Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e Computação
Departamento de Computação e Automação

Utilização Conjunta de Métodos Estruturados e Orientados a Objetos: Proposta de uma Abordagem Neo-clássica

André Felipe Dias

Orientadora:
Profª. Dr.ª Beatriz Mascia Daltrini

Outubro de 1997

Dissertação apresentada à Faculdade de Engenharia Elétrica e Computação da Universidade Estadual de Campinas, como parte dos requisitos exigidos para obtenção do título de Mestre em Engenharia Elétrica.

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

D543u

Dias, André Felipe
Utilização conjunta de métodos estruturados e orientados a objeto: proposta de uma abordagem neoclássica. / André Felipe Dias.--Campinas, SP: [s.n.], 1997.

Orientadora: Beatriz Mascia Daltrini
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Engenharia de software. 2. Análise de sistemas. 3. Programação orientada a objetos (Computação). I. Daltrini, Beatriz Mascia. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Agradecimentos

A Deus, pela intuição, inspiração e "bom senso" ; ^)

À Bia, minha orientadora, pela calma, paciência e, principalmente, por ter acreditado no meu trabalho.

À Claudia, Naur, Dino, Benito, Wellington pelas opiniões (muitas até úteis : ^)) e pela colaboração para as sessões de análise com os cartões CRC.

À Viviane, pelo carinho e compreensão.

Ao Prof. Estevão, por ter me ensinado a Análise Essencial, método sem o qual esse trabalho não teria se realizado.

À CAPES, pelo apoio financeiro.

A todos aqueles que contribuíram com opiniões para a elaboração da argumentação do trabalho, principalmente pelas opiniões contrárias às minhas.

Resumo

Esta dissertação apresenta a proposta de um método de desenvolvimento de sistemas que busca combinar o melhor que o paradigma Funcional e o paradigma de Orientação a Objetos têm a oferecer. É mostrado que a decomposição funcional de muitos sistemas é necessária, mesmo que seja utilizado um método Orientado a Objetos para o desenvolvimento, implicando a necessidade de utilização conjunta de diferentes paradigmas. As duas abordagens mais significativas de captura de requisitos funcionais são analisadas com o objetivo de se obter o método de Análise mais adequado para fornecer uma especificação às fases posteriores de desenvolvimento, baseadas no paradigma de Orientação a Objetos. O método híbrido (não formalizado) apresentado baseia-se nos métodos: Análise Essencial, cartões CRC e Responsibility Driven-Design. Conseguiu-se uma ótima integração entre os diferentes paradigmas. Alguns resultados obtidos foram: independência entre as fases de Análise e Projeto, transição suave entre as fases e importância equilibrada entre os diferentes paradigmas. Algumas heurísticas pertencentes à Análise Essencial foram estendidas ao paradigma OO, novos modelos foram introduzidos e, outros, aprimorados. O método obtido fornece uma base bastante sólida para o desenvolvimento de software.

Abstract

This thesis presents a proposal of a software development method. Its goal is to combine the best from Functional and Object-Oriented paradigms. It shows that functional decomposition is necessary, even if an OO method is used, meaning that different paradigms are necessary. The two most significant approaches for functional requirement capturing are analyzed. The best of the two was chosen for providing a specification for use in later developments stages, which are based on the OO paradigm. The hybrid method (non-formalized) is based on Essential Analysis, CRC cards and Responsibility Driven-Design. A very good integration between the paradigms was achieved. The related results are: clear distinction between Analysis and Design stages, smooth transition between stages and equilibrated importance between paradigms. Some of the Essential Analysis' heuristics are extended to the OO paradigm, new models are introduced and some existing models are improved. The method provides a solid base for software development.

Índice

Lista de Figuras

Lista de Tabelas

1. INTRODUÇÃO

A crise de software continua tão atual como há 20 anos atrás. Mesmo com as melhorias nos métodos e técnicas de engenharia de software, e ferramentas de apoio ao desenvolvimento, a demanda por software ainda cresce mais rápido do que os aumentos na produtividade de software [SOM92a].

Nesse cenário, não há espaço para o desenvolvimento de software de modo indisciplinado, baseado em intuição, inspiração e "bom-senso"[1]. O aprimoramento técnico é cada vez mais necessário para possibilitar a um gerente o controle do processo de desenvolvimento de software e fornecer ao profissional uma base sólida para construir software de alta qualidade e de um modo produtivo [PRE92].

Em meio à crise, o paradigma de Orientação a Objetos (OO) apareceu com a promessa de impulsionar o desenvolvimento de software a patamares nunca antes alcançados, através de uma melhor modelagem do problema, maior reusabilidade e manutenibilidade. Passada a euforia inicial em torno dos objetos, começa-se a ter uma visão mais realista do paradigma de orientação a objetos e seu impacto na indústria em geral [PAN95]. Stroustrup posiciona o paradigma OO da seguinte forma [MAR95] (p. *vii*):

> *Vamos deixar uma coisa clara. O Projeto Orientado a Objetos (OOD) não vai salvar o mundo do software - isso nem mesmo está ficando mais perto. Aplicações projetadas pelo OOD ainda serão difíceis de estimar, ainda serão difíceis de implementar, ainda serão difíceis de serem mantidas; e ainda terão bugs... O que o OOD fará é fornecer algumas novas ferramentas úteis que se pode empregar enquanto se projeta aplicações de software.*

Embora não seja a tão esperada "bala de prata", o paradigma OO realmente indicou um novo caminho de pesquisa a ser explorado. Objetos mostraram-se adequados à busca por reusabilidade e manutenibilidade. Os conceitos de abstração, encapsulação, ocultação de implementação[2] e classificação, que antes existiam separados, tomaram outra dimensão quando utilizados em conjunto.

---

[1] Evitou-se referenciar esse processo como uma "forma de arte" pois é depreciar a arte, que possui várias técnicas e métricas muito bem definidas.

[2] O termo comumente empregado, ocultação da informação (information hiding), passa a idéia errada de que a informação não está acessível, quando, na verdade, apenas os detalhes de implementação estão escondidos.

Por outro lado, houve um retrocesso no processo de desenvolvimento, principalmente na fase de análise do problema, trazido pelos defensores mais puristas do paradigma OO. A necessidade de mostrar que os métodos OO eram capazes de sustentar um processo completo de desenvolvimento, contribuiu para que modelos e heurísticas pertencentes aos métodos estruturados fossem desprezados. A maior parte dos métodos OO considera que o paradigma de objetos exige um tratamento e abordagem inteiramente novos de um problema. Essa não utilização trouxe um grande retrocesso, já que grande parte do trabalho de construção de um processo de desenvolvimento, principalmente o de análise, poderia ter sido reusado e estendido.

Dentre as falhas pertencentes aos métodos OO que poderiam ter sido evitadas com uma abordagem híbrida pode-se citar:

1. **precariedade na identificação dos objetos do sistema.** Ironicamente, o maior problema da Análise Orientada a Objetos (OOA) é justamente a identificação dos objetos do sistema [BOO94] [CON89] [LOY90] [SOM92b]. Essa deficiência é mais visível na precariedade com que os objetos são extraídos a partir de especificações textuais em linguagem natural, técnica utilizada pela maioria dos métodos OO (*e.g.*, OOSE, OMT, OOAD-Booch, Fusion)[3]. No entanto, a definição de um contexto funcional do sistema, trazida por um método estruturado, poderia auxiliar muito a identificação dos objetos.

2. **falha na distinção entre as fases de Análise e Projeto.** À primeira vista, essa característica pode ser considerada uma vantagem pois indica uma transição suave entre os modelos de Análise e Projeto, mantendo-se uma continuidade representacional, sem perda de informações. Entretanto, alterações em quaisquer tipos de requisitos implicam uma mudança em vários modelos, tanto de Análise quanto de Projeto, devido à inexistência de distinção entre essas fases. A localidade de alterações poderia ser ampliada definindo-se claramente a fronteira entre Análise e Projeto. Heurísticas fornecidas pela Análise Essencial, por exemplo, podem ser de grande valia na definição desses limites.

3. **inexistência de uma especificação da funcionalidade geral de um sistema.** Conforme identificado em [FIC92], objetos não representam um meio adequado de representação da funcionalidade "fim-a-fim" de um sistema. As funcionalidades desejadas pelos usuários estão espalhadas entre diversos objetos, impossibilitando uma visão mais panorâmica dessas funcionalidades. Por outro lado, os métodos estruturados possuem modelos específicos para esse fim (*e.g.*, Diagrama de Fluxos de Dados), que poderiam ser incorporados por métodos OO para esse fim.

4. **carência de heurísticas e métricas de apoio à modelagem.** A avaliação da modelagem feita por métodos OO ainda é dependente, em grande parte, da opinião de profissionais mais experientes, sendo comumente classificada em adequada ou inadequada. Mesmo propostas de particionamento por eventos (*e.g.*, *use cases*) que vem sendo incorporado por métodos OO, não aproveitam toda a experiência anterior, dando margem a dúvidas de modelagem que já tinham sido solucionadas por métodos estruturados anteriores, principalmente pela Análise Essencial [MEN84].

O desenvolvimento em termos de objetos retrocedeu em direção a uma "forma de arte", já que essas deficiências nos métodos OO impõem certos questionamentos, que os profissionais menos talentosos e

[3] No Capítulo 3, é feita uma análise mais detalhada das deficiências desse tipo de abordagem.

experientes não estão preparados para enfrentar. Jackobson, por exemplo, demonstra isso através da afirmação em relação a transição entre os modelos do OOSE durante o processo de construção do software [JAC92] (p. 126):

> *Estas transformações de modelos não são tão mecânicas como foram indicadas, pelo contrário, o desenvolvimento desses modelos é uma atividade incremental e criativa que requer muito esforço de desenvolvedores talentosos.*

A colocação de Sommerville [SOM92a] (p. 3) de que "alguns dos erros feitos por engenheiros de software nos anos 70 ainda são repetidos" serve bem para ilustrar a situação atual dos métodos OO. Diversas falhas presentes nos métodos OO podem ser solucionadas com um reaproveitamento de heurísticas dos métodos clássicos tais como a Análise Essencial.

Não se espera, com isso, sugerir que os métodos clássicos não possuam suas deficiências. Se assim o fosse, grande parte dos problemas da crise de software já teria sido resolvido e os métodos OO não teriam ganho tanta popularidade. Não, o que se deseja mostrar nesse trabalho é que as qualidades e deficiências dos métodos clássicos e OO são complementares, e uma abordagem equilibrada entre o paradigma Funcional e OO no desenvolvimento de sistema produziria um resultado muito mais completo.

A proposta de utilização conjunta de métodos clássicos e OO não é recente. Vários métodos híbridos foram propostos mas nenhum deles conseguiu grande popularidade. Entre os principais motivos desse pouco sucesso, pode-se citar:

1. **falha na justificativa de utilização conjunta.** A principal razão apontada para um desenvolvimento conjunto entre os trabalhos anteriores é a preservação dos investimentos feitos em métodos estruturados, geralmente na forma de treinamento de profissionais, aquisição de ferramentas de apoio e sistemas já desenvolvidos. Uma abordagem híbrida significaria uma mudança incremental no processo de desenvolvimento de sistemas. Atualmente essa justificativa já perdeu a força devido à grande aceitação dos métodos OO.

2. **falha na integração entre métodos Estruturados e OO.** Não foi apresentada nenhuma proposta concreta que conseguiu combinar, de modo satisfatório, os diferentes paradigmas. Muitos modelos e heurísticas estruturados ou foram modificados ou foram abandonados, de forma que o resultado final tornou-se confuso, e sempre privilegiando um dos paradigmas, em geral o paradigma OO.

3. **pressão mercadológica.** Essa é uma força que não deve ser subestimada. Com um volume de recursos relacionados a software que gira em torno de centenas de bilhões de dólares anuais só nos EUA, seria no mínimo ingênuo considerar que todos os esforços em engenharia de software, muitas vezes conflitantes, visam apenas a melhoria da área como um todo. Junto com a idéia de um novo paradigma, muitas consultorias, palestras, ferramentas de apoio, treinamentos, livros, revistas são comercializados. No caso dos métodos híbridos, o reaproveitamento de um método estruturado implicaria uma mudança apenas incremental e, portanto, menos dinheiro envolvido.

Ao invés de tentar trilhar o mesmo caminho que outros, e cair na mesma armadilha, buscou-se uma abordagem diferente neste trabalho:

1. **justificativa.** Principalmente em relação a software de negócio [PRE92], a utilização de uma abordagem funcional de um problema é, mais do que desejável, absolutamente necessária. Além de não fornecer uma representação adequada de funcionalidade do sistema em alto nível,

objetos fazem parte da visão do projetista do sistema. Para o usuário, o sistema é melhor descrito pelos serviços que oferece.

2. **integração entre os paradigmas.** A integração é possível e desejável pois as deficiências de uma abordagem podem ser anuladas pelas qualidades da outra. Mesmo se tratando de paradigmas diferentes, a transição entre um e outro pode ser suave, mantendo-se uma importância equilibrada entre os paradigmas em relação ao processo total de desenvolvimento.

3. **pressão mercadológica.** Começa a se abrir espaço para uma reconciliação entre o paradigma Funcional e o OO. Exemplo disso é a grande atenção que vem sendo dada ao modelo *use case* [JAC92], um modelo predominantemente funcional, apesar de fazer parte de um método OO. Pelo mesmo caminho, espera-se que outros trabalhos possam ser avaliados sem um posicionamento "separatista" por parte dos acadêmicos e profissionais da área.

O objetivo desse trabalho é apresentar um esboço de um método de desenvolvimento que integra, de maneira equilibrada e igualitária, um método Estruturado (Análise Essencial) e métodos OO (cartão CRC e RDD) mantendo uma distinção clara entre cada fase de desenvolvimento e, ao mesmo tempo, uma transição suave entre as fases. Espera-se, com essa junção, um processo claro, simples e controlável de desenvolvimento, que forneça a um profissional uma base sólida para construir software de alta qualidade e de um modo produtivo.

1.1 Organização da Dissertação

No Capítulo 2, alguns trabalhos mais significativos relacionados com a pesquisa são descritos mais detalhadamente. Em sua maioria, os trabalhos coletados tratam de propostas de utilização conjunta de métodos Estruturados e OO.

No Capítulo 3, é feita uma análise a respeito da utilização conjunta de métodos Estruturados e OO. São analisadas as características comumente encontradas em requisitos de sistemas de informação, que inviabilizam uma abordagem OO. Três estudos de caso são apresentados, comparando a modelagem produzida pelos *use cases* e a modelagem produzida pela Análise Essencial, fornecendo uma base para avaliação das qualidades e deficiências de cada uma. Também são sugeridas formas pelas quais os métodos estruturados podem auxiliar o desenvolvimento através de objetos.

No Capítulo 4, é feito uma análise de propostas anteriores em termos das deficiências na integração dos diferentes paradigmas. Algumas características desejadas de um método híbrido são apresentadas, juntamente com a proposta de um método visando alcançar tais características.

No Capítulo 5, são apresentadas as conclusões a respeito do trabalho e propostas de extensões futuras.

1.2 Bibliografia

[BOO94] BOOCH, G. *Object-Oriented Analysis and Design with Applications.* 2ª Ed. The Benjamin/Cummings Publishing Company Inc. 1994. pp. 589.

[CON89] CONSTANTINE, Larry L.; Object-Oriented and Structured Methods - Toward Integration, *American Programmer*, Vol. 2, No. 7-8 Summer 1989, pp. 34-40.

[FIC92] FICHMAN, Robert G. & KEMERER, Chris F. Object-Oriented and Conventional Analysis and Design Methodologies - Comparison and Critique. *Computer*, October 1992, pp. 22-39.

[JAC92] JACKOBSON, I.; Christerson, M.; Jonsson, P.; e Övergaard, G.; *Object-Oriented Software Engineering: a Use Case Driven Approach*, Reading, Massachusetts, Addison-Wesley, 1992.

[LOY90] LOY, Patrick H. A comparison of Object-Oriented and Structured Development Methods. *ACM Software Engineering. Notes.* Vol. 15, No. 1, January 1990, pp. 104-124.

[MAR95] MARTIN, Robert Cecil. Preface. In: *Designing Object-Oriented C++ Applications. Prentice Hall Inc.*, 1995, p. *vii.*

[MEN84] McMENAMIN, Sthephen M. & PALMER, John F.; *Análise Essencial de Sistemas.* São Paulo. McGraw-Hill. 1991, pp. 561. (tradução de "Essential Systems Analysis", 1984)

[PAN95] PANCAKE, Cherri M. (coord.);The Promise and the Cost of Object Technology: A Five-Year Forecast. *Communications of the ACM.* October 1995, Vol. 38, No. 10, pp. 33-49.

[PRE92] PRESSMAN, Roger S.; Software and Software Engineering. In: *Software Engineering - A Practitioner's Approach*, 3ª Ed., McGraw-Hill International Editions, 1992, Cap. 1, pp. 3-38.

[SOM92a] SOMMERVILLE, Ian. Introduction. In: *Software Engineering*, 4ª Ed., Addison-Wesley Publishing Company, 1992, cap. 1, pp. 1-20.

[SOM92b] SOMMERVILLE, Ian. Object-Oriented Design. In: *Software Engineering*, 4ª Ed, Addison-Wesley Publishing Company, 1992, cap.11, pp. 191-218.

2. REVISÃO BIBLIOGRÁFICA

Atualmente, existem mais de 30 tipos diferentes de métodos de Análise e Projeto OO. Qualquer apanhado geral da atual situação dos métodos OO deve estabelecer um critério para selecionar apenas uma parcela significativa dos métodos de interesse.

Uma forma de classificar os métodos OO é de acordo com a ênfase de qual característica do objeto é utilizada como guia para o desenvolvimento do sistema: comportamento ou estrutura. Um objeto pode ser considerado, de um modo simplista, como uma encapsulação de dados e procedimentos que manipulam esses dados. Por esse ponto de vista, ao se iniciar o projeto de um sistema baseado em objetos, deve-se escolher uma das características, já que as duas formam visões ortogonais de uma mesma abstração [BOO94a][1].

Essa diferenciação entre as abordagens pode ser claramente observada nos métodos correntes. Enquanto alguns privilegiam a estrutura dos objetos como base para todo o trabalho posterior de desenvolvimento (*e.g.*, OMT, Booch), outros enfatizam o comportamento de um objeto em detrimento de sua estrutura (*e.g.*, RDD, OBA). Os defensores dos métodos dirigidos a comportamento (behavior-driven) argumentam que a encapsulação impossibilita o acesso aos dados internos de um objeto, que só pode ser observado e manipulado através dos serviços e operações que oferece (*e.g.*,, Wirfs-Brock [WIR90]). Por outro lado, defensores dos métodos dirigidos a estrutura consideram difícil entender como considerações a respeito de dados e estrutura podem ser completamente evitados ou adiados [SOU9?].

Os métodos OO escolhidos para serem apresentados nesse capítulo são, em sua maioria, dirigidos a comportamento. O motivo dessa escolha é a melhor possibilidade de integração que esse tipo de método oferece com os métodos tradicionais de análise, em particular com o método de Análise Essencial. Um estudo mais aprofundado sobre a escolha de métodos orientados a comportamento em detrimento dos orientados a dados será apresentado no Capítulo 4.

1 Para Booch [BOO94a], as visões OO e algorítmica é que são ortogonais e não podem ser usadas simultaneamente na construção de um sistema complexo. Entretanto, por esse ponto de vista, é desconsiderada uma possível decomposição baseada numa visão estrutural do sistema e a característica de objetos serem encapsulações de estrutura e comportamento formando entidades coesas

Outro conjunto de métodos escolhidos trata da utilização conjunta de métodos Estruturados e OO de desenvolvimento, representando algumas das abordagens feitas até o momento para a elaboração de uma abordagem híbrida.

2.1 Análise Essencial (AE)

A Análise Essencial de sistemas (AE), criada em 1984 por McMenamin & Palmer [MEN84], é uma ramificação da Análise Estruturada (SA) e foi a primeira formalização do particionamento orientado a eventos para o desenvolvimento da análise de sistemas. A AE propõe uma nova forma de se enxergar o sistema e produzir sua especificação, através de uma série de heurísticas próprias tais como, definição do sistema como um conjunto de mecanismos de evento-resposta, neutralidade tecnológica, distinção entre requisitos falsos e verdadeiros presentes na análise, entre outras.

Na AE, o sistema é visto como um conjunto de mecanismos evento-resposta formado por atividades independentes e completas, que podem ser desenvolvidas, compreendidas e verificadas individualmente. Essa independência entre as atividades é fruto de uma intercomunicação mínima, feita somente através da memória essencial (Figura 2.1). A memória essencial, por sua vez, está particionada de acordo com os objetos[2] do domínio de análise do sistema.

[2]O termo objeto empregado por McMenamin & Palmer refere-se ao agrupamento de elementos de dados relacionados, encontrados nos DFD's e Depósitos de dados. Na verdade, está mais próximo do conceito de entidade usado no Modelo Entidade Relacionamento.

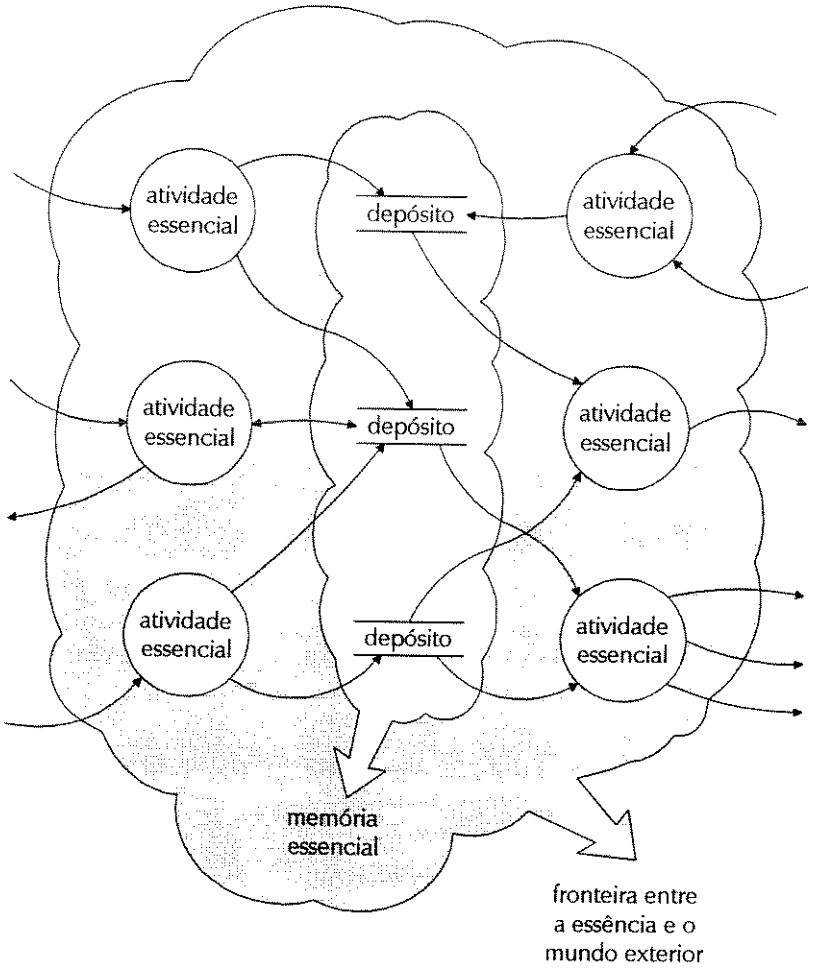


Figura 2.1 DFD típico de um sistema, após particionamento por eventos. Fonte: [MEN84].

A AE consegue produzir uma especificação precisa, completa e consistente através da distinção entre os requisitos falsos e verdadeiros de um sistema, surgidos durante o processo de análise.

Os **requisitos falsos** representam preferências irracionais ou tecnológicas, voltadas para uma determinada tecnologia de implementação. Esse tipo de requisito faz parte do domínio da solução do problema (projeto) e não deve ser considerado nessa fase pois torna o modelo de análise produzido desnecessariamente complexo.

Os **requisitos verdadeiros** são características ou capacidades que um sistema deve ter para cumprir sua finalidade, independentemente de como o sistema é implementado.

O conceito de **tecnologia perfeita** define que o analista, ao efetuar a modelagem do sistema, deve desconsiderar quaisquer tipos de limitação tecnológica que possam existir tais como capacidade de processamento, limitação de memória, velocidade de transmissão, consumo de energia, plataforma de desenvolvimento, linguagem utilizada etc.. Essa neutralidade tecnológica fornece uma base para que o analista possa capturar a **essência do sistema**: todas as características que um sistema de respostas planejadas deve possuir, se o sistema for implementado com tecnologia perfeita.

O processo de análise da AE é constituído por 4 passos [MEN84]:

1. identificar o objetivo do sistema, através do levantamento das deficiências do ambiente que ele deve suprir;

2. definir a que eventos o sistema deve responder para cumprir a funcionalidade desejada do sistema e as respectivas respostas (atividades) fundamentais (que justificam a necessidade do sistema);
3. identificar a informação que o sistema precisa através da análise do conjunto de elementos de dados produzidos pelas respostas fundamentais;
4. por último, definir as atividades responsáveis pelo ciclo de vida da informação do sistema, atividades custodiais, que ainda não foram cobertas pelas atividades anteriores.

O processo de análise é interativo e não se baseia em especificações em linguagem natural para a elaboração de seus modelos. A especificação completa do sistema é descrita pelo conjunto de mecanismos evento-resposta levantados.

A utilização da análise essencial, na fase inicial de desenvolvimento, possibilita uma especificação completa, clara e consistente do sistema. As heurísticas do método fazem com que dois analistas diferentes possam produzir especificações muito parecidas. Isso fornece uma melhor padronização e fonte de comparação da qualidade dos modelos produzidos.

Segundo McMenamin & Palmer [MEN84], através da AE os limites do sistema são muito bem definidos pelo conjunto de atividades essenciais. Como cada atividade é independente, a manutenção do sistema é muito facilitada pois fica reduzida a alteração de um evento-resposta por vez, sem o perigo de interferência entre atividades. Há, dessa forma, uma modularização das funcionalidades do sistema, que pode ser expandido ou reduzido através da inclusão ou exclusão de eventos-resposta.

A AE, através de suas heurísticas, derrubou um dos argumentos mais utilizados contra a decomposição funcional, o de que, durante o processo de desenvolvimento, as funções sofrem mais alterações que os dados sobre os quais atuam, resultando em grandes perturbações na estrutura do projeto [RUM91] [BOO94a].

O particionamento por eventos fornece uma abordagem *middle-out* do sistema. Mostra o sistema em um nível de abstração intermediário, com precisão suficiente para o entendimento do mecanismo evento-resposta, sem sobrecarregar os desenvolvedores e usuários com excesso de detalhes, respeitando o limite da mente humana em lidar com a complexidade.

Os conceitos e heurísticas trazidas pela AE contribuíram para um enorme avanço na área de Engenharia de Requisitos, além de ter influenciado diversos métodos (*e.g.*, OOSE, OBA) que apareceram posteriormente e utilizam o princípio de particionamento por eventos e outras heurísticas para o desenvolvimento de sistemas.

2.2 OOSE (JACKOBSON)

A abordagem OOSE (Object-Oriented Software Engineering), proposta por Jackobson *et al.* [JAC92], baseou-se em técnicas de programação OO, modelagem conceitual e design de blocos para a elaboração de um processo de desenvolvimento de software e de vários modelos produzidos em cada fase.

Durante a análise são produzidos os modelos de requisitos e de análise do sistema. O modelo de requisitos consiste no conjunto de *use cases* produzidos, que especificam a funcionalidade completa do sistema. Um *use case* é uma descrição textual informal de interações entre os usuários (atores) e o sistema, de acordo com o ponto de vista do usuário. A partir desses *use cases* é construído um modelo de análise

que especifica a estrutura do sistema de acordo com os relacionamentos e agrupamentos de objetos lógicos, divididos em três categorias: objetos de interface, controle e entidade[3].

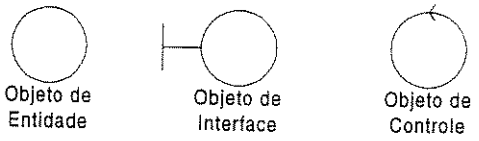


Figura 2.2 Tipos de objetos usados para estruturar o sistema no modelo de análise. Fonte: [JAC92] (p. 134).

Os objetos de interface modelam o comportamento e a informação que depende da interface do sistema. Os objetos de entidade modelam a informação do sistema que deve ser mantida por um longo tempo. Objetos de controle, por sua vez, agrupam a funcionalidade de um *use case* que não está amarrada a nenhum outro objeto, e coordenam a execução de um *use case*. Em outros métodos OO, a funcionalidade alocada nos objetos de controle é distribuída entre vários outros objetos, tornando mais difícil mudar esse comportamento.

Segundo os autores, a definição dos objetos de acordo com essas categorias, apesar de não produzir objetos correspondentes à entidades do mundo real, produz um sistema mais adaptável à mudanças. Como as mudanças mais comuns são na interface e funcionalidade do sistema, as mudanças ficam restritas, principalmente, aos objetos de interface e controle.

O conceito de subsistema é utilizado como mecanismo para reduzir a complexidade do sistema. Classes de objetos do modelo de análise são agrupadas em subsistemas de acordo com o acoplamento funcional observado entre elas.

A fase seguinte, de construção, é responsável por projetar e implementar o sistema levantado de acordo com os modelos da fase anterior. Os modelos produzidos por essa fase são o Modelo de Projeto e o Modelo de Implementação. O Modelo de Projeto procura especificar completamente os objetos levantados na análise através de outros modelos adicionais que identificam o ambiente de implementação, através da identificação dos objetos de projeto e da construção de diagramas de interação entre objetos. O Modelo de Implementação é o código fonte gerado para cada objeto de projeto.

A última fase, de teste, usa os modelos anteriores para produzir um modelo de teste que é o resultado do teste do modelo de implementação.

Todo o processo do OOSE é conduzido pelos *use cases* para verificação e validação dos resultados produzidos.

2.3 CRC Cards

CRC (Classe/Responsabilidade/Colaboradores) é uma técnica exploratória, proposta por de Beck & Cunningham [BEC89], usada na fase de OOD que busca identificar objetos, independentemente da linguagem de implementação e do ambiente em que vai operar, baseado nas três dimensões que definem o papel de um objeto no projeto: o nome da classe, suas responsabilidades e seus colaboradores.

[3] As categorias se assemelham ao padrão de projeto (*design pattern*) MVC comumente usado em Smalltalk-80 [KRA88].

Inicialmente propostos como ferramenta para auxiliar o ensino de conceitos básicos do paradigma OO, os cartões CRC acabaram mostrando-se como um modo efetivo de análise de cenários, fornecendo uma ferramenta útil de desenvolvimento pois facilita o "brainstorming" e a comunicação entre desenvolvedores [BOO94b] [BEC89].

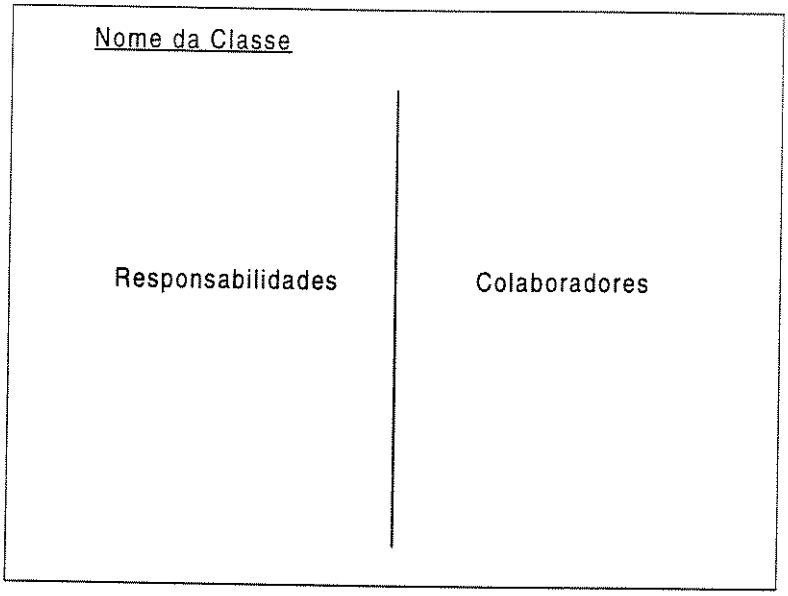


Figura 2.3 Cartão CRC básico originalmente proposto [BEC89].

Cartões CRC são cartões de papelão onde são escritas as informações de uma classe (Figura 2.3). A utilização de uma ferramenta não baseada em computador é proposital e apresenta diversas vantagens [BEC89]:

1. são baratos, portáveis, prontamente disponíveis e familiares;
2. possibilitam uma organização espacial informal indicando padrões de colaboração que auxiliam na compreensão do projeto, principalmente quando este está incompleto e pouco entendido.
3. a interação dos desenvolvedores com os cartões fornece uma noção mais "concreta" dos objetos preparando melhor os desenvolvedores sob esse ponto de vista.

A análise dos cenários é feita através de uma abordagem antropomórfica em que os participantes do projeto atuam desempenhando o papel de cada objeto representado pelos cartões, procurando desenvolver as responsabilidades estabelecidas e interagir com os objetos colaboradores (outros participantes) para cumprir suas responsabilidades.

De acordo com o levantado por Beck & Cunningham [BEC89], o projeto através dos cartões CRC proporciona um progresso a partir dos pontos conhecidos do sistema em relação aos pontos desconhecidos, diferente das abordagens top-down ou bottom-up. Durante um exercício aplicado por eles, foram observados dois grupos que chegaram essencialmente ao mesmo projeto a partir de seqüências opostas, um iniciando por *drivers* de dispositivo e outro por modelos de alto nível.

2.4 Responsibility-Driven Design (RDD) - Rebecca Wirfs-Brock

Diferentemente de métodos mais populares de desenvolvimento OO (*e.g.*, OMT, Booch) que baseiam-se principalmente na estrutura dos objetos, o RDD [WIR90] enfatiza o comportamento, consideran-

do os objetos como agentes que interagem através de um relacionamento cliente-servidor, podendo ser descritos basicamente através de suas responsabilidades, colaborações e contratos. Os conceitos de responsabilidades e colaborações foram estendidos a partir da proposta dos cartões CRC, que serviu de base para o RDD e é parte integrante e fundamental no processo de design proposto pelo método.

Responsabilidades de um objeto são todos os serviços que ele fornece para todos os contratos do qual faz parte. Incluem dois itens: o conhecimento que um objeto mantém e as ações que pode executar. A atribuição de responsabilidades segue algumas heurísticas: a inteligência do sistema deve ser o mais igualmente distribuída, as responsabilidades de cada classe devem ser declaradas de forma a generalizá-las, e responsabilidades devem ser partilhadas entre classes de objetos relacionados.

Um contrato é uma lista de requisições que um cliente pode fazer a um servidor (Figura 2.4). Ambas as partes devem honrar o contrato: o cliente só pode requisitar o que o contrato especifica e o servidor deve atender adequadamente a essas requisições.

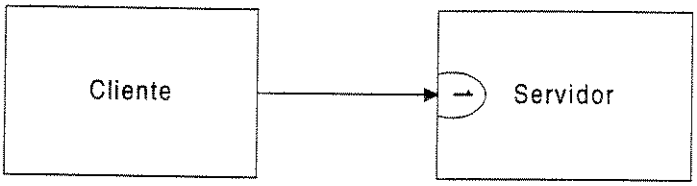


Figura 2.4 grafo de colaboração

Colaboração é o meio pelo qual um objeto requisita um serviço de outro objeto, com o objetivo de auxiliar no cumprimento de suas responsabilidades. O grafo de colaboração representa as colaborações entre os objetos que podem ocorrer durante o funcionamento do sistema.

O processo de design se inicia através de uma fase exploratória em que classes, responsabilidades e colaboradores são identificados através da análise gramatical dos requisitos do sistema, e registrados em cartões CRC (Figura 2.5). Com os cartões, exercitam-se os cenários de uso em busca de refinamento das informações coletadas.

NOME DA CLASSE	Concreta/Abstrata
SUPERCLASSES:	
SUBCLASSES	
DESCRIÇÃO:	
Responsabilidade	**Colaboração**
declaração da responsabilidade Resp1.	colaboradores para executar Resp1
Resp2.	Colaboradores para Resp2.
Etc.	etc.

Figura 2.5 Cartão CRC básico usado pelo RDD

A exploração inicial é seguida pela fase de análise detalhada, que busca fatorar as responsabilidades comuns de modo a construir uma hierarquia e otimizar as colaborações entre as classes. A definição da hierarquia de classes tem como objetivo agrupar responsabilidades, maximizar a coesão de cada classe e minimizar o acoplamento (redução do número de contratos).

O método utiliza um mecanismo de encapsulação chamado subsistema, que nada mais é do que um conjunto de classes, ou outros subsistemas, que colaboram entre si de modo a cumprir um conjunto de responsabilidades. Esse mecanismo tem como objetivo possibilitar encapsulações aninhadas, de modo a reduzir a complexidade inerente dos sistemas. O processo de identificação dos subsistemas é auxiliado pelo grafo de colaboração, na busca de padrões de interação.

Os passos de desenvolvimento são:

1. **Encontrar Classes**: a partir de uma especificação textual são extraídos nomes referentes a entidades conceituais, categorias de objetos, objetos físicos e outros, e construída uma lista de classes candidatas. Atributos de classes e possíveis superclasses também são levantadas.
2. **Encontrar responsabilidades e associá-las a classes**: através da análise dos requisitos na procura por verbos ou informação que algum objeto do sistema deve manter ou manipular. A análise de cenários é indicada como uma das formas mais úteis para se encontrar responsabilidades. As responsabilidades levantadas devem ser associadas às classes de forma que a inteligência do sistema seja igualmente distribuída, comportamentos mantidos com informação relacionada e as responsabilidades divididas entre classes relacionadas.
3. **Encontrar Colaborações**: Responsabilidades associadas a cada classe são examinadas e considera-se quais outras classes precisam colaborar para o cumprimento de cada responsabilidade.

4. **Definir Hierarquias:** Uma hierarquia de classes é definida de acordo com as responsabilidades comuns apresentadas entre classes. Contratos são definidos agrupando responsabilidades usadas pelos mesmos clientes.

5. **Definir Subsistemas:** Através do grafo de colaboração são identificados padrões comuns de colaboração que fornecem uma pista para a identificação de subsistemas. Classes em um subsistema devem fornecer um pequeno e altamente coeso conjunto de responsabilidades, além de serem independentes entre si.

6. **Definir Protocolos:** As interfaces ou assinaturas dos métodos a que cada classe responde formam os protocolos para acesso à classe. Detalhes de projeto são especificados nessa fase.

Contrastando com os métodos orientados a dados, que iniciam o processo de desenvolvimento estabelecendo a hierarquia de classes de acordo com atributos, RDD procura exercitar a simulação do comportamento e interações dos objetos que formam o sistema.

2.5 OBA (OBJECT BEHAVIOR ANALYSIS)

OBA foi proposto por Rubin e Goldberg [RUB92] para descrever como conduzir a análise de uma situação de problema. A abordagem primeiro procura entender o que acontece no sistema, definido como *comportamento do sistema.* Os comportamentos levantados são descritos em termos de interação entre vários objetos através do fornecimento e contrato de serviços entre eles. O OBA procura fornecer um processo de análise controlável e previsível, de modo que estimativas possam ser feitas para o gerenciamento do Desenvolvimento, e rastreável de modo que os resultados obtidos possam ser justificados de acordo com os objetivos e metas ao quais estão associados (rastreabilidade).

OBA é composto de cinco passos:

0. **Definição do contexto da análise.** Como ressaltado pelos autores, essa fase geralmente é negligenciada pelos métodos de análise. São identificados metas e objetivos, recursos apropriados para a análise, atividades centrais e um plano de análise preliminar. O levantamento das atividades centrais do sistema será a base para o processo de definição do script e para a divisão do trabalho entre equipes. Para a identificação dos objetivos, é sugerida a esquematização do ciclo de vida do sistema, o exame do sistema corrente ou a identificação das tarefas-chave de alto nível que devem ser executadas.

1. **Entendimento do problema através do foco no comportamento**; são especificados cenários de uso que cobrem todos os caminhos possíveis dentro das funções do sistema, semelhantes aos *use cases* usados pelo OOSE [JAC92], conforme indicado pelos autores. Os cenários de uso são capturados e expressos em uma tabela que indica o conjunto de interações entre possíveis objetos do sistema, classificados em iniciadores ou participantes, semelhantes aos objetos clientes e servidores de RDD [WIR90]. A interação entre os objetos é especificada em um contrato, também semelhante ao RDD, descrito em termos do objeto iniciador, objeto participante, uma ação (comportamento do iniciador que causou o pedido do serviço ao participante) e um serviço (fornecido pelo participante) (Figura 2.6). As pré e pós condições são descritas em termos dos estados dos objetos e servem para fazer a ligação entre os scripts que, dessa forma, fornecem uma visão abrangente de como as ações devem progredir no sistema. Outros modelos também produzidos são glossários de serviços, atributos e participantes.

Nome do Script:	Modification.1.example
Autor:	Donna
Versão:	1.0
Pré-condição:	existe (Planilha), visível(Planilha)
Pós-condição:	modificada (Planilha)
Trace:	Atividade Fundamental: Modificação

Iniciador	Ação	Participante	Serviço
Usuário	seleciona D1	Planilha	selecionar uma célula
Usuário	digita texto NEW	D1	definir conteúdo como texto
Usuário	define estilo do texto para negrito	D1	definir estilo do texto como negrito
Usuário	seleciona Linha 2	Planilha	selecionar linha
Usuário	define estilo do texto para negrito	Linha 2	definir estilo do texto como negrito

Figura 2.6 Script para a modificação em uma planilha de cálculo. Exemplo parcial do apresentado em [RUB92] (p. 53)

2. **Definição dos objetos que apresentam esse comportamento**; os possíveis objetos contidos nos scripts são analisados em busca dos objetos de análise do sistema. A seleção dos objetos é feita de acordo com os serviços oferecidos por eles. A informação coletada é organizada em cartões de modelagem de objetos (OMC), adaptados a partir dos cartões CRC [BEC89] e [WIR90] (Figura 2.7).

3. **Classificação dos objetos e identificação de seus relacionamentos.** Procura reorganizar as informações coletadas, completando as informações do OMC, criando hierarquias de classes, reorganizando serviços e classes, etc., através de técnicas de abstração e especialização. A definição de uma hierarquia de classes, diferentemente do RDD que baseia-se exclusivamente nos serviços de uma classe, utiliza também os atributos levantados para fatorar classes de objetos.

4. **Modelagem da dinâmica do sistema.** São construídos modelos de ciclo de vida dos objetos e de seqüenciamento de operações. O modelo de ciclo de vida é documentado através de StateCharts [HAR87], também usada por outro métodos (*e.g.*, OMT). Os scripts são considerados grupos de atividades que podem ser vistos como respostas a um único evento. Os estados dos objetos por sua vez, podem ser obtidos a partir das pré e pós condições declaradas nos scripts. O fluxo de controle descreve as seqüências de operações que ocorrem em resposta a um evento. Para sua documentação é sugerida a utilização de redes de Petri ou StateCharts.

OBJECT MODELING CARD		
Name of Object: Row		
Inherits From:		
Version: 1.0		
Attributes/Logical Properties	**Traces**	
Style	Modification.1.example	
Height	Modification.1.example	
Provided Services	**Traces**	
set text style to bold	Modification.1.example	
resize height	Modification.1.example	
Contracted Services	**objects**	**traces**
Card Trace		

Figura 2.7 Exemplo de cartão para a classe Row extraído do modelo apresentado em [RUB92] (p. 58)

2.6 DOMAIN OBJECT IDENTIFICATION THROUGH EVENTS AND FUNCTIONS

O trabalho de Poo e Lee [POO95] apresenta uma abordagem baseada em particionamento por eventos para a identificação dos objetos do domínio do problema. Como considera a identificação dos objetos um processo difícil, procura evitar a análise de nomes, verbos e adjetivos presentes na especificação dos requisitos para esse fim. Os objetos do domínio são identificados através do exame de eventos que caracterizam um domínio, e da informação requisitada que um sistema deve fornecer. É recomendado que o conjunto de objetos do sistema sejam identificados prematuramente no processo de desenvolvimento, com o objetivo de se evitar que decisões inadequadas feitas em fases iniciais sejam propagadas para fases posteriores refletindo na arquitetura e manutenibilidade do sistema.

1. **Listar todos os eventos e informação requisitada.** Recomendado para facilitar o processo de identificação dos objetos. Um evento é definido como um ponto instantâneo no tempo em que algo acontece. A ocorrência de um evento muda os estados dos objetos representados no sistema. Podem participar de um evento um ou mais objetos.

2. **Definição do roteiro do evento.** Para cada evento listado, um roteiro do evento é usado para descrever o evento em termos de seus participantes, efeitos, restrições e controles (Figura 2.8). Objetos são identificados incrementalmente a partir desses roteiros.

3. **Identificação de objetos a partir do roteiro do evento.** Os participantes dos eventos são considerados classes de objetos em potencial. A informação manipulada ou requisitada é então atribuída aos possíveis objetos na forma de atributos de dados. Além das propriedades estáticas dos objetos, o comportamento dinâmico dos objetos também é identificado. Para cada objeto participante em um evento, é definida uma operação para o objeto que refletirá a sua participação

no evento. As operações identificadas são operações de mudança do estado do objeto. Junto dos objetos do domínio, existem os objetos de suporte, cujos estados não são afetados por sua participação nos eventos. O objeto de suporte tem como objetivo facilitar o desenrolar do evento. No exemplo usado, de um sistema controle de empréstimos de uma biblioteca, o objeto bibliotecário é citado como exemplo de um objeto de suporte.

4. **Definir relacionamentos entre objetos.** Para cada evento, também são identificados os relacionamentos entre os objetos que participam do evento, registrados através do Diagrama de Relacionamento de Objetos (ORD), similar ao modelo de objetos do OMT. A construção de ORDs em nível de eventos permite aos analistas se concentrarem em um pequeno subconjunto dos objetos que existem no domínio da aplicação, reduzindo a complexidade associada com grandes números de classes de objetos.

O resultado do processo acima é um conjunto de objetos de domínio e de suporte, com grande parte das características estáticas, dinâmicas, relacionamentos e políticas de negócios (policy) definidos. Nas fases posteriores de desenvolvimento, a informação levantada é consolidada e refinada. Na fase de projeto, é analisado *como* o sistema é usado, e novos objetos são incluídos.

Nome do evento:	Devolver um livro
Fonte:	Sócio da Biblioteca
Significado:	Um sócio da biblioteca devolve um livro que ele pegou emprestado anteriormente. Uma multa é imposta ao empréstimo caso tenha havido atraso na devolução.
Conjuntos Participantes	
Sócio da Biblioteca	{ graduando, pós-graduando, docente, funcionário }
Livro	
Restrições Procedimentais (Policy) Pré-Evento	nenhum
Entradas:	
Livro:	número de acesso, data do empréstimo, período de empréstimo.
Sócio da Biblioteca:	número de identificação, contador de empréstimos.
Mudanças:	
1.	Sócio devolve o livro.
2.	Livro é retornado.
3.	Decremente o contador de empréstimos do Sócio
4.	Livro é marcado com a data de devolução.
5.	Data-prevista-da-entrega = Data-do-Empréstimo + período-de-empréstimo
Processamentos Pós-Evento	
1.	Se o livro está atrasado, então calcule a multa por atraso.
2.	Se o livro está reservado, então imprimir um cartão de notificação.
Políticas de Negócios (policy)	
1.	Um livro está atrasado se a data da devolução do livro é posterior a data prevista para devolução. Dias-de-atraso = data-da-devolução - data-esperada-da-devolução.

Figura 2.8 Roteiro para o evento "Devolver Livro" [POO95] (p. 612).

2.7 Object-Oriented Requirements Specification (OOS) - Bailin

Bailin [BAI89] propõe uma abordagem para a passagem da análise estruturada ao OOD. Parte de um DFD já estabelecido para a construção de um EDFD (Entity Data Flow Diagram) em que os processos que transformam entradas em saídas são organizados em funções que operam sobre os mesmos dados (entidades).

No método OOS, entidades possuem estados que podem persistir através dos diversos ciclos de execução do sistema e podem ser decompostas em sub-entidades ou funções. As funções existem somente para transformar entradas em saídas e não possuem estados entre os ciclos. Podem ser decompostas unicamente em sub-funções.

As entidades são classificadas em dois tipos: ativas e passivas. Entidades ativas executam operações importantes e devem ser detalhadas ainda na fase de análise. Entidades passivas são de menor importância, podendo ser tratadas como "caixas-pretas" até a fase de projeto, e são representadas por fluxos de dados ou depósitos de dados. Entidades passivas, entidades ativas e funções são tratadas diferentemente durante o processo de análise.

O método OOS é composto pelas seguintes etapas:

1. Identificar as entidades do domínio do problema. Processo feito a partir de nomes de processos no DFD, substantivos presentes na especificação da base de dados e especificação dos requisitos, e diagrama ER.
2. Distinguir entre entidades ativas e passivas. Realizado de acordo com a importância das operações que a entidade realiza em termos dos requisitos do sistema, que são feitas por entidades ativas, em comparação com operações cujo detalhamento pode ser adiado até a fase de projeto, executadas por entidades passivas.
3. Estabelecer fluxos de dados entre entidades ativas. É criado um EDFD nível 0 em que os processos (nós) são as entidades ativas e fluxos e depósitos de dados são entidades passivas.
4. Decompor entidades e funções em sub-entidades/funções e sub-funções. Verificar se cada entidade ou função do EDFD pode ser decomposta em níveis mais baixos. Para cada sub-entidade identificada, criar um novo EDFD e continuar o processo de decomposição. Os passos 4 a 6 devem ser executados interativamente.
5. Verificar a existência de novas entidades. A cada estágio de decomposição, deve-se considerar se novas entidades são necessárias de acordo com as novas funções introduzidas, reorganizando o EDFD se necessário.
6. Agrupar funções sob as novas entidades. Identificar todas as funções executadas por ou sobre novas entidades. Entidades passivas devem ser reclassificadas em ativas, se necessário, e o EDFD reorganizado, se necessário.
7. Associar entidades aos domínios apropriados. Cada entidade deve ser associada a um domínio e deve-se criar um ERD para cada domínio. O objetivo é reduzir o diagrama original da aplicação.

Apesar de não adotar a terminologia de OO, o método OOS utiliza princípio de encapsulação, classificação e herança. O ERD documenta a classificação dos objetos bem como a hierarquia de herança. O mapeamento das funções em entidades está bastante próximo do conceito de encapsulação.

2.8 Synthesis (Page-Jones & Weiss)

Com uma enorme bagagem cultural em desenvolvimento estruturado, Page-Jones e Weiss propuseram um método, chamado Synthesis [PAG89], com o objetivo de combinar as qualidades dos métodos estruturados com métodos OO de modo a contornar as limitações próprias de cada abordagem isoladamente.

Após o processo de análise inicial, são produzidos 3 modelos que compõem o modelo essencial de objetos do sistema:

1. Modelo Aumentado de Informação: define entidades e relacionamentos, políticas de criação, recuperação, alteração e deleção de cada um deles (ciclo de vida), e diagramas de transição de estado (STD).
2. Dicionário de Eventos para a Aplicação. Contém estímulos, respostas, restrições, mecanismos de reconhecimento de eventos etc. Nomes e verbos contidos na especificação são usados para sugerir tipos de entidades e relacionamentos no Modelo Aumentado de Informação.
3. Diagrama de Contexto da Aplicação.

Concluídos os modelos, os passos seguintes de Análise tratam de identificar os objetos, a partir das entidades e relacionamentos do Modelo Aumentado de Informação. Cada evento também pode se tornar um objeto dependendo do tipo de evento. Uma hierarquia é definida para mostrar herança e um objeto, representando o sistema, é declarado no topo da hierarquia de classe. Os objetos definidos são completados com métodos (serviços) iniciais (que cuidam do ciclo de vida do objeto), atributos e STD.

A partir de cada evento presente no Dicionário de Eventos é construído um Diagrama de Vizinhança que contém todos os objetos e mensagens envolvidos na resposta a um evento. Para cada evento, é estabelecido um objeto responsável para reconhecê-lo, um objeto para gerenciar o evento, e métodos para a resolução do evento. Cada diagrama de vizinhança pode ser dividido em níveis de abstração para a melhor representação, e são enriquecidos com textos adicionais de especificação de métodos, dicionário de mensagens, apêndices e glossários.

A fase de Projeto trata de adaptar os objetos produzidos na análise ao ambiente de operação: alocação de processadores, linguagem de programação, utilitários, formação de pacotes de objetos e codificação.

2.9 Structured Analysis Scaffolding for Object-Oriented Development

Publicado em 1989, o artigo escrito por Sully [SUL89] não propõe um novo método de desenvolvimento de sistema, mas analisa a possibilidade de utilização conjunta da abordagem convencional estruturada de desenvolvimento de software e abordagem OO, e apresenta um conjunto de heurísticas que devem ser desenvolvidas para transformar as descrições textuais e gráficas produzidas pela AE em um formato OO.

Para Sully, as abordagens Estruturada e OO podem ser integradas de modo a criar um estilo de desenvolvimento mais efetivo e sinergético. A abordagem dos métodos estruturados é mais adequada para capturar os requisitos do sistema, incentivando até mesmo a participação do usuário/cliente no processo. Não se espera que o usuário/cliente seja ciente da natureza baseada em objetos do sistema, que é responsabilidade do analista de sistemas e do projetista; assim, a abordagem OO é mais apropriada à implementação do sistema, possibilitando o reuso de componentes e maior facilidade de manutenção.

Os modelos produzidos pela AE fornecem fácil visualização e grande acessibilidade a desenvolvedores e usuários/clientes. Conseguem capturar as três dimensões básicas de um sistema através do DFD (Funcional), ERD (Estrutural) e STD (Temporal). Informação textual adicional é usada para suplementar os modelos na forma de dicionário de dados e especificações de processos (na forma de linguagem estruturada ou especificações compactas de pré e pós condições).

Uma abordagem OO pode ser obtida a partir dos modelos estruturados essenciais através da aplicação de um conjunto de heurísticas. Essas heurísticas servem para recombinar a informação dos modelos em torno de objetos. As entidades do ERD fornecem uma boa base para a escolha de objetos básicos do sistema; o STD contribui para a análise do ciclo de vida de uma entidade. O autor ressalta que os processos do DFD raramente correspondem diretamente (de 1 para 1) a métodos de objetos, mas uma correspondência pode ser feita pela desmontagem da especificação do processo e uma recolocação em um método de um objeto.

Para facilitar o processo de transformação, algumas alterações no formato e papel dos modelos estruturados são sugeridas. O ERD passa a identificar candidatos a objetos do sistema. A lista de eventos passa a ser descrita em termos de atores, ações e objetos. O modelo comportamental (STD) do sistema permanece essencialmente o mesmo, adicionando-se representações de objetos como opção. Os modelos da fase de implementação enfocam os objetos e a tecnologia que será usada.

Para realizar a transformação dos modelos o artigo enfatiza que é necessária a criação de regras nas seguintes áreas:

1. mapear Objetos do ERD em Objetos do OOD;
2. mapear o ciclo de vida do STD em Objetos do OOD;
3. "Bolhas" do DFD em métodos de objetos;
4. DFDs centrados em controle para posicionamento de objetos;
5. refinamento de objetos/reclassificação para herança;
6. reconhecimento de papéis de objetos (*e.g.*, sensores, controladores, tradutores);
7. conduzir uma transação por um caminho;
8. conservação das pré e pós condições;
9. medidas numéricas para indicar refinamento (*e.g.*, pré e pós condições de métodos, linhas de pré e pós condições da AE);
10. derivação (re-engenharia) de objetos a partir de Diagramas de Estruturas.

2.10 Estratégia de Mapeamento de Modelos de Software Orientados a Função para Modelos de Software Orientados a Objeto

No artigo elaborado por George e Carter [GEO96], é apresentada uma estratégia para o mapeamento de modelos orientados a funções em modelos orientados a objetos, além da discussão de algumas questões envolvidas no processo. Há uma grande preocupação com o suporte automatizado ao mapeamento.

A principal razão apontada pelos autores para a estratégia apresentada é a de que, em alguns ambientes, existe uma base muito grande de conhecimento em abordagens orientadas a função, tanto em sistemas bem-sucedidos desenvolvidos quanto em desenvolvedores com vasta experiência, além de existirem atividades de análise dependentes de domínio que são melhor conduzidas por métodos orientados a função. Nesses casos, é necessário o apoio de uma abordagem que seja simultaneamente orientada a funções e objetos, com uma transição fácil e consistente de modelos orientados a função para modelos orientados a objetos.

Da Análise Estruturada são usados basicamente o DFD, ERD e dicionário de dados. Não são consideradas extensões do DFD para tempo real, tais como fluxos e processos de controle, pois têm significado apenas em determinados domínios. A variação do DFD utilizado, chamado de FDFD (Flattened Data Flow Diagram), não possui níveis hierárquicos de abstração, significando que apenas processos de nível de abstração mais baixos são mostrados no diagrama. Associado com cada processo há uma descrição textual da seqüência de operações envolvidas em cada processo. De acordo com os autores, o uso do FDFD foi pega emprestada de Shumate & Keller [SHU92], para os quais o FDFD permitia ao processo iniciar com um único DFD contendo todos os requisitos do software e permite uma visão total do sistema, que pode ser estudado e contemplado de uma só vez.

A proposta também introduz um novo modelo, chamado de OSMD (Object Structure and Message Diagram), que visa representar características estáticas e dinâmicas dos objetos simultaneamente, incorporando objetos, atributos e operações, associações entre objetos, e as mensagens que podem ser trocadas entre objetos (Figura 2.9).

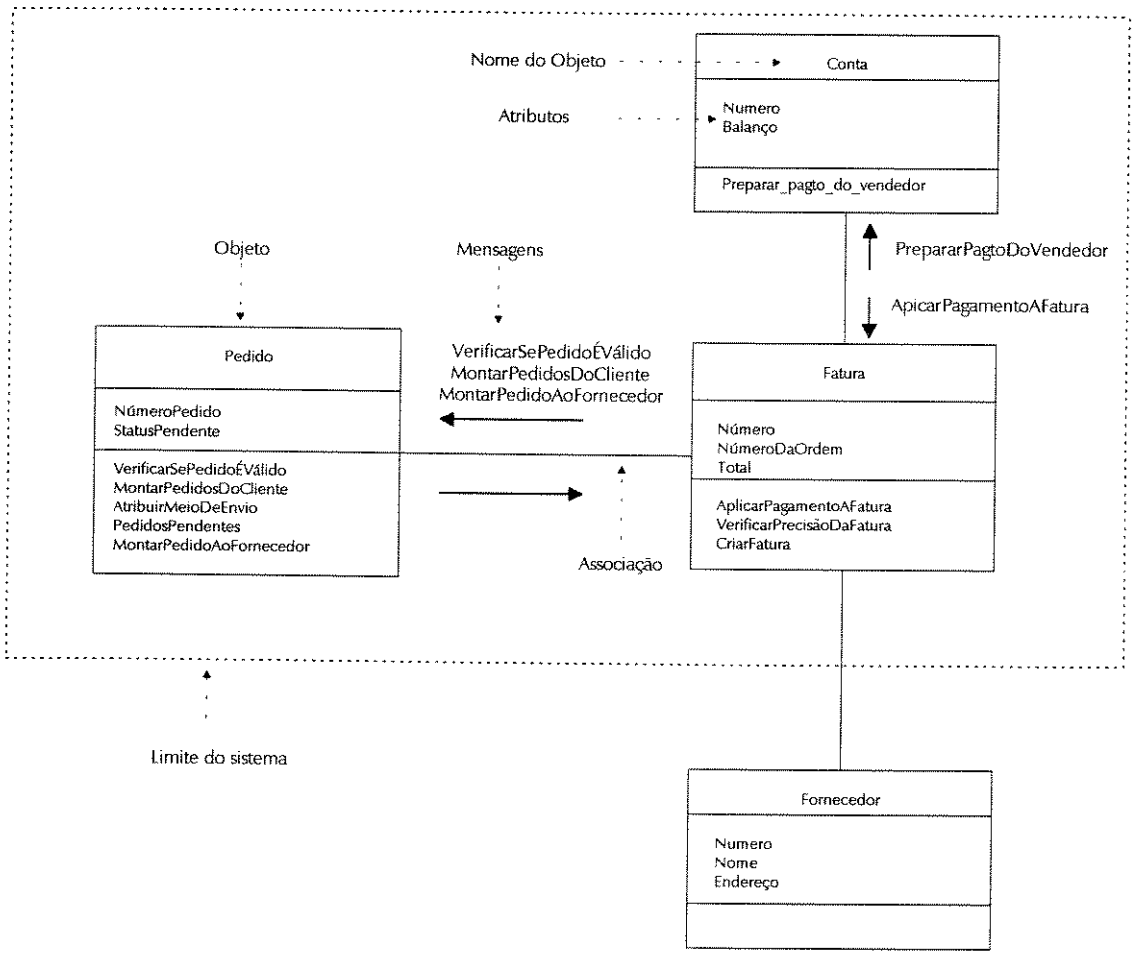


Figura 2.9 Um exemplo de diagrama OSMD. Retirado de [GEO96] (p. 58)

A estratégia de mapeamento utiliza, inicialmente, um modelo orientado a função, composto por um DFD, um diagrama entidade relacionamento e um dicionário de dados, que é traduzido em um modelo OSMD (Figura 2.10). Os autores ressaltam que, apesar de uma estratégia completamente automatizada ser irrealista, as regras e heurísticas usadas fornecem um certo grau de automação. O processo interativo é constituído por 6 fases, com saídas claras ao final de cada fase:

1. **Identificação de Objetos.** Essencialmente, a identificação dos objetos é feita a partir do mapeamento das entidades do ERD, de terminais no DFD, de depósitos de dados no DFD, e, em alguns casos, de relacionamentos no ERD que correspondem a objetos associativos. A criação de um objeto central, que represente o sistema, é opcional e desejável em determinadas circunstâncias.

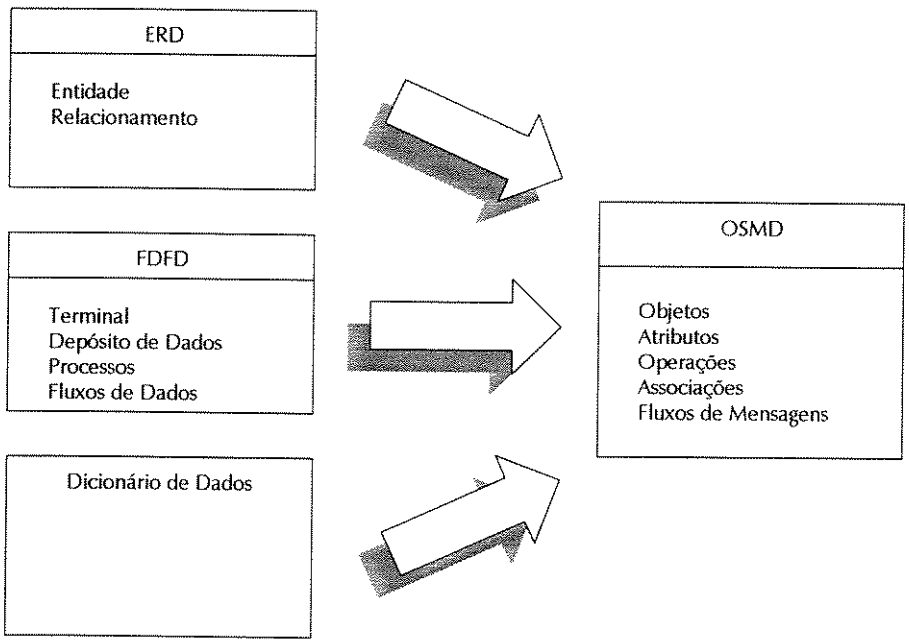


Figura 2.10 Fontes de informação para o OSMD. Retirado de [GEO96] (p. 59)

1. **Identificação dos Atributos.** Segue imediatamente a identificação dos objetos de modo que os atributos possam ser usados para auxiliar na identificação das operações na próxima fase. Os atributos são obtidos basicamente do dicionário de dados.

2. **Identificação e Atribuição de Operações.** É a fase mais crucial e mais complexa do processo de mapeamento. As funcionalidades do sistema são mapeadas para objetos do mundo na forma de operações. A estratégia é auxiliada pela utilização do FDFD e os processos definidos são mapeados diretamente em operações, não existindo o problema de lidar com decomposição de combinação de processos em operações. A identificação dos atributos, feitas na fase anterior, auxiliam a identificação das operações dos métodos na medida que, "se um processo tem um item de dado como entrada que corresponde a um atributo de um objeto, então a operação correspondente a esse processo possivelmente envolve alguma manipulação do atributo, e deve, portanto, ser definida como uma operação desse objeto" (p. 60).

3. **Identificação de Associações entre Objetos.** Até essa fase, os objetos foram definidos como "centros de processamento". Na fase de identificação de associações entre objetos, os objetos serão interligados de forma a interagirem. As associações entre os objetos podem ser interpreta-

das como caminhos de comunicação pelos quais as mensagens podem ser enviadas. A cardinalidade das associações não são identificadas. O ERD é considerado uma ótima fonte para se procurar informação a respeito de possíveis interações entre objetos.

4. **Identificação de Mensagens entre Objetos.** O objetivo dessa fase é capturar os aspectos dinâmicos da interação entre objetos através da identificação das mensagens que fluem entre os objetos. Essa fase deve vir após as duas fases anteriores pois as chamadas das operações de um objeto devem estar distribuídas por entre os objetos com os quais se associa. Como as mensagens dos objetos causa a invocação da operação correspondente no objeto alvo, cada operação do objeto é simplesmente reclassificada como uma mensagem. Esta fase lida com o exame da especificação do processo associado com a operação de modo a se decidir a partir de qual objeto a mensagem é invocada.

5. **Criação e Refinamento do OSMD.** Após completas as fases anteriores é elaborado uma representação do sistema na forma de um diagrama OSMD, que deve, então, ser refinado através da experiência e entendimento do sistema por parte do usuário. É esperada uma automatização do processo de desenho do OSMD, embora a definição dos limites do sistema e o refinamento dependa do usuário.

2.11 Bibliografia

[BAI89] BAILIN, S. C. An Object-Oriented Requirements Specification Method, *Communications of The ACM*. Vol. 32, No. 5, May 1989, pp. 608-623.

[BEC89] BECK, Kent & CUNNINGHAM, Ward. A Laboratory For Teaching Object-Oriented Thinking. *SIGPLAN Notices*. Vol. 24, No. 10, October 1989.

[BOO94a] BOOCH, G.; Complexity. In: *Object-Oriented Analysis and Design with Applications*, 2ª Ed., The Benjamin/Cummings Publishing Company Inc., 1994, cap. 4, pp. 145-168.

[BOO94b] BOOCH, G.; Classification. In: *Object-Oriented Analysis and Design with Applications*, 2ª Ed., The Benjamin/Cummings Publishing Company Inc., 1994, cap. 4, pp. 145-168.

[GEO96] GEORGE, Joseph & CARTER, Bradley, D. A strategy for Mapping from Function-Oriented Software Models to Object-Oriented Software Models. *Software Engineering Notes*, Vol. 21, No. 2, Março 1996, pp. 56-63.

[HAR87] HAREL, D. StateCharts: A visual formalism for complex systems. In: *Science of Computer Programming*. Vol. 8, No. 3, North Holland, 1987, pp. 231-274. *Apud* [RUB92].

[JAC92] JACKOBSON, I.; Christerson, M.; Jonsson, P e Övergaard, G.; *Object-Oriented Software Engineering: a Use Case Driven Approach*, Reading, Massachusetts, Addison-Wesley, 1992.

[KRA88] KRASNER, Glenn E.; POPE, Stephen T.; A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, Vol. 1, No. 3, August/September 1988, pp. 26-49.

[MEN84] McMENAMIN, Sthephen M. & PALMER, John F. *Análise Essencial de Sistemas*. São Paulo. McGraw-Hill. 1991. pp. 567 (tradução de Essential Systems Analysis, 1984)

[PAG89] PAGE-JONES, Meilir & WEISS, Steven. Synthesis: An Object-Oriented Analysis and Design Method. *American Programmer*, Vol. 2, No. 7-8 Summer 1989, pp. 64-67.

[POO95] POO, Danny C. C. & LEE, Shwu-Yi; Domain object identification through events and functions. *Information and Software Technology*, Vol. 37, No. 11, 1995, pp. 609-621.

[RUM91] RUMBAUGH, James *et al.* Comparison of Methodologies. In: *Object-Oriented Modeling and Design.* Prentice Hall International, 1991. Cap. 12, pp. 266-277.

[RUB92] RUBIN, Kenneth S., GOLDBERG, Adele. Object Behavior Analysis. *Communications of the ACM*, Vol. 35, No. 9, September 1992, pp. 48-62

[SHU92] SHUMATE, K. & KELLER, M.; *Software Specifications and Design: Disciplined Approach for Real Time Systems.* Wiley Inc., New York, 1992. *Apud* [GEO96]

[SOU9?] D'SOUZA, Desdemond. Behavior-Driven vs. Data-Driven: A Non-Issue?, s.n.t., URL http://www.iconcomp.com, 199?.

[SUL89] SULLY, Phil; Structured Analysis: Scaffolding for Object-Oriented Development. *American Programmer*, Vol. 2, No. 7-8 Summer 1989, pp. 76-82.

[WIR90] WIRFS-BROCK, Rebecca; WILKERSON, Brian & WIENER, Lauren. *Designing Object-Oriented Software.* Prentice Hall Inc. 1990, pp. 341.

3. Utilização Conjunta de Métodos Estruturados e Orientados a Objetos

A utilização conjunta de métodos Estruturados e OO, sempre que proposta, é tema de debate fervoroso. Yourdon [YOU89] chegou a dividir os metodologistas do paradigma OO em dois grupos, **revolucionários** e **sinteticistas**, de acordo com sua posição em relação aos métodos convencionais (Estruturados). Os revolucionários acreditam que a orientação a objeto representa uma mudança radical, tornando os métodos convencionais de análise e projeto ultrapassados (*e.g.,* [BOO94b] [COL94] [JAC92] [RUM91b] [SHA90]).

Os sinteticistas, por outro lado, consideram que a OO trouxe uma evolução aos princípios já existentes da engenharia de software, e consideram a integração entre os métodos Estruturados e OO, em uma única abordagem, possível e até mesmo desejável (*e.g.,* [CON89] [GEO96] [KHA89] [PAG89] [SHU91] [SUL89] [VAZ93] [WAR89]).

Esse debate foi mais acalorado no fim dos anos 80 e começo dos 90, quando o paradigma OO começava a sair do meio acadêmico e os métodos OO começavam a aparecer. Com o passar dos anos, a idéia da utilização conjunta foi perdendo força, não porque seus argumentos tivessem sido derrubados, mas principalmente porque prevaleceu a idéia de um paradigma OO revolucionário, sem vínculo com outros métodos ou paradigmas convencionais.

A inexistência de uma razão forte o suficiente para justificar uma abordagem conjunta foi a principal causa do pouco sucesso que os métodos híbridos obtiveram. A argumentação em torno da adoção de uma mudança radical no processo de desenvolvimento, representada pelos métodos puramente OO, em relação aos riscos de gerenciamento de projetos e perda do investimento já feito em treinamento de pessoal especializado, ferramentas de apoio e base de conhecimento adquirido nos métodos existentes, não foi suficiente para incentivar a escolha de um método híbrido de desenvolvimento que representasse apenas uma mudança incremental, e preservasse o investimento e conhecimento já adquiridos. Fichman & Kemerer comentam [FIC92] (p. 39):

Embora pouca evidência empírica exista para apoiar muitas das reivindicações feitas em favor da orientação a objeto, o peso da opinião formada entre muitos profissionais e acadêmicos do estado-da-arte da área favorece orientação a objetos como um "idéia melhor" para o desenvolvimento de software que abordagens convencionais.

Contudo, o aproveitamento de modelos funcionais pelos métodos OO não foi descartada completamente. Com o amadurecimento dos métodos OO nessa década, modelos e processos foram revistos, e outros introduzidos. Esse aprimoramento foi fruto da grande popularidade que o paradigma conseguiu, sendo utilizado em muitos projetos reais que possibilitaram grande retorno de opiniões a respeito dos pontos fortes e fracos de cada parte do método. Nesse processo, um modelo predominantemente funcional vem sendo incorporado por métodos OO como meio de captura dos requisitos de um sistema: o modelo *use case* [JAC92].

A aceitação desse modelo, até mesmo pelos defensores "puristas" (*e.g.,* Booch), pode ser interpretada como uma reavaliação da posição dos metodologistas OO revolucionários em relação ao paradigma Funcional, já que os modelos puramente OO mostram-se limitados e insuficientes como ferramenta de expressão dos requisitos de um sistema. Essa forma de utilização de modelos Funcionais, apesar de estar longe de ser considerada uma abordagem híbrida, pode pelo menos complementar os métodos OO.

Esse capítulo procura analisar as causas da deficiência dos métodos OO em lidar com os requisitos do sistema através da análise das ferramentas auxiliares usadas pela OOA para esse fim. Primeiramente, são apresentadas as características mais comuns dos requisitos de um sistema, revelando a importância da descrição dos serviços do sistema esperados pelo usuário, tipicamente funcionais e em um nível de abstração elevado. Como a noção de funcionalidades gerais, desassociadas a objetos, não é compatível diretamente com o paradigma OO, os métodos OO utilizam ferramentas auxiliares para lidar com isso. Dentre essas ferramentas, são analisadas duas mais representativas, com o objetivo de se avaliar a completeza e exatidão dos requisitos produzidos por cada uma delas, através de três estudos de casos. É mostrado que a Análise Essencial produz uma qualidade e quantidade de informação superior aos *use cases*, sendo mais adequada para a fase inicial de análise, restando o problema da reorganização da informação, distribuída em três dimensões diferentes, funcional (DFD), estrutural (ERD) e temporal (STD), em termos de objetos. Por fim são apresentadas algumas vantagens que podem ser trazidas em uma abordagem híbrida tais como na identificação dos objetos do sistema, na identificação dos serviços de cada objeto, no apoio ao processo de desenvolvimento do sistema, e em métricas de qualidade.

3.1 Requisitos de Um Sistema

Geralmente, um sistema de informação tem como objetivo suprir alguma deficiência no ambiente em que irá operar [MEN84b] [ZAV97] tais como a automatização de um processo manual, controle das informações produzidas, extração de informações relevantes mais eficientemente etc., visando o aumento da produtividade e eficiência. As transformações no ambiente são feitas através de ações (funções) e, dessa forma, a finalidade de um sistema pode ser descrita de acordo com os serviços (funções) que presta aos usuários, e que justificam a sua existência[1].

Sommerville [SOM92a] descreve uma especificação de requisitos como um documento estruturado que define os serviços do sistema em detalhes. Contém dois tipos de requisitos: **funcionais** e **não-**

funcionais. Os requisitos funcionais descrevem os serviços que o usuário do sistema espera. Os requisitos não-funcionais descrevem o conjunto de restrições sob as quais o sistema deve operar, e os padrões que devem ser seguidos na entrega do sistema.

Em sua maioria, os métodos OO partem de uma especificação do sistema já pronta para a identificação das classes, idéia reforçada até mesmo pela própria definição de OOA: "um método no qual os requisitos são examinados sob a perspectiva de classes e objetos encontrados no vocabulário do domínio do problema" [BOO94b] (p. 516).

Outra característica muito comum aos métodos OO é a inexistência de um modelo OO que expresse as funcionalidades gerais de um sistema. Fichman & Kemerer [FIC92], em sua avaliação, definem as funcionalidades de um sistema como "processos fim-a-fim". De acordo com este estudo, os métodos convencionais (estruturados e orientados a dados) fornecem ferramentas bem estabelecidas (*e.g.,* DFD) para modelar esse tipo de processo. Os métodos OO avaliados, por sua vez, não possuíam nenhum modelo específico para esse fim, embora partes dos processos estivessem distribuídos entre os objetos dos sistema através de serviços, operações, responsabilidades etc.. Vale ressaltar a afirmação feita por Fichman & Kemerer a respeito da causa dessa deficiência (p. 38):

> *Essa falha de suporte ao processos globais não é surpresa, já que o conceito de processo global, não subordinado a nenhum objeto individual, parece estar em desacordo com o espírito da orientação a objeto.*

Para a especificação dos requisitos e a posterior identificação dos objetos, os métodos OO utilizam ferramentas auxiliares. Booch sugere diversas abordagens que podem ser usadas como ponto de partida para OOA [BOO94b]:

1. **Abordagens Clássicas**: buscam objetos a partir de papéis mais comuns que estes exercem no sistema: coisas tangíveis, papéis, eventos, interações, pessoas, lugares, conceitos etc..
2. **Análise Comportamental**: os objetos são agrupados de acordo com padrões comuns de comportamento. São citados os métodos RDD e OBA como representantes dessa categoria.
3. **Análise de Domínio**: procura identificar classes e objetos comuns a um determinado domínio, através da análise de similaridades e diferenças com sistemas existentes.
4. **Análise *Use Case***: segundo Booch, é uma prática que pode ser usada em conjunto com as abordagens anteriores de modo a conduzir o processo de análise de modo significativo. É uma forma particular ou padrão de uso, um cenário que inicia com algum usuário do sistema iniciando alguma transação ou seqüência de eventos inter-relacionados.
5. **Cartões CRC**: considerado um modo efetivo de análise de cenários.
6. **Descrição Textual Informal**: apesar de ser muito próxima aos *use cases*, Booch considera essa abordagem inadequada pois a linguagem informal não é muito precisa e os resultados dependem da habilidade de quem escreve os requisitos.

1 Sommerville considera que a nossa visão "natural" de muitos sistemas é funcional, e a adaptação para um ponto de vista OO é difícil e leva a um problema de identificação de objetos de um sistema [SOM92b].

7. **Análise Estruturada**: possível *front end* para OOD. Booch aponta essa técnica atraente apenas devido ao grande número de analistas habilitados em Análise Estruturada e o número de ferramentas existentes.

Das abordagens sugeridas, apenas a 4, 6 e 7 conseguem definir os requisitos de um sistema com variados graus de sucesso a partir do zero, *i.e.*, sem que haja qualquer tipo de definição prévia dos requisitos do sistema a ser desenvolvido. As demais abordagens necessitam de algum tipo de especificação prévia do sistema para basear a análise dos objetos, geralmente usando as abordagens 4 ou 6, e em alguns casos a abordagem 7.

A Análise de Domínio, por exemplo, necessita que uma especificação do sistema seja produzida para que o domínio ao qual o sistema pertence seja definido. O Cartão CRC, de acordo com a proposta de Beck & Cunningham [BEC89], é uma ferramenta de OOD para identificação dos objetos, e não de especificação dos requisitos, que também precisam já estar definidos de alguma forma. Em seu artigo, Beck & Cunningham citam um exemplo em que foi utilizado um DFD como base para a identificação posterior de objetos, responsabilidades e colaboradores.

Outra constatação interessante, é que as abordagens 4, 6 e 7, usadas pelos métodos OO, descrevem basicamente requisitos funcionais[2], que são posteriormente mapeados em objetos, atributos e operações, através de uma determinada heurística (*e.g.*, análise sintática da especificação textual), descaracterizando o paradigma OO como suficiente para a especificação das funcionalidades globais do sistema[3].

A funcionalidade dos requisitos é comumente encontrada nas especificações pois os usuários tendem a expressar suas necessidades em termos de funcionalidades esperadas do sistema, e não pelos objetos que o compõem [SOM92b]. A visão do sistema através dos objetos pertence aos projetistas do sistema e não aos usuários, que não devem ser obrigados a entendê-la [SUL89] [CHEN94].

Essa diferença entre as visões do usuário e do projetista sugere que abordagens diferentes sejam usadas para capturar cada tipo de visão, e uma estratégia definida para reorganizar a informação presente na visão do usuário para a visão do projetista. Dessa forma duas questões são levantadas:

1. Qual a melhor forma de se capturar os requisitos funcionais do sistema de maneira a se conseguir uma melhor participação do usuário durante o processo?
2. Como reorganizar a informação levantada, em termos de objetos que compõem o sistema, para as fases posteriores de desenvolvimento?

3.1.1 Problemas da Utilização de Requisitos em Forma de Descrições Textuais Informais

A utilização de descrições textuais informais, como definição dos requisitos de um sistema para a identificação dos objetos, foi proposta primeiramente por Abbott [ABB80] e depois adaptado por Booch em sua chamada "estratégia informal" [BER95]. Consistia de um parágrafo descrevendo a solução para o problema definido. Os requisitos do sistema eram expressos no vocabulário do domínio da aplicação enquanto objetos, métodos e atributos eram identificados através dos nomes, verbos e adjetivos presentes

---

[2] Berard [BER95] considera que "*use cases*, da forma como são comumente descritos e usados, são quase, se não inteiramente, de natureza funcional". Jacobson *et al.* sugerem que o modelo de requisitos, composto pelo conjunto de *use cases*, pode ser usado como especificação de requisitos funcionais [JAC92] (pp. 161-162).

[3] Para simplificar ainda mais as alternativas de abordagens para OOA, pode-se considerar os *use cases* como uma evolução da descrição textual informal, o que resulta em apenas 2 alternativas: *use cases* e análise estruturada.

nos requisitos. Essa abordagem, que foi e ainda é utilizada por muitos métodos de OOA, sofreu muitas críticas por apresentar diversas falhas inerentes à linguagem natural [BOO94b] [GAN79] [RUM91a] [SHA90] [SOM92a]:

1. **redundância**: a mesma informação pode estar presente em lugares diferentes, em formas diferentes;
2. **imprecisão e ambigüidade**: diferentes leitores interpretam o mesmo texto de forma diferente;
3. **difícil avaliação da qualidade**: não é possível avaliar se o documento está completo e consistente;
4. **documentos muito extensos**: os documentos são geralmente muito longos para serem lidos e assimilados;
5. **excesso de abrangência**: descreve requisitos verdadeiros, decisões de implementação, política da empresa, restrições de performance etc., no mesmo documento;
6. **descrição influenciada**: a qualidade da descrição depende principalmente das pessoas que a formularam, variando de acordo com seus níveis de entendimento do problema e capacidade de expressar suas necessidades. As especificações em linguagem natural assumem que o leitor possui um entendimento básico do domínio do problema, deixando de fora, portanto, o conhecimento "óbvio" da aplicação [SOM92b]. Dessa forma, a probabilidade de omissão de funcionalidades e características do sistema é muito maior.

Segundo Berard [BER95], a abordagem de Booch/Abbott sofreu muita resistência pelos desenvolvedores devido à informalidade da descrição em parágrafos. Muitos sugeriram um maior rigor através do uso de gráficos (*e.g.*, DFD, STD) ou técnicas formais. Berard ainda cita: "contudo, hoje em dia, o uso da mesma técnica, sob o nome de 'use case', é amplamente popular".

Realmente, a transição da especificação textual informal para os *use cases* não foi acompanhada por um maior rigor que evitasse os problemas anteriormente citados. Apenas uma notação gráfica bem simples foi introduzida (Figura 3.1), e alguns passos foram propostos para a criação dos *use cases* [JAC92]:

1. **Identificação dos Atores.** Atores modelam qualquer coisa que precise trocar informação com o sistema. Pode ser humano ou mesmo outro sistema. Para o levantamento dos atores, começa-se verificando a quem o sistema pretende servir. À medida que as funções importantes do sistema são investigadas, mais atores são identificados.
2. **Identificação dos *Use Cases*.** Para cada curso completo de eventos iniciados por um ator, é identificado um *use case*. É sugerida a leitura dos requisitos do sistema sob o ponto de vista dos atores e a realização de perguntas como:
    - Quais as tarefas principais de cada ator?
    - O ator deverá ler/escrever/mudar alguma informação do sistema?
    - O ator deverá informar ao sistema a respeito de mudanças externas?
    - O ator deseja ser informado sobre mudanças inesperadas?
3. **Descrição de Interfaces.** Deve ser descrita a interação do usuário com a interface do sistema. Caso sejam protocolos de hardware, podem ser feitas referências a vários padrões. O objetivo dessas descrições é eliminar várias possibilidades de mal entendimentos.

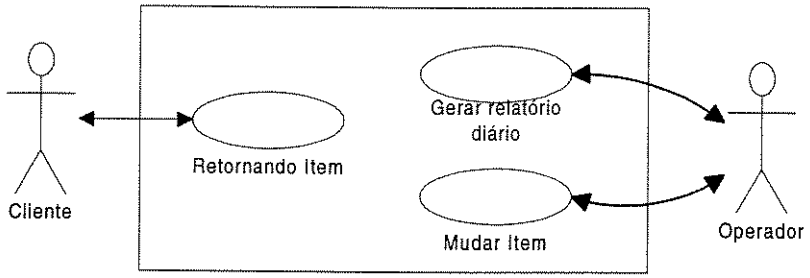


Figura 3.1 Exemplo de modelo *use case* extraído de [JAC92] (p. 156). O sistema é limitado por uma caixa. Cada ator é representado por uma pessoa fora da caixa e cada *use case* é representado por uma elipse dentro da caixa.

Berard aponta vários outros problemas, além dos referentes à descrição textual, presentes na abordagem *use case*, tais como [BER95]:

- *use cases* são descrições funcionais de interações entre o usuário e o sistema. Como esses requisitos são usados para gerar uma solução OO, existem problemas de tradução da informação, que está localizada em torno de funções, para objetos.
- apesar de ser dito que o conjunto de todos os *use cases* descrevem toda a funcionalidade do sistema, é muito difícil demonstrar que toda a funcionalidade foi tratada através do conjunto de *use cases* utilizados.
- mesmo com poucos *use cases* catalogados, é possível que dois ou mais sejam conflitantes. O conflito entre *use cases* diferentes é muito difícil de ser localizado. À medida que seu número aumenta, também aumenta a probabilidade de contradições entre eles.
- à medida que mudanças são introduzidas, torna-se impraticável a manutenção do conjunto de *use cases* de forma que continuem consistentes e válidos.
- não há um senso comum em relação ao nível de detalhe que um *use case* deve possuir. Pode haver ›s em que existam tão poucos detalhes que a especificação seja ambígua. Em outros, o excesso de ..lhes faz com que mesmo a menor alteração dos requisitos exija que o *use case* seja re-escrito.
- *use cases* descrevem requisitos de análise, projeto e restrições de implementação na mesma especificação, dificultando o discernimento entre a funcionalidade esperada do sistema e decisões de projeto.

3.2 Comparação entre o Particionamento por Eventos da AE e Use Cases (OOSE)

Tanto a Análise Essencial quanto os *use cases* buscam especificar os serviços que um determinado sistema deve oferecer, através de um particionamento dirigido a eventos, diferindo principalmente no ponto de vista sob os quais os eventos e respostas são especificados: enquanto a AE usa o ponto de vista do sistema e do usuário, os *use cases* capturam a resposta a um evento apenas sob a óptica do usuário.

Enquanto os *use cases* consideram um evento qualquer transação executada entre um ator e o sistema em um diálogo a AE considera como evento uma mudança no ambiente do sistema e o conjunto de ações executadas pelo sistema como a resposta a esse evento. Dessa forma, pode-se considerar cada *use case* equivalente a um mecanismo de evento-resposta (atividade essencial) da AE e vice-versa.

Ambas as abordagens procuram fornecer uma especificação que sirva para a comunicação com os usuários e para a utilização das fases posteriores de desenvolvimento, sem abrir mão da completeza e exatidão das informações levantadas[4].

É necessário ressaltar que a comparação entre um método (AE) e um modelo (*use cases*) se deve ao fato de ambos poderem ser usados como ponto de partida para as mesmas etapas posteriores de desenvolvimento OO[5], tais como na identificação de classes, atributos, serviços e responsabilidades de objetos.

Como será visto através dos estudos de caso das subseções a seguir, a AE produz uma quantidade e qualidade de informação superior aos *use cases*.

3.2.1 Caso 1: Sistema de Controle de Máquina de Reciclagem

O sistema exemplo, usado por Jackobson para explanação dos conceitos por ele apresentados em seu livro [JAC92], controla uma máquina de reciclagem de garrafas, latas e engradados. O sistema possui 3 *use cases* que podem ser resumidos da seguinte forma [JAC92] (p. 156):

1. Retornando Item: iniciado pelo usuário quando este deseja retornar garrafas, latas ou engradados.
2. Gerar Relatório Diário: iniciado pelo operador da máquina quando deseja imprimir informação relativa aos itens retornados, depositados no dia.
3. Mudar Item: usado pelo operador para mudar informação no sistema.

Para ilustrar o fluxo de descrição de um *use case* típico[6], Jackobson escolhe detalhar o evento "Retornando item" e por isso, a análise do caso 1 será feita sobre esse evento.

3.2.1.1 Análise do Evento segundo os Use Cases

O detalhamento feito por Jackobson para o *use case* "Retornando item" [JAC92] (p. 157-158) descreve a seqüência de interações como:

1. O curso dos eventos inicia quando o usuário pressiona o botão "iniciar" no painel. Os sensores do painel são então ativados.
2. Agora, o usuário pode então retornar os itens via painel do usuário. Os sensores informam ao sistema que um objeto foi inserido e também "medem" o item depositado e retornam o resultado ao sistema.
3. O sistema usa o resultado medido para determinar o tipo de item depositado: lata, garrafa ou engradado.

---

[4] Especificações formais, apesar de fornecer uma ferramenta de especificação mais rigorosa de requisitos, é inadequada para ser usada diretamente na comunicação com o usuário.

[5] Apesar de outros modelos fazerem parte do OOSE [JAC92], apenas os *use cases* tem sido incorporados por outros métodos OO (*e.g.*, Fusion, Catalysis, UML, OOAD-Booch) como ferramenta de captura dos requisitos.

[6] Esse trabalho considera que a especificação produzida por Jackobson retrata uma descrição típica encontrada nas especificações produzidas por *use cases* em geral.

4. O total diário é incrementado com o tipo de item de depósito recebido, da mesma forma que o número de itens de depósito retornados com o determinado tipo de item que o usuário retornou.

5. Quando o usuário tiver retornado todos os itens de depósito, ele pede o recibo pressionando o botão "recibo" no painel do usuário.

6. O sistema compila a informação que será impressa no recibo. Para cada tipo de item de depósito, seu valor de retorno e número de itens por cliente é extraído.

7. A informação é impressa, com uma nova linha para cada item, pela impressora de recibos.

8. Finalmente, o total geral para todos os itens de depósito retornados é extraído pelo sistema e impresso pela impressora de recibo.

O *use case* descrito acima ainda pode ser estendido de forma a incorporar um outro *use case*. Uma extensão é usada para adicionar ou mudar o comportamento de um *use case* [JAC92]. No exemplo usado, a extensão descreve o comportamento do sistema em caso de ocorrer um problema no recebimento dos itens depositados (p. 159):

9. "Item está entalado" é inserido em "Retornando item" quando o usuário deposita um item que fica entalado na máquina de reciclagem. O operador é chamado e o usuário não pode inserir mais itens até que o operador informe a ele que a máquina pode ser usada novamente.

3.2.1.2 Análise do Evento segundo a Análise Essencial

Para a Análise Essencial, um nome provável para o mesmo evento seria "Cliente retorna itens recicláveis". A troca do nome é para caracterizar o evento externo para o qual há uma resposta planejada do sistema[7]. O nome anterior "Retornando Item" reforça a idéia de um processo contínuo de acompanhamento da atividade feita pelo cliente, e que está de acordo com a filosofia dos *use cases*, mas não com a AE.

Em um primeiro momento, através de um DFD nível 1, procura-se mostrar a resposta básica apre- pelo sistema (Figura 3.2):

[7] O sistema de respostas planejadas produzido pela AE responde a dois tipos de eventos previstos: *eventos externos*, que são iniciados por entidades no ambiente, e *eventos temporais*, que são iniciados pela passagem do tempo [MEN84a].

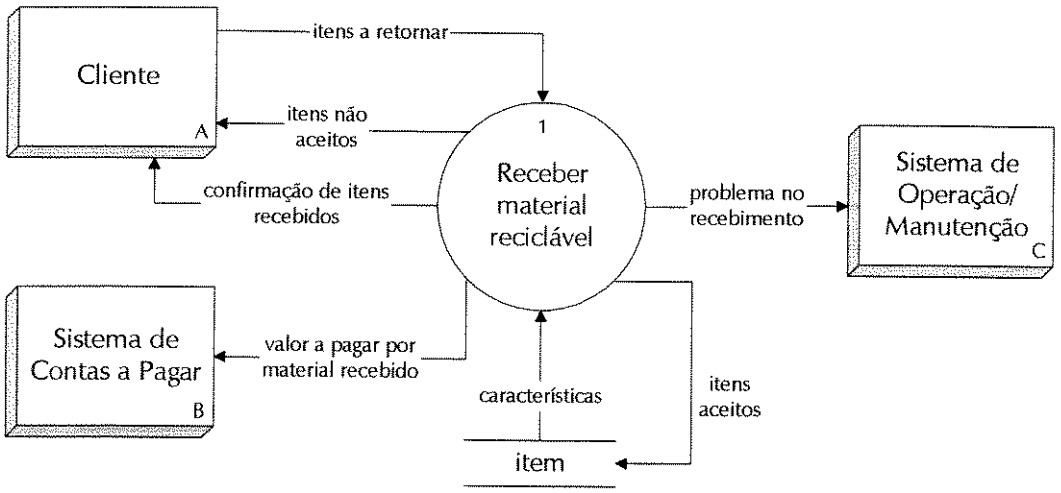


Figura 3.2 DFD nível 1 para o evento "Usuário retorna itens recicláveis"

Os elementos e estruturas de dados, processos e entidades contidos no DFD são especificados com mais detalhes no dicionário de dados. Por exemplo (Tabela 3.1):

Tabela 3.1 Exemplo de entradas para o Dicionário de Dados do Caso 1

Elemento/Estrutura de Dados	Descrição/Composição
itens a retornar	{ tipo do item + peso do item }
itens não aceitos	{ tipo do item + motivo }
tipo do item	{ garrafa | lata | engradado | tipo-não-identificado }
confirmação de itens recebidos	{{ tipo do item + peso total + valor por unidade de peso + valor total a receber por tipo de item } + valor total da operação + data da operação + número da operação}
valor a pagar por material recebido	{ número da operação + data da operação + valor total da operação }

Além do dicionário de dados, a Análise Essencial utiliza o Modelo Entidade Relacionamento (MER ou ERD) e Diagramas de Transição de Estado (STD) para completar a especificação dos requisitos do sistema.

Níveis de detalhamento podem ser conseguidos naturalmente através do particionamento *top-down* que os níveis de DFDs fornecem. Em um nível inferior, as funções são mais especializadas e novos fluxos, muito detalhados para um nível mais elevado de abstração, podem ser documentados (Figura 3.3).

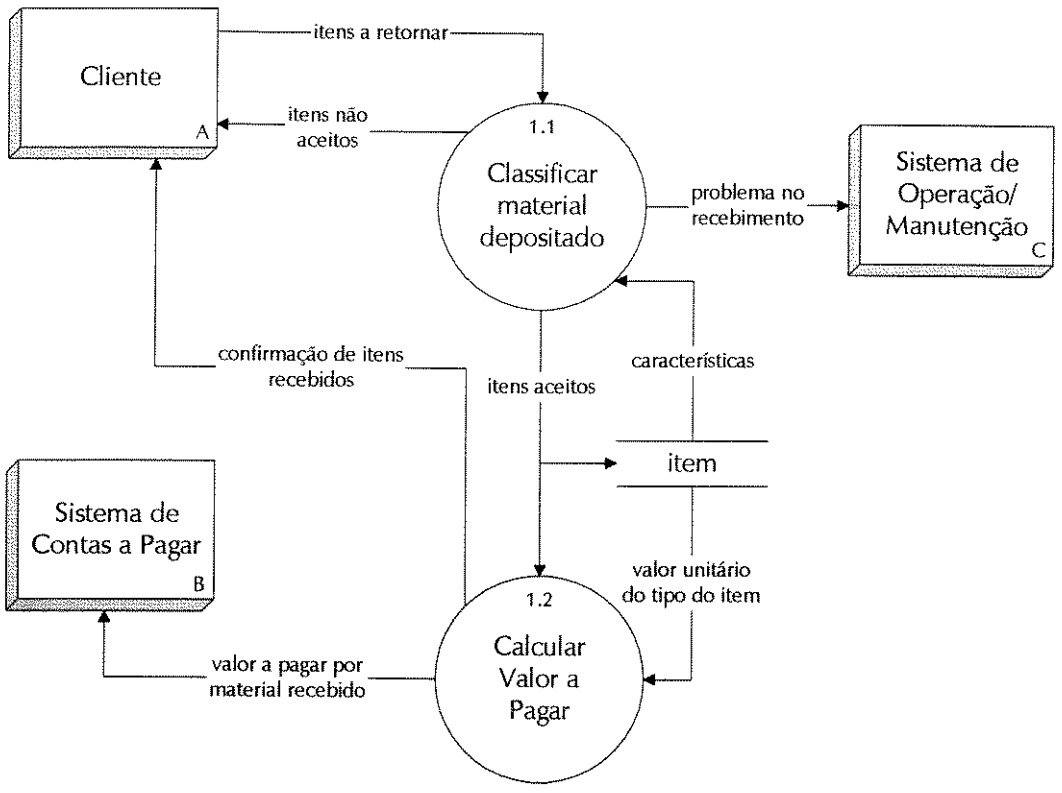


Figura 3.3 DFD nível 2 para o evento "Cliente retorna itens recicláveis".

3.2.1.3 Comparação entre Análise Essencial e Use Cases para o Caso 1

A análise do evento segundo o *use case* "Retornando item" apresentou diversas falhas já citadas na seção anterior:

- Na mesma declaração, parágrafo 2 da especificação, foram encontrados requisitos de análise (necessidade de identificação do item retornado) e implementação (o retorno é feito via um painel de controle; um sensor informa e mede; o recibo é obtido pressionando-se o botão recibo, a informação é impressa pela impressora de recibos).
- mesmo analisando apenas um *use case* de exemplo, é possível se ter uma idéia da dificuldade que seria refletir uma alteração nos requisitos em um conjunto de *use cases* do sistema e mantê-los consistentes e válidos. Descrições textuais não facilitam a identificação das alterações e o impacto dessas alterações por outras partes da especificação.
- não fica claro o nível necessário de detalhe nem a abrangência que um *use case* deve tratar. No exemplo, foram especificados não só a funcionalidade do sistema mas também interação com o usuário, restrições de projeto e *lay-out* de relatório. Não há um discernimento claro de o que faz parte da análise, projeto e implementação, dificultando as fases posteriores de desenvolvimento, que se basearão nesse documento.

Diferentemente dos *use cases*, a AE apresentou uma resposta clara e consistente ao evento:

- O nível de detalhe da especificação pôde ser dividido entre os níveis do DFD e os modelos produzidos (MER, STD[8] e Dicionário de Dados), sem sobrecarga cognitiva aos usuários e desenvolvedores.
- A especificação resultante é livre de requisitos de projeto e implementação graças aos conceitos de requisitos falsos e verdadeiros, e tecnologia perfeita. Mudanças na forma de implementação do sistema, tais como, *lay-out* do painel e relatórios, forma de entrada dos itens etc. não acarretam alterações nos modelos de análise do sistema mas apenas nos modelos das fases posteriores de desenvolvimento.
- A independência entre as respostas de um sistema faz com que alterações nas especificações sejam facilmente acomodadas, mantendo consistente o conjunto de requisitos levantados.

3.2.2 Caso 2: Evento: "Caixa de Supermercado Registra Venda de Produtos"

McMenamim & Palmer definem dois testes baseados numa definição mais formal pelos quais cada atividade modelada tem que passar para ser qualificada como atividade essencial [MEN84a] (pp. 80-81):

1. *A atividade tem de conter todas as ações que seriam executadas como resposta a um e apenas um evento se o sistema fosse implementado com tecnologia perfeita.*
2. *Quando todas as atividades que formam a atividade essencial tiverem sido executadas, o sistema terá de ficar inativo até que o evento em questão venha a ocorrer novamente ou até que venha a ocorrer um evento diferente. Se o sistema pode começar imediatamente a executar outra atividade sem que nenhum outro evento tenha ocorrido, então a atividade essencial sendo testada não contém todas as atividades que deveria conter.*

Cada atividade essencial produzida de acordo com esses princípios contém a resposta completa a um evento, e toda a comunicação direta e imediata entre as sub-atividades se encontra dentro de cada atividade [MEN84a]. A intercomunicação entre atividades, que é toda a informação que deve ser lembrada entre eventos, é mínima, feita pela memória essencial do sistema. Como conseqüência, consegue-se um alto grau de modularidade funcional do sistema.

Através do estudo de caso seguinte, pretende-se mostrar que a modelagem *use case*, devido à inexistência de diretivas de modelagem similares às da AE, não garante que um mecanismo evento-resposta tenha sido modelado completamente.

O problema é agravado pelo fato de a modelagem de cada *use case* ser feita de acordo com o ponto de vista do usuário do sistema (a análise essencial também especifica os mecanismos do ponto de vista do sistema). Em um grande número de casos, um evento percebido pelo usuário é apenas parte da resposta que o sistema deve fornecer para ser considerado completo.

Como estudo de caso será feita a análise da resposta produzida por um **Sistema de Controle de Estoque de um Supermercado** para uma venda registrada por um Caixa.

3.2.2.1 Resposta Produzida de acordo com *Use cases*

Do ponto de vista do Caixa do Supermercado (Figura 3.4), basta que o Sistema de Controle de Estoque informe a descrição e o preço de venda unitário e total para cada produto relacionado para que a

[8] Embora não tenha sido enfatizado pela Análise Essencial e nem por este trabalho, a descrição do aspecto comportamental do sistema, incluindo a interação do sistema com as entidades externas, pode ser feita através do STD. No Capítulo 5, é sugerido, como extensão, um estudo mais profundo da descrição desse aspecto comportamental de sistemas dentro do método proposto.

resposta do sistema seja considerada completa. A descrição em *use case* costuma ir um pouco mais além, descrevendo a interação do usuário com a interface que ele encontrará no sistema [JAC92]. Essa mistura de requisitos dá a impressão (errada) ainda maior de que a resposta do sistema foi completamente analisada.

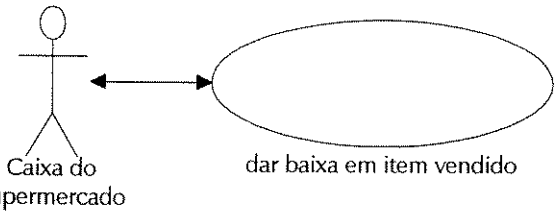


Figura 3.4 Resposta completa do sistema sob o ponto de vista do usuário, de acordo com as heurísticas da modelagem *use case*.

Entretanto, para a manutenção do funcionamento desejado do sistema, é necessário que a reposição do estoque seja feita em algum momento. Já que a resposta anterior está completa de acordo com a modelagem *use case*, é mais provável que a reposição do estoque seja tratada em um segundo momento, de acordo com uma das possibilidades abaixo:

1. Um agente almoxarife, responsável pelo acompanhamento do estoque, solicita um pedido de reposição a um fornecedor ao verificar que o estoque se posiciona abaixo de um determinado nível (Figura 3.5). Apesar de estar de acordo com o ponto de vista do usuário, o sistema fica subtraído de sua função principal, e que justifica a sua existência, de fornecer um controle automatizado do estoque, cumprindo apenas um papel secundário de intermediador.

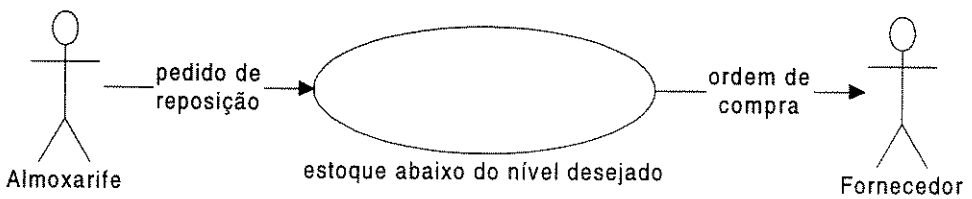


Figura 3.5 *use case* de acordo com o ponto de vista do usuário mas que retira do sistema uma das funcionalidades esperadas de controle automatizado do estoque.

1. Uma amostragem periódica poderia ser feita em relação ao nível do estoque para verificar se há algum caso de material abaixo do desejado, e então, faz-se o pedido de reposição (Figura 3.6). Há dois problemas nessa abordagem. Primeiro, não está de acordo com o ponto de vista do usuário do sistema, *i.e.*, o sistema gera uma resposta aparentemente espontânea, sem a solicitação por parte de um usuário. Segundo, a amostragem periódica representa uma complexidade absolutamente desnecessária do sistema (a amostragem periódica), que é propagada para fases posteriores de desenvolvimento do sistema.

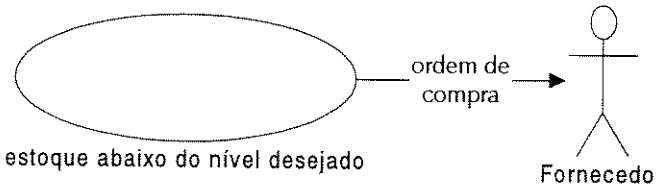


Figura 3.6 *use case* para reposição de estoque inicia espontaneamente, sem a solicitação de um ator.

3.2.2.2 Resposta Produzida pela AE

Do ponto de vista do usuário, a relação de produtos vendidos faz parte do evento iniciador. O envio da descrição e o preço dos produtos vendidos é apenas parte da resposta que o sistema deve fornecer (Figura 3.7). Para que a atividade essencial esteja completa, e o sistema possa ficar inativo após sua execução[9], é preciso que a verificação da necessidade de reposição de estoque também seja parte da resposta ao evento (ponto de vista do sistema). Dessa forma, o pedido de reposição pode ser feito logo após o recebimento de uma solicitação de material que posicione o estoque abaixo de um nível de reposição adequado.

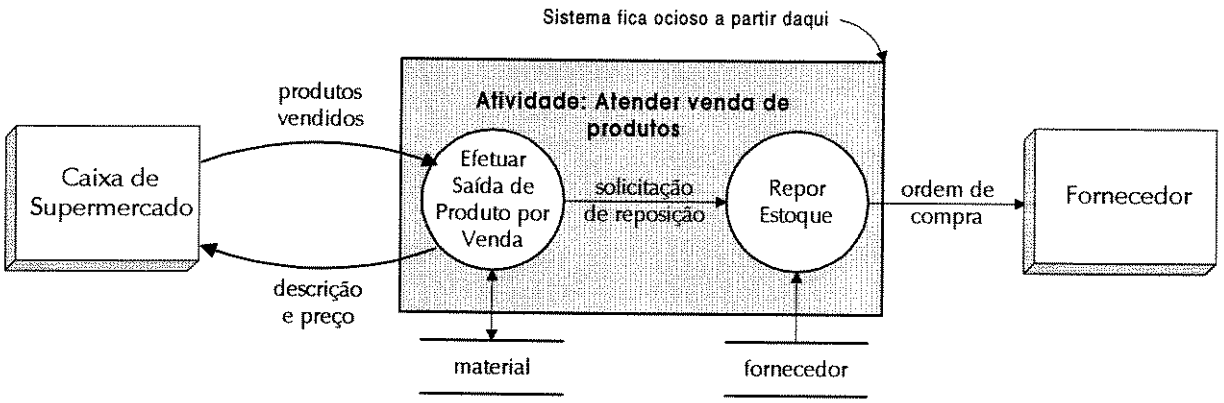


Figura 3.7 Resposta completa do sistema sob o ponto de vista do sistema (AE). Baseado em figura de [MEN84a]

3.2.2.3 Comparação entre Análise Essencial e *Use cases* para o Caso 2

Apesar de parecer "óbvia" a necessidade de reposição de estoque para um sistema de controle de estoque, trata-se de uma típica atividade interna com o objetivo de manter o funcionamento correto do sistema, atividade esta que nem sempre é visível do ponto de vista do usuário, mas apenas do ponto de vista do sistema.

A observação do sistema sob o ponto de vista do usuário não garante que cada atividade seja modelada completamente, pois o que é um evento para o usuário, é apenas parte da resposta planejada completa que um sistema deve fornecer para cumprir sua finalidade. Dessa forma, os *use cases* não garantem uma especificação completa da funcionalidade que o sistema deve possuir. Esse problema é agravado ainda mais em grandes sistemas, com muitos *use cases*, em que a completeza da especificação é muito mais difícil de ser avaliada e uma falha na especificação acarreta custos maiores de correção.

No caso 2, a modelagem feita de acordo com as heurísticas da modelagem *use case* apresenta respostas insatisfatórias, mesmo sob o ponto de vista do usuário. O estudo de caso 2 ilustra três tipos de problemas que podem ser inseridos pela modelagem *use case* na modelagem desse tipo de resposta:

1. **O sistema apresenta uma resposta incompleta.** O ponto de vista do usuário, usado para a modelagem, não garante que atividades internas, necessárias para continuidade do funcionamento do sistema, sejam reconhecidas e modeladas.
2. **Esvaziamento da funcionalidade do sistema.** A execução das atividades custodiais, que é responsabilidade do sistema, é atribuída erroneamente a agentes externos ao sistema.
3. **O sistema apresenta uma modelagem desnecessariamente complexa.** A identificação tardiamente de atividades custodiais, geralmente impõe soluções cuja complexidade poderia ser evitada.

Enquanto as diretrizes formais, que uma atividade essencial deve cumprir, garantem que é fornecida uma resposta completa e independente a um evento, a falta de diretrizes semelhantes por parte da modelagem *use case*, apesar de não excluir a obtenção de um resultado semelhante ao produzido pela AE, não conduz a modelagem nesse sentido, deixando grande parte da responsabilidade pela qualidade da análise a cargo da experiência do analista. No *use case* "dar baixa em item vendido", por exemplo, um analista *experiente* pode complementar a falta de heurísticas com sua experiência, e produzir uma resposta equivalente a da AE (Figura 3.8).

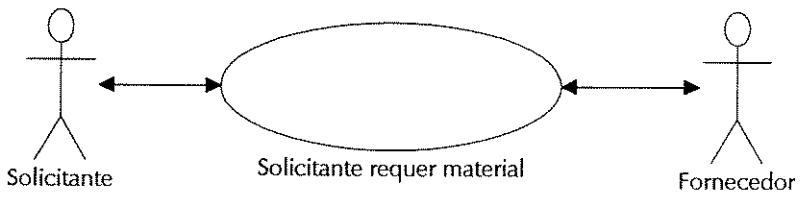


Figura 3.8 *use case* complementado com a experiência do analista.

As respostas produzidas pela modelagem *use case* não formam um conjunto modular de funcionalidades do sistema tão consistente e completo se comparado com o mesmo resultado produzido pela AE. Um modelo produzido pela AE, através das heurísticas que possui, é mais completo do que o produzido pelo conjunto de *use cases* equivalentes.

3.2.3 Caso 3: O Sistema avisa que Produtos estão com Validade Próxima do Fim

Alguns tipos de evento são iniciados, não por agentes externos ao sistema, mas pela chegada de determinado momento no tempo em que o sistema deve agir, de acordo com um planejamento anterior. Como exemplos podemos citar:

1. um sistema de controle de cobrança que deve avisar que certo pagamento está atrasado 60 dias;
2. um sistema de controle de estoque de produtos perecíveis que acusa o fim de validade de determinados produtos que devem ser retirados do estoque.

[9] No exemplo apresentado, a exigência de que o sistema deva ficar inativo após a execução de uma resposta planejada limita a abrangência do conceito de tecnologia perfeita, que consideraria válida uma amostragem contínua do estoque como sugerida anteriormente.

3. um sistema que deve emitir periodicamente o pagamento de funcionários.
4. um protetor de tela que é iniciado automaticamente se o computador ficar inativo por um período de tempo determinado.

Deve ficar claro que um evento temporal corretamente identificado dispara atividades essenciais independentemente da tecnologia usada na implementação do sistema [MEN84a]. Mesmo que um sistema seja capaz de verificar a cada segundo quais os produtos vencidos, por exemplo, a chegada da data de validade limite de um produto é que realmente dispara o evento.

Para o estudo de caso 3 será considerado um sistema de controle de estoque de um supermercado que deve informar automaticamente ao Sistema de Promoção de Vendas que determinados produtos estão com o prazo de validade próximo do fim. Com isso, o Sistema de Promoção de Vendas pode tomar medidas para evitar a perda dos produtos.

3.2.3.1 Resposta Produzida pela AE

Para caracterizar o evento como temporal, pode-se nomeá-lo de "Momento de informar proximidade de vencimento da validade de produtos perecíveis". A resposta a eventos temporais, comumente produzidas pela AE, segue um padrão característico, em que o fluxo da informação parte do sistema para as entidades externas. A resposta é iniciada aparentemente de modo espontâneo já que o estímulo do evento temporal não é explicitado no DFD e sim na especificação do processo[10] (Figura 3.9).

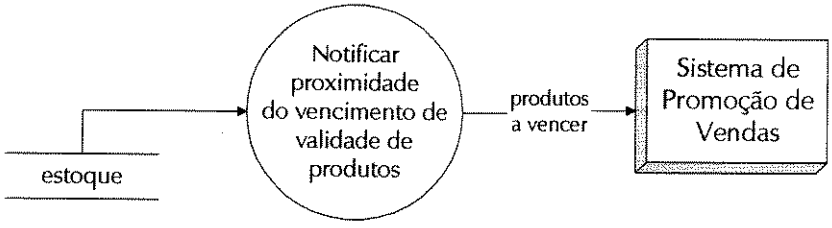


Figura 3.9 Resposta produzida, através da AE, para o caso 3.

Parte da especificação da função Notificar proximidade do vencimento de validade de produtos ficaria algo similar a:

```
Para cada lote de produto cujo ( fim-da-validade - dia-atual ) <= 15 dias
    faça notificar o Sistema de Promoção de Vendas
```

3.2.3.2 Resposta Produzida de acordo com *Use Cases*

Segundo as heurísticas da modelagem *use case*, cada *use case* é iniciado por um ator. Essa limitação já prejudica a identificação correta do evento e a modelagem de uma resposta adequada. Para contornar a limitação, pode-se denominar o evento como "Gerente solicita relação de produtos com vencimento da validade próximos do fim". O *use case* resultante é ilustrado pela Figura 3.10.

[10] O estímulo temporal pode ser considerado um fluxo de controle e não é representado no DFD produzido pela AE.

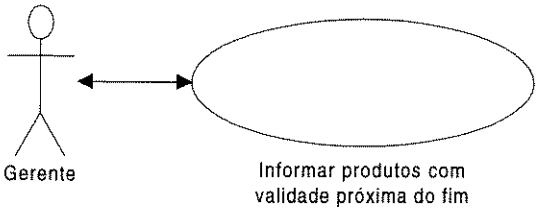


Figura 3.10 Gerente solicita relação de produtos com prazo de validade próximos do fim.

A descrição do *use case* ficaria:

1. O curso dos eventos inicia quando o Gerente seleciona a opção "Informar produtos com validade próxima do fim" no menu do sistema.
2. O Gerente informa, através de um campo apropriado, o número de dias de antecedência do fim da validade que deve ser processado.
3. Uma lista é montada contendo a relação, em ordem decrescente de número de dias restantes, de todos os lotes produtos cuja validade satisfazem o critério estipulado.
4. A informação é impressa na tela ou na impressora, dependendo da escolha feita pelo Gerente, com uma linha para cada produto.

3.2.3.3 Comparação entre Análise Essencial e Use Cases para o Caso 3

Além de algumas deficiências já citadas nas análises dos estudos de caso 1 e 2, o *use case* do caso 3 não corresponde exatamente ao desejado. Ao invés do sistema fornecer um acompanhamento automático do estoque, tomando a iniciativa de informar a proximidade do vencimento dos produtos, a responsabilidade da inspeção periódica fica a cargo de um agente externo, o Gerente, e parte da funcionalidade do sistema é subtraída. Além disso, uma falha na inspeção periódica, devido à falha humana, pode acarretar prejuízos desnecessários para o supermercado.

Já a modelagem produzida de acordo com a AE, especifica que o sistema é responsável por informar ao Sistema de Promoção de Vendas, mesmo que a solicitação não tenha partido deste (reação a um evento temporal). Dessa forma, fornece apoio à gerência para a tomada de decisões que maximizem os lucros. Não há definição de como essa funcionalidade será implementada pois se trata de uma decisão de projeto. Independentemente da implementação, essa funcionalidade deve ser fornecida pelo sistema.

Diferentemente do caso anterior, em que as heurísticas da modelagem *use case* não excluíam uma modelagem similar a da AE, a exigência de que o curso de eventos seja iniciado por um usuário exclui a modelagem de uma resposta (aparentemente) espontânea do sistema (reação a eventos temporais).

3.2.4 *Resumo dos Resultados dos Estudos de Caso*

A Tabela 3.2 apresenta os resultados obtidos através dos estudos de caso de um modo resumido.

Tabela 3.2 Resumo da comparação entre Análise Essencial e *Use Cases*

	Análise Essencial	Use Cases
Classificação	Método	Modelo
Ponto de vista que utiliza para especificar os requisitos	usuário e sistema	usuário
Nível de Detalhes	dividido entre os modelos	apenas descrição textual informal
Manutenibilidade	boa	ruim
Visibilidade da Funcionalidade	boa, através do DFD	ruim
Requisitos que define	apenas análise	análise e implementação misturados
Detalhamento da especificação	bom	variável
Possui heurísticas de modelagem	sim	não
Garante resposta completa ao evento	sim	não; permite modelagem que: 1. produz resposta incompleta; 2. retira funcionalidade do sistema; 3. insere complexidade.
Responde a evento temporal	sim	não. Usa modelagem alternativa insatisfatória.

3.3 Mapeamento entre Requisitos Funcionais e Objetos

As seções anteriores mostraram que é necessária a utilização de uma especificação que apresente os requisitos funcionais de um sistema. Justificada a utilização conjunta de métodos Estruturados e OO, resta ainda a questão da transição entre o paradigma Funcional e o paradigma da Orientação a Objetos. De acordo com Berard [BER95]:

> *Problemas freqüentemente aparecem, em esforços de desenvolvimento de software, quando estratégias de localização* diferentes *são usadas durante o* mesmo *esforço de desenvolvimento. Por exemplo, suponha que nos seja apresentado um conjunto de requisitos* funcionais *e que seja pedida a produção de uma solução* orientada a objeto. *Isto significa que deverá haver um deslocamento fundamental nas estratégias de localização em algum lugar entre os requisitos e a criação do produto final. Esta mudança freqüentemente resulta na introdução de erros significativos.*

Nem todos os autores consideram essa transição problemática. Para Shumate [SHU91], citado por George e Carter [GEO96], a transformação dos requisitos contidos nos modelos de análise orientados a função em modelos de projeto OO não é necessariamente ruim, pois análise e projeto têm finalidades bem diferentes. Sully [SUL89] acredita que as duas abordagens são complementares e a abordagem estruturada possui características que conduzem à construção de objetos correspondentes. Cita ainda que

"aspectos de cada uma devem ser preservados pois tratam de diferentes partes do modelo do sistema total" (p. 76).

A transição entre os modelos de Análise Estruturada (em especial a Análise Essencial) e OO está longe de ser uma tarefa dantesca. Da mesma forma que os métodos OO, a AE procura capturar as três dimensões que compõem um sistema: **Funcional** (Diagrama de Fluxo de Dados), **Estrutural** (Modelo Entidade Relacionamento) e **Dinâmica** (Diagrama de Transição de Estados). A diferença principal entre as abordagens é bem descrita por Rumbaugh *et al.* ao comparar SA ao OMT [RUM91b] (p. 267):

> *SA/SD e modelagem OMT têm muito em comum. Ambas as metodologias usam construções de modelagem similares e suportam as três visões ortogonais de um sistema. A diferença entre SA/SD e OMT é principalmente uma questão de estilo e ênfase.*

A questão principal da transição entre modelos produzidos pela AE para modelos OO não está na "transformação" dos requisitos de uma abordagem para outra, e sim, na **reorganização** da informação levantada. Enquanto os modelos Estruturados capturam a informação regidos pela funcionalidade que o sistema deve oferecer, os modelos OO buscam organizar essa informação em termos de objetos do sistema. A partir da reorganização do produto da AE, informação específica de objetos (*e.g.*, contratos, encapsulamento) pode ser incluída nas fases posteriores de desenvolvimento.

Dessa forma, as abordagens são complementares e adequadas em momentos distintos do desenvolvimento de um sistema.

3.4 Decomposição Funcional Guiando o Projeto dos Objetos do Sistema

Mudanças na especificação do sistema comumente fazem parte do processo de desenvolvimento. À medida que aumenta o entendimento do sistema por parte dos clientes e desenvolvedores envolvidos, possíveis alterações são realizadas na funcionalidade e estrutura do Sistema [PRE92].

A idéia de que um sistema baseado em objetos é menos abalado por alterações nos requisitos funcionais de um sistema fundamenta-se na suposição de que objetos são entidades independentes, com interfaces e classificação claramente definidas e estáveis. Desse modo, espera-se que uma mudança na especificação funcional possa ser facilmente acomodada através da inclusão ou exclusão de novos objetos, interfaces e relacionamentos [RUM91b] [BOO94a] [JAC92]. Contudo, nem objetos são independentes uns dos outros[11], nem há garantia de que mudanças nas interfaces dos objetos não existirão ou não acarretarão grandes efeitos por todo o sistema.

Segundo McGregor & Sykes [MCG92][MCG92], a dependência entre classes resulta de dois fatores: ou uma classe contém outra classe (através de herança ou composição), ou um método de uma classe invoca o método de outra classe (envio de mensagens). Portanto, a estabilidade do sistema baseado em objetos depende tanto da qualidade da taxonomia quanto da identificação dos serviços dos objetos produzidos pela OOA.

[11] Durante a discussão a respeito da construção de bibliotecas de componentes, Carrol ressalta a necessidade de se desacoplar classes [CAR93].

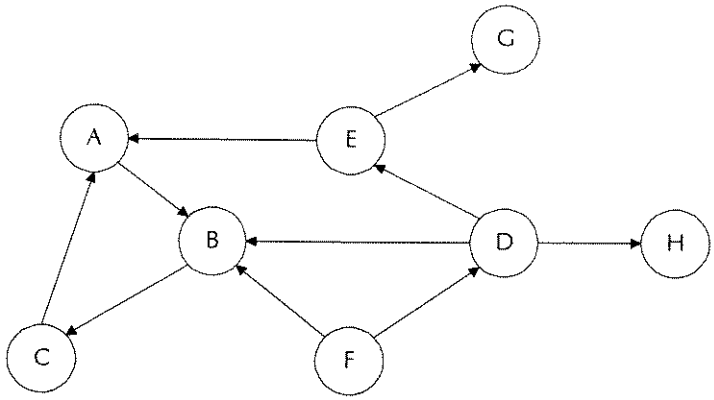


Figura 3.11 Exemplo de grafo de dependência entre classes produzidas por herança, composição ou envio de mensagens. Fonte [SOU94] (p.44).

A Figura 3.11 ilustra um exemplo de um grafo de dependência de classe. Cada nó representa uma classe de objeto e os arcos direcionados representam as interdependências de herança, composição ou de invocação de um método, de acordo com [MCG92].

Como mostrado por Soukup [SOU94], quando um grafo de dependência contém ciclos, grupos de classes precisam ser verificadas (debugged) e testadas em conjunto. No exemplo da Figura 3.11, apenas as classes *G* e *H* podem ser testadas individualmente, o que mostra o grau de interligação entre os objetos de um sistema.

Outra observação que pode ser extraída em relação à interdependência de classes de objetos, a partir da Figura 3.11, é o impacto que uma mudança na taxonomia de classes pode ter na estrutura do sistema. Em caso de exclusão da classe *B*, por exemplo, a alteração poderia ser propagada nas classes que dela dependem direta ou indiretamente, implicando uma redistribuição da responsabilidade da classe *B* entre as classes *A*, *C*, *D*, *E* e *F*.

O desejo de que a estrutura do sistema baseado em objetos sofra poucas modificações no decorrer do desenvolvimento é comprometido pela precariedade com que é feita a identificação dos objetos relevantes ao sistema, seus atributos e operações associadas[12]. Além de não haver uma métrica que possibilite qualificar uma modelagem baseada em objetos[13], os objetos do sistema são geralmente extraídos a partir de nomes presentes na especificação textual de requisitos (Figura 3.12). Os resultados obtidos são imprecisos devido às deficiências desse tipo de especificação, conforme mostrado nas seções anteriores. De forma análoga, a identificação das operações de um objeto pela identificação dos verbos também é precária devido à falta de exatidão e completeza desses requisitos.

[12] Diversos autores (*e.g.*, [BOO94b] [CON89] [SOM92b] [LOY90]) consideram a identificação de objetos um dos maiores problemas no desenvolvimento OO.

[13] De acordo com Booch [BOO94b] (p. 145), "não existem receitas simples para identificação de classes e objetos. Não existe uma estrutura de classe 'perfeita' nem um conjunto 'correto' de objetos".

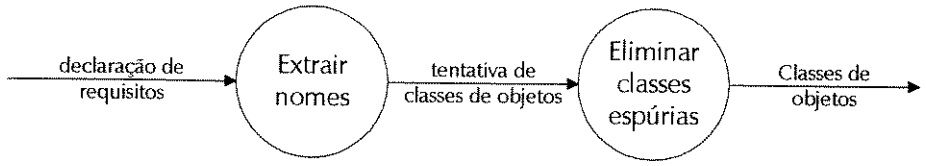


Figura 3.12 Processo mais comum de identificação de classes de objetos feitas por métodos OO. Figura retirada de [RUM91a] p. 153.

Dessa forma, é improvável que a estrutura de um sistema possa ser sólida se é construída sobre bases não muito estáveis. As alterações que serão realizadas sobre o sistema, feitas de acordo com a evolução do entendimento do problema, dar-se-á em fases mais adiantadas do desenvolvimento, em que o custo de uma alteração é muito maior.

Por outro lado, a utilização da AE na fase inicial de análise do sistema baseado em objetos poderia melhorar muito a situação, trazendo inúmeras vantagens para diversos pontos das fases posteriores de OOA/OOD, tais como:

1. **Identificação dos Objetos do Sistema.** A vantagem da utilização da AE é o aproveitamento do ERD produzido, que já contém o agrupamento da informação com que o sistema precisa lidar em entidades e relacionamentos, ajustados de acordo com o contexto funcional do sistema[14]. Cada entidade do ERD é um candidato natural a objeto do domínio do problema, facilitando, assim, o processo de identificação dos objetos. Posteriormente, restará apenas a identificação dos objetos do domínio da solução, responsabilidade essa da fase de OOD. Recomenda-se a utilização da notação de modelos de objetos já durante a AE, devido à semelhança entre a notação do ERD e o modelo de objetos do OMT[15], conforme ilustrado pelas Figura 3.13 e Figura 3.14.

[14] Toda a informação contida nos fluxos de dados e descritas mais detalhadamente pelo dicionário de dados são consideradas na formação das entidades do ERD.

[15] Rumbaugh *et al.* [RUM91b] consideram a modelagem de objetos utilizada no OMT uma evolução do ERD tradicional, e apresentam duas figuras, 12.1 e 12.2 (p. 272), para ilustrar a semelhança entre as notações.

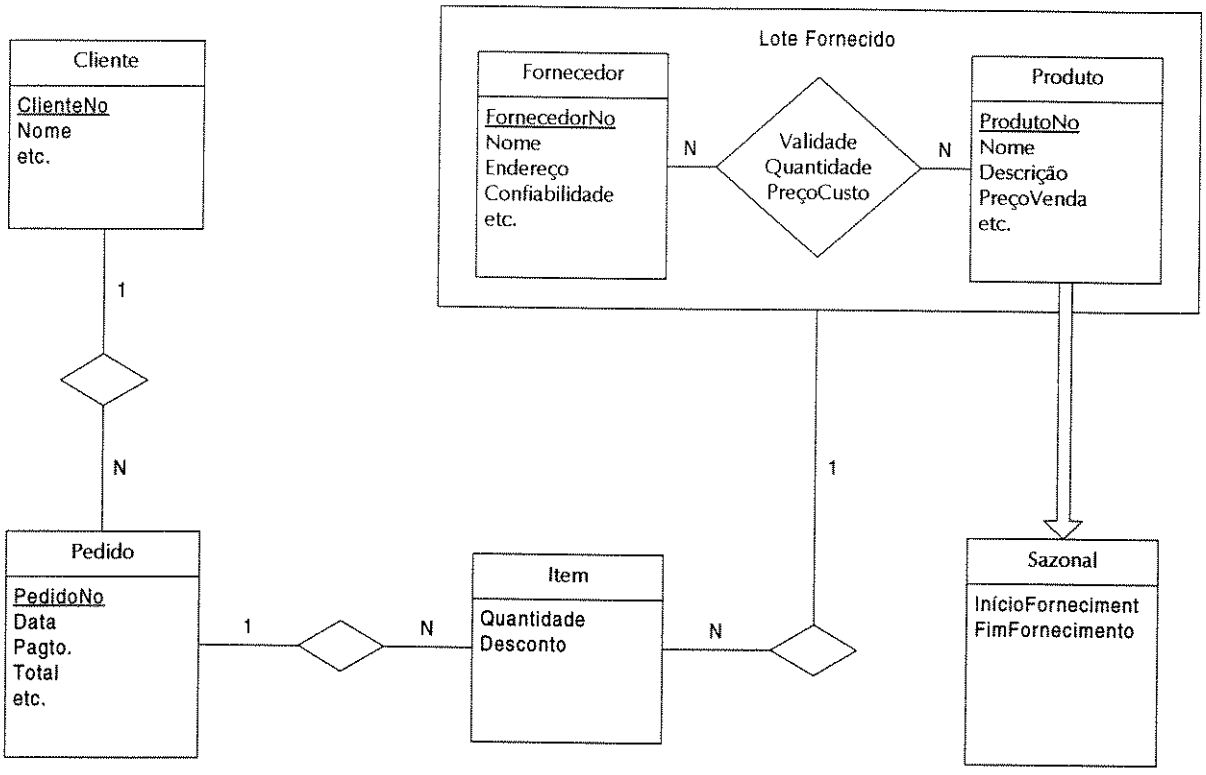


Figura 3.13 Exemplo para ilustrar a notação de ERD para um sistema de controle de vendas e estoque, produzido pela AE.

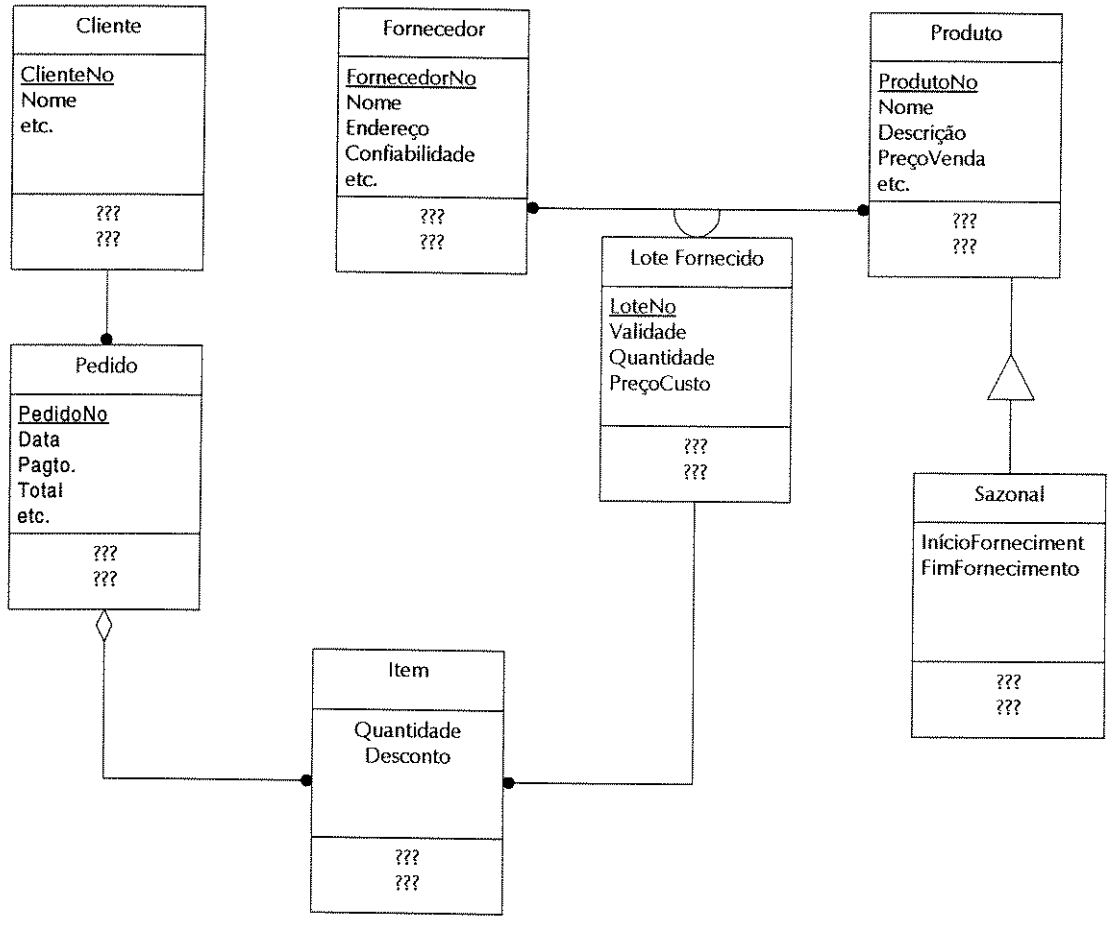


Figura 3.14 Modelo de Objetos equivalente da Figura 3.13, segundo o modelo de objetos do OMT, produzido a partir do ERD.

1. **Identificação das Responsabilidades de cada Objeto.** Cada mecanismo evento-resposta representa um cenário de uso do sistema, e especifica todas as responsabilidades que devem ser cumpridas nesse cenário. O processo de identificação e distribuição das responsabilidades e colaborações entre os objetos pode ser feito de acordo com cada resposta planejada (Figura 3.15). Cada mecanismo evento-resposta fornece um roteiro que a interação dos objetos deve seguir para realizar cada funcionalidade do sistema.

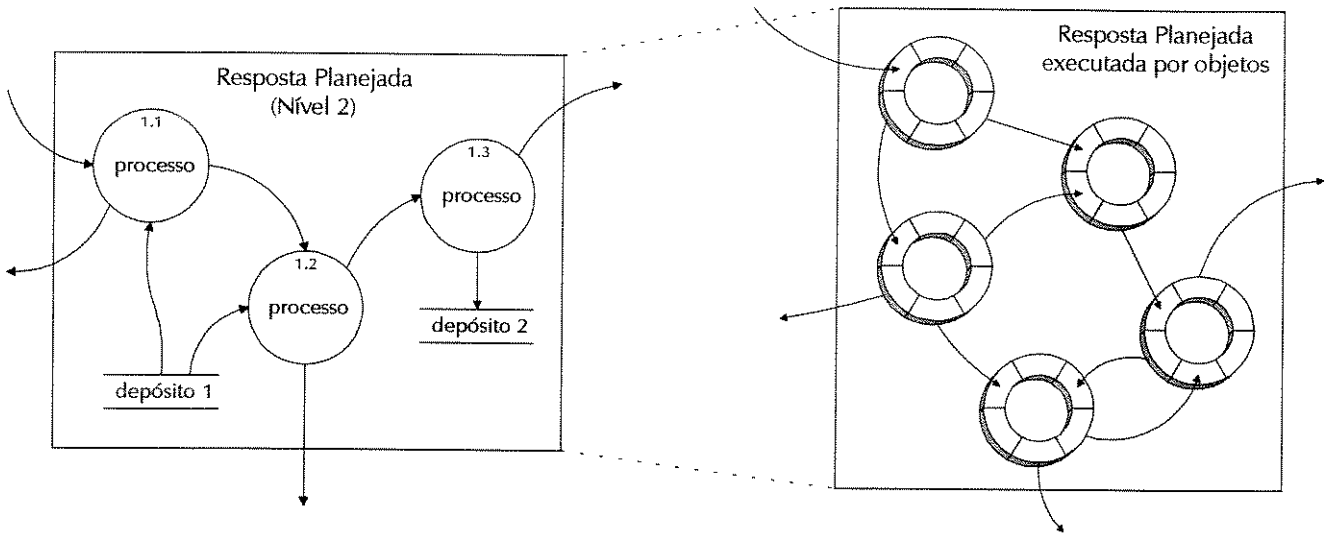


Figura 3.15 Resposta planejada executada por objetos do sistema.

1. **Apoio ao Processo de Desenvolvimento do Sistema.** As respostas planejadas, produzidas pela AE, podem ser usadas como guias para as fases posteriores de desenvolvimento na verificação e validação do sistema. Independentemente dos objetos que comporão o sistema, a funcionalidade de cada resposta deve ser satisfeita para que a implementação do sistema seja considerada adequada. Além disso, como cada resposta é independente, o sistema pode ser desenvolvido incrementalmente de acordo com módulos de funcionalidades fornecidos por cada resposta planejada. Novas técnicas de gerenciamento de projeto podem ser elaborados para a divisão da responsabilidade de implementação de cada resposta entre grupos distintos.

2. **Métricas de Qualidade.** Enquanto a modelagem produzida por métodos OO ainda sofre da deficiência de métricas adequadas de avaliação de qualidade e completeza, graças às heurísticas da AE, analistas de sistemas conseguem produzir modelagens muito similares e aferir precisamente a qualidade e completeza de outras modelagens, independentemente de seu nível de experiência. Em uma abordagem híbrida, as heurísticas da AE poderiam ser estendidas aos métodos OO, definindo limites claros a serem respeitados pelos analistas durante o desenvolvimento, que dessa forma se beneficiariam da padronização que a AE fornece.

3.5 Considerações Finais

Esse capítulo procurou mostrar, através da caracterização dos requisitos típicos de um sistema, que a utilização de uma especificação funcional durante a fase de análise do sistema, mais do que desejável, é absolutamente necessária para certos tipos de sistemas. Contudo, os métodos OO não possuem modelos apropriados para capturar a funcionalidade global do sistema e precisam utilizar ferramentas auxiliares

para tal tarefa, principalmente devido ao conceito de funções, não subordinadas a um objeto, não estar de acordo com o princípio do paradigma OO.

Dentre as ferramentas usadas pelos métodos OOA citadas, a modelagem *use case* e a Análise Essencial são representantes significativos dos únicos tipos de abordagens que realmente conseguem capturar a funcionalidade a partir do zero (especificações textuais e análise estruturada), e oferecem um particionamento semelhante.

Os estudos de caso apresentados mostraram que a Análise Essencial fornece uma especificação muito mais completa e consistente do que a fornecida pela modelagem *use case*, devido às heurísticas e aos modelos que utiliza, mostrando-se muito mais adequada para um posterior desenvolvimento OO.

Visto que é necessário uma abordagem funcional do sistema, uma estratégia de transição entre a AE e métodos OO precisa ser desenvolvida de forma a reorganizar a informação contida nos modelos da AE em termos de objetos, incluindo-se, posteriormente, informação específica relacionada com o paradigma OO.

A estabilidade do sistema baseado em objetos pode ser melhorada com o apoio da AE, cujo resultado fornece uma base sólida sobre a qual fases posteriores de desenvolvimento podem ser executadas.

3.6 Bibliografia

[ABB80] ABBOTT, R. J.; Report on Teaching Ada, *Technical Report SAI-81-313-WA*, Revised December 1980, Research Report for DARPA Order No. 3456, Contract No. MDA 903-80-C-0188, Science Applications, Inc. McClean, Virginia, 1980. *Apud* [LOY90].

[BEC89] BECK, Kent & CUNNINGHAM, Ward. A Laboratory For Teaching Object-Oriented Thinking. *SIGPLAN Notices*. Vol. 24, No. 10, October 1989.

[BER95] BERARD, Edward V.; Be Careful With 'Use Cases'. http:\\www.toa.com, The Object Agency, 1995.

[BOO94a] BOOCH, G. Complexity. In: *Object-Oriented Analysis and Design with Applications*. 2ª Ed. The Benjamin/Cummings Publishing Company Inc. 1994. cap. 1. pp. 3-26.

[BOO94b] BOOCH, G.; Classification. In: *Object-Oriented Analysis and Design with Applications*, 2ª Ed., The Benjamin/Cummings Publishing Company Inc., 1994, cap. 4, pp. 145-168.

[CAR93] CARROL, Martin. Design of The USL Standard Components. *C++ Report*. Vol. 5, No. 5, June 1993, pp. 34-39, 53.

[CHEN94] CHEN, J. & CHEN, Y.; A New Object-Oriented Method Integrating Jackson Structured Programming Method, *The Journal of Systems and Software*, No. 24, February 1994, pp. 125-137. *Apud* [GEO96].

[COL94] COLEMAN, D. et al., Object-Oriented Development: The Fusion Method, Prentice-Hall, Englewood Cliffs, New Jersey, 1994.

[CON89] CONSTANTINE, Larry L.; Object-Oriented and Structured Methods - Toward Integration, *American Programmer*, Vol. 2, No. 7-8 Summer 1989, pp. 34-40.

[FIC92] FICHMAN, Robert G. & KEMERER, Chris F. "Object-Oriented and Conventional Analysis and Design Methodologies - Comparison and Critique", *Computer*, October 1992, pp. 22-39.

[GAN79] GANE, Chris & SARSON, Trish. Análise Estruturada de Sistemas. *Livros Técnicos e Científicos Editora S.A.*, 1983, pp. 257. (tradução de Structured systems analysis: tools and techniques, 1979).

[GEO96] GEORGE, Joseph & CARTER, Bradley, D. A strategy for Mapping from Function-Oriented Software Models to Object-Oriented Software Models. *Software Engineering Notes*, vol. 21, No. 2, Março 1996, pp. 56-63.

[JAC92] JACKOBSON, I.; Christerson, M.; Jonsson, P e Övergaard, G.; *Object-Oriented Software Engineering: a Use Case Driven Approach*, Reading, Massachusetts, Addison-Wesley, 1992.

[KHA89] KHALSA, G. K.. Using Object Modeling to Transform Structured Analysis into Object-Oriented Design. *Proc. of the 6th Washington Ada Symposium*, Washington, June 26-29, 1989, pp. 201-212. *Apud* [GEO96].

[LOY90] LOY, Patrick H. A comparison of Object-Oriented and Structured Development Methods. *ACM Software Engineering. Notes.* Vol. 15, No. 1, January 1990, pp. 104-124.

[MCG92] McGREGOR, J. D. and SYKES, D. A. *Object Oriented Software Development Engineering Software for Reuse,* Van Nostrand Reinhold, New York, 1992, *Apud* [SOU94].

[MEN84a] McMENAMIN, Sthephen M. & PALMER, John F. Princípios de Particionamento para Modelos Essenciais. In: *Análise Essencial de Sistemas.* São Paulo. McGraw-Hill. 1991. Cap. 7. pp. 75-102. (tradução de "Essential Systems Analysis", 1984)

[MEN84b] McMENAMIN, Sthephen M. & PALMER, John F. Estratégias para Modelar a Essência de um Sistema. In: *Análise Essencial de Sistemas.* São Paulo. McGraw-Hill. 1991. Cap. 10. pp. 155-179. (tradução de "Essential Systems Analysis", 1984)

[PAG89] PAGE-JONES, Meilir & WEISS, Steven. Synthesis: An Object-Oriented Analysis and Design Method. *American Programmer*, Vol. 2, No. 7-8 Summer 1989, pp. 64-67.

[PRE92] PRESSMAN, Roger S.; Requirements Analysis Fundamentals. In: *Software Engineering - A Practitioner's Approach*, 3ª Ed., McGraw-Hill International Editions, 1992, Cap. 6, pp. 173-205.

[RUM91a] RUMBAUGH, James *et al.* Analysis. In: *Object-Oriented Modeling and Design.* Prentice Hall International, 1991. Cap. 8, pp. 148-197.

[RUM91b] RUMBAUGH, James *et al.* Comparison of Methodologies. In: *Object-Oriented Modeling and Design.* Prentice Hall International, 1991. Cap. 12, pp. 266-277.

[SHA90] SHLAER, Sally & MELLOR, Stephen. O Papel do Modelo de Informação no Desenvolvimento de Sistemas. In: *Análise de Sistemas Orientada para Objetos.* São Paulo. McGraw-Hill. 1990. Cap. 9. pp. 110-134. (tradução de Object-Oriented Systems Analysis, 1988)

[SHU91] SHUMATE, K., Structured Analysis and Object-Oriented Design are Compatible. *ACM Ada Letters* 11 (May/June), 78-90, 1991. *Apud* [GEO96].

[SOM92a] SOMMERVILLE, IanSoftware Requirements Definition. In: *Software Engineering*, 4ª Ed, Addison-Wesley Publishing Company, 1992, cap.3, pp. 47-63.

[SOM92b] SOMMERVILLE, Ian. Object-Oriented Design. In: *Software Engineering*, 4ª Ed, Addison-Wesley Publishing Company, 1992, cap.11, pp. 191-218.

[SOU94] SOUKUP, Jiri. *Tamming C++: pattern classes and persistence for large projects.* Addison-Wesley Publishing Company, Inc. 1994, cap. 2.

[SUL89] SULLY, Phil; Structured Analysis: Scaffolding for Object-Oriented Development. *American Programmer*, Vol. 2, No. 7-8 Summer 1989, pp. 76-82.

[VAZ93] VAZQUEZ, Federico. Using Object Structured Development to Implement a Hybrid System. *Software Engineering Notes*, vol. 18, no 4 October 1993, pp. 44-53.

[WAR89] WARD, Paul T., How to Integrate Object Orientation with Structured Analysis and Design, *IEEE Software* 6 (March), 1989, pp. 74-82.

[YOU89] YOURDON, Edward. Object-Oriented Observations. *American Programmer*, vol. 2, No. 7-8, august, 1989, pp. 3-9.

[ZAV97] ZAVE, Pamela & JACKSON, Michael. Four Dark Corners of Requirements Engineering. *ACM Transactions on Software Engineering and Methodology*, Vol. 6, No. 1, January 1997, pp. 1-30.

4. PROPOSTA DE DESENVOLVIMENTO

A utilização conjunta de métodos estruturados, até mesmo usando AE, e métodos OO não é recente. Contudo, as propostas anteriores não conseguiram explorar completamente a integração desejada entre métodos de paradigmas diferentes. Como relatado no capítulo anterior, a busca inicial era para se manter a base de conhecimento acumulado e investimentos realizados nos métodos convencionais. Como será mostrado em uma das seções desse capítulo, na conjunção dos métodos Estruturados e OO, os primeiros geralmente representavam um papel secundário, tendo sido parcialmente utilizados, com modelos alterados e heurísticas abandonadas. Os objetos do sistema, na maioria dos métodos híbridos, apenas fornecem um encapsulamento de funções e depósitos de dados, mantendo um particionamento claramente funcional.

A necessidade de uma abordagem mais balanceada, com igual importância entre a visão Funcional, obtida na Análise, e a visão OO, obtida nas fases posteriores de desenvolvimento, será apresentada juntamente com uma sugestão, não formalizada, de desenvolvimento conjunto de métodos Estruturados e OO que busca manter esse equilíbrio, produzindo, assim, um resultado realmente sinergético de desenvolvimento.

4.1 PROPOSTAS ANTERIORES DE DESENVOLVIMENTO CONJUNTO

Junto com a popularização do paradigma de Orientação a Objetos no fim dos anos 80 surgiu a idéia de uma utilização conjunta com os métodos Estruturados, já estabelecidos e amplamente conhecidos. Entre as propostas coletadas, a tônica comum é o papel secundário exercido pela Análise Estruturada no processo de desenvolvimento. Quatro tipos de abordagens são comuns:

1. uso parcial de Análise Estruturada para auxílio na especificação do sistema;
2. alteração de modelos de métodos convencionais para adequação ao paradigma OO;
3. utilização de particionamento de eventos para especificação do sistema[1];

[1] Considera-se a utilização do particionamento por eventos como uma forma de se especificar a funcionalidade do sistema derivada da primeira formalização feita nesse sentido, trazida pela AE.

4. utilização das abordagens anteriores combinadas em diferentes graus.

O método OOSE, proposto por Jackobson *et al.* [JAC92] propõe uma variação do particionamer por eventos da AE, a análise *use case*, cujas falhas já foram discutidas no capítulo anterior. Durante construção do Modelo de Análise, sugere a definição de objetos responsáveis por lidar com a interface funcionalidade de cada evento e a informação do sistema, produzindo uma concentração da funcionalidade do sistema nos objetos de controle[2]. O encapsulamento de cada funcionalidade do sistema em ca objeto de controle proporciona uma modelagem OO apenas cosmética, com forte semelhança à mode gem produzida pela Análise Estruturada (Figura 4.1).

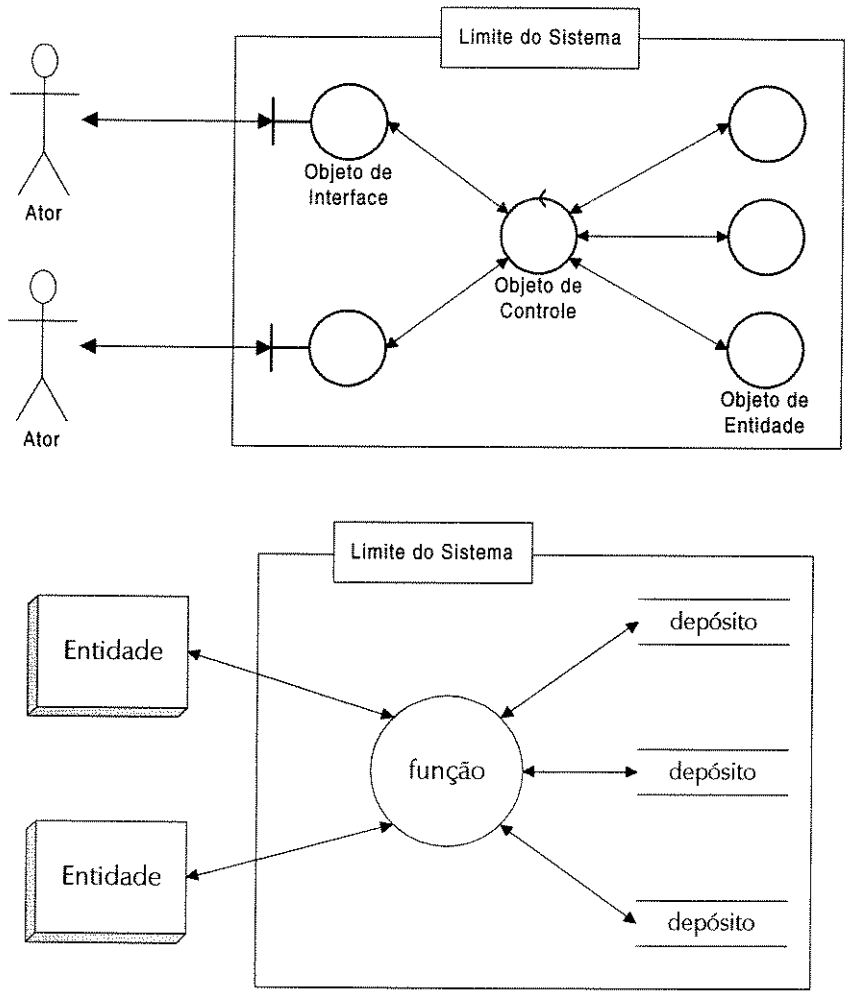


Figura 4.1- Modelagem do sistema de acordo com objetos de controle, interface e entidade (Figura 4.1a), modelagem equivalente produzida pela Análise Estruturada (Figura 4.1b).

Page-Jones e Weiss [PAG89], em seu método, Synthesis, preferiram modificar alguns modelos pr sentes na Análise Estruturada e Análise Essencial, aproximando-os do paradigma OO, tais como o Mode de Informação Aumentado, equivalente ao ERD, e o Dicionário de

[2] Conforme o trabalho de Sharble e Cohen [SHA93], no qual é feita uma comparação qualitativa entre as modelagens produzidas p métodos orientados a Responsabilidade e a Dados, "a presença de classes controladoras e controladas é, essencialmente, uma ma festação do encapsulamento pobre dessas classes." (p. 70).

Eventos, antes especificado através de respostas planejadas da AE e agora através de especificações textuais. A utilização de objetos para representar o sistema, gerenciadores e reconhecedores dos eventos, impõe uma certa tendência funcional na modelagem dos objetos, na medida em que produz uma distribuição desigual de responsabilidades de execução da funcionalidade do sistema, sobrecarregando os objetos gerenciadores.

O método OOS, de Bailin [BAI89], transporta prematuramente o enfoque da análise, da funcionalidade do sistema para as entidades que executam essa funcionalidade. Apesar de usar inicialmente um DFD para identificar as entidades do domínio do problema, a ênfase nos serviços que o sistema oferece não é mantida. Em seu lugar é dado um enfoque na comunicação entre entidades ativas do sistema[3], expressa através do EDFD[4].

Como exemplo de propostas que usaram parcialmente os conceitos da Análise Estruturada, pode-se citar os trabalhos de George e Carter [GEO96], e Vazquez [VAZ93]. A característica principal é o mapeamento direto dos modelos produzidos pela Análise Estruturada em objetos do sistema. Os serviços dos objetos são identificados a partir de funções do DFD e os objetos e seus relacionamentos a partir do ERD. Apenas o DFD de nível mais baixo é usado, com todas as funções expostas em um único modelo. A abstração da complexidade do sistema, conseguida através dos vários níveis do DFD, é perdida, sobrecarregando os desenvolvedores com o grande número de processos presentes em um único modelo, número este que facilmente extrapola os limites humanos de tratamento de complexidade (7±2 *blocos* de informação por vez). Além disso, dificilmente existe uma correspondência de 1 para 1 entre métodos de objetos e processos no DFD [SUL89]. A tentativa de mapeamento direto de processos em métodos pode conduzir o processo de análise a uma modelagem já direcionada para esse fim.

Por último, têm-se os métodos que incorporaram o particionamento por eventos. O método OBA, proposto por Rubin e Goldberg [RUB92], e o trabalho de Poo e Lee [POO95], por exemplo, mostram avanço na busca de uma melhor identificação dos objetos do domínio do problema do que a oferecida pela análise de nomes, verbos e adjetivos presentes na especificação dos requisitos. Os eventos têm um papel importante, tanto na especificação da funcionalidade do sistema, quanto na especificação do comportamento dos objetos. OBA acena uma proximidade com a utilização de neutralidade tecnológica na análise, e mostra uma preocupação com rastreabilidade e a elaboração de um processo previsível e controlável, facilitando o gerenciamento do projeto. Em [POO95], é fornecida uma especificação bem rica de cada evento do sistema. A ressalva para ambos os trabalhos é que, desde a especificação dos eventos, as responsabilidades já são atribuídas ao possível objeto que irá cumpri-la. A definição da funcionalidade do sistema e a atribuição da responsabilidade de executar tais funcionalidades deveriam representar dois momentos diferentes no desenvolvimento do sistema: no primeiro, procura-se responder "o que" deve ser feito; no segundo momento, tenta-se responder "por quem". Objetos de suporte usados em [POO95] são análogos aos objetos gerenciadores de eventos, objetos de controle e entidades ativas, já citados anteriormente.

[3] De forma similar ao OOSE e ao Synthesis, o método OOS modela distintamente entidades ativas e passivas. No EDFD nível 0, entidades ativas são representadas por processos e entidades passivas por fluxos e depósitos de dados.

[4] EDFD nada mais é do que uma forma modificada de DFD que enfatiza entidades e funções ao invés de processos.

Na tabela abaixo são resumidas as principais características de cada proposta analisada nesta seção.

Tabela 4.1- Resumo da análise das propostas discutidas na seção.

	OOSE	Synthesis	OOS	[GEO96] [VAZ93]	OBA	[POO95]
usa particionamento por eventos	X	X			X	X
modifica algum modelo da Análise Estruturada		X	X	X		
produz objetos ativos (controladores) e passivos	X	X	X			X
atribuição prematura de responsabilidades aos objetos		X	X	X	X	X

O uso de objetos controladores, como os objetos de controle do OOSE e os objetos gerenciadores do evento do Synthesis, não conduz a uma modelagem balanceada da divisão das responsabilidades entre os objetos do sistema. Wirfs-Brock *et al.* [WIR90] afirmam que (p. 64):

> *Esta abordagem tem o efeito de centralizar o controle com o objeto inteligente (objeto controlador). De fato, tem algo de um sabor procedural: o objeto inteligente terá muito mais a mesma finalidade de um módulo controlador central de um programa procedural, enquanto outros objetos se comportarão como estruturas de dados tradicionais.*

Apesar dessa abordagem fornecer vantagens similares às da AE, não é muito próxima do espírito do paradigma OO[5].

4.2 Características Desejáveis de um Método Híbrido

Baseado no que foi analisado até o momento, definem-se algumas características desejáveis que uma proposta de utilização conjunta de métodos Estruturados e OO deve possuir:

1. **independência entre as fases de Análise e Projeto.** A análise e o projeto do sistema servem a propósitos distintos. A modelagem feita na fase de análise deve especificar o que o sistema deve fazer. As fases posteriores, baseados na análise, devem decidir como e por quem.
2. **transição suave entre as fases de Análise e Projeto.** Apesar de as fases serem distintas, com novas informações inseridas em cada uma, a transição entre elas deve se dar de forma suave, mantendo-se a rastreabilidade entre os modelos.
3. **importância equilibrada entre decomposição Funcional e OO.** A priorização de um ou outro paradigma pode levar a um não aproveitamento completo do que cada um tem a oferecer. Para um relacionamento realmente sinergético é necessário que o melhor de cada um possa ser explorado adequadamente. A interferência negativa entre os paradigmas também deve ser evitada

[5] No capítulo seguinte, é discutida, a título de extensão, a proposta de um estudo comparativo mais profundo das implicações de modelagens de sistemas baseados em objetos mais próximos da AE, e sistemas baseados em objetos com responsabilidade mais bem distribuídas, em relação à reusabilidade e manutenibilidade.

de modo a realmente se obter funções que se comportem como funções e objetos que se comportem como objetos.

A dificuldade em se cumprir esses requisitos é que eles se mostram um pouco contraditórios. A exigência de uma importância equilibrada entre decomposição Funcional e OO impõe uma certa limitação na transição suave entre um paradigma e outro. Como manter um equilíbrio entre os diferentes paradigmas e, mesmo assim, conseguir uma transição suave entre as fases de desenvolvimento?

A necessidade de independência entre as fases de Análise e Projeto vai de encontro ao desejo de uma transição suave entre essas fases. Entre os métodos OO, principalmente entre os dirigidos a estrutura de dados (*e.g.* OMT, OOAD - Booch), observa-se que não existe uma divisão clara entre onde termina a Análise e onde começa o Projeto[6]. A inexistência dessa linha divisória entre as diferentes fases é citada por muitos metodologistas OO como uma vantagem já que "fornece uma transição mais suave entre os modelos de análise e projeto do que a fornecida pelas metodologias estruturadas" [FIC92] (p. 36). Mesmo as propostas anteriores de utilização conjunta de paradigmas Estruturados e OO não conseguiram definir claramente esses limites, devido à falta de heurísticas definindo esses limites e à inexistência de uma transição adequada entre as fases. Dessa forma, permanece a questão: como conciliar a distinção entre fases diferentes de desenvolvimento e a transição suave entre essas mesmas fases?

Contudo, nem todos os quesitos geram contradições. A distinção entre os objetivos de cada fase de desenvolvimento, juntamente com o quesito de uma importância equilibrada entre decomposição Funcional e OO, sugere que haja uma divisão de responsabilidades entre os métodos Estruturados e OO em relação a respostas das perguntas: o que, como e por quem que cada fase procura responder. Através dessa linha de raciocínio, restaria apenas a questão da transição suave entre paradigmas diferentes.

4.3 Proposta Não-Formalizada de Um Método Híbrido

Apesar de não formalizada, a proposta que será apresentada a seguir, referente às fases de Análise e Projeto de Sistemas, sugere um método viável para se alcançar as características levantadas na seção anterior, através da combinação da Análise Essencial [MEN84], Cartões CRC [BEC89] e RDD [WIR90][7] (Figura 4.2).

[6] De acordo com Fichman e Kemerer [FIC92], "a maior parte das metodologias (OO) atuais recomendam que a análise e o projeto sejam realizados interativamente, até mesmo concorrentemente" (p. 36).

[7] Apesar de o método RDD incorporar a técnica dos cartões CRC em seu processo de identificação de classes, a separação entre um trabalho e outro foi feita para ressaltar a importância dos cartões na proposta apresentada.

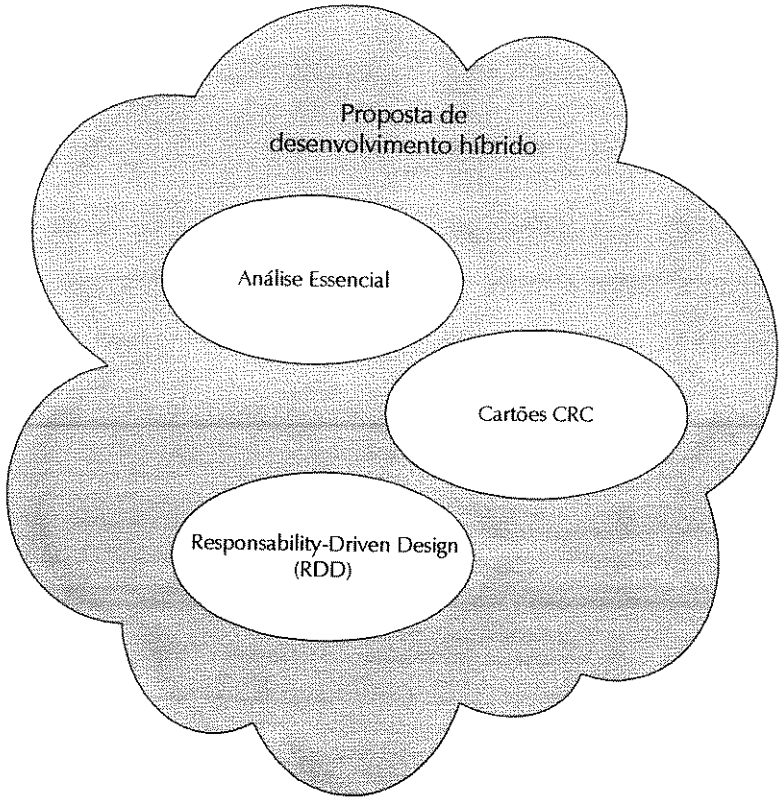


Figura 4.2 Métodos usados para a elaboração da proposta apresentada.

A fase de Análise se preocupa com a definição de o que o sistema deve fazer. Conforme visto no capítulo anterior, os requisitos tratados nessa fase são os requisitos funcionais, que constituem a visão do sistema sob o ponto de vista dos usuários[8]. A Análise Essencial foi associada a essa fase pois produz uma modelagem que, além de corresponder ao ponto de vista do usuário, fornece uma ótima especificação, em termos de completeza e exatidão, para as fases posteriores de desenvolvimento.

A fase de Projeto, que responde por quem e como cada funcionalidade do sistema é executada, é dividida em duas etapas. Na primeira delas, busca-se executar completamente cada resposta planejada, levantada pela AE, apenas através de interações entre os **objetos essenciais** do sistema, obtidos inicialmente a partir de entidades do MER e entidades externas de cada resposta planejada.

A definição de objeto essencial, discutida com mais detalhes nas próximas seções, procura estender o conceito de tecnologia perfeita de [MEN84] ao contexto de objetos, facilitando o processo de reorganização da informação. Desse modo, desconsideram-se também restrições de implementações relacionadas a objetos, tais como persistência, distribuição em rede, intercomunicação de objetos, limitações da linguagem de implementação etc.

Na etapa seguinte, a essência do sistema é materializada em uma implementação. Para isso, é usado um **objeto adaptado**, que tem como objetivo adequar a essência de um objeto essencial a um determinado conjunto de restrições de implementação. Para facilitar a resolução dos problemas ligados à implementação, os problemas serão isolados e sua resolução delegada a **objetos de apoio à implementação**, cuja única função é auxiliar os objetos adaptados na adequação da essência de um objeto. Esta última eta-

[8] A visão do sistema através dos objetos pertence aos projetistas do sistema e não aos usuários.

pa não se baseia em nenhum método específico. Consiste de uma proposta de implementação sugerida por este trabalho.

Na primeira etapa da fase de projeto usa-se a técnica dos cartões CRC [BEC89] e o método RDD (Responsibility Driven Design) [WIR90] pois distribuem a inteligência do sistema[9] mais igualmente entre os objetos do sistema. Essa escolha é reforçada pela conclusão do trabalho de Sharble e Cohen [SHA93], no qual é feita uma análise qualitativa de métodos dirigidos a dados e métodos dirigidos a responsabilidades (p. 73):

> *Em resumo, o método dirigido a responsabilidade (em relação ao método orientado a dados analisado) produziu um projeto muito menos complexo e exibiu um apoio muito mais forte ao encapsulamento e classificação, maior habilidade de aumentar a coesão e reduzir o acoplamento, e um melhor uso de herança para reduzir redundância. A abordagem dirigida a responsabilidade parece ser significativamente mais efetiva para a produção de software que possa ser mantido, estendido e re-usado.*

Embora a proposta híbrida apresentada até o momento não esteja formalizada, percebe-se que os quesitos levantados na seção anterior estão bem próximos de serem cobertos:

- a especificação de o que o sistema deve fazer é definida pela AE, sem que nenhuma alteração de suas heurísticas ou modelos, mantendo, assim, a separação da fase de Projeto, que é tratada através dos cartões CRC e RDD;
- a transição entre a Análise e o Projeto é feita naturalmente devido a afinidade natural entre a AE e os cartões CRC. A AE, através das respostas planejadas que modela, fornece os cenários adequados para a utilização dos cartões CRC.
- na medida em que desempenham papéis distintos mas complementares no desenvolvimento, o paradigma Funcional e o OO possuem uma importância equilibrada.

4.4 Resumo da Proposta

O processo de Análise e Projeto é dividido em 3 fases, cada qual respondendo a uma determinada dimensão de modelagem:

1. **Análise do sistema sob o ponto de vista funcional.** Essa fase trata de responder a pergunta de o que deve ser feito pelo sistema. Para isso, a Análise Essencial [MEN84] é usada sem nenhuma alteração de suas heurísticas ou modelos. Assim, reforça-se a independência entre fases distintas de desenvolvimento.
    1.1. identificar o objetivo do sistema, através do levantamento das deficiências do ambiente que ele deve suprir;
    1.2. definir a que eventos o sistema deve responder para cumprir a funcionalidade desejada do sistema e as respectivas atividades fundamentais (que justificam a necessidade do sistema);
    1.3. identificar a informação que o sistema precisa através da análise do conjunto de elementos de dados produzidos pelas respostas fundamentais;

[9] "Uma classe incorpora mais ou menos inteligência de acordo com quanto ela sabe ou pode fazer, e quantos outros objetos pode afetar" **[WIR90]** (p. 64).

1.4. definir as atividades responsáveis pelo ciclo de vida da informação do sistema, atividades custodiais, que ainda não foram cobertas pelas atividades anteriores.

2. **Análise das respostas planejadas sob o ponto de vista dos objetos essenciais do sistema.** Nessa fase, cada resposta planejada, especificada na fase anterior, é executada através da distribuição de responsabilidades e definição de colaborações entre os objetos essenciais do sistema. A idéia é que a funcionalidade essencial do sistema deva ser reorganizada em objetos essenciais. Responde-se, assim, por quem serão executadas as funcionalidades do sistema. Os passos 2.2 e 2.3 devem ser feitos iterativamente até a obtenção de um resultado adequado.

   2.1. identificar os objetos essenciais iniciais. Definir um cartão CRC para cada entidade presente no ERD e para cada entidade externa que interage com o sistema.

   2.2. para cada resposta planejada, efetuar uma abordagem antropomórfica na qual cada objeto, representado por um cartão, procura desenvolver responsabilidades e interagir com outros objetos essenciais participantes (colaboradores) de modo a cumprir as responsabilidades de cada atividade essencial. As responsabilidades que devem ser cumpridas em uma resposta planejada são expressas através de especificações, em português estruturado e/ou em pré e pós condições, de cada atividade pertencente a esta resposta. As responsabilidades devem ser associadas a cada classe de modo que a inteligência do sistema seja igualmente distribuída, com comportamentos mantidos com informação relacionada e as responsabilidades divididas entre classes relacionadas. Ao final da análise de cada atividade essencial, construir um Diagrama de Colaboração entre Objetos para a resolução da atividade.

   2.3. definição de hierarquias e contratos. O modelo entidade-relacionamento (MER) definido na análise deve ser refinado, aprimorando a hierarquia de classes de acordo com a fatoração de responsabilidades comuns entre as classes[10]. Contratos são definidos agrupando-se as responsabilidades usadas pelos mesmos clientes.

3. **Inclusão de objetos de apoio à implementação.** Objetos essenciais são replicados em objetos adaptados para que possam sofrer adaptações da essência à realidade de implementação. Os objetos adaptados contam com o auxílio dos objetos de apoio à implementação para a adequação da essência. Recomenda-se que a adequação seja feita principalmente através de delegação, ao invés de priorizar alterações na hierarquia de classes, de modo a preservar, tanto quanto possível, a estrutura original da hierarquia de objetos essenciais e facilitar a rastreabilidade dos modelos.

   3.1. especificar detalhes de implementação de cada elemento e fluxo de dado contido na especificação de análise.

   3.2. definir dos protocolos dos métodos correspondentes às responsabilidades de cada objeto adaptado.

   3.3. definir os problemas relacionados à implementação de cada método e as possíveis soluções.

   3.4. refazer os passos anteriores, considerando agora os objetos de apoio encontrados.

[10] Note que esse critério de refinamento difere daquele usado por métodos OO dirigido a dados, que se baseiam na estrutura de dados para efetuar a classificação.

3.5. construir diagramas equivalentes aos da fase anterior, de modo a visualizar a implementação feita.

Para acompanhar a explanação do processo de desenvolvimento será usado como um **Sistema de Controle de Estoque** de um supermercado[11].

4.5 Fase 1: Análise Sob o Ponto de Vista Funcional

A fase inicial de análise de um sistema é sempre cercada de dúvidas. Muitas vezes, nem mesmo os clientes/usuários sabem exatamente o que desejam. Os limites do sistema não estão bem definidos, e a enorme quantidade de informação relevante levantada, através de entrevistas com clientes, relatórios desejados, manuais, sistemas existentes etc., exige alguma forma de organização que facilite o trabalho do analista.

A Análise Essencial consegue lidar muito bem com esse tipo de problema. Além de fornecer um meio adequado de comunicação com o usuário sem perda de exatidão e completeza, define claramente os limites do sistema, através da identificação dos eventos e a elaboração das respectivas respostas. Para isso, conta com diversas heurísticas que guiam o processo e balizam a modelagem.

Os quatro passos principais sugeridos por McMenamin e Palmer [MEN84] para especificação de uma nova essência do sistema são tratadas nas próximas subseções[12]. Durante a modelagem, somente as heurísticas da AE devem ser utilizadas, sem interferências das próximas fases de desenvolvimento, evitando-se assim, a condução da modelagem para um mapeamento posterior em objetos. Esse direcionamento da especificação nem mesmo é necessário, uma vez que a transição entre as fases não se trata de um mapeamento direto da funcionalidade especificadas em objetos ou métodos de objetos.

Como será visto posteriormente, a especificação produzida nessa fase servirá de roteiro para a atuação dos objetos essenciais do sistema.

4.5.1 *Passo 1.1 - Identificar o Objetivo do Sistema*

O objetivo de um sistema geralmente é suprir alguma deficiência no ambiente onde vai operar. Para resolver essas deficiências, interage com o ambiente através de respostas planejadas.

Um sistema de reservas de passagens aéreas, por exemplo, tem um conjunto de finalidades: fornecer aos clientes uma informação rápida e precisa a respeito da disponibilidade de vôos, minimizar a falta de disponibilidade de assentos, e minimizar a perda de receita devido aos não-comparecimentos.

A definição de um objetivo para o sistema fornece uma declaração geral de sua finalidade, de forma a concentrar esforços na direção apontada. Geralmente possui algumas características comuns, tais como ser tecnologicamente independente, existir em muitos níveis e ser uma combinação de finalidades menores.

Para um Sistema de Controle de Estoque (SCE), usado como exemplo por esse trabalho, define-se sua declaração geral de finalidade como:

---

[11] A análise mais detalhada, referente a esse projeto, se encontra no Apêndice A.

[12] Salvo menção contrária, todos os conceitos utilizados nas subseções a seguir referem-se implicitamente ao trabalho [MEN84].

Fornecer um acompanhamento automático e preciso da entrada e saída dos produtos, manter o estoque em níveis adequados e minimizar possíveis perdas de produtos perecíveis.

4.5.2 *Passo 1.2 - Identificar Atividades Fundamentais*

Definido o objetivo do sistema, deve-se, então, identificar os eventos aos quais o sistema deve responder para atingir os objetivos. Um **evento** é alguma mudança no ambiente do sistema. Uma **resposta planejada** é um conjunto de ações planejadas que são executadas pelo sistema quando ocorre um determinado evento.

Uma **atividade essencial** consiste de duas partes: uma resposta planejada e uma definição do estímulo da atividade. A resposta planejada é o conjunto de ações efetuadas pelo sistema para executar a atividade. A definição do estímulo permite que a atividade reconheça a sua chegada. Cada atividade essencial só responde a um único evento, sendo um número atribuído a cada atividade de acordo com o evento ao qual responde.

Nesse momento da modelagem, busca-se identificar apenas as atividades fundamentais do sistema, que são apenas um subconjunto de todas as respostas planejadas que o sistema oferece. **Atividades fundamentais** sempre dialogam com entidades externas[13] e são as atividades que fazem cumprir as *expectativas* que o mundo exterior tem do sistema[14], *i.e.*, executam uma tarefa que é parte da finalidade declarada do sistema.

Em um sistema de respostas planejadas, as respostas são disparadas por eventos que ocorrem no ambiente que envolve o sistema. Os eventos podem ser de quatro tipos:

1. **Externo**: esse tipo de evento é provocado por uma entidade externa do ambiente do sistema. Eventos externos ocorrem em intervalos de tempo imprevisíveis e são identificados através do exame das ações das entidades externas sobre o sistema.
2. **Controle**[15]: é uma variação de um evento externo no qual uma entidade externa solicita informações contidas no sistema ou a respeito da situação do sistema. Estão relacionados ao acompanhamento do funcionamento do sistema.
3. **Temporal**: um evento temporal é provocado pela chegada de determinado momento no tempo em que o sistema deve agir automaticamente, de acordo com um planejamento anterior. É identificado a partir do exame dos horários que governam as ações automáticas do sistema. Como exemplo, tem-se o Evento 3 da Tabela 4.2, que indica que a cada intervalo fixo de tempo deve ser feito um inventário do estoque.
4. **Temporal Relativo**[16]: é um tipo de evento temporal relativo a uma determinada informação que o sistema armazena. Como exemplo temos o Evento 4 da Tabela 4.2 que indica que a cada in-

[13] Uma atividade fundamental sempre produz uma mensagem para o mundo exterior.

[14] Ponto de vista do usuário do sistema.

[15] Em [MEN84], esse tipo de evento é apenas referenciado como Evento Externo.

[16] Em [MEN84], esse tipo de evento é apenas referenciado como Evento Temporal.

tervalo fixo de tempo relativo à validade do produto (informação armazenada) deve-se notificar proximidade do fim da validade.

Para identificação dos eventos, deve-se procurar alterações no ambiente (eventos) para as quais o sistema deve fornecer uma resposta planejada. Deve ficar claro que o sistema não responde a qualquer evento, mas apenas a alguns relacionados de alguma forma ao objetivo definido.

A especificação inicial de uma atividade fundamental deve conter três elementos:

1. **declaração do evento ao qual o sistema deve responder**. O nome do evento deve refletir algum acontecimento no ambiente que disparou uma resposta planejada. *Geralmente* são escritos na forma: `sujeito + verbo ativo + objeto`. (vide Tabela 4.2)
2. **definição do estímulo que caracteriza o evento**. Toda a informação adquirida no momento em que ocorre o evento é, por definição, parte do estímulo. Para um evento externo, o estímulo é o fluxo de dados inicial fornecido pela entidade externa, que deve ser especificado no dicionário de dados. Para um evento temporal, o estímulo é o momento no tempo para o qual a resposta planejada reage.
3. **resposta para o estímulo**. Da mesma forma que o estímulo externo, as respostas para um evento externo ou temporal são feitas através de fluxos de dados, que também devem ser especificados no dicionário de dados.

No caso do SCE, o objetivo definido sugere que, inicialmente, deve-se procurar eventos relacionados à:

1. Entrada de Material;
2. Saída de Material;
3. Manutenção do estoque;
4. Controle de produtos perecíveis em estoque.

Imaginando-se onde o sistema vai operar, um supermercado, podemos definir alguns eventos que disparam atividades fundamentais, descritos na Tabela 4.2.

Tabela 4.2 Relação de eventos para o Sistema de Controle de Estoque

Evento	Tipo
1. Ponto de Venda informa venda de produtos	Externo
2. Fornecedor faz entrega de produtos	Externo
3. Momento de realizar inventário	Temporal
4. Momento de informar produtos com prazo de validade a vencer	Temporal Relativo
5. Momento de solicitar reposição de estoque	*Temporal Relativo?*

O Evento 5 exemplifica um evento que foi identificado de modo *errado* durante a fase inicial de análise, pois a necessidade de reposição deve ser verificada a cada saída de produto[17]. Ele será mantido durante a explicação dos passos da análise para mostrar como as heurísticas da AE tratam esse tipo de falha na modelagem.

Um evento identificado, que não havia sido sugerido pelo objetivo do sistema, é a necessidade de se realizar inventários periódicos, de modo a sincronizar a quantidade dos produtos armazenados pelo sistema, com o que realmente existe em estoque. Essa discrepância entre o calculado e o real se deve ao fato de ocorrer perda de produtos por furto, dano, produtos que foram retirados como sendo outros etc., perdas essas que estão além do alcance do acompanhamento imediato do sistema.

Como exemplo dos passos de análise serão modeladas as respostas planejadas aos eventos 1 e 5. A Figura 4.3 apresenta a resposta parcial do sistema ao evento `Ponto de Venda informa venda de produtos`. A Figura 4.4 apresenta a resposta planejada parcial para o evento `Momento de solicitar reposição de estoque`.

O primeiro sinal de que há algum problema no particionamento do Evento 5 se encontra na definição do estímulo do evento. Como o sistema deve controlar a necessidade de reposição do estoque, o estímulo não é externo, dessa forma deveria ser um estímulo de um evento temporal. No entanto, não fica claro qual o momento no tempo em que essa resposta planejada deve ser disparada. A resolução completa dessa questão é definida no próximo passo.

[17] A saída do estoque através de uma venda é apenas um tipo de saída de produto. Como exemplo de outros tipos de saída, podemos apresentar a saída por perda ou furto, que são efetuadas como resposta a outro evento quando, da mesma forma, é feita a verificação da necessidade de reposição.

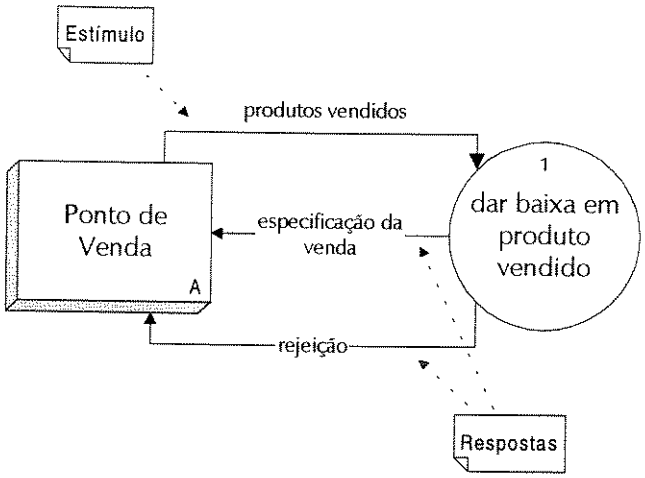


Figura 4.3 Definição do estímulo e resposta para o evento "Ponto de Venda informa venda de produtos".

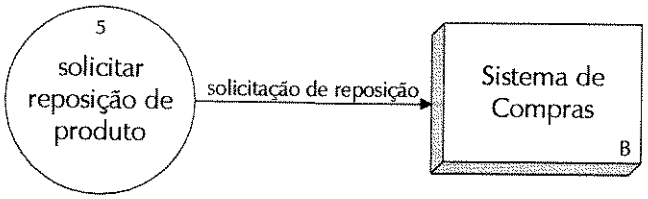


Figura 4.4 Resposta inicial ao evento erroneamente identificado.

A especificação dos fluxos fica:

Fluxo	produtos vendidos
Origem-Destino	A-1
Descrição	relação de produtos vendidos cujas quantidades restantes devem ser atualizadas no estoque
Composição	número do caixa + { código do produto + número do lote + quantidade }

Fluxo	especificação da venda
Origem-Destino	1-A
Descrição	informações relativas aos produtos vendidos
Composição	{ código do produto + nome do produto + seção em que se encontrava + preço unitário }

Fluxo	rejeição
Origem-Destino	1-A
Descrição	indica que a solicitação de venda de determinados produtos foi negada
Composição	{ código do produto + mensagem de erro + mensagem de providência a tomar }

Fluxo	solicitação de reposição
Origem-Destino	5-B
Descrição	relação de produtos para os quais devem ser feitos pedidos de reposição de modo a manter o estoque dentro de limites desejados
Composição	{ código do produto }

4.5.3 *Passo 1.3 - Identificar a informação que o Sistema Necessita*

O processo de identificação da informação necessária ao sistema é iniciado pelo exame de um conjunto de elementos de dados produzidos por uma resposta fundamental. Procura-se analisar como a informação de saída foi obtida, de modo a se identificar a informação de entrada necessária para produzir essa informação, bem como a fonte dessa informação. Toda a informação necessária a uma atividade fundamental tem de vir do ambiente do sistema, no momento da ocorrência do evento através do estímulo correspondente, ou da memória essencial.

A memória essencial consiste em todos os dados que o sistema precisa conhecer para que possa realizar suas atividades fundamentais. Esses dados são produzidos pelo próprio sistema (em um processamento anterior qualquer) ou capturados do mundo exterior (para uma utilização futura, quando esses dados não estarão disponíveis).

Freqüentemente, como resultado desse processo de especificação da informação que deve estar contida na memória essencial, pode-se constatar que informação necessária é produzida apenas por outros eventos diferentes, ainda não identificados, que devem ser modelados de modo a manter a informação do sistema consistente e completa. Isso implica uma continuidade do processo de desenvolvimento, em que um evento leva a identificação de outro. Esse processo já conduz a modelagem de forma a cobrir toda a informação que o sistema precisará.

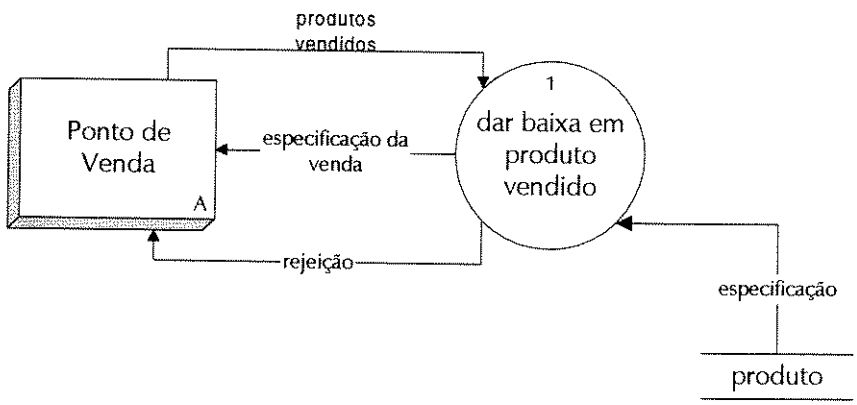


Figura 4.5 Resposta planejada parcial para o evento 1, produzida no passo 1.3.

A resposta ao Evento 1 (Figura 4.5) obtém parte de sua informação do ambiente, através da entidade Ponto de Venda, e parte da memória essencial, a especificação do produto. A necessidade da especificação do produto indica que em um outro momento, outra atividade essencial cadastrou essa informação no sistema. Entretanto, como não foi identificado nenhum evento que dispare uma resposta planejada que

mantenha essa informação, novos eventos devem ser identificados. Essa preocupação é melhor tratada na próxima subseção.

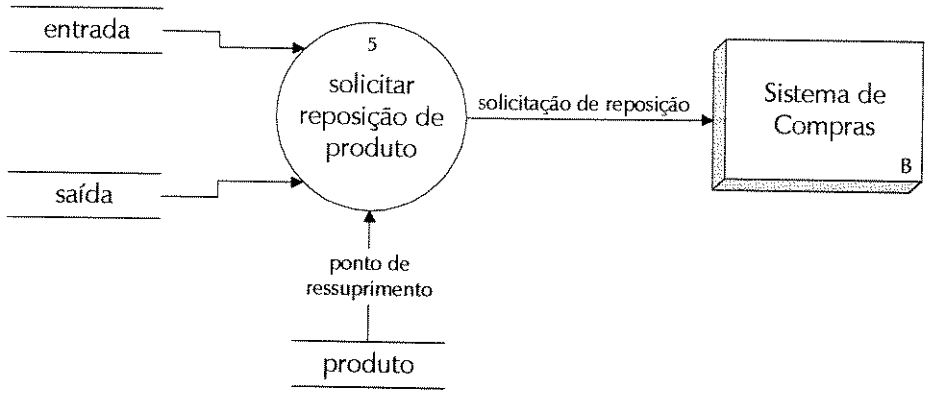


Figura 4.6 Resposta planejada para o evento 5, produzida pelo passo 1.3.

Para o Evento 5 do SCE, ilustrado pela Figura 4.6, pode-se perceber que na verdade não há nenhum estímulo temporal nem externo associado ao evento. A atividade "solicitar reposição de produto" executa um acompanhamento contínuo do nível do estoque e, quando constata um produto com nível abaixo do mínimo, solicita a reposição. Considerando-se apenas a heurística de tecnologia perfeita, não haveria problema de se modelar a resposta dessa forma. Contudo, existe ainda a heurística que define que o sistema deve ficar inativo enquanto espera a ocorrência de um evento. Se isso não ocorre, é porque houve uma falha de modelagem pois alguma atividade essencial não forneceu uma resposta completa a um evento.

Logo, deve-se eliminar este evento e anexar a funcionalidade de sua resposta planejada a outra(s) resposta(s) planejada(s), identificadas a partir de ações que possam posicionar o estoque abaixo do nível desejado. Para a resposta ao evento 1, por exemplo, a resposta planejada completa, após registrar a saída de produtos por venda, deve verificar e providenciar, se necessário, a reposição do produto (Figura 4.7).

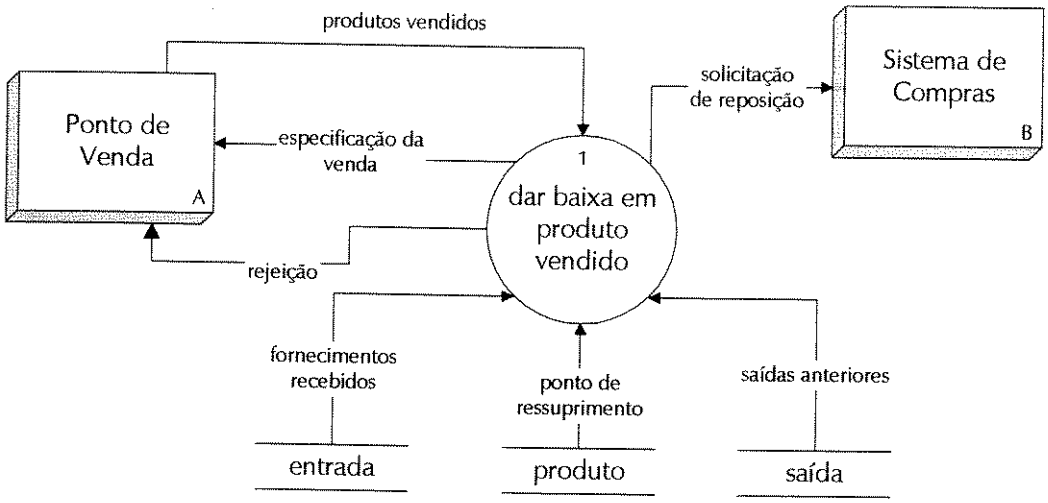


Figura 4.7 Resposta para o evento 1, incorporando a funcionalidade da resposta planejada para o evento 5.

4.5.4 *Passo 1.4 - Garantir o Ciclo de Vida da Informação do Sistema*

Toda a informação com a qual o sistema precisará lidar já foi especificada nos passos anteriores da fase de análise. Agora, resta garantir que o ciclo de vida completo de toda essa informação seja coberto. Para isso, deve-se identificar quando cada elemento de informação deve ser armazenado pela primeira vez, quando ele é atualizado e quando deve ser removido da memória essencial.

Cada resposta planejada já identificada também pode contribuir para a manutenção do ciclo de vida da informação do sistema. Muitas vezes, o resultado do processamento de uma resposta planejada pode tomar uma das seguintes formas:

1. o resultado será necessário à própria atividade ou a outra atividade fundamental em outro momento;
2. o resultado modifica uma informação armazenada;
3. o resultado torna determinada informação obsoleta.

Logo, deve-se rever as respostas já planejadas em busca desse comportamento e documentá-lo.

No exemplo do SCE, a atividade "dar baixa em produto vendido" resultante (Figura 4.8 e Figura 4.9) deve registrar a saída do material na memória essencial para um processamento futuro quando essa informação será útil, por exemplo, para calcular a quantidade restante de produto. O pedido feito pelo Sistema de Compras também deve ser armazenado pois será necessário durante o recebimento do pedido, que deverá ser conferido.

Nem sempre o ciclo de vida consegue ser coberto apenas com as atividades essenciais já identificadas. Outras atividades são necessárias, chamadas **atividades custodiais**, que tratam apenas do armazenamento e atualização da memória essencial.

A diferença básica entre as atividades modeladas até o momento e as atividades custodiais é o tipo de resposta que a atividade fornece:

1. atividade puramente fundamental: apenas fornece uma resposta ao ambiente, sem interferência no conteúdo da memória essencial.
2. atividade puramente custodial: a resposta é apenas uma atualização da memória essencial. Geralmente relacionada com operações simples de inclusão, alteração, exclusão de informação que é usada em outro momento por uma atividade fundamental.
3. atividade mista (fundamental e custodial): fornece uma resposta ao ambiente e atualiza a memória essencial.

As atividades modeladas até esse ponto do desenvolvimento são, na maioria, mistas.

Grande parte dos eventos adicionais que disparam atividades custodiais já foram identificados na fase anterior, ao se estabelecer a especificação da informação necessária de cada atividade fundamental. O evento 6 (Tabela 4.3), por exemplo, dispara uma atividade custodial que trata da inclusão da especificação de um novo produto na memória essencial. A necessidade de uma atividade que tivesse essa finalidade foi detectada no passo anterior, quando foi constatada a necessidade da informação relacionada à especificação do produto.

Durante a revisão de todos os eventos identificados, deve-se também procurar por outros eventos externos e temporais, ainda não identificados, que também possam alterar a memória essencial, princi-

palmente os que provocam a obsolescência de informação. Durante o desenvolvimento do SCE, um novo evento temporal foi identificado, para avisar que produtos do estoque com validade vencida devem ser removidos (evento 5, Tabela 4.3).

Alguns eventos adicionais para o SCE são descritos na Tabela 4.3.

Tabela 4.3 Eventos adicionais para o Sistema de Controle de Estoque.

Evento	Tipo
5. Momento de solicitar remoção de produtos vencidos	Temporal Relativo
6. Sistema de Compras cancela pedido de reposição de produto	Externo
7. Fornecedor altera especificação do produto	Externo
8. Fornecedor deixa de produzir produto	Externo
9. Supermercado decide comercializar novo produto	Externo

A especificação completa da atividade 1 "dar baixa em produto vendido" foi dividida em dois níveis (Figura 4.8 e Figura 4.9) para evitar a sobrecarga de informação em um único nível. Note que alguma informação foi suprimida no nível 1, sem, no entanto, prejudicar o entendimento da resposta básica que a atividade fornece. Detalhes menores, mas não menos importantes, da atividade estão contidos no nível 2.

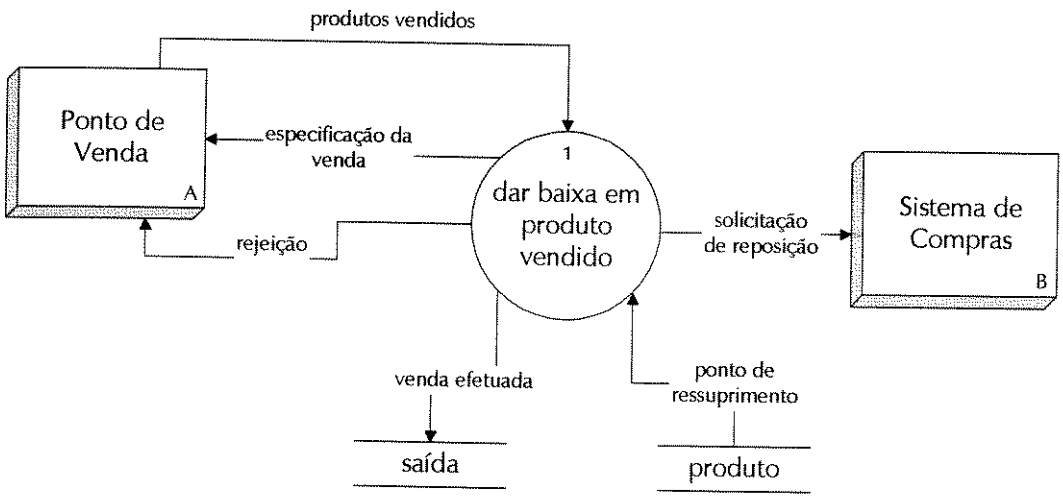


Figura 4.8 Resposta completa (nível 1) para o evento 1.

Processo	dar baixa em produto vendido
Número	1
Tipo	(X) Fundamental - (X) Custodial
Descrição	gerenciar a saída de produtos do estoque realizada através de venda dos mesmos
Especificação	para cada produto vendido faça obter especificações do produto relevantes à venda informar especificações do produto ao caixa

registrar saída por venda
se necessário, solicitar reposição de produto

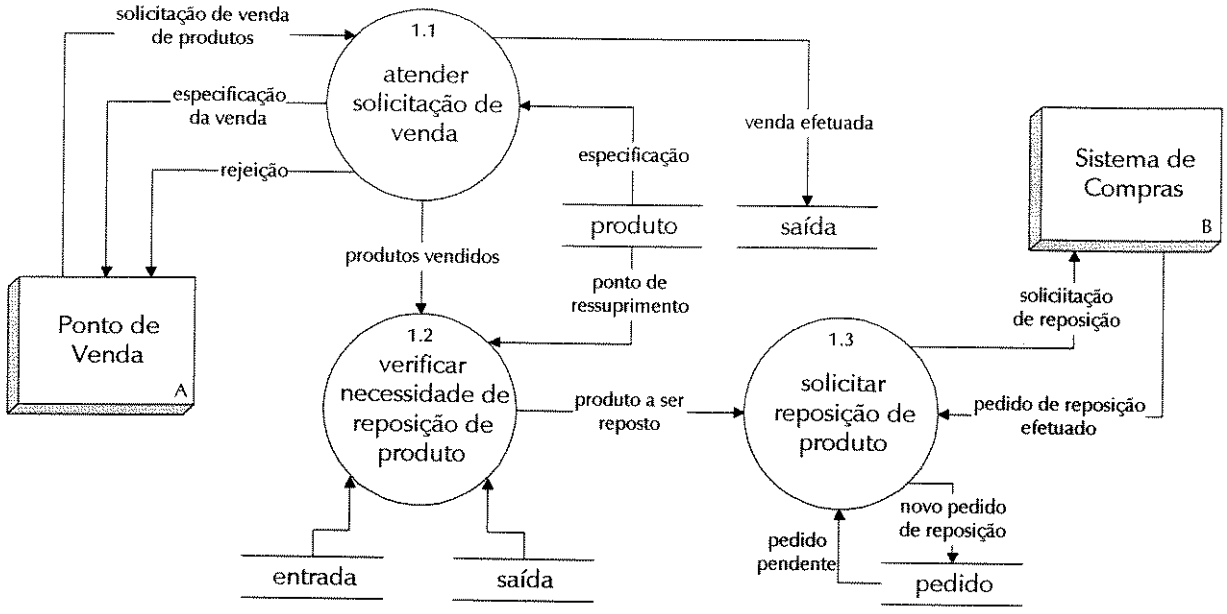


Figura 4.9 Resposta planejada completa (nível 2) para o evento 1.

Processo	atender solicitação de venda
Número	1.1
Tipo	(X)Fundamental - (X)Custodial
Descrição	fornecer informações ao caixa de supermercado referente à venda de produtos
Especificação	para cada produto faça obter nome, seção, preço de venda do produto se produto não estiver cadastrado, informar erro informar ao caixa nome, seção e preço de venda do produto registrar saída de produto

Processo	verificar necessidade de reposição de produto
Número	1.2
Tipo	( )Fundamental - (X)Custodial
Descrição	verifica se a saída de produtos realizada posicionou o nível em estoque do produto abaixo do ponto de ressuprimento
Especificação	para cada produto faça calcular quantidade disponível de produto se quantidade disponível < ponto de ressuprimento então providenciar reposição de produto

Processo	solicitar reposição de produto
Número	1.3
Tipo	(X)Fundamental - (X)Custodial
Descrição	solicitar ao sistema de a compra de determinado produto em falta no estoque.
Especificação	se não houver nenhuma reposição pendente de produto, então solicitar ao Sistema de Compras a compra de produto registrar pedido efetuado por Sistema de Compras para futura conferência

A explosão de uma "bolha" em várias, não é arbitrária. A função "dar baixa em produtos vendidos", em nível 1, possui uma especificação mais geral do que deve ser feito em resposta a esse evento. Essa especificação é detalhada em um nível 2, onde são fornecidas informações mais detalhadas de cada passo da resposta. O processo de explosão pode indicar que novas informações são necessárias ou produzidas, devendo-se, nesse caso, repetir os passos anteriores de análise em um processo de refinamento da modelagem.

Dificilmente se consegue todas as atividades apropriadas na primeira vez que se percorre todas os passos de análise. Uma certa iteração na qual atividades são refinadas, eventos inseridos, nova informação identificada etc., sempre é necessária. Para isso, deve-se percorrer novamente os passos, mas não necessariamente na mesma ordem.

4.6 Fase 2: Análise das Respostas Planejadas Sob o Ponto de Vista dos Objetos Essenciais do Sistema

Nesta fase do desenvolvimento é feita a transição entre o paradigma Funcional e o OO. Para que a informação obtida na análise seja suavemente reorganizada em objetos, esse trabalho propõe que as respostas planejadas, fornecidas pela AE, sejam usadas em conjunto com os cartões CRC[18] para a obtenção de objetos. Contudo, apenas a utilização desses métodos não garante uma transição suave. Os cartões CRC devem antes ser preparados para lidar apenas com a essência do sistema[19]. Para isso, esse trabalho estende o conceito de tecnologia perfeita da AE ao paradigma OO através do conceito de objetos essenciais, discutido posteriormente.

A transição entre as fases é muito facilitada pela afinidade natural entre a AE e os cartões CRC. Os cartões CRC são uma ferramenta de análise de cenários enquanto a AE fornece cenários já completamente especificados para a análise através dos cartões.

---

[18] Em [BEC89], um exemplo em que um DFD foi usado como base para o projeto com cartões CRC é apenas citado, sem nenhuma referência aos modelos envolvidos ou ao processo de transição. Não foi encontrado na literatura nenhum outro caso de utilização conjunta entre métodos estruturados e cartões CRC.

[19] De acordo com [BEC89], as características descritas em um cartão CRC, Classe/Responsabilidades/Colaborações, foram definidas com o objetivo de se conseguir uma independência da linguagem de programação ou ambiente operacional, similar à obtida pela Análise Estruturada [DeM78] através dos processos, fluxos e depósitos de dados. Embora o resultado esteja próximo do obtido por esse trabalho, a não utilização do conceito de tecnologia perfeita permite que outras características tecnológicas sejam incorporadas na modelagem dos objetos.

Na proposta original do cartão CRC [BEC89] e mesmo nas extensões presentes em [WIR90] e [RUB92], há muita complexidade envolvida em um mesmo momento: além de se procurar identificar cada funcionalidade do sistema (um possível cenário de uso), identificam-se os objetos, as funcionalidades desses objetos em relação a funcionalidade do sistema (responsabilidades) e as colaborações entre esses objetos. Os objetos iniciais são extraídos de uma descrição do sistema (processo cujas falhas foram analisadas no capítulo anterior) e refinados posteriormente [WIR90] [RUB92].

Com a utilização das respostas planejadas, fornecidas pela AE, a complexidade com a qual os cartões CRC devem tratar em um mesmo momento fica extremamente reduzida pois:

1. a funcionalidade total do sistema já está descrita pelo conjunto de respostas planejadas;
2. toda as responsabilidades que devem ser cumpridas pelos objetos a serem identificados já estão definidas na especificação de cada resposta planejada;
3. toda a informação necessária ao sistema já está especificada;
4. todos os objetos iniciais necessários para a análise com os cartões já estão definidos.

Logo, a análise pelos cartões CRC deve apenas tratar da distribuição igualitária das responsabilidades e definição das colaborações entre os objetos.

Segundo [WIR90], o relacionamento entre os objetos do sistema se baseiam em um **contrato** (conjunto de serviços oferecidos por um objeto) entre um **cliente** (solicitante dos serviços) e um **servidor** (fornecedor dos serviços). A **responsabilidade** de um objeto abrange todos os contratos que fornece através do conhecimento que possui e das ações que pode realizar. A **colaboração** entre os objetos se dá quando um objeto requisita o serviço de outros objetos para que possa cumprir suas responsabilidades.

O resultado que deve ser obtido é um conjunto de objetos essenciais que executam apenas a essência do sistema. Para isso, é necessário que se estenda os conceitos definidos pela AE de *tecnologia perfeita* ao paradigma OO[20].

O conceito de tecnologia interna perfeita[21] serve para identificar a essência do sistema pois elimina restrições de implementação da modelagem que, de outra forma, poderiam mascarar uma característica essencial necessária [MEN84].

Nesse contexto, um **objeto essencial** é um objeto construído com tecnologia perfeita, com a qual questões relacionadas à tecnologia de implementação tornam-se irrelevantes. Durante a identificação desse tipo de objeto, deve-se desconsiderar limitações tecnológicas tais como as relacionadas à persistência, protocolos de acesso e comunicação, plataforma de desenvolvimento, sistema operacional, linguagem de programação, velocidade de processamento, distribuição em rede etc.

Com a utilização dos objetos essenciais na transição entre as fases de análise e projeto, consegue-se manter a essência do sistema, mesmo após a distribuição das responsabilidades e colaborações, pois ocorre apenas a reorganização da informação da análise, para o ponto de vista do paradigma OO. Como não existe nenhuma referência à tecnologia de implementação, os objetos essenciais mantêm-se imunes a alte-

[20] Originalmente, os componentes básicos de uma tecnologia perfeita são o processador perfeito e a memória (*container*) perfeita [MEN84].

[21] O conceito de tecnologia perfeita não se estende ao ambiente no qual o sistema opera. Para fins de modelagem, apenas a tecnologia interna é considerada perfeita.

rações tecnológicas, restringindo o impacto desse tipo de alteração aos objetos das fases posteriores de projeto, tratados na Fase 3.

4.6.1 Passo 2.1 - Identificar os Objetos Essenciais Iniciais

As entidades presentes no DFD (entidades externas) e as entidades do MER[22] comporão o conjunto de objetos essenciais que serão usados inicialmente nessa fase. Estas entidades representam candidatos naturais a objetos pois fornecem e mantêm a informação presente na memória essencial, restando apenas que o comportamento adequado seja associado[23].

Para o SCE, o MER obtido é apresentado na Figura 4.10. Note que foi evitado o uso de herança nesse modelo. Como ainda não foi associado comportamento às classes, um processo de generalização ou especialização se basearia exclusivamente na estrutura das entidades, o que poderia direcionar prematuramente os objetos essenciais a uma hierarquia pré-concebida. A hierarquia final dos objetos será obtida somente ao final dessa fase, baseada apenas nas responsabilidades comuns entre os objetos, e não na estrutura.

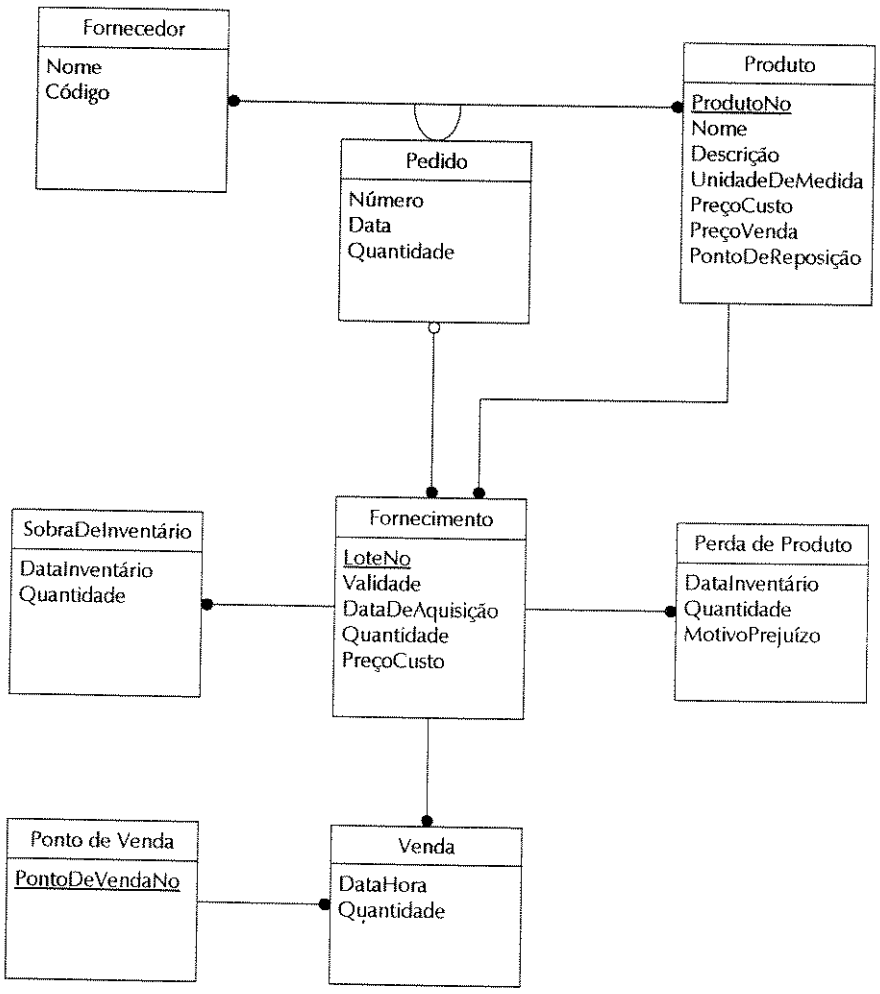


Figura 4.10 MER produzido na fase 1, evitando-se uso de herança baseada na estrutura de dados.

[22] O MER é obtido ainda na fase anterior, à medida que a informação presente na memória essencial é definida.

[23] A distribuição das responsabilidades entre os objetos essenciais iniciais é tratada na próxima etapa.

Nem todas as entidades do MER e as entidades externas presentes no DFD são necessárias para a análise de cada resposta planejada. Apenas as entidades relacionadas à informação tratada em cada atividade essencial precisam ser consideradas ao se distribuir responsabilidades.

4.6.2 Passo 2.2 - Distribuição de Responsabilidades e Identificação de Colaborações entre Objetos Essenciais

Para cada objeto essencial inicial é definido um cartão CRC. Como a técnica de análise através dos cartões incentiva o trabalho em equipe, recomenda-se que se forme uma pequena equipe e se distribua os cartões entre os membros.

Para melhor explicar o processo de análise que será empregado, cabe aqui uma analogia entre esse passo de desenvolvimento e uma peça de teatro.

A especificação do sistema fornecida pela AE funciona como uma peça de teatro, em que cada atividade essencial representa um ato completo dessa peça. Os objetos essenciais representam os atores e o roteiro é fornecido pela especificação completa de cada ato (atividade essencial). Contudo, os papéis (responsabilidades) de cada ator não estão estabelecidos e, durante o ensaio, serão estabelecidos esses papéis e com quem cada ator vai contracenar (colaborações). Como os atos da peça não são monólogos, deve-se distribuir os papéis tão equilibradamente quanto possível entre os atores.

Durante a sessão de desenvolvimento, sugere-se que se mantenha um "diretor" da peça para coordenar o trabalho de cada integrante da equipe que, por sua vez, encenará um ou mais objetos essenciais. À medida que as responsabilidades de cada objeto vão sendo distribuídas e colaborações identificadas, elas devem ser anotadas nos respectivos campos do cartão CRC.

A execução de cada atividade essencial inicia pelo objeto responsável por fornecer o estímulo. Em caso de atividades que respondam a eventos temporais, o objeto mais adequado a iniciar a resposta é aquele mais proximamente relacionado com a informação a ser tratada na atividade.

A seqüência das ações executadas por cada atividade essencial é descrita pela seqüência do fluxo da informação entre as funções de cada DFD enquanto as responsabilidades que devem ser cumpridas estão expressas no nome e na mini-especificação de cada função.

A interação entre os objetos se dá na forma de colaborações. Muitas vezes um objeto não possui todo o conhecimento necessário para executar uma determinada responsabilidade sua e precisa da ajuda de outros objetos, que possuem esse conhecimento, para que possa fornecer determinado serviço [WIR90].

Conforme [WIR90], cada responsabilidade deve ser mantida com informação relacionada sempre que possível. Entretanto, existirão casos em que uma responsabilidade é na verdade um conjunto de responsabilidades menores relacionadas. Nesses casos, o melhor a fazer é dividir ou compartilhar essas responsabilidades mais específicas entre outros objetos mais apropriados. Cada responsabilidade que é compartilhada entre classes também representa uma colaboração entre estas classes.

Na atividade 1 do SCE "dar baixa em produto vendido", por exemplo, a responsabilidade de registrar a venda de produtos é composta por responsabilidades menores tais como obter especificação de produtos, registrar saída de produtos e verificar necessidade de reposição de produto. A divisão dessas responsabilidades foi feita entre os objetos `Venda` e `Produto` (Figura 4.11) pois estão melhor preparados para lidar com cada uma delas.

Ao se identificar colaborações, [WIR90] sugere que se façam determinadas perguntas para cada responsabilidade de cada classe:

1. A classe é capaz de preencher essa responsabilidade sozinha?
2. Se não, o que ela precisa?
3. De qual outra classe ela pode obter o que necessita?

A interação entre os objetos na resolução de uma resposta planejada, obtida durante as sessões com os cartões CRC, podem ser documentadas através de um Diagrama de Colaboração entre Objetos (DCO). Nele são registrados a ordem em que os objetos colaboram, quais as responsabilidades envolvidas, e os dados que são trocados entre as classes.

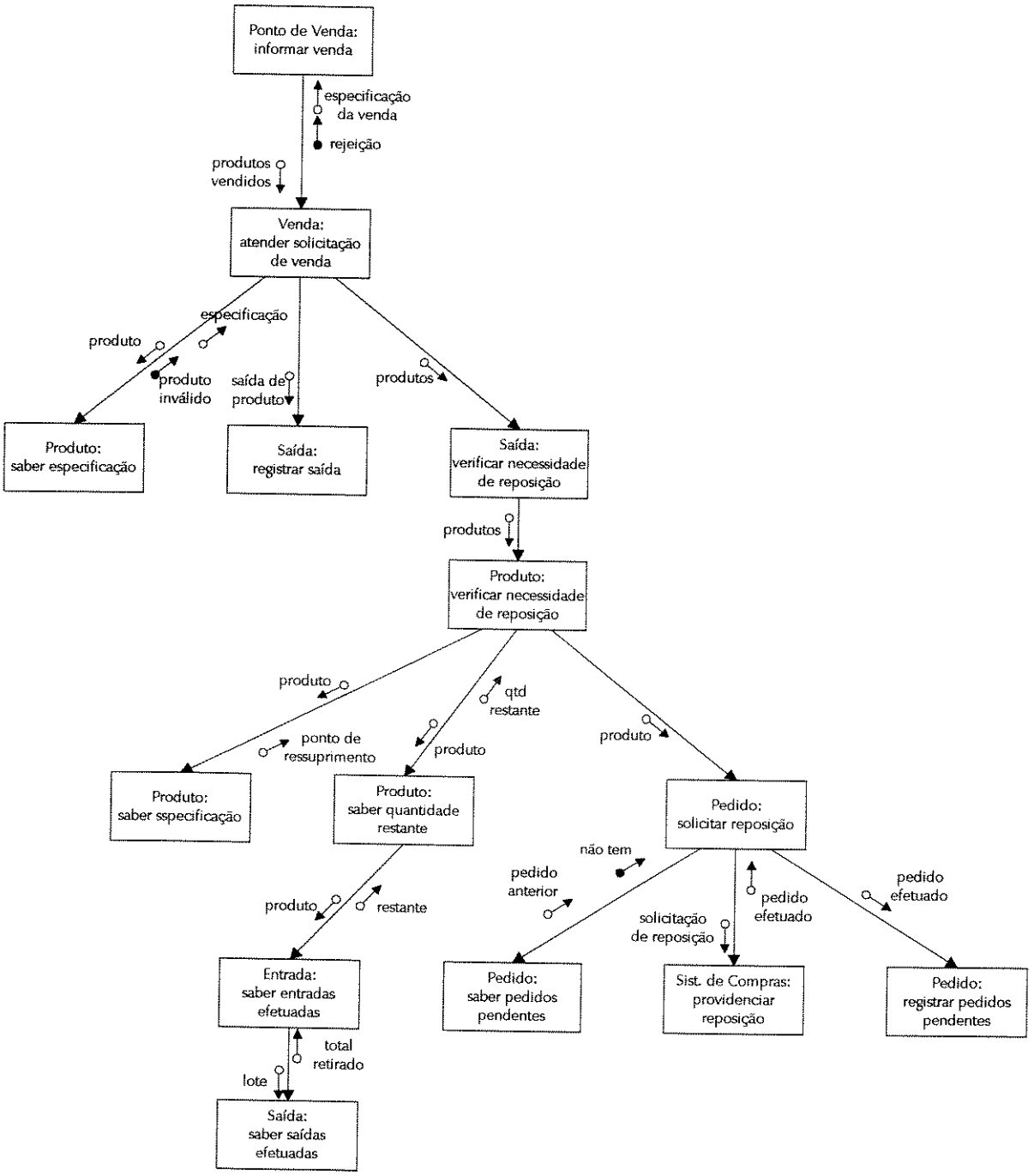


Figura 4.11 Diagrama de Colaboração de Objetos (DCO) para a resposta ao evento 1 do SCE.

A Figura 4.11 ilustra um DCO para a resposta ao evento 1 da Tabela 4.2. A leitura do diagrama é feita do mesmo modo que em um diagrama de estrutura do Projeto Estruturado [PAG80]. A utilização de uma variação do diagrama de estrutura do Projeto Estruturado possibilita uma visualização clara das responsabilidades envolvidas e as colaborações entre os objetos em cada resposta planejada, sem comprometer o conceito de tecnologia perfeita.

Cada caixa do diagrama de colaboração de objetos contém o nome do objeto e uma de suas responsabilidades. Embora os fluxos de dados entre os objetos estejam especificados, note que cada responsabilidade declarada não representa um protocolo de acesso ao objeto. Uma única responsabilidade de um objeto poderá ser fornecida, posteriormente, por vários protocolos de acesso.

Os fluxos de dados presentes no DCO representam a informação que um cliente precisa enviar a um servidor de modo que este possa fornecer o serviço requisitado.

Na Figura 4.11, o evento é provocado pelo objeto `Ponto de Venda` que produz o estímulo com a relação de produtos que estão sendo vendidos. De acordo com a especificação, uma responsabilidade que deve ser associada é atender solicitação de venda. O objeto mais apropriado para fornecer esse serviço é o objeto `Venda`. Assim, atribui-se a responsabilidade `informar produtos a serem vendidos` ao objeto `Ponto de Venda`, que colaborará com o objeto `Venda`, que por sua vez terá a responsabilidade de `atender a solicitação de venda`.

O atendimento a uma solicitação de venda é um conjunto de responsabilidades que inclui desde a validação dos produtos até a solicitação de reposição de produto, caso o estoque fique abaixo de um determinado nível. Conforme descrito anteriormente, estas responsabilidades foram divididas entre vários objetos, que colaboram entre si.

Ao final da análise de cada resposta planejada é obtido:

1. um conjunto de cartões CRC onde são descritas as responsabilidades e colaborações de cada objeto (Figura 4.12);
2. um Diagrama de Colaboração entre Objetos que descreve a seqüência de interações entre os objetos ao executar uma resposta planejada.

Pedido (CLASSE)	(SUPERCLASSE)
RESPONSABILIDADES	COLABORADORES
verificar necessidade de reposição de produto	Produto
solicitar reposição de produto	Sistema de Compras
saber pedido de reposição de produto pendente	
conferir recebimento de fornecimento de produto	
dar entrada em fornecimento válido	Fornecimento
rejeitar entrada inválida de fornecimento	
saber data do pedido, quantidade pedida, fornecedor	
controla todo o processamento e informação relacionada à reposição de produtos do estoque. (DESCRIÇÃO)	

Figura 4.12 Exemplo de cartão CRC de um objeto essencial obtido ao final da análise de todas as respostas planejadas.

4.6.3 Passo 2.3 Refinamento da Hierarquia e Elaboração de Contratos

Ao se chegar nessa fase, a modelagem das respostas planejadas segundo os objetos do sistema já deve ter alcançado certa estabilidade. Deve-se, então, procurar refinar a modelagem obtida, "enxugando" a modelagem.

Cada classe pode ser vista como um conjunto de responsabilidades [WIR90]: desse modo, a fatoração de responsabilidades comuns produz uma hierarquia de classes na qual uma subclasse herda as responsabilidades de uma ou mais superclasses.

Ao final da fatoração das responsabilidades, classes que não tenham responsabilidades devem ser descartadas. Se uma classe não contribui com nenhuma nova responsabilidade, nem redefine uma responsabilidade herdada, ela não tem utilidade para o sistema [WIR90].

Em contrapartida, classes que tenham responsabilidades, mas que não acrescentem novos atributos devem ser mantidas. O objeto Venda (Figura 4.13), por exemplo, que apesar de não acrescentar nenhum novo atributo aos herdados, fornece serviços diferenciados em relação a superclasse Saída. Caso fosse eliminada unicamente por não acrescentar nenhum atributo à estrutura[24], o sistema ou deixaria de oferecer esses serviços, ou teria de acomodá-los em outros objetos, piorando a qualidade da modelagem.

Ao invés de incentivar o uso da herança, conforme feito em [WIR90], recomenda-se que se privilegie o uso de delegação. Na delegação, a responsabilidade de uma classe é redirecionada a outra classe que realmente a fornecerá. Diversas vantagens do uso de delegação para extensão e adaptação de objetos podem ser encontrados em [THO97].

A hierarquia de classes resultantes pode ser descrita através do modelo de objetos do OMT [RUM91]. Embora não mostre as responsabilidades relacionadas a cada classe, contém mais informação que a notação empregada em [WIR90], que apenas ilustra o relacionamento de herança entre as classes.

[24] Em uma hierarquia baseada na estrutura dos objetos, não há sentido em se ter uma classe sem novos atributos. Uma solução comum para se manter uma classe, é criar atributos adicionais meramente artificiais, de modo a satisfazer o critério.

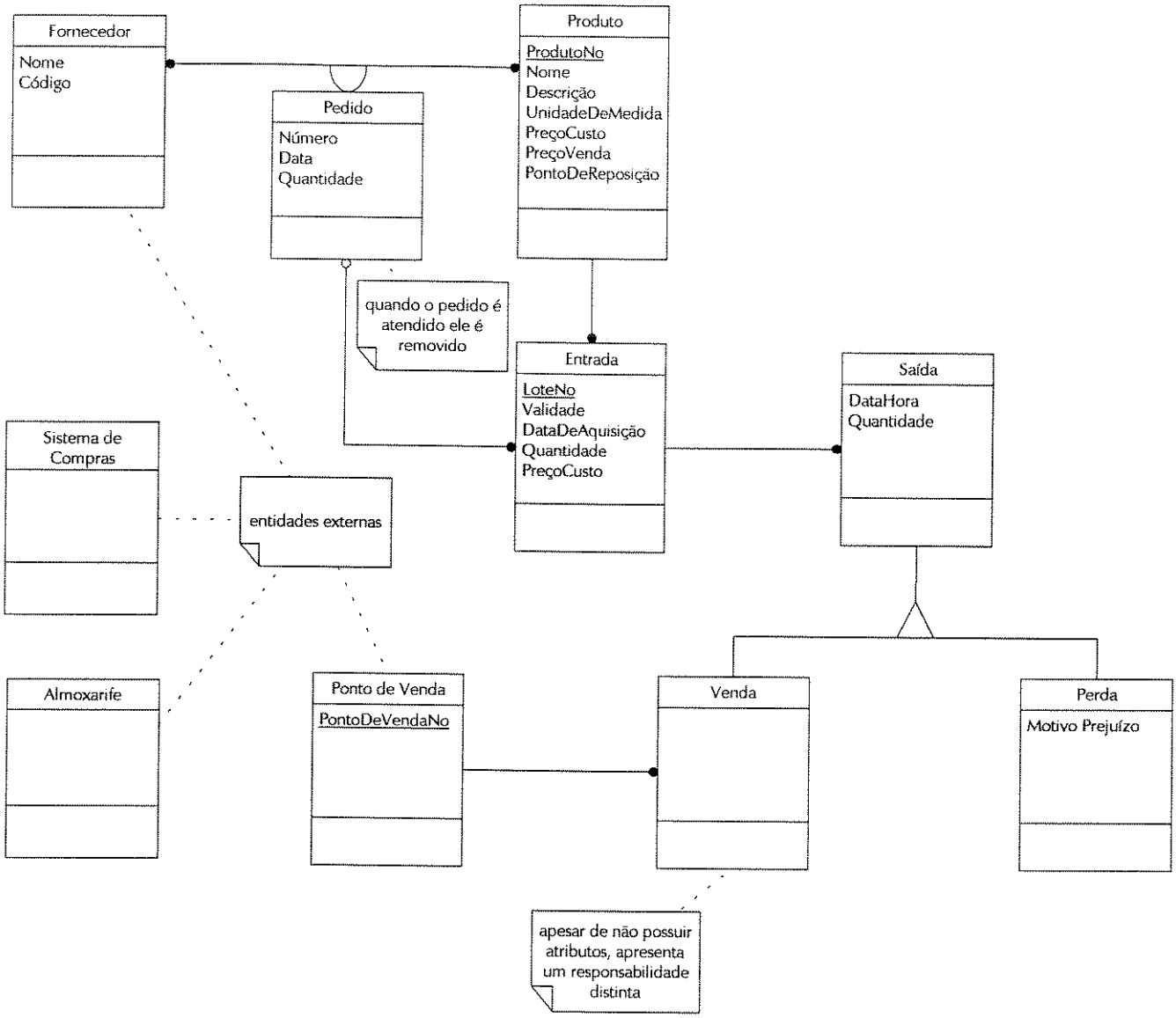


Figura 4.13 Modelo de estrutura dos objetos essenciais do SCE.

Após refinada a hierarquia de classes, pode-se definir os contratos para cada classe. Conforme [WIR90], contratos funcionam como um mecanismo de abstração adicional que diminui a complexidade associada ao projeto dos objetos. Um contrato agrupa um conjunto coeso de responsabilidades relacionadas que podem ser usadas por um cliente[25]. Também pode ser visto como um indicador geral dos usos distintos que podem ser feitos de uma classe. Os relacionamentos entre classe, contratos e responsabilidades, obtidos ao final desse passo, é ilustrado na Figura 4.14.

[25] O RDD não fornece uma definição clara de o que representa um conjunto coeso de responsabilidades, contando apenas com o "bom senso" do projetista. Um estudo posterior deve definir melhor o conceito de coesão de contratos, talvez usando conceitos já consagrados do Projeto Estruturado [PAG80].

Figura 4.14 Relação entre classe, contrato e responsabilidades.

A definição de contratos é feita pelos Diagramas de Colaboração entre Classes. Nele pode-se ver claramente os serviços que um objeto requisita de outro em uma determinada mensagem enviada. Com todos os DCOs do sistema pode-se mais facilmente ter a visão de interações entre os objetos e dessa forma definir os contratos mais adequados.

Algumas das responsabilidades associadas a uma classe não representam comportamento ou serviços disponíveis a outros objetos. Esse tipo de responsabilidade é denominado de **responsabilidade privada**[4] (MIRELI). Para o objeto Saída (Figura 4.11) a responsabilidade verificar necessidade de negociação só é acessada pelo próprio objeto, indicando que se trata de uma responsabilidade privada.

Diferentemente do RDD, neste passo é recomendado que também se organizem as responsabilidades privadas em contratos privados, de forma que estes contratos também sejam visualizados em um nível de especificação detalhado de uma classe, onde sejam mostrados detalhes de funcionamento interno.

A adição ou alteração de funcionalidade do sistema pode acarretar mudanças na visibilidade de um serviço. A responsabilidade saber pedido pendente do objeto Pedido pode ser identificada como uma responsabilidade privada se for considerado apenas um subconjunto de atividades essenciais do sistema. Caso uma nova atividade essencial estabeleça que uma entidade externa possa fazer uma consulta de pedidos pendentes, essa responsabilidade continuará existindo, mas passará a ser pública.

O processo de identificação de contratos deve começar pelas subclasses. Novos contratos devem ser definidos apenas para subclasses que adicionem novas funcionalidades significativas usadas por clientes diferentes (MIRELI). A classe Venda, por exemplo, apenas fornece uma especialização do tratamento de seção de produtos que serve a um mesmo fim que a seção de produtos fornecida pela classe Saída. Portanto, não há necessidade de se definir um novo contrato. A Venda pode oferecer esse serviço dentro de um mesmo contrato definido pela sua superclasse Saída (Figura 4.15).

A cada contrato deve ser dado um nome e uma numeração (MIRELI). O nome de um contrato deve expressar de modo genérico o conjunto de responsabilidades que contém.

[4] Isso ocorre devido à necessidade de classificação mais detalhada de responsabilidades, de maneira similar à classificação usada por linguagens de programação, em responsabilidades privadas ou protegidas.

Saída (CLASSE)	SUPERCLASSE
RESPONSABILIDADES	COLABORADORES
1.1 registrar saída de produto de um determinado fornecimento	Fornecimento
1.2 rejeitar saída inválida	Produto, Fornecimento
2 saber quantidade, data e produto retirado	
P.1 verificar necessidade de reposição de produto	Produto
representa a saída de certa quantidade de produto referente a um fornecimento específico.	DESCRIÇÃO

Venda (CLASSE)	SUPERCLASSE: Saída
RESPONSABILIDADES	COLABORADORES
1.3 informar preço de produto vendido	Produto
1.4 atender a solicitação de saída de produto por venda	Entrada, Produto
representa uma venda de certa quantidade de produto referente a um lote.	DESCRIÇÃO

Figura 4.15 O objeto Venda mantém o mesmo contrato de Saída apesar de refinar o tratamento dado à saída de produto.

A numeração dos contratos sugerida pelo RDD segue uma mesma seqüência para todas as classes do sistema. Entretanto, isso leva a uma dependência da documentação das classes a um sistema específico, dificultando o trabalho de utilização da classe em outro sistema. Nesse trabalho, recomenda-se que a numeração dos contratos deva seguir a numeração seqüencial dentro de uma classe ou mesmo ramo de herança de classes (Figura 4.15), caracterizando, assim, que os contratos estão relacionados a uma classe em particular. A numeração de acordo com a classe também favorece a idéia de que uma classe se comporta como um componente dentro do sistema.

Após definidos os contratos, pode-se representar as colaborações entre os objetos do sistema em uma forma gráfica através de um Grafo de Contratos e Colaborações entre Objetos (GCC), que é uma extensão do Grafo de Colaborações (GC) inicialmente proposto pelo RDD.

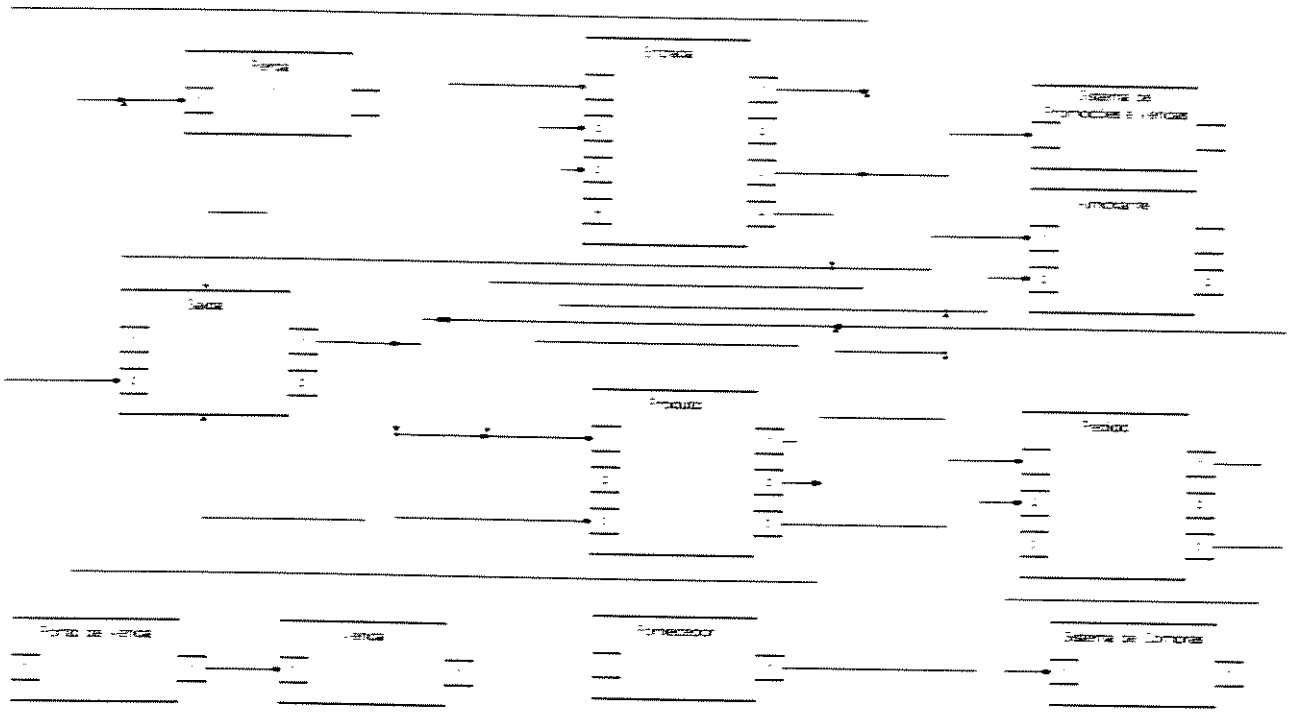


Figura 4.16 Grafo de Contratos e Colaborações para o SOE

A modelagem feita até o momento apenas considerou a tecnologia perfeita, obtendo, assim, a essência do sistema. Deseja-se que as alterações introduzidas devido à adaptação da essência do sistema à realidade de implementação não desfigurem a modelagem já feita. Os modelos de análise devem permanecer absolutamente imunes a requisitos de implementação, e a essência deve continuar servindo como especificação do sistema.

Para isso, a transição entre a essência e a implementação será feita a partir de objetos derivados dos objetos essenciais, que servirão de isolamento entre as primeira fases, na qual as atividades e objetos essenciais do sistema foram definidos, e a terceira fase, na qual as características de implementação serão inseridas.

A partir de cada objeto essencial é obtido um **objeto adaptado**, que será a encarnação[27] do objeto essencial em uma determinada realidade de implementação (Figura 4.17). Um objeto adaptado tem como objetivo adequar a essência do objeto essencial ao conjunto de restrições de uma determinada implementação do sistema, respeitando toda a especificação definida, que permanece inalterada.

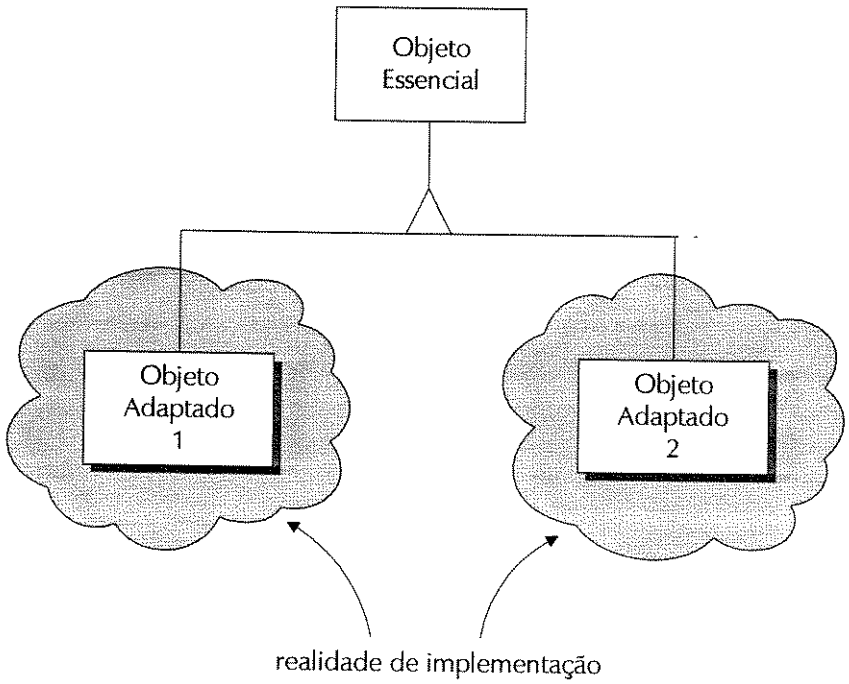


Figura 4.17 Relação entre Objeto Essencial e Objeto Adaptado.

A rastreabilidade entre os modelos de Análise e Projeto é mantida através da equivalência entre os objetos essenciais e os objetos adaptados., já que existe uma correspondência entre os atributos e métodos de um objeto adaptado e atributos e responsabilidades de um objeto essencial.

[27] O termo encarnação é usado, da mesma forma que em [MEN84], para representar a materialização de um conceito.

Da mesma forma, existe uma correspondência entre os novos modelos produzidos nesta fase e os modelos da fase anterior. Os modelos produzidos nesta fase acrescentam características de implementação necessárias, fornecendo modelos equivalentes aos da fase anterior[28].

Um objeto adaptado não conseguirá sozinho resolver todos os problemas relacionados à implementação, e nem deve. A estratégia que será usada procura delegar grande parte dos problemas que os objetos adaptados encontrarão a outros objetos[29], chamados **objetos de apoio à implementação**. Os objetos de apoio têm como única finalidade auxiliar os objetos adaptados a adequarem sua essência à realidade de implementação.

Como exemplo de objetos de apoio à implementação tem-se:

1. tipos de dados abstratos de dados: listas, filas, pilhas, árvores etc.
2. objetos pertencentes a *framework* de apoio fornecido por um ambiente de implementação. O Delphi™[30], por exemplo, fornece diversos objetos que podem ser usados e estendidos tais como formulários, botões, objetos para lidar com Banco de Dados etc.

Na verdade, um objeto de apoio pode ser composto por objetos menores, dependendo do tipo de problema que deve solucionar. Quando um problema é complexo o suficiente, pode ser dividido em partes menores que são delegadas a outros objetos de apoio menores, cabendo ao objeto de apoio inicial apenas o gerenciamento da resolução do problema.

Principalmente em relação a objetos de apoio mais complexos, a relação entre um objeto adaptado e um objeto de apoio costuma seguir um padrão de projeto (em inglês, design pattern). Em [GAM95] são encontrados diversos padrões de colaborações entre objetos que podem ser identificados e usados nessa fase, tais como padrões estruturais (*e.g.*, adaptador, ponte, decorador), padrões de comportamento (*e.g.*, iterador, observador, comando) e padrões de criação (*e.g.*, construtor, protótipo). Como será visto, o problema que um objeto adaptado deve delegar a um objeto de apoio muitas vezes já sugere a escolha de um determinado padrão de projeto.

O ponto central dessa fase é como implementar cada responsabilidade do objeto adaptado. Os problemas geralmente relacionados com cada responsabilidade são:

1. especificação em nível de implementação dos fluxos e elementos de dados com os quais cada objeto trabalha;
2. definição dos métodos correspondentes a cada responsabilidade;
3. otimização e implementação de cada método.

Para cada tipo de problema um tipo de objeto de apoio costuma aparecer. Como, em um único trabalho, não é possível lidar com todas as nuances tecnológicas que porventura aparecerão durante o projeto, as subseções a seguir procuram apresentar os problemas e as respectivas soluções mais comuns.

---

[28] No Apêndice A, é possível visualizar a correspondência entre um DCO e um Diagrama de Estrutura para uma mesma atividade essencial.

[29] Neste trabalho, a técnica de delegação é empregada como parte integrante do método, ao contrário de outros métodos OO (e.g., OMT, OOAD-Booch), em que a técnica é sugerida como uma alternativa de modelagem e apenas em determinadas situações.

[30] Delphi é marca registrada da Borland International, Inc.

4.7.1 Definir Características de Implementação de Fluxos e Elementos de Dados

A primeira dependênciacom que um objeto adaptado tem de lidar é a dos tipos de dados com que vai trabalhar. Na especificação da fase de Análise apenas os elementos de dados haviam sido identificados, sem uma preocupação explícita com características de implementação tais como tipo e tamanho.

Cada elemento de dado deve ser detalhado através de especificações relacionadas à implementação, tais como tipos da linguagem de programação ou do banco de dados a serem utilizados. Através dessa especificação é possível observar se algum tipo de dado mais complexo ou específico será necessário para tratar, por exemplo, um elemento de dado que estará codificado. Nesse caso, um objeto de apoio é criado para tratar desse problema.

Da mesma forma, a especificação dos fluxos de dados auxiliam a identificação de objetos de apoio para tratar a informação relacionada. Geralmente, os objetos de apoio relacionados à fluxos de dados são tipos abstratos de dados tais como listas, conjuntos, árvores, etc.

4.7.2 Definir Protocolos de Classes

Para um objeto essencial, uma responsabilidade é descrita como uma declaração geral do serviço ou conhecimento que deve ser fornecida. No entanto, para um objeto adaptado cada responsabilidade é cumprida através de um ou mais métodos de um objeto. Para o objeto Saída, por exemplo, a responsabilidade saber especificação de saída de produto compreende saber os dados referentes à data, quantidade, produto e lote. Assim, o objeto adaptado Saída deve fornecer métodos que informem esses dados.

Ao se considerar a definição dos protocolos dos métodos de um objeto, é necessário levar em consideração a linguagem de programação que será usada. Algumas permitem certos tipos de métodos que outras não permitem.

Através da especificação de implementação dos fluxos de dados definidos no passo anterior junto com a especificação fornecida pelo DCO, é mais fácil identificar os possíveis protocolos de acesso a um objeto. Para o objeto `Venda`, por exemplo, a responsabilidade atender venda de produtos requer que o fluxo produtos vendidos seja recebido como parâmetro e seja devolvida uma relação de especificações e produtos não aceitos e respectivas justificativas. Assim, pode-se definir um método para o objeto `Venda` com o seguinte protocolo[31]:

```
procedure atenderSolicitaçãoDeVenda(
                CaixaNo              : Integer,
                ProdutosVendidos     : TProdutosVendidos,
          var   EspecificaçãoDeVenda: TEspecificaçãoDeVenda,
          var   RejeiçãoDeVenda      : TRejeiçãoDeVenda );
```

4.7.3 Problemas Relacionados a Colaborações

Os fluxos de dados trocados entre os objetos indicam que tanto o objeto que envia os fluxos, quanto os objetos que recebem os fluxos têm conhecimento do objeto de apoio (se ele existir) relacionado ao fluxo.

Infelizmente, o objeto relacionado ao fluxo de dados muitas vezes tem de lidar com outros problemas além dos relacionados ao elementos de dados. Alguns tipos específicos de problemas estão relaciona-

[31] Exemplo baseado na linguagem Object Pascal.

dos principalmente à adaptação da comunicação entre um objeto adaptado e um outro objeto, que geralmente está fora dos limites do sistema (uma entidade externa), já que objetos adaptados dentro do mesmo sistema têm menos chance de requerer uma adaptação da comunicação entre eles.

As entidades externas que haviam sido consideradas objetos essenciais na fase anterior, podem requerer que a informação sofra uma adaptação. A troca de informação com uma pessoa, por exemplo, exige que a informação seja tratada através de uma interface gráfica. Outro exemplo, é a comunicação entre o objeto `Ponto de Venda` e o objeto `Venda`, ambos do SCE, que é feita através de uma rede, e, portanto, tem de lidar com protocolos de acesso, divisão da informação em pacotes etc.

Em ambos os casos, o mesmo objeto que trata o fluxo também deve tratar todos os outros problemas envolvidos na comunicação. Dessa forma, a comunicação que era feita diretamente entre os objetos, é feita de forma indireta, através de um tradutor.

4.7.4 *Implementação e Otimização de Métodos*

Sem dúvida alguma, o ponto mais problemático do projeto. Isolados os problemas de interface dos métodos, passagem de parâmetros e colaboração, ainda restam os problemas relacionados à implementação dos serviços que um método fornece.

Três problemas típicos encontrados são:

1. persistência;
2. otimização;
3. reconhecimento de eventos temporais.

4.7.4.1 Persistência

O problema da persistência aparece quando um objeto precisa guardar determinadas informações que serão usadas posteriormente. Geralmente, um Sistema Gerenciador de Banco de Dados (SGBD) relacional é empregado para esse fim, exigindo, assim, uma estratégia de mapeamento do objeto para uma tabela. Uma classe pode precisar de mais de uma tabela para armazenar seus dados, implicando uma estratégia para armazenamento e recuperação da informação distribuída.

Da mesma forma que os outros problemas, as questões relativas a persistência podem ser delegadas a objetos de apoio. O Delphi™, por exemplo, fornece um seu *framework*, objetos apropriados para lidar com banco de dados relacionais, que auxiliam bastante o trabalho de delegação.

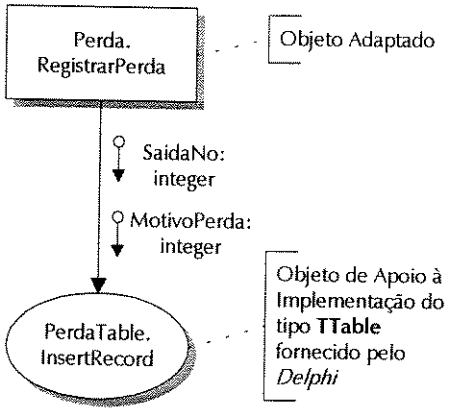


Figura 4.18 Exemplo de implementação de persistência usada no SCE.

A Figura 4.18 ilustra uma solução usada no SCE para o problema de persistência do objeto `Perda`.

4.7.4.2 Otimização

Na modelagem da essência do sistema, uma informação podia ser obtida instantaneamente, independentemente da complexidade ou velocidade do processamento dessa informação. No mundo real, entretanto, tais fatores podem implicar a inviabilidade de um processamento diferente toda vez que a informação é necessária.

Considere como exemplo a responsabilidade do objeto `Produto` de saber a quantidade restante em estoque. Se a cada vez que se requisita essa informação for necessário todo o processamento envolvido (Figura 4.11), o tempo de resposta será proibitivo.

Uma solução é manter a informação calculada sempre disponível para ser usada quando necessária. Esse tipo de informação pode ser calculado uma vez e mantido atualizado através de pequenas modificações, que possuem um custo de processamento muito menor. No caso do objeto `Produto`, a quantidade restante pode ser mantida em arquivo e atualizado a cada entrada ou saída de material.

Esse tipo de solução exige um sincronismo entre cada operação que altera a informação calculada, operações estas que podem estar em vários objetos, e o objeto que fornece a informação calculada.

No exemplo do SCE, o sincronismo deve ser feito a cada operação de entrada ou saída de produto, quando o objeto `Produto` deve ser informado das variações para que possa atualizar a quantidade restante calculada.

Nesse exemplo, como apenas 3 objetos, todos conhecidos, estão envolvidos, pode-se incluir responsabilidades de implementação aos objetos `Entrada` e `Saída` para que informem ao objeto `Produto` qualquer variação da quantidade em estoque de produto. Entretanto, quando vários objetos estão envolvidos, ou não se conhece exatamente quais são esses objetos, recomenda-se o sincronismo através da alteração dos objetos adaptados envolvidos conforme o padrão de projeto *observer* [GAM95].

4.7.4.3 Reconhecimento de Eventos Temporais

Na modelagem da AE, o estímulo temporal que caracteriza um evento temporal é fornecido automaticamente pela passagem do tempo. Considerando o mundo real, no entanto, é necessário que algum mecanismo informe ao objeto apropriado que determinado momento desejado do tempo chegou.

A dificuldade não está na identificação do momento no tempo pois objetos temporizadores são comumente encontrados em *frameworks* de ambientes de desenvolvimento, possibilitando que este tipo de serviço seja prestado. O problema é quando informar a um objeto temporizador o tempo desejado para notificação.

Para um evento temporal simples (absoluto) esse problema não existe pois os intervalos de tempo envolvidos são fixos. Para um evento temporal relativo, entretanto, a situação é outra. O tempo desejado é relativo a uma informação armazenada. Portanto, a programação de um temporizador está atrelada ao ciclo de vida da informação relacionada ao evento temporal relativo.

Por exemplo, um produto perecível, ao ser recebido pelo SCE, deve programar um objeto temporizador para que os momentos de proximidade do fim da validade e o próprio fim da validade sejam notificados (Figura 4.19).

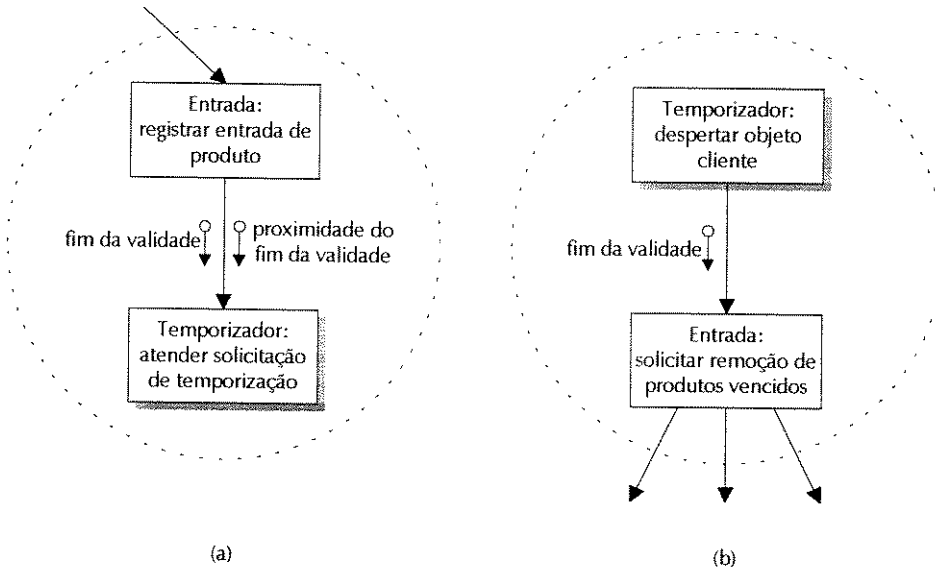


Figura 4.19 Exemplo de mecanismo de notificação de evento temporal.

4.7.5 *Refinamento dos Objetos de Apoio à Implementação*

Os mesmos passos definidos para os objetos adaptados, devem ser repetidos, agora concentrando-se nos objetos de apoio à implementação. Desse modo, mantém-se um refinamento sucessivo do problema até um nível elementar e facilmente solúvel.

4.7.6 *Elaboração de Modelos de Projeto*

Após realizadas as adaptações necessárias, pode-se construir os modelos equivalentes aos da fase anterior para refletir as mudanças. Para cada DCO pode-se construir um Diagrama de Estrutura [PAG80] equivalente. O GCC pode ser ampliado de modo a abranger os objetos de apoio[32].

4.8 Comparação com Propostas Similares

O desejo de se obter uma distinção clara entre as fases de Análise e Projeto, mas mantendo a rastreabilidade entre os modelos, é uma preocupação comum à maioria dos métodos de desenvolvimento de sistemas. Contudo, apesar de demonstrar uma preocupação explícita com estas questões, não apresentam uma proposta de como se chegar a tais resultados. Outros não apresentam uma proposta completa.

Em [MON92], por exemplo, existe uma proposta de divisão das fases de análise e projeto através da observação do tipo de objeto que é tratado em cada fase. Os chamados objetos semânticos representam coisas ou conceitos pertencentes ao domínio do problema e são tratados durante a análise. Durante a fase de projeto, os objetos semânticos são estendidos e objetos de interface, de aplicação e de base/utilitários são incluídos. Todavia, o trabalho de [MON92] não acrescenta nenhuma proposta nova de como se chegar a tal solução, além de não fornecer nenhuma diretiva concreta de identificação de requisitos falsos e verdadeiros, como os da AE, para se obter objetos semânticos e de domínio da solução.

[32] É proposto como extensão o estudo de notações mais apropriadas para objetos de apoio dentro de um GCC.

Em [JAC92], existe uma preocupação na separação entre os modelos de análise e projeto, com transformações dos objetos de cada fase. Durante a transição, os objetos produzidos na análise são convertidos em blocos que encapsularão todas as mudanças que passaram a ser introduzidas devido às restrições de implementação. Além da mudança de notação, pouco é acrescentado de modo a se manter a distinção entre os modelos de cada fase.

4.9 Considerações Finais

A proposta apresentada de um método híbrido consegue cumprir todos os quesitos desejados que foram propostos.

A separação entre as fases de análise e projeto é claramente observável. Mesmo assim, a transição permaneceu suave devido à afinidade entre os métodos usados como base. As heurísticas presentes na AE foram estendidas para as fases posteriores de desenvolvimento, facilitando o processo como um todo.

A introdução do conceito de objetos essenciais, adaptados e de apoio à implementação possibilita a obtenção de níveis de abstração da complexidade envolvida, semelhantes aos níveis de um DFD. Cada atividade essencial pode ser analisada de diversos pontos de vista, com variados graus de complexidade, dependendo do modelo usado sem perder a rastreabilidade da especificação.

Durante a fase de projeto, novos modelos foram aprimorados e adaptados ao processo de desenvolvimento proposto.

A fase de implementação do projeto (fase 3) fornece alguns passos a serem seguidos e identifica problemas e soluções comumente encontrados. Contudo, adaptações dos passos dessa fase a domínios específicos ainda podem ser melhor explorados.

4.10 Bibliografia

[BAI89] BAILIN, S. C. An Object-Oriented Requirements Specification Method, *Communications of The ACM*. Vol. 32, No. 5, May 1989, pp. 608-623.

[BEC89] BECK, Kent & CUNNINGHAM, Ward. A Laboratory For Teaching Object-Oriented Thinking. *SIGPLAN Notices*. Vol. 24, No. 10, October 1989.

[BOO94a] BOOCH, G. Complexity. In: *Object-Oriented Analysis and Design with Applications*, 2ª Ed., The Benjamin/Cummings Publishing Company Inc., 1994, cap. 4, pp. 145-168.

[BOO94b] BOOCH, G.; Classification. In: *Object-Oriented Analysis and Design with Applications*, 2ª Ed., The Benjamin/Cummings Publishing Company Inc., 1994, cap. 4, pp. 145-168.

[DeM78] DeMARCO, T. *Structured Analysis and System Specification*. New York, Yourdon Press, 1978. *Apud* [BEC89].

[FIC92] FICHMAN, Robert G. & KEMERER, Chris F. Object-Oriented and Conventional Analysis and Design Methodologies - Comparison and Critique. *Computer*, October 1992, pp. 22-39.

[GAM95] GAMMA, Erich *et al. Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company Inc., 1995, pp. 395.

[GEO96] GEORGE, Joseph & CARTER, Bradley, D. A strategy for Mapping from Function-Oriented Software Models to Object-Oriented Software Models. *Software Engineering Notes*, vol 21, No. 2, Março 1996, pp. 56-63.

[HAR87] HAREL, D. StateCharts: A visual formalism for complex systems. In: *Science of Computer Programming*. Vol. 8, No. 3, North Holland, 1987, pp. 231-274. *Apud* [RUB92].

[JAC92] JACKOBSON, I.; Christerson, M.; Jonsson, P e Övergaard, G.; *Object-Oriented Software Engineering: a Use Case Driven Approach*, Reading, Massachusetts, Addison-Wesley, 1992.

[KRA88] KRASNER, Glenn E.; POPE, Stephen T.; A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, Vol. 1, No. 3, August/September 1988, pp. 26-49.

[MEN84] McMENAMIN, Sthephen M. & PALMER, John F. *Análise Essencial de Sistemas*. São Paulo. McGraw-Hill. 1991. pp. 567 (tradução de Essential Systems Analysis, 1984)

[MON92] MONARCHI, David E. & PUHR, Gretchen I.; A Research Typology for Object-Oriented Analysis and Design. *Communications of the ACM*, September 1992, vol. 35 (9), pp. 35-47.

[PAG80] PAGE-JONES, Meilir. *Projeto Estruturado de Sistemas*. McGraw-Hill, 1988, pp. 396. (tradução de *The pratical guide to structured systems design*, 1980)

[PAG89] PAGE-JONES, Meilir & WEISS, Steven. Synthesis: An Object-Oriented Analysis and Design Method. *American Programmer*, Vol. 2, No. 7-8 Summer 1989, pp. 64-67.

[POO95] POO, Danny C. C. & LEE, Shwu-Yi; Domain object identification through events and functions. *Information and Software Technology*, Vol. 37, No. 11, 1995, pp. 609-621.

[RUM91] RUMBAUGH, James et al. *Object-Oriented Modeling and Design*. Prentice Hall International, 1991. pp. 500.

[RUB92] RUBIN, Kenneth S., GOLDBERG, Adele. Object Behavior Analysis. *Communications of the ACM*, Vol. 35, No. 9, September 1992, pp. 48-62.

[SHA93] SHARBLE, Robert C. & COHEN, Samuel S.; The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods. *ACM Software Engineering Notes*. Vol. 18, No. 2, April 1993 pp. 60-73.

[SHU92] SHUMATE, K. & KELLER, M.; *Software Specifications and Design: Disciplined Approach for Real Time Systems*, Wiley Inc., New York, 1992. *Apud* [GEO96]

[SOU9?] D'SOUZA, Desdemond. Behavior-Driven vs. Data-Driven: A Non-Issue?, s.n.t., URL http://www.iconcomp.com, 199?.

[SUL89] SULLY, Phil; Structured Analysis: Scaffolding for Object-Oriented Development. *American Programmer*, Vol. 2, No. 7-8 Summer 1989, pp. 76-82.

[THO97] THORPE, Danny. *Delphi Component Design*. Addison-Wesley Developers Press, 1997, pp. 348.

[VAZ93] VAZQUEZ, Federico. Using Object Oriented Structured Development to Implement a Hybrid System. *ACM Software Engineering Notes*. Vol. 14, No. 4, October 1993, pp. 44-53.

[WIR90] WIRFS-BROCK, Rebecca; WILKERSON, Brian & WIENER, Lauren. Designing Object-Oriented Software. Prentice Hall Inc. 1990, pp. 341.

5. CONCLUSÕES

A idéia de que os métodos OO conseguem, sozinhos, capturar toda a essência de um sistema é equivocada. A própria definição de OOA sugere que um estudo prévio do problema precisa ser feito.

No Capítulo 3, diversas "abordagens iniciais" foram consideradas, resultando em apenas duas que realmente conseguiam capturar os requisitos de um sistema a partir do zero: Análise Essencial[1] e Modelagem *Use Case*. Estas duas abordagens capturam, basicamente, requisitos funcionais de um sistema, reforçando ainda mais a idéia de que o ponto de vista do usuário é mais voltado às funcionalidades que um sistema deve oferecer, do que aos objetos que o compõe.

A primeira contribuição desse trabalho foi mostrar que a necessidade de uma decomposição funcional de muitos sistemas é, mais do que desejável, absolutamente necessária, independentemente do paradigma ou abordagem que serão empregados nas fases posteriores de desenvolvimento.

A efetividade com que a Análise Essencial e a modelagem *use case* capturam os requisitos de um sistema foi analisada através de 3 estudos de caso que confirmaram a superioridade com que a AE produz uma especificação funcional de qualidade e quantidade muito superior àquela produzida pelos *use cases*, embora a especificação da AE ainda possa ser complementada com um maior detalhamento dos aspectos comportamentais de um sistema. Apesar de os *use cases* capturarem diversas características necessárias a um sistema, inclusive aspectos comportamentais, a utilização de apenas uma forma de descrição dessas características e a falta de heurísticas de modelagem comprometem a especificação produzida[2].

Dessa forma, a segunda contribuição desse trabalho foi mostrar que a aceitação da modelagem *use case* como padrão *de facto* por vários métodos, conforme relatado em [RUM94], é demasiadamente precipitada.

A principal razão imaginada para a escolha da modelagem *use case* em detrimento da AE para a análise de um sistema é que os *use cases* são apresentados como pertencentes ao paradigma OO, enquanto a AE faz parte de um conjunto de métodos relacionados ao paradigma funcional. Dessa forma, ao

[1] A Análise Essencial foi considerada uma evolução da Análise Estruturada convencional.

[2] Embora desejável, um estudo que visasse incorporar as heurísticas da AE aos *use cases* não foi abordado por este trabalho.

se utilizar os *use cases* mantém-se um desenvolvimento OO "puro". Essa visão separatista contribui para que a Análise Essencial permaneça isolada dos métodos "puramente" OO.

Alguns trabalhos, em que foram propostas abordagens conjuntas de métodos Estruturados e OO, foram analisados no Capítulo 4, mostrando que ainda existe espaço para um método que forneça uma integração maior entre os diferentes paradigmas.

A proposta do método, apresentado nesse trabalho, visa combinar o melhor que cada paradigma tem a oferecer. Para isso, foram escolhidos métodos mais significativos de cada paradigma como base para um método híbrido. Procurou-se resgatar vários modelos e heurísticas clássicos, e estendê-los ao paradigma OO, mantendo-se a distinção entre diferentes fases de desenvolvimento, transição suave entre essas fases e importância equilibrada entre os paradigmas. O resultado obtido foi uma integração significativamente sinergética entre os paradigmas, que permite um maior controle do processo de desenvolvimento e uma base mais sólida para construção de software de qualidade de um modo mais produtivo. Esta foi a terceira contribuição deste trabalho.

5.1 Extensões

Durante a elaboração do trabalho, diversas extensões foram imaginadas para aprimorar e complementar o estudo feito. Nas próximas subseções, algumas propostas de extensões são apresentadas junto com uma pequena bibliografia sugerida para se iniciar a pesquisa.

5.1.1 Ferramenta Case

Não há dúvida da importância da documentação de todo o desenvolvimento de um sistema. Contudo, sem uma ferramenta CASE é praticamente impossível manter a documentação completa e atualizada, e, portanto, útil, mesmo para sistemas pequenos.

A proposta apresentada favorece a sua automatização através de uma ferramenta CASE pois possui um processo de desenvolvimento claro, com modelos e notações bem definidas.

Existem diversas ferramentas CASE no mercado. Algumas até mesmo possibilitam a adequação e extensão de métodos existentes ou outros métodos. Antes de se iniciar a construção de uma ferramenta CASE a partir do zero, ou mesmo estender uma já existente, é necessário um estudo de viabilidade mais detalhado das existentes em relação a diversas características que uma ferramenta deve ter, tais como:

- controle de versões;
- auxílio na elaboração de casos de teste;
- apoio ao gerenciamento de projeto;
- possibilite o trabalho em equipe;
- integração com o ambiente de implementação;

Bibliografia sugerida: [McC93] [McC96] [PRE92] [SOM92].

5.1.2 Estudo Comparativo entre Estratégias de Distribuição da Inteligência do Sistema entre Objetos

Jackobson *et al.* [JAC92] afirmam que o particionamento dos objetos usados no OOSE (objetos de controle, de entidade e de interface) aumenta a manutenibilidade do sistema. Como foi mostrado no Capítulo 4, este tipo de abordagem concentra a inteligência do sistema em poucos objetos, produzindo uma

modelagem muito próxima à da Análise Estruturada, pois os objetos produzidos encapsulam, basicamente, funções e depósitos de dados, fornecendo, assim, um particionamento apenas cosmético do sistema em objetos. A concentração da funcionalidade em poucos objetos é comum aos métodos OO dirigidos a estrutura de dados. Os métodos OO dirigidos a comportamento, por outro lado, possuem heurísticas para distribuição de responsabilidades entre os objetos do sistema.

A idéia de um sistema com a inteligência mais igualmente distribuída entre os objetos que possui parece estar mais de acordo com a filosofia que rege o paradigma OO. Logo, existe uma contradição entre a afirmação de que o paradigma OO produz sistemas mais manuteníveis, com objetos mais facilmente extensíveis e reusáveis, e a obtenção desses resultados através de um particionamento com objetos "funcionais", i.e., que basicamente encapsulam funções e depósitos de dados.

É necessário, portanto, um estudo comparativo entre as diferentes estratégias de distribuição de inteligência entre os objetos, e suas implicações em termos de reuso, extensão e manutenção, com o objetivo de esclarecer alguns pontos ainda obscuros relacionados aos métodos OO:

1. Quais as características mínimas que o particionamento de um sistema em objetos deve produzir para estar de acordo com o paradigma OO?
2. Qual é o melhor tipo de distribuição de inteligência entre os objetos para se obter melhor manutenibilidade, extensibilidade e reusabilidade?

Em [SHA93] é feito um estudo comparativo entre o particionamento de um mesmo sistema de acordo com um método OO dirigido a estrutura de dados e com um outro método OO dirigido a comportamento, mostrando que os métodos dirigidos a comportamento produzem modelagens menos complexas. Contudo, uma análise complementar entre esses tipos de métodos poderia ser feita, reforçando a semelhança entre o particionamento produzido pelo método OO dirigido a estrutura de dados e o paradigma funcional. Outro ponto a ser melhor estudado é a análise do impacto desses tipos de particionamento em termos de manutenibilidade, extensibilidade e reusabilidade de um sistema.

Bibliografia sugerida: [JAC92] [WIR90] [RUB92] [SHA93].

5.1.3 Estudo a Respeito do Particionamento dos Objetos do Sistema

Existe uma tendência atual de se procurar dividir os objetos de um sistema de acordo com cada *use case* [RUM94]. Espera-se dessa forma, que cada caso de uso utilize apenas um subconjunto de objetos do sistema, de modo a diminuir a complexidade inerente da estrutura do sistema total, que pode chegar facilmente a centenas de objetos relevantes.

Entretanto, em [MEN84], existe uma argumentação contrária ao particionamento da memória essencial por eventos. Entre algumas das razões apontadas está a necessidade de intercomunicação entre as respostas planejadas (equivalentes aos *use cases*), aos quais o sistema responde, ser feita apenas pela memória essencial. Deste modo, a utilização dos mesmos objetos em diferentes respostas planejadas é necessária.

Sugere-se, portanto, um estudo mais profundo a respeito dos pontos de vista apresentados com o objetivo de esclarecer as implicações de cada um em relação ao processo de desenvolvimento, completeza da modelagem, facilidade de obtenção etc.

Bibliografia sugerida: [MEN84], [RUM94].

5.1.4 Integração entre Projeto Estruturado e OOD

Na literatura, é difícil encontrar trabalhos que proponham a utilização conjunta de métodos Estruturados e OO de Projeto. Poucos autores consideram que haja alguma afinidade entre Projeto Estruturado (SD) e OOD. Entretanto, conforme mostrado neste trabalho, o Diagrama de Colaboração entre Objetos possui uma clara semelhança com o Diagrama de Estrutura do SD, sem, no entanto, perder a ligação com o paradigma OO. Dessa forma, sugere-se que modelos e notações pertencentes ao SD também podem ser estendidos e utilizados em uma abordagem híbrida. Embora haja estudos que busquem estender os conceitos de coesão e acoplamento para o paradigma OO (*e.g.*, [PAG92]), os resultados obtidos ainda não são satisfatórios, pois ainda não existe uma aplicabilidade direta dos conceitos sobre os modelos produzidos.

A utilização de responsabilidades, contratos e colaborações também implica uma nova nuance a ser considerada para ser avaliada em termos de coesão e acoplamento. Não existem heurísticas mais rigorosas que guiem a escolha de responsabilidades, nem o agrupamento de responsabilidades em contratos. A própria definição de que um contrato é um conjunto coeso de responsabilidades já sugere que uma aproximação ao SD deve ser estudada. O critério de coesão poderia ser estendido para tratar de responsabilidades e contratos, e o critério de acoplamento para tratar de colaborações entre classes.

Bibliografia sugerida: [PAG80] [PAG92] [SHL92] [WIR90].

5.1.5 Gerenciamento de Projetos

Um método de desenvolvimento deve possibilitar um controle do processo de desenvolvimento, fornecendo métricas para qualidade e para estimativas de recursos envolvidos.

Como extensão do trabalho apresentado, propõe-se a definição de métricas de qualidade (além das fornecidas pela AE) e estimativas de recursos para a proposta apresentada. Sugere-se ainda a inclusão de informações durante o desenvolvimento, já visando a construção de um histórico de desenvolvimento que possa servir como base de análise e refinamento.

Bibliografia sugerida: [MEN84] [FAY96] [HUM95] [PAG90] [SOM92] [PRE92] [WHI94].

5.1.6 Extensão para Modelagem de Comportamento do Sistema

Embora o STD tenha sido incorporado pela Análise Estruturada Moderna [you89?], a Análise Essencial, por ser anterior, não faz uso desse diagrama. Como este trabalho foi baseado na Análise Essencial, não existe uma preocupação explícita na documentação do aspecto comportamental de um sistema. Assim, questões como a interação entre um usuário e o sistema, por exemplo, foram desconsideradas por este trabalho.

Para suprir essa deficiência, recomenda-se um estudo mais detalhado com o objetivo de incorporar, ao método proposto, modelos apropriados à documentação de aspectos comportamentais de um sistema de acordo com cada fase, sem, contudo, alterar as heurísticas presentes no método proposto tais como, tecnologia perfeita, requisitos falsos e verdadeiros etc.

Bibliografia sugerida: [WAR85] [YOU89]

5.1.7 Extensão para Sistemas de Tempo Real

Tanto a AE como os métodos OO possuem especializações para tempo real. A proposta apresentada neste trabalho pode ser estendida de modo a fornecer um método híbrido para desenvolvimento de sistemas de tempo real. A questão principal envolvida é se a transição entre a AE para tempo real e os

métodos OO de tempo real também pode ser feita suavemente através dos cartões CRC. É necessário, também, que se estude a adaptação de métodos OO para tempo real em relação aos conceitos de responsabilidades, contratos e colaborações.

Bibliografia sugerida: [WAR85]

5.1.8 Documentação de Domínios Específicos

As duas primeiras fases de desenvolvimento da proposta apresentada capturam a essência do sistema, sem considerar restrições de implementação. Essa mesma característica pode ser estendida em busca da documentação de domínios de problemas específicos tais como para sistemas de informação, que possuem muitas similaridades que podem ser modeladas em um nível mais alto de abstração.

A busca por um padrão de sistemas de domínios específicos pode auxiliar, de modo similar aos padrões de projeto, a aumentar a produtividade de desenvolvimento de software.

Bibliografia sugerida: [KRU92] [GAM95].

5.2 Bibliografia

[FAY96] FAYAD, Mohamed E. & CLINE, Marshall; Managing Object-Oriented Software Development. *IEEE Computer*, September 1996, pp. 26-31.

[GAM95] GAMMA, Erich *et al. Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Publishing Company Inc., 1995, pp. 395.

[HUM95] HUMPHREY, Watts S. *A Discipline for Software Engineering.* Addison-Wesley Publishing Company, 1995, pp. 789.

[JAC92] JACKOBSON, I.; Christerson, M.; Jonsson, P e Övergaard, G.; *Object-Oriented Software Engineering: a Use Case Driven Approach*, Reading, Massachusetts, Addison-Wesley, 1992.

[KRU92] KRUEGER, C. W.; Software Reuse; *ACM Computing Surveys*, June 1992.

[McC93] McCONNELL, Steve C.; *Code Complete: a practical handbook of software construction.* Microsoft Press, 1993, pp. 857.

[McC96] McCONNELL, Steve C.; *Rapid Development: taming wild software schedules.* Microsoft Press, 1996, pp. 647.

[MEN84] McMENAMIN, Sthephen M. & PALMER, John F. *Análise Essencial de Sistemas.* São Paulo. McGraw-Hill. 1991. pp. 567 (tradução de Essential Systems Analysis, 1984)

[PAG80] PAGE-JONES, Meilir. *Projeto Estruturado de Sistemas.* McGraw-Hill, 1988, pp. 396. (tradução de *The pratical guide to structured systems design*, 1980)

[PAG90] PAGE-JONES, Meilir; *Gerenciamento de Projetos: guia prático para restauração da qualidade em projetos e sistemas de processamento de dados.* McGraw-Hill, 1990, pp. 327.

[PAG92] PAGE-JONES, Meilir; Comparing Techniques by Means of Encapsulation and Connascence. *Communications of The ACM*; september 1992, Vol. 35, No. 9, pp. 147-151.

[PRE92] PRESSMAN, Roger S. *Software Engineering - A Practitioner's Approach*, 3ª Ed., McGraw-Hill International Editions, 1992, pp. 793.

[RUB92] RUBIN, Kenneth S., GOLDBERG, Adele. Object Behavior Analysis. *Communications of the ACM*, Vol. 35, No. 9, September 1992, pp. 48-62.

[RUM94] RUMBAUGH, James. Getting started (Using use cases to capture requirements); *Journal of Object-Oriented Programming*, September 1994.

[SHA93] SHARBLE, Robert C. & COHEN, Samuel S.; The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods. *ACM Software Engineering Notes*. Vol. 18, No. 2, April 1993 pp. 60-73.

[SHL92] SHLAER, S. & MELLOR, S. *Object Lifecycles: Modeling the World in States*. Englewood Cliffs, New Jersey; Yourdon Press, 1992.

[SOM92] SOMMERVILLE, Ian. *Software Engineering*, 4ª Ed., Addison-Wesley Publishing Company, 1992, pp. 649.

[WAR85] WARD, Paul T. & MELLOR, Stephen J.; *Structured Development for Real-Time Systems*. Yourdon Press Computing Series, 1985, Vol. 2, pp. 163.

[WHI94] WHITAKER, Ken. *Managing Software Maniacs: finding, managing, and rewarding a winning development team*. John Wiley & Sons, Inc. 1994, pp. 219.

[WIR90] WIRFS-BROCK, Rebecca; WILKERSON, Brian & WIENER, Lauren. *Designing Object-Oriented Software*. Prentice Hall Inc. 1990, pp. 341.

[YOU89] YOURDON, E. N.; *Modern Structured Analysis*, Prentice-Hall, 1989.

APÊNDICE A - EXEMPLO DE APLICAÇÃO

CONSIDERAÇÕES SOBRE A ANÁLISE REALIZADA

O sistema apresentado não tem por objetivo ser completo, de um ponto de vista comercial. O objetivo é ilustrar a aplicação do método proposto neste trabalho. Apesar da idéia original ser desenvolver completamente o sistema, a falta de uma ferramenta CASE de apoio prejudicou a manutenção da documentação associada. Decidiu-se, então, privilegiar a documentação da parte de Análise e Projeto, correspondente às fases 1 e 3 do método proposto. Apenas as atividades 1 a 6 foram desenvolvidas.

DESCRIÇÃO DO PROBLEMA

Empresas que trabalham com produtos necessitam de uma ferramenta automatizada para acompanhar a entrada, saída e perda de material, de forma a manter um retrato fiel do que acontece no estoque e fornecer informações relevantes para que decisões de gerenciamento possam ser feitos.

Apesar de um Sistema de Controle de Estoque (SCE) poder servir a vários tipos de empresas, a análise de um domínio comum desse tipo de sistema não foi explorada. Ao invés, o SCE especificado concentra-se em servir a um supermercado.

OBJETIVO DO SISTEMA

Fornecer um acompanhamento automático e preciso da entrada e saída dos produtos, manter o estoque em níveis adequados e minimizar possíveis perdas de produtos perecíveis.

RESPOSTAS PLANEJADAS

A identificação dos eventos aos quais o sistema responde funciona como um contexto funcional do sistema. Cada resposta planejada pode ser interpretada como uma meta que o sistema deve alcançar para cumprir sua finalidade.

Para o SCE, os eventos levantados foram:

Evento	Tipo
1. Ponto de Venda informa venda de produtos	Externo
2. Fornecedor faz entrega de produtos	Externo
3. Momento de realizar inventário	Temporal
4. Momento de informar produtos com prazo de validade a vencer	Temporal Relativo
5. Momento de solicitar remoção de produtos vencidos	Temporal Relativo
6. Sistema de Compras cancela pedido de reposição de produto	Externo
7. Fornecedor altera especificação do produto	Externo
8. Fornecedor deixa de produzir produto	Externo
9. Supermercado decide comercializar novo produto	Externo

Uma funcionalidade básica, adicionada ao sistema apresentado, difere bastante de um SCE comumente encontrado no mercado: o controle de produtos perecíveis. Entretanto, para que fosse possível um acompanhamento preciso de cada lote de produto perecível, foi necessário exigir que cada lote recebido, e seus respectivos produtos, seja identificado individualmente.

A limitação tecnológica para a implementação dessa funcionalidade não está no sistema[1], e sim, no ambiente. Atualmente, os produtos já vem com um código de barra impresso na embalagem, seguindo um padrão nacional. Nesse código impresso não há nenhuma informação a respeito do lote ou validade dos produtos, dificultando assim, um possível controle de validade.

Logo, o desejo de um controle de produtos perecíveis impõe um rigor maior durante o recebimento de produto, em que cada lote deve ser devidamente identificado. Dessa forma, esse tipo de controle talvez seja inviável para grandes supermercados, que movimentam um grande volume de mercadorias.

[1] A Análise Essencial apresenta o conceito de tecnologia perfeita, no qual as limitações tecnológicas, que poderiam desviar ou esconder uma funcionalidade necessária, são desconsideradas na fase de análise.

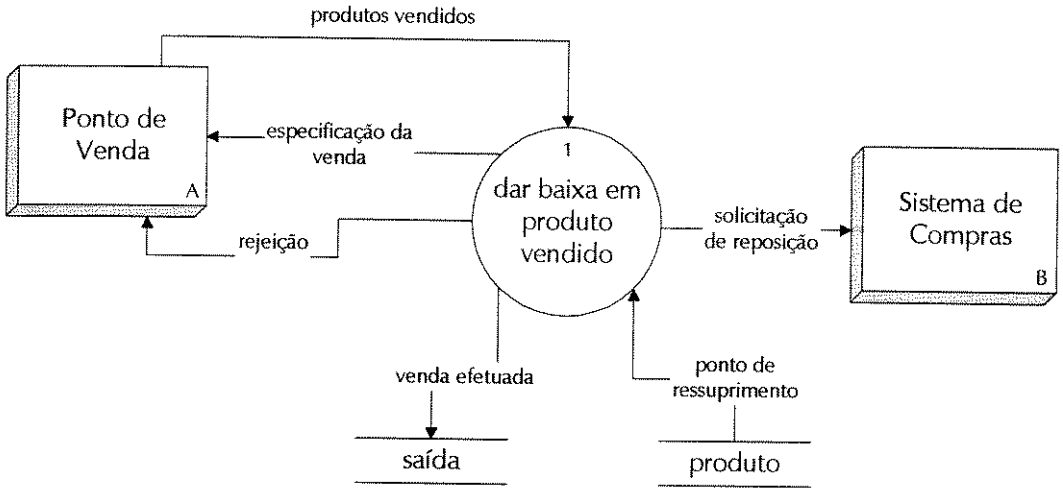
FASE 1:

1. PONTO DE VENDA INFORMA VENDA DE PRODUTOS

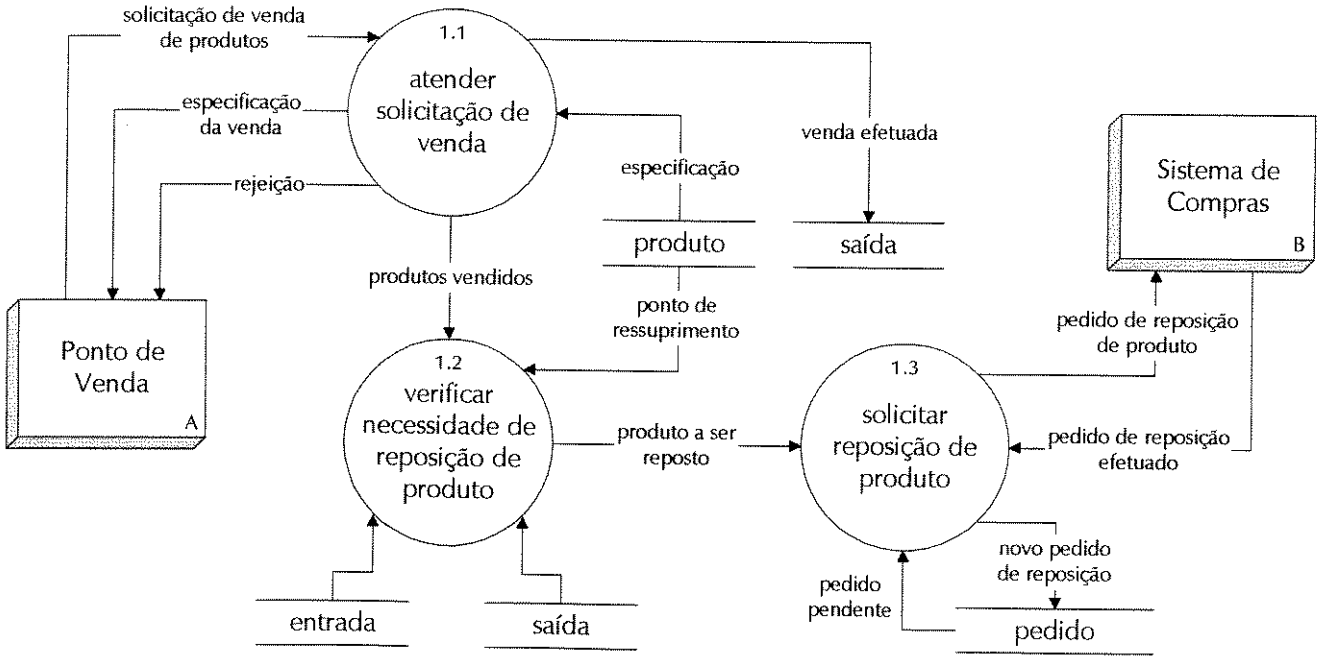
DESCRIÇÃO

Um ponto de venda do supermercado deseja registrar a venda de produtos feita a um cliente do supermercado.

Tipo de Evento: `Externo`



DFD NÍVEL 2



FLUXOS DE DADOS

Fluxo	solicitação de venda de produtos
Origem-Destino	A-1, A-1.1
Descrição	relação de produtos vendidos cujas quantidades restantes devem ser atualizadas no estoque
Composição	número do caixa + { código do produto + número do lote + quantidade }

Fluxo	especificação da venda
Origem-Destino	1-A, 1.1-A
Descrição	informações relativas ao produto vendido
Composição	{ código do produto + nome do produto + seção em que se encontrava + preço unitário }

Fluxo	venda efetuada
Origem-Destino	1-saída, 1.1-saída
Descrição	saída de produtos do estoque
Composição	{ código produto + quantidade saída + número do lote + data e hora + motivo da saída (venda) + número do caixa }

Fluxo	rejeição
Origem-Destino	1-A, 1.1-A
Descrição	indica que a solicitação de venda foi negada
Composição	mensagem de erro + mensagem de providência a tomar

Fluxo	solicitação de reposição
Origem-Destino	1.3-B
Descrição	relação de produtos para os quais devem ser feitos pedidos de reposição de modo a manter o estoque dentro de limites desejados
Composição	{ código do produto }

Fluxo	produtos vendidos
Origem-Destino	1.1-1.2
Descrição	quantidade de produtos retirados do estoque
Composição	{ código do produto + quantidade }

Fluxo	pedido de reposição efetuado
Origem-Destino	B-1.3
Descrição	informa o pedido de reposição efetuado a um fornecedor para que o sistema possa fazer uma conferência futura quando o fornecimento chegar.
Composição	{ código do produto + quantidade solicitada + fornecedor + número do pedido + data do pedido }

Fluxo	pedido pendente
Origem-Destino	pedido-1.3
Descrição	vide B-1.3
Composição	

PROCESSOS

Processo	dar baixa em produto vendido
Número	1
Tipo	(X)Fundamental (X)Custodial
Descrição	gerenciar a saída de produtos do estoque realizada através de venda dos mesmos
Especificação	para cada produto vendido faça obter especificações do produto relevantes à vemda informar especificações do produto ao caixa registrar saída por venda se necessário, solicitar reposição de produto

Processo	atender solicitação de venda
Número	1.1
Tipo	(X)Fundamental (X)Custodial
Descrição	fornecer informações ao caixa de supermercado referente à venda de produtos
Especificação	para cada produto faça obter nome, seção, preço de venda do produto se produto não estiver cadastrado, informar erro informar ao caixa nome, seção e preço de venda do produto registrar saída de produto

Processo	verificar necessidade de reposição de produto
Número	1.2
Tipo	( )Fundamental (X)Custodial
Descrição	verifica se a saída de produtos realizada posicionou o nível em estoque do produto abaixo do ponto de ressuprimento
Especificação	para cada produto faça calcular quantidade disponível de produto se quantidade disponível < ponto de ressuprimento então providenciar reposição de produto

Processo	solicitar reposição de produto
Número	1.3
Tipo	(X)Fundamental (X)Custodial
Descrição	solicitar ao sistema de a compra de determinado produto em falta no estoque.
Especificação	se não houver nenhuma reposição pendente de produto, então solicitar ao Sistema de Compras a compra de produto registrar pedido efetuado por Sistema de Compras para futura conferência

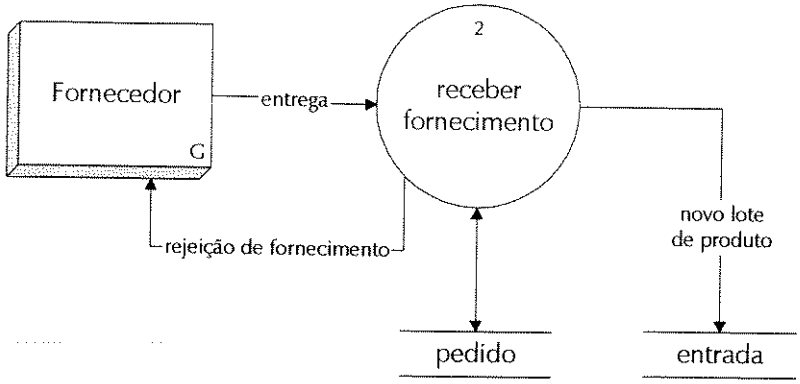
2. Fornecedor faz entrega de Produtos

Descrição

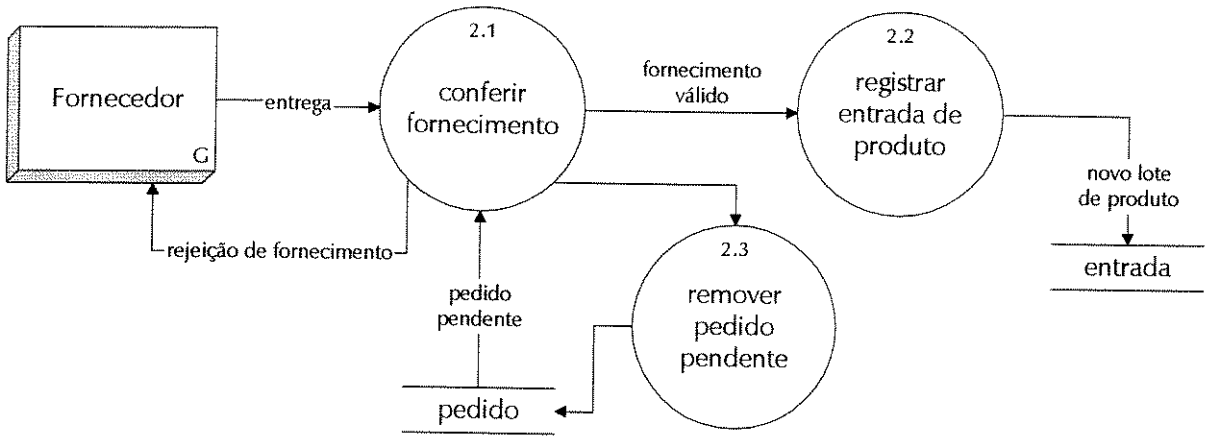
Quando o fornecedor faz uma entrega, deve haver uma verificação do material antes de sua aceitação.

Tipo de Evento: Externo

DFD NÍVEL 1



DFD NÍVEL 2



FLUXOS DE DADOS

Fluxo	entrega
Origem-Destino	C-2, C-2.1
Descrição	entrega de um fornecimento de produto
Composição	número do pedido + fornecedor + { número do lote + código do produto + quantidade + validade + custo }

Fluxo	rejeição de fornecimento
Origem-Destino	2-C, 2.1-C
Descrição	fornecimento entregue não confere com o pedido
Composição	razão da rejeição

Fluxo	fornecimento válido
Origem-Destino	2.1-2.2
Descrição	fornecimento recebido e conferido de acordo com o pedido
Composição	idêntico a fluxo 2-C

Fluxo	novo lote de produto
Origem-Destino	2.2-recebimento
Descrição	lote de produto para reposição do estoque
Composição	{ número do lote + fornecedor + data do recebimento + validade + preço de custo + quantidade }

PROCESSOS

Processo	Receber fornecimento
Número	2
Tipo	(X)Fundamental (X)Custodial
Descrição	
Especificação	conferir recebimento se recebimento válido, então atualizar estoque remover pedido pendente senão, rejeitar estoque

Processo	conferir fornecimento
Número	2.1
Tipo	(X)Fundamental (X)Custodial
Descrição	conferir fornecimento entregue com o pedido anteriormente.
Especificação	obter pedido pendente através do número do pedido se pedido não existir, então rejeitar fornecimento senão, verificar se produto e quantidade conferem se conferir, então aceitar fornecimento retirar pedido pendente senão, rejeitar fornecimento

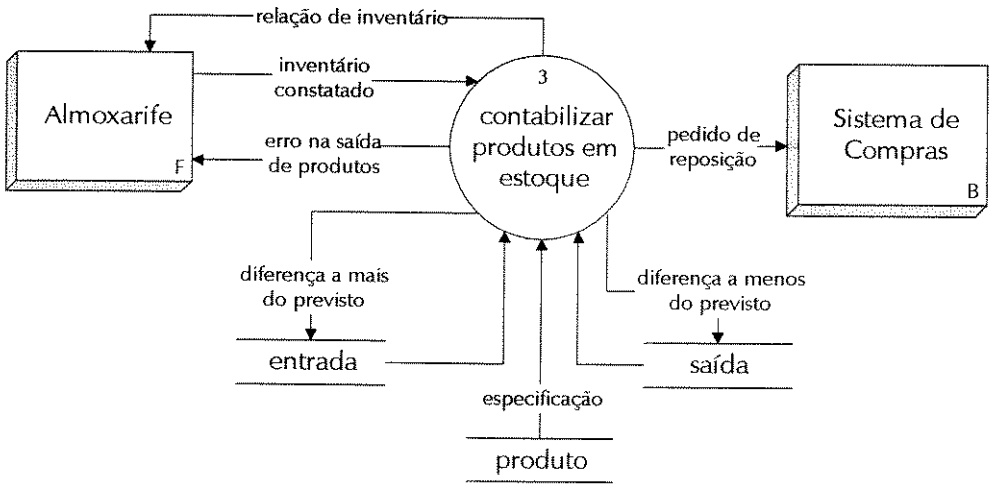
3. Momento de Realizar Inventário

Descrição

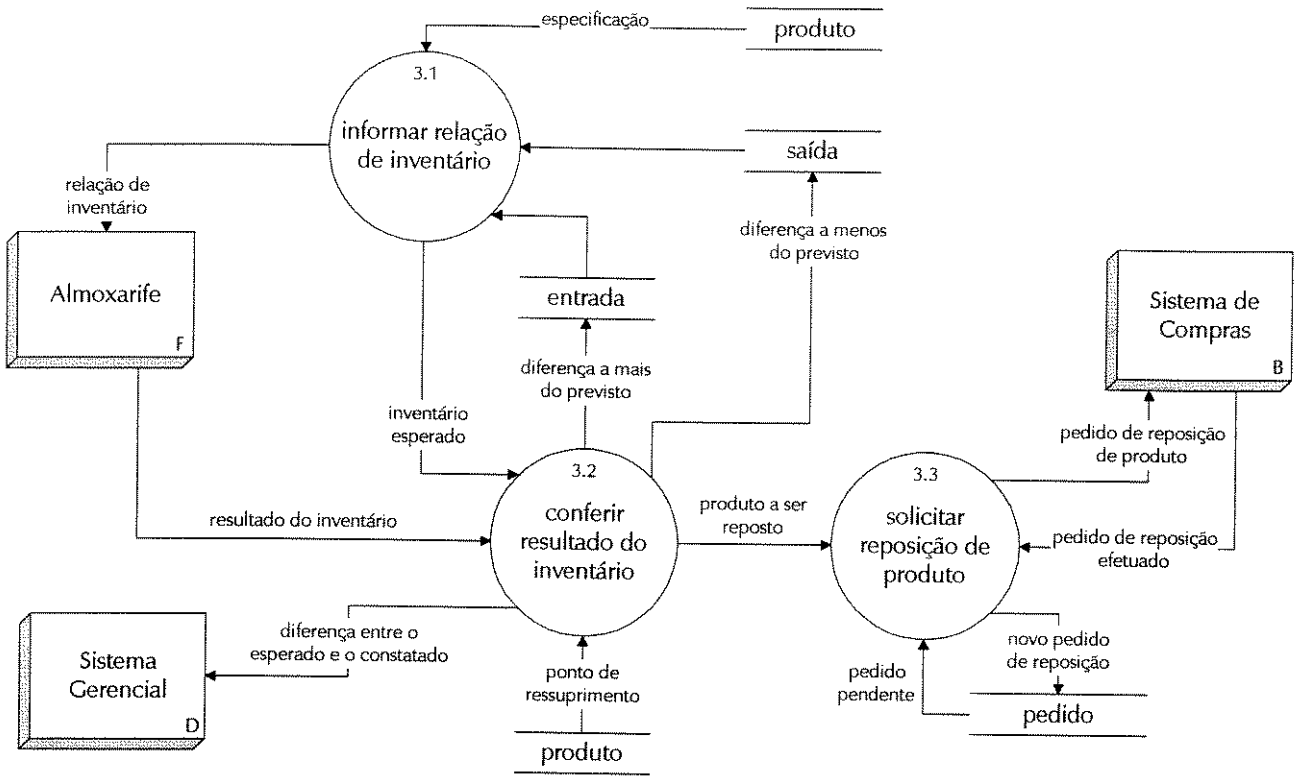
Periodicamente, é necessário fazer um inventário do que realmente existe em estoque.

Tipo de Evento: Temporal

DFD nível 1



DFD NÍVEL 2



FLUXOS DE DADOS

Fluxo	relação de inventário
Origem-Destino	3-F, 3.1-F
Descrição	inventário segundo o armazenado pelo sistema
Composição	{ código do produto + nome do produto + número do lote + quantidade disponível estimada }

Fluxo	resultado de inventário
Origem-Destino	F-3, F-3.2
Descrição	relação de inventário constatado.
Composição	{ código do produto + número do lote + quantidade disponível estimada + [motivo da perda (danificado, estragado)] }

Fluxo	inventário esperado
Origem-Destino	3.1-3.2
Descrição	vide fluxo 3-F
Composição	

Fluxo	diferença entre o esperado e o constatado
Origem-Destino	3.2-D
Descrição	diferença no inventário devido a furto, dano, estrago etc.
Composição	{ código produto + nome do produto + número do lote + quantidade + motivo da perda(furto, dano, estragado) }

Fluxo	diferença a mais do previsto
Origem-Destino	3.2-entrada
Descrição	produtos que sobraram ao final do inventário. Provavelmente, houve troca de código com outro produto na venda.
Composição	{ código do produto + número do lote + quantidade excedente }

Fluxo	diferença a menos do previsto
Origem-Destino	3.2-saída
Descrição	produtos que deveriam estar presentes no inventário. As causas mais comuns do prejuízo são: Furto Danificado. Chuva, vento, queda Estragado. Má conservação, vencimento da validade
Composição	{ código do produto + número do lote + quantidade a menos + motivo da prejuízo }

Fluxo	pedido de reposição efetuado
Origem-Destino	B-3.3
Descrição	vide B-1.3
Composição	

PROCESSOS

Processo	contabilizar produtos em estoque
Número	3
Tipo	(X)Fundamental (X)Custodial
Descrição	inventariar o estoque de modo a atualizar os dados calculados em relação ao real.
Especificação	montar o inventário armazenado informar relação de inventário conferir inventário armazenado com o constatado registrar perdas e sobras solicitar reposição de produtos, se necessário

Processo	informar relação de inventário
Número	3.1
Tipo	(X)Fundamental ( )Custodial
Descrição	montar e informar relação de inventário
Especificação	para cada produto em estoque faça para cada lote de produto faça calcular a quantidade disponível total através da soma de todas as entradas subtraído da soma de todas as saídas informar relação de inventário por produto e lote

Processo	conferir inventário
Número	3.2
Tipo	(X)Fundamental ( )Custodial
Descrição	contabiliza a diferença entre o medido e o esperado, registrando perdas e sobras.
Especificação	para cada produto em estoque faça para cada lote de produto faça registrar prejuízo por dano e estrago medido se inventário medido < calculado então registrar perda se inventário medido > calculado então registrar excedente se total disponível < ponto de ressuprimento, solicitar reposição informar Sistema Gerencial diferença no inventário

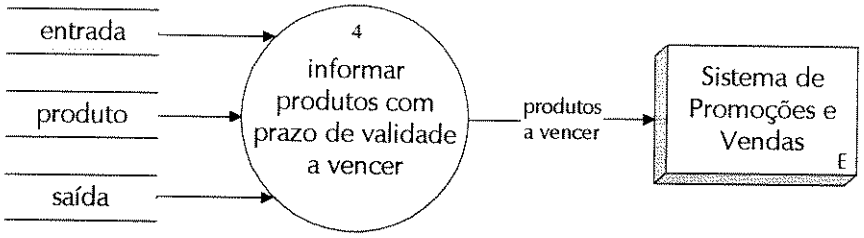
Processo	solicitar reposição de estoque
Número	3.3
Especificação	vide processo 1.3

4. Momento de informar produtos com prazo de validade a vencer

Descrição

Determinado número de dias antes do fim da validade de certos produtos perecíveis, o SCE deve alertar ao Sistema de Vendas a proximidade do fim da validade, de modo que este possa tomar providências para vender o produto, minimizando o possível prejuízo.

Tipo de Evento: `Temporal`



Fluxos de Dados

Fluxo	entrada de produtos
Origem-Destino	entrada-4
Descrição	todas as entradas de produtos armazenadas até o momento.
Composição	{ código do produto + número do lote + data de validade + quantidade }

Fluxo	saída de produtos
Origem-Destino	saída-4
Descrição	todas as saídas de produtos efetuadas até o momento.
Composição	{ código do produto + número do lote + quantidade }

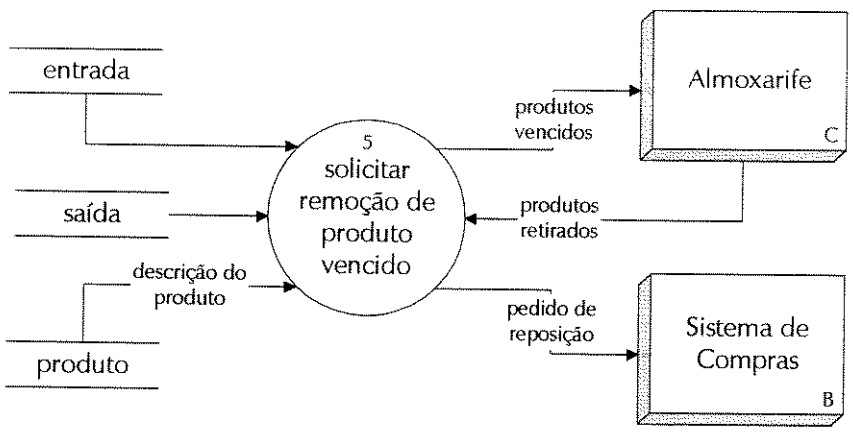
Fluxo	especificação do produto
Origem-Destino	produto-4
Descrição	
Composição	{ nome do produto + número de dias de antecedência do fim da validade a informar + seção + unidade de medida }

Fluxo	produtos a vencer
Origem-Destino	4-E
Descrição	relação de produtos cuja validade estão próxima do fim
Composição	{ código do produto + nome do produto + número do lote + quantidade restante + número de dias para o fim da validade }

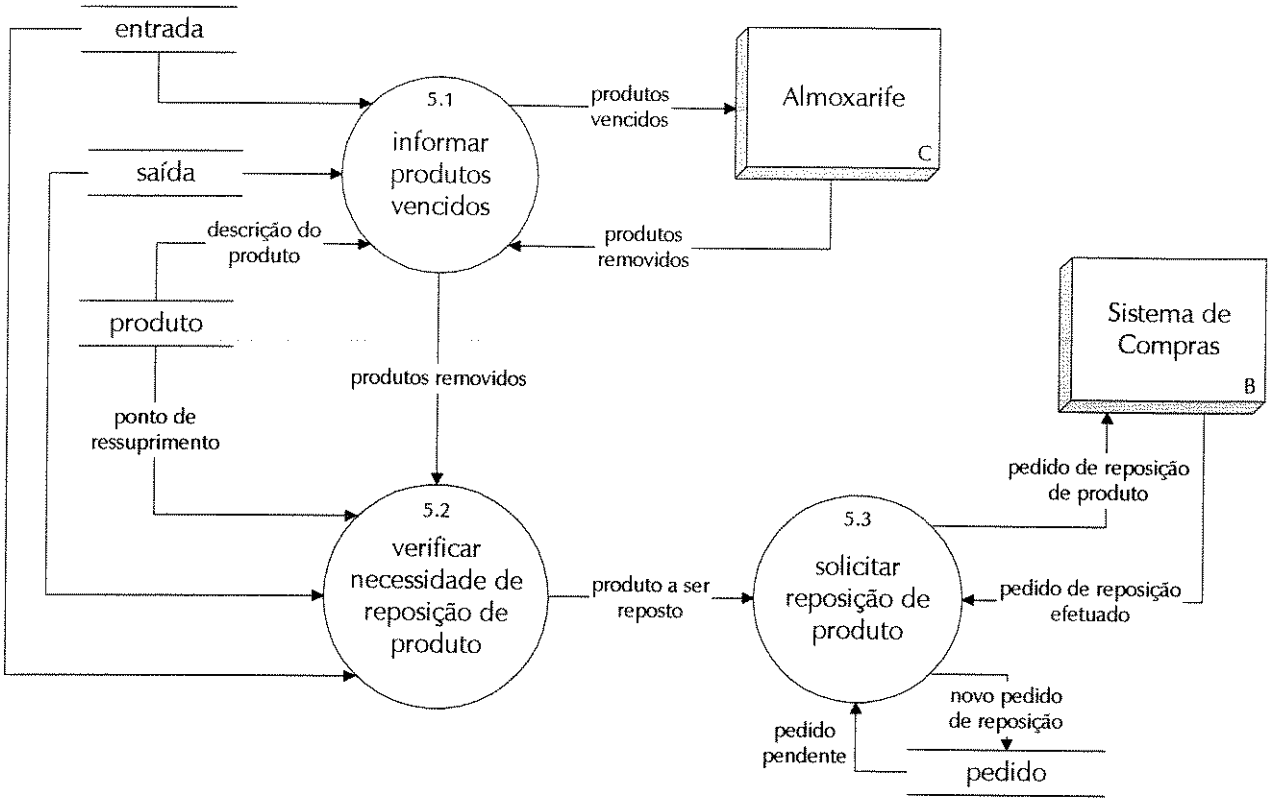
PROCESSOS

Processo	informar produtos com validade a vencer
Número	4
Tipo	(X)Fundamental ( )Custodial
Descrição	informa ao Sistema de Vendas que determinados produtos estão para vencer, de modo a tentar se evitar prejuízos futuros.
Especificação	para cada produto faça para cada lote de produto faça calcular quantidade restante se (quantidade restante > 0) e (número de dias de antecedência < (fim da validade de lote - dia atual)) então relatar produto, lote, quantidade restante e número de dias para fim da validade

5. MOMENTO DE SOLICITAR REMOÇÃO DE PRODUTOS VENCIDOS



DFD Nível 2



Fluxos de Dados

Fluxo	entrada de produtos
Origem-Destino	entrada-5, entrada-5.1
Descrição	vide fluxo entrada-4
Composição	

Fluxo	saída de produtos
Origem-Destino	saída-5, saída-5.1
Descrição	vide fluxo saída-4
Composição	

Fluxo	especificação do produto
Origem-Destino	produto-5, produto-5.1
Descrição	vide fluxo produto-4
Composição	

Fluxo	produtos vencidos
Origem-Destino	5-C, 5.1-C
Descrição	relação de produtos cuja validade se esgotou
Composição	{ código do produto + nome do produto + número do lote + quantidade vencida }

PROCESSOS

Processo	solicitar remoção de produto vencido
Número	5
Tipo	(X) Fundamental (X) Custodial
Descrição	solicita que todos os produtos vencidos em determinada data sejam retirados.
Especificação	para cada produto faça para cada lote de produto faça calcular quantidade restante se (quantidade restante > 0) e (fim da validade de lote < dia atual) solicitar retirada registrar saída de produtos vencidos verificar necessidade de reposição de estoque de produto

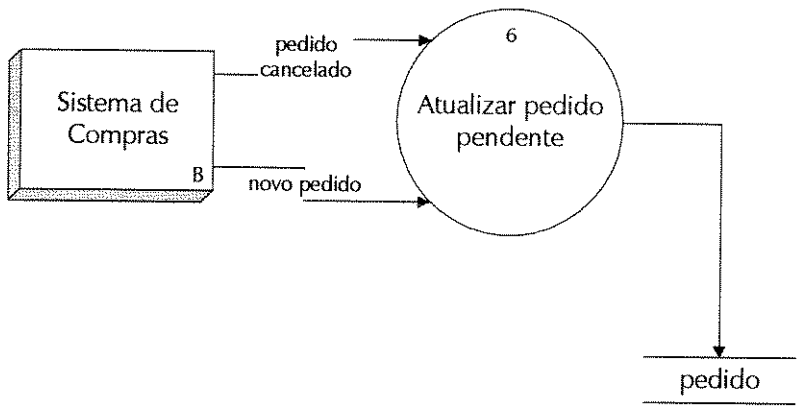
Processo	informar produtos vencidos
Número	5.1
Tipo	(X) Fundamental (X) Custodial
Descrição	solicita que todos os produtos vencidos em determinada data sejam retirados.
Especificação	para cada produto faça para cada lote de produto faça calcular quantidade restante se (quantidade restante > 0) e (fim da validade de lote < dia atual) solicitar retirada registrar saída de produtos vencidos

Processo	verificar necessidade de reposição de estoque
Número	5.2
Descrição	vide processo 1.2
Especificação	

Processo	solicitar reposição de estoque
Número	5.3
Descrição	vide processo 1.3
Especificação	

6. SUPERMERCADO DECIDE COMERCIALIZAR NOVO PRODUTO

DFD NÍVEL 1



FLUXOS DE DADOS

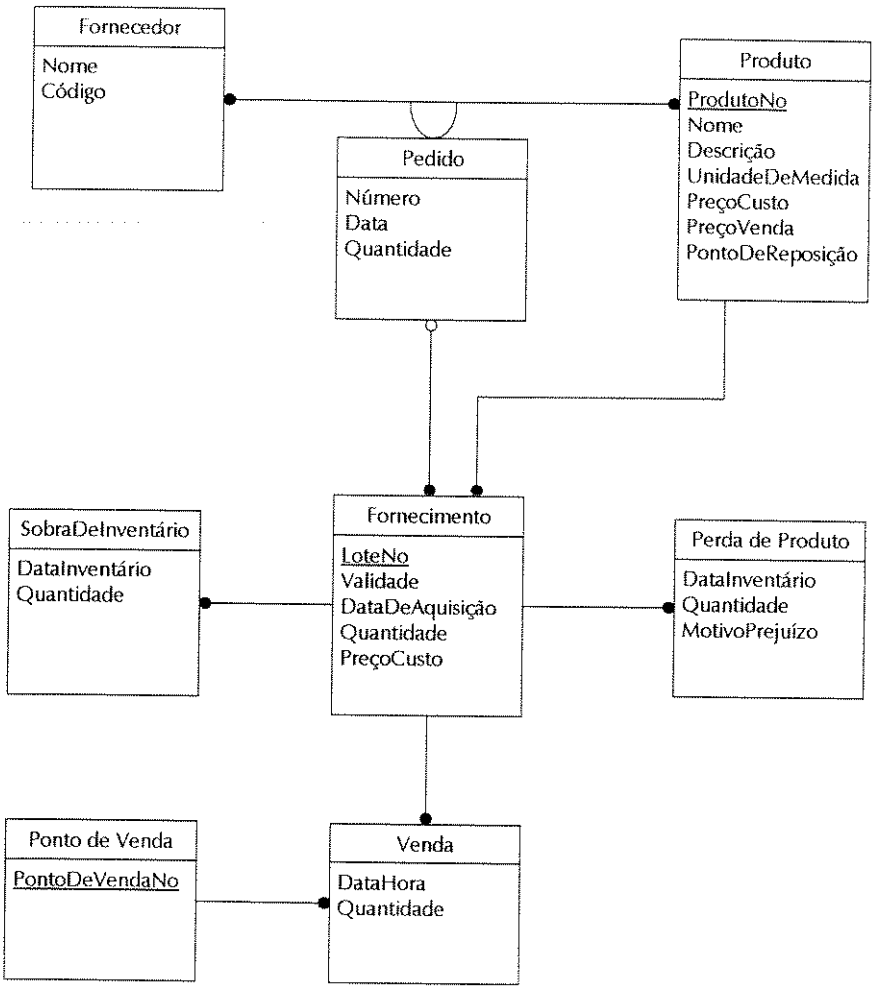
Fluxo	pedido cancelado
Origem-Destino	B-6
Descrição	pedido que foi cancelado.
Composição	{ número do pedido }

Fluxo	novo pedido
Origem-Destino	B-6
Descrição	pedido de fornecimento já feito.
Composição	vide fluxo B-3.3

PROCESSOS

Processo	atualizar pedido pendente
Número	6
Tipo	( )Fundamental (X)Custodial
Descrição	
Especificação	retirar pedido pendente anterior cadastrar novo pedido pendente

Modelo Inicial de Objetos



CARTÕES CRC

Produto CLASSE	SUPERCLASSE
RESPONSABILIDADES	COLABORADORES
1.1 saber dados cadastrais (nome, descrição etc.)	
1.2 saber quantidade disponível em estoque	Entrada(2)
1.3 saber ponto de ressuprimento (valor mínimo em estoque)	
1.4 saber prazo de antecedência de aviso de fim de validade	
2.1 solicitar realização de inventário	Almoxarife(2)
2.2 conferir inventário medido	Entrada(3), Perda(1)
2.3 saber relação de inventário	Entrada(2)
3. verificar necessidade de reposição	Pedido(1)
Representa um tipo de produto contido no estoque.	DESCRIÇÃO

Contratos de `Produto`:

saber especificação de produto

controlar inventário

Entrada CLASSE	SUPERCLASSE
RESPONSABILIDADES	COLABORADORES
1.1 saber quantidade, data da entrada, produto a que se refere	
1.2 saber preço de custo	
1.3 saber data do fim da validade	
1.4 saber quantidade restante	Saída(2)
1.5 registrar entrada de produto no estoque	
2 reincorporar sobras de inventário	
3 informar proximidade do fim da validade	Produto(1), Saída(2), Sistema de Promoções e Vendas(1)
4 solicitar remoção de produtos vencidos	Almoxarife(1), Perda(1)
representa uma entrada de um produto em estoque.	DESCRIÇÃO

Contratos de `Entrada`:

saber especificação de lote recebido;

reincorporar sobra de inventário.

3. informar proximidade do fim da validade;

4 solicitar remoção de produtos vencidos;

Saída (CLASSE)	SUPERCLASSE
RESPONSABILIDADES	COLABORADORES
1.1 registrar saída de produto de um determinado fornecimento	Entrada(1)
1.2 rejeitar saída inválida	Produto(1), Entrega(1)
1.3 saber quantidade, data, lote e produto retirado	
P.1 verificar necessidade de reposição de produto	Produto(3)
DESCRIÇÃO	
representa a saída de certa quantidade de produto referente a um fornecimento específico.	

Contratos de `Saída`:

registrar saída de produtos do estoque;

saber saída de produtos;

P.1 verificar necessidade de reposição de produto;

Pedido (CLASSE)	SUPERCLASSE
RESPONSABILIDADES	COLABORADORES
1 providenciar reposição de produto	Sistema de Compras(1)
2.1 saber pedido pendente de reposição de produto	
2.2 saber data do pedido, quantidade pedida, fornecedor	
3 conferir recebimento de fornecimento de produto	
P.1 dar entrada em fornecimento válido	Entrada(1)
P.2 rejeitar entrada inválida de entrega	
DESCRIÇÃO	
representa um pedido de reposição para um determinado produto que está abaixo do nível mínimo.	

Contratos de `Pedido`:

providenciar preposição de estoque;

saber especificação de pedidos;

conferir recebimento de produto

P.1 dar entrada em fornecimento válido;

P.2 rejeitar entrada inválida de fornecimento;

Venda (CLASSE)	Saída (SUPERCLASSE)
RESPONSABILIDADES	COLABORADORES
1.4 informar preço de produto vendido	Produto(1)
1.5 atender a solicitação de saída de produto por venda	Entrada(1), Produto(1)
representa uma venda de certa quantidade de produto referente a um lote. (DESCRIÇÃO)	

Contratos de `Venda`:

registrar saída de produto do estoque;

Perda (CLASSE)	Saída (SUPERCLASSE)
RESPONSABILIDADES	COLABORADORES
1.4 saber motivo da perda	
1.5 saber prejuízo	
representa a saída por perda devido a furto, estrago, intempéries etc. (DESCRIÇÃO)	

Contratos de `Perda`:

registrar saída de produto do estoque;

Fornecedor CLASSE	SUPERCLASSE
RESPONSABILIDADES 1.1 entregar fornecimento 1.2 saber validade e quantidade em cada lote	COLABORADORES Pedido(3)
	DESCRIÇÃO

Contratos de `Fornecedor`:
fornecer produtos;

Almoxarife CLASSE	SUPERCLASSE
RESPONSABILIDADES 1. retirar produtos vencidos 2.1 fazer conferência do inventário 2.2 informar resultado de conferência do inventário	COLABORADORES Produto(2)
administrador do estoque	DESCRIÇÃO

Contratos de `Pedido`:
providenciar preposição de estoque;
saber especificação de pedidos;
conferir recebimento de produto

Sistema de Compras (CLASSE)	SUPERCLASSE
RESPONSABILIDADES 1.1 providenciar reposição do estoque 1.2 informar ordem de compra efetuada 1.3 saber fornecedor, quantidade pedida e data do pedido	COLABORADORES Pedido(2)
departamento responsável por efetuar compras de produtos para o supermercado.	DESCRIÇÃO

Contratos de `Sistema de Compras`:

providenciar preposição de estoque;

Ponto de Venda (CLASSE)	SUPERCLASSE
RESPONSABILIDADES 1. informar a venda de produtos	COLABORADORES Venda(1)
representa um terminal de venda do supermercado	DESCRIÇÃO

Contratos de `Ponto de Venda`:

informar venda de produtos;

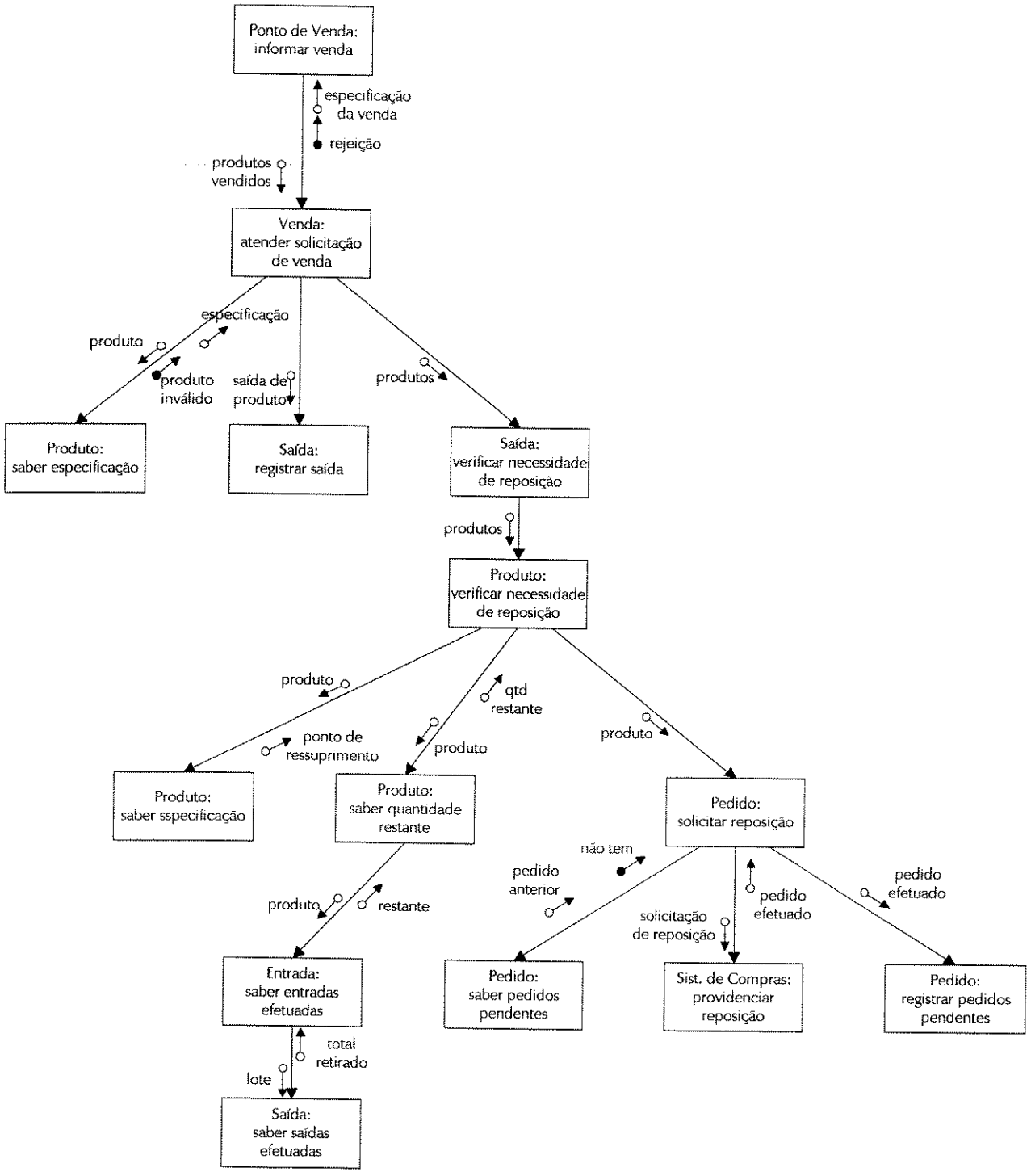
Sistema de Promoções e Vendas (CLASSE)	SUPERCLASSE
RESPONSABILIDADES 1. desovar produtos com prazo de validade a vencer	COLABORADORES
representa um terminal de venda do supermercado	DESCRIÇÃO

Contratos de `Sistema de Promoções e Vendas`:

desovar produtos com prazo de validade a vencer;

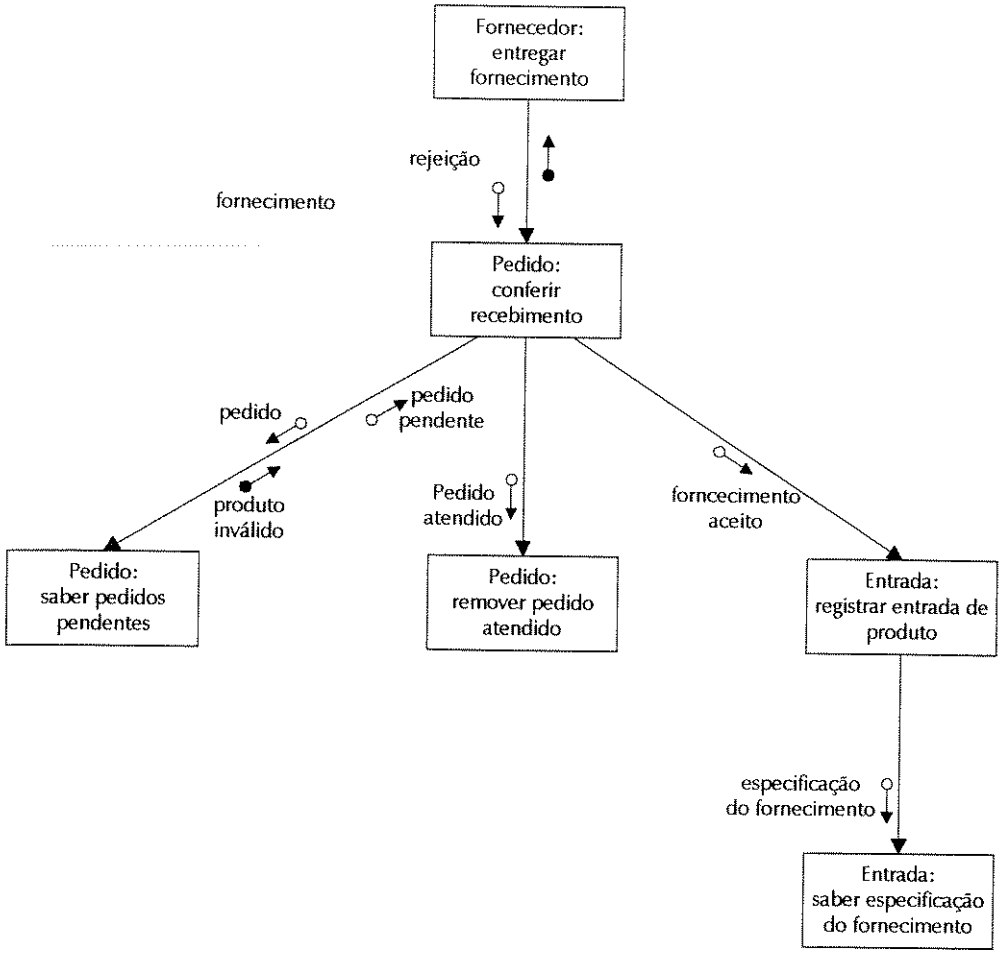
Diagramas de Colaboração entre Objetos

Atividade 1 - Dar Baixa em Produtos Vendidos



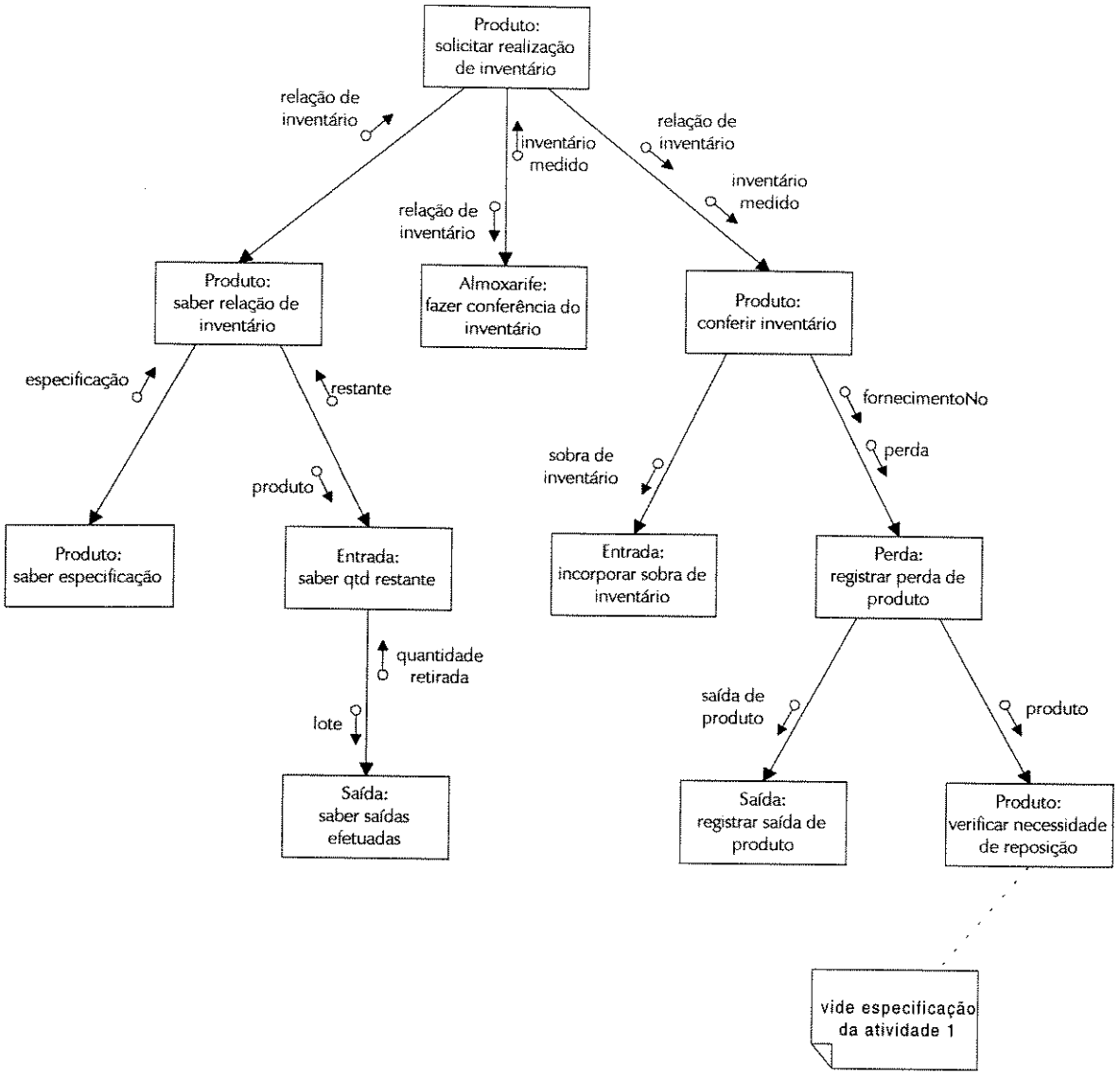
Diagramas de Colaboração entre Objetos

Atividade 2 - Receber Fornecimento



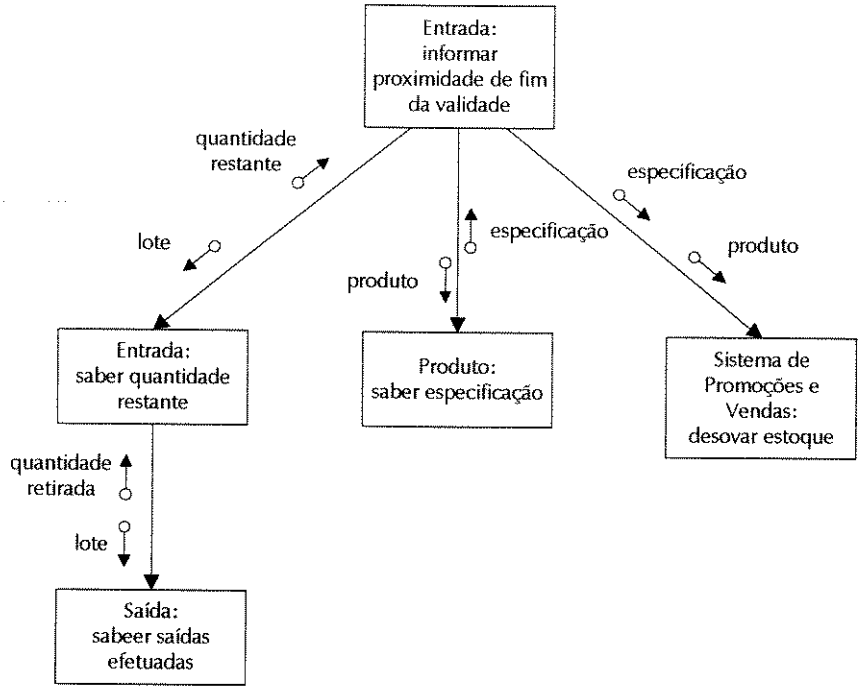
DIAGRAMAS DE COLABORAÇÃO ENTRE OBJETOS

ATIVIDADE 3: CONTABILIZAR PRODUTOS EM ESTOQUE



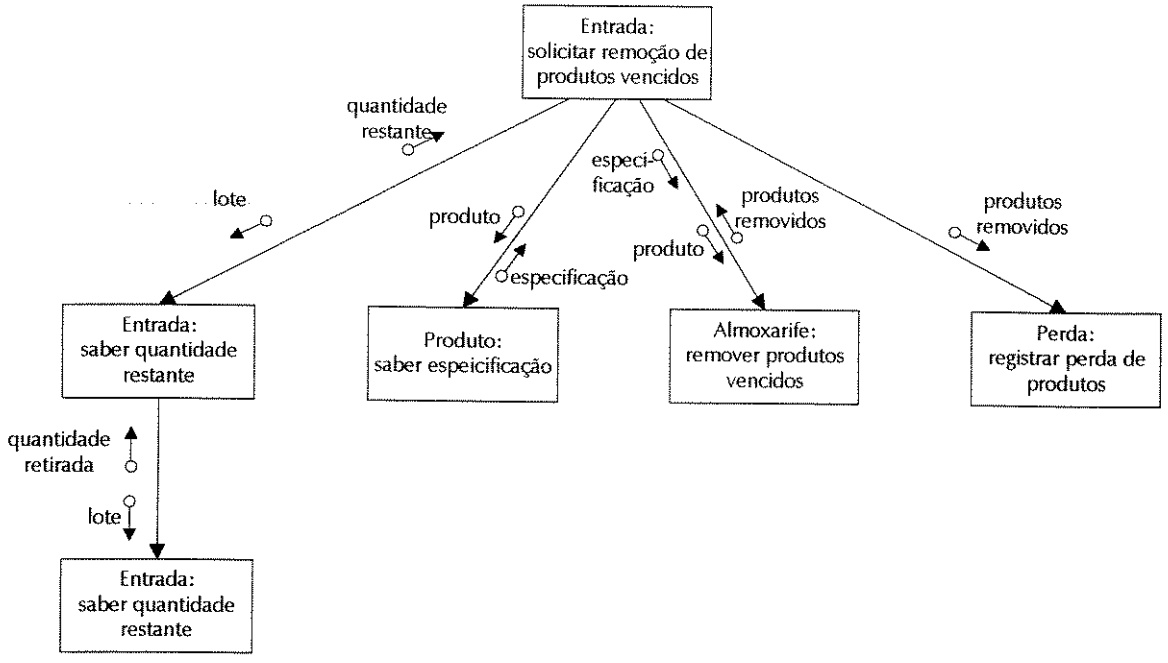
Diagramas de Colaboração entre Objetos

Atividade 4: Informar produtos com validade a vencer



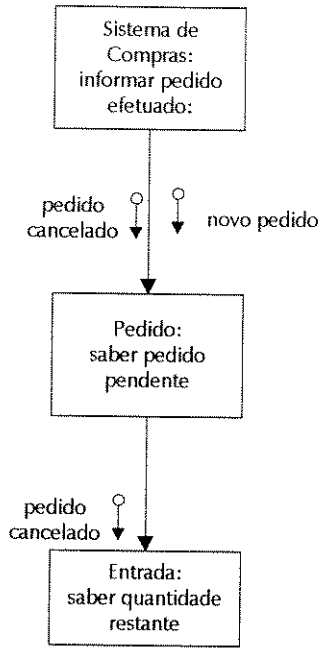
Diagramas de Colaboração entre Objetos

Atividade 5: Solicitar remoção de produto vencido

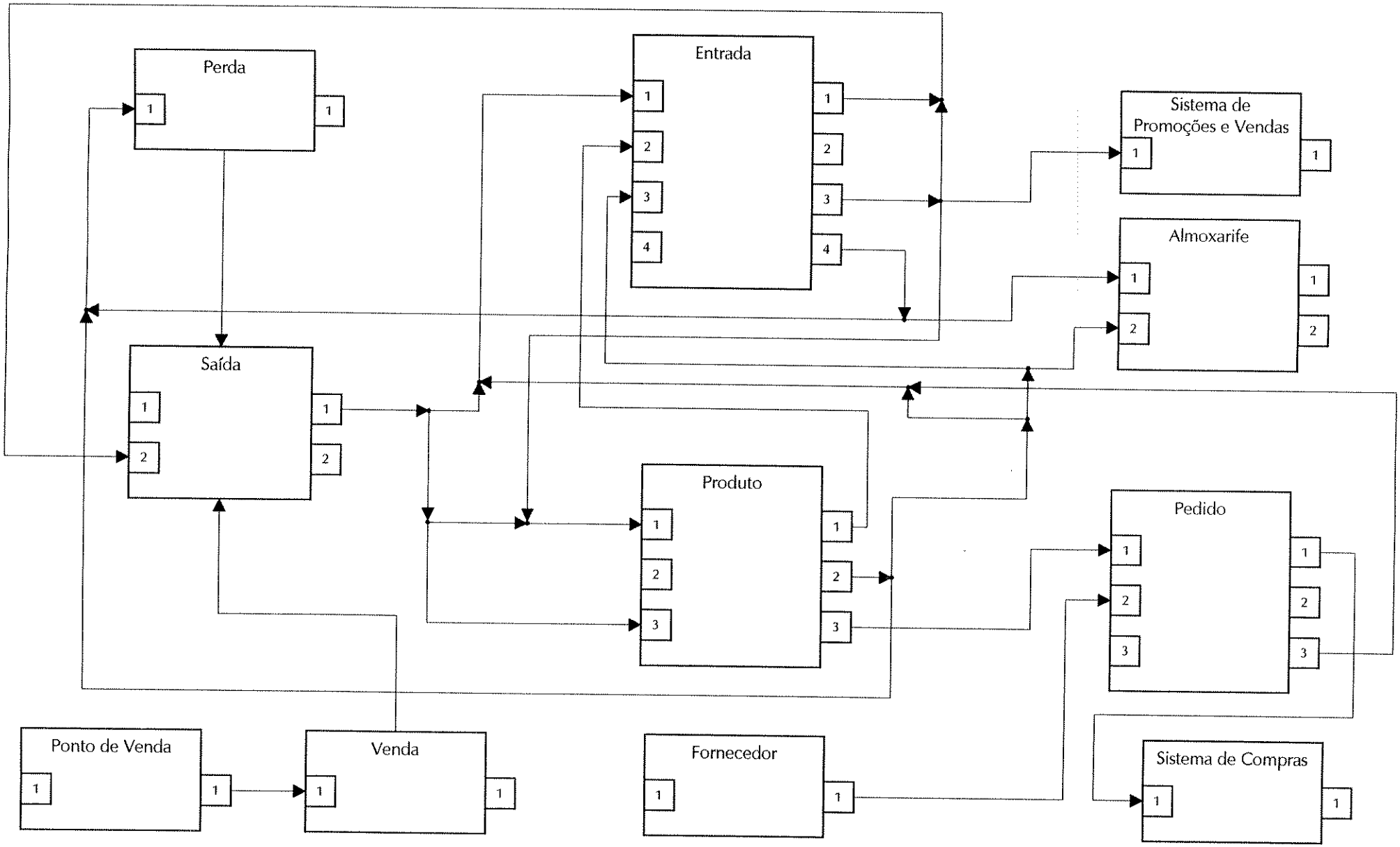


Diagramas de Colaboração entre Objetos

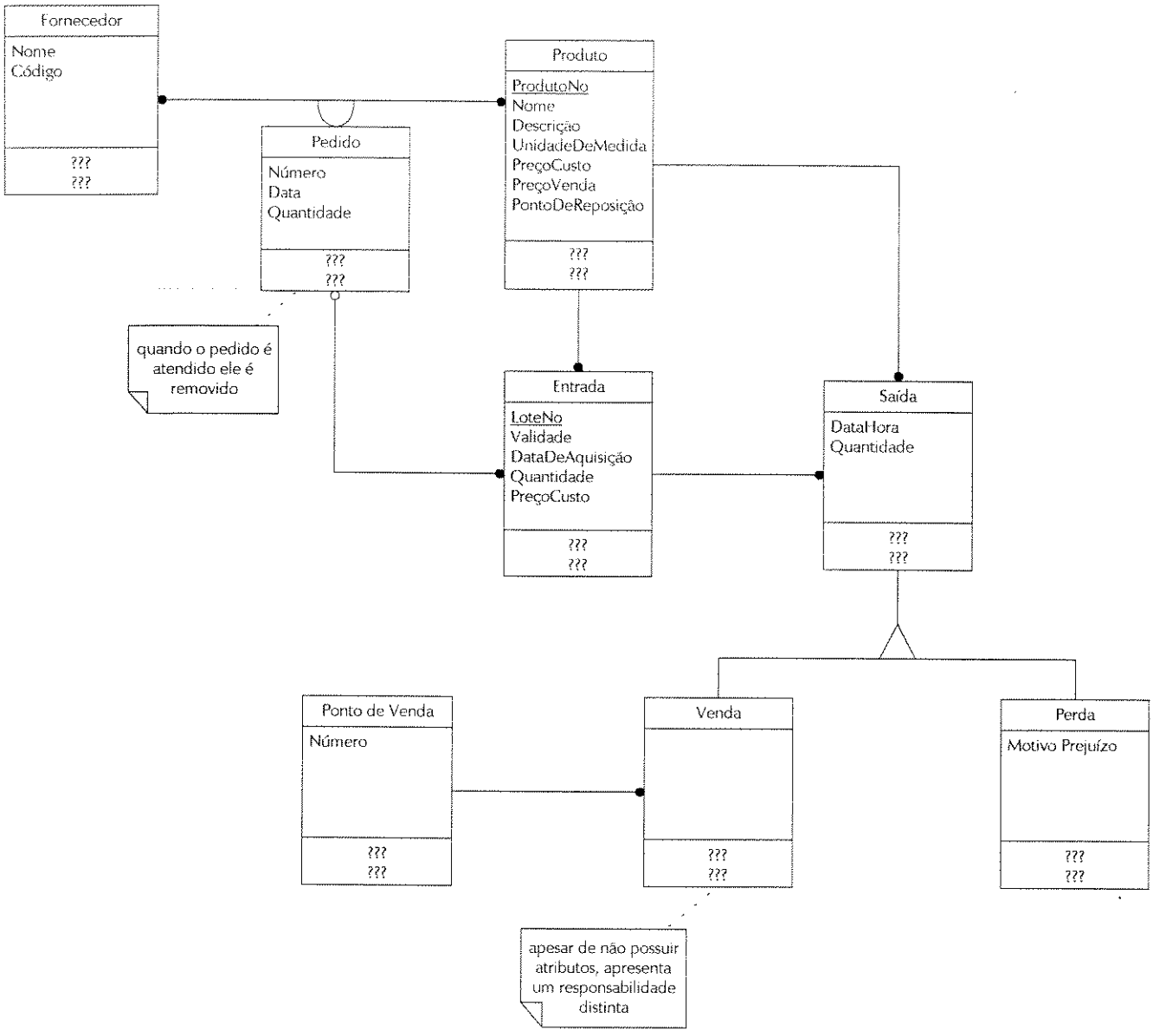
Atividade 6: atualizar Pedido Pendente



Grafo de Colaboração e Contratos



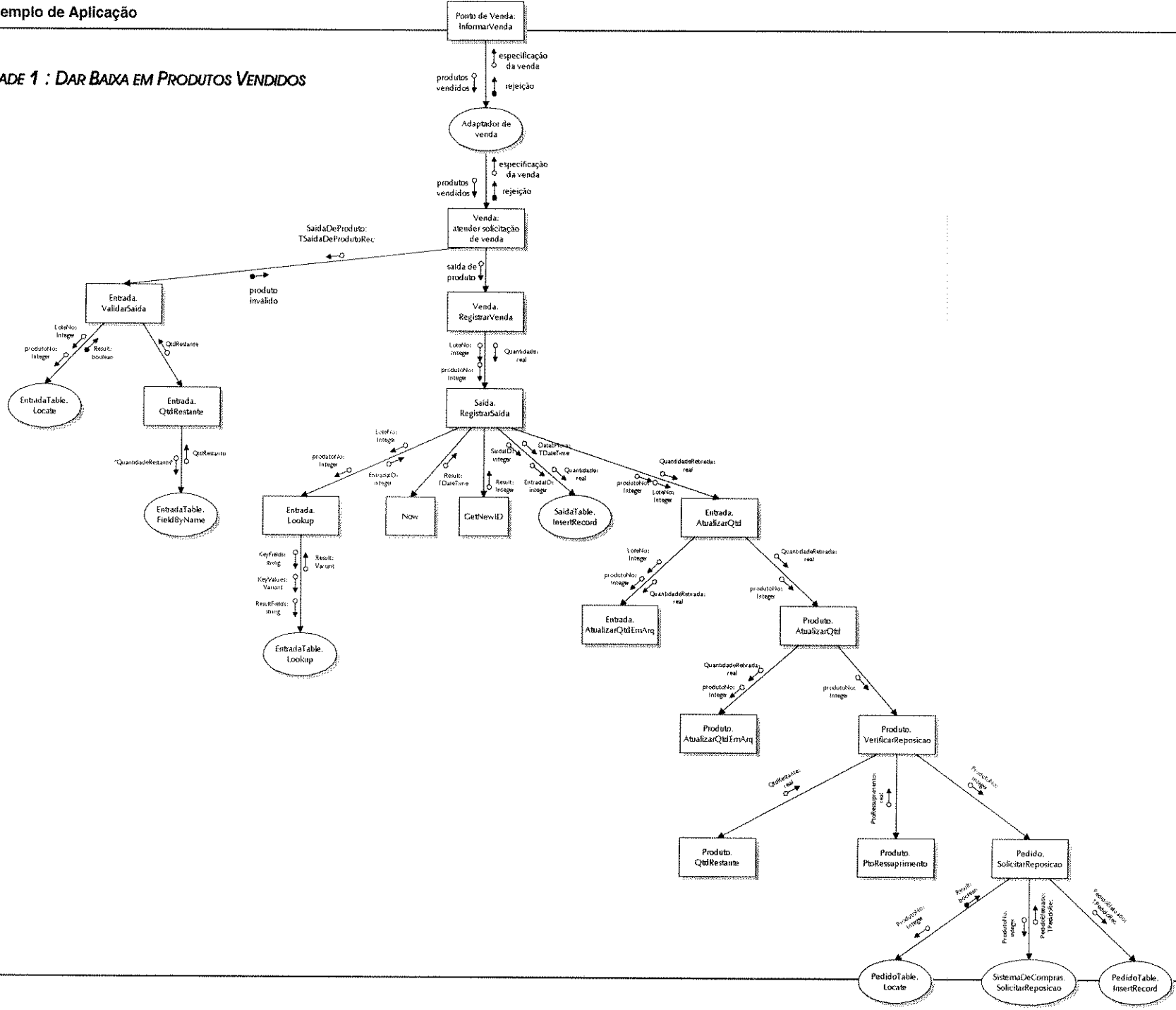
Modelo de Objetos Refinado



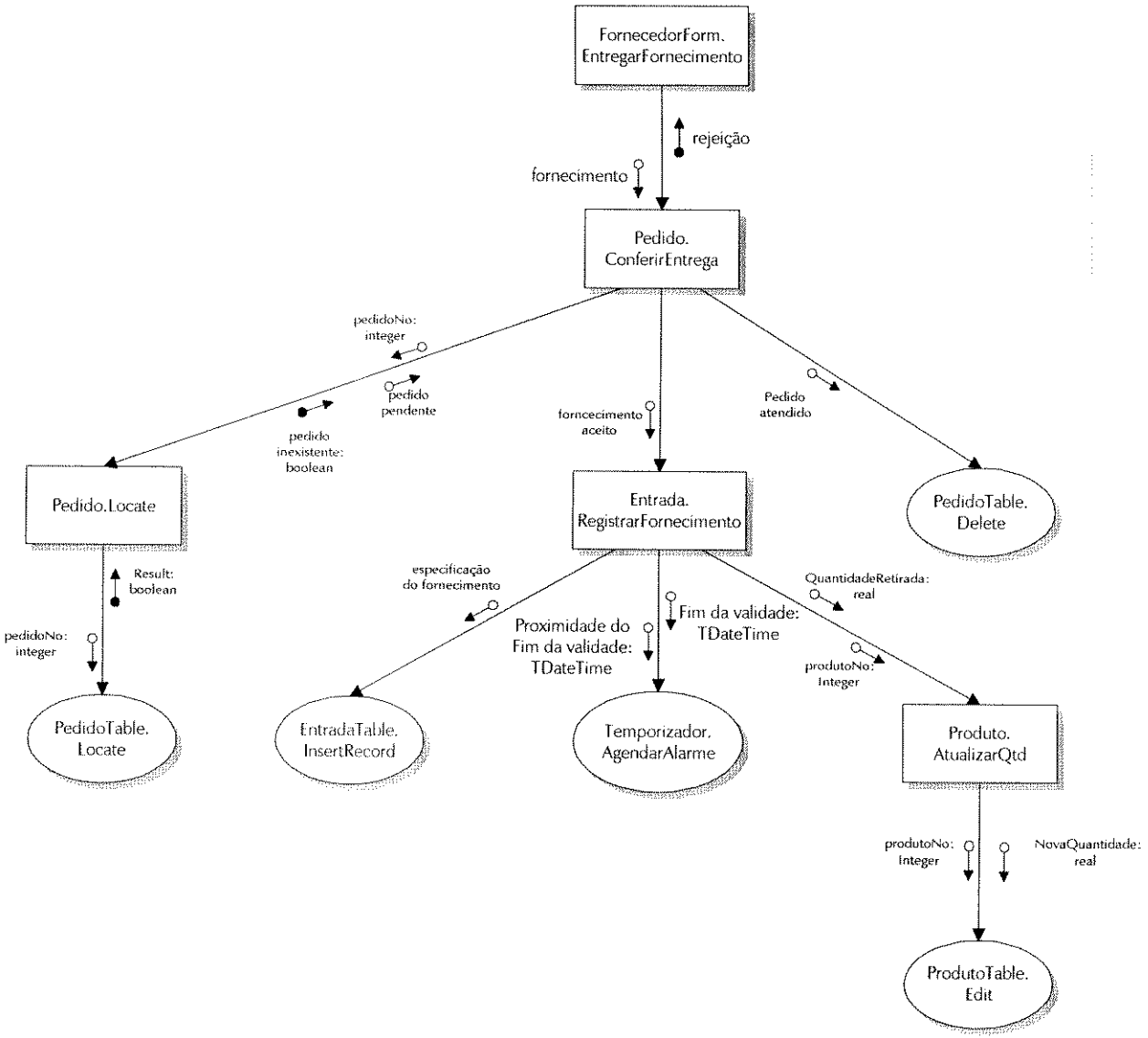
Diagramas de Estrutura

A implementação dos Diagramas de Colaborações entre Objetos (vide documentação da fase anterior) é melhor visualizada através dos Diagramas de Estrutura (DE) resultantes. Nos DEs é possível ver as soluções de implementação adotadas correspondente à cada DCO. O DE contém apenas objetos adaptados e objetos de apoio à implementação.

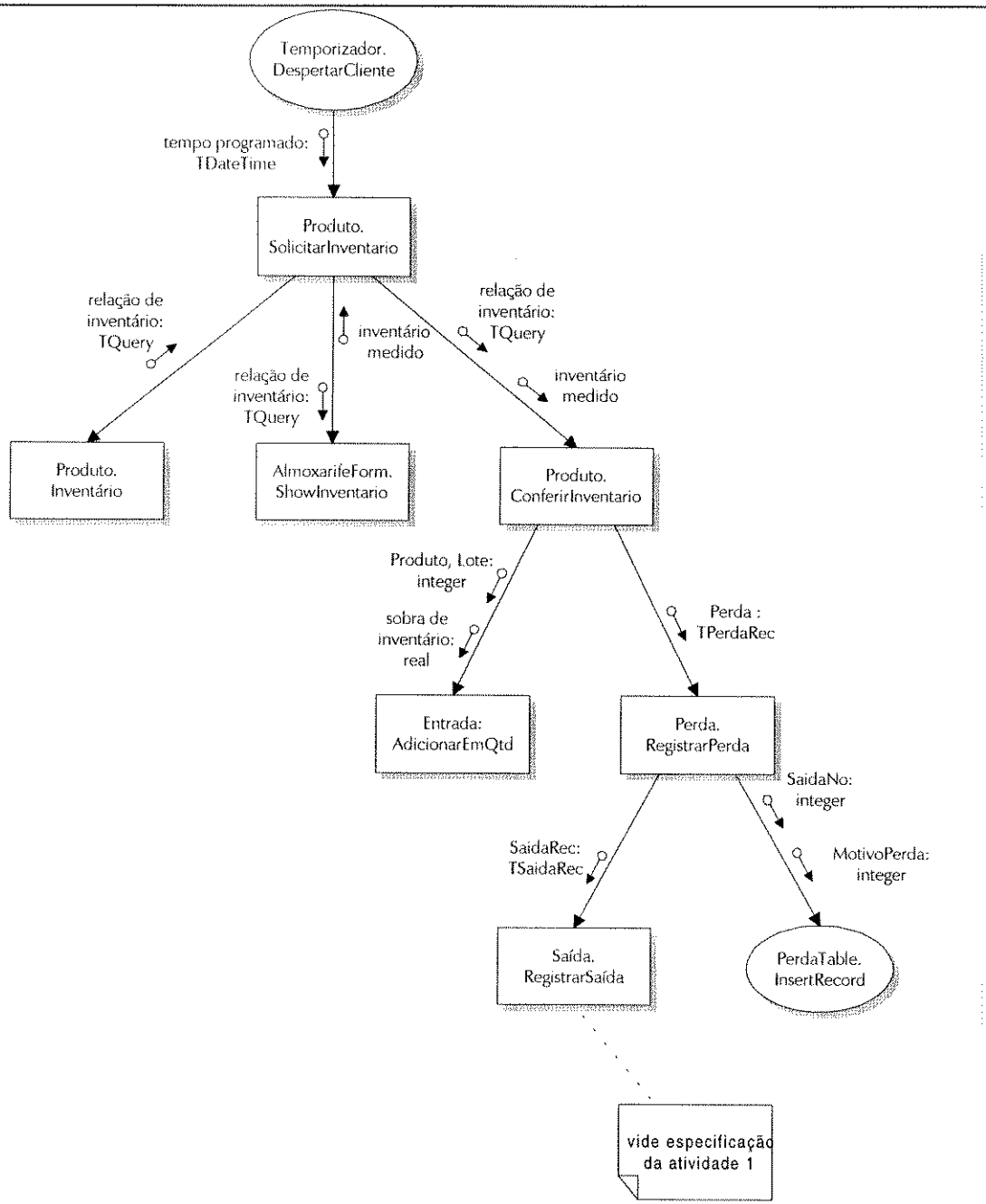
DE PARA ATIVIDADE 1 : DAR BAIXA EM PRODUTOS VENDIDOS



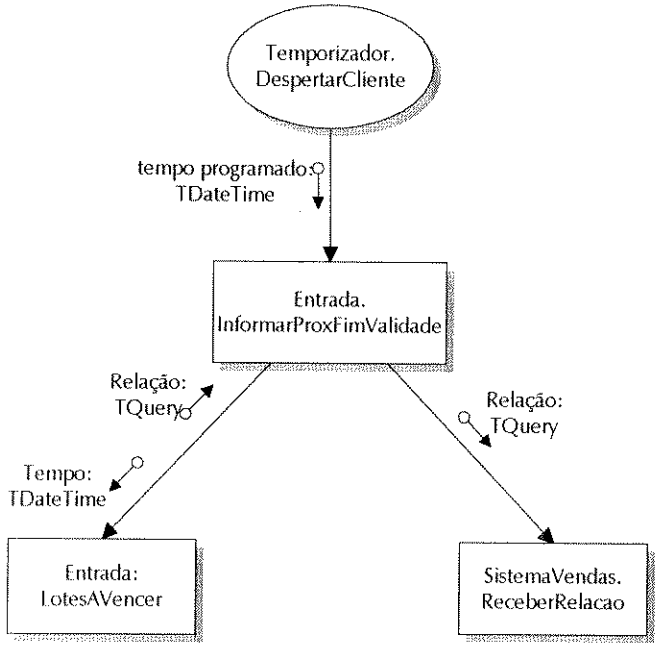
DE PARA ATIVIDADE 2: RECEBER FORNECIMENTO



DE PARA ATIVIDADE 3:



DE PARA ATIVIDADE 4:



DE PARA ATIVIDADE 5: