PARTIÇÃO DE SISTEMAS NO PARADIGMA DE OBJETO

SYSTEM PARTITIONING IN OBJECT-ORIENTED PARADIGM

Profª Drª Beatriz M. DALTRINI*
Sergio R. SIGRIST**

ABSTRACT

Several mechanisms of system partitioning in object-oriented paradigm are available in literature, but there is not a consensus about which of them shows a more granularity context vision than a class of objects provides. This paper surveys three mechanisms often quoted: *subject*, *ensemble* and *subsystem*.

KEY WORDS: Systems Partitioning, Object Grouping, Object-Oriented Analysis.

RESUMO

Diversos mecanismos de partição de sistemas no paradigma de orientação a objetos são encontrados na literatura, mas não existe ainda um consenso sobre qual unidade oferece uma visão de contexto de maior granularidade do que o proporcionado por uma classe de objetos. Este artigo examina três mecanismos citados com freqüência: *subject*, *ensemble* e *subsystem*.

PALAVRAS-CHAVE: Partição de Sistemas, Agrupamento de Objetos, Análise Orientada a Objetos.

1. INTRODUÇÃO

Alguns problemas de software são grandes e complexos para serem compreendidos como um todo. Por essa razão, dividem-se tais problemas em partes menores que sejam mais compreensíveis, estratégia que passou a ser conhecida como "dividir e conquistar", e estabelecem-se as interfaces entre elas para que a funcionalidade total do sistema seja atingida.

BOOCH (1991) considera que o desafio atual consiste no desenvolvimento de aplicações que exibem um conjunto rico de comportamentos, como por exemplo, sistemas que controlam a rota num tráfego aéreo ou que imitam certos aspectos da inteligência humana. Para esses tipos de software, a decomposição é essencial.

Na metodologia estruturada, em essência, decompõe-se o problema em partes e estabelece-se uma representação hierárquica das funções, onde o elemento de maior nível cresce a medida em que se caminha nos sentidos horizontal e vertical da representação (PRESSMAN, 1992).

Uma maneira similar deve ser definida na orientação a objetos (OO), que permita a decomposição de grandes sistemas logo nos estágios iniciais do processo de desenvolvimento. Em geral, os métodos de análise/"design" propõem meios para realizar a decomposição, porém o tema ainda encontra-se em debate.

FICHMAN & KEMERER (1992) consideram que o enfoque "*top-down*" da análise estruturada é preferível em relação à maneira "*bottom-up*" da OO, porque classes e instâncias, mesmo as de mais alto nível, têm menor granularidade e são definidas tarde demais no processo de desenvolvimento para proporcionar uma base de partição. Para MONARCHI & PUHR (1992), enquanto o diagrama de nível micro consiste de objetos e seus interrelacionamentos, não fica claro quais construções devem ser mostradas num diagrama de nível macro, representado pelo agrupamento de classes.

(*) Profª Drª Beatriz M. Daltrini - Professora de Engenharia de Computação e Automação Industrial - Faculdade de Engenharia Elétrica - Universidade Estadual de Campinas - FEE/UNICAMP - Caixa Postal 6101 - CEP 13081-970 - Campinas (SP)
(**) Sérgio R. Sigrist - Aluno de mestrado em Engenharia de Computação e Automação Industrial - Faculdade de Engenharia Elétrica - Universidade Estadual de Campinas - FEE/UNICAMP - Analista de Sistemas do Centro de Informática na Agricultura da Universidade de São Paulo - USP/Piracicaba - Av. Pádua Dias, 11 - Caixa Postal 9 - CEP 13418-900 - Pircacicaba (SP)

A orientação a objetos evolui rapidamente e novas contribuições são apresentadas. Este artigo examina as caraterísticas das propostas de partição citadas com freqüência na literatura e acrescenta mais detalhes do que os encontrados em FICHMAN & KEMERER (1992) e MONARCHI & PUHR (1992).

2. DEFINIÇÕES

Diversos autores têm se posicionado sobre a importância de se ter um vocabulário comum para discutir a OO. Alguns termos empregados no presente texto serão apresentados a seguir.

Classe é uma coleção de atributos e serviços comuns e ***instância*** é uma ocorrência ou a realização de uma classe. Em geral, o termo ***objeto*** refere-se a ambos, classe e instância (MONARCHI & PUHR, 1992).

Atributos são elementos de dados que descrevem uma instância de um objeto. ***Serviço*** é o processamento realizado por um objeto quando este recebe uma mensagem (COAD & YOURDON, 1990). ***Encapsulamento*** significa a manipulação dos atributos de um objeto unicamente pelos serviços definidos neste mesmo objeto.

Os ***relacionamentos*** mais importantes entre objetos, por serem diretamente representados na implementação, são ***agregação e classificação*** (PITTMAN, 1993). Agregação é usada para modelar como um objeto faz parte de outro objeto, ou como um objeto representa uma coleção de outros objetos. Classificação, ou relacionamento superclasse/subclasse, é realizada através da hierarquia de ***herança***, onde atributos e serviços são transmitidos das classes gerais para as classes específicas.

Outros termos também são definidos na OO. Um estudo aprofundado pode ser encontrado em SNYDER (1993), cujo trabalho resultou num modelo abstrato atualmente adotado pelo Object Management Group, Inc[1].

3. PARTIÇÃO NO PARADIGMA DE OBJETOS

Os elementos de partição encontrados nas abordagens OO de diferentes autores são *ensemble, subject* e *subsystem*.

Ensembles (De CHAMPEAUX et al, 1993) são objetos com propriedades especialmente definidas para forçar uma aproximação com o enfoque "*top-down*" de desenvolvimento.

Subjects (COAD & YOURDON, 1990) são formas de representar um agrupamento de objetos a partir dos relacionamentos de agregação e classificação.

Subsystem (WIRFS-BROCK & JOHNSON, 1990) é um conjunto de classes (e possivelmente outros *subsystems*) que cooperam entre si para satisfazer um conjunto comum de responsabilidades.

Esses três elementos serão mostrados em detalhes a seguir.

3.1. ENSEMBLE

Ensemble (De CHAMPEAUX, 1993) são geralmente grandes encapsulamentos de objetos com propriedades especiais definidas para representar diferentes camadas de abstração e permitir a decomposição "*top-down*".

Ensembles assemelham-se a objetos, pois possuem atributos, podem ter uma máquina de transição de estados associada e podem interagir com outros objetos. Diferem dos objetos pelo fato de agrupar objetos (ou outros *ensembles*) de menor nível denominados *componentes*. Os componentes ficam ocultos de outros objetos e interagem somente com os componentes pertencentes ao mesmo *ensemble*. Desta maneira, um *ensemble* pode ser visto como um *gateway* entre seus componentes e o resto do sistema.

Outra diferença de *ensemble* e objeto é que um objeto pode ser concebido como uma máquina seqüencial, enquanto *ensemble* denota uma entidade com paralelismo interno. Num sistema bancário, uma ***conta*** pode ser um objeto somente quando uma transação for realizada. Por outro lado, a "gerência de empréstimos" pode ser representada como um *ensemble* porque seus componentes, as seções de empréstimo, podem operar em paralelo.

Um *ensemble* oculta os detalhes de seus componentes que são irrelevantes fora do *ensemble*, da mesma maneira que uma linguagem de programação orientada a objeto oculta detalhes internos de implementação dos objetos.

Dois tipos de atributos podem ser definidos: atributos de *ensemble* e atributos de componentes. Atributos de *ensemble* referem-se somente ao *ensemble*, enquanto que atributos de componentes são compartilhados por todos os componentes do ensemble. Por exemplo, uma esquadra, representada como *ensemble*, pode ter os atributos "quantidade de embarcações" (atributo de *ensemble*) e "direção" (atributo compartilhado pelos componentes).

(1) Associação entre empresas para padronizar termos e definições da orientação a objetos.

Na ausência de atributos de componentes, pode-se desenvolver um modelo de transição de estados e introduzir um *trigger* para modelar a interação entre ensemble e componentes. *Trigger* difere de pedido de serviço ou transmissão de mensagem, seu único efeito é promover uma mudança de estado no objeto receptor. Um *ensemble* Esquadra pode receber um aviso "retornar ao porto" e transmiti-lo a todos os seus componentes. Um exemplo de interação entre um *ensemble* e um componente pode ser a Esquadra dando uma direção específica para uma das embarcações.

As propriedades de *ensemble* podem ser resumidas da seguinte maneira:

- *Ensemble* é um objeto cujos componentes funcionais são objetos ou outros *emsembles*.

- Um componente é parte de ,no mínimo, um e, no máximo, um *ensemble*. Essa propriedade garante que o relacionamento ensemble/componente seja não-transitivo e define múltiplas camadas de abstração;

- Um *ensemble* serve como mediador das interações entre os seus componentes e as entidades fora dele;

- Os componentes podem interagir somente entre si e não com objetos fora do *ensemble* do qual fazem parte;

- Um *ensemble* é responsável pela criação e eliminação de seus componentes.

A notação de DE CHAMPEAUX é mostrada na figura 1.

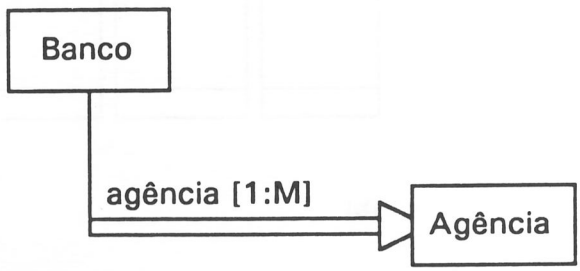

Figura 1 - Exemplo de um *Ensemble* (de Champeaux et al, 1993)

No exemplo tem-se um sistema bancário modelado com o *ensemble*. Os retângulos representam classes. A linha dupla significa que Banco representa um *ensemble* cujos componentes são as agências bancárias.

Os autores propõem artifícios pra realizar a comunicação entre *ensemble* e componente, através da herança múltipla, e expor parcialmente, quando necessário, o comportamento de um particular componente de um *ensemble* ao sistema. Também são propostas formas de modelar um sistema completo a partir de *ensembles*, por meio de adição de atributos especiais nos componentes do *ensembles*.

Uma característica final de *ensemble* é a implementação. Devido às propriedades de encapsulamento e de comunicação, *ensemble* pode ser visto como um objeto e ser manipulado diretamente na implementação.

3.2. SUBJECT

Subject é um mecanismo conceitual definido no método Object-Oriented Analysis (COAD & YOURDON, 1990). Este método propõe realizar um processo de análise em cinco etapas para construir o modelo de objetos:

- Encontrar objetos e classes;

- Identificar estruturas;

- Definir *subject*;

- Definir atributos dos objetos e conexões de instância;

- Definir serviços dos objetos e conexões de mensagem.

Neste processo, *subject* é visto como um meio de facilitar a compreensão de um modelo composto por muitos diagramas. Os autores citam o trabalho de Miller[2], segundo o qual a capacidade de memória de trabalho humana é limitada a "5 até 9 itens ao mesmo tempo". Este resultado pode ser aplicado na OO, colocando-se de 5 a 9 ícones de objetos num desenho.

O ponto de partida para a definição de *subject* são as chamadas estrutura de montagem, que representa o relacionamento de agregação, e estrutura de classificação. Segundo os autores, ambas expressam um método básico de organização do pensamento humano. A estrutura de classificação proporciona uma importante partição do problema, pois distingue agrupamentos de objetos mutuamente exclusivos.

A figura 2 mostra um exemplo da notação de COAD & YOURDON.

(2) MILLER, G. The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological Review, p. 81-97, Mar. 1956.

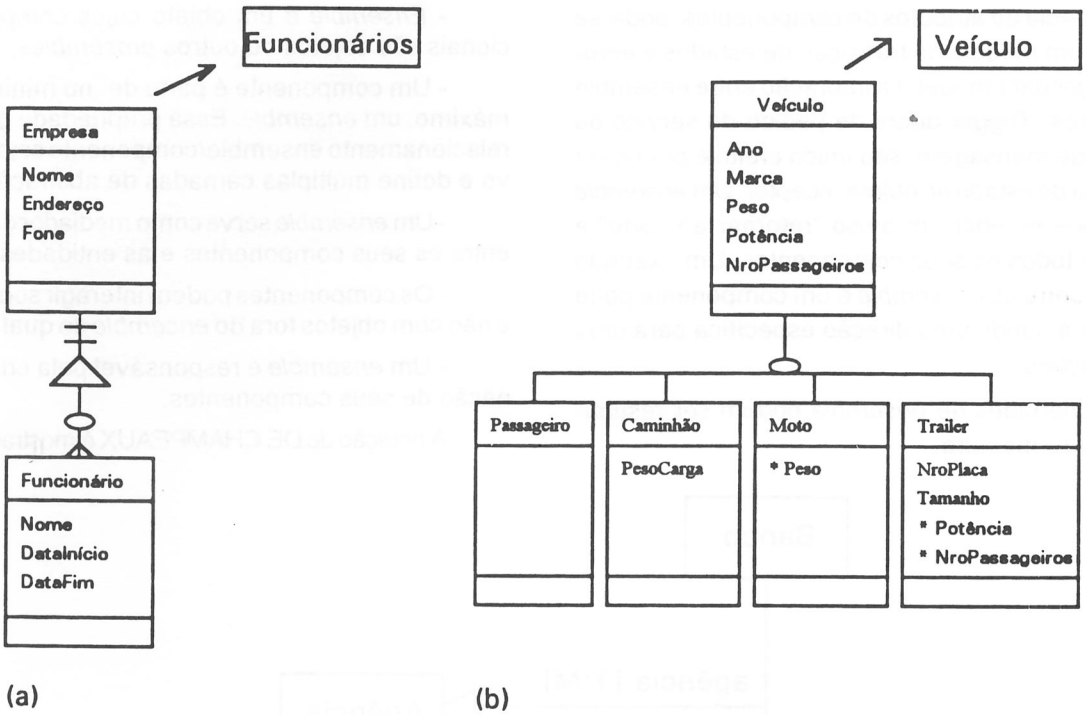

Figura 2 - Estruturas de agregação(a) e classificação (b) (COAD & YOURDON, 1990)

Na figura 2 (a), uma Empresa é mapeada a um conjunto de Funcionários (estrutura de agregação representando coleções). Esta estrutura sugere um *subject* Funcionários, representado pelo retângulo.

Na figura 2 (b), Veículo é uma classe geral e Passageiro, Caminhão, Moto e Trailer são classes especializadas. Esta estrutura pode ser entendida como um *subject* **Veículo**. O mecanismo de herança fica visível. Atributos não herdados pelas classes especializadas são anotados com um asterisco. Novos atributos podem ser alocados às classes especializadas, como indicado na classe especializada Trailer.

Os autores propõem um guia para a identificação de *subject*. Por exemplo, se uma estrutura de classificação apresentar entre 5 e 9 classes, esta estrutura é candidata a *subject*.

A introdução de *subject* está associada à complexidade do modelo, devendo em grandes projetos ser feita logo no início. A recomendação é que uma equipe de analistas faça uma rápida identificação de objetos, estruturas e um conjunto inicial de *subjects*. Este procedimento é denominado "blitz". Tais *subjects* podem ser associados às equipes de projeto e posteriormente refinados.

3.3. SUBSYSTEM

Subsystem (WIRFS-BROCK JOHNSON, 1990) é um conceito definido no método Responsability-Driven Design. O método baseia-se nas responsabilidades e colaborações entre objetos através do modelo cliente-servidor e o mecanismo denominado *contrato*.

As responsabilidades de um objeto são todos os serviços que ele proporciona aos objetos que os solicitam ou os serviços que ele passa a outros (colaboração) objetos. Assim, os objetos assumem as suas responsabilidades executando a sua computação ou colaborando com outros objetos.

Contrato descreve a forma pela qual um cliente pode interagir com o servidor. Ambos devem subscrever um contrato: o cliente apresenta os seus pedidos e o servidor responde apropriadamente aos pedidos. Uma classe pode suportar um ou vários contratos, dependendo se seus serviços forem usados por um ou mais clientes.

Nesse contexto, um *subsystem* significa um conjunto de classes (e possivelmente outros *subsystems*) que cooperam entre si para satisfazer um conjunto comum de responsabilidades. *Subsystems* não são suportados diretamente por linguagens orientadas a objeto, não existindo portanto durante uma execução. Figuram apenas como uma construção da análise. São maneiras de pensar sobre grandes sistemas. Um meio para verificar se um agrupamento de classes forma um *subsystem*

é tentar *dar um nome* a este agrupamento.Se for possível escolher o nome, a função de maior nível que as classes cooperam para encontrar é um *subsystem*.

MARTIN (1993) refere-se a *subsystem* como um *container* de classes. Quando um pedido de Serviço for enviado ao *subsystem*, o pedido é delegado à classe dentro do *subsystem* que suporta o pedido. Olhando de fora, cada *subsystem* assemelha-se a uma classe que suporta seus próprios contratos. Outras classes comunicam-se com o *subsystem* porém não conhecem nada do seu interior. Do mundo exterior, não há diferença entre uma classe, um subsystem ou mesmo uma aplicação total. Tudo são objetos encapsulados com contratos para proporcionar os serviços.

O grafo de colaboração é a ferramenta empregada para representar subsystem, como mostra a figura 3.

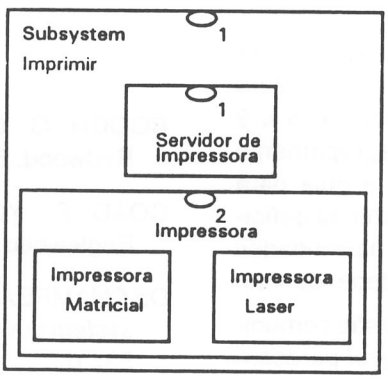

Figura 3 - Grafo de colaboração (WIRFS-BROCK & JOHNSON, 1990)

A figura 3 mostra um *subsystem* denominado Imprimir.Contratos são indicados por semicírculos e classes são representadas por retângulos. O *subsystem* Imprimir encapsula as classes Servidor e Impressora. As subclasses Impressora Matricial e Impressora Laser ficam aninhadas dentro da superclasse Impressora. Todas essas classes colaboram com a impressão de arquivos. Embora a classe Servidor colabore com uma outra classe Fila, esta classe não faz parte do *subsystem* Imprimir porque as instâncias de Fila são usadas por classes fora do *subsystem* Imprimir. Uma classe faz parte de um *subsystem* somente se ela satisfizer a finalidade deste *subsystem*.

Subsystems simplificam a fase de projeto de grandes aplicações na medida em que podem ser tratados como classes. Uma aplicação pode ser decomposta em vários *subsystems* e estes podem ser modelados até que todos os detalhes sejam especificados.

Em razão dos clientes usarem a funcionalidade de um *subsystem* através de um conjunto bem definido de contratos, a funcionalidade de um *subsystem* pode ser ampliada sem romper com o resto da aplicação. Um novo contrato pode ser definido, ou um contrato já existente pode ser ampliado para incluir uma nova funcionalidade. No *subsystem* do exemplo dado, pode-se acrescentar a habilidade de imprimir num determinado instante ou imprimir um numero específico de cópias. Os contratos existentes vão adequadamente lidar com esta nova funcionalidade: o *subsystem* Impressora imprime o arquivo (contrato já existente) mas de diversas maneiras (nova funcionalidade).

4. CONCLUSÃO

Foram mostradas em detalhes as caraterísticas de *ensemble, subject* e *subsystem,* que procuram oferecer uma visão de maior nível do que a representada por uma classe de objetos. A Tabela 1 faz uma síntese dos principais itens descritos no artigo.

Tabela 1 - Síntese de itens descritos

	Ensemble	*Subject*	*Subsystem*
representação	sim	sim	sim
visão de contexto	agregação(c/ restrições)	agregação; classificação	classes funcionais; classificação
guia	sim	sim	sim
comunicação	entre *ensembles; interensemble*	entre *subjects*	entre *subsystems; intersubsystem*
encapsulamento	sim	não	sim
implementável	sim	não	não

Os mecanismos de agrupamento examinados apresentam vários pontos em comum. Todos proporcionam uma ferramenta de representação. Em cada representação fica implícita a visão de contexto. A base é formada por um relacionamento especifico, agregação e classificação, entre as classes pertencentes ao agrupamento. Aqui *subsystem* difere dos demais pois inclui classes que trabalham juntas para obter uma função de mais alto nível.

Todos os mecanismos sugerem um guia para obter a partição. Guias para *subject* (construir uma estrutura de classificação ou de agregação contendo de 5 a 9 classes) e *subsystem* (tenta nomear um agrupamento de classes) são explícitos, enquanto que o guia para derivar um *ensemble* pode ser inferido a partir da definição de ensemble, ou seja, tentar identificar as entidades que decompõem o problema em partes independentes.

Outros conceitos fundamentais da OO são comunicação e encapsulamento. *Ensemble* e *subject* parecem oferecer um suporte mais eficiente. Ao receberem uma mensagem, *ensemble* e *subject* não realizam diretamente o serviço, apenas atuam como "distribuidores" de serviço aos seus elementos. Para utilizar os serviços proporcionados por um ensemble ou subsystem deve-se explicitamente dirigir um pedido ao *ensemble* ou ao *subsystem*, que ocultam os detalhes de implementação de seus elementos internos. Em *subject* não fica claro como os elementos de uma estrutura interagem entre si.

Outra característica importante é a implementação. Enquanto *subsystem* e *subject* são construções da análise, uma representação baseada em *ensemble* é mais robusta pois permite que *ensembles* definidos na análise possam persistir diretamente na implementação.

De uma maneira geral devemos concordar com MONARCHI & PUHR (1992) no sentido de que ainda não existe uma construção (*ensemble, subject, subsystem*, ou outras não analisadas neste artigo) mais indicada para realizar a partição de sistemas. Os guias são heurísticos, não há uma padronização quanto à notação e as unidades de partição são definidas no contexto de um método em particular, portanto deve-se conhecer as sutilezas de cada método. Por exemplo, para usar toda a extensão de *ensemble* deve-se ter familiaridade com máquina de estados, "*trigger*" e herança múltipla.

Ensemble parece ser mais completo e tende a oferecer uma melhor perspectiva ao analista. A definição de propriedades especiais impõe mais rigor, permite criar diferentes camadas de abstração da aplicação e realizar a decomposição logo no início da análise.

REFERÊNCIAS

BOOCH, G. *Object-oriented design with applications*. Redwood, Benjamin/Cummings,1991,580 p.

COAD, P.; YOURDON, D. *Object- oriented analysis*. Englewood Cliffs, Prentice-Hall, 1990,532 p.

DE CHAMPEAUX, D.;LEA, D.; FAURE, P, *Object-oriented system development.* Reading,Addison-Wesley, 1993, 532 p.

FICHMAN, R. G.; KEMERER, C. F. Object-oriented and conventional analysis and design methodologies:comparison and critique. *IEEE Computer*, v 25, nº 10, pp 22-39, Oct, 1992.

MARTIN, J. *Principles of object-oriented analysis and design*. Englewood Cliffs, Prentice-Hall, 1993, 412 p.

MONARCHI, D. E.; PUHR, G. I. A research typology for object-oriented analysis and design. *Communications of ACM*, v 35, nº 9, pp 35-47, Sep., 1992.

PITTMAN, M. Lessons learned in managing object-oriented development, *IEEE Software*, v.10, nº 1,pp 43-53,Jan.,1993.

PRESSMAN, R. S. *Software engineering: a practitioner's approach*. 3ed. New York,McGraw-Hill, 1992, 793 p

SNYDER, A. The essence of objects: concepts and terms. *IEEE Software*, v. 10, nº 1, pp 31-42, Jan.,1993.

WIRFS-BROCK, R. J.; JOHNSON, R. E. Surveying current research in object-oriented design. *Communications of ACM*, v. 3, nº 9,pp. 104-124, Sep.,1990.