixo, expressa a funcionalidade do sistema. Seus elementos são objetos (representados como caixas), atividades que consomem e produzem objetos (representadas por retângulos de cantos arredondados), agentes externos (representadas

68 E, conseqüentemente, de eventos.

por retângulos sombreados) e fluxos que transportam objetos (representados por setas). Este diagrama expressa basicamente a mesma semântica do DFD, ou seja, transformação de informações - neste caso de objetos, ao invés de dados. Os autores estabelecem que na construção do diagrama deve ser respeitado o princípio de uma atividade produzir apenas um objeto (de fato, objetos de uma mesma classe) - não havendo restrições para consumo de objetos.

O diagrama de fluxo de objetos admite estruturação - semelhante ao DFD - a partir do refinamento de atividades[69]. A informação contida no diagrama é uma descrição do sistema (gobal) e não das classes separadamente[70]. O uso do diagrama de fluxo de objetos para a modelagem de sistemas complexos na forma proposta pela metodologia de análise supõe o refinamento do diagrama de fluxo de objetos (das atividades) em diagramas de fluxo de objetos, até que se obtenham detalhes para a representação de uma atividade como um esquema de eventos[71].

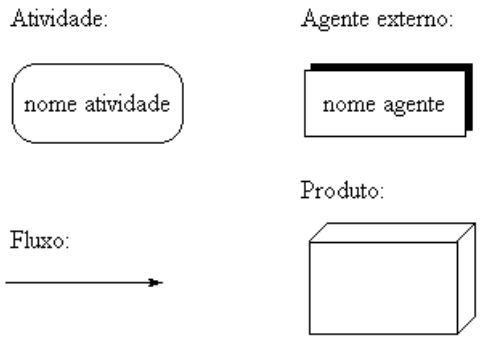


Figura 6.11 - elementos sintáticos do diagrama de fluxo de objetos

6.3 As etapas de construção de uma especificação na metodologia de Martin e Odell

A metodologia de Martin e Odell descreve os passos para a descrição de um sistema a nível de análise, mas não a nível de projeto. A seguir serão apresentados os passos propostos para a análise, segundo descrito em [MAR 95], acrescidos de comentários. Quanto a projeto, serão apresentados os objetivos a serem buscados, conforme proposto pelos autores, porém sem que isto se reflita num produto de projeto (uma descrição), uma vez que a metodologia não aborda este aspecto.

6.3.1 Passos da análise

A análise preconizada pelos autores busca primordialmente, a descrição dinâmica do sistema, a partir da construção dos esquemas de eventos. Em um sistema tido como "grande ou intricado demais para ser entendido usando-se técnicas

---

69 Atividades podem ser refinadas em diagramas de fluxo de objetos ou em esquema de eventos, semelhante ao que ocorre com as operações dos esquemas de eventos. Isto demonstra uma equivalência entre os conceitos de operação e atividade.

70 Da mesma forma que ocorre com o DFD usado em OMT.

71 Salienta, porém, que isto não tem necessariamente de ocorrer, pois, diagramas de fluxos de objetos e esquemas de eventos constituem maneiras diferentes de descrever processos. E a representação em uma forma não tem que se transformar obrigatoriamente, na outra.

comportamentais", é proposta uma descrição inicial do sistema através de um diagrama de fluxo de objetos e o refinamento sucessivo deste (atividades expandidas como novos diagramas de fluxo de objetos), até que as atividades dos diagramas mais refinados possam ser descritas a partir de esquemas de eventos[72].

A ênfase da análise do comportamento é a construção dos esquemas de eventos. As demais técnicas de modelagem voltadas ao aspecto dinâmico, são destinadas à obtenção de informações para os esquemas de eventos, e seu uso é opcional[73]. A construção do esquema de eventos se dá de forma top-down: eventos são identificados num nível de abstração elevado; estes eventos originam esquemas de eventos; os eventos presentes nos esquemas de eventos gerados dão origem a novos esquemas de eventos[74]; este ciclo de refinamento se encerra quando os diagramas de mais baixo nível não apresentarem eventos internos[75]. O resultado deste processo é a descrição do comportamento de um sistema a partir de um conjunto de esquemas de eventos. Cada esquema de eventos é construído de trás para frente: primeiro identifica-se o evento, depois a operação que o produz, as condições de controle para estas operações, as regras de gatilho e os eventos que as disparam.

A obtenção do esquema de objetos (ou opcionalmente, da descrição de herança, agregação e associações entre classes, em diagramas distintos, conforme apresentado) se dá a partir de uma seqüência de passos procedidos paralelamente aos passos da descrição dinâmica[76].

Obtenção da descrição dinâmica (esquema de eventos):

Passo 0: definir o foco da análise
Delimitar o domínio da aplicação que será objeto da análise.
Identificar o tipo de evento meta: identificar os eventos (tipos de evento) que o sistema irá produzir (eventos meta), bem como os eventos a que será submetito (eventos iniciais). Isto define as fronteiras do sistema em termos de eventos. Os passos seguintes correspondem a um procedimento cíclico que busca descrever o que deve haver entre os tipos de eventos identificados[77].

---

72 Quando uma atividade deve ou não ser descrita através de um esquema de eventos, depende basicamente do "*feeling*" de quem procede a análise. A metodologia não trata deste aspecto.

73 A metodologia não explicita que uma descrição de sistema deva necessariamente possuir diagramas de fluxo de objetos, diagramas fence, ou diagrama de mensagens.

74 Um esquema de eventos gerado a partir de um evento de um esquema de nível superior, é obtido de fato, pelo refinamento da operação correspondente. A operação é que origina um novo esquema de eventos, e não o evento. O evento originador aparecerá como "saída" do diagrama refinado.

75 Os esquemas de eventos neste nível apresentarão eventos de entrada externos ao escopo do diagrama, que disparam operações, e eventos de saída, resultantes destas operações - sem eventos "internos", entre operações.

76 Cabe novamente salientar, que a metodologia propõe que a obtenção do modelo estrutural seja conseqüência da construção do modelo comportamental (esquema de eventos).

77 O primeiro esquema de eventos produzido, segundo propõe a metodologia, bem como os diagramas resultantes do refinamento progressivo deste diagrama, correspondem a uma descrição do comportamento do sistema como um todo, e não de seus objetos separadamente.

Passo 1: tornar claro o evento
Identificar o tipo de evento básico: classificar o tipo de evento (criação de objeto, classificação de objeto etc.), buscar descrições de estado de objeto que antecede e sucede o evento (o que auxilia a descrição estrutural).
Nomear o tipo de evento: o nome do evento deve refletir o objeto envolvido, a passagem do pré-estado para o pós-estado e indicar a operação envolvida.

Passo 2: generalizar o tipo de evento
Generalizar o tipo de evento e escolher o nível de abstração adequado: situar o tipo de evento identificado em uma estrutura hierárquica de subtipos e supertipos e escolher o nível adequado à descrição em curso.
Integrar o tipo de evento: após a generalização do evento, verificar se ele já foi especificado (algum dos níveis da estrutura hierárquica, em outra parte da especificação), para evitar duplicação de esforço.

Passo 3: Definir condições de operação
Identificar a operação: identificar a operação exigida para produzir o evento especificado no passo 2
Determinar se a operação é interna ou externa: operações externas não exigem refinamento, pois, extrapolam o escopo sob descrição - pelo menos em relação ao esquema de objetos sob descrição.
Identificar as condições de controle: identificar as condições necessárias ao início da operação.
Normalizar as condições de controle: expressar o conjunto de condições em uma expressão booleana (por exemplo, SE condição 1 OU condição 2, ENTÃO ...).

Passo 4: Identificar causas da operação
Identificar os tipos de eventos acionadores: identificar que eventos devem ocorrer para fazer com que uma condição de controle seja avaliada como verdadeira.
Identificar condições de controle complexas.
Normalizar os tipos de eventos acionadores: produzir o agrupamento das regras de gatiho em um número adequado de losângulos de modo a expressar adequadamente as operações "E" e "OU" sobre os eventos acionadores da operação.
Especificar as regras de gatilho: especificar as funções associadas às regras de gatilho, responsáveis por obter os argumentos a serem enviados à operação.

Passo 5: refinar os resultados do ciclo
Generalizar o tipo de evento gatilho: buscar uma descrição mais geral para para os eventos iniciais.
Especializar o tipo de evento meta: proceder um detalhamento dos eventos meta (buscar subtipos).
Verificar eventos duplicados.

Obtenção da descrição estrutural:

Passo 1: A busca de descrições de estado de objeto no passo da descrição dinâmica leva à identificação de classes que devem estar presentes no esquema de objetos.

Passo 2: A estrutura hierárquica de tipos de evento obtida leva à identificação de novos estados de objeto, enriquecendo o modelo estrutural.

Passo 3: Os termos presentes nas condições de controle exigem classes correspondentes no esquema de objetos.

Passo 4: As funções associadas às regras de gatilho dos diagramas de eventos retratam ligações entre classes, o que se reflete nas asssociações entre classes, do esquema de objetos.

Passo 5: O refinamento do esquema de eventos também corresponde a um refinamento do esquema de objetos: rearranjo das estruturas de herança e agregação, bem como das associações.

6.3.2 Diretrizes para o projeto

No item "visão geral da metodologia" foram apresentadas em linhas gerais as informações buscadas na etapa de projeto. Além deste dado, os autores apenas apresentam uma corresponência entre os elementos dos modelos usados e estruturas das linguagens de programação orientadas a objetos. Nenhum modelo adicional é proposto para registrar este mapeamento. Assim, ou uma forma de expressá-lo é adotada, ou a passagem dos modelos para o programa se dá diretamente na etapa de implementação - neste caso o projeto corresponderia apenas ao refinamento dos modelos da análise. Os autores não entram no mérito desta discussão.

Será apresentada a seguir, a correspondência entre os elementos dos modelos usados e estruturas das linguagens de programação orientadas a objetos, segundo proposto. Nesta descrição serão apresentados os elementos da análise e descrita a sua correspondência em projeto - que equivale à correspondência com a implementação[78].

Correspondência entre os elementos do modelo estrutural (esquema de objetos) e estruturas das linguagens de programação orientadas a objetos

Classes

As classes da análise serão classes de projeto, porém nem todas as classes definidas na análise darão origem a classes, no projeto[79].

Associações

Associações refletem a necessidade de um objeto manter informação sobre outro. Isto é traduzido como atributo. A estrutura de dados definida para os atributos deve comportar as restrições de cardinalidade expressas na associação. Nem sempre os dois lados da associação precisam manter informação sobre o lado oposto (associações unidirecionais, neste caso)

---

78 Se for admitido que o projeto refina os modelos da análise, a etapa de projeto corresponderia a duas subetapas: refinamento de modelos e mapeamento de estruturas. O produto da última subetapa reflete a implementação.

79 Algumas se refletirão em atributos, como será apresentado na correspondência das associações.

Atributos que traduzem associações podem inclusive incorporar uma das classes associadas - fazendo com que uma classe da análise não tenha uma classe correspondente no projeto. A figura abaixo ilustra esta situação: na análise foram identificadas as classes Carro e Ano e a associação fabricação entre elas, que descreve o ano de fabricação de um carro - *carro* fabricado em *ano*. A nível de projeto pode existir a classe Carro com o atributo anoFabricacao, que traduz a mesma realidade, sem a necessidade de se implementar uma classe Ano.

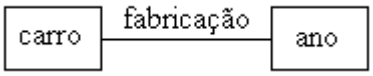


Figura 6.12 - associação onde uma das classes se transforma em atributo da outra

Estruturas de generalização

Herança é uma característica das linguagens de programação orientadas a objetos - a correspondência é direta. Para linguagens sem o mecanismo de herança múltipla, como Smalltalk, um caminho alternativo deve ser buscado[80].

Classificação múltipla e dinâmica

A classificação múltipla e dinâmica é um mecanismo conceitual da metodologia que não encontra correspondência direta nas linguagens de programação orientadas a objetos, como já citado. Para isto os autores propõem uma solução pouco ortodoxa, denominada fatiamento de objetos (object slicing). O fatiamento de objetos consiste em fazer com que um objeto da análise (uma instância) corresponda a mais de um objeto em tempo de execução - um para cada classificação do objeto. Desclassificar o objeto de uma classe corresponde a eliminar uma das instâncias, mantendo as demais.

Correspondência entre os elementos do modelo dinâmico (esquema de eventos) e estruturas das linguagens de programação orientadas a objetos

Operações

As operações dos esquemas de eventos serão traduzidas em métodos de classe.

Regras de gatilho

As regras de gatilho, por não encontrarem correspondencia direta em linguagens, são implementadas em partes, segundo suas finalidades:

- Invocar operação: métodos (operações) são invocados por meio de mensagens;
- Avaliar a condição de controle: corresponde a um método que retorna um valor verdadeiro ou falso;
- Preencher argumentos da operação: um ou mais métodos devem ser invocados para a obtenção dos argumentos;
- Detectar eventos: "usualmente é implementada pesquisando-se ativamente o ambiente do sistema para detectar que algum evento predefinido ocorra".

80 Como a delegação [RUM 94].

Esquemas de eventos e programador de eventos

Segundo Martin e Odell, a vantagem dos esquemas de eventos é a sua potencialidade de serem completos e executáveis, capazes de proporcionar uma descrição completa do código da aplicação. Os esquemas de eventos são melhor implementados com um código escrito de maneira voltada aos eventos, como é usado em Graphic User Interfaces, GUI. Nesta abordagem, a aplicação é controlada por um programador de eventos. O programador de eventos permite a seleção de um evento de uma fila de eventos e executa o processamento adequado antes de passar ao evento seguinte.

6.4 Exemplo de uso da metodologia de Martin e Odell

Análise

Aplicação cíclica dos passos 1 a 5, para a construção do esquema de eventos.

- Passo 1: o evento básico produzido pelo sistema é o estado corrente de uma partida;
- Passo 2: este evento básico pode ser refinado em dois tipos de evento: partida em estado intermediário e partida em estado final;
- Passo 3: a operação que produz este evento é "proceder lance", e a condição para que a operação ocorra é que a partida esteja em um estado diferente do estado final;
- Passo 4: dois eventos devem ocorrer para que a operação inicie, que são estado de partida (inicial ou intermediário) e usuário ter autorizado o lance;
- Passo 5: refinar o resultado dos passos anteriores, se necessário.

A figura abaixo traduz estas informações na sintaxe do esquema de eventos.

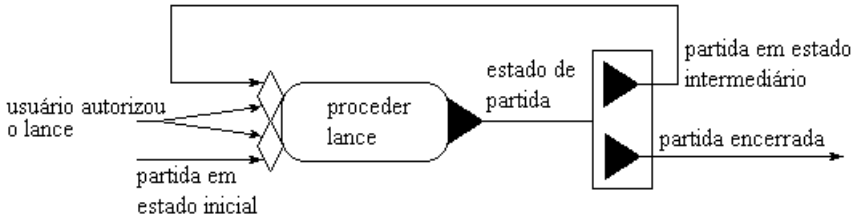


Figura 6.13 - resultado do primeiro ciclo de descrição de eventos

A aplicação dos passos acima aos eventos causadores da operação "proceder lance", de forma recursiva, até chegar apenas a eventos produzidos por operações externas, completará o esquema de eventos. A figura abaixo apresenta o esquema de eventos em um nível de abstração que admite refinamento.

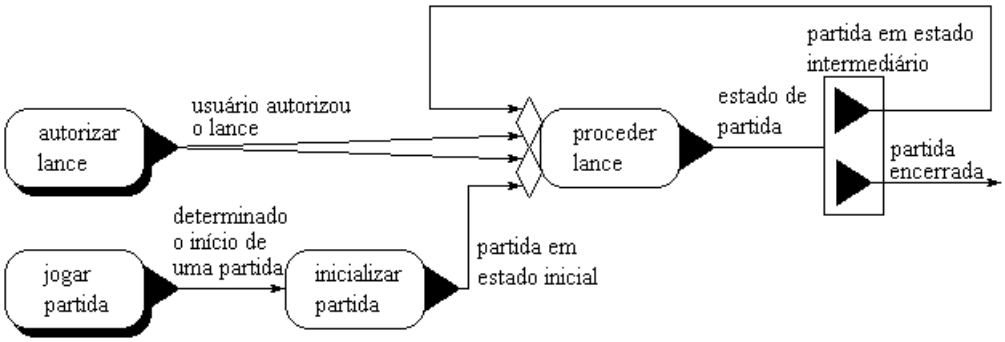


Figura 6.14 - esquema de eventos do jogo Corrida

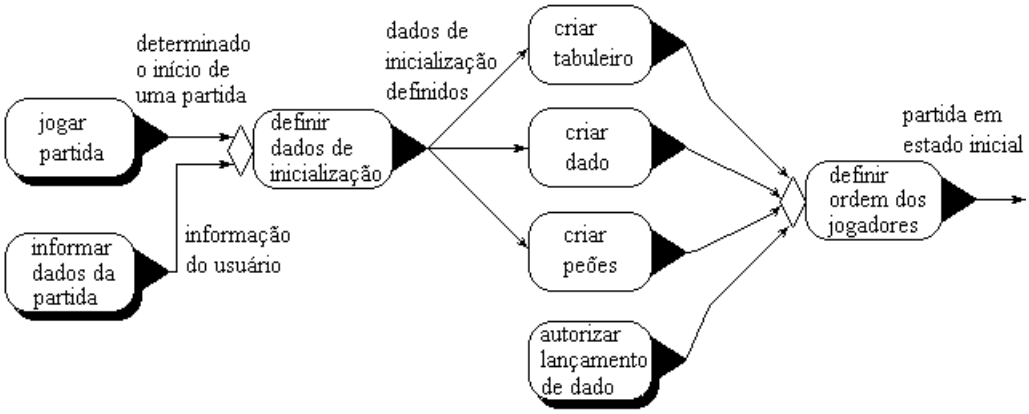


Figura 6.15 - esquema de eventos resultante do refinamento da operação "inicializar partida"

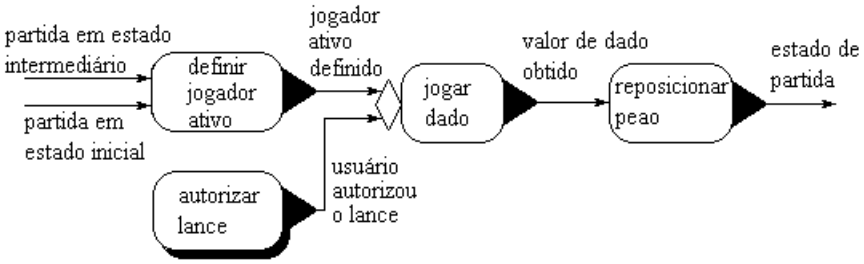


Figura 6.16 - esquema de eventos resultante do refinamento da operação "proceder lance"

Cada operação interna dos dois últimos esquemas de eventos (resultantes de refinamentos) devem ser refinadas em novos esquemas de eventos até que nenhum diagrama de mais baixo nível contenha eventos internos. Isto não será procedido neste exemplo, para não torná-lo excessivamente longo.

Os eventos identificados levam à identificação das classes envolvidas. A figura abaixo apresenta o esquema de objetos obtido.

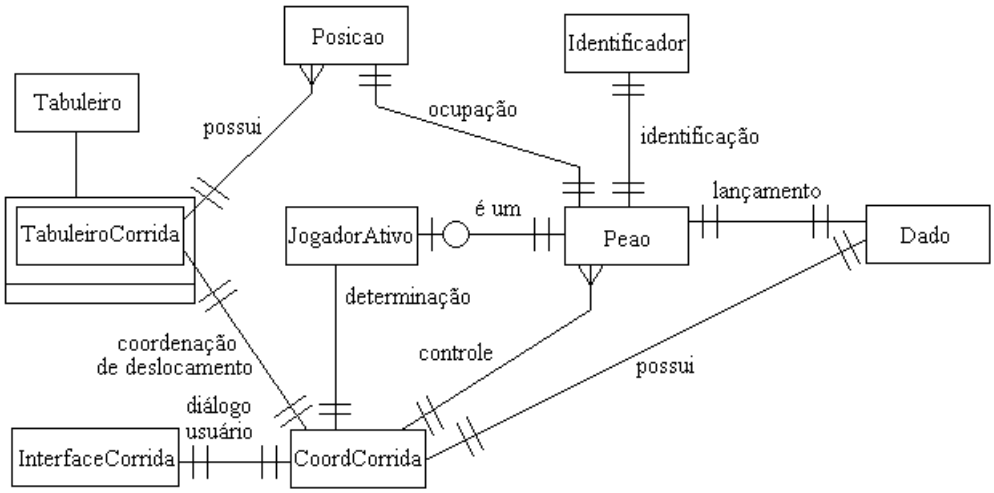


Figura 6.17 - esquema de objetos do jogo Corrida

Como a metodologia não estabelece passos ou modelos para a etapa de projeto, esta não será tratada. Ressalta-se porém, que mesmo um um refinamento do esquema de eventos que leve às operações mais elementares, ainda deixará para o projeto a tarefa de definir os atributos das classes, bem como a alocação de métodos às classes.

7 Metodologia Fusion

7.1 Visão geral da metodologia

A metodologia Fusion [COL 94] é uma metodologia orientada a objetos, cuja abordagem é classificada como dirigida a dado. O desenvolvimento de uma aplicação passa pelas etapas de análise, projeto e implementação. Na análise é construído o modelo de objetos e o modelo de interface. O modelo de objetos de Fusion é muito semelhante ao modelo de objetos de OMT, ou seja, com uma semântica muito próxima aos diagramas ER. O modelo de interface contém a modelagem dinâmica desenvolvida na etapa de análise. Durante a análise não existe a noção de método de classe, mas de operações do sistema. São identificadas as possíveis interações do sistema a desenvolver com seu meio externo - as "operações" - de uma forma semelhante à construção do modelo de requisitos de OOSE. Após, as operações são detalhadas. Na descrição das operações durante a análise, o sistema é sempre visto como um todo, e nunca como um conjunto de objetos interagindo.

Durante o projeto é produzida uma descrição do conjunto de classes, visando a implementação - são definidos novos atributos e o conjunto de métodos. O projeto utiliza modelos diferentes dos modelos da análise. A especificação é composta pelos modelos da análise e pelos modelos de projeto.

A implementação consiste na tradução da especificação produzida, em linguagem de programação. Fusion supõe o uso de uma linguagem de programação orientada a objetos. A implementação não será tratada no presente trabalho.

7.1.1 Análise e projeto

A metodologia Fusion não possui uma etapa de análise de requisitos e estabelece o início do processo de desenvolvimento da especificação, a partir de uma descrição textual do sistema a desenvolver. A análise inicia com a construção do modelo de objetos. A partir do conhecimento do domínio são identificadas as classes, e é construída uma estrutura de classes. Fusion divide os atributos de uma classe em dois grupos: os atributos dado, que armazenam informações inerentes à classe que os possui, e os atributos objeto, que armazenam a referência a outros objetos. Durante a construção do modelo de objetos, apenas os atributos dado são incluídos na descrição. A tarefa seguinte da análise é a identificação e descrição das operações do sistema, ou seja, as possíveis interações do sistema com o meio externo. Esta descrição é o conteúdo do modelo de interface, que descreve a dinâmica do sistema como um todo - diferente de OMT, por exemplo, que descreve a dinâmica do sistema sob a ótica de objetos que interagem. O modelo de interface se constitiui dos modelos de operação e de ciclo de vida. O modelo de operação descreve cada operação identificada na forma de texto estruturado. O modelo de ciclo de vida estabelece as possibilidades de ordenamento de operações, durante uma execução.

A primeira atividade da etapa de projeto é a construção dos grafos de interação de objetos para as operações. Cada operação descrita através de um modelo de operação, origina pelo menos um grafo de interação de objetos, que descreve a interação de objetos necessária para a execução da operação. Esta descrição identifica os métodos das classes, de forma semelhante à identificação de OMT a partir das descrições de

cenários. A atividade seguinte é a construção dos grafos de visibilidade, que concentram o conjunto de referências a objetos que cada classe deve manter, o que define os atributos objeto das classes. A terceira atividade é a construção das descrições de classe, que em um formato semelhante a um header de C++, descreve cada classe: nome da classe, superclasses (caso haja), atributos e assinatura dos métodos[81]. O último modelo gerado na etapa de projeto é o conjunto de grafos de herança, que expressam a estrutura hierárquica de herança das classes do sistema descrito.

O processo de modelagem é iterativo. Aspectos definidos em um modelo podem alterar os anteriormente elaborados. Um dicionário de dados é construído ao longo das etapas de análise e projeto, descrevendo de forma textual, classes, atributos, métodos e conceitos presentes na modelagem, como predicados, eventos, agentes, etc.

7.1.2 Técnicas de modelagem para a descrição de uma aplicação

O modelo de objetos adotado pela metodologia Fusion é muito semelhante ao modelo de objetos de OMT, que é baseado no modelo de entidade-relacionamento (ER) [CHE 76]. Seus principais elementos são as classes, com sua organização estrutural - representadas como retângulos contendo o nome da classe e seus atributos - e os relacionamentos entre as classes - representados como traços interligando classes. Como em OMT, cada um dos dois elementos do técnica de modelagem apresenta uma correspondência com os elementos do diagrama ER. O modelo de objetos, descreve a estrutura estática do sistema e é o primeiro dos modelos a ser construído.

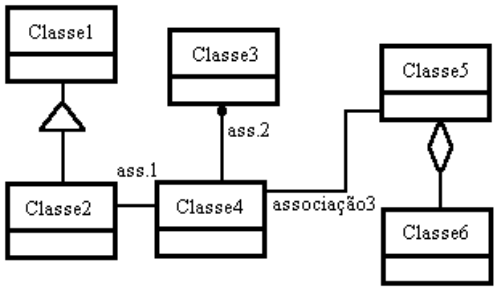


Figura 7.1 - exemplo de modelo de objetos

O modelo de interface - o outro técnica de modelagem usado na etapa de análise, além do modelo de objetos - visa descrever como o sistema sob modelagem, interage com o meio a sua volta. Utiliza o conceito de operação para descrever estas interações. Segundo a metodologia, uma operação do sistema se constitui de um evento de entrada (originado por um agente externo) e do efeito que ele pode ter. A descrição de uma operação envolve o sistema, agentes, e eventos trocados entre eles. O modelo de interface é a composição de dois modelos: o modelo de operação e o modelo de ciclo de vida. O modelo de operação é uma descrição textual em formato estruturado. Cada operação é descrita nos seguintes termos: nome da operação, descrição textual do que consiste a operação, relação dos dados utilizados (destacando os que são e os que não são alterados), relação dos possíveis eventos de saída originados pela operação (e dos agentes a que são destinados), definição da precondição para que a operação ocorra

---

81 A única descrição prevista pela metodologia para o corpo dos métodos, é descrição textual, no dicionário de dados.

(predicado) e o estado do sistema resultante da ocorrência da operação (predicado contendo a lista de resultados possíveis: atributos alterados, objetos criados ou destruídos e eventos de saída gerados). O modelo de operação não é capaz de fornecer uma visão temporal da seqüência de passos para a execução da operação.

Operação:	inicialização de partida
Descrição:	obtém informações do usuário; procede definição da ordem dos jogadores e apresenta partida no estado inicial
Lê:	
Muda:	número de casas do tabuleiro, número de jogadores, ordem dos jogadores
Envia:	valores de lançamento de dado, estado da partida, ordem dos jogadores
Assume:	
Resulta:	partida no estado inicial: a posição de todos os peões é a casa zero

Figura 7.2 - exemplo de esquema do modelo de operações

O modelo de ciclo de vida apresenta uma sintaxe parecida com a de um diagrama BNF, e descreve as seqüências de operações possíveis para o sistema, desde o início até o encerramento de uma execução, ou seja, descreve o ciclo de vida de uma execução do sistema - também porta a informação de temporalidade das etapas de uma operação, ausente no modelo de operação. Das quatro metodologias anteriormente descritas, apenas OMT e OOSE trabalham com a noção de descrição de operações (com cenários, o que é uma abordagem diferente da adotada por Fusion, por tratar a operação como um conjunto de interações entre objetos[82]). E nenhuma das duas utiliza uma técnica de modelagem específica para descrever as seqüências realizáveis de operações - esta seqüência geralmente pode ser observada no diagrama de estados do objeto responsável pelo controle da operação do sistema. Assim, este técnica de modelagem usada por Fusion apresenta uma nova modalidade de descrição.

ciclo_de_vida JogoCorrida = inicialização_de_partida . (procedimento_de_lance)$^{+}$

inicialização_de_partida = ...

Figura 7.3 - exemplo de modelo de ciclo de vida

Durante a análise a ênfase é a identificação e detalhamento das operações do sistema, sob a ótica do sistema como um todo. Na etapa de projeto estas informações são estendidas para o conjunto de classes, originando os métodos, os atributos objeto e novos atributos dado. O grafo de interação de objetos é o primeiro modelo gerado na etapa de projeto. Este diagrama descreve a interação entre objetos para a realização de uma operação do sistema. Cada operação identificada origina um grafo de interação de

82 OOSE possui o modelo de requisitos, que descreve as operações (*use cases*, na nomenclatura de OOSE) como a interação do sistema com o meio externo - sem envolver os objetos que constituem o sistema. Este modelo, porém, é praticamente todo textual e serve como fonte de informação para a construção dos cenários, onde ocorre a descrição da operação de modo semiformal. Está-se considerando portanto, que o conjunto de cenários é que constitui a descrição das operações na especificação do sistema - o modelo de requisitos assume menor importância.

objetos. Como o grafo de interação de objetos pode ser estruturado, uma operação complexa pode originar um conjunto de grafos. Objetos (instâncias) são representados como retângulos e passagens de mensagem, como setas apontando para o objeto servidor. Uma operação de sistema pode envolver um número qualquer de passagens de mensagem - conseqüentemente, de execução de métodos de classe de objetos servidores, por solicitação de clientes. Um dos objetos envolvidos na interação é escolhido para assumir o papel de coordenador da operação (decisão de projeto). Cabe ao coordenador iniciar a operação: o método de classe que inicia a operação é um método do coordenador, ativado por ação de um agente. A partir da execução deste método, as demais interações ocorrem. Os demais objetos envolvidos na operação são chamados colaboradores.

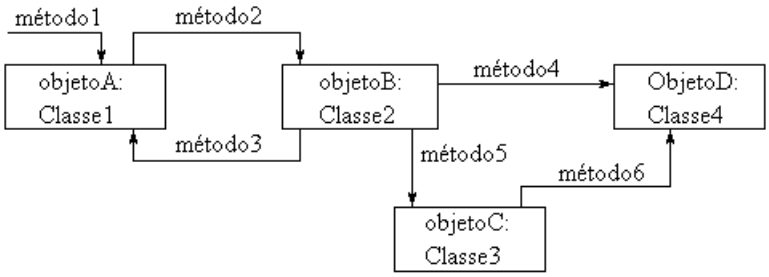


Figura 7.4 - exemplo de grafo de interação de objetos

Grafos de visibilidade concentram a informação das referências a outros objetos que uma classe mantém - e que define os atributos objeto. Esta informação é obtida dos grafos de interação de objetos. Os grafos de visibilidade organizam a informação para cada classe do sistema: para cada classe é construído um grafo de visibilidade. A notação envolve retângulos representando as classes, e setas que saem da classe modelada e apontam para as classes cuja referência ela mantém. A sintaxe do grafo de visibilidade diferencia a referência mantida segundo os seguintes aspectos: duração de referência (permanente ou dinâmica[83]), visibilidade do servidor (exclusiva a um cliente ou não), ligação do servidor ao cliente (se a destruição do objeto cliente implica na destruição do servidor, ou não) e mutabilidade da referência (se uma vez definida a referência a um objeto, pode ser alterada em tempo de execução).

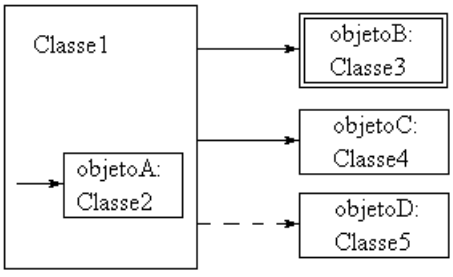


Figura 7.5 - exemplo de grafo de visibilidade

83 Visibilidade permanente implica em manter um atributo que referencia um objeto e visibilidade dinâmica corresponde possuir a visibilidade de um objeto apenas durante a execução de um método. Por exemplo, no sistema modelado como ilustração das metodologias de desenvolvimento de software, a classe Peao não possui um atributo que referencia uma instância de Dado, mas recebe a referência em uma chamada de método. Isto exemplifica visibilidade dinâmica.

É gerada uma descrição de classe para cada classe do sistema. Como o técnica de modelagem anterior, organiza por classes, uma informação disponível no restante da especificação. A descrição de cada classe apresenta um formato semelhante a um header de C++ e contém: nome da classe, superclasses (caso haja), atributos e assinatura dos métodos. A busca de informações para a composição das descrições ocorre da seguinte forma: atributos dado são buscados no modelo de objetos, atributos objeto (referências) são buscados nos grafos de visibilidade, métodos de classe são buscados nos grafos de interação de objetos, superclasses são buscadas nos grafos de herança.

Os grafos de herança descrevem estrutura de herança das classes do sistema descrito. A sintaxe destes grafos é exatamente a sintaxe de herança do modelo de objetos. Duas particularidades da metodologia justificam a existência deste técnica de modelagem, em adição à representação de herança que é feita no modelo de objetos. A primeira é referente à notação do modelo de objetos: Fusion admite que vários diagramas componham o modelo de objetos, inclusive com repetição de classes. Este formato é o mesmo de OOSE, e diferente das demais metodologias. Com isto, as classes relacionadas por herança podem estar representadas em diferentes diagramas do modelo de objetos, não ressaltando este relacionamento. A segunda particularidede de Fusion que justifica o grafo de herança, é que este é um técnica de modelagem que descreve herança na fase de projeto, enquanto o modelo de objetos é um técnica de modelagem da etapa de análise. Fusion admite que relações de herança sejam definidas na etapa de projeto (como decisão de projeto) e não tenham correspondência com a modelagem da etapa de análise. A busca de relações de herança se dá: no modelo de objetos, nos grafos de interação de objetos e descrições de classe (à procura de funcionalidade comum) e nos grafos de visibilidade (à procura de estruturas de referências comuns).

7.2 Elementos sintáticos das técnicas de modelagem da metodologia Fusion

7.2.1 Modelo de objetos

A figura 7.6 reúne o conjunto de elementos sintáticos da metodologia Fusion, para a construção do modelo de objetos.

A semântica dos elementos é a mesma definida para OMT, apesar das diferenças sintáticas[84]. Fusion também apresenta associações ternárias e herança múltipla.

Para efeito de estruturação, Fusion admite que o diagrama do modelo de objetos seja particionado em subdiagramas. O modelo de objetos completo é a união dos subdiagramas. Se dois símbolos de classe com o mesmo nome aparecem em diferentes subdiagramas, trata-se da mesma classe. Os atributos da classe resultam da união de todos os atributos da classe nos vários subdiagramas. As associações são tratadas de forma similar.

---

84 OMT possui mais elementos sintáticos para a construção do modelo de objetos.

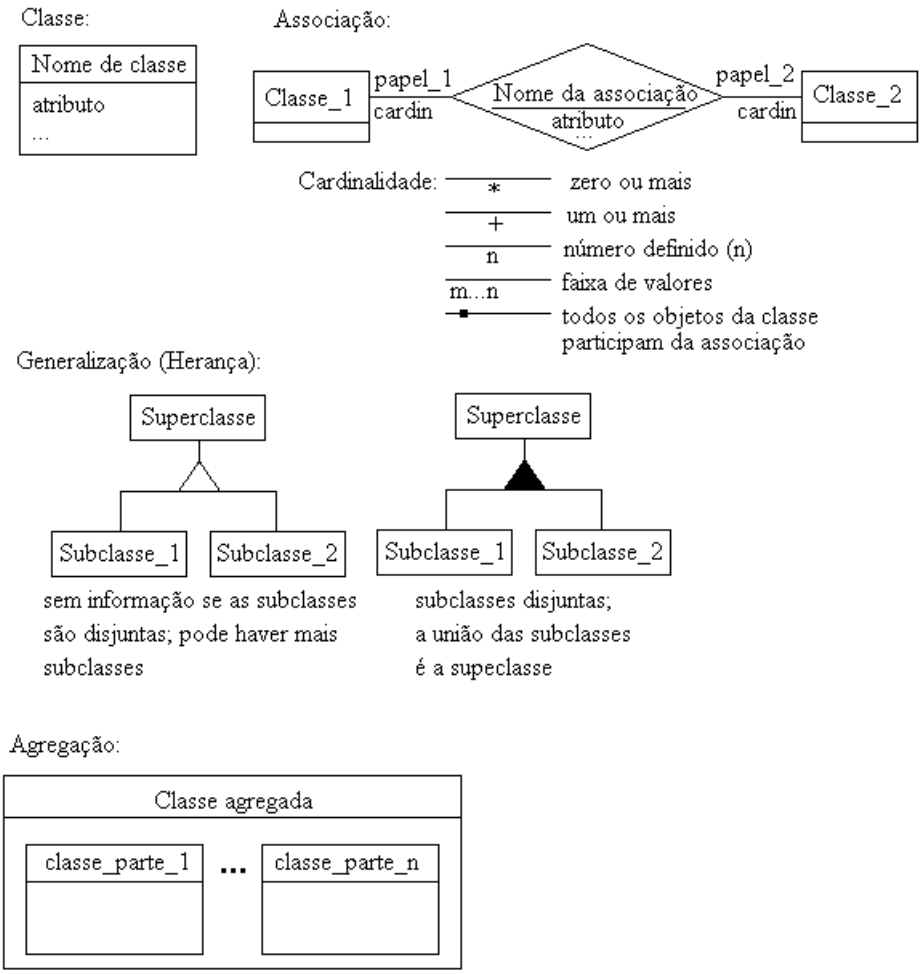


Figura 7.6 - elementos sintáticos do modelo de objetos

7.2.2 Modelo de interface

Modelo de operação

O modelo de operação consiste em um esquema de texto estruturado, como apresentado na figura 7.7, que lista o conjunto de características que a metodologia estabelece como necessárias e suficientes para a descrição completa de uma operação de sistema[85].

No item Lê, a palavra chave *suprido* precedendo um item, indica que o item é um parâmetro da operação. No item Muda, a palavra chave *novo* precedendo um identificador de objeto, indica que a operação cria uma nova instância.

85 Observe-se que não há uma conotação de evolução temporal, em termos do desenvolvimento da operação.

Operação:	nome da operação
Descrição:	descrição textual, informal e concisa
Lê:	lista de todos os valores que a operação pode acessar, sem alterá-los
Muda:	lista de todos os valores que a operação pode acessar, podendo alterá-los
Envia:	lista de todos os agentes a que a operação pode enviar eventos e os eventos respectivos
Assume:	predicado, que contém uma lista de cláusulas que podem assumir a condição true ou false, e que define a condição necessária para que a operação se realize (precondição)
Resulta:	predicado que define a condição do sistema (estado) após a ocorrência da operação

Figura 7.7 - formato do modelo de operação

Modelo de ciclo de vida

O ciclo de vida do sistema (em uma execução) é descrito por uma sentença:

***ciclo de vida* sistema = expressão ciclo de vida**

onde "expressão ciclo de vida" é definida segundo a linguagem abaixo apresentada:

alfabeto:	qualquer evento de entrada ou saída (evento de saída é prefixado por #).
operadores:	seja x e y expressões ciclo de vida, então:
	x.y denota x é seguido de y;
	x|y denota a ocorrência de x ou y;
	x* denota zero ou mais ocorrências de x;
	x^+ denota uma ou mais ocorrências de x;
	[x] denota que x é opcional;
	x||y significa possibilidade de concorrência de elementos de x e y.
substituições:	uma expressão pode ser nomeada em uma substituição:
	nome = expressão ciclo de vida
	nome pode ser usado em outras expressões, mas substituições não podem ser recursivas.
precedência de operadores:	em ordem decrescente, a precedência é:
	[ ] , * , $^+$, . , | , ||
	expressões podem ser postas entre parênteses para alterar a precedência default.

Figura 7.8 - linguagem do modelo de ciclo de vida

7.2.3 Grafo de interação de objetos

A figura abaixo reúne o conjunto de elementos sintáticos da metodologia Fusion, para a construção dos grafos de interação de objetos.

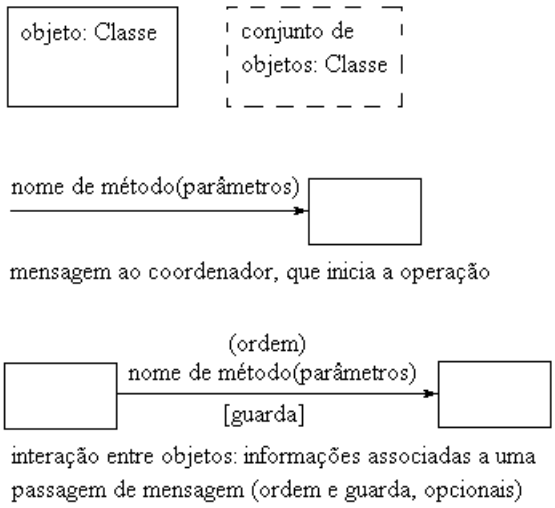


Figura7.9. - elementos sintáticos do grafo de interação de objetos

7.2.4 Grafo de visibilidade

A figura abaixo reúne o conjunto de elementos sintáticos da metodologia Fusion, para a construção dos grafos de visibilidade.

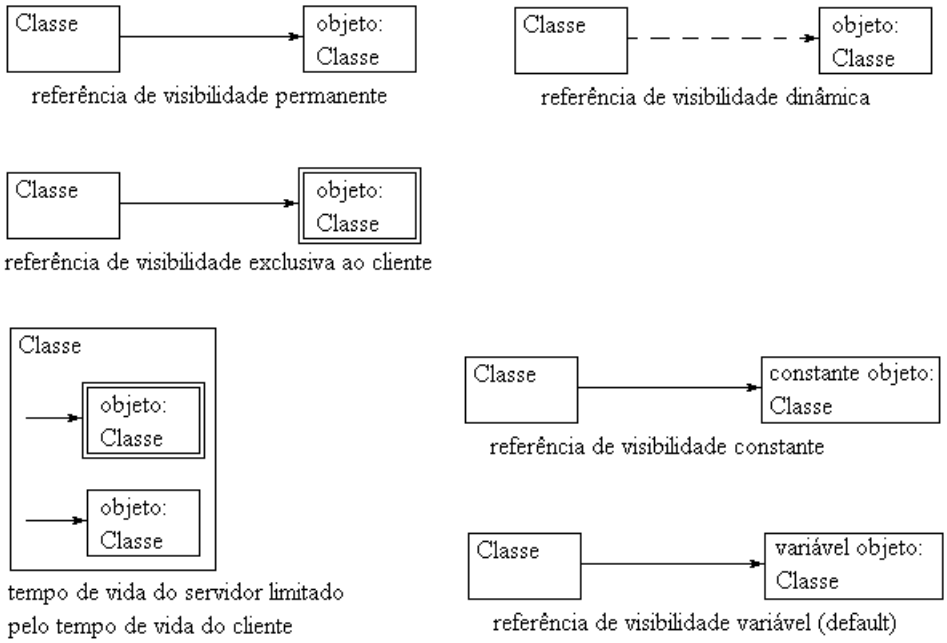


Figura 7.10 - elementos sintáticos do grafo de visibilidade

7.2.5 Descrição de classe

A descrição de classe utiliza o padrão de descrição apresentado abaixo. Expressões entre os sinais "[" e "]" são opcionais e expressões entre os sinais "<" e ">" são preenchidas com informações das classes.

```
Classe  <nome de classe>  [é  <nomes de superclasses>  ]

        // para cada atributo:
        [atributo] [mutabilidade] <nome de atributo> : [partilhamento] [limitação] <tipo>
                        .
                        .
                        .
        // para cada método:
        [método]  <nome de método> ( <lista de argumentos> )  [ : <tipo de retorno> ]
                        .
                        .
                        .
fimClasse
```

Figura 7.11 - formato da descrição de classe

Onde:

- mutabilidade: *constante* ou *variável* (variável quando não indicado);
- partilhamento: *partilhado* por mais de um objeto ou *exclusivo* (partilhado quando não indicado);
- limitação: tempo de vida do atributo (objeto referenciado) *limitado* ao tempo de vida do objeto, ou *não-limitado* (não-limitado quando não indicado).

7.2.6 Grafo de herança

Os grafos de herança usam os mesmos símbolos do modelo de objetos para a representação de classes e de herança.

7.3 As etapas de construção de uma especificação na metodologia Fusion

Conforme já descrito, a análise se preocupa com a identificação e organização das classes, atributos dado e com a identificação e descrição das operações do sistema. Não são tratados durante a análise os métodos de classe e nem atributos objeto (referências). Os instrumentos de modelagem da análise são o modelo de objetos, que contém as classes com sua estrutura e atributos dado, e o modelo de interface, que descreve as operações do sistema. O modelo de interface é composto pelos modelos ciclo de vida, que estabelece as seqüências de operações possíveis para o sistema, e o de operação, que descreve cada operação do sistema sob a ótica da interação do sistema como um todo com seu meio externo.

No projeto as informações sobre operações são transportadas para as classes, originando atributos objeto e métodos. O projeto produz uma descrição das classes do sistema sob descrição, na forma de uma relação de seu conjunto de atributos (dado e objeto) e assinatura de métodos. Esta informação fica concentrada no modelo de descrição de classe, que é o último modelo a ser construído na etapa de projeto. Outros três modelos são usados no projeto para o transporte das informações das operações do sistema para as classes: os grafos de interação de objetos, que descrevem as operações na forma de interação entre objetos, os grafos de visibilidade, que descrevem as referências mantidas pelas classes e os grafos de herança, que descrevem a estrutura hierárquica de herança.

A metodologia Fusion utiliza um dicionário de dados que concentra em forma textual, detalhes não contidos nos demais modelos. O dicionário de dados é construído ao longo das etapas de análise e projeto. Fusion recomenda um dicionário em forma tabular, contendo três colunas: nome da entrada (informação), tipo (atributo, classe, método, predicado, operação, agente, evento, etc.) e descrição textual.

7.3.1 Passos da análise

Desenvolver um modelo de objetos para o domínio do problema

- identificar classes: buscar entre candidatos como objetos físicos, pessoas, organizações e abstrações (por exemplo, discrepâncias, lista de alocação);
- identificar associações (relacionamentos): buscar entre candidatos como comunicações (elementos que interagem), conteúdo (elementos que contém elementos), ligações (parentesco, vizinhança etc.) e ações (por exemplo, lançamento de dados no jogo Corrida);
- identificar herança;
- identificar agregação (Fusion, além de usar agregação com a semântica dos demais métodos, admite que agregação possa ser usada para estruturar um modelo de objetos complexo: um objeto de um modelo de objetos pode ser desdobrado em um conjunto de objetos);
- identificar atributos dado;
- definir cardinalidades;
- identificar invariantes: restrições, imposições inerentes ao sistema, são identificadas e colocadas no dicionário de dados (por exemplo, a ordem dos jogadores em uma partida deve ser a ordem decrescente, do valor obtido por cada jogador, em um lançamento de dado).

Determinar a interface do sistema

- identificar agentes, operações do sistema e eventos:
  - identificar o que é interno e o que é externo ao sistema sob descrição;
  - identificar operações do sistema: as interações entre sistema e agentes;
  - construir cenários para as operações[86], para identificar os eventos trocados entre o sistema e os agentes.

---

86 A sintaxe dos cenários é a mesma de OMT, porém em Fusion os cenários são usados para modelar a interação entre sistema e agentes, ao invés de modelar a interação de objetos, como é feito em OMT: o sistema é representado por um traço vertical, assim como cada agente, e os eventos trocados entre sistema e agentes, são representados por setas.

- produzir o modelo de objetos do sistema, através do acrésimo de um contorno do sistema no modelo de objetos do domínio: a partir da determinação da interface do sistema, é elementar identificar que classes do modelo de objetos do domínio fazem parte do sistema e que classes correspondem a agentes externos.

Desenvolver um modelo de interface

- desenvolver um modelo ciclo de vida:
  - generalizar os cenários para formar expressões ciclo de vida nomeadas: expressões ciclo de vida podem expressar o conteúdo de mais de um cenário, já que os cenários não são capazes de expressar repetição, opcionalidade e não-determinismo;
  - combinar as expressões ciclo de vida para formar o modelo ciclo de vida[87];
- desenvolver um modelo de operação
  - para cada operação do sistema, definir inicialmente os itens "assume" e "resulta" do esquema de descrição de operação:
    - descrever cada aspecto resultante da operação como uma subcláusula de "resulta";
    - usar o modelo ciclo de vida para achar os eventos de saída da operação;
    - verificar que "resulta" não admite valores indesejados;
    - incluir invariantes identificados para o sistema, nas cláusulas "assume" e "resulta";
    - avaliar as cláusulas "assume" e "resulta";
    - atualizar as operações e eventos do dicionário de dados;
  - extrair as cláusulas "envia", "lê" e "muda", das informações de "assume" e "resulta".

Checar o modelo de análise

- a especificação pode ser considerada completa em relação aos requisitos, se:
  - todos os possíveis cenários estão contidos no modelo ciclo de vida;
  - todas as operações do sistema estão definidas por um esquema;
  - todas as informações estáticas estão contidas no modelo de objetos do sistema;
  - qualquer outra informação está contida no dicionário de dados;
- um teste de consistência simples verifica se:
  - todas as classes, associações e atributos citados no modelo de operação estão presentes no modelo de objetos do sistema;
  - o limite que define o modelo de objetos do sistema é compatível com a interface do sistema estabelecida pelo modelo ciclo de vida
  - todas as operações do modelo ciclo de vida tem um esquema
  - todos os identificadores em todos os modelos tem uma definição no dicionário de dados;
- um teste de consistência semântica verifica se:

---

87 O alfabeto do modelo ciclo de vida é composto pelos eventos identificados. Operações (que correspondem a conjuntos de eventos) não fazem parte do alfabeto e podem ser representadas como expressões ciclo de vida (uma expressão ciclo de vida pode ser ainda uma parte de uma operação). Uma forma racional de compor o modelo ciclo de vida é fazer com que a expressão principal contenha apenas nomes de expressões ciclo de vida (substituições), que correspondam a operações do sistema (e cada operação seja detalhada abaixo).

- há consistência entre os eventos de saída dos modelos ciclo de vida e de operação;
- o modelo de operação preserva os invariantes definidos (no dicionário de dados) para o sistema;
- a "execução" de cenários é compatível com os esquemas do modelo de operação.

7.3.2 Passos do projeto

Desenvolver os grafos de interação de objetos

- construir um grafo de interação para cada operação identificada:
  - identificar os objetos envolvidos;
  - definir os papéis dos objetos (controlador e colaboradores);
  - definir as mensagens entre objetos;
  - construir o grafo;
- refinar os grafos desenvolvidos:
  - buscar outras alternativas de projeto, visando o aprimoramento da especificação (em aspectos como minimização de comunicações, alocação de funcionalidade, etc.);
  - decomposição hierárquica de grafos de operações complexas: um método de um grafo de interação de objetos pode gerar um esquema de modelo de operação, que gera um outro grafo de interação de objetos (o que pode se expandir por mais de um nível de profundidade);
- checar o modelo construído:
  - checar se cada classe do modelo de objetos do sistema aparece em pelo menos um grafo de interação de objetos;
  - checar se o efeito funcional de cada grafo é compatível com o especificado no modelo de operação - checar que toda cláusula "resulta" é satisfeita;

Desenvolver os grafos de visibilidade

- para cada classe, buscar nos grafos de interação de objetos as referências mantidas pela classe (visibilidade);
- classificar a visibilidade identificada;
- construir um grafo para cada classe;
- checar o modelo construído:
  - se a visibilidade de cada classe reflete as associações do modelo de objetos;
  - se um servidor exclusivo não é referenciado por mais de um cliente;
  - se todas as passagens de mensagem dos grafos de interação de objetos correspondem a referências nos grafos de visibilidade;

Desenvolver as descrições de classe[88]

- buscar os atributos dado no modelo de objetos;
- buscar os atributos objeto (referências) nos grafos de visibilidade;

---

88 Fusion propõe a seqüência de desenvolvimento de projeto aqui apresentada, porém há uma interdependência entre os grafos de herança e as descrições de classe. Assim, mesmo desconsiderando a iteratividade comum ao desenvolvimento de uma especificação, após a elaboração dos grafos de herança, é preciso atualizar as descrições de classe com informações de herança.

- buscar os métodos nos grafos de interação de objetos;
- buscar as superclasses (caso haja) nos grafos de herança;
- checar o modelo construído;

Desenvolver os grafos de herança
- buscar herança no modelo de objetos do sistema;
- nos grafos de interação de objetos e nas descrições de classe, buscar funcionalidade comum a mais de uma classe, o que pode indicar a possibilidade de herança;
- nos grafos de visibilidade, buscar classes que possuam uma estrutura de referências comum, o que também pode indicar a possibilidade de herança;
- checar o modelo construído.

7.4 Exemplo de uso da metodologia Fusion

Análise

Desenvolvimento do modelo de objetos

O desenvolvimento do modelo de objetos é semelhante ao que já foi procedido para as metodologias dos capítulos anteriores. Assim, não será procedido o desenvolvimento de um modelo de objetos para o domínio. O modelo de objetos da figura abaixo já é o modelo de objetos do sistema. O modelo de objetos do domínio poderia incluir adicionalmente, a figura do jogador (usuário) e os elementos físicos de interface (vídeo, mouse). Com isto, está praticamente definida a interface do sistema - exceto pela necessidade de identificar as operações do sistema.

Observe-se que o modelo de objetos é idêntico ao desenvolvido com OMT (figura 4.16), exceto pela ausência de métodos[89] e de atributos que guardam referências a outros objetos.

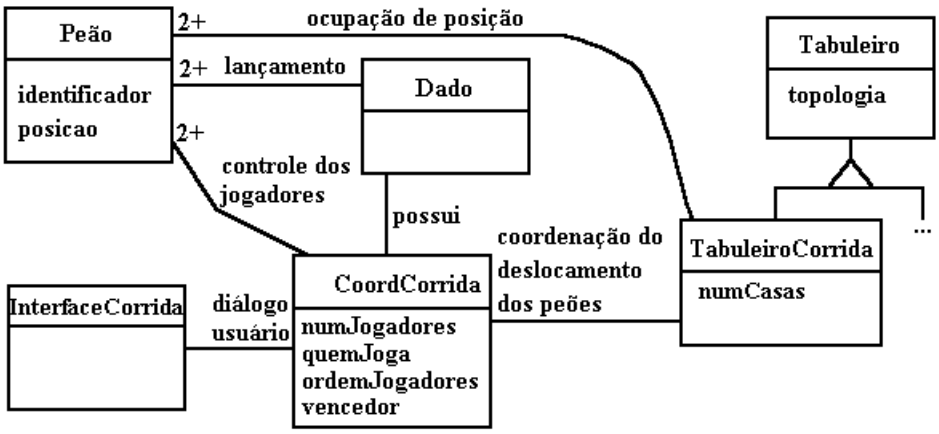


Figura 7.12 - modelo de objetos do sistema

Determinação da interface do sistema

agente: usuário;

---

89 Observe-se ainda que o modelo de objetos da análise de OMT também não apresenta métodos. Assim, em termos de análise esta diferença não haveria. Porém, OMT estende o desenvolvimento do modelo de objetos para projeto, o que não ocorre em Fusion.

operações do sistema: inicialização de uma partida, desenvolvimento de uma partida[90];
cenários para as operações:

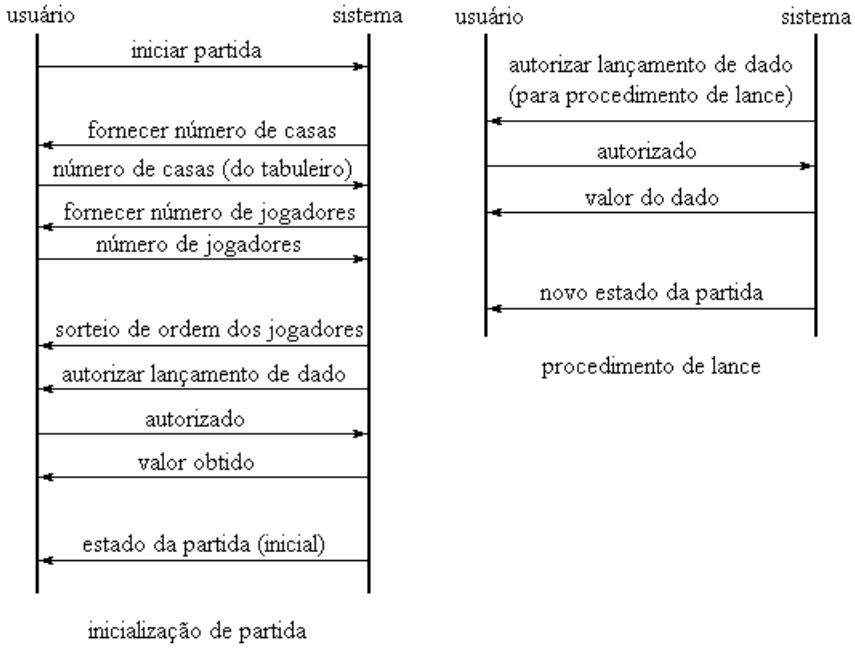


Figura 7.13 - cenários para as operações

Desenvolvimento do modelo de interface

Modelo ciclo de vida:

ciclo_de_vida JogoCorrida = inicialização_de_partida . (procedimento_de_lance)$^+$

inicialização_de_partida = iniciarPartida .
# fornecerNumeroDeCasas .
numeroDeCasas .
fornecerNumeroDeJogadores .
numeroDeJogadores .
sorteioDeOrdem .
(# autorizarLancamentoDeDado .
autorizado .
valorObtido)$^+$.
estadoDaPartida

procedimento_de_lance = (# autorizarLancamentoDeDado .
autorizado .
valorObtido)$^+$.
(# estadoIntermediarioDaPartida | # partidaConcluida)

90 Observe-se que as operações de Fusion são equivalentes aos *use cases* de OOSE e aos cenários de OMT.

Modelo de operações

Operação:	inicialização de partida
Descrição:	obtém do usuário o número de casas do tabuleiro e o número de jogadores; procede o sorteio para definição da ordem dos jogadores e apresenta partida no estado inicial
Lê:	
Muda:	número de casas do tabuleiro, número de jogadores, ordem dos jogadores
Envia:	valores de lançamento de dado, estado da partida, ordem dos jogadores
Assume:	
Resulta:	partida no estado inicial: a posição de todos os peões é a casa zero

Operação:	procedimento de lance
Descrição:	procede um lance da partida: peão lança dado e avança o número correspondente de casas; se atingir a última casa do tabuleiro, a partida está encerrada
Lê:	ordem dos jogadores, número de casas
Muda:	quem joga, posição do peão, vencedor
Envia:	valor(es) de lançamento de dado, estado da partida
Assume:	

Resulta:	se posicao = 0 e (dado = 1 ou dado = 6), posicao = 1;
	se posicao ≠ 0 e (numCasas - 1) > (posicao + dado), posicao = posicao + dado;
	se posicao = (numCasas - 1), vencedor = true;
	se dado ≠ 6, quemJoga = próximo da lista.

Projeto

Grafos de interação de objetos

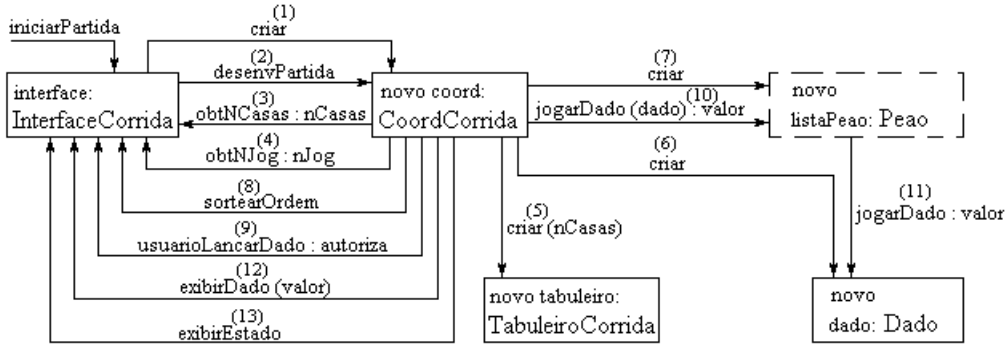


Figura 7.14 - grafo de interação de objetos para a operação inicialização_de_partida

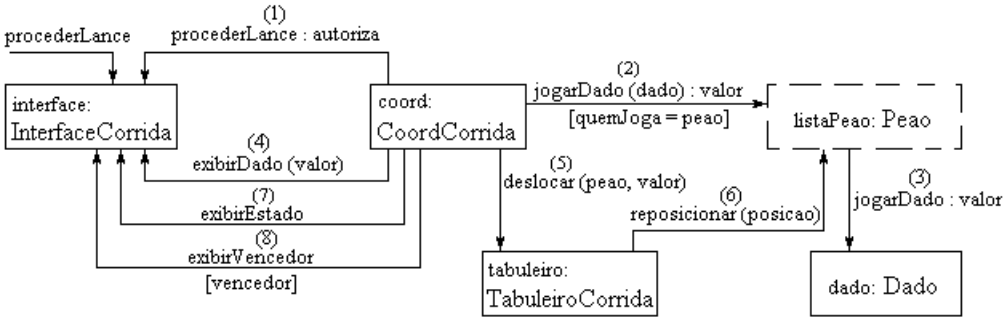


Figura 7.15 - grafo de interação de objetos para a operação procedimento_de_lance[91]

Grafos de visibilidade

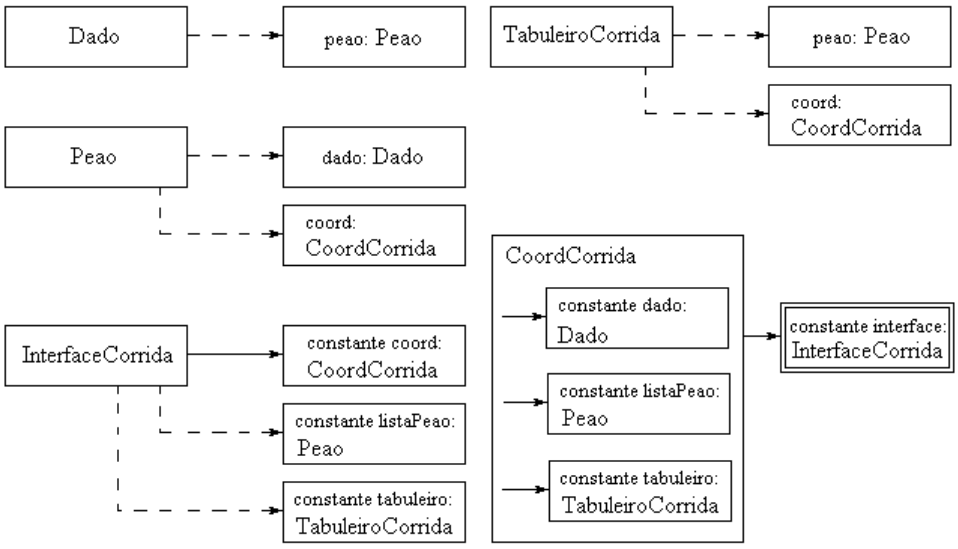


Figura 7.16 - grafos de visibilidade

Grafo de herança:

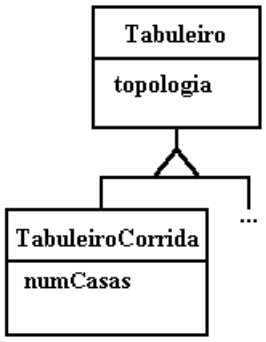


Figura 7.17 - grafo de herança

91 O procedimento de um lance ocorre por iniciativa da instância de CoordCorrida (coord), que solicita ao jogador da vez, a autorização para o procedimento de seu lance. Na sintaxe do grafo de interação de objetos, uma operação ocorre a partir de um evento produzido por um agente - o que não contempla a situação da operação procedimento de lance. Assim, a seta rotulada com "procederLance" que define o objeto interface como coordenador, expressa a resposta à comunicação do objeto coord e não o início da operação.

Descrições de classe:

Classe **CoordCorrida**

atributo **quemJoga** : Peao
atributo **numJogadores** : int
atributo **vencedor** : bool
atributo constante **interface** : exclusivo limitado InterfaceCorrida
atributo constante **tabuleiro** : limitado TabuleiroCorrida
atributo constante **dado** : limitado Dado
atributo constante **p1 ... pn**[92] : limitado Peao

metodo **desenvPartida**

Classe **Peao**

atributo **identificador** : string
atributo **posicao** : int

metodo **setIdent** (string : identificador)
metodo **getIdent** : string
metodo **reposicionar** (int : posicao)
metodo **getPosic** : int
metodo **jogarDado** (Dado : dado) : int

Classe **TabuleiroCorrida** é Tabuleiro

atributo **numCasas** : int

metodo **criar** (int : casas)
metodo **getDimens** : int
metodo **deslocar** (Peao : peao, int : casas)

Classe **Dado**

metodo **jogarDado** : int

Classe **InterfaceCorrida**

atributo constante **coord** : limitado CoordCorrida

metodo **obterNumCasas** : int
metodo **obterNumJog** : int
metodo **exibirEstado** (Peao : p1..pn, TabuleiroCorrida : tabuleiro)
metodo **sorteioOrdem**
metodo **usuarioLancarDado** : bool
metodo **exibirDado** (int : valor)
metodo **procederLance** : bool
metodo **exibirVencedor** (Peao : peao)
metodo **iniciarPartida**

---

92 Isto de fato, se refere a uma lista de peões, na ordem de procedimento de lances.

Dicionário de dados

O dicionário de dados será omitido da presente especificação, por já haver sido elaborado um dicionário de dados para o jogo Corrida no capítulo quatro, na modelagem segundo OMT. O dicionário de dados de Fusion difere pela possibilidade de descrever conceitos diferentes de classes, atributos e métodos. Está sendo admitido que produzir o dicionário de dados por esta diferença, não geraria um acréscimo de informação relevante, em termos da ilustração do uso da metodologia Fusion - haja vista a simplicidade do sistema jogo Corrida.

8 Avaliação das metodologias de desenvolvimento de aplicações

O objetivo do presente trabalho é a avaliação de metodologias de análise e projeto orientadas a objetos sob a ótica de sua utilização no desenvolvimento de frameworks. Um primeiro aspecto a considerar é que as metodologias descritas nos capítulos anteriores estão voltadas ao desenvolvimento de aplicações, e não de frameworks. Assim, neste capítulo as metodologias serão avaliadas e comparadas sob a ótica de seu uso no desenvolvimento de aplicações. No próximo capítulo serão apresentados os conceitos associados a frameworks e se avaliará como as metodologias voltadas a aplicações podem contribuir para o seu desenvolvimento.

8.1 As possíveis visões de um sistema

Uma modelagem de sistema construída segundo a abordagem de orientação a objetos - assim como sob outras abordagens - deve ter a capacidade de descrever este sistema sob uma ótica estática e sob uma ótica dinâmica, como discutido no capítulo 2.

Espera-se da modelagem estática, a descrição dos componentes de um sistema. Em um sistema orientado a objetos espera-se que a modelagem estática seja eficiente em identificar (durante sua elaboração) e descrever (quando concluída) os objetos (classes), seus atributos e relacionamentos. Em contrapartida, a elaboração da modelagem dinâmica de um sistema orientado a objetos deve levar à identificação e descrição da colaboração entre objetos e dos métodos das classes.

A descrição da estrutura estática neste contexto, é feita por um modelo de objetos (com esta ou outra denominação), que contém classes e associações entre classes (podendo conter ou não, atributos, métodos, subsistemas, aspectos de modelagem dinâmica incluídos etc.). Nas várias metodologias o modelo de objetos é basicamente uma variação do diagrama de entidade-relacionamento (ER), onde as classes correspondem às entidades e as associações entre classes, aos relacionamentos. Em função dos acréscimos sintáticos, uma técnica de modelagem pode ser mais expressiva do que outra, para destacar determinados detalhes da implementação, como a diferenciação entre associação estática e dinâmica adotada em OOSE e ausente em OMT, por exemplo.

A modelagem estática de um sistema apresenta uma certa uniformidade entre as metodologias orientadas a objetos. Não se observa nestas metodologias o mesmo grau de uniformidade em termos de modelagem dinâmica. O objetivo da modelagem dinâmica sob a abordagem de orientação a objetos, é descrever a colaboração entre objetos, bem como identificar e detalhar (para posterior implementação) os métodos das classes. Nas metodologias variam tanto os técnicas de modelagem usadas, como a heurística associada à tarefa de identificar e descrever métodos. De um modo geral as metodologias usam diagrama de transição de estado (com variações sintáticas e acréscimos diversos), descrições de cenários e outras técnicas de modelagem.

Além da questão da modelagem ser estática ou dinâmica, um outro aspecto a considerar é o foco da modelagem, ou seja, se volta a atenção ao sistema como um todo, ou a uma classe isoladamente. Em uma aplicação orientada a objetos, a implementação consiste de um conjunto de classes (que possuem atributos e métodos). Implementar um sistema é, portanto, implementar as classes que o constituem. Porém, para compreender

um sistema, normalmente não basta observar cada uma de suas classes, mas também é necessário dispor de alguma descrição da ligação entre as classes. Os técnicas de modelagem usados pelas metodologias também variam em termos da capacidade de descrever o sistema como um todo ou as suas partes (classes).

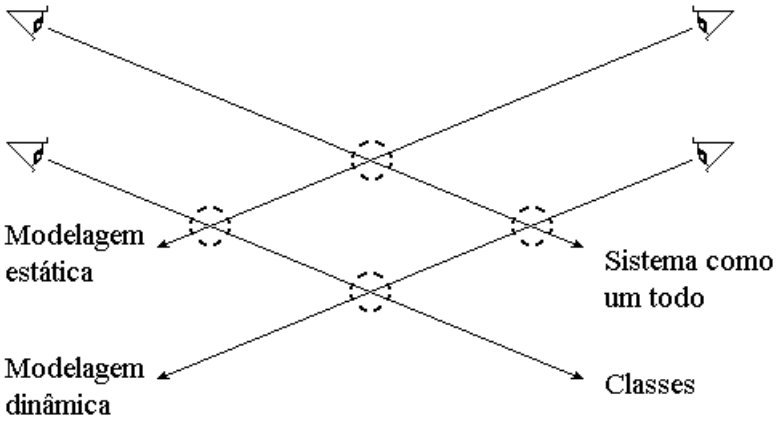


Figura 8.1 - visões de uma especificação de sistema

A descrição isolada de cada classe é importante, pois é a descrição mais próxima da implementação: isto inclui uma descrição completa de cada classe, incluindo os atributos (tipos, estruturas de dados, restrições etc.), os métodos (detalhamento do algoritmo associado ao método, explicitação da dependência entre métodos de uma classe etc.) e a interface, a parte do objeto externamente visível. Além da importância para a implementação, uma descrição completa de uma classe é importante para viabilizar sua manutenção e sua reutilização.

A descrição de um sistema orientado a objetos como um todo, é a descrição da interação entre as classes que compõem o sistema - como o sistema é construído a partir da composição de classes, qual a dependência entre classes etc. No desenvolvimento de um sistema, esta visão é fundamental para a distribuição de responsabilidades entre classes desenvolvidas (o que resulta na definição dos métodos destas classes) e classes reutilizadas (de biblioteca), como também para a integração dessas classes. Na manutenção, auxilia na localização de que classes devem ser alteradas para modificar ou incrementar a funcionalidade do sistema.

Em termos de modelagem estática, descrição do sistema como um todo consiste no conjunto de classes que compõem o sistema, suas associações e estruturas de herança e agregação. Descrição de cada classe separadamente consiste na relação dos atributos e assinaturas de métodos. Em termos de modelagem dinâmica, descrição do sistema como um todo consiste em descrever a interação temporal entre as classes, e a descrição de cada classe consiste na descrição do corpo métodos. Tendo em vista estes aspectos - que devem ser abordados em uma modelagem - passa-se à verificação de como eles estão refletidos nas cinco metodologias apresentadas.

As técnicas de modelagem se voltam a uma das possíveis visões do sistema, porém não exclusivamente. O modelo de objetos de OOSE, por exemplo, que é uma técnica de modelagem voltada para o aspecto estático, possui associações dinâmicas, que modelam um aspecto dinâmico.

8.2 Análise comparativa de técnicas de modelagem utilizadas pelas metodologias OOAD

8.2.1 Modelagem estática

Os modelos de objetos das metodologias descrevem a estrutura de classes que compõe um sistema, ou seja, classes, associações e estruturas de herança e agregação - o que corresponde a uma descrição do sistema como um todo. A representação de classes no modelo de objetos da metodologia de Coad e Yourdon agrupa o nome da classe e a relação de atributos e métodos; possibilita ainda a diferenciação de classes abstratas e classes concretas, a partir da notação de "classe & objeto". A representação de OMT é semelhante, porém, não diferencia classes abstratas, inclui tipagem para os atributos e relação de argumentos e tipo de retorno para os métodos. Em OOSE é possível incluir o nome da classe e a relação de atributos (com tipo e cardinalidade[93] - esta última, característica exclusiva de OOSE); não há diferenciação de classes abstratas e nem a relação de métodos. Na metodologia de Martin e Odell o símbolo de classe contém apenas o nome da classe[94]. Em Fusion classes abstratas são diferenciadas (escrevendo o nome da classe em itálico) e o símbolo de classe agrupa a relação de atributos, com tipagem.

Em termos de comparação, observam-se aspectos em que alguma das metodologias se destaca. A diferenciação de classe abstrata e classe concreta de Coad e Yourdon é de mais fácil leitura que a de Fusion - as demais metodologias não tratam esta diferenciação. Em termos de informação associada à relação de atributos, há destaque para OOSE, que além da tipagem, inclui cardinalidade. E em termos de informação associada à relação de métodos, há destaque para OMT, que inclui relação de argumentos (com tipagem) e tipo de retorno.

informação / metodologia	atibutos	métodos	diferenciação abstrata / concreta
Coad e Yourdon	nome	nome, argumentos	sim
OMT	nome, tipo	nome, argumentos, retorno	não representa
OOSE	nome, tipo, cardinalidade	não representa	não representa
Martin e Odell	não representa	não representa	não representa
Fusion	nome, tipo	não representa	sim

Tabela 8.1 - Informações representáveis no símbolo de classe, além do nome da classe

Quanto a associações, as metodologias podem ser divididas em dois grupos. Num primeiro grupo, que inclui OMT, Fusion e a metodologia de Martin e Odell, as associações são semanticamente semelhantes às relações do diagrama ER. Informam em um nível de abstração elevado que há alguma espécie de dependência entre as classes. Como isto vai se refletir em atributos ou métodos, ou em que classes, não é claro na

93 Quantos elementos daquele tipo de atributo um objeto pode conter.

94 Não apenas em termos de representação, a metodologia, não possui a noção de atributo na modelagem de objetos. Assim, o que os outras metodologias representam como atributo, Martin e Odell representam como associação entre duas classes.

representação do modelo de objetos. Esta informação deve ser buscada em outros modelos.

No segundo grupo, que inclui OOSE e a metodologia de Coad e Yourdon, há dois tipos de associação - estática e dinâmica. Neste caso uma representação estática (que pode ser rotulada ou não na metodologia de Coad e Yourdon, e que deve ser rotulada em OOSE) representa a necessidade de conhecer a existência de uma classe, o que se reflete em atributos. O fato da associação estática de OOSE ser direcional permite definir em que classe haverá atributo de referência à outra classe, sendo portanto, mais expressiva que a representação de Coad e Yourdon[95]. A associação dinâmica (que é parte da modelagem dinâmica, incluída no modelo de objetos) representa a necessidade de interação em tempo de execução, e por ser direcional (nas duas metodologias), permite destacar o objeto controlador[96] da interação - o que localiza em uma classe, pelo menos um método de resposta à solicitação. Em OOSE, por usar a mesma sintaxe da associação estática, a associação dinâmica não pode ser rotulada (uma seta não-rotulada significa associação dinâmica); na metodologia de Coad e Yourdon é recomendado que a seta contenha nomes de métodos - os métodos solicitados - ou nenhum rótulo, correspondendo neste caso a um nível de abstração mais elevado. A não-rotulação nos dois casos é uma deficiência que obriga a busca do significado da associação em outros modelos. Uma rotulação em um nível de abstração mais elevado que nome de método, seria uma solução mais adequada. Um aspecto questionável na notação de associações de OOSE (além da necessidade da duplicação de associações em caso de bidirecionalidade) é que uma associação estática indica conhecimento, mas não interação em tempo de execução. Assim, em uma situação em que duas classes mantenham referência uma da outra e que uma é cliente da outra, implica na existência de quatro associações entre estas classes no diagrama (duas estáticas e duas dinâmicas). Esta descrição assume um nível de abstração muito próximo da implementação. Isto torna o modelo de objetos de difícil leitura, pela tendência a conter grande quantidade de associações. Há outras abordagens possíveis para o tratamento de associações diferenciadas. Em [MOR 94], por exemplo, é adotada a seguinte semântica para as associações: a associação dinâmica implica em interação sem manutenção de referência, e a associação estática, manutenção de referência e que possibilita interação em tempo de execução.

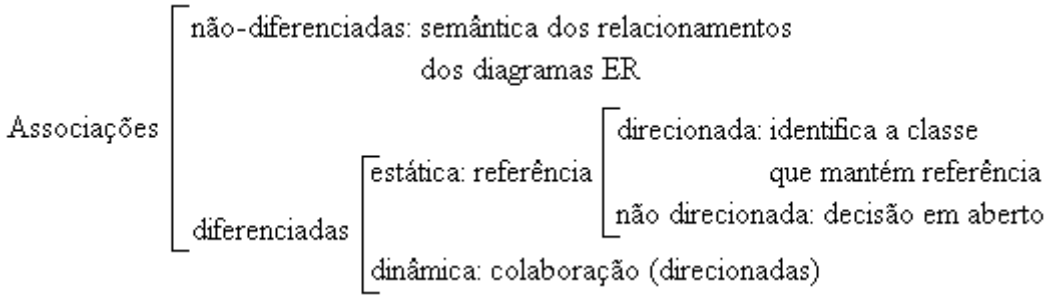


Figura 8.2 - associações usadas em modelos de objetos

95 Uma crítica quanto à sintaxe de OOSE, é que uma associação bidirecional (tanto estática quanto dinâmica) deve ser representada como duas associações unidirecionais, o que acarreta uma maior quantidade de elementos no modelo de objetos, tornado-o necessariamente mais complexo.

96 A expressão objeto controlador, originada na metodologia Fusion, se refere ao objeto que inicia uma operação do sistema. Os demais objetos envolvidos na operação são chamados colaboradores.

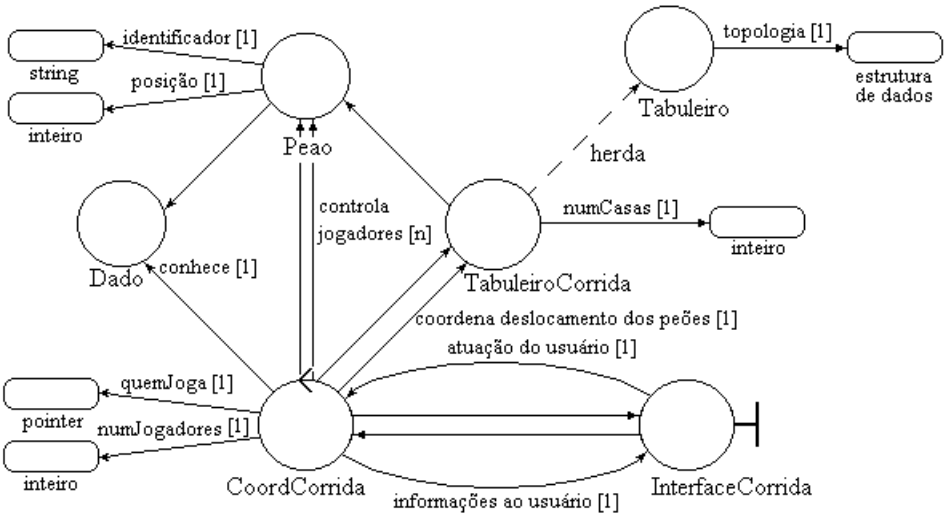


Figura 8.3 - modelo de objetos referente ao use case "procedimento de lances" (reprodução da figura 5.9)

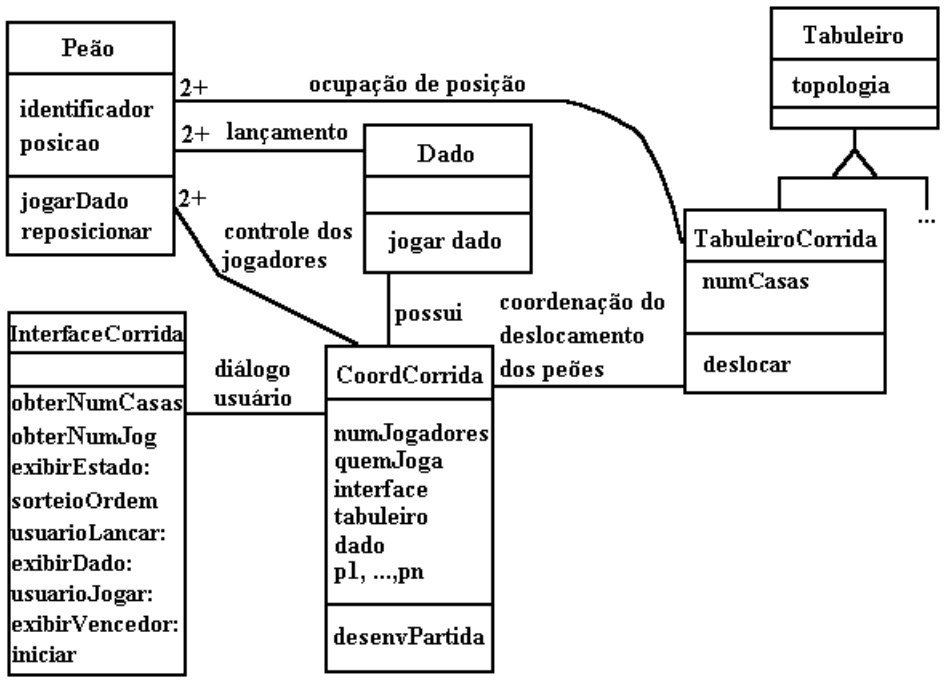


Figura 8.4 - modelo de objetos de OMT (reprodução da figura 4.16)

Apesar das deficiências ressaltadas, a representação de associações adotada por Coad e Yourdon e OOSE é mais expressiva que a das metodologias do primeiro grupo. Além disso, a direcionalidade da associação estática de OOSE a torna a mais expressiva. Uma ilustração da eficácia de diferenciar associações pode ser observada nos exemplos desenvolvidos. Observando as associações entre as classes CoordCorrida, Peao e Dado, na modelagem de OOSE, verifica-se que peao e dado (instâncias) interagem em tempo de execução, mas não mantêm conhecimento um do outro na forma de atributos - a referência do *dado* é passada pelo *coordenador* ao *peão* como argumento, em uma chamada de método. Isto, é expressso no modelo de objetos de OOSE, como pode ser observado na figura 8.3, que apresenta o modelo de objetos para o *use case* "procedimento de lances" desenvolvido no capítulo 5. Este nível de detalhamento não é claro no modelo de objetos de OMT. Isto pode ser observado na figura 8.4, que

apresenta o modelo de objetos de OMT para o jogo Corrida, desenvolvido no capítulo 4. Neste caso, a associação "lançamento" mantida entre peão e dado, pode significar manutenção de referência, interação, ou as duas coisas. O significado preciso deve ser buscado em outros modelos.

Todas as metodologias apresentam mecanismos sintáticos igualmente expressivos para a representação de herança e agregação.

Em todas as metodologias há a necessidade de complementar a modelagem estática com descrição textual - dicionário de dados em OMT e Fusion, e mecanismos semelhantes nas demais. Martin e Odell não citam esta necessidade, mas em relação às outras metodologias, sua modelagem ficaria bem menos expressiva sem esta complementação.

8.2.2 Modelagem dinâmica

A modelagem dinâmica do sistema como um todo, na metodologia de Coad e Yourdon, se resume às associações de comunicação (dinâmicas) incluídas no modelo de objetos. A modelagem dinâmica das classes isoladamente é composta por um diagrama de transição de estado para a classe, e um fluxograma para cada método. A descrição dos algoritmos dos métodos é eficiente para a implementação, porém a descrição da interação entre classes é bastante deficiente, tanto para produzir, como para interpretar a especificação: não há mecanismos de desenvolvimento que levem aos métodos (como os cenários, por exemplo), e nem formas de acompanhar a cadeia de cooperação gerada em uma chamada de método (exceto analisando cada algoritmo).

A modelagem dinâmica do sistema como um todo, em OMT, utiliza os diagramas de eventos (como ferramenta de identificação do comportamento dinâmico) e os diagramas de fluxo de eventos (parte do modelo dinâmico). Além deste, o DFD que constitui o modelo funcional, também descreve o comportamento do sistema. A descrição do aspecto dinâmico em OMT é mais expressiva que a de Coad e Yourdon. A modelagem dinâmica das classes isoladamente utiliza statecharts, que são mais expressivos que os diagramas de transição de estado usados por Coad e Yourdon. OMT estabelece que deve ser projetado o algoritmo dos métodos, mas não prevê onde colocar esta informação - o que constitui uma deficiência da metodologia.

Em OOSE, a modelagem dinâmica do sistema como um todo[97], inclui as associações dinâmicas do modelo de objetos (específicas por *use case*, ou não) e os diagramas de interação (semelhantes aos diagramas de eventos de OMT, porém, mais expressivos). A modelagem dinâmica das classes isoladamente, utiliza o diagrama SDL, que corresponde a um diagrama de transição de estado com elementos sintáticos de fluxograma, o que permite a descrição do algoritmo dos métodos. Os diagramas SDL descrevem todos os métodos em um mesmo diagrama (que pode ficar bastante extenso). Neste diagrama a descrição da existência de uma instância necessariamente começa com uma mensagem de criação, que cria e inicializa a instância. Os demais métodos consistem em ramificações no diagrama que, invariavelmente tem um recebimento de mensagem com o nome do método como primeiro elemento.

A metodologia de Martin e Odell dispõe do esquema de eventos, para a modelagem dinâmica do sistema como um todo (e que é a principal técnica de

---

97 Também existe o modelo de requisitos que modela o comportamento do sistema como um todo, porém, além dele ser basicamente textual, suas informações estão incluídas nos diagramas de interação.

modelagem). Como mecanismos auxiliares, dispõe do diagrama de mensagens e do diagrama de fluxo de objetos. O esquema de eventos é um diagrama de transição de estado, com extensões sintáticas, onde é descrito o comportamento global do sistema. O diagrama de fluxo de objetos se assemelha ao DFD. A informação representada no diagrama de mensagens corresponde às associações dinâmicas rotuladas com o conjunto de métodos associados a conexão. Para a descrição de cada classe separadamente só é proposto o diagrama fence (de transição de estados). O detalhamento do comportamento dinâmico associado às classes é protelado para a etapa de projeto, para que os autores não propõem técnicas de modelagem ou etapas de desenvolvimento de especificação.

Fusion utiliza o modelo de interface (modelo de operação mais modelo ciclo de vida) para a modelagem dinâmica do sistema como um todo, sob a ótica da interação do sistema com o meio externo - semelhante à abordagem do modelo de requisitos de OOSE, porém mais formalizado. Nesta modelagem elaborada durante a etapa de análise, o sistema é tratado como um único módulo, e não como um conjunto de classes. Os grafos de interação de objetos elaborados durante a etapa de projeto, modelam o comportamento dinâmico do sistema como um todo sob a ótica da interação entre objetos (de forma semelhante aos técnicas de modelagem que descrevem cenários em OMT e OOSE). Quanto à modelagem dinâmica das classes isoladamente, Fusion não utiliza diagrama de transição de estado e nem dispõe de de técnica de modelagem para a descrição do corpo dos métodos, a não ser a descrição textual contida no dicionário de dados.

Em termos de modelagem dinâmica do sistema como um todo, a metodologia de Coad e Yourdon se mostra a de menor capacidade de descrição, pois só dispõe de conexões de mensagem, mecanismos equivalentes às conexões dinâmicas de OOSE - a conexão dinâmica de OOSE perde em expressividade, por não ser rotulada.

Os diagramas de fluxo de eventos de OMT e o diagrama de mensagens de Martin e Odell são outras técnicas de modelagem que se equivalem. A metodologia de Coad e Yourdon, OOSE e Fusion não dispõem de mecanismos deste tipo.

Outra equivalência em termos do tipo de informação expressa, se observa entre as técnicas de modelagem esquema de eventos (Martin e Odell), diagrama de eventos (OMT), diagrama de interação (OOSE) e grafo de interação de objetos (Fusion). Todas trabalham com as noções de evento, e da evolução de estados do sistema a partir da ocorrência de eventos. A primeira técnica de modelagem, por descrever o sistema como um elemento e não como um conjunto de classes, se mostra menos adequada que as outras à abordagem de orientação a objetos (a associação das informações às classes não é trivial, como nos outros técnicas de modelagem). O diagrama de interação de OOSE é mais expressivo que o diagrama de eventos de OMT, por permitir representação da duração do método, de execução condicional, looping e não-determinismo. O grafo de interação de Fusion e o diagrama de eventos de OMT se equivalem em expressividade, exceto pela possibilidade de incluir guardas nos eventos de Fusion - que possibilita condicionar a sua ocorrência, mas não com a mesma expressividade da técnica de modelagem de OOSE.

Quanto ao DFD usado em OMT e o diagrama de fluxo de objetos, de Martin e Odell, expressam comportamento funcional do sistema como um todo (sem a

capacidade de descrever a funcionalidade das classes), o que pode ser incluído nos outros modelos, dispensando seu uso[98].

Observa-se no aspecto da descrição do comportamento dinâmico do sistema como um todo, um certo equilíbrio entre as metodologias (exceto a de Coad e Yourdon, que se mostra deficiente). Um destaque mais importante é o diagrama de interação de OOSE, que se mostra mais expressivo que os mecanismos equivalentes das outras metodologias.

informação / metodologia	aspectos de modelagem dinâmica incluí-dos no modelo de objetos	enumeração das interações em tempo de execução (colaboração)	evolução de estados a partir da ocorrência de eventos[99]
Coad e Yourdon	conexões de mensagem	não apresenta	não apresenta
OMT	não apresenta	diagrama de fluxo de eventos	diagramas de eventos (cenários)
OOSE	conexões dinâmicas	não apresenta	diagramas de interação (cenários)
Martin e Odell	não apresenta	diagrama de mensagens	esquema de eventos
Fusion	não apresenta	não apresenta	grafo de interação de objetos

Tabela 8.2 - equivalências observadas entre as técnicas de modelagem usadas para a descrição dinâmica do sistema como um todo

Quanto à modelagem dinâmica das classes separadamente, a metodologia Fusion não adota técnicas de modelagem, sendo a menos expressiva. A metodologia de Martin e Odell só propõe o uso de diagrama de transição de estado[100]. A modelagem da metodologia de Coad e Yourdon se mostra mais expressiva que a de Martin e Odell e Fusion, pois além de diagrama de transição de estado, inclui fluxogramas para a descrição dos métodos. O diagrama SDL usado por OOSE inclui as informações referentes à transição de estado, bem como a descrição do algoritmo dos métodos, equivalendo à metodologia de Coad e Yourdon em expressividade - porém como concentra toda a descrição da classe em um mesmo diagrama, de um único nível hierárquico, tende a prejudicar a inteligibilidade do modelo. OMT utiliza statecharts, cuja expressividade é maior que a dos diagramas de transição de estados (utiliza diagramas de múltiplos níveis hierárquicos, além de representar concorrência). Em termos de descrição de transição de estados, o mecanismo de OMT é superior em

98 Considera-se aqui a dispensa do uso do DFD como parte da documentação de projeto, e não como ferramenta adicional para busca de informações sobre o sistema, durante a modelagem. Isto volta a ser discutido na avaliação de OMT.

99 Apenas a metodologia de Martin e Odell utiliza a noção de estado do sistema (o modelo de ciclo de vida de Fusion fornece esta visão, sem descrever os estados). Nas metodologias de OMT e OOSE os estados do sistema são representados na especificação como o intervalo entre eventos, sem a preocupação em proceder sua descrição. Nestas metodologias (assim como na metodologia de Coad e Yourdon) a descrição de estados é feita para as classes, separadamente. A descrição do estado do sistema nestes casos geralmente pode ser observada na descrição de estados do objeto controlador do sistema.

100 Deve ser considerado porém, que a metodologia não inclui projeto e que sempre pode ser afirmado que este tipo de modelagem será realizado nesta etapa.

expressividade ao das outras metodologias, porém falta a OMT precisar como e onde serão expressos os algoritmos dos métodos.

metodologia	técnica de modelagem
Coad e Yourdon	diagrama de transição de estados
OMT	statechart
OOSE	diagrama SDL
Martin e Odell	diagrama de transição de estados
Fusion	não apresenta

Tabela 8.3 - equivalências observadas entre as técnicas de modelagem usadas para a descrição dinâmica das classes, sob a ótica de evolução de estados a partir da ocorrência de eventos

metodologia	técnica de modelagem
Coad e Yourdon	fluxograma
OMT	não apresenta
OOSE	diagrama SDL
Martin e Odell	não apresenta
Fusion	não apresenta

Tabela 8.4 - mecanismos para a descrição da funcionalidade associada às classes - detalhamento do algoritmo dos métodos

8.3 Avaliação das metodologias considerando conjunto de técnicas de modelagem e processo de desenvolvimento

8.3.1 Metodologia de Coad e Yourdon

A metodologia de Coad e Yourdon apresenta como principal vantagem a sua simplicidade - o que também constitui sua principal desvantagem. O mérito desta metodologia reside na simplicidade das técnicas de modelagem e dos procedimentos de trabalho que usa, o que torna a metodologia fácil de ser aprendida e utilizada. Esta característica, porém, limita seu uso a aplicações pouco complexas. Os autores não propõem uma forma eficaz de determinação dos métodos das classes; não há uma modelagem mais elaborada do comportamento dinâmico do sistema, com vista à identificação dos métodos. Na etapa de definição dos métodos, estes surgem a partir do conhecimento do domínio e são descritos. Isto é diferente de outras metodologias, como OOSE, que utiliza o diagrama de interação para modelar a interação entre classes e a partir daí, extrair os métodos.

Esta deficiência se reflete não apenas na modelagem, como na interpretação de uma especificação a partir da metodologia de Coad e Yourdon. A descrição de cada classe é clara, mas não a interação entre as classes.

Uma outra vantagem desta metodologia é o uso dos mesmos mecanismos nas etapas de análise e projeto, e com relação direta aos elementos da implementação: as classes identificadas nas etapas de análise e projeto correspondem às classes implementadas.

8.3.2 Metodologia OMT

OMT adota um procedimento claramente delimitado em etapas (análise e projeto), para a construção de uma especificação: constrói seus três modelos na análise (sem enfatizar a identificação de métodos de classe) e completa estes modelos no projeto. No conjunto (técnicas de modelagem e processo de desenvolvimento) se mostra a mais versátil das quatro metodologias pela possibilidade dos modelos se completarem mutuamente: durante a construção de um modelo são levantadas informações que complementam ou levam à alteração dos demais. Isto permite que a construção de uma especificação ocorra de forma iterativa[101]. OMT tem a capacidade de se adaptar à modelagem de sistemas diferentes (sistemas de bancos de dados, de interface interativa etc.), através do aumento ou diminuição da carga de informação - e conseqüentemente, da importância - de cada um dos três modelos.

Além do que já foi comentado quanto às técnicas de modelagem usadas por OMT, pode-se observar algumas deficiências. OMT dá grande importância à descrição de associações do modelo de objetos, e inclui nestas, as informações que são passíveis de representação nos relacionamentos dos diagramas ER. No modelo de objetos de OMT, se uma especificação incluir todas as informações permitidas (atributos, métodos, associações ternárias etc.), obrigará a que algumas associações do modelo de objetos sejam traduzidas como classes, na implementação. A decisão se uma associação se tornará uma classe na implementação, ou se suas informações serão distribuídas a outras classes, não é trivial. Isto transfere para a implementação uma responsabilidade maior que a simples tradução de classes, como ocorre na metodologia de Coad e Yourdon, por exemplo. OMT, porém, é flexível o bastante para que se possa fugir deste caso (evitando um estilo de modelagem que concentre informação em associações).

O modelo funcional, por utilizar DFD, herda das abordagens não-orientadas a objetos a característica de descrever o sistema sob uma ótica funcional e estender esta ótica aos vários níveis de refinamento. Falta-lhe a capacidade de transferir a funcionalidade identificada no sistema, mais diretamente para seus componentes (as classes): associar um processo do DFD a um método de uma determinada classe não é um procedimento trivial; além disto, os dados presentes em um DFD servem como buffers entre processos, não sendo relacionados de forma direta aos elementos da estrutura de classes. O DFD porém, pode ser útil no estágio inicial de desenvolvimento para modelar a funcionalidade da aplicação numa ótica top-down, contribuindo para um melhor entendimento do sistema, o que viria a auxiliar na identificação das classes de objetos[102]. Não deve contudo, constituir um dos pilares de uma especificação de projeto orientada a objetos, como adotado em OMT. De fato, o uso de DFD como parte da documentação de projeto, conforme já comentado, poderia ser dispensado - os outros técnicas de modelagem tem a capacidade de expressar as suas informações[103].

Quanto ao modelo dinâmico, uma deficiência pode ser observada no diagrama de eventos - e que reflete na sua correspondência com o diagrama de estados (statechart). O diagrama de eventos permite representar apenas uma execução do sistema "em linha reta", isto é, não possibilita a representação de execução, de looping

---

101 Em contraste com metodologias que estabelecem uma seqüência de construção de modelos, em que o modelo anterior fornece informações aos modelos posteriores, induzindo a construção da especificação, a um ciclo de vida *waterfall*.

102 Este ponto de vista é adotado na metodologia de Martin e Odell.

103 Supõe-se aqui o uso de algum mecanismo de descrição dos algoritmos dos métodos, como fluxograma ou pseudocódigo, à semelhança de OOSE e da metodologia de Coad e Yourdon.

(um conjunto de eventos que se repete, antes da ocorrência de outro evento) e de não-determinismo (que em determinado momento existe a possibilidade de ocorrência de um, entre um conjunto de eventos). Esta característica impede uma correspondência direta entre os eventos do diagrama de eventos e do diagrama de estados: a passagem do primeiro diagrama para o segundo exige refinamento dos eventos identificados (já que o diagrama de estados permite representação de execução condicional, looping e não-determinismo), identificação de novos eventos (já que é complexo prever cada caminho de execução em um diagrama de eventos) - e tudo isto tendo o cuidado de manter a coerência entre eventos compartilhados por diferentes diagramas de estados.

8.3.3 Metodologia OOSE

Como OMT, OOSE também adota um seqüenciamento de etapas sem lacunas, para a geração de uma especificação. Além de um modelo de objetos com vantagens em relação a outros (apesar da representação de classes ser menos expressiva), utiliza mecanismos expressivos para a modelagem dinâmica - os diagramas de interação e os diagramas SDL.

Algumas deficiências podem ser observadas. Uma primeira, bastante clara é o excesso de informalismo na análise. O modelo de requisitos é praticamente todo baseado em descrição textual, com todos os riscos de incompleteza e ambigüidade, característicos deste tipo de descrição. Também o modelo de análise é muito dependente de descrição textual - é composto de modelos de objetos associados aos *use cases* e descrição textual. Somente no modelo de projeto é que a descrição é mais baseada em técnicas de modelagem semiformais.

Outro ponto de validade duvidosa é a própria característica da metodologia de ser dirigida a *use cases*. Os *use cases* constituem um mecanismo de validade inquestionável, para a modelagem dinâmica - são usados com outra denominação em OMT e Fusion, e fazem falta na metodologia de Coad e Yourdon. É questionável porém, a tendência da centralização da modelagem nos *use cases* produzir uma especificação excessivamente voltada à funcionalidade, com as deficiências observadas nas abordagens funcionais - dificuldade de manutenção e de reutilização, características buscadas pela orientação a objetos [MEY 88] [COA 92]. Em particular, a concentração de funcionalidade do *use case* em objetos de controle parece uma transgressão do conceito de tipo abstrato de dado - pilar teórico da orientação a objetos - por gerar elementos essencialmente funcionais, mais relacionados com alguma funcionalidade específica de um caso de uso da aplicação (*use case*), que com os objetos semânticos - funcionalidade esta mais sujeita a alterações, que os conceitos do domínio, como já discutido [MEY 88]. Também o fracionamento do modelo de objetos em diagramas por *use cases* - sem a obrigatoriedade de existir um diagrama que contenha o conjunto de classes - dificulta a visão global do sistema. No exemplo desenvolvido no capítulo 5 se optou por construir um único modelo de objetos para projeto, para registrar esta visão global, porém, isto não é obrigatório - o que possibilita a propagação desta visão fracionada ao longo de todas as etapas, sem a existência de outra técnica de modelagem que garanta a existência de uma visão global, na especificação.

Esta característica da metodologia de ser voltada a funcionalidade a torna mais próxima da análise estruturada - que a metodologia de Coad e Yourdon, por exemplo - o que pode ser uma vantagem, na medida que deve ser de mais fácil assimilação aos

usuários desta abordagem - vantagem vista com reservas, em função das limitações da análise estruturada [COA 92].

8.3.4 Metodologia de Martin e Odell

A visão conceitual de classes e atributos contida na metodologia de Martin e Odell além de estranha em relação ao paradigma de orientação a objetos, como discutido no capítulo 6, e à visão das demais metodologias, não encontra correspondência em linguagens de programação orientadas a objetos (como reconhecem os autores). Assim, deixa para a etapa de projeto (não tratada pela metodologia) a incumbência de compatibilizar a visão conceitual da análise com mecanismos implementáveis[104]. Além disto, a modelagem estática se resume a um modelo de objetos (menos expressivo que os das outras metodologias), e a modelagem dinâmica, a uma descrição global do sistema. A metodologia se constitui na menos eficiente das quatro avaliadas. Apesar do argumento de que aspectos não abordados na metodologia, que é de análise, seriam abordados em projeto, a especificação de Martin e Odell é menos expressiva que a modelagem de análise das demais metodologias, e menos fiel ao paradigma de orientação a objetos.

8.3.5 Metodologia Fusion

A etapa de análise de Fusion utiliza duas técnicas de modelagem que se complementam mutuamente e que podem ser desenvolvidos em paralelo - o modelo de objetos e o modelo de interface. Esta característica promove um entendimento gradativo do sistema sob análise, e um processo iterativo de especificação durante a análise - semelhante ao que ocorre em OMT.

A descrição de operações (ou de cenários, ou de *use cases*) corresponde à descrição de uma possível situação de processamento a que o sistema pode ser submetido. O modelo de ciclo de vida utilizado por Fusion é uma técnica de modelagem capaz de descrever as seqüências de operações possíveis para o sistema, desde o início até o encerramento de uma execução, ou seja, descreve o ciclo de vida de uma execução do sistema - algo que não é feito pelas outras metodologias analisadas através de uma técnica de modelagem específica, constituindo um mérito de Fusion.

Fusion adota a diferenciação entre atributos dado e atributos objeto. Esta distinção juntamente com o grafo de visibilidade, fazem com que a especificação de um sistema contenha a relação de objetos referenciados por cada classe, de uma forma destacada e de fácil leitura. Analisando os exemplos de especificação desenvolvidos com as metodologias apresentadas, observa-se a capacidade do grafo de visibilidade de tornar claro o conjunto de referências mantido por cada classe, bem como detalhes destas referências (exclusividade, constância etc.).

A metodologia Fusion apresenta um certo inconveniente decorrente da distinção aplicada aos atributos: é estabelecido que o modelo de objetos (construído na etapa de

---

104 É questionável a validade de justificar a incapacidade de gerar uma especificação completa com o pretexto de que se está procedendo análise, e a especificação se tornará completa no projeto - principalmente quando não se discute o que fazer na etapa de projeto. Este tipo de justificativa se apóia no fato de não haver um limite unanimemente aceito entre as responsabilidades da etapa de análise e as responsabilidades da etapa de projeto.

análise) não deve apresentar atributos objeto. Isto é compreensível quando o atributo objeto simplesmente mantém uma referência a um objeto com que interage em tempo de execução - esta informação é expressa pelas associações, em um nível de abstração mais elevado. Há situações, porém, em que é importante que o atributo objeto esteja presente no modelo de análise, por conter uma informação relevante para o entendimento do sistema. Um exemplo disso se observa para a classe CoordCorrida do exemplo desenvolvido. Esta classe mantém referência aos peões do jogo - atributos objeto dispensáveis no modelo de análise, conforme descrito - porém, a classe CoordCorrida possui o atributo "quemJoga", que é um atributo objeto porque é uma referência a um peão, mas que contém a informação de que a responsabilidade de conhecer quem procede cada lance, é do coordenador. No exemplo desenvolvido este atributo objeto foi incluído no modelo de objetos, apesar da recomendação contrária da metodologia.

Durante toda a etapa de análise a visão dinâmica considera o sistema como um único módulo e a interação do sistema com o meio externo é descrita no modelo de interface, através da especificação de operações. É questionável a validade de protelar a noção de interação entre objetos para a etapa de projeto, em um sistema especificado segundo a abordagem de orientação a objetos - questiona-se se de fato ocorre análise orientada a objetos, se nenhum aspecto dinâmico identificado é associado às classes.

Outra característica negativa observada é que o modelo de operação não é capaz de fornecer uma visão temporal da seqüência de passos para a execução da operação - não sendo capaz de descrever completamente as operações. Esta visão pode ser expressa no modelo de ciclo de vida. Porém, não há obrigatoriedade de uma operação constituir um nome de substituição do modelo de ciclo de vida, e nem as operações fazem parte do alfabeto deste modelo.

O aspecto negativo mais relevante de Fusion é a ausência na etapa de projeto, de mecanismos de descrição da evolução de estados dos objetos e de descrição do algoritmo dos métodos. Descrição do algoritmo dos métodos é feita apenas no dicionário de dados, de forma textual. Comparando a construção de uma especificação em Fusion com a construção de uma especificação em OOSE, por exemplo, em Fusion ficaria faltando toda a etapa de "descrição do comportamento dinâmico dos blocos". Em Fusion, construção da especificação de projeto é considerada concluída quando as descrições de classe estão completas - e as descrições de classe possuem apenas as assinaturas de métodos. Assim, é deixada para a etapa de implementação a definição dos algoritmos, o que parece uma conclusão precipitada da etapa de projeto, que implica em uma especificação incompleta - considerando que a descrição da evolução de estados dos objetos e do corpo dos métodos deve fazer parte da especificação de um sistema.

8.4 Ilustração de alguns aspectos tratados na comparação de metodologias a partir de um outro exemplo de aplicação

Definição de uma nova aplicação

Aspectos referentes a deficiências de técnicas de modelagem utilizadas e de processos de desenvolvimento adotados, mostram-se mais prejudiciais quanto maior a complexidade da aplicação tratada. Como a aplicação usada para ilustrar cada metodologia era relativamente simples, será proposta a seguir uma nova aplicação - um pouco mais complexa - e discutidas as deficiências das metodologias frente à elaboração de sua especificação. Não é intenção da apresentação desta nova aplicação nem elaborar um segundo exemplo com cada metodologia, e nem reapresentar a avaliação de cada

metodologia - o que viria a estender o presente trabalho, sem uma carga de informações adicionais relevante. O procedimento adotado será a apresentação informal dos requisitos, e a avaliação do impacto da maior complexidade sobre as metodologias - em função de suas características.

A nova aplicação consiste de uma modificação das regras do jogo Corrida anteriormente proposto, e será tratada como jogo *Corrida "Plus"*. A diferença é a associação de penalizações e bonificações às casas do tabuleiro. Isto tornará o procedimento de um lance mais complexo, pois não se tratará mais de deslocar um peão de uma casa até outra, mas também avaliar a conseqüência de ocupar esta nova casa. Além disto, ao contrário do jogo Corrida, o lance de um jogador poderá afetar outro jogador, o que intensifica a interação entre objetos. A descrição das características do jogo *Corrida "Plus"* é apresentada a seguir.

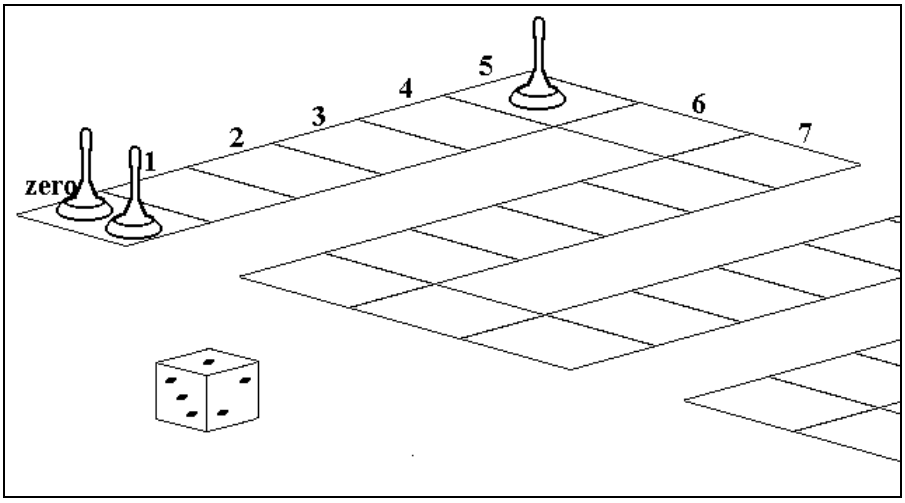


Figura 8.5 - ilustração do jogo "Corrida Plus"

Descrição das regras

O jogo "Corrida Plus" envolve dado, peões e tabuleiro. O jogo de que poderão participar dois ou mais jogadores, consiste em deslocar um peão ao longo das casas de um tabuleiro, da posição inicial, à posição final. O jogador que chegar primeiro à posição final é o vencedor. O número de casas do tabuleiro que um jogador desloca o seu peão é definido a partir do lançamento de um dado.

O tabuleiro define um percurso linear (sem bifurcações) e tem um determinado número de "casas" numeradas. A posição inicial corresponde ao número zero e para sair dela, o jogador deve obter no dado o valor 1 ou o valor 6. Com qualquer dos dois valores o jogador passa para a posição 1 (casa 1); com qualquer outro valor, permanece na casa zero. Os demais movimentos ao longo da partida consistem em deslocar o peão de um número de casas igual ao número obtido no dado. Quando o jogador obtiver no dado um valor que deslocaria seu peão para além da última casa, não é procedido movimento (o peão permanece na casa em que estava antes do lançamento do dado). Sempre que o jogador obtiver o valor 6 no dado, terá direito a realizar mais um lance (além daquele que esta sendo realizado, e em que foi obtido o valor 6).

À exceção das casas zero e um, nenhuma outra casa pode ser ocupada por mais de um peão. Quando em um lance um peão passar a ocupar uma casa anteriormente ocupada por outro peão, o primeiro ocupante será penalizado com um recuo de seis casas (penalização por ter sido alcançado). Caso o recuo do peão penalizado o leve a

uma casa já ocupada, recuará mais uma casa - este procedimento se repetirá até que o peão penalizado chegue a uma casa vazia. Um peão pode recuar apenas até a casa um, mesmo que isto implique em um recuo de número de casas inferior ao estabelecido - por exemplo, um peão alcançado na casa quatro recuará até a casa um, mesmo se esta estiver ocupada.

Penalizações e bonificações podem estar associadas às casas do tabuleiro. Penalizações correspondem a recuos de um certo número de casas ou à perda de um ou mais lances; bonificações, a um avanço de casas ou ao direito a um número de lances consecutivos. A Informação da quantidade de casas (a recuar ou a avançar) ou do número de lances (ganhos ou perdidos) está associada à casa que possui a penalização ou bonificação.

O jogador que realiza o primeiro lance é aquele que em um sorteio prévio, tirar o número mais elevado num lançamento de dado. A ordem dos demais acompanha a ordem decrescente dos valores obtidos no dado. Empates devem ser resolvidos por um novo lançamento de dados (envolvendo apenas os jogadores empatados, para resolver o que fica na posição de ordenamento disputada - o perdedor, no caso de dois, fica na posição de ordenamento seguinte).

Requisitos para um sistema que implemente o jogo corrida.

- Apresentação no vídeo dos elementos do jogo: dado e tabuleiro com peões posicionados;
- Diferenciação visual dos peões, que os identifique;
- Interferência do usuário através de teclado ou mouse;
- O sistema deve antes de uma partida, perguntar ao usuário o número de jogadores e a dimensão do tabuleiro;
- O sistema deve apresentar uma configuração "default" de casas com penalização e bonificação, mas deve ser possível ao usuário criar novas configurações, se assim desejar, durante a inicialização (após definir o número de casas do tabuleiro);
- O sistema deverá proceder a determinação de quem inicia a partida;
- O sistema deverá determinar o jogador que deve realizar um lance e solicitar que jogue;
- O sistema deverá informar quando um jogador for vencedor e indicar o final da partida;
- O valor obtido no lançamento de um dado deve ser apresentado no vídeo;
- Uma partida deve poder ser interrompida a qualquer instante.

Aspectos da modelagem da nova aplicação a partir das metodologias tratadas

Em termos de identificação de classes e definição de estruturas, como já citado, as metodologias se equivalem. Assim, seus modelos de objetos serão equivalentes. A figura 8.6 apresenta o modelo de objetos do jogo Corrida Plus, segundo a notação de OMT. No modelo apresentado foram omitidos os métodos de classes - é um modelo preliminar.

A principal diferença entre este modelo de objetos e o do jogo corrida em termos de estrutura de classes, é a presença da classe Casa. Diferente do jogo corrida, uma casa de um tabuleiro pode ter características de penalização ou bonificação associadas e além disto deve conhecer o peão que a ocupa - para o tratamento do caso em que um outro peão venha a ocupar a mesma casa.

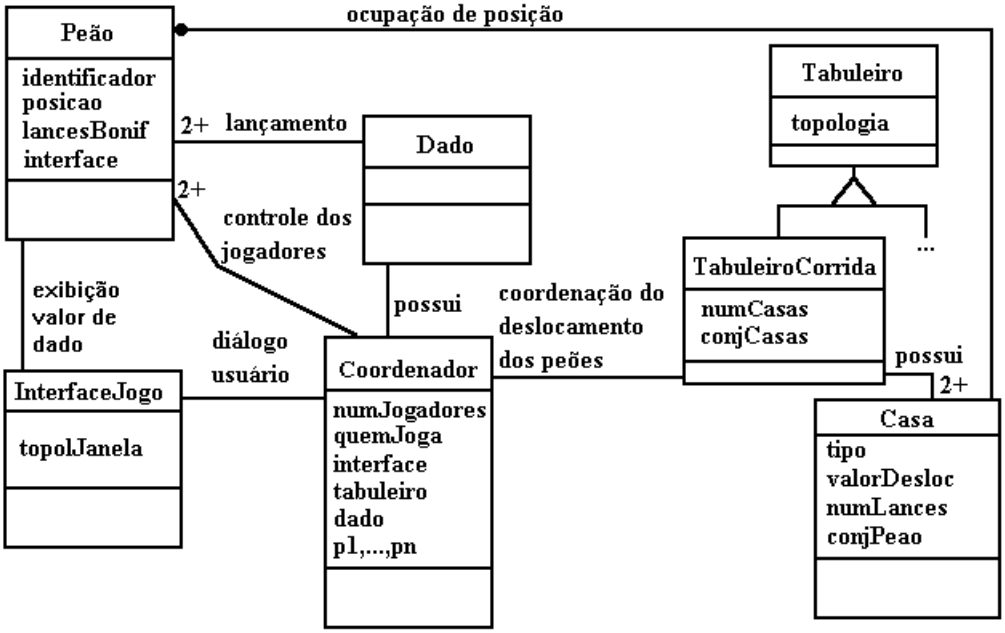


Figura 8.6 - Modelo de objetos do jogo Corrida Plus

A principal diferença entre a construção da especificação das duas aplicações reside na modelagem dinâmica. Neste aspecto pesa a maior ou menor capacidade de cada metodologia em identificar os métodos de classe - e em função dos métodos, novos atributos. As figuras 8.7 a 8.13 apresentam alguns dos possíveis caminhos de processamento no procedimento do lance de uma partida, a partir de diagramas de eventos (da metodologia OMT). No jogo Corrida o procedimento de um lance consiste simplesmente em lançar o dado e reposicionar o peão; no jogo Corrida Plus, além disto deve ser aplicada a bonificação ou penalização associada à nova casa ocupada - caso haja - e penalizar o peão que esteja ocupando a casa - se um peão a estiver ocupando. A busca destas variações de caminho de processamento deixa transparecer a necessidade de métodos que não estão claros à primeira vista. Por exemplo, a classe Casa dispõe de dois métodos diferentes para alojar um peão: alojaExclusivo, para alojar incondicionalmente um peão na casa (o peão que está procedendo o lance), e alojaSeVazio, para alojar um peão penalizado, ou seja, só alojará o peão se a casa estiver vazia. Uma outra diferença é a necessidade de o peão manter uma referência do objeto de interface, para acessar o método exibirDado - diferente da modelagem centralizada do jogo Corrida, que foi procedida nos capítulos anteriores. A necessidade deste atributo também é mostrada pelas descrições de cenário. Nas metodologias de Martin e Odell e de Coad e Yourdon a constatação destas necessidades resulta do esforço intelectual do desenvolvedor, enquanto nas metodologias OMT, OOSE e Fusion, resulta quase diretamente da descrição de cenários (*use cases*).

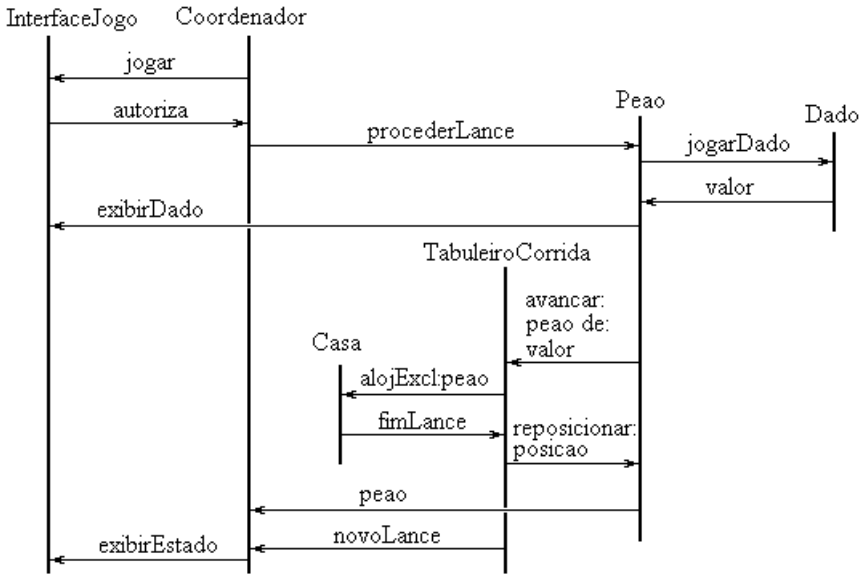


Figura 8.7 - desenvolvimento de um lance em uma partida - peão avança para casa simples, desocupada

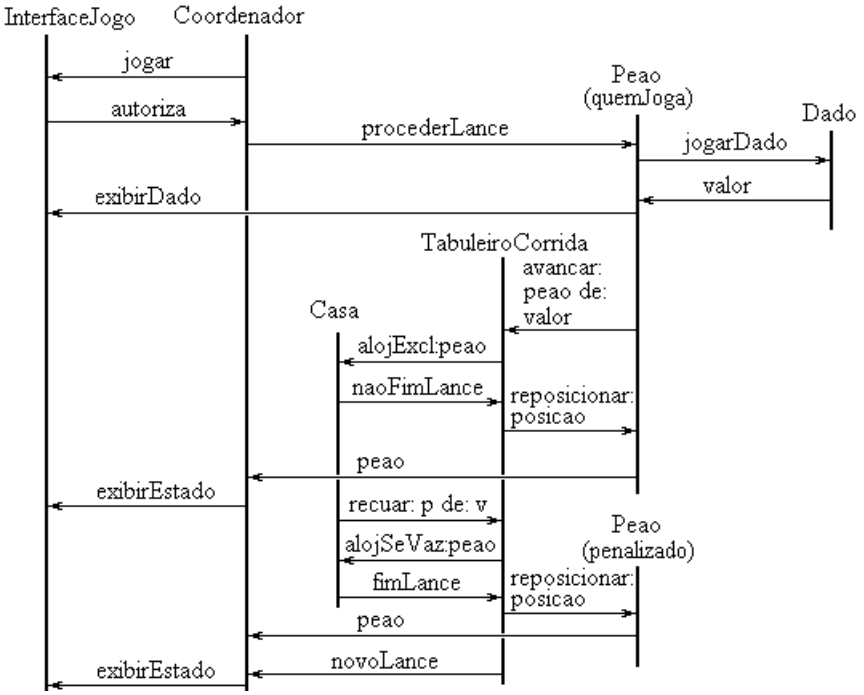


Figura 8.8 - desenvolvimento de um lance em uma partida - peão avança para casa simples, ocupada; peão penalizado recua para casa desocupada e simples

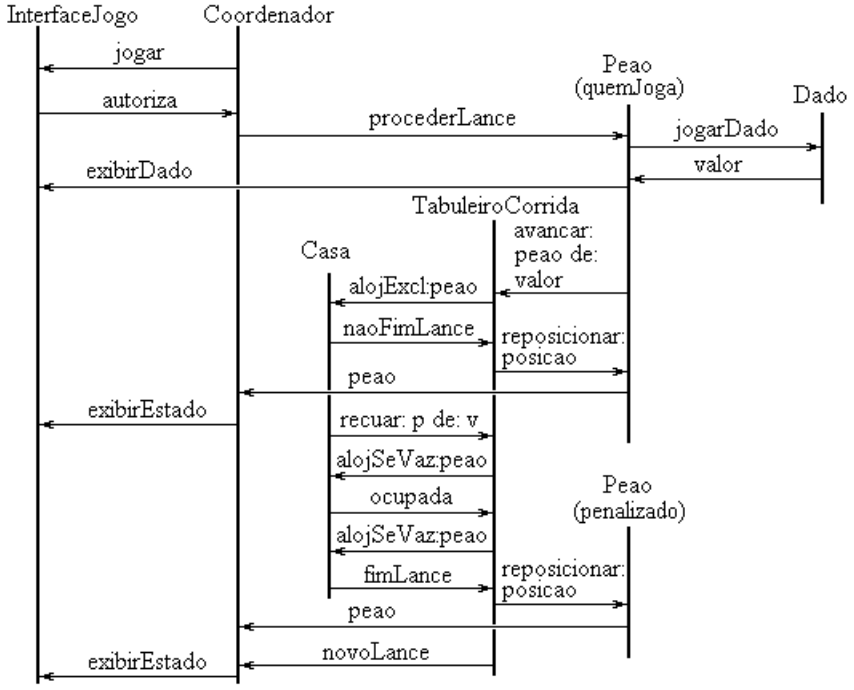


Figura 8.9 - desenvolvimento de um lance em uma partida - peão avança para casa simples, ocupada; peão penalizado recua inicialmente para casa ocupada, e após, para uma casa desocupada e simples

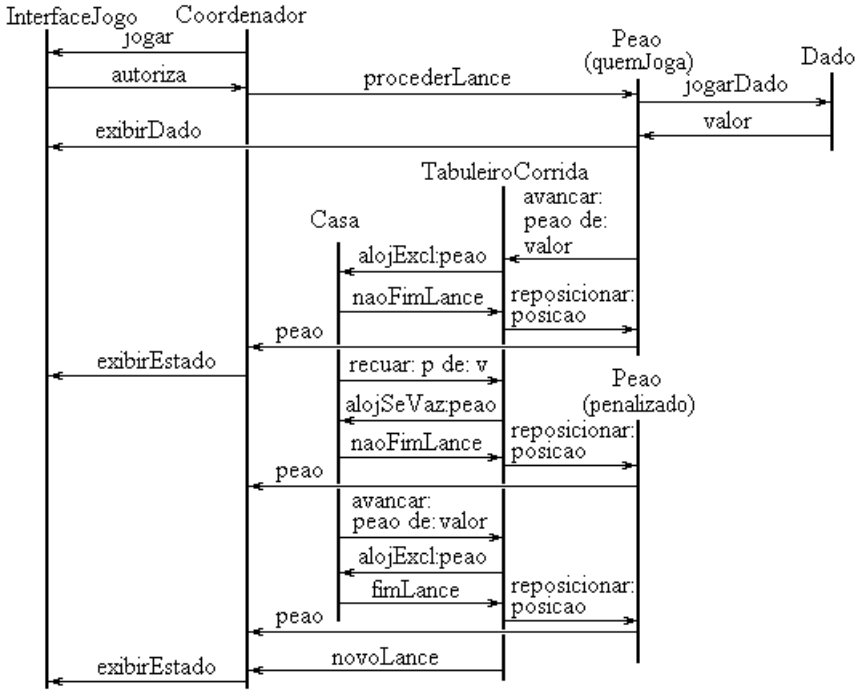


Figura 8.10 - desenvolvimento de um lance em uma partida - peão avança para casa simples, ocupada; peão penalizado recua para casa desocupada, com bonificação de avanço

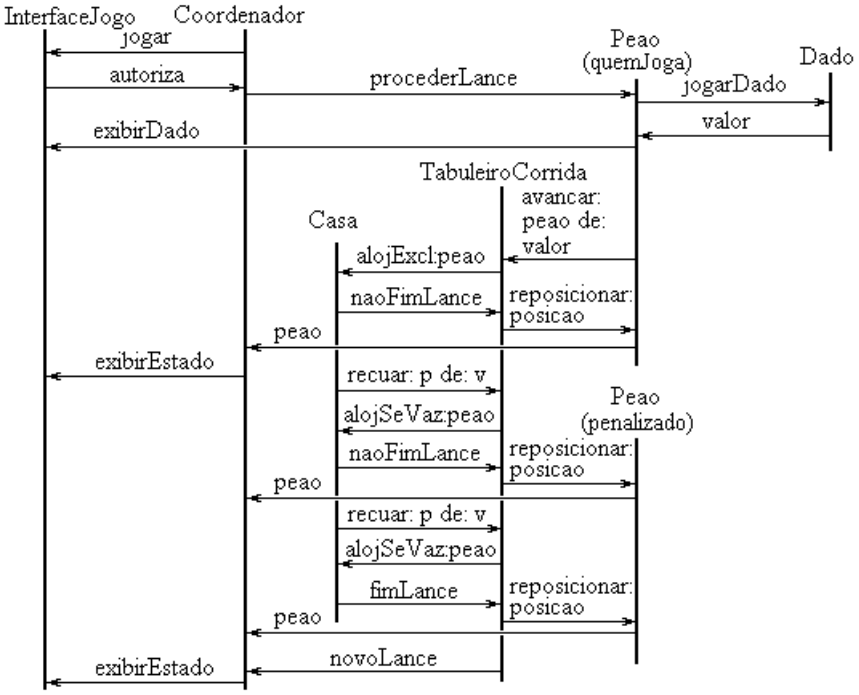


Figura 8.11 - desenvolvimento de um lance em uma partida - peão avança para casa simples, ocupada; peão penalizado recua para casa desocupada, com penalização de recuo

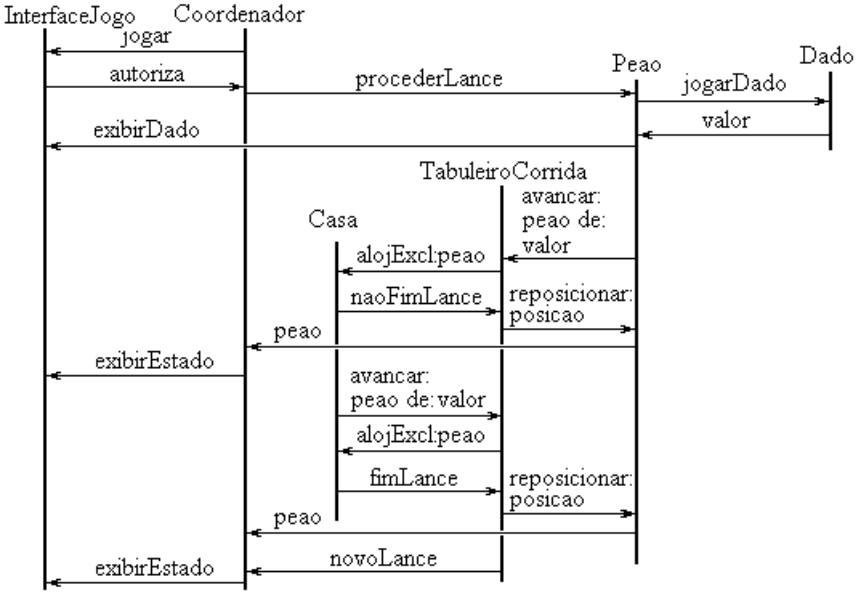


Figura 8.12 - desenvolvimento de um lance em uma partida - peão avança para casa com bonificação de avanço

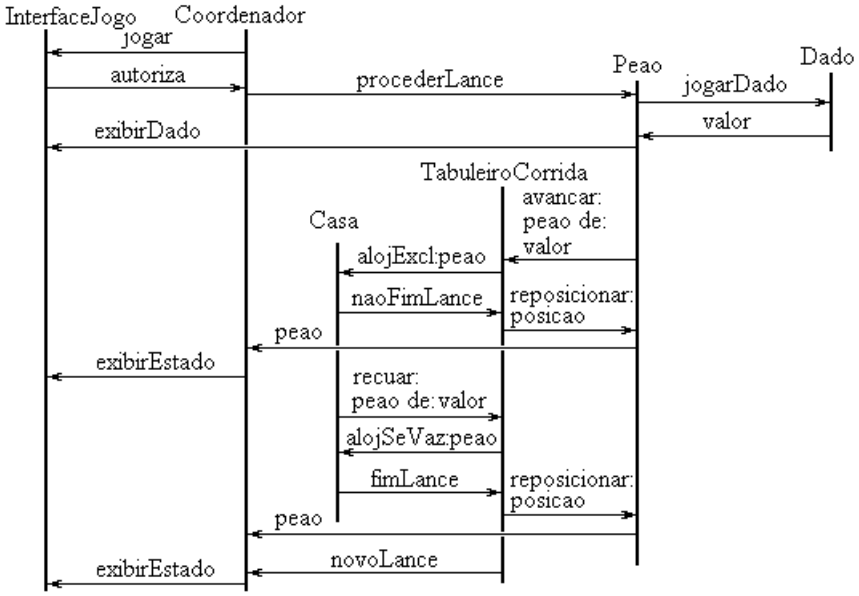


Figura 8.13 - desenvolvimento de um lance em uma partida - peão avança para casa com penalização de recuo

Neste exemplo do procedimento de lance, cabe salientar que a sintaxe dos diagramas de interação de OOSE permite representar as possibilidades de diferentes caminhos de uma forma menos fragmentada, como ocorre em OMT. Isto está ilustrado no diagrama apresentado abaixo. Por outro lado, a alternativa de concentrar em um único diagramatodos os cenários, pode levar a um diagrama de difícil leitura. É possível, porém, produzir uma especificação menos fragmentada que em OMT, sem recorrer a alternativa de produzir apenas um diagrama.

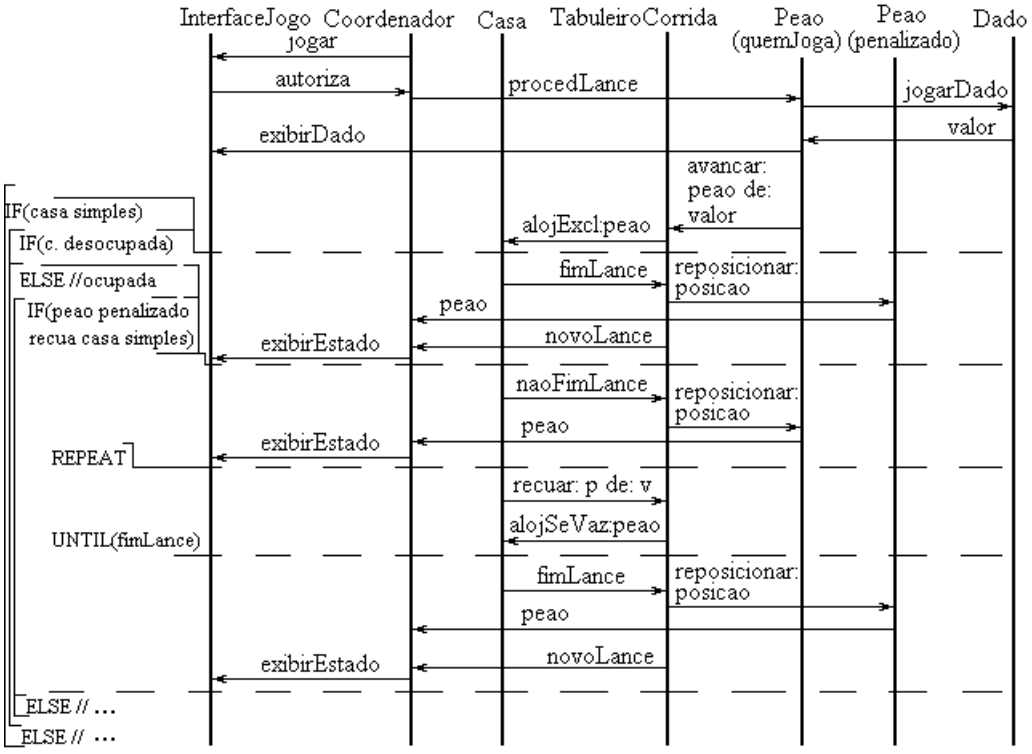


Figura 8.14 - desenvolvimento de um lance em uma partida - inclusão de pseudocódigo à esquerda do diagrama, como adotado em OOSE

O exemplo acima (modelagem parcial do jogo Corrida Plus) ilustra a diferença de capacidade das metodologias apresentadas em modelar o comportamento dinâmico de um sistema. Observa-se que as metodologias menos eficazes em modelar a interação dinâmica entre instâncias, serão menos eficientes na identificação dos métodos das classes, bem como na capacidade de registrar a dependência entre classes.

9 Frameworks

Até o capítulo anterior foram descritas e avaliadas metodologias de desenvolvimento de software orientadas a objetos voltadas ao desenvolvimento de aplicações específicas. Estas metodologias serão tratadas daqui em diante por metodologias OOAD[105].

No presente capítulo será apresentada a noção de framework, que constitui uma evolução em termos de reutilização no desenvolvimento de software - em relação ao desenvolvimento direto de aplicações, como tratado até aqui. Serão apresentadas e avaliadas técnicas de desenvolvimento de frameworks, e buscado o relacionamento entre suas características e as características das metodologias OOAD.

9.1 Níveis de reutilização de software

A reutilização de software - em contrapartida ao desenvolvimento de aplicações desde o início - é vista como um fator que pode levar ao aumento da produtividade da atividade de desenvolvimento de software, na medida em que o uso de elementos de software[106] já desenvolvidos e depurados, reduzirá o tempo de desenvolvimento, de testes e as possibilidades de introdução de erros ao longo do desenvolvimento.

A produção de elementos de software reutilizáveis se coloca como um requisito para que ocorra reutilização de software. O seguimento dos princípios que norteiam a modularidade, como a criação de módulos com interfaces pequenas, explícitas e em pequena quantidade, e o uso do princípio da ocultação de informação, tem sido ao longo dos últimos anos o principal elemento para a produção de bibliotecas de componentes reutilizáveis [MEY 88].

A reutilização de elementos de software pode ocorrer a nível de código, ou de projeto, sendo esta a forma mais importante [DEU 89]. A reutilização de código consiste na utilização direta de trechos de código já desenvolvido. A reutilização de projeto consiste no reaproveitamento de concepções arquitetônicas de uma aplicação em outra aplicação, não necessariamente com a utilização da mesma implementação.

A granularidade influencia na reutilização em termos de aumento da produtividade no desenvolvimento de software. Um programador que usa linguagem C, por exemplo se utiliza de bibliotecas de funções. Isto se classifica como reutilização de rotinas[107] [MEY 88]. A reutilização a nível de módulo corresponde a um nível superior que a reutilização de rotinas. A reutilização de classes em orientação a objetos segundo Meyer, corresponde à reutilização de módulo. Quando uma classe é reutilizada um conjunto de rotinas é reutilizado (métodos), bem como uma estrutura de dados (atributos). Uma classe que implementa uma estrutura de dados pilha, por exemplo, contém esta estrutura de dados e o conjunto de procedimentos para a sua manipulação - este nível de reutilização não é possível a rotinas. Reutilizar classes, portanto, tende a

---

105 Diferenciando em termos de nomenclatura, as metodologias de desenvolvimento de aplicações das metodologias de desenvolvimento de frameworks.

106 A expressão *elementos de software* é usada aqui de forma genérica, não se referindo necessariamente a código (podendo abranger os produtos da análise e do projeto).

107 A reutilização de rotina pode se dar a nível de projeto quando, por exemplo, um programador se utiliza do projeto de uma rotina já implementada para gerar uma implementação mais eficiente. A reutilização pode se dar a nível de projeto ou código (ou ambos) independente do nível de granularidade.

ser mais eficiente que reutilizar rotinas[108], e de mais alto nível de granularidade (uma classe de objetos tende a ser um elemento de software mais complexo que uma rotina isolada). Meyer em 1988 enfatizava a necessidade da disponibilidade de componentes genéricos e afirmava que "as classes de linguagens orientadas a objetos podem ser vistas como módulos de projeto bem como módulos de implementação" [MEY 88]. A reutilização de classes pode ocorrer sob dois enfoques diferentes:

- composição: consiste em usar as classes disponíveis em bibliotecas para originar os objetos da implementação;
- herança: consiste em aproveitar a concepção de classes disponíveis em bibliotecas, no desenvolvimento de outras, a partir de herança.

Uma característica comum à reutilização de rotinas e à reutilização de classes é que cabe a quem desenvolve a aplicação estabelecer a arquitetura da aplicação: que elementos compõem a aplicação e o fluxo de controle entre eles. Além disto, há em comum o fato da reutilização ser de componentes isolados, cabendo ao desenvolvedor estabelecer sua conexão ao sistema em desenvolvimento. Em um programa C, o programador determina a chamada de funções; em um programa Smalltalk o programador procede a interligação das classes (desenvolvidas e reutilizadas).

Um framework orientado a objetos[109] se situa num patamar de reutilização superior à reutilização de classes por reutilizar classes interligadas, ao invés de isoladas. Neste caso ocorre reutilização de código e projeto em um nível de granularidade superior às situações anteriores.

9.2 Frameworks orientados a objetos

Allen Wirfs-Brock [WIA 91] propõe a seguinte definição para framework: "um esqueleto de implementação de uma aplicação ou de um subsistema de aplicação, em um domínio de problema particular. É composto de classes abstratas e concretas e provê um modelo de interação ou colaboração entre as instâncias de classes definidas pelo framework. Um framework é utilizado através de configuração ou conexão de classes concretas e derivação de novas classes concretas a partir das classes abstratas do framework". Rebecca Wirfs-Brock e Ralph Jonhson [WIR 90] propõem a seguinte definição para framework: "uma coleção de classes concretas e abstratas e as interfaces entre elas, e é o projeto de um subsistema". Em outra publicação, Ralph Jonhson acrescenta: "não apenas classes, mas a forma como as instâncias das classes colaboram" [JOH 93].

A diferença fundamental entre um framework e a reutilização de classes de uma biblioteca, é que neste caso são usados componentes isolados, cabendo ao desenvolvedor estabelecer sua interligação, e no caso do framework, é procedida a reutilização de um conjunto de classes interrelacionadas - interrelacionamento estabelecido no projeto do framework. As figuras abaixo ilustram esta diferença. A figura 9.1 contém uma aplicação que reutiliza classes de uma biblioteca de classes - usando a notação de OMT. Em cinza, estão representadas as classes reutilizadas e em preto, o que é definido pelo desenvolvedor da aplicação, ou seja, outras classes e as associações entre classes. A figura 9.2 contém uma aplicação desenvolvida a partir de

---

108 Em termos de produtividade, no desenvolvimento de software.

109 As expressões *framework orientado a objetos* e *framework* serão consideradas sinônimos no presente texto.

um framework. A diferença entre os dois casos é que na aplicação da figura 9.1, o desenvolvedor deve estabelecer as associações entre classes e no caso da utilização do framework, como ilustrado na figura 9.2, as associações são predefinidas, cabendo ao desenvolvedor da aplicação conectar as classes desenvolvidas de uma forma predeterminada pelo framework (na figura 9.2 como na figura 9.1, está representado em cinza o que é predefinido e em preto, o que é acrescentado pelo desenvolvedor da aplicação).

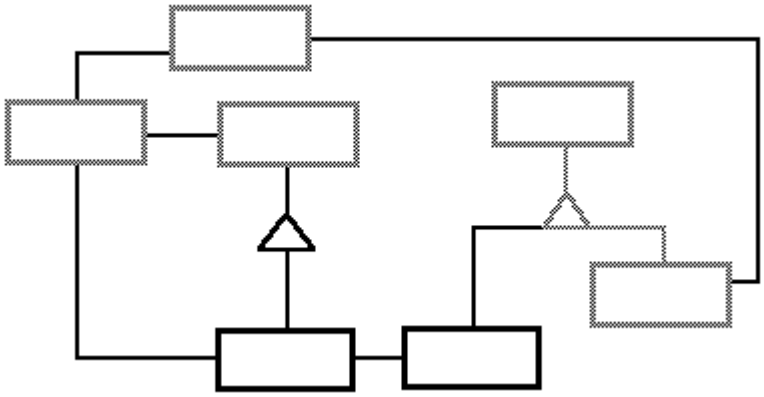


Figura 9.1 - uma aplicação orientada a objetos com reutilização de classes

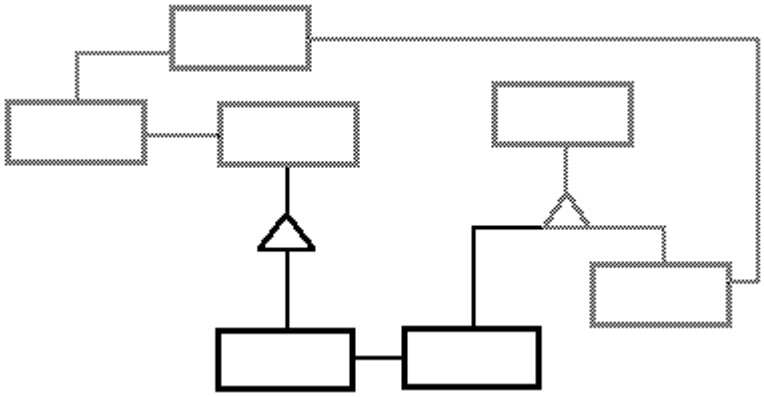


Figura 9.2 - uma aplicação orientada a objetos a partir de um framework

Dois aspectos caracterizam um framework [TAL 95]:

- Os frameworks fornecem infraestrtura e projeto: frameworks portam infraestrutura de projeto disponibilizada ao desenvolvedor da aplicação, que reduz a quantidade de código a ser desenvolvida, testada e depurada. As interconexões preestabelecidas definem a arquitetura da aplicação, liberando o desenvolvedor desta responsabilidade. O código escrito pelo desenvolvedor visa estender ou particularizar o comportamento do framework, de forma a moldá-lo a uma necessidade específica.
- Os frameworks "chamam", não são "chamados": um papel do framework é fornecer o fluxo de controle da aplicação. Assim, em tempo de execução, as instâncias das classes desenvolvidas esperam ser chamadas pelas instâncias das classes do framework.

Um framework se destina a gerar diferentes aplicações para um domínio. Precisa portanto, conter uma descrição dos conceitos deste domínio. As classes abstratas de um framework são os repositórios dos conceitos gerais do domínio de aplicação. No

contexto de um framework, um método de uma classe abstrata pode ser deixado propositalmente incompleto para que sua definição seja acabada na geração de uma aplicação. Apenas atributos que se aplicam a todas as aplicações de um domínio, são incluídos em classes abstratas.

Os frameworks são estruturas de classes interrelacionadas, que permitem não apenas reutilização de classes, mas minimizam o esforço para o desenvolvimento de aplicações - por conterem o protocolo de controle da aplicação (a definição da arquitetura), liberando o desenvolvedor de software desta preocupação. Os frameworks invertem a ótica do reuso de classes, da abordagem bottom-up[110] para a abordagem top-down: o desenvolvimento inicia com o entendimento do sistema contido no projeto do framework, e segue no detalhamento das particularidades da aplicação específica, o que é definido pelo usuário do framework. Assim, a implementação de uma aplicação a partir do framework é feita pela adaptação de sua estrutura de classes, fazendo com que esta inclua as particularidades da aplicação [TAL 95].

Um outro aspecto a considerar é o nível de granularidade de um framework. Frameworks podem agrupar diferentes quantidades de classes, sendo assim, mais ou menos complexos. Além disto, podem conter o projeto genérico completo para um domínio de aplicação, ou construções de alto nível que solucionam situações comuns em projetos. Deutsch admite framework de uma única classe (com esta nomenclatura) que consiste de "uma classe que fornece especificação e implementação parcial mas que necessita de subclasses ou parâmetros para completar a implementação"[111]. Frameworks com mais de uma classe são denominados frameworks de múltiplas classes [DEU 89].

Segundo a granularidade, Pree denomina frameworks de aplicação aqueles que constituem um projeto genérico para um domínio, e simplesmente frameworks aqueles que representam uma microarquitetura consistindo de poucos componentes, a ser usada como parte de um projeto [PRE 95]. Em [GAM 94] é adotada uma classificação semelhante, porém com a denominação framework quando se trata do projeto genérico para um domínio, e design pattern no caso de uma microarquitetura. Ainda são apresentadas as seguintes diferenças entre frameworks e design patterns: design patterns são mais abstratos que frameworks, design patterns são menores (menos elementos arquitetônicos) que frameworks, design patterns são menos especializados que frameworks. No presente trabalho a expressão framework será usada independente de granularidade, porém, sempre supondo que contenha mais de uma classe.

9.3 Aspectos da geração de aplicações a partir do framework

De acordo com a forma como deve ser utilizado, um framework pode se classificar como dirigido a arquitetura ou dirigido a dados. No primeiro caso a aplicação deve ser gerada a partir da criação de subclasses das classes do framework. No segundo caso, diferentes aplicações são produzidas a partir de diferentes combinações de objetos, instâncias das classes presentes no framework. Frameworks dirigidos a arquitetura são mais difíceis de usar, pois, para a geração das subclasses exigem um profundo conhecimento do projeto do framework, bem como um esforço no desenvolvimento de

---

110 Desenvolvimento bottom-up é o que ocorre quando classes de objetos de uma biblioteca são interligadas, para a contrução de uma aplicação.

111 Ao longo do presente trabalho a expressão framework será usada sempre pressupondo mais de uma classe (com interligações), ficando porém, registrada esta classificação de Deutsch.

código. Frameworks dirigidos a dado são mais fáceis de usar, porém são menos flexíveis. Uma outra abordagem seria uma combinação dos dois casos, onde o framework apresentaria uma base dirigida a arquitetura e uma camada dirigida a dado. Com isto, possibilita a geração de aplicações a partir da combinação de objetos, mas permite a geração de subclasses [TAL 94]. Ralph Jonhson adota esta classificação de frameworks, porém usa a denominação caixa branca para o framework dirigido a arquitetura e caixa preta para o dirigido a dado [JOH 92].

Há portanto, duas maneiras básicas de gerar aplicações a partir de um framework: a partir de combinações de instâncias das classes concretas do framework, ou a partir da definição de classes - ou ainda a partir de combinações das duas possibilidades. A possibilidade de usar uma ou outra forma é uma característica particular de cada framework, definida em seu projeto. O projeto do framework estabelece a flexibilidade possível, impondo uma limitação (de combinação de objetos ou de criação de novas classes) a ser obedecida pelo usuário. Um framework deve dispor de uma documentação específica para ensinar o usuário a gerar aplicações, estabelecendo estas limitações - o que é diferente da documentação de projeto.

Na customização do framework a partir da criação de classes fica imposto um conjunto de classes que deve ser definido pelo usuário (subclasses das classes do framework). Para a criação de um editor de hipertexto a partir do framework ET++, por exemplo, o usuário necessita obrigatoriamente gerar, dentre outras, uma subclasse de Application, que conterá a sobreposição do método DoMakeManager (que dispara a execução da aplicação), e uma subclasse de Document, que conterá a definição do documento [PRE 95]. Também cabe ressaltar que além de um conjunto obrigatório de subclasses, o usuário não é totalmente livre para criar ou alterar funcionalidades a partir da criação de classes, uma vez que a posse do controle da aplicação está com as classes do framework e não com as classes do usuário. Por exemplo, um framework que gera aplicações para um domínio específico pode definir uma interface com o usuário padronizada (padrão Microsoft-Windows, ou algum outro) e não dar ao usuário a possibilidade de alterar isto.

Em termos ideais, um framework deve abranger todos os conceitos gerais de um domínio de aplicação, deixando apenas aspectos particulares para serem definidos nas aplicações específicas. Neste caso, na geração de aplicações, o usuário do framework não deve precisar criar classes que não sejam subclasses de classes abstratas do framework. Se isto for alcançado, o framework terá conseguido de fato ser uma generalização do domínio.

9.4 Metodologias de desenvolvimento de frameworks

Assim como existem propostas de metodologias de desenvolvimento de aplicações orientadas a objetos (como aquelas tratadas nos capítulos anteriores), também existem propostas de metodologias para o desenvolvimento de frameworks. Este capítulo se aterá a três metodologias: projeto dirigido por exemplo (example-driven design) [JOH 93], projeto dirigido por hot spot (hot spot[112] driven design) [PRE 95] e a metodologia de projeto da empresa Taligent [TAL 94].

---

112 A expressão hot-spot foi mantida em Inglês e significa "parte importante", significado este não obtido a partir de sua tradução literal.

Cada uma das metodologias OOAD descritas nos capítulos anteriores é composta por um conjunto de técnicas de modelagem e um processo de desenvolvimento. As três metodologias de desenvolvimento de frameworks que serão descritas a seguir, descrevem como produzir frameworks sem se ater a técnicas de modelagem - de fato, não amarrando as metodologias a técnicas de modelagem específicas, como fazem as metodologias OOAD. Assim, fica subentendido que um procedimento concreto de desenvolvimento de framework, pela necessidade de documentação formal (ou semiformal), deve adotar uma OOAD como base de trabalho, alterando seus procedimentos em função das especificidades do desenvolvimento de um framework - conforme estabelecem as metodologias de desenvolvimento de frameworks. Com isto, as metodologias OOAD e de desenvolvimento de frameworks se completam - o que justifica o tratamento simultâneo dos dois casos, como feito no presente trabalho.

9.4.1 Projeto dirigido por exemplo [JOH 93]

Johnson estabelece que o desenvolvimento de um framework para um domínio de aplicação é decorrente de um processo de aprendizado a respeito deste domínio, que se processa concretamente a partir do desenvolvimento de aplicações ou do estudo de aplicações desenvolvidas. Porque as pessoas pensam de forma concreta, ao invés de abstrata, a abstração do domínio - que é o próprio framework - é obtida a partir da generalização de situações concretas - as aplicações. Abstrações são obtidas de uma forma bottom-up, a partir do exame de exemplos concretos: aspectos semelhantes de diferentes aplicações podem dar origem a classes abstratas que agrupam as semelhanças, cabendo às classes concretas do nível hierárquico inferior a especialização para satisfazer cada caso.

O processo de generalização ocorre a partir da busca de elementos que recebem nomes diferentes, mas são a mesma coisa; recorrendo à parametrização para eliminar diferenças; particionando elementos para tentar obter componentes similares; agrupando elementos similares em classes.

O processo de desenvolvimento segundo o projeto dirigido por exemplo, atravessa as etapas de análise, projeto e teste. A forma tida como ideal para desenvolver um framework (por avaliar um número considerado adequado de exemplos) é:

1 - Análise do domínio:

- aprender as abstrações já conhecidas;
- coletar exemplos de programas que poderiam ser desenvolvidos a partir do framework (no mínimo, quatro);
- avaliar a adequação de cada exemplo.

2 - Projetar uma hierarquia de classes que possa ser especializada para abranger os exemplos (um framework) - nesta etapa o autor recomenda a utilização de design patterns[113];

---

113 Os design patterns [GAM 95] e os metapatterns [PRE 95] são microarquiteturas (pequeno conjunto de classes) que representam soluções de situações comuns em projetos. Há um número destas microarquiteturas definidos na literatura e disponíveis para utilização. Sua utilidade é reconhecida tanto no desenvolvimento como na documentação de projetos. O presente trabalho não se aprofundará no tema, se limitando a citar que o uso deste recurso deve levar - como afirma o autor da metodologia sob análise - a um resultado mais satisfatório em termos de desenvolvimento de frameworks.

3 - Testar o framework usando-o para desenvolver os exemplos (implementar, testar e avaliar cada exemplo usado na primeira etapa, utilizando para isto, o framework desenvolvido).

Observe-se que o passo 2 corresponde à aplicação de uma metodologia OOAD, porém partindo de subsídios de análise, e visando não uma aplicação particular, mas um framework.

O autor afirma que este caminho ideal nunca é seguido, por limitações de ordem econômica ou de tempo:

- a análise adequada de exemplos particulares demanda tempo e esforço, o que implica em custo;
- na medida em que os softwares - potenciais exemplos do domínio - funcionam, não há incentivo financeiro para convertê-los em um novo software (que seriam gerados a partir do framework);
- o esforço de análise acaba sendo mais voltado ao software tido como mais importante.

Em função da dificuldade prática de analisar um mínimo de quatro aplicações já desenvolvidas para então começar a desenvolver o framework, é estabelecido um procedimento tido como adequado (e economicamente viável) para o desenvolvimento de um framework: a partir de duas aplicações similares que se necessite desenvolver, proceder paralelamente o desenvolvimento do framework e das duas aplicações, procurando maximizar a troca de informações entre os três desenvolvimentos. Segundo o autor, para que este procedimento possa ser realizado, as equipes de desenvolvimento devem conter pessoas com experiência de aplicações do domínio tratado - como uma forma de capturar informações de outras aplicações.

Procedendo uma comparação preliminar entre esta primeira metodologia de desenvolvimento de frameworks e as metodologias OOAD descritas nos capítulos anteriores, observa-se que o projeto dirigido por exemplo carece de um conjunto de técnicas de modelagem e de um processo de desenvolvimento detalhado - de fato, demanda uma metodologia OOAD para gerar um framework. Uma vez que as metodologias OOAD não são voltadas ao desenvolvimento de frameworks, a metodologia OOAD adotada deve ser adaptada a esta situação: não se faria por exemplo, a união de OMT com projeto dirigido por exemplo, mas a adaptação dos procedimentos de OMT aos passos do projeto dirigido por exemplo[114].

9.4.2 Projeto dirigido por hot spot [PRE 95]

Uma aplicação orientada a objetos é completamente definida. Um framework, ao contrário possui partes propositalmente não definidas - o que lhe dá a capacidade de ser flexível e se moldar a diferentes aplicações. Os hot spots são as partes do framework mantidas flexíveis. A essência da metodologia é identificar os hot spots na estrutura de classes de um domínio, e a partir disto construir o framework. Pree propõe a seguinte seqüência de procedimentos:

---

114 Também seria necessária uma adaptação dos modelos adotados, mas esta discussão será desenvolvida após a apresentação do conjunto de metodologias de desenvolvimento de frameworks.

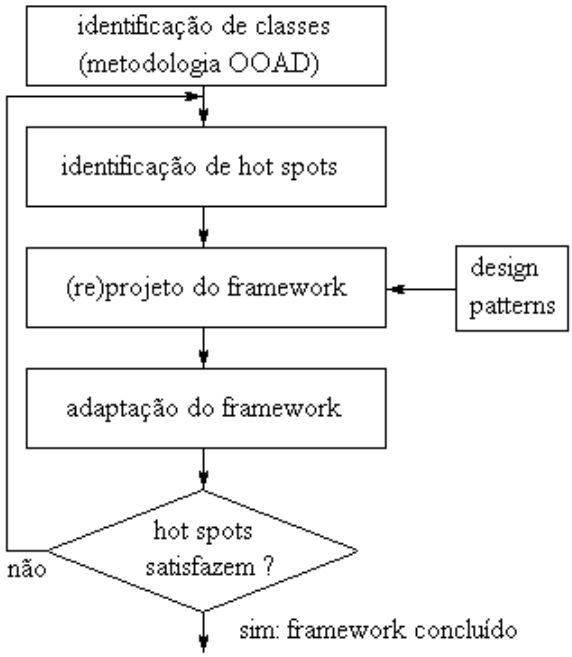


Figura 9.3 - as etapas do projeto dirigido por hot spot para o desenvolvimento de um framework - extraído de [PRE 95]

A primeira etapa, a identificação de classes, demanda uma metodologia OOAD. O desenvolvedor do framework, a partir de informações de especialistas do domínio, define uma estrutura de classes - semelhante ao que é feito no desenvolvimento de uma aplicação orientada a objetos.

Na segunda etapa, também com o auxílio de especialistas do domínio, são identificados os hot spots. É preciso identificar que aspectos diferem de aplicação para aplicação, e definir o grau de flexibilidade que deve ser mantido em cada caso. Quando dados são os aspectos que diferem entre aplicações, a conseqüência é que classes abstratas agruparão os dados (atributos) comuns e classes concretas, específicas para aplicações, agruparão os dados específicos. Quando funções são os aspectos que diferem entre aplicações, a conseqüência é a presença nas classes além dos métodos base (métodos completamente definidos), de métodos abstratos (em classes abstratas) que definem apenas o protocolo do método, métodos template, que chamam outros métodos (hook) para completar seu processamento e métodos hook, que flexibilizam o comportamento dos métodos template. O grau de flexibilidade definirá a necessidade ou não de flexibilidade em tempo de execução.

Hot spots são documentados com cartões hot spot. A ilustração abaixo apresenta o formato proposto para estes cartões.

Nome do hot spot	☐ flexibilidade em tempo de execução
descrição geral da semântica	
comportamento do hot spot em pelo menos duas situações específicas	

Figura 9.4 - formato de um cartão hot spot

Cálculo de taxa	☑ flexibilidade em tempo de execução
cálculo de taxa a ser cobrada quando itens alugáveis são retornados; o cálculo é baseado em parâmetros específicos da aplicação	
sistema de hotel: o cálculo resulta de taxa do quarto * número de noites + chamadas telefônicas + consumo do frigobar sistema de aluguel de carros: o cálculo resulta de taxa do modelo de carro * numero de dias + taxa por quilômetro * (quilometragem rodada - quilometragem livre) + valor do reabastecimento	

Figura 9.5 - exemplo de um cartão hot spot

A terceira etapa, o projeto do framework (ou a reelaboração do projeto), ocorre após a identificação dos hot spots. Consiste em modificar a estrutura de classes inicialmente definida, de modo a comportar a flexibilidade requerida. Nesta etapa se definirá o que o usuário do framework deve fazer para gerar uma aplicação - que subclasses deve definir, a que classes deve fornecer parâmetros para a criação de objetos, quais as combinações de objetos possíveis.

A quarta etapa consiste num refinamento da estrutura do framework a partir de novas intervenções de especialistas do domínio. Se após isto o framework for avaliado como satisfatório, está concluída uma versão do framework (a questão é: os hot spots definidos dão ao framework o grau de flexibilidade necessário para gerar as aplicações requeridas ?), caso contrário, retorna-se à etapa de identificação dos hot spots.

Segundo Pree, na etapa de projeto o desenvolvimento do framework é centrado em design patterns. A aplicação de design patterns por parte do desenvolvedor é fundamental para a flexibilidade do framework - não apenas para gerar aplicações, mas para futuras alterações no framework.

9.4.3 Metodologia de projeto da empresa Taligent [TAL 94]

A metodologia proposta pela empresa Taligent difere das anteriores pelo conjunto de princípios que norteia o desenvolvimento de frameworks. Primeiramente, a visão de desenvolver um framework que cubra as características e necessidades de um domínio é substituída pela visão de possuir um conjunto de frameworks estruturalmente menores e mais simples, que usados conjuntamente, darão origem às aplicações. A justificativa para isto é que "pequenos frameworks são mais flexíveis e podem ser reutilizados mais freqüentemente". Assim, a ênfase passa a ser o desenvolvimento de frameworks pequenos e direcionados a aspectos específicos do domínio.

Um outro aspecto é que uma das linhas mestras do processo de desenvolvimento é tornar o uso do framework o mais simples possível, através da minimização da quantidade de código que o usuário deve produzir, através de:

- disponibilidade de implementações (classes) concretas que possam ser usadas diretamente;
- minimização do número de classes que devem ser derivadas;
- minimização do número de métodos que devem ser sobrepostos.

A metodologia propõe a seqüência de quatro passos, apresentados a seguir.

Identificar e caracterizar o domínio do problema:
- analisar o domínio e identificar os frameworks necessários (para o desenvolvimento de aplicações), o que pode incluir frameworks desenvolvidos e a desenvolver;
- examinar soluções existentes;
- identificar as abstrações principais;
- identificar o limite de responsabilidades do framework em desenvolvimento (considerando que se esteja no processo de desenvolvimento de um framework específico);
- validar estas informações com especialistas do domínio.

Definir a arquitetura e o projeto:
- refinar a estrutura de classes obtida no passo anterior, centrando atenção em como os usuários interagem com o framework:
  - que classes o usuário instancia ?
  - que classes o usuário deriva (e que métodos sobrepõe) ?
  - que métodos o usuário chama ?
- aperfeiçoar o projeto com o uso de design patterns;
- validar estas informações com especialistas do domínio.

Implementar o framework:
- implementar as classes principais;
- testar o framework (gerando aplicações);
- solicitar a terceiros o procedimento de teste
- iterar para refinar o projeto.

Desdobrar o framework:
- providenciar documentação na forma de diagramas, receitas (como usar o framework para desenvolver determinada aplicação) e exemplos de aplicações simples (incluindo o código da implementação, que complementa o framework);
- manter e atualizar o framework, seguindo as seguintes regras:
  - corrigir erros imediatamente;
  - adicionar novas características ocasionalmente;
  - mudar interfaces tão infreqüentemente quanto possível ("é melhor adicionar novas classes que alterar a hierarquia de classes existente, ou adicionar novos métodos que alterar ou remover métodos existentes").

9.5 Análise comparativa das metodologias de desenvolvimento de frameworks

9.5.1 Aspectos em comum

As três metodologias apresentadas possuem uma série de características comuns. Uma primeira a destacar é a busca de informações do domínio de aplicação em aplicações já elaboradas - o projeto dirigido a exemplo preconiza a busca das abstrações do domínio fundamentalmente a partir da utilização desta fonte. Cabe ressaltar que basear-se em aplicações existentes para construir um modelo do sistema durante o

processo de análise, não é uma exclusividade das metodologias de desenvolvimento de frameworks. É um procedimento também adotado por metodologias OOAD ([COA 92], por exemplo).

Nenhuma das três metodologias de desenvolvimento de frameworks adotou técnicas de modelagem para a descrição do framework. Um framework assim como uma aplicação desenvolvida segundo a orientação a objetos, carece de descrições estática e dinâmica tanto a nível de classes separadamente, como a nível de framework, como discutido no capítulo 8. Assim, fica subentendido que um procedimento de desenvolvimento de frameworks, pela necessidade de documentação, deve adotar uma metodologia OOAD como base de trabalho, alterando seus procedimentos em função das especificidades do desenvolvimento de frameworks. Com isto, as metodologias OOAD podem vir a complementar as metodologias de desenvolvimento de frameworks. Quanto às técnicas de modelagem, devem ser adaptadas aquelas que procedem descrição do sistema como um todo, nos seguintes aspectos:

- o modelo de objetos deve mostrar as classes a serem acrescentadas pelo usuário do framework para o desenvolvimento de uma aplicação (conjunto mínimo necessário, todas as classes possíveis);
- o modelo dinâmico quando descreve a interação entre instâncias, deve destacar que instâncias são de classes a serem criadas, qual o papel destas classes na definição de uma aplicação, bem como definir que métodos de classes abstratas precisam ser definidos nas classes a serem criadas (e que métodos podem ser redefinidos).

Os procedimentos de construção de modelos de metodologias OOAD de um modo geral, não demandam alteração, pois em uma etapa do processo de desenvolvimento, as três metodologias de desenvolvimento de frameworks determinam a construção de uma especificação completa, o que consiste em seguir os procedimentos da metodologia OOAD adotada. A única ressalva a fazer é que a análise é voltada a produzir uma descrição geral do domínio, e é norteada pela metodologia de desenvolvimento de frameworks - o procedimento de análise do domínio é um aspecto de grande relevância em uma metodologia voltada a frameworks.

Um outro ponto comum é a recomendação do uso de patterns. Patterns representam o registro da experiência em projeto. É unânime entre os autores das metodologias apresentadas que a não utilização de patterns prejudicará o desenvolvimento, tanto em termos de dispender mais tempo para chegar à solução de problemas de projeto, como em termos de não dispor da solução mais adequada. Também é uma característica comum, que nenhuma das três metodologias estabelece especificamente, como utilizar patterns.

O teste do framework produzido necessariamente passa pelo desenvolvimento de aplicações. Todas as metodologias demandam este requisito para o encerramento do processo de desenvolvimento.

9.5.2 Aspectos particulares

As metodologias apresentadas possuem muitos pontos comuns. Os aspectos em que diferem são apenas quanto à ênfase a algum aspecto particular ao longo do desenvolvimento, e não pontos conflitantes.

O que diferencia o projeto dirigido por exemplo das demais metodologias é a ênfase à análise do maior número possível de aplicações já desenvolvidas. Idealmente, é do conjunto de aplicações que se origina a descrição do domínio (transposta para as

classes abstratas), assim como é nas diferenças entre as aplicações que se identificam as flexibilidades necessárias ao framework.

A característica principal do projeto dirigido por hot spots é a ênfase sobre os pontos de flexibilidade do framework, ou seja, os hot spots. Procedida a descrição do domínio, são buscados e documentados os hot spots; o desenvolvimento consiste em adaptar a estrutura de classes aos hot spots identificados. Se após um ciclo de desenvolvimento o framework for julgado inadequado, a atenção volta-se à redefinição dos hot spots.

Os aspectos característicos da metodologia proposta pela Taligent, que a diferenciam das demais, são os princípios para o desenvolvimento de frameworks:

- substituição da visão de desenvolver um framework que cubra as características e necessidades de um domínio, pela visão de desenvolver um conjunto de frameworks estruturalmente menores e mais simples que usados conjuntamente, darão origem às aplicações;
- preocupação ao longo do desenvolvimento em tornar o uso do framework o mais simples possível, através da minimização da quantidade de código que o usuário deve produzir.

Estas características das três metodologias não são mutuamente exclusivas:

- todas podem comportar a busca de informações do domínio em aplicações já desenvolvidas, como apregoado pelo projeto dirigido por exemplo;
- todas podem comportar a ênfase à busca de hot spots na estrutura de classes, do projeto dirigido por hot spots;
- todas podem comportar a ênfase em minimizar a quantidade de código que o usuário deve desenvolver, como proposto pela Taligent;
- desenvolver frameworks extensos que cubram todas as necessidades das aplicações de um domínio ou desenvolver pequenos frameworks que devem ser usados em conjunto para a geração de aplicações, como propõe a Taligent, é uma decisão de projeto - pode ser adotada por qualquer metodologia.

10 Uso de técnicas de modelagem para o desenvolvimento de frameworks

O que se propõe a seguir é uma aglutinação de técnicas de modelagem julgadas mais adequadas, dentro do conjunto de metodologias descrito nos capítulos anteriores. Este conjunto apresenta limitações que são discutidas no final do capítulo.

Alguns princípios nortearam a definição do conjunto de técnicas de modelagem proposto. O primeiro deles foi que o mesmo conjunto de técnicas de modelagem da análise é usado em projeto - acrescendo uma técnica de modelagem no projeto, para detalhar o algoritmo dos métodos. Assim, o produto da análise está sujeito a alterações ao longo do projeto, facilitando a evolução iterativa da estrutura de classes do framework, ao longo de seu desenvolvimento. Além disto, não é estabelecida uma fronteira rígida entre as duas etapas. Considera-se que a análise deve produzir uma descrição do framework em termos da estrutura de classes, com atributos e métodos (estes apenas assinatura), e que o projeto deve elaborar os algoritmos dos métodos e refinar as estruturas dos atributos, completando a descrição e produzindo a solução computacional a ser implementada.

Um segundo princípio seguido, é que a modelagem deve abranger as visões estática e dinâmica, havendo assim técnicas de modelagem voltados para cada uma. Estas visões podem ser direcionadas ao sistema como um todo (conjunto de classes interrelacionadas) ou às classes separadamente. O conjunto de técnicas de modelagem adotado cobre todos estes aspectos.

10.1 Técnicas de modelagem para especificação de frameworks

Uma especificação construída sob a ótica da orientação a objetos é a descrição de um conjunto de classes - as classes individualmente e suas interdependências. Isto se aplica tanto a especificação de uma aplicação quanto à descrição de um framework. Conforme as considerações apresentadas no capítulo oito, a especificação de um sistema deve abranger uma descrição estática, que apresente suas estruturas, e uma descrição dinâmica, que descreva a operação do sistema. A descrição dinâmica apresenta dois aspectos: a descrição comportamental, que se preocupa com o fluxo de controle do sistema, ou seja, a seqüência de execução, e a descrição funcional, que se preocupa em descrever os fluxos de dados e suas transformações entre unidades funcionais. Em um sistema baseado em orientação a objetos as modelagens estática e dinâmica devem ter a capacidade de descrever as classes individualmente e o sistema como um todo - caso este que consiste em descrever o interrelacionamento entre as classes.

O conjunto de técnicas de modelagem de uma metodologia OOAD deve abranger todos estes aspectos. O mesmo se aplica ao conjunto de técnicas de modelagem para a descrição de um framework. A rigor, os técnicas de modelagem de uma metodologia OOAD são adequados à descrição de um framework, consideradas as ressalvas do capítulo anterior. Podem ser acrescentadas informações adicionais ao conjunto de técnicas de modelagem da metodologia OOAD, na forma de outros técnicas de modelagem, para melhor caracterizar as particularidades do framework, como os

cartões hot spot, da metodologia dirigida por hot spot, ou o uso de metapatterns ou design patterns para destacar detalhes da dependência entre classes[115].

A partir destas considerações e da avaliação de metodologias OOAD procedida no capítulo oito (de técnicas de modelagem e processo de desenvolvimento), propõe-se um conjunto de técnicas de modelagem para a especificação de um framework, apresentado a seguir.

10.1.1 Modelagem estática

As classes abstratas de um framework são repositórios de conceitos gerais do domínio de aplicação. Além disto, a geração de aplicações a partir de um framework pode ser baseada em criação de subclasses de classes abstratas. Com isto, é importante que em um modelo de objetos de um framework haja distinção gráfica entre classes abstratas e classes concretas.

O projeto de um framework estabelece a flexibilidade permitida ao usuário para a geração de aplicações. Assim, a representação de classes deve distinguir graficamente as classes que devem (ou que podem) ser acrescentadas pelo usuário, para o desenvolvimento de uma aplicação.

A notação de modelo de objetos de metodologias OOAD nem sempre adota a distinção entre classe abstrata e concreta (das cinco metodologias OOAD apresentadas, apenas Fusion e a metodologia de Coad e Yourdon adotam a distinção). Além disto, não existe o conceito de *classe a acrescentar* no contexto do desenvolvimento de aplicações orientadas a objetos - não sendo portanto, tratado pelas metodologias OOAD. Estes fatores forçam que se adapte a notação do modelo de objetos das metodologias OOAD, para que estas possam ser usadas no desenvolvimento de frameworks.

A figura 10.1 sugere uma notação para a diferenciação entre as classes abstratas do framework, classes concretas do framework e classes a serem criadas pelo usuário do framework na geração de aplicações.

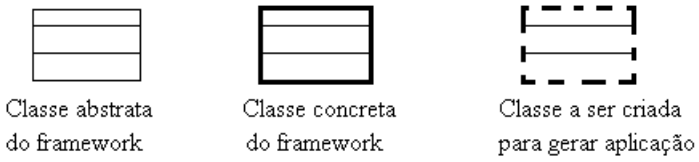


Figura 10.1 - distinção gráfica das classes de um framework

Esta diferenciação na notação além de mostrar onde é possível a extensão da estrutura de classes, permite distinguir a obrigatoriedade ou não da criação de classes - e no caso da obrigatoriedade, se é possível a inclusão de mais de uma classe. A criação de classes por parte do usuário (subclasses de classes do framework), poderá se enquadrar em uma das situações apresentadas na figura 10.2. No primeiro caso o projeto prevê que o usuário do framework deve produzir exatamente uma subclasse de uma classe abstrata - a estrutura de controle projetada para o framework não comporta mais de uma

---

115 O uso de metapatterns ou design patterns como elementos de descrição do projeto, consiste basicamente em destacar uma parte do modelo de objetos e descrever como o respectivo padrão de objeto se aplica àquele caso (em forma textual). É uma informação complementar que facilita o entendimento do framework para quem conhece o padrão usado - é uma descrição por semelhança com um caso conhecido. O uso de metapatterns ou design patterns não será tratado no presente trabalho.

subclasse. No segundo caso o usuário do framework deve produzir pelo menos uma subclasse, mas tem a possibilidade de gerar mais subclasses da classe abstrata. O terceiro caso é parecido com o segundo, exceto pelo fato do framework incluir na estrutura, uma ou mais subclasses concretas. Assim, deixa de existir a obrigatoriedade do usuário produzir uma subclasse.

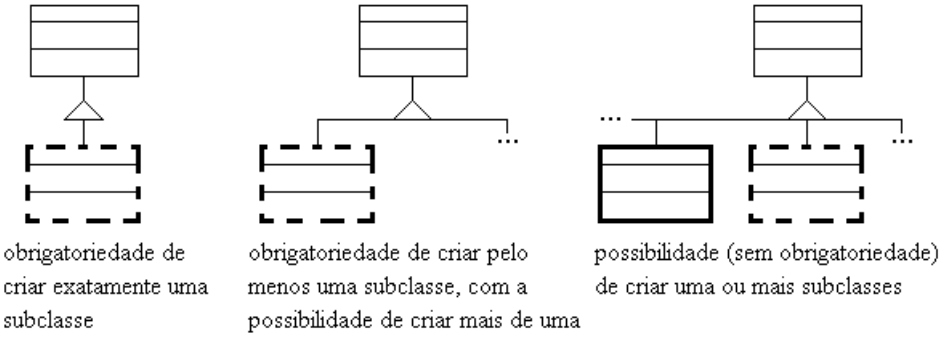


Figura 10.2 - possíveis situações das subclasses criadas pelo usuário do framework

Um método de uma classe abstrata pode ser classificado como [JOH 91]:

- abstrato: um método que não é completamente definido na classe abstrata; deve ser definido em uma subclasse;
- template: é definido em termos de outros métodos (métodos hook);
- base: é completamente definido.

Na execução de um método template ocorre a chamada de pelo menos um método hook. O método hook por sua vez, pode ser um método abstrato, base ou template [PRE 95].

No contexto de um framework, um método de uma classe abstrata pode ser deixado propositalmente incompleto para que sua definição seja acabada na geração de uma aplicação. Este é o caso por exemplo, de um método abstrato pertencente a uma classe abstrata do framework, que deve ser sobreposto na subclasse gerada pelo usuário, no desenvolvimento de uma aplicação. A notação que relaciona os métodos[116] de uma classe abstrata de um framework deve explicitar esta classificação (abstrato, template ou base), pois é uma informação relevante do projeto: aspectos de implementação mantidos flexíveis, e que devem ser definidos na geração de aplicações. As metodologias OOAD não possuem uma notação para esta classificação por não ser um aspecto relevante no contexto do desenvolvimento de aplicações específicas.

O exemplo abaixo apresenta uma parte do modelo de objetos de um framework hipotético para a visualização de hiperdocumentos. Os nomes de associação, relação de atributos e métodos foram omitidos.

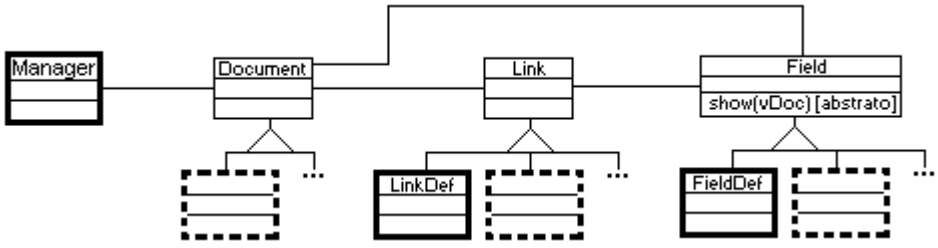


Figura 10.3 - um exemplo de modelo de objetos de um framework

[116] Em OMT e na metodologia de Coad & Yourdon a relação dos métodos é posta no símbolo de classe, do modelo de objetos; OOSE e Fusion relacionam métodos em um modelo à parte; a metodologia de Martin & Odell procede apenas análise, em que não chega a definir métodos.

O uso da notação sugerida torna claro que para a geração de uma aplicação, o usuário precisa gerar pelo menos uma subclasse de *Document*, e pode gerar subclasses de *Link* e *Field*. Esta informação não poderia ser obtida diretamente de um modelo de objetos produzido segundo a sintaxe de uma metodologia OOAD. Isto demonstra que a distinção de classes proposta, produz um aumento de expressividade para o modelo de objetos.

A identificação do tipo de método nas classes abstratas do framework também explicita aspectos necessários para a geração de aplicações. O método *show* da classe *Field*, por exemplo, que é classificado como abstrato no diagrama acima, deve ser sobreposto em todas subclasses concretas de *Field* geradas pelo usuário.

A figura 10.4 reúne a sugestão de um conjunto de elementos sintáticos para a construção do modelo de objetos. A simbologia se baseia principalmente na representação de OMT.

A representação de classes diferencia classe abstrata e não abstrata, além de criar uma representação para a classe a ser acrescentada pelo usuário do framework. Também inclui tipagem e cardinalidade aos atributos (opcional) e classificação para os métodos (base, abstrato, template) - também opcional.

A existência de vários elementos sintáticos de uso opcional está baseada na idéia de possibilitar a geração de um primeiro modelo de objetos de nível de abstração elevado. O levantamento de informações ao longo do processo de desenvolvimento permitirá um detalhamento das informações do modelo de objetos, usando neste caso, os elementos sintáticos opcionais para a geração de um modelo de objetos mais refinado.

Foi adotada a distinção entre associação estática e dinâmica. Um marcador de direcionalidade de associação foi adotado para opcionalmente ser acoplado às associações, como nas associações direcionadas de OOSE. Em uma associação estática o marcador indica que a classe que contém o símbolo tem conhecimento da outra classe. O marcador colocado nas duas extremidades de uma associação estática indica que uma classe tem conhecimento da outra. A ausência de marcador deixa em aberto no modelo de objetos, quem mantém referência. No caso da associação dinâmica, a seta aponta para a classe cujo objeto é chamado, no início de uma interação (diferencia quem inicia a interação). O marcador colocado nas duas extremidades de uma associação dinâmica indica alternância de iniciativa em diferentes situações de interação. A ausência de marcador deixa em aberto quem inicia procedimentos de interação.

Os qualificadores de associação estática permitem que o modelo de objetos contenha as mesmas informações sobre a associação estática, que podem ser representadas no grafo de visibilidade de Fusion (em que os qualificadores são baseados). A representação da limitação do tempo de vida não possui um qualificador, porém, esta característica pode ser expressa com o uso de agregação.

A semântica adotada para a associação de agregação é a seguinte: o objeto da classe agregada possui o objeto da classe parte, como definido pela delegação [RUM 94] (em termos de implementação, um atributo da classe agregada aponta para um objeto da classe parte). O tempo de vida do objeto da classe parte é limitado ao tempo de vida do objeto da classe agregada. Uma classe agregada não é definida exclusivamente pela união de objetos de classes parte, mas pode possuir seus próprios atributos e métodos. O acesso a um objeto de classe parte só ocorre a partir de métodos da classe agregada, ou se a classe agregada passar sua referência. A cardinalidade da agregação estabelece a quantidade de objetos da classe parte que podem estar contidos em um objeto da classe agregada. Uma relação de agregação indica que a classe

agregada mantém referência da classe parte (equivalente a uma associação estática) e que a classe agregada é cliente da classe parte (equivalente a uma associação dinâmica). A reciprocidade (casse parte manter referência ou ser cliente da classe agregada) demanda um símbolo adicional de associação.

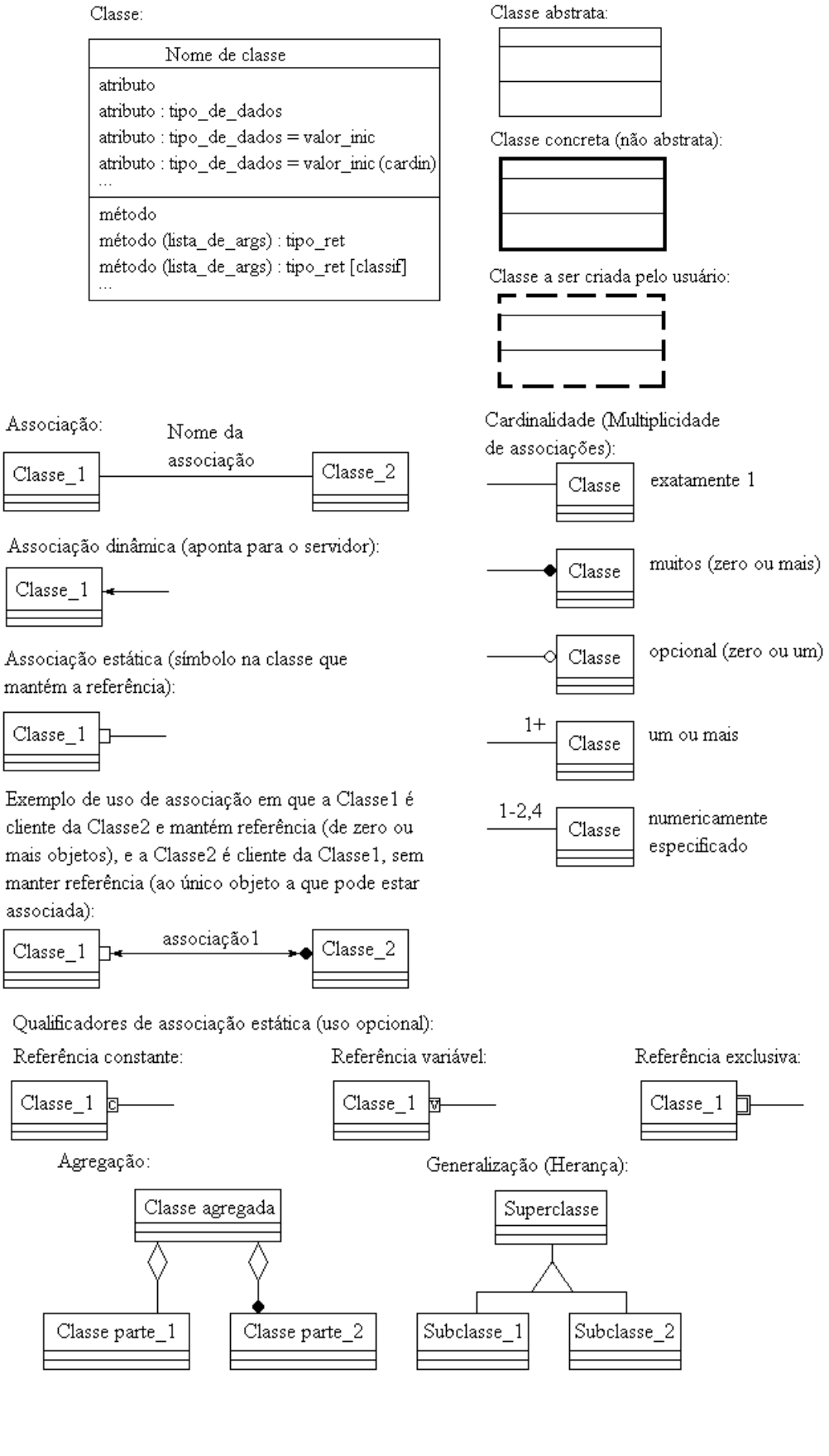


Figura 10.4 - elementos sintáticos do modelo de objetos

O modelo de objetos é passível de divisão em subsistemas, como proposto em OMT (capítulo quatro). Cada subsistema engloba aspectos do sistema que compartilham algumas propriedades comuns - funcionalidade, localização física etc. Em termos da estrutura de classes, um subsistema consiste num pacote de classes interrelacionadas, com uma interface bem definida com outros subsistemas.

Um dicionário de dados complementará o conjunto de modelos e conterá uma descrição textual das classes, descrição dos atributos, descrição textual dos métodos e uma seção para a descrição de outras características não relacionadas diretamente a classes específicas, à semelhança do que é feito em Fusion. A descrição de uma classe pode ser particularmente útil para os frameworks, por poder conter a descrição do papel da classe na geração de aplicações - no caso da descrição das classes abstratas, instruções para a criação de subclasses.

10.1.2 Modelagem dinâmica

O modelo dinâmico quando descreve a interação entre instâncias, deve diferenciar as instâncias de classes a serem criadas na geração de uma aplicação, das instâncias de classes do framework. Existem três situações possíveis a descrever:

- o objeto do diagrama é uma instância de classe do framework;
- o objeto do diagrama é uma instância de classe definida pelo usuário;
- o objeto do diagrama pode ser instância de classe do framework ou de classe criada pelo usuário.

A notação das metodologias OOAD não permite esta diferenciação, pois no contexto de uma aplicação específica, os objetos dos cenários são sempre instâncias de classes concretas da aplicação. Assim, para que se possa distinguir a origem dos objetos presentes em um cenário, é necessário uma alteração da sintaxe de representação dos objetos. A figura 10.5 contém uma sugestão de notação para a diferenciação das instâncias. O primeiro caso é semelhante ao que se observa em um cenário de aplicação: o objeto é identificado com o nome da classe a que pertence, e pode ser chamado para a execução dos métodos desta classe (ou métodos concretos de superclasses). Nos dois últimos casos a classe do objeto no diagrama será definida na geração de uma aplicação. Assim, o objeto é identificado com um nome de classe abstrata, indicando que corresponderá a uma instância de uma subclasse. Os métodos das chamadas ao objeto são métodos da classe abstrata. No último caso é indicado que o objeto pode ser uma instância de classe do framework, ou de classe criada pelo usuário. As classes das duas situações possuem a mesma superclasse (a classe abstrata indicada no diagrama).

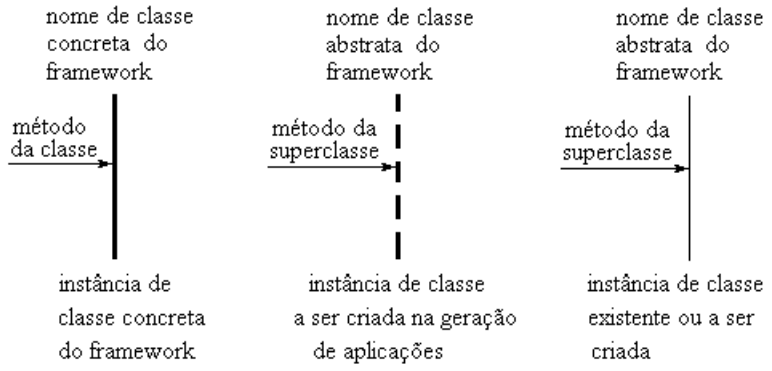


Figura 10.5 - diferenciação entre instâncias de classes do framework e da aplicação

A figura abaixo apresenta um cenário do framework para a visualização de hiperdocumentos (do exemplo anterior).

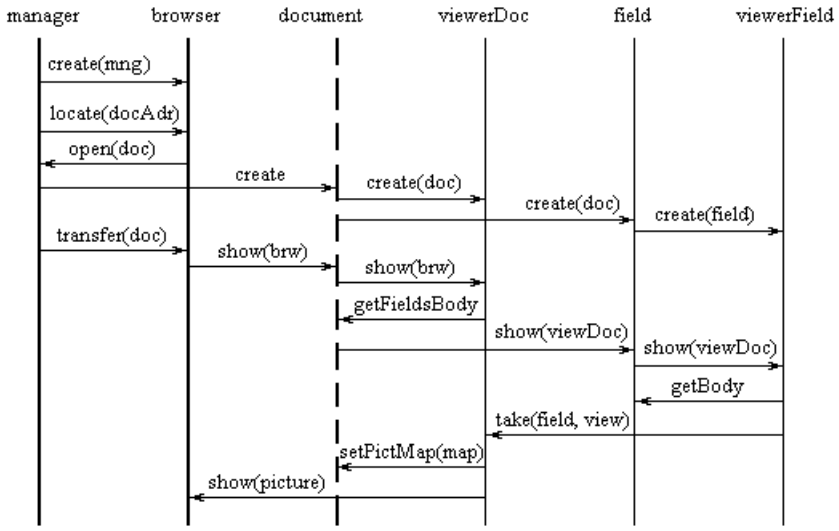


Figura 10.6 - um exemplo de modelagem de cenário de um framework

A notação adotada torna evidentes aspectos das classes a serem produzidas na geração de aplicações: os métodos a serem implementados, os clientes que usam estes métodos, se estes métodos dependem de outros métodos a serem implementados etc. Assim, observa-se um ganho de expressividade na modelagem dinâmica, a partir da alteração sintática proposta. Esta notação pode ser modificada de diversas maneiras, como representando a extensão dos métodos, adotando mecanismos de repetição, execução condicional de métodos (aspectos cobertos pela notação de OOSE), incluindo representação de chamada a métodos de classe (como pode ocorrer em Smalltalk), representando concorrência, interação com conjuntos de objetos.

A figura 10.7 apresenta a sugestão de um conjunto de elementos sintáticos para a construção de diagramas de interação (especificação de cenários). A simbologia se baseia principalmente na representação de OOSE. Adota uma notação alternativa ao pseudo-código de OOSE, baseada em diagramas de ação [MAR 91] destacando apenas a condição envolvida e os limites de uma execução condicional, looping ou não determinismo (representado por um *case*). Interação com conjuntos de objetos deve ser resolvida por meio de métodos de classe. Concorrência foi representada, como um destaque à possibilidade (e impossibilidade) de execução concorrente de métodos.

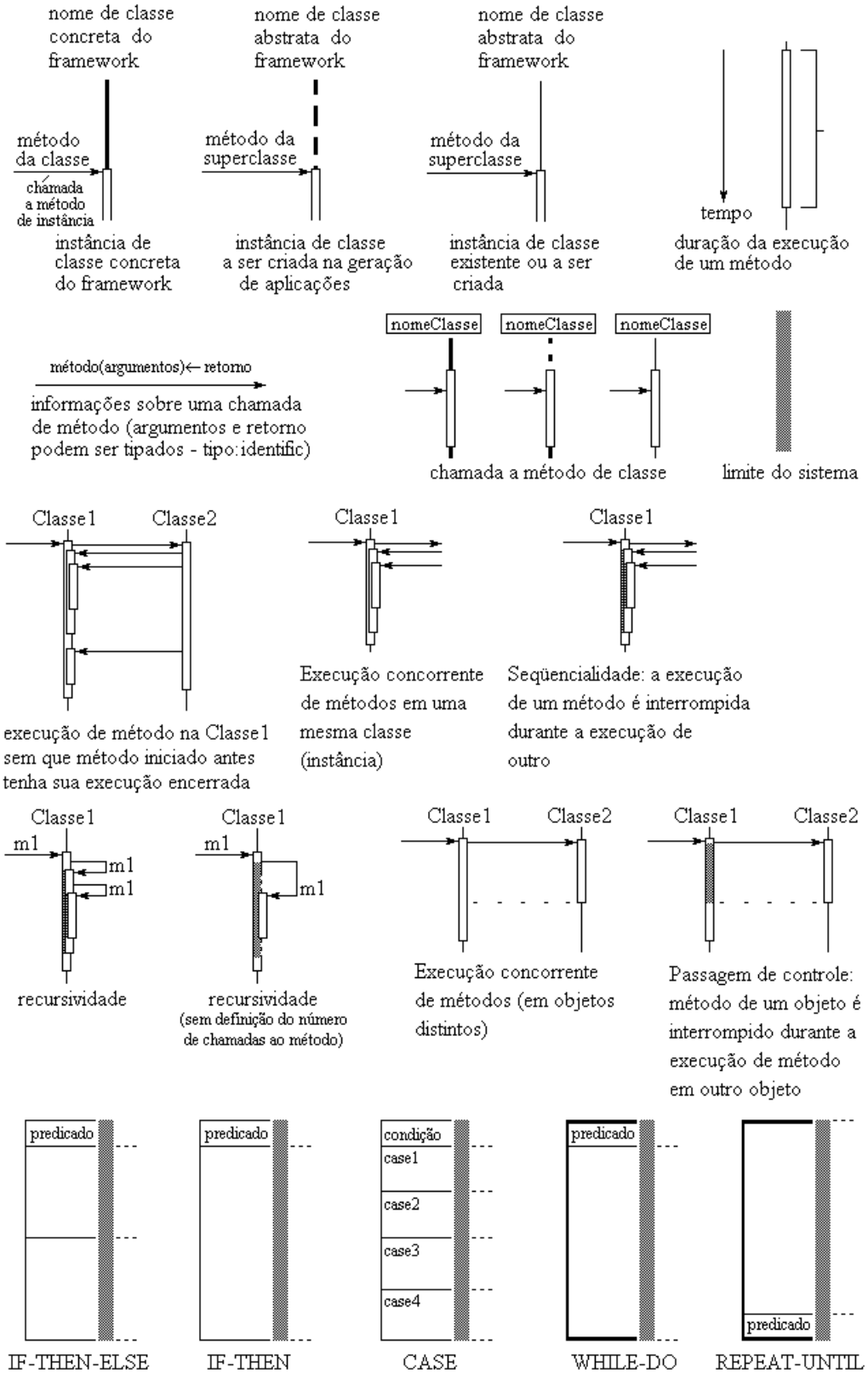


Figura 10.7 - elementos sintáticos do diagrama de interação

Um cenário descreve uma operação do sistema. O modelo de ciclo de vida da metodologia Fusion completa a modelagem dinâmica do sistema como um todo, descrevendo as seqüências de operações possíveis para o sistema, desde o início até o encerramento de uma execução, ou seja, o ciclo de vida de uma execução do sistema.

Para que esta técnica de modelagem seja usada em complemento às descrições de cenários, a única adaptação sintática necessária é que o alfabeto fique restrito a identificadores de cenários.

A modelagem dinâmica do sistema como um todo é complementada pelas associações dinâmicas incluídas no modelo de objetos, que comportam uma visão de alto nível da dependência dinâmica das classes em um único diagrama. Trata-se de uma visão diferente daquela expressa nos diagramas de interação, dado que estes são voltados a cenários - e mais de um é usado por especificação.

A modelagem dinâmica do comportamento particular de cada classe será feita a partir de statecharts (capítulo quatro) e diagramas SDL (capítulo cinco). Os statecharts descrevem a evolução de estados dos objetos, a partir da ocorrência de eventos. Neste caso não é necessário fazer alteração sintática para a especificação de frameworks. A descrição de uma classe concreta do framework é equivalente à descrição de classe de uma aplicação. Como a notação dos statecharts admite estruturação, um estado de uma classe abstrata do framework pode corresponder a um conjunto de estados da classe concreta de uma aplicação, sendo que o diagrama da superclasse sempre está contido no diagrama da subclasse.

Os diagramas SDL serão usados com ênfase à descrição dos algoritmos dos métodos - podendo omitir a representação de estados quando isto não for relevante para a descrição dos algoritmos. Como extensão sintática, será admitida a associação de um label a um símbolo de execução de tarefa, permitindo assim, a estruturação do diagrama - possibilidade inexistente na proposta original. A vantagem de usar diagramas SDL para o detalhamento de métodos é que este modelo além dos elementos de processamento de um fluxograma (execução de tarefa, ponto de decisão, destruição de instância), ainda destaca as comunicações efetuadas, o que é importante para explicitar a dependência de outros métodos de classes.

Uma técnica de modelagem adicional para a visão dinâmica

Grande parte da flexibilidade definida no projeto de um framework reside nos métodos de classe. Assim, a documentação do framework deve ter a capacidade de conter todos os aspectos de projeto definidos sobre os métodos, o que vai além dos algoritmos dos métodos. Anteriormente foi apresentada a classificação de Johnson para os métodos de classes abstratas como abstrato, template ou base [JOH 91]. Esta classificação pode ser estendida para classe concretas (considerando que uma classe concreta não poder apresentar um método abstrato). Um método de uma classe (qualquer que seja sua classificação) pode ser um método hook, sob a ótica de um método template que o acesse.

Além desta classificação, outros aspectos referentes a métodos devem poder ser representados nas técnicas de modelagem usadas no projeto de frameworks. Um primeiro aspecto adicional é a rede de colaborações de métodos originada a partir de um método template. Na medida que um método hook pode ser um método template, esta rede de colaborações pode ter mais de um nível de profundidade. Esta informação está contida nos cenários, mas sua clareza pode ser prejudicada pelo fato de poder estar pulverizada por vários diagramas, pois os cenários são voltados à descrição de situações de processamento concretas, e não a este tipo de sistematização de informação.

Quando um método template tem o próprio método como método hook, pode estar sendo expressa recursividade ou uma solicitação a um outro objeto da mesma

classe. A representação de métodos deve diferenciar claramente as duas situações, o que pode ser feito a partir de um destaque sintático à ocorrência de recursividade.

Um método pode ser disponível a outros objetos, ou de uso restrito ao objeto que o contém (método interno). Esta diferenciação deve ser representável.

Outro aspecto a diferenciar é se o método é um método de instância ou um método de classe, como ocorre em Smalltalk. Este aspecto também requer uma diferenciação sintática.

Como a descrição da rede de cooperação estabelecida em uma estrutura de classes não é explicitada nas técnicas de modelagem das metodologias OOAD, apresenta-se a sugestão de adotar uma técnica de modelagem adicional, o diagrama de cooperação, cuja sintaxe está apresentada abaixo. Esta técnica de modelagem tem a capacidade de expressar todos estes aspectos referentes a métodos de classe. Deixa claro o que a estrutura do framework determina para os métodos das classes a serem criadas na definição de aplicações - diferencia métodos que precisam ser definidos, métodos que podem ser redefinidos e métodos definidos que não devem ser alterados. Cabe salientar que esta técnica de modelagem pode ser adotada por metodologias OOAD.

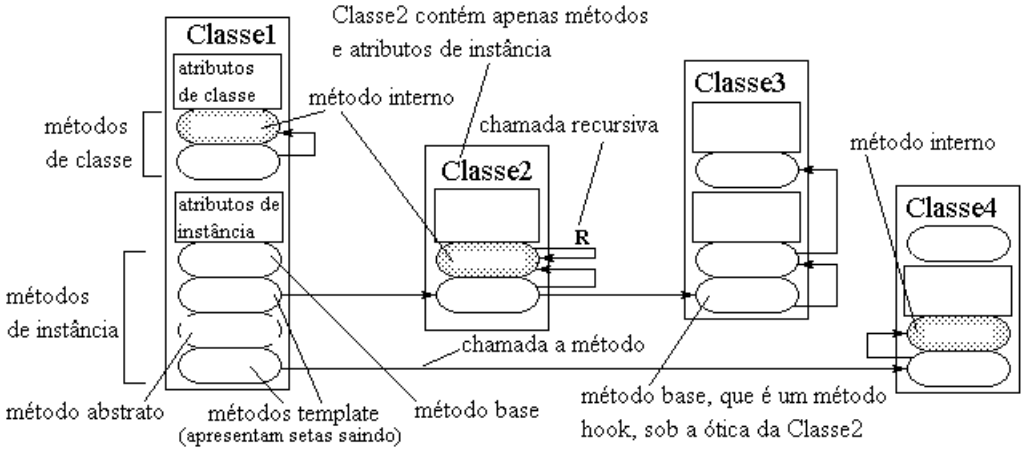


Figura 10.8 - elementos sintáticos do diagrama de cooperação

O diagrama de cooperação pode ser feito segundo diferentes óticas. Pode ser elaborado um único diagrama para um framework. Para frameworks com uma grande quantidade de métodos e de chamadas a métodos, esta opção pode ser impraticável, por gerar um diagrama de difícil leitura. Uma outra opção é gerar um diagrama para cada classe, exprimindo apenas um nível de cooperação: os métodos chamados pelos métodos da classe descrita, e as chamadas aos métodos da classe descrita feitas por métodos de outras classes. Se o diagrama gerado por esta ótica for de difícil leitura, pode ser elaborado um diagrama por classe contendo apenas as chamadas a métodos originadas na classe descrita. Uma outra ótica possível seria o destaque apenas às chamadas a métodos da classe descrita. Havendo a disponibilidade de uma ferramenta automatizada para a elaboração e visualização de diagramas de cooperação, todas estas óticas poderiam estar disponíveis - e o usuário da ferramenta poderia optar pela exibição de uma ou outra. Uma ferramenta automatizada poderia inclusive integrar os modelos de objetos e de cooperação em um mesmo modelo - a visão desejada para exibição seria selecionada pelo usuário.

Algumas particularidades de projeto que não são tão explicitadas em outras técnicas de modelagem, podem ser destacadas com o uso do diagrama de cooperação.

Lista-se a seguir um conjunto destas situações, que ilustram a vantagem da utilização desta técnica de modelagem.

- método abstrato que também é template: um método template a princípio é um método com um algoritmo completamente definido. A flexibilidade que dispõe está concentrada na redefinição dos métodos hook acessados. Um método que depende de métodos hook e não está completamente definido é um método abstrato, mas com característica de template. Esta situação que é representada com o símbolo de método abstrato e com setas saindo, indica a necessidade de completar a definição do método, assim como uma flexibilidade adicional através da definição de métodos hook.
- método template que demanda definição de métodos hook: se algum dos métodos hook acessados por um método template for abstrato, a criação de uma aplicação demanda a definição deste método.
- método template que não demanda definição de métodos hook: se todos os métodos hook acessados por um método template forem métodos base (ou métodos template que acessem apenas métodos base[117]), a criação de uma aplicação pode ocorrer sem a redefinição de métodos hook.
- método template que demanda definição de métodos hook em diferentes níveis de cooperação: verificar se um método template demanda a definição de algum método abstrato, implica em seguir toda a cadeia de cooperação iniciada no método à procura de métodos abstratos.
- funcionalidades mantidas inflexíveis: métodos base de classes que não podem ser redefinidas[118], ou métodos template de classes que não podem ser redefinidas, que acessam métodos base de classes que também não podem ser redefinidas.

Técnicas de modelagem que explicitem a cadeia de métodos envolvidos na execução de um determinado método, como feito pelo diagrama de cooperação, podem auxiliar a fatoração da estrutura de classes, pela identificação de semelhanças funcionais.

10.2 Limitações da proposta e possível caminho para a sua evolução

A avaliação da possibilidade das metodologias OOAD serem usadas para o desenvolvimento de frameworks, originou o presente estudo, ainda em andamento. A proposta de técnicas de modelagem para a especificação de frameworks apresentada foi elaborada a partir da coleta das características julgadas positivas nas metodologia OOAD e de desenvolvimento de frameworks. Um objetivo buscado na definição do conjunto de técnicas de modelagem, foi minimizar a dependência de descrição textual - aspecto negativo que se encontra em algumas metodologias - procurando os técnicas de modelagem semiformais mais adequados às várias visões que se necessita gerar para a especificação de um sistema. Os acréscimos sugeridos às técnicas de modelagem geram um ganho de expressividade, como demonstrado nos exemplos. Esta contribuição

---

117 Esta seqüência de acesso a métodos template pode prosseguir por vários níveis, desde que os métodos hook do final da cadeia sejam métodos base.

118 O projeto de um framework define quais classes podem dar origem a subclasses. Considera-se classes que não podem ser redefinidas, aquelas que, segundo a estrutura de controle definida no projeto de framework, não podem dar origem a subclasses, não podendo portanto, ter seus métodos sobrepostos.

porém, não é suficiente para a definição de um procedimento completo de desenvolvimento de frameworks.

O projeto de frameworks envolve conceitos não explorados suficientemente pelas metodologias OOAD. Um primeiro aspecto é a necessidade de mecanismos que levem mais diretamente à descrição do domínio. As metodologias OOAD de um modo geral, determinam "identificar classes", "identificar estruturas", mas sem estabelecer como, assim como não se aprofundam na questão da fatoração da estrutura de classes . Um possível caminho para transpor esta limitação é o estudo de técnicas de Análise de Domínio que levem a uma sistematização dos procedimentos de identificação de classes e estruturas. Uma outra possibilidade a ser buscada em complemento a esta, é o estudo de técnicas de Engenharia Reversa que sistematizem o processo de extrair uma descrição orientada a objetos de uma aplicação já desenvolvida (segundo qualquer abordagem).

Um outro aspecto a ser melhor explorado é o uso de design patterns no desenvolvimento de frameworks. A utilização deste recurso no desenvolvimento de frameworks tende a minimizar o esforço de desenvolvimento, por dispor soluções para problemas comuns de projeto [GAM 94]. É preciso definir mais precisamente o papel dos patterns no processo de desenvolvimento, o estágio do processo em que são usados e, mais importante, de que forma utilizá-los para gerar um ganho de qualidade no desenvolvimento de frameworks.

11 Conclusão

Neste trabalho foram apresentadas as metodologias de desenvolvimento de aplicações de Coad e Yourdon, OMT, OOSE, Fusion e de Martin e Odell - todas baseadas em orientação a objetos. Foram apresentados os técnicas de modelagem usados, e os passos para a construção de uma especificação de sistema (processo de desenvolvimento).

O desenvolvimento de um exemplo para cada metodologia a partir de uma mesma descrição, tornou a apresentação das metodologias mais completa, permitindo inclusive comparar o conjunto de modelos produzido por cada uma. A partir da descrição das metodologias e das especificações geradas, foi feita uma avaliação de suas potencialidades e deficiências.

Foram apresentadas as características dos frameworks e três metodologias de desenvolvimento de frameworks - Projeto Dirigido por Exemplo, Projeto Dirigido por Hot Spot e metodologia da empresa Taligent. Os frameworks foram situados como uma evolução do processo de desenvolvimento de software orientado a objetos, em relação a abordagens anteriores, pela característica de viabilizarem a reutilização de software em larga escala.

Analisando metodologias de desenvolvimento de frameworks propostas, observa-se a ausência de técnicas de modelagem e de um processo de desenvolvimento detalhado. Tais características fazem parte de metodologias OOAD. Assim, a partir da análise de metodologias OOAD e de metodologias de desenvolvimento de frameworks, discutiu-se como adaptar mecanismos das metodologias OOAD para seu uso no desenvolvimento de frameworks.

A discussão além de comparar metodologias, destacou os aspectos de metodologias OOAD que necessitam de adaptação para seu uso no desenvolvimento de frameworks. Foram propostas alterações nas técnicas de modelagem das metodologias OOAD e demonstrados os acréscimos obtidos, em termos de expressividade. A diferenciação entre classes abstratas e concretas do framework, e classes a serem acrescentadas à estrutura do framework, é um aspecto importante do projeto, que deve ser refletido na notação de projeto utilizada. O mesmo ocorre com a classificação de métodos. Foi proposto um conjunto de técnicas de modelagem para a documentação do projeto de um framework.

O estudo desenvolvido é uma etapa parcial de um esforço de pesquisa ora em curso, que visa analisar e propor técnicas de modelagem e contribuições ao processo de desenvolvimento de frameworks. Outros aspectos ainda precisam ser explorados para que se atinja esta meta, tais como a definição de um conjunto ótimo de técnicas de modelagem para a especificação de um framework, assim como de um processo de desenvolvimento; uso de metodologias de captura de características do domínio (técnicas de Análise de Domínio e de Engenharia Reversa); utilização de design patterns e outras abordagens.

Bibliografia

[BOO 91] BOOCH, G. **Object oriented design** - with applications. Redwood City, California: Benjamin Cummings, 1991.

[CCI 88] CCITT. **Specification and description language (SDL)**. Geneva: CCITT, 1988. Recommendation Z.100.

[CHE 76] CHEN, P. P. S. The entity-relationship model - toward a unified view of data. **ACM Transactios on Database Systems**. v.1, n.1, mar. 1976.

[COA 92] COAD, P., YOURDON, E. **Análise baseada em objetos**. Rio de Janeiro: Campus, 1992.

[COA 93] _____. **Projeto baseado em objetos**. Rio de Janeiro: Campus, 1993.

[COL 94] COLEMAN, D. *et all*. **Object-oriented development**: the Fusion method. Englewood Cliffs: Prentice Hall, 1994.

[DEM 79] DEMARCO, T. **Structured analysis and system specification**. Englewood Cliffs: Prentice Hall, 1979.

[DEU 89] DEUTSCH, P. Frameworks and reuse in the smalltalk 80 system. In: Biggerstaff, Ted. **Software reusability**. New York: ACM Press, 1989. v.1, p. 57-71.

[FOW 94]. FOWLER, M. Describing and comparing object-oriented analysis and design methods. In: Carmichael. **Object development methods**. New York: SIGS Books, 1994. p. 79-109.

[GAM 94] GAMMA, E. **Design patterns**: elements of reusable object-oriented software. Reading: Addison-Wesley, 1994.

[GAN 78] GANE, C., SARSON, T. **Structured systems analysis**: tolls and techniques. Englewood Cliffs: Prentice Hall, 1979.

[HAR 87] HAREL, D. *et all*. Statecharts: a visual formalism for complex systems. **Science of Computer Progamming**. n.8, p.231-274, 1987.

[HOD 94] HODGSON, R. Contemplating the universe of methods. In: Carmichael. **Object development methods**. New York: SIGS Books, 1994. p. 111-132.

[JAC 92] JACOBSON, I. *et all*. **Object-oriented software engineering -** a use case driven approach. Reading: Addison-Wesley, 1992.

[JOH 88] JOHNSON, R. E., FOOTE, B. Designing reusable classes. **Journal of object-oriented programming**. p. 22-35, jun./jul. 1988.

[JOH 91] JOHNSON, R. E., RUSSO, V. F. **Reusing object-oriented designs**. Urbana: University of Illinois , 1991. Technical Report UIUCDCS91-1696.

[JOH 92] JOHNSON, R. E. Documenting frameworks using patterns. In: Object-Oriented Programming Systems, Languages and Applications Conference - OOPSLA, 1992, Vancouver. **Proceedings...** Vancouver.

[JOH 93] JOHNSON, R. E. **How to design frameworks**. In: Object-Oriented Programming Systems, Languages and Applications Conference - OOPSLA, 1993, Washington. **Proceedings...** Washington.

[MAR 95] MARTIN, J., ODELL, J. **Análise e projeto orientados a objetos**. São Paulo: Makron Books, 1995.

[MAR 91] MARTIN, J. **Técnicas estruturadas e CASE**. São Paulo: Makron Books, McGraw-Hill, 1991.

[MEY 88] MEYER, B. **Object-oriented software construction**. Englewood Cliffs: Prentice Hall, 1988.

[MON 92] MONARCHI, D. E., PUHR, G. A research typology for object-oriented analysis and design. **Communications of the ACM**. v.35, n.9. sep. 1992.

[MOR 94] MOREIRA, A. M. D. **Rigorous object-oriented analysis**, Stirling: University of Stirling, aug. 1994. Doctor of Philosophy Thesis.

[PAS 94] PASTOR, E. A. **Estudo comparativo de metodologias de desenvolvimento de software**. Porto Alegre: UFRGS/II/CPGCC, 1994. Trabalho Individual.

[PER 95] PEREZ, M., ANTONETI, J. A qualitative comparison of object-oriented methodologies. In: 5th. International Symposium on Systems Research, Informatics and Cybernetics, Baden, aug. 1995. **Proceedings...** Baden.

[PRE 95] PREE, W. **Design patterns for object oriented software development**. Reading: Addison-Wesley, 1995.

[RUM 94] RUMBAUGH, J. *et all*. **Modelagem e projetos baseados em objetos**. Rio de Janeiro: Editora Campus, 1994.

[TAL 94] TALIGENT. **Building object-oriented frameworks**. Taligent Inc. white paper, 1994. (disponível em WWW no endereço *http://www.taligent.com*)

[TAL 95] TALIGENT. **Leveraging object-oriented frameworks**. Taligent Inc. white paper, 1995. (disponível em WWW no endereço *http://www.taligent.com*)

[WIR 90] WIRFS-BROCK, R., JOHNSON, R. E. Surveying current research in object-oriented design. **Communications of the ACM**. v.33, n.9. sep. 1990.

[WIA 91] WIRFS-BROCK, A. *et alli*. Designing reusable designs: Experiences designing object-oriented frameworks. In: Object-Oriented Programming Systems, Languages and Applications Conference / European Conference on Object-Oriented Programming, oct. 1991, Ottawa. **Addendum to the proceeddings of OOPSLA/ECOOP'90**. Ottawa.

[YOU 89] YOURDON, E. **Modern structured analysis**. Englewood Cliffs: Prentice Hall, 1989.