UNIVERSIDADE FEDERAL DA PARAÍBA
Centro de Ciências e Tecnologia
Coordenação de Pós-graduação em Informática

STL como Fonte Padronizadora de Métodos Orientados a Objetos na Especificação de Software

Ismênia Mangueira Soares Medeiros

Campina Grande, setembro de 1996.

UNIVERSIDADE FEDERAL DA PARAÍBA
Centro de Ciências e Tecnologia
Coordenação de Pós-graduação em Informática

Ismênia Mangueira Soares Medeiros

STL como Fonte Padronizadora de Métodos Orientados a Objetos na Especificação de Software

Dissertação apresentada ao Curso de Mestrado em Informática da Universidade Federal da Paraíba, em cumprimento às normas para obtenção do Grau de Mestre.

José Antão Beltrão Moura

(Orientador)

Álvaro Francisco de Castro Medeiros

(Co-orientador)

Campina Grande, setembro de 1996

M488s Medeiros, Ismenia Mangueira Soares
STL como fonte padronizadora de metodos orientados a objetos na especificacao de software / Ismenia Mangueira Soares Medeiros. - Campina Grande, 1996.
76 f.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Software - 2. Computacao - 3. Dissertacao I. Moura, Jose Antao Beltrao, Dr. II. Medeiros, Alvaro Francisco de Castro, Prof. III. Universidade Federal da Paraiba - Campina Grande (PB) IV. Título

CDU 004.4(043)

STL COMO FONTE PADRONIZADORA DE MÉTODOS ORIENTADOS A OBJETOS NA ESPECIFICAÇÃO DE SOFTWARE

ISMÊNIA MANGUEIRA SOARES MEDEIROS

DISSERTAÇÃO APROVADA EM 27.09.96

PROF. JOSÉ ANTÃO BELTRÃO MOURA, Ph.D
Presidente

PROF. ULRICH SCHIEL, Ph.D
Examinador

PROF. GUSTAVO HENRIQUE MATOS BEZERRA MOTTA, M.Sc
Examinador

CAMPINA GRANDE - PB

Agradecimentos

Agradeço primeiramente a Deus por estar terminando este trabalho.

Depois, agradeço, sinceramente, aos meus pais, que sempre me incentivaram e me mostraram os melhores caminhos a percorrer. A Álvaro, pelo companheirismo, pela orientação, compreensão e amor. À minha filha Anna Carolina, por ter entendido as minhas ausências e às minhas irmãs pelo apoio, carinho e amizade dispensados.

Quero dizer "muito obrigada" a Antão por tão bem me orientar e conduzir neste trabalho.

Obrigada ao professor Ulrich por ter atendido sempre que eu precisei.

Aos professores do DSC pela disposição em compartilhar seus conhecimentos, em especial à Bibi pelo apoio, carinho e amizade. À Josenilta, as meninas da Coordenadoria de Extensão, Jô da cantina e a todos que fazem o DSC.

Segue um carinho especial, em forma de agradecimento a todos os meus amigos de estudo e brincadeiras: Norminha, Micha, Nícia, Bel, Edberto, Vitor, Gilson, Carlos, Ricardo Mario Ernesto, Paty, Dário, Tárcio, Fernanda, Robson, enfim a todos.

À Alberto e a todo pessoal de suporte do LabCom. À Aninha e Vera pelo carinho, amizade e presteza com que sempre me atenderam na COPIN. Ao pessoal da Miniblio pela eficiência, atenção e amizade que nos dispensa: Zeneide, Manuela e Arnaldo.

Agradeço também aos meus colegas da ETFAL, em especial aos professores Alberto e Mário César. A UFPB/COPIN e CAPES, pelo uso dos equipamentos e suporte financeiro. Aos meus colegas do DI da UFPB, por me apoiarem nesta fase final de trabalho. A Aloysius e Islene pelo apoio técnico dispensado.

Lista de Figuras

Figura 1.1 - Organização geral das ferramentas CASE, 3

Figura 1.2 - Ciclo-de-vida do desenvolvimento com CASE, 5

Figura 2.1 - O Micro Processo de Desenvolvimento, 18

Figura 2.2 - Formas de relacionamentos binários, 23

Figura 3.1 - Exemplos de mecanismos de transferência, 42

Figura 3.2 - Diagrama P1-P2, 49

Figura 3.3 - Representação de atributos, 50

Figura A.1 - Exemplo do controle do portão, 74

Figura B.1 - Definição dos conceitos orientados a objetos em STL, 76

Lista de Quadros

Quadro 3.1 - Pacote STL, 46

Quadro 3.2 - Sentença STL, 46

Quadro 3.3 - Cláusula STL, 48

Lista de Tabelas

Tabela 2.1 - Características herdadas de acordo com os métodos, 33

Tabela 2.2 - Classificação dos conceitos OMT, 34

Tabela 2.3 - Classificação dos conceitos CYOOA, 35

Tabela 2.4 - Classificação dos conceitos Booch, 36

Tabela 2.5 - Classificação dos conceitos OOSA, 37

Tabela 2.6 - Classificação dos conceitos W^3OOD, 38

Tabela 2.7 - Representação dos conceitos, segundo o aspecto estrutural, comuns aos métodos, 38

Tabela 2.8 - Representação dos conceitos segundo o aspecto comportamental, em cada método 39

Tabela 3.1 - Símbolos utilizados em STL, 45

Tabela 3.2 - Nomes dos conceitos STL, 47

Tabela 3.3 - Os conceitos STL em função das categorias, 53

Tabela 4.1 - Representação dos conceitos STL em função das categorias e dos conceitos dos métodos, 55

Tabela 4.2 - Representação dos relacionamentos, 58

Tabela 4.3 - Conceitos dos métodos em função dos conceitos STL, 59

Tabela 4.4 - As Relações STL em função dos conceitos dos métodos estudados, 63

Sumário

1. Introdução

1.1 - As Ferramentas CASE

1.2 - Compatibilidade entre Ferramentas CASE

1.3 - Linguagem de Transferência de Semântica (STL)

1.4 - Objetivos

1.5 - Contribuições do Trabalho

1.6 - Organização da Dissertação

2. Uma Revisão dos Médodos de Especificação de Software Orientados a Objetos

2.1 - Introdução

2.2 - Método Rumbaugh (OMT)

2.2.1 - Análise

2.2.2 - Projeto do Sistema

2.2.3 - Projeto dos Objetos

2.3 - Método Coad/Yordon (CYOOA)

2.3.1 - Identificação de Classes e Objetos

2.3.2 - Identificação de Assuntos

2.3.3 - Identificação de Estruturas

2.3.4 - Definição de Atributos

2.3.5 - Definição de Serviços

2.4 - Método Booch

2.4.1 - Identificar Classes e Objetos em um Dado Nível de Abstração

2.4.2 - Identificar as Semânticas e Relacionamentos das Classes e Objetos

2.4.3 - Implementar Classes e Objetos

Sumário

2.5 - Método Shlaer e Mellor (OOSA)

2.5.1 - Identificar Classes e Objetos no Domínio do Problema

2.5.2 - Identificar os Atributos dos Objetos

2.5.3 - Identificar os Relacionamentos entre Objetos

2.5.4 - Identificar Subtipos/Supertipos e Objetos Associativos

2.5.5 - Identificar Máquinas de Estados para o Ciclo de Vida do Objeto e Relacionamentos com Ciclo de Vida.

2.5.6 - Identificar Sistemas Dinâmicos através do Modelo de Comunicação do Objeto.

2.5.7 - Identificar Modelos de Processos para Ações que Ocorrem nos Estados.

2.6 - Método Wirfs-Brock

2.6.1 - Identificação de Classes

2.6.2 - Responsabilidades

2.6.3 - Colaborações

2.6.4 - Contratos

2.6.5 - Especificação de Protocolos

2.7. Identificação dos Principais Conceitos e Características dos Métodos Estudados

2.7.1 - Identificação dos Principais Conceitos do Método OMT

2.7.2 - Identificação dos Principais Conceitos do Método CYOOA

2.7.3 - Identificação dos Principais Conceitos do Método BOOCH

2.7.4 - Identificação dos Principais Conceitos do Método OOSA

2.7.5 - Identificação dos Principais Conceitos do Método W^3OOD

2.7.6 - Características dos Métodos Estudados

Sumário

2.7.7 - Características Herdadas de acordo com os Métodos

2.8 - Classificação dos Métodos segundo os Aspectos Estrutural e Comportamental.

2.9 - Conclusão

3. Linguagem de Transferência de Semântica

3.1 - Modelo para Transferência de Informações entre Ferramentas

3.2 - A Linguagem STL

3.3 - Conceitos em STL

4. Representação dos Principais Conceitos dos Métodos através de STL

4.1 - Identificação dos Conceitos STL

4.2 - STL Suporta os Conceitos dos Métodos Estudados?

4.3 - Adequação STL para Representar Conceitos

5. Sumário, Conclusão e Sugestões

5.1 - Sumário

5.2 - Conclusão

5.3 - Sugestões para Trabalhos Futuros

Apêndice A

Apêndice B

Referências Bibliográficas

RESUMO

Analisar como os principais conceitos dos métodos de especificação de software orientados a objetos de Rumbaugh, Coad/Yordon, Booch, Shlaer e Mellor, e Wirfs-Brock, poderiam ser descritos em termos de uma linguagem de transferência de semântica é a motivação central desta pesquisa.

Foram escolhidos cinco métodos de especificação de software de acordo com a sua relevância acadêmica e bibliografia disponível. Estes métodos tiveram seus conceitos identificados e classificados de acordo com os aspectos estrutural e comportamental enfatizando assim suas capacidades de modelagem estática e dinâmica. Estes conceitos foram mapeados em função dos conceitos da Linguagem de Transferência de Semântica - STL, que foi estudada neste trabalho não apenas sob a visão para comunicação entre ferramentas CASE, mas também como fonte padronizadora dos conceitos utilizados pelos cinco métodos de especificação de software orientados a objetos estudados.

A necessidade de padronização na comunicação entre ferramentas CASE é premente e objeto de estudo da comissão IEEE que estuda esta proposta de padronização sob o número 1175 que trata da transferência de informação semântica entre módulos ou componentes. Espera-se com este trabalho oferecer uma análise de como STL representa os conceitos dos métodos de especificação de software estudados.

ABSTRACT

The central motivation of this research is to analyze how the main concepts of object-oriented software specification methods from Rumbaugh, Coad/Yordon, Booch, Shlaer e Mellor, e Wirfs-Brock, could be mapped into Concepts of Semantic Transfer Language.

The academic relevance and the available bibliographic resources were the reason for the choice of these five object-oriented software specification methods. The method's concepts were identified and classified according to the structural and behavioral aspects emphasizing their static and dynamic modeling capabilities. These concepts were mapped into concepts of the Semantic Transfer Language - STL. In this work, STL was considered as a standardizing resourse for object-oriented software specification concepts instead of a mechanism to provide communication between CASE tools.

The market need a communication standard for CASE tools. The IEEE Standard 1175 committee is working toward a semantic transfer language for semantic transfer information between modules or components. The main contribution of this research is to analyze how STL concepts support the five concepts of object-oriented software specification methods.

1. Introdução

Este trabalho está inserido no contexto da orientação a objeto e discute como os conceitos dos métodos de especificação de software orientados a objetos podem ser representados, usando os conceitos de um meta-modelo que especifica a Linguagem de Transferência de Semântica - STL [IEEE 92].

Serão estudados cinco métodos de especificação de software orientados a objetos: OMT de Rumbaugh [RUMB 91], CYOOA de Coad/Yordon[COAD 91], Booch [BOOC 94], OOSA de Shlaer e Mellor [SHLA 88][SHLA 91], e Wirfs-Brock [WIRF 90]. Seus principais conceitos serão identificados e classificados segundo os aspectos estrutural e comportamental, para que possa ser melhor observada a capacidade de modelagem estática e dinâmica dos métodos.

STL será estudada sob o ponto de vista de uma fonte padronizadora para os conceitos identificados nos métodos considerados. Desta forma, STL terá seus principais conceitos classificados de acordo com a estruturação da informação e comportamento da informação, possibilitando, assim, o mapeamento empírico de seus conceitos com os conceitos dos métodos estudados.

Apesar de existirem vários métodos de especificação de software[RUMB 95], a escolha dos métodos foi pautada pela relevância acadêmica, pela contribuição à indústria e pela bibliografia disponível. Estes métodos, na maioria das vezes, utilizam palavras diversas para representar os conceitos da orientação a objeto. Esta pesquisa estuda como relacionar nomes diferentes, em métodos diferentes, para conceitos comuns mapeando-os em STL.

Nos capítulos seguintes serão identificados os principais conceitos de cada método e analisados segundo o padrão desenvolvido pela IEEE, Comissão 1175 para comunicação entre ferramentas CASE. Aqui vale resaltar a importância destas ferramentas para o incremento de produtividade no processo de desenvolvimento de software.

Nas seções seguintes, serão apresentadas uma visão geral das ferramentas CASE, o problema da compatibilidade entre ferramentas e uma introdução à Linguagem de Transferência

de Semântica. Serão também apresentados os objetivos e contribuições do trabalho, assim como a organização da dissertação.

1.1 - As Ferramentas CASE

As ferramentas CASE sempre ocuparam um espaço importante na história da Engenharia de Software, seja pelo seu esforço de automatização de algumas fases do processo de desenvolvimento de software, através da criação de ambientes de desenvolvimento, seja pela implementação de artefatos de software contendo novos paradigmas - como é o caso de vários produtos disponíveis no mercado que embutem os métodos de especificação de software orientado a objetos.

Na realidade, para fazer frente à crise de software, onde cada vez mais o hardware é lançado no mercado e só algum tempo depois o software se torna disponível[SOMM 96], não só os aspectos de automatização de processos existentes devem ser endereçados. É necessário um esforço para o uso efetivo de padronização e novas filosofias [MEDE 94] para o desenvolvimento de software. Neste contexto, o importante no desenvolvimento de produtos ou ferramentas CASE é que, nos anos noventa, eles terão que suportar o desenvolvedor na modelagem de várias dimensões dos métodos de especificação de software orientados a objetos permitindo comparação e reaproveitamento de especificações[MEDE 96].

As primeiras versões das ferramentas CASE tinham limitações relativas à integração de módulos. Já as novas ferramentas usam uma padronização a nível de organização. Conforme pode-se ver na Figura 1.1, onde um repositório é o **núcleo** da ferramenta CASE sendo responsável por diversas atividades: gerenciamento e controle do projeto; reusabilidade de software; geração de código, integração de diferentes conjuntos de ferramentas CASE; compartilhamento de informação sobre o sistema; integridade entre todos os componentes do modelo.

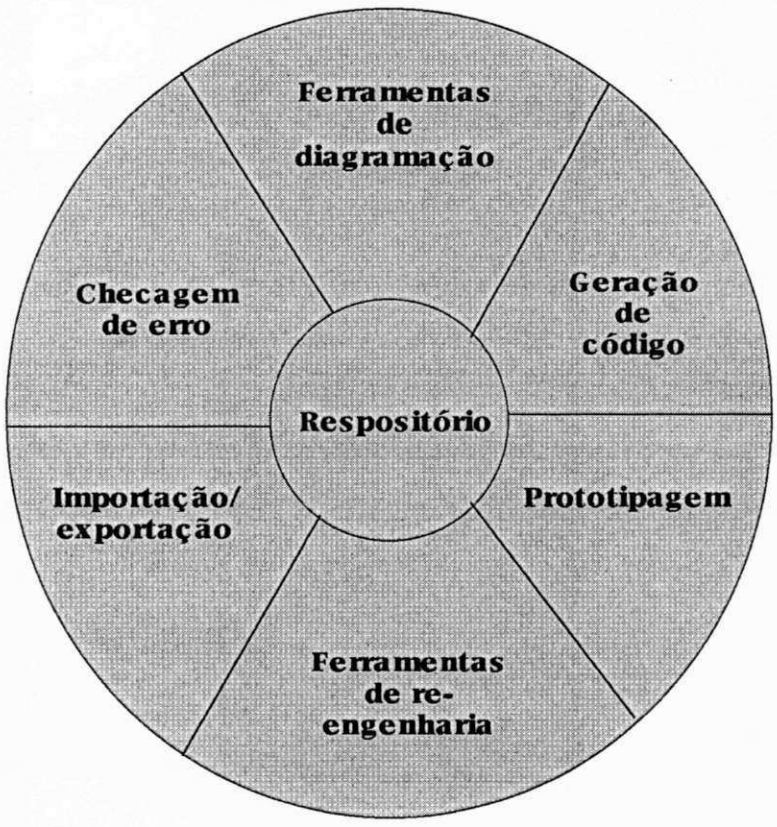

Figura 1.1:
Organização Geral das Ferramentas CASE

Para centralizar tantas atividades, o repositório precisa manter diversas informações sobre o modelo. Tomando-se como base a descrição de objetos por exemplo, as seguintes informações deveriam estar presentes no repositório: identificação (nome único); definição (significado); tipo, sinônimo; composição (sub-componentes); parentesco - objeto no qual este objeto é componente; regras para usar este objeto; quem/quando criar o objeto; status, versão. Entretanto, manter a integridade de todas estas informações exige técnicas das mais avançadas em banco de dados. Muitas vezes, as informações relativas à organização ou estrutura de dados dos repositórios são guardadas como segredo comercial, o que dificulta a comunicação entre ferramentas CASE.

Tempo de criação de uma especificação

Autores como Yordon [YOUR 90] e Andrew Topper[TOPP 94] falam no esforço para a construção da primeira versão de uma especificação orientada a objetos usando uma ferramenta CASE. Eles dizem que leva quase o mesmo tempo para criar um diagrama usando uma ferramenta CASE quanto trabalhando manualmente. A diferença, ainda segundo Yordon e Topper, é que as correções podem ser feitas com mais rapidez usando as ferramentas CASE, uma vez que elas automatizam a edição gráfica do modelo e ainda fazem testes de consistência e "completeza". Por fim, a produtividade obtida usando-se ferramentas CASE vai depender das habilidades do desenvolvedor com os métodos utilizados e um domínio completo da ferramenta nas várias etapas do processo de desenvolvimento de software[GANE 90][MART 95].

Ciclo-de-vida do processo de desenvolvimento

Conforme podemos ver na Figura 1.2, o ciclo-de-vida do processo de desenvolvimento de software usando ferramentas CASE pode ser analisado em duas fases distintas. Na primeira, o computador auxilia o desenvolvedor através da automatização de algumas atividades como o fornecimento de todo um ferramental gráfico para o desenho dos diagramas que compõem o modelo. Já na segunda fase, o computador é o responsável pela geração de informações e resultados. Não há intervenção do desenvolvedor. As atividades são executadas com base nas informações extraídas do repositório. Um exemplo é a geração de código que pode ser feita para uma determinada linguagem sem a intervenção do desenvolvedor, apenas com as informações do repositório.

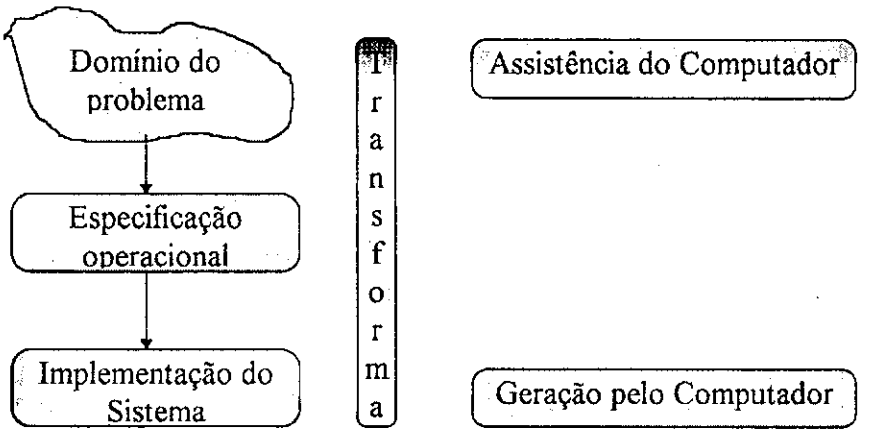

Figura 1.2: Ciclo-de-vida do Desenvolvimento com CASE

Produtos CASE disponíveis no mercado

A maioria dos produtos tidos como ferramentas CASE apresentam-se contendo um editor gráfico com capacidade para trabalhar com vários métodos já embutidos (por exemplo, os métodos [BOOC 94], [COAD 91], [RUMB91], e [WIRF 90]). Isto não quer dizer que o desenvolvedor não possa trabalhar com novos métodos. As ferramentas ObjectMaker[1] e SystemArchitet[2] utilizam o conceito de *metamodelo*. Ou seja, ao desenvolvedor é permitido a criação de novos modelos ou a personalização dos já existentes.

Segundo Andrew Topper *et. al.* [TOPP 94], existem mais de trezentos produtos CASE no mercado e que eles podem ser classificados segundo os critérios: *dados*, *controle* e *funcionalidade*. Entretanto, há deficiência no suporte do comportamento dinâmico dos sistemas e na comunicação entre módulos ou componentes distribuídos por diferentes fabricantes.

[1]ObjectMaker é uma ferramenta CASE produzida por Mark V System Limited - CA.
[2]System Achitet é uma ferramenta CASE produzida por Popkin Software e System Company. - NY

1.2 - Compatibilidade entre Ferramentas CASE

Segundo Booch e Rumbaugh [BOOC 95a][RUMB 95], o número atual de métodos de especificação de software orientado a objetos é superior a 70 e com tendências a crescer ainda mais. Surge, com este crescimento, uma grande preocupação em como as ferramentas irão se comunicar entre si, ou ainda, qual a ferramenta que deverá ser escolhida usando que método para representar um problema.

1.3 - Linguagem de Transferência de Semântica (STL)

"*STL é usada para transferir informações de forma semântica entre ferramentas dentro do contexto de uma organização humana, de uma plataforma de sistema de computação ou/e de uma aplicação computacional*" [IEEE 92]. Esta transferência de informações é muito importante, uma vez, que dentro de um ambiente computacional, é possível encontrar aplicações desenvolvidas dentro das mais diversas metodologias de desenvolvimento de software orientados a objetos.

Um exemplo da necessidade do uso de uma Linguagem de Transferência de Semântica é quando, no processo de desenvolvimento de software, é preciso fazer uma transferência de informações entre as ferramentas usadas no ambiente computacional.

Como a STL é uma linguagem usada para descrever informações contidas em uma determinada ferramenta, de forma a documentar textualmete estas informações, ela é fácil para ler e escrever por programadores, usando apenas um editor de textos para o desempenho desta tarefa.

1.4 - Objetivos

O principal objetivo deste trabalho é verificar se é possível representar, através dos conceitos da Linguagem de Transferência de Semântica - STL[IEEE 92], os conceitos identificados nos métodos de especificação de software orientados a objetos de:

- Rumbaugh[RUMB 91];

- Coad/Yordon [COAD 91];
- Booch [BOOC 94];
- Shlaer e Mellor[SHLA 88] [SHLA 91];
- Wirfs-Brock[WIRF 90].

Os métodos de Especificação de Software relacionados acima, são usados pelas ferramentas CASE para automatizar as atividades do desenvolvimento de software, desde a fase de especificação até a geração de código. O problema é que cada ferramenta tem uma forma particular de representar o software sendo desenvolvido. Isto implica em falta de integração entre ferramentas CASE, principalmente quando se trata de produtores ou fabricantes de softwares diferentes.

A solução, portanto, é estabelecer uma forma comum de representação dos métodos de especificação de software que permitam às ferramentas CASE trabalharem com suas representações internas particulares, mas que possam trocar informações com outras ferramentas CASE. Esforços neste sentido estão concretizando-se através da Comissão IEEE que estuda a proposta de padronização sob o número 1175. Ela trata de uma linguagem de transferência de informação semântica (STL - Semantic Transfer Language) entre módulos ou componentes.

É de particular interesse desse trabalho analisar como o metamodelo STL representa os conceitos dos métodos de especificação de software citados anteriormente.

É objetivo também oferecer um estudo comparativo entre estes métodos no que diz respeito à capacidade de modelagem estática e dinâmica de cada um.

1.5 - Contribuições do Trabalho

Espera-se contribuir de duas formas: a primeira é através da identificação do problema de compatibilidade entre os conceitos utilizados por diferentes métodos de especificação de software orientado a objetos; a segunda é através da crítica do metamodelo IEEE 1175 [IEEE 92], no qual STL está inserida, verificando se os conceitos dos cinco métodos são suportados.

A discussão sobre os conceitos principais que cada modelo utiliza, colocando estes

conceitos em termos dos conceitos STL, representa um esforço inicial para a construção de uma plataforma comum de especificação de software orientados a objetos.

Uma crítica ao modelo estabelecido no documento IEEE 1175 [IEEE 92] pode ser citada como contribuição. Com a evolução dos modelos de especificação de software orientados a objetos houve expansão dos conceitos tornando ambíguo alguns conceitos STL.

1.6 - Organização da Dissertação

Ao longo deste trabalho, serão usadas as palavras *método* e *metodologia*. Segundo Silveria Bueno [BUEN 90], o significado da palavra *método* é o modo de proceder; ordem que se segue para alcançar um fim determinado. Já a palavra *metodologia* representa tratado dos métodos; orientação para o ensino de uma disciplina ou método.

Enquadrando os conceitos acima descritos neste trabalho, um *método* estabelece um molde ou ordem que deve seguir para se especificar o software. A *metodologia,* por sua vez, orienta ou disciplina a aplicação correta do método. Ou seja, ela ensina a usar o *método. Método* e *metodologia* compõem um *modelo*.

Esta dissertação está organizada como segue.

O Capítulo 2 oferece uma visão geral dos métodos Rumbaugh, Coad/Yordon, Booch, Shlaer e Mellor, e Wirfs-Brock, com a identificação de seus principais conceitos.

No Capítulo 3 será apresentada a Linguagem de Transferência de Semântica. Adiante-se que os conceitos pertinentes à Linguagem STL estão em inglês e em itálico. Isto porque o padrão 1175 foi projetado em inglês e a sua tradução textual para outra língua poderia provocar a perda do significado de algumas claúsulas para a comunicação entre ferramentas CASE.

No Capítulo 4 será examinado como mapear os conceitos STL em função dos conceitos dos métodos estudados, buscando identificar possíveis limitações desta representação para nortear extensões futuras da linguagem.

O quinto capítulo contém um sumário do trabalho com conclusões sobre as representações dos conceitos dos métodos estudados e dos conceitos STL. Neste capítulo também serão feitas considerações finais e sugestões para trabalhos futuros.

2. Uma Revisão dos Métodos de Especificação de Software Orientados a Objetos

2.1 - Introdução

Por alguns anos a orientação a objetos(OO) esteve associada apenas às linguagens de programação. Entretanto, como é visto em [MART 95], este cenário está mudando e cada vez mais os métodos de especificação de software OO ganham destaque no processo de desenvolvimento de software. Uma das vantagens do uso do paradigma OO é descrever conceitos do mundo real de forma mais natural que os métodos convencionais.

Neste capítulo será feito um estudo dos métodos de especificação de software orientado a objetos de Rumbaugh[RUMB 91], Coad/Yordon[COAD 91], Booch [BOOC 94], Shlaer e Mellor[SHLA 88][SHLA 91], e Wirfs-Brock[WIRF 90]. Serão identificados os principais conceitos destes métodos, assim como suas principais características. Finalmente, cada método será classificado de acordo com os aspectos estrutural e comportamental.

Identificar conceitos não é uma tarefa fácil. Muitas vezes não há consenso na determinação deles devido às diferentes visões de um mesmo cenário. Por isso, o exemplo que compõe o Apêndice A será utilizado neste capítulo para exemplificar o processo de identificação de alguns conceitos, objetivando desta forma um melhor entendimento dos métodos estudados.

2.2 - Método Rumbaugh (OMT)

O Método de análise chamado "Object Modeling Technique" - OMT, consiste de três fases:

- Modelagem do Objeto, onde as classes com seus respectivos atributos e relacionamentos são identificados.
- Modelagem Dinâmica, onde o comportamento dos objetos é especificado.
- Modelagem Funcional na qual as necessidades funcionais dos objetos são declaradas.

Estas três fases de modelagem descrevem os objetos no sistema e seus relacionamentos, descrevem as interações dos objetos no sistema e descrevem as transformações de dados do sistema. Estes modelos fazem parte da fase de análise que consiste em desenvolver um modelo do que supostamente o sistema faz, sem levar em consideração a implementação.

A fase de projeto consiste da otimização, refinamento e extensão do modelo do objeto, modelo dinâmico e modelo funcional detalhando-os para a implementação.

A seguir focalizaremos os principais pontos das fases de Análise, Projeto de Sistemas e Projetos de Objetos extraídos de Rumbaugh[RUMB 91].

2.2.1 - Análise

A fase de Análise diz respeito ao entendimento e modelagem da aplicação e ao domínio no qual ela opera. A meta da análise é desenvolver um modelo, em termos de objetos e relacionamentos, fluxo de controle dinâmico e transformações funcionais. A descrição inicial do problema, o modelo do objeto, modelo dinâmico e modelo inicial, são passos que caracterizam esta fase e que serão vistos a seguir.

2.2.1.1 - Escrever ou obter uma descrição inicial do problema

Normalmente a descrição inicial do problema utiliza verbos e nomes para relatar as particularidades de um determinado procedimento ou rotina do mundo real que se deseja resolver.

A fase de análise caracteriza-se pelo estudo das características do problema a ser resolvido. O ponto inicial são as entrevistas entre usuários e analistas. Uma descrição do problema é o documento básico sobre o qual deve-se aplicar as etapas do método de Rumbaugh.

2.2.1.2 - Construir o modelo do objeto

Com base na descrição do problema é feita a identificação das classes, os atributos são adicionados aos objetos e conexões, as classes são organizadas e simplificadas usando o conceito de herança, e é feito um cenário do problema e finalmente as classes são agrupadas em módulos.

Dessa forma, tomando como exemplo a especificação do problema do Apêndice A, os candidatos a classe poderiam ser sublinhados na especificação, relacionados a parte e os que representassem uma classe irrelevante ou uma classe redundante ou um atributo de outra classe seriam eliminados. "Portão" seria uma classe. "Dispositivo" não seria uma classe por ter o mesmo sentido que "controlador" e dessa forma ser redundante.

As associações poderiam ser destacadas dos verbos contidos na especificação do problema. E os atributos identificados a partir de frases possessivas. "O portão baixa quando o trem passa no sensor 2" seria uma associação.

O modelo do objeto pode ser sintetizado através da seguinte expressão:

modelo do objeto = diagrama do modelo do objeto + dicionário de dados

Ou seja, existe uma representação gráfica e um repositório onde as informações textuais se concentram.

2.2.1.3 - Desenvolver um modelo dinâmico

Em qualquer modelo a análise dos requisitos é feita a partir de observações do mundo real feita por analistas para que os requisitos essênciais ao sistema sejam detectados. De acordo com a abordagem OMT, um cenário é construído a partir de uma situação do mundo real do cliente, ressaltando assim suas propriedades mais importantes. Nesta fase não há preocupação a nível de implementação. O especialista da aplicação deve ser capaz de entender e criticar a especificação.

Inicialmente, uma descrição do problema a ser resolvido deve ser feita de forma a fornecer uma visão conceitual do sistema proposto. O Modelo Dinâmico prepara cenários para seqüência de interações típicas; identifica eventos entre objetos e prepara o caminho do evento para cada cenário; prepara um diagrama do fluxo do evento para o sistema; desenvolve um diagrama de estado para cada classe que tenha um comportamento dinâmico importante e, finalmente, checa a competência e completeza de eventos entre porções do diagrama de estado.

O modelo dinâmico representa o controle da informação: a seqüência de eventos, estados e operações que ocorrem com um sistema de objetos. É uma coleção de diagramas de estados que interagem entre si via eventos, por isso podemos vê-lo como sendo:

modelo dinâmico = diagrama de estado + diagrama global do fluxo do evento

2.2.1.4 - Construir um modelo funcional

A computação dentro de um sistema é descrita pelo modelo funcional, o qual mostra como os valores de saída na computação são alcançados a partir dos valores de entrada, independente da ordem na qual são processados, ou seja, sem indicar como, por quem ou porque os valores são processados. Segundo Grahan [GRAH 94],“ *O modelo funcional não se distingue de uma DFD*[1] *tanto no que diz respeito a intenção quanto a funcionalidade”.*

Um modelo funcional pode ser construído através da identificação dos valores de entrada e saída; do uso do diagrama de fluxo de dados necessários para mostrar depêndencias funcionais;

da descrição do que cada função faz e da especificação dos critérios de otimização. Desta forma podemos ver o modelo funcional como sendo:

modelo funcional = diagrama de fluxo de dados + "restrições"

O modelo funcional tem a maior parte de sua especificação descrita de forma gráfica, através de um DFD. Informações complementares sobre a funcionalidade do sistema podem ser descritas de forma textual e são chamadas de restrições. Na realidade, as restrições podem aparecer também no modelo do objeto e dinâmico [RUMB 91]. As restrições podem especificar a dependência parcial ou completa entre entidades de um DFD.

De acordo com o que foi visto no modelo do objeto, no modelo dinâmico e no modelo funcional é possível dizer que a fase de análise desenvolve um modelo do que um sistema deve fazer. Este modelo é obtido através do resultado de cada uma das fases de construção do modelo do objeto, desenvolvimento do modelo dinâmico e da construção do modelo funcional. Todas as fases acima citadas devem ser verificadas, feita a interação entre as mesmas e finalmente refinadas.

2.2.2 - Projeto do Sistema

O projeto do sistema determina a arquitetura do mesmo, decide como otimizá-lo, define uma estratégia para resolver o problema e estuda a alocação de recursos.

Através do modelo do objeto, o sistema é organizado em subsistemas e suas prioridades estabelecidas para construir a versão inicial do projeto. Decisões são tomadas acerca de comunicação interprocessos, armazenamento de dados e implementação do modelo dinâmico.

2.2.3 - Projeto dos Objetos

Na fase do projeto dos objetos são abordados detalhes de implementação. É decidido o que é necessário para a representação funcional de um sistema sem entrar em detalhes a nível de linguagem ou banco de dados a serem utilizados. Isto é feito de acordo com a estratégia

---

[1] DFD é uma abreviação para Diagrama de Fluxo de Dados

estabelecida durante o projeto do sistema. A fase de Projetos aproxima-se mais do nível de implementação, cujos esforços estão concentrados na escolha das estruturas de dados e dos algoritmos para implementar cada classe.

Os algoritmos escolhidos determinam o comportamento e as estruturas de dados vão servir de base para a estrutura do objeto que provavelmente será otimizada visando uma implementação eficiente. É nesta etapa que as associações e atributos têm sua implementação determinada e os subsistemas são empacotados em módulos.

Esta fase pode ser detalhada através de passos como: a obtenção das operações para o modelo do objeto, o projeto de algoritmos para implementar operações, a otimização de caminhos de acesso para dados, a implementação de associações, a representação exata dos atributos do objeto e o empacotamento das classes e associações em módulos. O produto resultante desta fase seria a junção da documentação já discutida até aqui. O que pode ser expresso por:

projeto do objeto = modelo do objeto detalhado + modelo dinâmico detalhado + modelo funcional detalhado

Todas as atividades descritas no projeto do objeto servem para detalhar o sistema independentemente das ferramentas a serem utilizadas. A fase de projeto refina o modelo do sistema gerando um documento que reúne detalhes do modelo do objeto, do modelo dinâmico e do modelo funcional.

O Método de Rumbaugh[RUMB 91] sugere um modelo onde estão presentes os principais aspectos do domínio da aplicação. Este modelo contém uma descrição detalhada dos objetos com suas respectivas propriedades e comportamentos, sendo o mesmo construído na fase de análise e detalhado na fase de projeto com o objetivo de descrever e otimizar sua implementação. É um método voltado parra as fases de especificação, projeto e implementação do ciclo de desenvolvimento, e pode ser utilizado em qualquer tipo de sistema.

2.3 - Método Coad/Yordon (CYOOA)

Para Coad e Yordon[COAD 91], a análise "*é o estudo de um domínio de problema, que leva a uma especificação de comportamento observável externamente.*" A análise é feita para identificar o que é necessário ao sistema, quais são as suas características e para determinar o que o sistema deve fazer para que o cliente fique satisfeito.

O processo de análise deve começar com um documento cuja finalidade é formalizar as necessidades do cliente e estabelecer as prioridades. Esta formalização dar-se-á através de uma série de discussões envolvendo especialistas do domínio da aplicação, clientes, projetistas e outras partes interessadas.

Neste método as Classes e Objetos são identificados e refletem o domínio do problema, assim como as responsabilidades deste modelo em relação ao domínio do problema. A metodologia CYOOA, consiste das seguintes atividades: a) identificação de classes e objetos; b) identificação de estruturas; c) identificação de assuntos e d) definição de atributos e definição de serviços, que passamos a discutir a seguir.

2.3.1 - Identificação de Classes e Objetos

Para Coad/Yordon[COAD91], "*Um objeto é a abstração de alguma coisa dentro do domínio do problema, exprimindo as capacidades de um sistema de manter informações sobre ela, interagir com ela, ou ambos*". E, "*Uma classe é uma descrição de um ou mais objetos por meio de um conjunto uniforme de Atributos e Serviços, incluindo uma descrição de como criar novos objetos na classe*".

As Classes e seus respectivos Objetos devem expressar o domínio do problema e as responsabilidades do sistema. Os analistas devem perceber o domínio do problema, estudá-lo e entendê-lo. Só assim será possível definir bem as Classes e Objetos como uma abstração do mundo real, tomando como referência o domínio do problema.Os candidatos a classe devem ser listados com a razão pela qual está sendo proposto como candidato. Serão aceitos os candidatos que atenderem aos critérios acima descritos. No exemplo do Apêndice A, "Trem" atende aos

critérios de Coad e Yordon e a razão para isto reside no fato de que faz parte do domínio do problema e tem responsabilidades.

2.3.2 - Identificação de Assuntos

Segundo Coad e Yordon [COAD 91], "*Um assunto é um mecanismo para orientar um analista em um modelo amplo e complexo*". Um assunto corresponde a um nível em um diagrama de fluxo de dados. Os assuntos compõem a área de um problema. A área do problema é decomposta em assuntos os quais correspondem com a notação de níveis ou camadas no diagrama de fluxo de dados.

2.3.3 - Identificação de Estruturas

Estrutura dentro do escopo da análise orientada a objetos - OOA, expressa a complexidade do domínio do problema pertinente às responsabilidades do sistema. Este termo é usado para descrever uma Estrutura Gen-Espec (generalização) e uma Estrutura Todo-Parte (especialização).

A Estrutura Gen-Espec é representada através de uma classe de generalização e de classes de especialização. Dentro dessas estruturas é aplicado o conceito de herança, onde as classes de especialização, herdam atributos e serviços da classe de generalização.

No exemplo do Apêndice A, a classe "semáforo" poderia ser colocada dentro da estrutura Gen-Espec, aplicando assim o conceito de herança onde "semáforo trem" e "semáforo carro" seriam classes de especialização herdando atributos e serviços da classe e generalização "semáforo".

2.3.4 - Definição de Atributos

Segundo Coad e Yordon[COAD 91], "*Um atributo é algum dado (informação de estado) para o qual cada objeto em uma classe tem seu próprio valor*". Os atributos são características atribuídas a um objeto, têm seu próprio valor e são identificados e posicionados

em suas respectivas classes. Este valor representa o estado de um objeto. Quando o valor de um atributo é alterado, significa que o estado do objeto também foi alterado. A manipulação dos valores dos atributos é feita através dos serviços do objeto.

2.3.5 - Definição de Serviços

Os serviços indicam o comportamento de um objeto dentro de uma classe. A definição textual de Coad e Yordon[COAD 91] para serviço é a seguinte: "*É um comportamento específico por cuja exibição um objeto é responsável*". Os Serviços são definidos através da identificação dos estados dos objetos, identificação dos serviços requeridos, identificação das conexões de mensagens, especificação dos serviços e da reunião da documentação de OOA.

Como foi visto nos tópicos apresentados acima, o método de análise orientada a objetos de Coad e Yordon é fortemente baseado na modelagem entidade-relacionamento do mundo dos dados. Depois da fase de análise, são projetados os componentes do sistema.

2.4 - Método BOOCH

Este método é voltado para a modelagem do objeto em termos de seus comportamentos, responsabilidades e colaborações ao invés de seus atributos e associações, enfatizando como os objetos alcançam seus objetivos.

O Método Booch consiste da representação dos aspectos estáticos do sistema, da estrutura lógica e física, e dos aspectos dinâmicos que dizem respeito ao diagrama de transição de estado e diagramas de tempo [GRAH 94].

Booch[BOOC 94] enfatiza que um processo de desenvolvimento é único e que os desenvolvedores devem encontrar um meio termo entre o formalismo do Macro Processo de desenvolvimento e o informalismo do Micro Processo de Desenvolvimento descrito por ele na Figura 2.1.

Na realidade, as atividades do Micro Processo de Desenvolvimento ilustrados na Figura 2.1, descritas a seguir, representam os passos do método Booch: a) identificar classes e objetos; b) identificar as semânticas das classes e objetos; c) identificar os relacionamentos entre as classes e objetos; e d) especificar a interface e a implementação das classes e objetos.

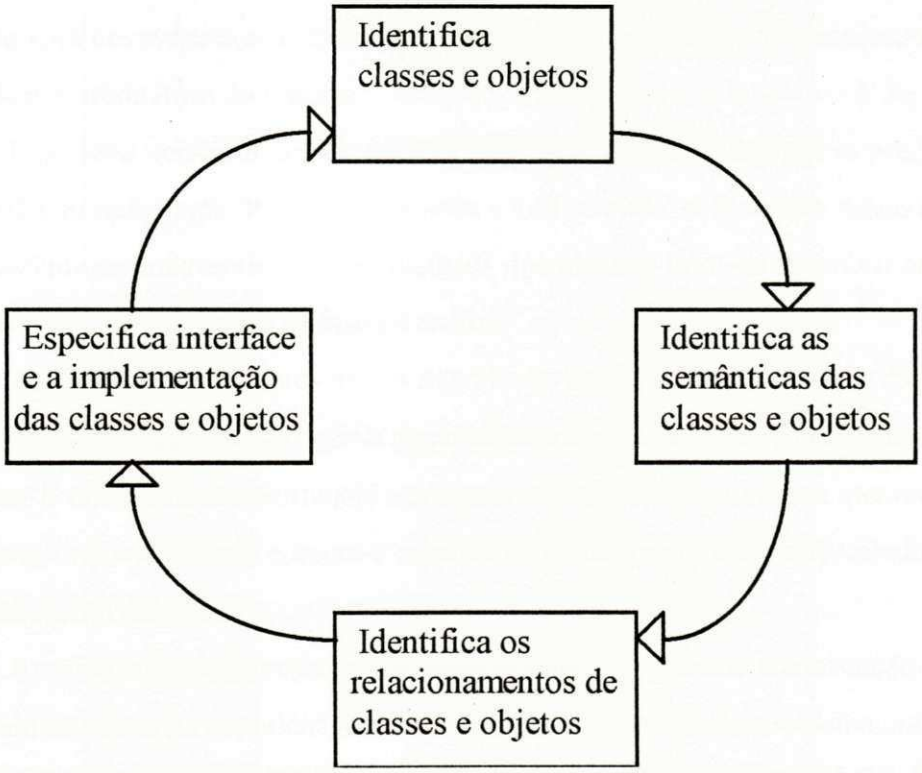

Figura 2.1 - O Micro Processo de Desenvolvimento

2.4.1 -Identificar Classes e Objetos em um Dado Nível de Abstração

Identificar classes e objetos em um dado nível de abstração é o primeiro passo dentro deste método e estabelece limites dentro do problema levantado. Os objetos devem ter regras bem definidas no domínio do problema assim como suas fronteiras devem também ser bem definidas. Como regras, Booch entende que um objeto deve ter uma identidade única (sendo identidade a propriedade na qual um objeto se distingue de outros), um comportamento e um estado. No exemplo do Apêndice A, "dispositivo" não tem uma identidade única pois refere-se ao controlador, devendo assim ser eliminada. "Portão" atenderia aos critérios de Booch, pois tem identidade única, comportamento e estado.

A definição de objeto para este método é altamente voltada ao aspecto comportamental, da mesma forma que o conceito de classe é uma abstração comportamental do objeto.

Segundo Booch[BOOC 94], na fase da análise, este passo é aplicado para decidir dentro do domínio do problema, o que é ou não interessante. Na fase de projeto este mecanismo é

aplicado para descobrir novas abstrações e durante a implementação é usado com o objetivo de simplificar a arquitetura do sistema.

É possível pensar em níveis de abstrações nos quais as informações precisam ser descritas em função de cada etapa. Por exemplo, após a análise pode-se descobrir novas abstrações devido a necessidade de um descrição mais detalhada dos objetos. Pode-se descobrir novas abstrações que não foram referenciadas na fase de análise.

As abstrações relevantes para o sistema devem ser colocadas em um dicionário de dados. É possível que durante este processo algumas informações além de adicionadas sejam também retiradas. É de essencial importância a criação deste dicionário, uma vez que o mesmo ajuda a estabelecer um vocabulário comum e consistente o qual permite aos que trabalham no projeto uma visão geral do mesmo.

As atividades aqui presentes são basicamente a descoberta e a invenção. Analistas procuram se basear na experiência de especialistas do domínio do problema uma vez que os mesmos são capazes de olhar para este domínio e identificar classes e objetos. É necessário também a habilidade de derivar novas classes e objetos do domínio da solução.

Esta fase é completada com sucesso quando é formado um dicionário consistente e contendo um amplo conjunto de abstrações.

2.4.2 - Identificar as Semânticas e Relacionamentos das Classes e Objetos

Este passo define semânticas e relacionamentos para os objetos que foram identificados na fase de identificação de classes e objetos. Deverá ajudar a compreender o comportamento das classes e objetos.

Para identificar as semânticas e relacionamentos para as classes e objetos identificados, é necessário saber que objetos usam que objetos, quais as mensagens que são enviadas entre os objetos, que classes estão associadas a que outras classes. Tudo isso está presente no diagrama do objeto, diagrama de classe, diagrama de tempo e diagrama de transição de estados, que devem ser construídos nessa fase. Nesse diagrama as linhas entre os objetos indicam o relacionamento *using* entre objetos e a direção das setas indicam o sentido que as mensagens estão sendo enviadas.

Segundo Booch[BOOC 94], na fase de análise, este passo é aplicado para alocar responsabilidades de acordo com os diferentes comportamentos que o sistema apresenta. Esta fase especifica as associações entre classes e objetos. Na parte de projeto, o objetivo é alcançar uma separação clara de assuntos entre as partes da solução apresentada, assim como especificar as colaborações responsáveis pelos mecanismos da arquitetura do sistema.

Esta fase produz um refinamento do dicionário de dados elaborado na fase de identificação de classes e objetos. Diagrama de classes, diagramas de objetos e diagramas de módulos também são produzidos nesta fase. As responsabilidades são fixadas às abstrações e, de acordo com o desenvolvimento, especificações podem ser criadas para as abstrações. Aqui, se houver alguma dificuldade para especificar de maneira clara as semânticas, é porque as abstrações não se apresentam de maneira satisfatória.

Neste método, a identificação das semânticas e relacionamentos das classes e objetos, é um passo completado com sucesso, quando é designado para cada abstração um conjunto de responsabilidades e operações. No que diz respeito a identificação dos relacionamentos, esta fase se completa quando são especificadas as semânticas e relacionamentos entre as abstrações.

2.4.3 - Implementar Classes e Objetos

Esta fase tem como objetivo enfatizar o aspecto comportamental e adiar o máximo possível as decisões acerca das representações das abstrações para evitar decisões prematuras a nível de implementação. Também faz parte a atualização do dicionário de dados, uma vez que novas abstrações podem ser descobertas ou inventadas durante a implementação das abstrações existentes.

Este método enfatiza como os objetos se relacionam através de suas interações comportamentais. Enfatiza também a colaboração entre os objetos e como um usa o outro, caracterizando assim uma abordagem baseada no comportamento.

O método Booch enfatiza como um objeto se comunica com outros objetos. É uma abordagem baseada no comportamento e oferece uma análise conceitual detalhada que procura capturar a semântica do sistema da melhor forma possível.

2.5 - Método Shlaer e Mellor (OOSA)

Esta abordagem se concentra quase que exclusivamente no uso da modelagem da informação como uma abordagem para identificação, classificação e abstração da informação sobre um domínio do problema [SHLA 88].

O Método proposto por Shlaer e Mellor enfatiza também a modelagem do comportamento do estado dos objetos (em termos de orientação a objeto) e a modelagem das ações realizadas de acordo com estes estados. Este método focaliza a modelagem da informação, uma vez que para Shlaer e Mellor isto é o que descreve e define o vocabulário e conceitualização do domínio do problema, o qual é o caminho apropriado para iniciar uma atividade de análise de software.

O Método proposto por Sally Shlaer e Stephen J. Mellor é chamado de OOSA - Análise de Sistemas Orientado para Objetos [SHLA 91] e é apresentado em dois livros. O primeiro livro da dupla "Object-Oriented Systems Analysis", concentra-se na modelagem da informação, uma vez que aborda como aplicar a sua técnica de modelagem de dados, como uma abordagem para a identificação, classificação e abstração da informação sobre o domínio do problema. Já o segundo, intitulado "Object Lifecycles", completa o primeiro, uma vez que introduz conceito de herança e discute a modelagem dinâmica, onde no modelo de estados são descritos os estados dos objetos e as transições entre eles. Este é um método que enfatiza fortemente a modelagem da informação.

Shlaer e Mellor [SHLA 88] mostram os passos necessários para o seu método: a) Identificar as classes de objetos no domínio do problema; b) Identificar os atributos dos objetos; c) Identificar os relacionamentos entre objetos; d) Identificar subtipos/supertipos e objetos associativos; e) Identificar diagramas para o ciclo de vida dos objetos; f) Identificar a dinâmica do sistema através dos modelos de comunicação do objeto; e, g) Identificação dos modelos de processo para as ações que ocorrem nos estados. Segue uma discussão sobre cada um dos tópicos acima.

2.5.1 - Identificar Classes e Objetos no Domínio do Problema

Shlaer e Mellor[SHLA 88] definem um objeto como sendo uma abstração de um conjunto de coisas do mundo real, as quais podem ser do tipo tangíveis, tais como um avião ou um trem; funções, como engenheiro ou médico; eventos, que como o próprio nome diz são ocorrências que acontecem em um tempo específico, como um vôo, ou um acidente. Também pode ser interações ou objetos de especificação. Dessa forma, "trem" seria um exemplo de classe no Apêndice A, por ser tangível segundo Shlaer e Mellor.

Após a identificação dos objetos, é necessário que se faça uma triagem de acordo com testes específicos para destacar objetos vagos ou com pouco significado para a modelagem do problema.

2.5.2 - Identificar os Atributos dos Objetos

Shlaer e Mellor[SHLA 88] definem atributos como características abstratas de objetos. A identificação dos atributos vem logo após a identificação das classes e pode ser classificada em três formas: *atributos descritivos*, os quais descrevem fatos que são próprios para todas as instâncias de um objeto; *atributos que nomeiam*, que determinam os nomes das instâncias; e *atributos referenciais*, os quais codificam relacionamentos entre objetos.

2.5.3 - Identificar os Relacionamentos entre Objetos

O Método de análise OOSA envolve identificação e definição de relacionamentos. Um relacionamento é a abstração de um conjunto de associações que contém sistematicamente diferentes espécies de coisas do mundo real[SHLA 91]. Os relacionamentos podem ser do tipo um para um (1:1), um para muitos (1:M) e muitos para muitos (M:M), como ilustra a Figura 2.2:

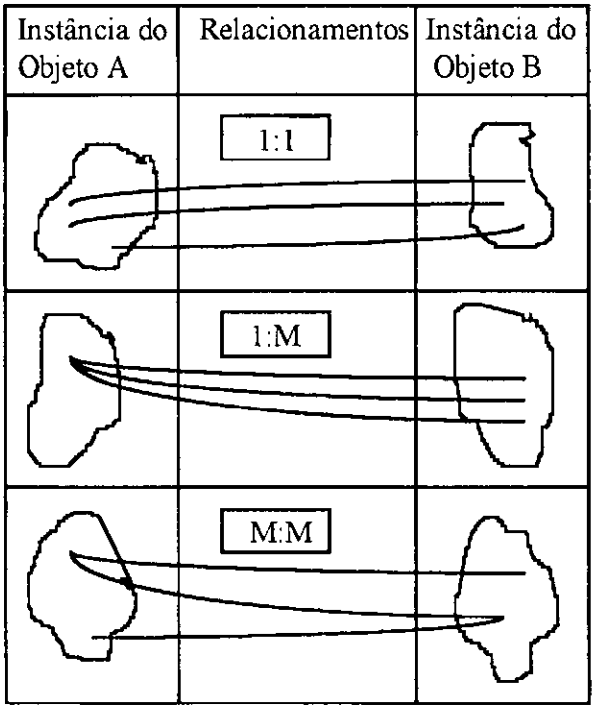

Figura 2.2 - Formas de relacionamentos binários [SHLA 88]

2.5.4 - Identificar Subtipos/ Supertipos e Objetos Associativos

É possível encontrar em alguns objetos especializados, certos atributos em comum. Neste caso é possível abstrair o objeto mais comum para representar as características compartilhadas pelos objetos especializados. Estes objetos são relacionados através de um relacionamento supertipo/subtipo.

Um objeto supertipo é análogo a uma "classe mãe", enquanto que um objeto subtipo é análogo a uma "classe filho". Ou seja, como o próprio nome diz, "a classe mãe" é aquela classe usada para a formação de novas classes. Naturalmente, os atributos comuns a objetos subtipos são alocados em objetos supertipos.

2.5.5 - Identificar Máquinas de Estados para o Ciclo de Vida do Objeto e Relacionamentos com Ciclos de Vida

O termo máquina de estado é usado para referir-se à um modelo de execução para uma instância particular. Uma máquina de estado pode ser vista como sendo uma cópia particular de um modelo de estado que é executado por uma instância simples.

Um *estado* representa uma condição do objeto que envolve um conjunto de regras, políticas e regulamentações. A cada estado é dado um nome o qual é único em seu modelo de estado[SHLA 91].

Já o comportamento de um objeto segue um ciclo de vida: os objetos são criados, passam por uma série de estágios e são eventualmente destruídos. Este ciclo de vida pode ser modelado neste método usando modelos de estados, especificamente máquinas de Moore. O modelo de estado de Moore pode ser visto como um conjunto de estados, um conjunto de eventos, um conjunto de regras de transição e ações.

Um conjunto de estados pode ser visto como sendo cada estado a representação de um estágio do ciclo de vida de uma instância que caracteriza um objeto. Regras de transição especificam como um novo estado é alcançado quando uma instância em um dado estado recebe um evento particular. Ação é uma atividade ou operação que deve ser executada quando uma instância chega em um estado. Uma ação é associada a cada estado.

Em alguns casos, uma associação entre coisas do mundo real avança através de estágios distintos durante os quais diferentes regras e políticas são aplicadas. Neste caso, a associação é abstraída como um relacionamento na forma de um objeto associativo como um atributo de estado corrente. Um modelo de estado é então construído para formalizar o ciclo de vida do relacionamento.

2.5.6 - Identificar Sistemas Dinâmicos através do Modelo de Comunicação do Objeto

O Modelo de Comunicação do Objeto (MCO) é composto de um resumo gráfico de comunicação de eventos entre modelos de estados e entidades externas, tais como operadores, dispositivos físicos e objetos de outros subsistemas[SHLA 91]. A forma de representação de um MCO é feita por uma forma oval achatada que recebe o nome do modelo do estado.

O modelo de estados representa o comportamento dos objetos ou de seus relacionamentos e é apresentado como um diagrama de transição de estado, onde as entidades externas que geram e recebem eventos são representadas através de um terminador. Uma seta é usada para representar um evento, saindo do modelo de estado ou entidade de geração para o que o recebe.

2.5.7 - Identificar Modelos de Processos para as Ações que Ocorrem nos Estados

Um diagrama de fluxo de dados constitui um modelo de processos, onde são expressas as atividades detalhadas tomando como base o modelo de estado. A ação é o ponto central de um sistema, uma vez que o funcionamento deste depende ou é declarado nas ações. Uma ação é executada por um estado ou um objeto.

Fisicamente, o modelo da informação consiste na organização e notação gráfica convenientes para descrição e definição do vocabulário, e conceitualização do domínio do problema[SHLA 91]. Este método é voltado para a análise, onde é elaborado o modelo da informação, modelos de estados e modelos de processos. É um método que também comporta as fases de projeto e de implementação.

2.6 - WIRFS-BROCK

Este modelo identifica, a partir da descrição do problema, os objetos necessários ao sistema, determina suas responsabilidades de acordo com o comportamento desses objetos e atribui estas responsabilidades a objetos específicos, determina também que colaboração deve ocorrer entre os objetos para cumprir estas responsabilidades.

Podemos citar os principais passos deste método, que constituem o que pode ser chamado de método de análise: identificação de classes, identificação das responsabilidades, das colaborações, das heranças de classes, contratos e protocolos de especificação[WIRF 90].

O método de análise será descrito nos tópicos a seguir.

2.6.1 - Identificação de Classes

As classes, assim como os objetos, servem para modelar o domínio da aplicação a ser desenvolvida. Para identificar estas classes, Wirfs-Broks *et al.* [WIRF 90] sugerem que deve ser observado quando mais de uma palavra define o mesmo conceito, de forma a ser escolhida a mais significante. Que os adjetivos devem ser olhados de forma cautelosa, assim como, durante este processo, algumas classes podem ser eliminadas e outras acrescentadas à lista.

Segundo a especificação do problema apresentado no Apêndice A, o conjunto de classes seria escolhido a partir dos nomes contidos na especificação do problema, de acordo com os critérios a seguir:

1 - O candidato à classe é um objeto fisico no domínio do problema?

2 - É uma entidade conceitual no domínio do problema?

3 - É uma categoria de classes? pode ser uma superclasse no modelo final.

4 - É ou tem sua própria interface?

5 - É o valor de um atributo?

Seriam eliminados os candidatos que não atendessem a 4 dos 5 critérios e que sugerissem um fato e não uma classe. Dessa forma, dentro da especificação do problema do Apêndice A, "sensor" seria uma classe por ser uma entidade conceitual, ser um objeto fisico e ter interface própria.

2.6.2 - Responsabilidades

Antes de descrever responsabilidades, alguns termos precisam ser explicados. O termo cliente diz respeito ao objeto que faz um pedido e o objeto que recebe o pedido e providencia o serviço é chamado de servidor. Um contrato descreve o caminho no qual um cliente interage com um servidor, ou seja, é uma lista de pedidos que um cliente pode solicitar a um servidor.

As responsabilidade incluem dois itens importantes: o conhecimento que um objeto mantém e as ações que um objeto executa. As responsabilidades de um sistema são todos os serviços que ele proporciona para todo contrato que ele suporta.

No exemplo do Apêndice A seria possível identificar as responsabilidades da seguinte forma: uma vez identificadas as classes, as frases com verbos que sugerem as ações de um objeto e as informações que um objeto deve manter, seriam selecionadas e um cenário seria montado. A partir deste cenário as responsabilidades seriam identificadas. As classes que não apresentarem responsabilidades devem ser rejeitadas. Neste exemplo, a classe "sensor" teria como responsabilidade detectar a presença de trens.

2.6.3 - Colaborações

A colaboração entre classes é um mecanismo que ajuda a distribuir as responsabilidades. Ou seja, algumas classes precisam da colaboração de outras classes para cumprir com uma responsabilidade. Segundo Wirfs-Brocks *et al.* "*As colaborações representam pedidos de um cliente para um servidor no cumprimento de uma responsabilidade do cliente. Uma colaboração é a inclusão do contrato entre o cliente e o servidor.*"[WIRF 90]. Na realidade, podem ocorrer duas situações: um objeto pode cumprir uma responsabilidade sozinho, ou seja, o cumprimento de uma responsabilidade pode não requerer necessariamente uma colaboração; ou um objeto pode precisar da colaboração de outros objetos, neste caso ele necessita enviar uma mensagem ao outro objeto. Segundo Wirfs-Brock *et al.*[WIRF 90],"*do ponto de vista do cliente, cada colaboração sua está associada com uma responsabilidade particular implementada pelo servidor*".

A colaboração dentro de uma aplicação é importante porque revela a comunicação entre as classes.

A identificação das responsabilidades pode proporcionar um ajuste dentro do projeto. Isto é possível, uma vez que, determinando as classes que desempenham o papel de cliente e de servidor dentro de um contrato, pode-se identificar a ausência ou o extravio de responsabilidades. O extravio de uma responsabilidade pode ser percebido pela identificação de uma colaboração sem estar associado a uma responsabilidade. Neste caso, é possível fazer um ajuste: adicionar a responsabilidade à classe, seja o problema identificado em classe, cliente ou servidor.

Segundo a especificação do problema apresentado no Apêndice A, seria possível estabelecer as colaborações entre classes, de acordo com a lista de responsabilidades. Onde algumas classes devem sugerir que para o cumprimento de sua responsabilidade, é necessário a colaboração com outra classe.

No exemplo apresentado no Apêndice A, a classe "sensor" exige colaboração com a classe "trem".

2.6.4 - Contratos

"Um contrato define um conjunto de pedidos que um cliente pode fazer a um servidor."[WIRF 90]. Um contrato ajuda a entender melhor um sistema uma vez que agrupa responsabilidades e estas responsabilidades, segundo Wirfs-Brock[WIRF 90], podem ser determinadas de acordo com o seguinte:

- Grupo de responsabilidades usadas pelos mesmos clientes - um contrato representa um conjunto de responsabilidades coesas, ou seja, as reponsabilidades são usadas pelos mesmos clientes;
- Aumentar a coesão das classes - aumentando a coesão das classes, a tendência é minimizar a quantidade de contratos suportados por cada classe. Uma classe deve suportar um conjunto coeso de contratos;
- Diminuir o número de contratos - um sistema simples é mais fácil de ser entendido, testado e mantido. E esta simplicidade pode ser alcançada com a diminuição do

número de contratos, sendo o melhor caminho para isto, ver quais responsabilidades devem ser generalizadas.

Um contrato entre duas classes representa uma lista de serviços que uma instância de uma classe pode pedir à uma instância de outra classe. Um serviço pode ser a realização de alguma ação ou o retorno de alguma informação.

Nesta fase pode ser desenhado um gráfico de colaboração, onde é mostrado a colaboração entre as classes e os contratos que as classes suportam.

2.6.5 - Especificação de Protocolos

"Um protocolo é um conjunto de assinaturas pelo qual uma classe responderá"[WIRF 90]. A construção de um protocolo para uma classe implica na especificação de assinaturas para os métodos que a classe implementará. É comum descobrir durante este processo, algumas falhas no projeto, como falta de precisão, falta de corretude ou falta de clareza. Assim, a especificação de protocolos contribui para a correção destas falhas.

Este método é voltado para o projeto e implementação. A fase de projeto é composta por uma fase inicial, onde as classes, responsabilidades e colaborações são definidas e por uma fase de Análise onde as hierarquias, subsistemas e protocolos são construídos.

2.7- Identificação dos Principais Conceitos e Características dos Métodos Estudados

Nesta seção serão identificados os principais conceitos dos Métodos OMT de Rumbaugh *et al.*, CYOOA de Coad e Yordon, Booch, OOSA de Shlaer e Mellor e W²OOD de Wirfs Broock *et al.* Será mostrada uma tabela de herança em função dos conceitos identificados nas seções anteriores Também será apresentado um resumo das principais características dos métodos estudados e uma classificação dos conceitos de acordo com os aspectos estruturais, funcionais e comportamentais de cada um..

2.7.1- Identificação dos Principais Conceitos do Método OMT

No Método OMT foram identificados os conceitos de Objeto, Classe, Links, Associação, Generalização, Diagrama de estados, Cenários, DFD, Eventos, Agregação e Herança.

2.7.2 - Identificação dos Principais Conceitos do Método CYOOA

No Método CYOOA foram identificados os conceitos de Classes e Objetos, Estruturas, Assunto, Atributos, Estados, Mensagens e Serviços.

2.7.3 - Identificação dos Principais Conceitos do Método Booch

No Método Booch foram identificados os conceitos de Classes, Objetos, Semânticas do comportamento e Relacionamentos. Os eventos e estados são representados através do diagrama de transição de estados.

2.7.4 - Identificação dos Principais Conceitos do Método OOSA

No Método OOSA foram identificados os conceitos de Classes e Objetos, Atributos, Relacionamentos, Subsistemas, Diagrama de Fluxo de Dados - DFD, Modelos de Estado, Subtipos, Supertipos e Objetos Associativos.

2.7.5 - Identificação dos Principais Conceitos do Método W^3OOD

No Método W^3OOD foram identificados os conceitos de Classes, Responsabilidades, Colaborações, Contratos, Subsistemas e Especificação de Protocolos. A especificação de protocolos é um conceito que só está presente no método W^3OOD e especificam uma assinatura para os serviços que implementam as responsabilidades.

2.7.6- Características dos Métodos Estudados

OMT

Este método enfatiza fortemente o aspecto da modelagem da informação, uma vez que começa com a construção da modelagem do objeto onde tem descrito os atributos e os relacionamentos entre os mesmos. Neste modelo são encontradas abstrações (superclasses) baseadas em atributos e relacionamentos comuns. Isto caracteriza-o como um método baseado na informação.

A herança está relacionada aos atributos e associações, ou seja, classes podem herdar atributos e associações de outras classes. Suporta também o conceito de múltipla herança, onde uma classe pode ter mais de uma Superclasse, podendo herdar diretamente características de mais de uma Superclasse.

Resumindo, Rumbaugh[RUMB 91] é um modelo baseado na informação, e que suporta, além dos conceitos descritos na seção 2.7.1, o conceito de classes abstratas, no qual uma classe não tem uma instância direta, e o conceito de múltiplas associações.

CYOOA

Este modelo também é baseado na informação e é caracterizado como um método de análise, tendo sua preocupação concentrada no domínio do problema.

Alguns conceitos deste método como classes e objetos, estruturas, atributos e serviços, são fundamentais. CYOOA suporta herança de atributos e serviços, ou seja, as classes podem herdar atributos e serviços de outras classes, entretanto não discute o caso de conflitos nos atributos herdados, no caso de herança múltipla.

A agregação é considerada um certo tipo de associação e um símbolo gráfico é utilizado para demonstrar isto.

BOOCH

Este é um método considerado como voltado ao projeto e baseado no comportamento. Booch enfatiza como os objetos se relacionam uns com os outros, e como se dá a colaboração entre eles. A herança se dá através dos serviços e responsabilidades comuns.

O Método Booch usa um relacionamento do tipo *using* que mostra quais os serviços que um objeto deve invocar. Para Booch, agregação é um tipo de modelagem alternativa para herança múltipla. Ele enfatiza que quando a soma das partes de um objeto é maior que o mesmo, então, neste caso, o conceito de agregação pode ser usado.

OOSA

Este método é baseado na informação, uma vez que enfatiza fortemente a modelagem da informação e é caracterizado como um método voltado para análise. É um método muito rico quanto a representação de estados, eventos, processos e objetos.

OOSA não associa serviços diretamente ao objeto, estando o comportamento do objeto ligado a modelagem dinâmica do mesmo. O uso de múltiplas associações é encorajado neste método, sendo agregação considerada um tipo de associação. O conceito de herança usa atributos, ou seja, as classes podem herdar atributos de outras classes.

W^3OOD

O modelo W^3OOD é baseado no comportamento e é voltado para o projeto. As responsabilidades descrevem o que um objeto deve fazer e o que deve conhecer. É um modelo que suporta o conceito de classes abstratas e usa as responsabilidades que são comuns no conceito de herança, ou seja as classes podem herdar responsabilidades de outras classes.

2.7.7 - Características Herdadas de acordo com os Métodos

Como é mostrado na Tabela 2.1, a herança está presente em cada um dos métodos estudados. As características herdadas diferem de acordo com cada método estudado.

Conceitos	OMT	CYOOA	Booch	OOSA	W³OOD
Associação	X				
Agregação					
Atributos	X	X	X	X	
Serviços		X	X		
Responsabilidades					X

Tabela 2.1 - Características herdadas de acordo com os métodos

Os métodos citados na Tabela 2.1 usam o que tem de comum em alguns conceitos para determinar a hierarquia de herança. O Método OMT de Rumbaugh usa atributos e associação para determinar a hierarquia de classes. O Método CYOOA de Coad e Yordon usa atributos e serviços. Booch usa o que tem de comum em estrutura e comportamento. Para Shlaer e Mellor, a hieraquia de herança é determinada através dos atributos. E Wirfs-Brock usa o que tem de comum nas reponsabilidades para determinar herança.

2.8 - Classificação dos métodos segundo os Aspectos Estrutural e Comportamental

Os principais conceitos dos métodos estudados neste trabalho foram identificados na Seção 2.7. Estes conceitos serão mapeados no Capítulo 4 em função dos conceitos STL que serão apresentados no Capítulo 3. Para uma melhor visualização do mapeamento, os conceitos identificados neste capítulo serão classificados nesta seção segundo os aspecto estrutural e comportamental.

O aspecto estrutural trata da modelagem estática dos objetos que inclui a associação entre objetos, agregação, e outros conceitos inerentes a cada método. O aspecto comportamental, por sua vez, trata da modelagem dinâmica onde estão descritos os conceitos pertinentes as

interações entre os objetos, o comportamento de cada um e as mensagens ou eventos que os objetos recebem ou enviam.

OMT

Aspecto Estrutural - Quanto ao aspecto estrutural, o modelo identifica objetos, classes, atributos e links. Identifica também as associações e agregações entre os objetos. O conceito de herança pode ser usado para simplificar classes e objetos. As classes e associações são agrupadas em módulos.

Aspecto Comportamental - O aspecto comportamental é representado através da construção do diagrama de estado para as atividades do objeto, mostrando os eventos que são enviados e recebidos e as ações que ele executa, ou seja, prepara cenários, identifica eventos entre objetos, prepara um evento para cada cenário e identifica valores de entrada e saída.

A Tabela 2.2 apresenta os principais conceitos do método OMT em relação aos aspectos Estrutural e Comportamental.

Principais Conceitos do Método	**Aspecto Estrutural**	**Aspecto Comportamental**
Classes e Objetos	X	
Atributos	X	
Links	X	
Associação	X	
Diagrama de estados		X
Cenários		X
Eventos		X
Generalização	X	
DFD		X
Agregação	X	

Tabela 2.2 - Classificação dos conceitos OMT

O Método OMT também trata o aspecto funcional dos objetos. Para ele o aspecto funcional modela ou decompõe o sistema em termo de suas funções, onde os valores de entrada e saída são identificados, assim como os eventos que ocorrem entre os objetos. Este aspecto pode ser sintetizado através de um diagrama de fluxo de dados para mostrar como o fluxo de informação é processado.

CYOOA

Aspecto estrutural - Quanto ao aspecto estrutural, são identificados os objetos e as estruturas e definidos os atributos, instâncias e assuntos. Os conceitos de generalização e agregação estão inseridos no conceito de estruturas, ou seja, as estruturas deste método representam agregação e generalização.

Aspecto comportamental - Quanto ao aspecto comportamental, o modelo identifica os estados dos objetos e identifica as mensagens.

A Tabela 2.3 apresenta os principais conceitos do método CYOOA em relação aos aspectos Estrutural e Comportamental.

Principais Conceitos do Método	Aspecto Estrutural	Aspecto Comportamental
Classes e Objetos	X	
Estruturas	X	
Assunto	X	
Atributos	X	
Estados		X
Mensagens		X
Serviços		X

Tabela 2.3 - Classificação dos conceitos CYOOA

BOOCH

Aspecto Estrutural - o diagrama de classes dá uma visão estrutural das classes dentro do sistema, mostrando seus relacionamentos. Para ter uma visão estrutural dos módulos dentro de um sistema usa-se um diagrama de módulos, onde é mostrada a alocação das classes e objetos dentro de um módulo. Booch modela associação através do relacionamento *using* entre classes.

Aspecto Comportamental - O aspecto comportamental do método Booch pode ser observado através do diagrama de estado e do diagrama de eventos.

A Tabela 2.4 apresenta os principais conceitos do método Booch em relação aos aspectos Estrutural e Comportamental.

Principais Conceitos do Método	Aspecto Estrutural	Aspecto Comportamental
Classes e Objetos	X	X
Semântica		X
Relacionamentos	X	X
Eventos		X
Estados		X

Tabela 2.4 - Classificação dos conceitos Booch

OOSA

Aspecto Estrutural - no aspecto estrutural desse modelo, o domínio do problema é identificado, os subsistemas são definidos e é construído o modelo da informação através de seus objetos, atributos e relacionamentos.

Aspecto Comportamental - o aspecto comportamental é caracterizado pela construção dos modelos de estados, ou seja, o ciclo de vida dos objetos, ciclo de vida dos relacionamentos e o modelo de comunicação do objeto. É caracterizado também pela construção do modelo do processo, ou seja, diagrama de fluxo de dados e o modelo de acesso ao objeto.

A Tabela 2.5 apresenta os principais conceitos do método OOSA em relação aos aspectos Estrutural e Comportamental.

Principais Conceitos dos Métodos	Aspecto Estrutural	Aspecto Comportamental
Classes e Objetos	X	X
Atributos	X	
Relacionamentos	X	X
Subsistemas	X	
DFD		X
Subtipos/Supertipos e Objetos Associativos	X	
Modelos de Estados		X

Tabela 2.5 - Classificação dos conceitos OOSA

W³OOD

Aspecto Estrutural - O aspecto estrutural deste modelo identifica os candidatos a classes, define responsabilidades, define colaborações, constrói gráficos de hierarquia que ilustram a herança, constrói hierarquias de classes, identifica possíveis subsistemas e simplifica colaborações, escreve um projeto de especificação para cada classe e escreve um projeto de especificação para cada subsistema.

Aspecto Comportamental - No aspecto comportamental este modelo define colaborações, escreve uma especificação de projeto para cada subsistema e escreve uma especificação de projeto para cada contrato e os protocolos.

ATabela 2.6 apresenta os principais conceitos do método W³OOD em relação aos aspectos Estrutural e Comportamental.

Principais Conceitos dos Métodos	Aspecto Estrutural	Aspecto Comportamental
Classes	X	
Responsabilidades	X	
Colaborações	X	X
Contratos		X
Subsistemas	X	X
Protocolos		X

Tabela 2.6 - Classificação dos conceitos W^{3}OOD

Serão mostrados na Tabela 2.7 os conceitos que são comuns a todos os métodos e como eles são representados de maneira diferente em cada uma dos métodos.

Termo comum	OMT	CYOOA	BOOCH	OOSA	W^{3}OOD
Objeto	Classes e objetos	Classe-&-objeto	Classes e objetos	Calsses e objetos	Classes
Generalização/ Especialização	"tem", herança	"é um" "é tipo de"	"é um"	Supertipo/ Subtipo/ "é um"	"é tipo de"
Agregação	"Todo-parte" ou "parte de"	Estrutura Todo-parte	"Parte de "	"consiste de"	
Associação	"Role names"	Objetos associativos	*using*	Objetos associativos	"Colaboração" (*)
Atributo	Atributo	Atributo	Atributo	Atributo	Responsabilidade (**)
Módulo	Subsistemas	Assunto	Módulo	Domínio do Problema	Subsistemas

Tabela 2.7 - Representação dos conceitos, segundo o aspecto estrutural, comuns aos métodos

(*) O conceito de colaboração do método W^{3}OOD está muito próximo ao conceito de associação presente nos outros métodos.

(**) O conceito Responsabilidade inclui atributos e serviços, uma vez que para W^{3}OOD, responsabilidade é o conhecimento que um objeto mantém e as ações que ele pode executar.

Na primeira linha da Tabela 2.7, é mostrado como os métodos fazem referência a classes e objetos. A segunda linha mostra os termos usados pêlos métodos para referenciar Generalização Especialização. Agregação é um conceito que esta presente em todos os métodos com exceção a W³OOD. A última linha da tabela 2.7, mostra como os métodos modelam associação entre objetos. Para Booch as associações são efetuadas através do relacionamento *using* entre classes. W³OOD não modela associação como os demais métodos, ao invés de associação ele usa o conceito de colaboração, onde as responsabilidades são distribuídas.

A Tabela 2.8 ilustra a representação em cada método dos conceitos classificados como fazendo parte do aspecto comportamental. Observe que os quatro primeiros métodos têm o mesmo conceito para estado do objeto e evento, e mesma representação para estado e transição de estado. Os métodos CYOOA e OOSA tem o conceito de *thread* de execução para cenários ou para controle de eventos. W³OOD usa os conceitos de Contratos, Cenário, Gráfico de Colaboração e Protocolos para expressar o aspecto comportamental e DFD está presente nos métodos OMT e OOSA.

Termo comum	OMT	CYOOA	BOOCH	OOSA	W³OOD
Estado do Objeto	Estado do Objeto	Estado do Objeto	Estado do Objeto	Estado do Objeto	
Interação entre objetos	Evento	Serviços	Evento e serviços	Evento e serviços	Contratos
Cenário	Cenário	Thred de Execução		Thred de Controle	Cenário
Diagrama de Estado	Diagrama de Estado	Tabela de Estado	Diagrama de Transição de Estado	Modelo de Estado	Gráfico de Colaboração
DFD	DFD			DFD	
Protocolos					Protocolos
Diagrama de Eventos	Diagrama de Eventos		Diagramas de Interação	Ciclo de vida do objeto	

Tabela 2.8 - Representação dos conceitos, segundo o aspecto comportamental, em cada método

2.10 - Conclusão

O exemplo apresentado no Apêndice A foi a mola precursora para a identificação dos principais conceitos dos cinco métodos estudados. A partir de um problema comum, construiu-se especificações em cada um dos métodos e foi possível determinar os conceitos principais e a base para o mapeamento entre os conceitos.

Este capítulo discutiu os principais conceitos presentes em cada um dos métodos estudados, classificou estes conceitos de acordo com os aspectos estrutural e comportamental, estabeleceu uma relação entre os métodos mostrando como cada autor representa seus conceitos. A importância desta discussão reside na identificação de mecanismos que balizem o engenheiro de software na escolha de métodos de especificação de software orientados a objetos.

Este capítulo mostrou algumas diferenças entre os métodos estudados.

Mostrou que W^{3}OOD e Booch são métodos com características semelhantes, uma vez que ambos se preocupam com o comportamento dos objetos. Booch trata de responsabilidades e colaborações através do diagrama do objeto, no qual são mostrados os serviços e objetos que um objeto pode chamar. W^{3}OOD representa responsabilidades e colaborações através do gráfico de colaboração. W^{3}OOD não modela associação entre classes como os outros métodos, seu método trabalha com colaboração. Booch modela associações usando o relacionamento *using* entre classes.

Todos os métodos estudados trabalham com o conceito de herança. Sendo que CYOOA e W^{3}OOD trabalham também com o conceito de classe abstrata. Os métodos de CYOOA, OMT, OOSA e Booch, trabalham com o conceito de agregação como sendo um tipo de associação. W^{3}OOD não modela agregação em seu método.

A classificação dos conceitos identificados neste capítulo, mostra a capacidade de modelagem estática e dinâmica dos métodos. Esta classificação será utilizada no Capítulo 4 para organizar o mapeamento desses conceitos em função dos conceitos STL que serão estudados e classificados adequadamente no Capítulo 3.

3. Linguagem de Transferência de Semântica

Neste capítulo será apresentada uma visão geral da Linguagem STL (Semantic Transfer Language). Está linguagem está descrita de forma mais detalhada no documento "*IEEE Trial-use Standard Reference Model for Computing System Tool Interconnection*" da Comissão IEEE 1175 [IEEE 92].

Neste trabalho, STL não será utilizada para descrever textualmente os conceitos dos métodos estudados. Não existe uma representação explícita de um conceito como "classe" , por exemplo, em STL. Entretanto, a forma como STL poderá tratar os conceitos dos métodos orientados a objetos pode ser vista no apêndice B.

3.1 - Modelo para Transferência de Informações entre Ferramentas

Quando há troca de informações entre ferramentas CASE, é necessário se estabelecer um formato ou uma sintaxe da informação e seu significado ou semântica. Para ajudar a integrar ferramentas CASE, e facilitar a transferência semântica de informações entre elas, é que surgiu o Padrão IEEE 1175 [IEEE 92]. A descrição desta mecânica pode ser dividida em quatro partes: a) mecanismos para transferência de informações; b) processos de transferência de informações; c) descrição das informações transferidas; e d) informações transferidas.

a) Mecanismos para transferência de informações

Estes mecanismos são formados pela combinação de dispositivos de hardware e software que fazem a transferência de informação. Conforme podemos visualizar através da Figura 3.1, a transferência de informações pode ser feita de várias formas: diretamente entre duas ferramentas (item ①) normalmente usando comunicação entre processos; comunicação por intermédio de arquivos (item ②); comunicação baseada em repositório (item ③) onde várias ferramentas compartilham um mesmo espaço de comunicação; e finalmente, comunicação remota (item ④) onde é necessário um serviço de apoio ou sistema de comunicação para fazer a transferência entre as ferramentas CASE.

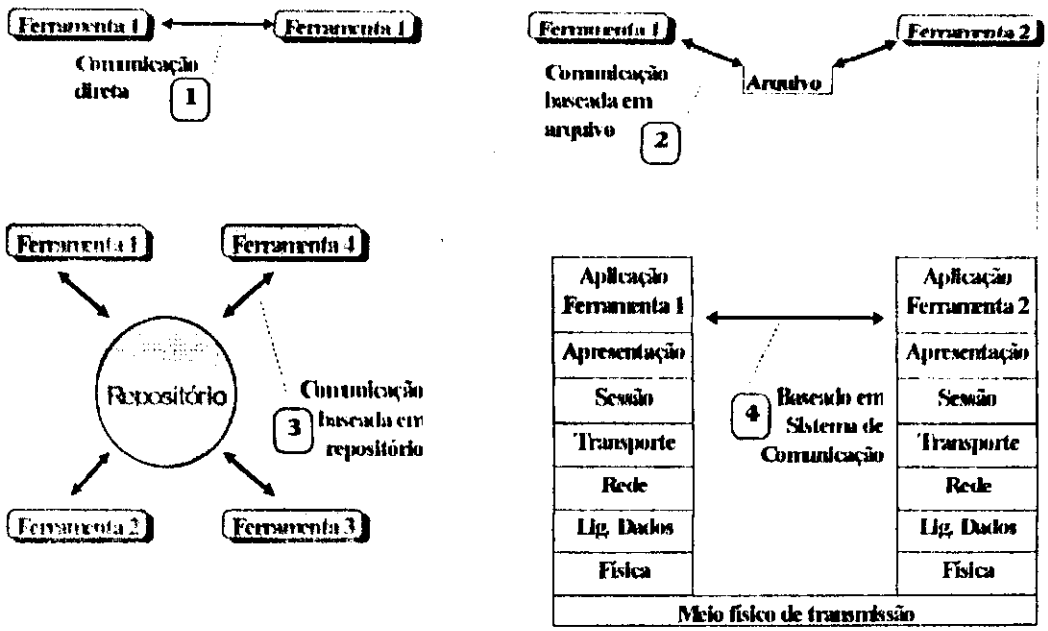

Figura 3.1 - Exemplos de mecanismos de transferência

Cada um dos mecanismos apresentados na Figura 3.1 serve a uma necessidade particular e deve ser selecionado de acordo com cada aplicação, levando em consideração o custo, o desempenho e outros atributos envolvidos que são importantes para a aplicação. Desta forma, STL foi projetada para ser independente do mecanismo de transferência utilizado [IEEE 92].

b) Processos de transferência de informações

Cada transferência de informações é composta de uma parte de *envio* e outra de *recepção*. Entretanto, pode não ser possível aos construtores de ferramentas estabelecer detalhes sobre o processo de transferência das partes *envia* e *recebe* para programas que ainda não foram construídos. Novas ferramentas CASE são anunciadas diariamente e continuarão a ser produzidas no futuro, com certeza. A política adotada pelo Padrão IEEE 1175 [IEEE 92] para acomodar estas mudanças pode ser vista como segue:

A ferramenta *envia* pode enviar qualquer informação que ela julgar necessária. A ferramenta *recebe* deve receber toda a informação que lhe for transmitida e estiver

disponível. Deve verificá-la e descartar as informações que não forem de interesse ou cujo conteúdo é questionável

A depender do usuário, a ferramenta que *recebe* deveria providenciar relatórios de diagnósticos descrevendo as informações descartadas e produzir um arquivo em STL contendo as informações descartadas.

c) Descrição da informação sendo transferida

A estrutura da informação a ser transferida precisa ser conhecida por todas as ferramentas que desejam se comunicar. A informação deve ser interpretada da mesma forma por todas as ferramentas. Cada unidade da informação transferida deve ter sua sintaxe (forma) e sua semântica (significado) definidos.

Informação sintática

A informação sobre a forma ou sintaxe da informação pode não ser usada na comunicação, mas será muito importante nos processos de *envia* e *recebe*. A documentação ISO/IEC 8824 (1990) e ISO/IEC 8825 (1990) descreve a padronização de unidade de transferência de informação cuja sintaxe está autocontida.

Informação semântica

Informação semântica representa o significado dos dados transferidos. As ferramentas que estão se comunicando só podem processar as informações de uma mesma maneira se elas compartilharem exatamente a mesma interpretação dos dados.

STL foi projetada de forma que qualquer ferramenta CASE atual poderá mapear ou transformar sua própria interpretação para uma forma padrão de interpretação, e, desta forma, estabelecer o significado da intercomunicação com outras ferramentas CASE.

d) Informação transferida

A informação sendo transferida pode ser usada para controle, gerenciamento, ou sobre um assunto ou tema cuja ferramenta é especialista.

Informações de controle

Informações de controle são aquelas usadas pelas ferramentas CASE que estão se comunicando para iniciar, parar, enviar, receber e reenviar uma informação. Diferenças entre

informação de controle e a transferência da informação de controle são o que separam métodos de transferência tais como transferência direta, sistemas baseados em repositório, sistemas baseados em arquivos, sistemas de comunicação. As informações de controle são tipicamente definidas pelos padrões existentes na plataforma utilizada.

Informações de gerenciamento

Informações de gerenciamento são utilizadas para ajudar a suportar as informações dos assuntos cujas ferramentas em comunicação são especialistas. Cada instância da informação sobre um conceito precisa ser gerenciada e controlada. Se as informações de gerenciamento e controle são transferidas pela conexão com a ferramenta, a informação deverá estar num formato padrão.

Informações sobre o assunto

A informação sobre o *assunto* é a informação atual que está sendo transmitida para a ferramenta. Informação sobre assunto inclui informações sobre requisitos de software, projetos, programas, e testes. Informações sobre assunto são descritas em termos de conceitos tais como *ações*, *dados*, *eventos*, *restrições*, e *estados*.

O assunto que é descrito através de STL (Conceitos representados em [IEEE 92] e discutidos mais adiante na Seção 3.3) trata dos seguintes conceitos: ações (dados, controle, e transformações de estado); informação (dados); eventos (tempo); lógica (condições); estados (contexto); relacionamento entre conceitos.

3.2 - A Linguagem STL

A Linguagem STL foi projetada para permitir interconexão entre ferramentas CASE. Ela foi projetada para ser compreendida diretamente por pessoas e pelo computador. Os principais simbolos com seus significados podem ser vistos na Tabela 3.1.

Símbolo	Significado
::=	é definido como
espaço	espaço, linhas em branco, ou <TABS>
x | y	escolha — ou *x* ou *y*
{x}	zero ou mais repetições de *x*
n{x}m	no mínimo *n* e no máximo *m* repetições de *x*
[x]	zero ou uma ocorrência de *x*
<x>	*x* deve ser substituído por outra construção da linguagem: *x* não é um elemento terminal da linguagem.
x	*x* não pode ser substituído por outro elemento da linguagem: *x* é um elemento terminal da linguagem.
NULL	expressão vazia
(texto)	o texto é um comentário e não faz parte da linguagem

Tabela 3.1: Símbolos utilizados para descrição da sintaxe em STL

Notação STL

A sintaxe STL é descrita usando-se uma versão modificada da forma *BNF* (Backus-Naur Form) [NAUR 60]. Os símbolos utilizados estão descritos na Tabela 3.1.

Pacote de informações STL

O conteúdo de uma transferência de informações é chamado de pacote de descrição do software, ou simplesmente *S_Packet*. Muitos *S_Packet* podem ser usados para descrever o mesmo sistema, hardware, ou produto de software, mas cada *S_Packet* deve conter informações sobre um, e apenas um, sistema, hardware, ou produto de software. A forma e as regras para interpretar *S_Packet* na notação BNF estão descritas no Quadro 3.1.

Quadro 3.1: Pacote STL

```
S_Packet ::= <identificação_da_setença>
 {<Sentença_STL>}
 pe_mark
```

A leitura do *S_Packet*, no Quadro 1, pode ser feita da seguinte forma: Um pacote de transferência é definido com uma *identificação_de_sentença* seguido por qualquer número de *Sentença_STL* seguidos pelos sete caracteres *pe_mark*. A identificação_de_sentença é uma sentença especial em STL projetada para suportar a transferência de informações de gerenciamento usando STL. Os caracteres *pe_mark* significam "package end mark[1]" e devem ser os últimos caracteres no pacote. As *Sentença_STL* podem vir em qualquer ordem.

Sentenças em STL

Todas as sentenças STL são construídas com as mesmas regras: Elas são compostas de outros elementos da linguagem como *palavras*, *cláusulas* e *pontuação*. No Quadro 3.2, podemos ver a definição de sentenças usando a sintaxe BNF.

Quadro 3.2: Sentença STL

```
<Sentença_STL> ::=
 <pal_chave_sentença> <identificador_da_sentença>
<Cláusula_STL> {;<Cláusula_STL>}.
```

A definição BNF, no Quadro 2, é lida da seguinte forma: Uma *Sentença_STL* é definida através de uma palavra chave que define uma setença, seguida de um identificador de sentença, seguida por uma ou mais *Cláusula_STL*, cada uma das quais deve ser separada por ponto e vírgula, sendo necessário um ponto final na última cláusula. A escolha da palavra chave da sentença é escolhida baseada nos conceitos da Tabela 3.2.

[1]Traduzindo para nosso idioma quer dizer: Marca de fim de pacote.

Action[2]	Data Key	Event Item
Colletion	Data Part	Event Type
Condition	Data Role	Graphical Symbol
Connection Path	Data Store	Object
Constant	Data Type	S_Packet
Data Item	Data View	State Transition

Tabela 3.2 - Nomes dos conceitos STL

Palavra chave da sentença

A palavra chave da sentença deve ser a primeira palavra em cada sentença STL. As palavras chaves da sentença identificam o conceito do software sendo descrito na sentença STL. A sintaxe em BNF de palavra chave da sentença é a seguinte:

```
<pal_chave_sentença> ::= <nome_de_conceito>
```

Onde *nome_de_conceito* é um dos nomes descritos na Tabela 3.2 e que serão discutidos nos capítulos seguintes. STL permite a extensão destes conceitos para poder suportar conceitos que surjam nas novas ferramentas CASE.

Identificador de sentença

Em cada sentença STL, o identificador é um termo simbólico usado como identificador único da sentença. O identificador pode ser introduzido pelo usuário ou pela própria ferramenta CASE . Garantir a unicidade do identificador é responsabilidade do criador do *S_Packet*.

[2]Preferimos manter todos os nomes dos conceitos em inglês, por consistência com a documentação original.

Cláusula STL

Cada cláusula STL descreve um relacionamento ou um atributo de uma sentença onde a cláusula aparece. A sintaxe BNF desta cláusula está especificada no Quadro 3.3.

Quadro 3.3: Cláusula STL

```
<Cláusula_STL> ::= <cláusula_relação> |
 <cláusula_atributo> |
 NULL
```

As cláusulas podem ou não estar presentes. Quando estão presentes, elas podem aparecer em qualquer ordem. Em qualquer sentença STL, uma sentença particular só deve estar presente uma e somente uma vez.

As cláusulas de relacionamento definem o relacionamento entre o sistema ou a instância do conceito de software sendo definido na sentença STL e um ou mais instâncias de conceitos definidos em outras sentenças STL. As cláusulas de relação podem descrever relacionamentos tais como abstração, agregação, conexão, apresentação, e restrição. O conjunto de cláusulas de relação é único para cada par de conceito de software. Veja a seguir a notação BNF:

```
<cláusula_relação> ::= <pal_chave_relação>
                       <lista_relação>
```

A última palavra em cada conjunto de cláusula de relação é o nome do conceito, cuja instância pode aparecer na lista de relação. Esta palavra funciona como classificador e permite checagem de tipos dentro da cláusula. Já a forma BNF da *pal_chave_relação* é a seguinte:

```
<pal_chave_relação> ::= <termo_relação><classif_relação>
<classif_relação> ::= <nome_conceito>
```

A *lista_relação* referencia as instâncias dos conceitos que participam do relacionamento. Sua forma BNF é a seguinte:

```
<lista_relação> ::= <membro_relação> {, <membro_relação>}
<membro_relação> ::=    <identificador_sentença> | TBD
```

A *lista_relação* contém um ou mais membros_relação. Este último pode ser um identificador de sentença ou uma palavra chave especial TBD que é a sigla para "To Be Determined" (a ser determinado) e é usado para permitir transferência de especificações que são conhecidas de forma incompletas no instante da transferência. Se a lista de relação contém mais de um identificador de sentença, os identificadores são separados por vírgula. Muitos relacionamentos podem ser expressos em duas formas complementares:

DATAITEM:	X	é usado pela ação	Y.
ACTION:	Y	usa dataitem	X.

As cláusulas de relação *é usado por* e *usa* expressam o relacionamento entre *X* e *Y*. Em STL, os relacionamentos podem ser expressos em qualquer uma ou ambas as direções. Por exemplo, a Figura 3.2 representa o Diagrama P1_P2 e o relacionamento produz.

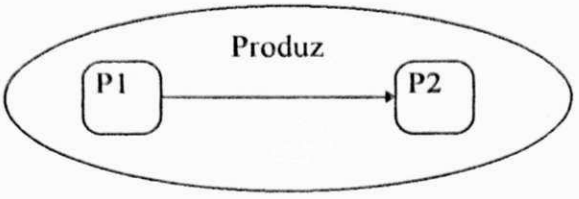

Figura 3.2 - Diagrama P1_P2

Observe que este diagrama pode ser descrito de duas formas: P2 é produzido por P1 ou P1 produz P2. Ou seja, pela riqueza do vocabulário STL uma ação pode ser descrita na voz ativa ou passiva.

Cláusula de atributo

Cláusula de atributo não contém nomes de conceitos. Ao invés disso, especificam propriedades intrínsecas ou características mensuráveis das instâncias dos conceitos

do software sendo definido nas sentenças em STL. A forma BNF das cláusulas de atributo é a seguinte:

<cláusula_atributo> ::= <pal_chave_atributo> <valor_atributo>

Por exemplo, através de um fragmento da descrição em STL do diagrama descrito na Figura 3.2 pode-se utilizar a palavra chave "P" que determina um processo e o valor do atributo "1" para indicar qual o processo, como ilustra a Figura 3.3.

```
P_1
        Gerada a partir de interrupcao
        Produz dataitem "produz"
        Tem duração máxima de 5 segundos
```

Figura 3.3 - Representação de atributos

Elementos da Linguagem STL

Os *tokens* ou elementos básicos da Linguagem STL são os *valor_atributo*, as *pal-chave-atributo*, identificadores de sentenças, ponto, vírgula e ponto e vírgula. Eles são compostos de um ou mais caracteres.

3.3 - Conceitos em STL

Nesta seção, os conceitos STL são apresentados de acordo com assuntos, propósito e tipo. Esta classificação mostra o que tem de comum entre os conceitos através do agrupamento em três categorias: Identificação da Informação, Estruturação da Informação e Comportamento da Informação.

3.3.1 - Categoria de Identificação da Informação - O conceito desta categoria identifica a informação contida no pacote de transferência cuja palavra-chave é S_Packet. Identificar a informação é mostrar o assunto de que o S_Packet trata e os dados de sua criação, como por exemplo: o assunto do pacote, a versão, quem criou o pacote e uma descrição.

3.3.2 - ***Categoria da Estruturação da Informação*** - Os conceitos que pertencem a esta categoria podem ser divididos em três grupos: grupo de valores; grupo de organização e grupo de imagem. O objetivo desta divisão é facilitar a interpretação humana e a aplicação da informação por outras instâncias de conceitos.

Grupo de valores - A STL tem dois conceitos que expressam valores: *Literal* e *Constant*. Estes conceitos determinam valores e são diferenciados por um identificador.

Grupo de organização - A STL utiliza dois conceitos pertinentes à organização da informação: *Collection* e *Object*. Estes conceitos organizam a informação em partes que expressam a estrutura hierárquica da aplicação.

Collection é um conceito que agrupa elementos que descrevem um software e *Object* é um conceito que expressa a agregação e encapsulamento das propriedades da aplicação.

Grupo de imagem - *GraphicSymbol* é um conceito que mapeia uma instância de conceito pertencente a categoria comportamental em uma instância de imagem definida externamente.

3.3.3 - ***Categoria do Comportamento da Informação*** - Contém os conceitos pertinentes a construção do pacote STL propriamente dito, mostrando como o mesmo expressa suas idéias ou como registra as operações de um sistema de hardware/software.

STL suporta seis conceitos para manipulação dos dados: *DataItem*, que descreve os dados em relação a lugar e tempo; *DataType*, descreve os tipos de valores que os dados podem assumir e um domínio estruturado de *DataType* que é descrito como uma estrutura de *DataPart*, ou seja, *DataPart* é um domínio estruturado de *DataType*; *DataKey* especifica identificadores para uma tabela de *DataType*; *DataRole* especifica regras para relacionamentos entre *DataTypes* e *DataView* especifica decomposição de subtipos para uma entidade *DataType*.

Nos conceitos *EventType* e *EventItem* os eventos são ocorrências que afetam a operação de um software, mas não é gerado pelo mesmo. O *EventType* descreve o tipo de evento e o *EventItem* descreve o evento propriamente dito.

Os conceitos *Condition* e *CondExpression* são conceitos lógicos, usados para descrever comparações entre uma ou mais representações das propriedades da aplicação. Outros conceitos como *Action* e *DataStore* são usados para descrever os efeitos das operações do software. *Action* providencia um caminho para especificar transformações e computações nas propriedades da aplicação e *DataStore* providencia um caminho para especificar a retenção das propriedades da aplicação entre ações.

O conceito *ConnectionPath* é usado para descrever o caminho que a informação percorre da fonte até o destino. Os conceitos *State* e *StateTransition* são usados para representar determinadas propriedades dentro de um software, assim como a seqüência com que elas ocorrem. Cada conceito *state* descreve um conceito diferente para uma ação. Já o *StateTransition* especifica a seqüência de estados.

A Tabela 3.3 mostra os conceitos STL em função das Categorias de Identificação, Estruturação e Comportamento da Informação. Nesta tabela, a linha que representa o conceito *Data*, engloba os conceitos *DataItem, DataType, DataPart, DataKey, DataRole e DataView*. A linha que representa o conceito Evento engloba os conceitos *EventItem e EventType*. A representação do conceito condição engloba os conceitos *Condition e CondExpression*. O conceito Estado engloba *State e StateTransition*.

Uma vez que os conceitos STL foram estudados e classificados segundo as categorias de estruturação e comportamento da informação, é possivel mapea-los em função dos conceitos dos cinco métodos estudados, como será visto no Capitulo 4.

Conceitos STL	**Identificação**	**Estruturação da Informação**	**Comportamento da Informação**
S_Packet	X		
Literal e *Constant*		X	
Collection e *Object*		X	
GraphicSymbol		X	
Date			X
Event			X
Condition			X
Action e *DataStore*			X
ConnectionPath			X
State			X

Tabela 3.3 - Os conceitos STL em função das categorias

4. Representação dos Principais Conceitos dos Métodos através de STL

O objetivo deste capítulo é analisar como os principais conceitos dos Métodos de Especificação de Software abordados no Capítulo 2 podem ser representados em função dos conceitos da Linguagem de Transferência de Semântica - STL [IEEE 92].

Trabalhar com conceitos, buscando sua representação, torna-se uma atividade empírica. Desta forma, será discutido aqui um conjunto de tabelas contendo o mapeamento entre os conceitos dos Métodos de Especificação de Software Orientado a Objetos e os conceitos que balizam o Padrão IEEE 1175, onde está definida a Linguagem de Transferência de Semântica (STL) [IEEE 92].

4.1 Identificação dos Conceitos STL

Os conceitos abordados nesta seção são responsáveis pela transferência da semântica da informação entre ferramentas e foram idealizados pela comissão IEEE 1175 [IEEE 92]. Como foi visto no Capítulo 3, onde este assunto já foi abordado, os principais conceitos que compõem a Linguagem STL são: *Action*, *Collection*, Condição, Conexão, Constante, *DataItem*, *DataKey*, *DataPart*, *DataRole*, *DataStore*, *DataType*, *DataView*, *EventItem*, *EventType*, *GraphicSymbol*, *Object*, *S_Packet*, *State* e *StateTransition*. Estes conceitos originalmente foram projetados para transferir informações entre ferramentas CASE. Aqui, usaremos STL como fonte padronizadora dos conceitos que dão suporte aos métodos de especificação de software orientados a objetos.

4.2 - STL Suporta os Conceitos dos Métodos Estudados?

Antes de discutir a tabela de mapeamento entre os conceitos dos métodos estudados em STL, vejamos como os conceitos STL se comportam diante da classificação adotada no Capítulo 2, segundo os aspectos estrutural e comportamental.

Os principais conceitos dos métodos identificados segundo o aspecto estrutural foram: objeto, generalização/especialização, agregação e associação, como pode ser observado na

Tabela 2.7. Segundo o Aspecto Comportamental, foram observados relacionamentos que envolvem conceitos como estado do objeto, eventos, diagrama de eventos, estados, diagrama de estados, cenários e linhas de execução conforme pôde ser observado na tabela 2.8 do Capítulo 2.

Agora, observe a Tabela 4.1 onde os Conceitos STL são colocados em função da Estruturação da Informação, do Comportamento da Informação e em função dos conceitos dos Métodos estudados.

Conceitos STL	**Estruturação da Informação**	**Comportamento da Informação**	**Conceitos dos Métodos**
Object	Agregação, Encapsulamento		Agregação, Objeto, Generalização/especialização e Associação
Date	Representação das propriedades da aplicação		Atributo
EventType e *EventItem*		Tempo e ocorrências de eventos	Eventos
Action e *DataStore*		Computação e armazenamento	Contratos, Serviços, Colaborações, Operação, Mensagens, Responsabilidades
ConectionPath		Caminho para propagação da informação	Links, DFD, Diagrama de Estado
Collection			Não há
State e *StateTransition*	Estado atual	Evolução do comportamento	Estado, Diagrama de Estado

Tabela 4.1 - Representação dos Conceitos STL em função de suas Categorias e dos Conceitos dos Métodos

A Tabela 4.1 é composta de quatro colunas. A primeira contém os principais conceitos da Linguagem STL que serão usados mais adiante para se relacionar com os conceitos dos cinco métodos estudados. A segunda e terceira colunas, chamadas respectivamente, "Estruturação da Informação" e "Comportamento da Informação" agrupam os conceitos STL segundo os aspecto estrutural e comportamental. Por fim, a quarta e última coluna lista os conceitos dos métodos que foram identificados no Capítulo 3. As seguir serão feitos alguns comentários sobre os conceitos apresentados nesta tabela:

Object - Em STL, o conceito *Object* representa um agrupamento lógico de dados e ação. Um objeto tem estado, ação, transição de estado, condição, recebe e envia mensagens. O significado da mensagem é especificado pela definição da ação e pode ser definida como sendo um *EventItem* ou *DataItem*. Objetos podem ainda encapsular outros objetos. Podem ser generalizados como um supertipo contendo ações, estados, transições, condições e *DataItem* comuns a vários subtipos.

Na Tabela 4.1, os conceitos *DataItem*, exceto *DataStore*, representam os atributos de uma aplicação.

Os conceitos *Action e DataStore* representam as propriedades de transformação do sistema. Eles podem ser usados para representar o fluxo de informação em diagramas dos métodos de especificação de software orientados a objetos.

Os conceitos *State* e *StateTransition* representam o estado e a sequência de transformações que um objeto pode sofrer.

Note que o termo "objeto" é usado em STL para suportar agregação e encapsulamento da informação a ser transferida entre ferramentas CASE. Objetos em STL podem ser combinados com outros conceitos para representar classes e objetos nos métodos. O encapsulamento e agregação dos objetos em STL permite a representação de diagramas presentes nos métodos os quais contém os conceitos de agregação, generalização/especialização(herança) e associações.

Em STL, o conceito *Data* é usado para suportar o conceito de objeto, uma vez que um objeto pode ser composto de dados mais ação e está associado à representação das propriedades da aplicação sendo modelada. Fazendo uma analogia deste conceito STL em

relação aos conceitos dos métodos estudados, *Data* pode suportar a representação dos atributos das classes e objetos.

Os conceitos *EventType* e *EventItem* estão associados ao comportamento da informação, refletindo o tempo ou instante onde houve mudança em determinados tipos de atributos de um objeto. O conceito correspondente nos métodos estudados é o conceito de evento. Apesar deste conceito não existir explicitamente no método de Wirfs-Brocks, os objetos recebem mensagens sinalizando, na realidade, a ocorrência de eventos externos ao objeto.

Note que o conceito *Action* em STL é uma forma de representar a computação e o armazenamento intermediário da informação a ser transferida entre ferramentas CASE. Alguns conceitos específicos a cada método tais como contratos, serviços, colaborações e responsabilidades poderiam ser descritos em termos de *Action* e *DataStore*.

O conceito *ConnectionPath* liga dois objetos e indica o caminho para a propagação da informação. É possível citar os *links*, diagramas de fluxo de dados e diagramas de estado nos cinco métodos estudados como conceitos usuários de *ConnectionPath*.

Finalizando os comentários gerais sobre a Tabela 4.1, o conceito *State* serve para mapear o conceito de mesmo nome nos métodos de especificação de software estudados, indicando através de diagrama de estado, a evolução do comportamento dos objetos de um método.

Apesar de STL ter sido projetada para transferir informações com semântica entre ferramentas CASE, pode-se identificar na sua descrição padrão IEEE 1175 [IEEE 92] palavras-chave também encontradas nos cinco métodos de especificação de software alvo deste trabalho. Os conceitos STL comportam cláusulas de atributos comuns e cláusulas de relacionamentos as quais são inerentes a vários conceitos e podem ser do tipo: Agregação, Associação, Especialização, como mostrado na Tabela 4.2.

Veja, as palavras extraídas da bibliografia sobre STL [IEEE 92], na coluna "Relacionamento entre conceitos STL", e da bibliografia sobre OMT de Rumbaugh [RUMB 91], CYOOA de Coad/Yordon[COAD 91], Booch [BOOC 94], OOSA de Shlaer e Mellor [SHLA 88][SHLA 91], e Wirfs-Brock [WIRF 90] listados na coluna sob o título "Relacionamento entre os conceitos dos métodos". Observe que alguns relacionamentos

podem ser mapeados diretamente. No caso da agregação, "é componente de" pode ser traduzido para "é parte de" e assim por diante nos demais relacionamentos. Em STL, um relacionamento que merece destaque é *Collection* (coleção). A ênfase, neste caso, é dada para a junção de conceitos com o propósito da sua transferência. Não ficou evidenciado um conceito semelhante nos cinco métodos de especificação de software orientados a objetos estudados.

Clausulas de Relação	**Relacionamento entre conceitos STL**	**Relacionamento entre os conceitos das métodos**
Agregação	"é componente de"	"é parte de" "consiste de"
Associação	*"uses" "carrie"*	*"uses" "Role names"*
Especialização/ Herança	*"has subtype" "is subtype of"*	*""has" "is kind of" "is a"*
Coleção	Agrupamento de conceitos	----- o ------

Tabela 4.2 - Representação dos Relacionamentos

A proposta aqui é usar um veículo comum, no caso o padrão 1175 [IEEE 92], para analisar os conceitos que dão suporte aos métodos de especificação de software orientados a objetos. A seguir será construída uma tabela onde será feito o mapeamento dos conceitos dos métodos estudados em função de STL.

Mapeando conceitos

- O primeiro passo é reunir em uma única tabela os aspectos estrutural e comportamental dos métodos estudados. A intenção é resumir os conceitos dos métodos de especificação de software orientados a objetos a um denominador comum para, a partir daí, identificar os conceitos correspondentes em STL. Como no Capítulo 2, um termo comum foi estendido aos conceitos comportamentais dos cinco métodos já classificados e uma nova coluna, contendo os conceitos STL, foi adicionada resultando na Tabela 4.3.

Termo comum	OMT	CYOOA	BOOCH	OOSA	W³OOD	STL
Atributo	Atributo	Atributo	Atributo	Atributo	Responsabi-lidade	*Data*
Classes e objetos	Classes	Classe-&-objeto	Classes e objetos	Classes	Classes	object *Action/Data*
Generalização/ Especialização	Herança	Estrutura Gen-Espec	Herança	Supertipo/ Subtipo/	Gráficos de hierarquia	object *DataType*
Agregação	Agregação	Estrutura Todo-parte	Multipla herança	"é parte de" "consiste de"		Object *DataType/* DataPart
Protocolo					Protocolo	Descrição Textual
DFD	DFD			DFD		*DataType/ DataItem/ ConnectionPath*
Associação	Associação	Objetos associativos	Relacionamento "using"	Objetos associativos	colaboração	Object ConnectionPath/ *DataItem*
Modularidade	Subsistemas	Assunto	Módulo	Domínio do Problema	Subsistemas	S_Packet
Estado do Objeto	Estado do Objeto	Estado do Objeto	Estado do Objeto	Estado do Objeto		State
Interação entre objetos	Evento	Serviços	Evento e serviços	Evento e serviços	Contratos	*EventType*
Cenário	Cenário	Thred de Execução	Diagramas de tempo	Thred de Controle	Cenário	*Action*
Diagrama de Estado	Diagrama de Estado	Tabela de Estado	Diagrama de Transição de Estado	Modelo de Estado		*StateTransition*
Diagrama de Eventos	Diagrama de Eventos	conexão de mensagens	Diagramas de Interação	Ciclo de vida do objeto	Gráfico de colaboração	*EventItem*

Tabela 4.3 - Conceitos dos métodos em função dos conceitos STL

Na Tabela 4.3, a discussão sobre os itens compreendidos entre a segunda e a sexta colunas encontram-se na Seção 2.9 do Capítulo 2. Estas palavras-chave foram extraídas da bibliografia que define cada um dos cinco métodos de especificação de software orientados a objetos estudados. Neste ponto, há interesse na discussão das colunas "Termo comum" e "STL". Cada linha da Tabela 4.3 será analisada a seguir.

Primeiro, o conceito atributo está presente em todos os métodos, sendo que em Wirfs-Brock este conceito está inserido no conceito de responsabilidade. Os conceitos do tipo *data* em STL representam os atributos.

Os conceitos "classes e objetos", estão mapeados em função do conceito STL *Object* como pode ser visto na segunda linha, este conceito está representando o encapsulamento de ação mais dados.

Na terceira linha, o termo "generalização" é colocado em função de *object*, ou mais especificamente do conceito de *DataType*. A "generalização/especialização" representa uma relação envolvendo classes. Para alguns conceitos STL, instâncias são especificadas mais efetivamente através de uma classificação hierárquica. Estes agrupamentos são baseados no grau de especialização através do uso de *DataType*. Por exemplo, a classe semáforo extraída do exemplo do Apêndice A, pode ser colocado como um gráfico de herança simples onde as classes semáforo, semáforo trem e semáforo carro são objetos nos métodos e podem ser traduzidos para STL. Note que neste caso, não se tem conhecimento de detalhes internos destes objetos.

No exemplo acima descrito, uma entidade *DataType* chamada semáforo pode ter especializações chamadas "semáforo trem" e "semáforo carro". O *DataType* semáforo é menos especifico, mais geral e mais abstrato(supertipo). As especializações são ditas serem mais especificas, menos geral e menos abstratas(subtipo).

Observando a quarta linha da Tabela 4.3 é possível identificar o conceito de agregação correspondendo ao mapeamento nos conceitos STL de *Object, DataType e DataPart*. STL trata agregação como uma instância a ser definida em níveis de estrutura ou agregação.

Protocolo é um conceito que só está presente no método de Wirfs-Brock, como pode ser visto na quinta linha da Tabela 4.3 e pode ser visto em STL como um descrição textual.

A sexta linha trata dos Diagramas de Fluxo de Dados que podem ser vistos em STL como sendo uma combinação dos conceitos *DataType, DataItem* e *ConnectionPath.*

A sétima coluna na Tabela 4.3 trata da tradução do conceito "associação" . É sobre este conceito que os métodos de especificação de sotftware mais divergem criando relacionamentos distintos. O termo associação, pode ser representado em STL através dos conceitos de *ConnectionPath* e *DataItem.* Na verdade existem vários conceitos em STL que combinados podem representar os conceitos pertinentes a parte estrutural dos métodos estudados. De uma forma geral, os objetos se relacionam através de um *DataRole*, onde a cardinalidade do relacionamento é destacada.

A oitava linha da Tabela 4.3 trata do mapeamento entre modularidade e *S_Packet.* "Modularidade" nos métodos de especificação de software representa como dividir a especificação em módulos ou pedaços. É possível mapear este conceito diretamente para S_Packet em STL, uma vez que é possível especificar livremente a granularidade de cada pacote a ser definido em STL de modo a contemplar a definição de "subsistemas", "assunto", "módulo" e "domínio" nos cinco métodos estudados.

A nona e décima linhas de Tabela 4.3 tratam do mapeamento do conceito estado do objeto para *State* em STL. O único método a não explicitar este conceito foi Wirfs-Brocks. Foi criada uma nova coluna chamada "interação entre objetos" para analisar o que estimula uma mudança de estado em um objeto. Na bibliografia estudada foi encontrada referência aos nomes "eventos" e "serviços" ou uma combinação de ambos para denotar a interação entre objetos, o que pode ser representado em STL por *EventType.*

- O termo *State* em STL pode representar o conceito de estado do objeto dos métodos estudados, uma vez que este conceito em STL representa o contexto no qual uma ação pode ocorrer.
- O cenário de execução de um procedimento está presente na décima primeira coluna da Tabela 4.3. Os termos selecionados da bibliografia que descrevem este conceito em cada um dos cinco métodos de especificação de software estudados resumem-se a "cenário", "linha de execução", "diagrama de tempo" e "linha de controle". A representação de cenário em STL é dependente da representação

particular em cada método, podendo ser descrita em termos de uma combinação de *Action, EventItem, EventType.*

Já o conceito *StateTransition* em STL representa a troca de estado do método sendo descrito, o que pode levar ao termo mapeado "diagrama de estado" na décima segunda linha.

Veja que o modelo OMT usa o termo "Diagrama de Estado", ao passo que os outros métodos CYOOA, BOOCH e OOSA usam, respectivamente, os termos "Tabela de Estado", "Diagrama de Transição de Estado" e "Modelo de Estado". O método de Wirfs-Brocks não é claro a este respeito, não foi identificado um conceito semelhante aos demais citados.

- O conceito de *EventItem*, em STL representa uma instância de um tipo de evento dentro de um contexto específico. Um *EventItem* pode ser carregado de uma ação para outra por um *connectionpath.* Um diagrama de eventos, dentro dos métodos de especificação de softwre especifica um conjunto de posibilidades de tempo no qual um evento pode ocorrer. Desta forma, um *EventItem* pode mapear um diagrama de eventos.

De uma forma geral, STL apresenta uma quantidade elevada de conceitos — como pode ser visto nos vários tipos de *Data (type, item, part, role, view e key)* — fazendo com que um pacote de informações a ser transferido usando STL possa ser escrito usando combinações diferentes deste conceito. Por exemplo, uma figura de um método pode ser descrita em função de *dataitem e collection* ou em função de *object* e *connectionpath.* Entretanto, todos os conceitos identificados nos métodos estudados são suportados em STL.

Aqui enfatiza-se o mapeamento dos principais conceitos dos cinco métodos estudados em função dos principais conceitos em STL como forma inicial de se estudar uma maneira para comparar o potencial de cada método. Não é objetivo deste trabalho representar especificações em STL, uma vez que este é um processo mecânico e pode ser automatizado utilizando-se artefatos de software como o *lex* e o *yacc* do Unix (foi deixado como sugestão para trabalhos futuros). O Apêndice B mostra como os conceitos STL combinados podem representar os conceitos dos métodos estudados.

A próxima seção mostrará como STL pode se adequar à representação dos principais conceitos dos métodos de especificação de software orientados a objetos estudados.

4.3 - Adequação STL para representar conceitos

Diferentes conceitos em STL providenciam diferentes cláusulas de relação. A Tabela 4.4 mostra alguns relacionamentos que são suportados por STL, os quais podem ser analisados de forma mais detalhada em função dos relacionamentos dos métodos estudados.

Conceito STL (Identificação)	**STL Relation**	**Conceito STL (Referencia)**	**Conceitos pertinentes aos Métodos**	**Booch**	**CYOOA**	**OMT**	**OOSA**	**WOOD**
Action	produces	*DataItem*	Evento/ Operação/ Mensagem	x	x	x	x	x
Action	receives	*EventItem*	Evento/ Mensagem	x	x	x	x	x
Action	transmits	*EventItem*	Evento	x	x	x	x	x
Action	uses	*DataItem*	Atributo	x	x	x	x	x
DataItem	is accepted by	*DataStore*	DFD/ Diagrama de Estados	-	x	x	x	-
DataItem	is produced by	*Action*	Evento/ Mensagem	x	x	x	x	x
DataItem	is store by	*DataStore*	DFD	-	x	x	x	-
DataItem	is suplied by	*DataStore*	DFD	-	x	x	x	-
DataItem	is used by	*Action*	Atributo/ Mensagem	x	x	x	x	x
DataRole	is involved in	*DataType*	Associações	x	x	x	x	x

Tabela 4.4 continua...

DataType	instantiates	*DataItem*	Instância/ Especialização	x	x	x	x	x
DataType	is a subtype in	*Dataview*	Herança	-	x	x	x	x
DataType	is partitioned by	*DataView*	Agregação/ Modulos/ Subsistemas	-	x	x	x	x
DataType	plays	*DataRole*	Objeto/ Especialização	-	x	x	x	x
DataType	involves	*DataRole*	Associação	x	-	x	-	x
DataView	partition supertype	*DataType*	Herança/ Especialização	-	x	x	x	x
DataView	provides subtype	*DataType*	Herança/ Especialização	-	x	x	x	x
EventItem	is an instance of	*EventType*	Evento	x	-	x	x	x
EventItem	is received by	*Action*	Evento	x	x	x	x	x
EventItem	is transmitted by	*Action*	Mensagem	x	x	x	x	x
EventType	instantiates	*EventItem*	Evento	x	x	x	x	x
State	characterizes	*DataItem*	Estado	-	x	-	x	-
StateTransition	is caused by	*Action*	Diagrama de Estado	x	x	x	x	-

Tabela 4.4 - As Relações STL em Função dos Conceitos dos Métodos Estudados

A Tabela 4.4 apresenta algumas cláusulas de relação em STL e relaciona estas cláusulas em função dos conceitos dos métodos estudados indicando através de um 'X' em que métodos se fazem presente. Na primeira coluna estão os conceitos STL relativos a identificação na cláusula de relação. Na segunda coluna estão presentes as cláusulas de relação entre conceitos STL. Na terceira coluna estão presentes os conceitos STL que são referenciados pelo conceito de identificação e pelas cláusulas de relação. Através desta tabela é possível perceber o quanto STL é detalhada. Na primeira linha da tabela, o conceito de ação

produz um dado. Nos métodos estudados uma ação pode ser representada através de contratos, serviços, colaborações, operações, mensagens e responsabilidades, uma vez que o conceito de ação é usado para descrever o comportamento do software no que diz respeito ao que causa as transformações dentro do software e quais são os seus efeitos.

Os relacionamentos que englobam o conceito *Action* podem ser vários, contudo, é analisado nesta tabela os relacionamentos do tipo: ação produz um *DataItem*, ação recebe um *EventItem*, ação transmite *EventItem* e ação usa *DataItem*. É possivel dizer que o cumprimento de uma responsabilidade produz um dado; que um contrato recebe um evento, transmite um evento e pode usar um dado.

Foi visto na Tabela 4.1 que os conceitos do tipo *Date* representam as propriedades da aplicação e podem ser vistos como atributos dentro dos métodos de especificação de software. Um *DataItem* em STL representa uma instância de um *DataType* dentro de um contexto especifico, onde *DataType* é um conceito que aglutina outros conceitos do tipo *Date* providenciando um caminho para especificar um conjunto de valores possíveis que um *DataItem* pode ter. Nesta tabela as linhas 5, 6, 7, 8 e 9 mostram as relações envolvendo *DataItem:* É aceito por um conceito do tipo *DataStore* que retém a informação por um determinado tempo. *DataItem* é produzido e pode ser usado por uma Ação, pode ser armazenado e fornecido por um *DataStore*.

O conceito *DataRole* providencia junto com o *DataType* a especificação da instância com a qual a entidade *DataType* está associada. Assim, este conceito pode ser envolvido e empregado por um *DataType*.

O conceito *DataStore* aceita um *DataItem*, é componente dele mesmo e armazena um *DataItem*. Também fornece um *DataItem*. Já um *DataType* instancia um *DataItem* é um subtipo em *DataView*. Onde, *DataView* providência a descrição da hierarquia de classes da entidade *DataType*.

Os conceitos do tipo evento representam o tempo e a ocorrência de eventos, nos métodos estudados um evento tem o mesmo sentido dos conceitos de *EventItem* que pode ser uma entrada ou saída para uma ação. Um *EventItem* representa uma instância de um *EventType* em um contexto especifico, sendo que *EventType* especifica um conjunto de

possibilidades de tempo na qual um ou mais *EventItens* pode ocorrer. Na Tabela 4.4 um *EventType* pode instanciar um *EventItem.*

O conceito *state* representa o contexto no qual uma ação pode ocorrer. Um *State* caracteriza um *DataItem.* Um *StateTransition* representa uma troca de estados no software e é causado por uma ação, conforme visto na Tabela 4.4.

Analisando este capítulo, é possível dizer que a área de engenharia de software está criando um novo ramo, que trata da especificação de software. Cada vez mais atenção é dada a esta fase do desenvolvimento de software, como pode ser visto nas publicações das revistas IEEE Software de Janeiro/96 (tema central Qualidade de Software) e Março/96 (tema central Engenharia de Requisitos). A edição de janeiro alerta para o problema da produção de código de qualidade e a de março relaciona a produção de um código de melhor qualidade com a Engenharia de Requisitos. Neste último número da IEEE Software, Brian Lawrence e Daniel Jackson [LAWR 96][JACK 96] discutem o problema do empiricismo desta fase de desenvolvimento de software, sendo o primeiro a favor de ferramentas que promovam o uso de métodos com pouco formalismo matemático. Já Daniel Jackson defende o formalismo com o argumento de maximizar a análise mecânica da especificação diminuindo sua ambigüidade.

Nos esforços dispendidos para a construção da tabela acima, foi constatada a necessidade de um nível de especificação seja perseguido para atender aos dois autores acima citados e minimizar a forma empírica como os métodos são empregados atualmente.

5 - Sumário, Conclusão e Sugestões

5.1 - Sumário

Os cinco métodos de especificação de software orientados a objetos de Rumbaugh *et al.*[RUMB 91], Coad e Yordon [COAD 91], Booch [BOOC 94], Shlaer e Mellor [SHLA 88] [SHLA 91], e Wirfs Broock *et al.* [WIRF 90] foram estudados neste trabalho, assim como a Linguagem de Transferência de Semântica - STL desenvolvida pela IEEE Comissão 1175 para comunicação entre ferramentas CASE. O objetivo deste trabalho foi analisar STL como uma fonte padronizadora dos métodos de especificação de software orientados a objetos. Para isto os seguintes passos foram seguidos: Primeiro, os métodos de especificação de software orientados a objetos tiveram seus principais conceitos identificados e classificados segundo os aspectos estrutural e comportamental; segundo, os principais conceitos da Linguagem STL foram estudados não apenas sob a visão para comunicação entre ferramentas CASE mas também como base padronizadora para os conceitos pertinentes aos cinco métodos estudados. E terceiro, objetivando criar uma discussão sobre a representação de conceitos, foi feito um mapeamento empírico entre STL e os conceitos principais que dão suporte aos métodos de especificação de software orientados a objeto.

Rumbaugh

O Método de Rumbaugh é bastante popular sendo chamado de "Object Modeling Technique" - OMT. Este método consiste das fases de Modelagem do Objeto, onde as classes com seus respectivos atributos e relacionamentos são identificados; Modelagem Dinâmica, onde o comportamento dos objetos é especificado; e a Modelagem Funcional na qual as necessidades funcionais dos objetos são declaradas. Estes três modelos descrevem, respectivamente, os objetos no sistema e seus relacionamentos descrevem as interações dos objetos no sistema e descrevem as transformações de dados do sistema.

Coad/Yordon

Para o Método de Coad e Yordon (CYOOA) o processo de análise consiste das fases de identificação das classes e objetos, da identificação das estruturas, da identificação de

assuntos,da definição de atributos e de serviços. O método de análise orientada a objetos de Coad e Yordon é fortemente baseado na modelagem entidade-relacionamento do mundo dos dados[COAD90].

Booch

O Método de Booch[BOOC 94] é voltado para a modelagem do objeto em termos de seus relacionamentos, responsabilidades e colaborações através de links (cliente/servidor). Booch descreve os passos do seu método como sendo a identificação de classes e objetos, a identificação das semânticas das classes e objetos, a identificação dos relacionamentos entre as classes e objetos, a especificação da interface e a implementação das classes e objetos.

Este método valoriza como os objetos se relacionam através de suas interações comportamentais, enfatizando assim a colaboração entre os objetos e como um usa o outro, caracterizando uma abordagem baseada no comportamento.

Método OOSA

O Método proposto por Sally Shlaer e Stephen J. Mellor é chamado de OOSA e se concentra quase que exclusivamente no uso da modelagem da informação como uma abordagem para identificação, classificação e abstração da informação sobre um domínio do problema [SHLA 88]. Os passos necessários para este método são a identificação das classes de objetos no domínio do problema, a identificação dos atributos dos objetos, a identificação dos relacionamentos entre objetos, a identificação de subtipos/supertipos e objetos associativos, a identificação de diagramas para o ciclo de vida dos objetos, a identificação da dinâmica do sistema através dos modelos de comunicação do objeto e finalmente a identificação dos modelos de processo para as ações que ocorrem nos estados.

Método W^3OOD

Para o método de Wirfs-Brocks, os objetos necessários são identificados a partir da descrição do problema. Daí as responsabilidades são determinadas de acordo com o comportamento desses objetos e como devem ocorrer as colaborações entre os objetos para cumprir estas responsabilidades. Os principais passos deste método constituem o que pode ser chamado de método de análise, onde há identificação de classes, identificação das

responsabilidades, identificação das colaborações, determinação das heranças de classes, definição dos contratos e protocolos de especificação[WIRF 90].

Cada método estudado teve seus principais conceitos identificados. Para OMT, os principais conceitos foram: Objeto, Classe, Links, Associação, Generalização, Agregação e Herança. Em CYOOA foram identificados os conceitos de Classes e objetos, Estruturas, Assunto, Atributos, Estados, Mensagens e Serviços. Para o método Booch foram identificados os conceitos de Classes, Objetos, Relacionamentos entre objetos. Para o método OOSA foram identificados os conceitos de Classes e Objetos, Atributos, Relacionamentos, Subsistemas, Diagrama de Fluxo de Dados - DFD, Subtipos, Supertipos e Objetos Associativos. E finalmente em Wirfs-Brocks foram identificados os conceitos de Classes, Responsabilidades, Colaborações, Contratos e Especificação de Protocolos.

As características dos métodos OMT, CYOOA, BOOCH, OOSA e W^{3}OOD foram ressaltadas na seção 2.8 do Capítulo 2 e na Tabela 4.3. Foi apresentada também a Tabela 2.1 onde estão presentes os conceitos que são herdados em cada método.

Cada método estudado teve seus conceitos classificados segundo o aspecto estrutural, funcional e comportamental. Estes conceitos foram colocados em função de um termo comum, uma vez que alguns métodos apresentaram nomes diferentes para conceitos similares, para permitir o mapeamento entre conceitos.

Linguagem de Transferência de Semântica - STL

STL está descrita em "*IEEE Trial-use Standard Reference Model for Computing System Tool Interconnection*" pela Comissão IEEE 1175 [IEEE 92]. Ela foi projetada para permitir interconexão entre ferramentas CASE e sua forma textual permite ser compreendida diretamente por pessoas e pelo computador. A sintaxe desta linguagem é descrita usando-se uma versão modificada da forma *BNF* (Backus-Naur Form) [NAUR 60]. Os conceitos STL são apresentados de acordo com assuntos, propósito e tipo. Esta classificação mostra o que tem de comum entre os conceitos através do agrupamento em três categorias: Identificação da Informação, que identifica a informação contida no pacote de transferência; Estruturação da Informação, que podem ser divididos em três grupos: grupo de valores, grupo de organização e grupo de imagem, e Comportamento da Informação, que contém os conceitos pertinentes a construção do pacote STL propriamente dito.

Mapeamentos dos conceitos dos Métodos OO em STL

Os principais conceitos dos Métodos de Especificação de Software Orientados a Objetos OMT de Rumbaugh *et al.*[RUMB 91], CYOOA de Coad e Yordon [COAD 91], Booch [BOOC 94], OOSA de Shlaer e Mellor [SHLA 88] [SHLA 91], e W^3OOD de Wirfs Broock *et al.* [WIRF 90] analisados foram mapeados em função dos conceitos da Linguagem de Transferência de Semântica - STL [IEEE 92].

O mapeamento entre os conceitos dos Métodos de Especificação de Software Orientado a Objetos e os conceitos da Linguagem de Transferência de Semântica (STL) [IEEE 92], está definido na Tabela 4.3 do Capitulo 4, que mostra como STL representa os conceitos dos métodos acima citados. O Apêndice B ilustra como STL combina conceitos para representar os conceitos dos métodos estudados

5.2 - Conclusão

A partir dos resultados apresentados pode-se concluir que a Linguagem de Transferência de Semântica suporta os principais conceitos dos métodos estudados. Tendo em vista as limitações do cenário de exemplo considerado, deduz-se que há uma deficiência no metamodelo, uma vez que o mesmo suporta um elevado número de cláusulas STL que contribui para mapeamentos ambíguos.

STL apresenta um elevado número de conceitos e cláusulas o que pode gerar ambigüidade. No entanto, o estudo feito nos métodos estudados ilustra as principais diferenças entre eles e justifica o elevado número de conceitos em STL, usados para permitir a transferência de informação semântica entre ferramentas CASE. Por outro lado, analisando os conceitos dos cincos métodos estudados, deduz-se uma deficiência no metamodelo que suporta o Padrão IEEE 1175, no qual está definida STL.

O elevado número de conceitos ou claúsulas STL contribui para mapeamentos ou descrições dos métodos em STL de maneira ambígua, pois um conceito pode ser empregado mais de uma vez no mapeamento dos conceitos dos cinco métodos analisados. É necessário portanto, um metamodelo com menos conceitos básicos e maior ortogonalidade entre eles, para poder representar os conceitos dos métodos de especificação de software orientados a objetos de forma menos empírica. Outra necessidade detectada nos métodos e em STL é a

construção de mecanismos que ofereçam um melhor detalhamento do comportamento dinâmico dos objetos e que permita rastreamento de mensagens entre objetos e que apoie a construção de ferramentas CASE voltadas para testes de integração de software orientados a objetos. Assim, o mapeamento entre os conceitos dos métodos de especificação de software orientados a objetos e STL, no futuro será feito de forma canônica.

Este estudo permitiu concluir algumas diferenças entre os métodos.

As associações entre classes estão presentes nos métodos, sendo que Booch modela associações usando o relacionamento *using* entre classes. Wirfs-Brocks não modela associação entre classes como os outros métodos. Ele usa a colaboração entre classes.

Todos os métodos estudados trabalham com o conceito de herança. Sendo que CYOOA e Wirfs-Brocks trabalham também com o conceito de classe abstrata. Os métodos de CYOOA, OMT, OOSA e Booch trabalham com o conceito de agregação. Wirfs-Brock não modela agregação em seu método.

Booch trata de responsabilidades e colaborações através do diagrama do objeto, no qual, através do relacionamento *using* presente no diagrama, são mostrados os serviços e objetos que um objeto pode chamar. Wirfs-Brock representa responsabilidades e colaborações através do gráfico de colaboração.

O diagrama de fluxo de eventos de Rumbaugh mostra como os eventos acontecem dentro do sistema e equivale ao diagrama de comunicação do objeto de OOSA. Ambos não mostram a sequencia na qual os eventos acontecem. OOSA e CYOOA mostram como os objetos acessam determinados serviços de outros objetos. Para isto OOSA usa o modelo de acesso ao objeto (OAM). CYOOA tem uma representação semelhante.

A bibliografia disponível deixa a desejar na descrição dos métodos e das metodologias para se aplicar os métodos definidos. De uma forma geral, a bibliografia poderia ser enriquecida com mais exemplos para melhorar a metodologia. Da mesma forma, os aspectos comportamentais dos métodos poderiam ser melhor detalhados.

5.3 - Sugestões para Trabalhos Futuros

As recomendações para trabalhos futuros podem seguir três diferentes direções:

- Aperfeiçoar o metamodelo STL - Foi verificado um elevado número de conceitos em STL. O metamodelo precisa de um nível superior de modelagem com menos conceitos e maior capacidade de combinação ortogonal entre eles;
- Extender o estudo a outros métodos - O número de métodos saltou de 5 para 70 nos últimos cinco anos [RUMB 95]. Novos conceitos são utilizados principalmente na modelagem de sistemas com necessidades de tempo real;
- Implementação - Um trabalho de implementação de filtros STL poderia ser de extrema utilidade para manter a compatibilidade de uma especificação compartilhada entre ferramentas CASE diferentes. Por exemplo, ObjectMaker mantém um dicionário central de dados no formato proprietário. Não é possível compartilhar estas informações porque não conhecemos nem sua sintaxe nem tão pouco sua semântica. Por outro lado, se ObjectMaker entendesse STL, seria possível importar/exportar especificações entre ferramentas distintas.

Concretamente, um trabalho de implementação deverá reservar tempo para estudar formatos proprietários e escrever filtros para STL. O esforço maior será na construção das estruturas de dados que suportem os formatos proprietários dos repositórios de dados uma vez que as ferramentas *lex* e *yacc* poderão ser usadas para construção "automática" do filtro em si.

A. Exemplo do Controle do Portão

Este Apêndice tem como objetivo apresentar um problema que ilustrará a identificação dos conceitos dos métodos de Rumbaugh [RUMB 91], Coad/Yordon [COAD 91], Booch [BOOC 94], Shlaer e Mellor [SHLA88][SHLA 91], e Wirfs-Brock [WIRF 90], apresentados no Capítulo 4.

A.1 Descrição do problema

Estamos interessados em um sistema que controle o portão que fecha a estrada que corta as linhas de trem, impedindo a passagem dos carros (Figura A.1). São duas linhas em paralelo (linha1 e linha2) e os trens trafegam a uma velocidade constante. Em cada linha existem 5 sensores usados para detectar a presença de um trem, e 3 dispositivos que controlam a passagem dos trens, chamados aqui de controladores. Existem dois semáforos, sendo que o semáforo para estrada é ligado no momento em que o trem começar a passar no sensor 1. Quando o trem passar no sensor 2, o portão deve baixar e o sensor 4 deve ser lido indicando se há obstáculos na linha de trem, devendo assim, acionar o semáforo para o trem . A campainha deve disparar no momento da passagem do trem pelo sensor 3. Já o sensor 5 serve para indicar que um trem acabou de passar. Os controladores, por sua vez, lêem dos sensores das linhas 1 e 2 a informação da presença ou ausência de trens e controlam o momento em que o semáforo, a campainha e o portão devem ser acionados ou não.

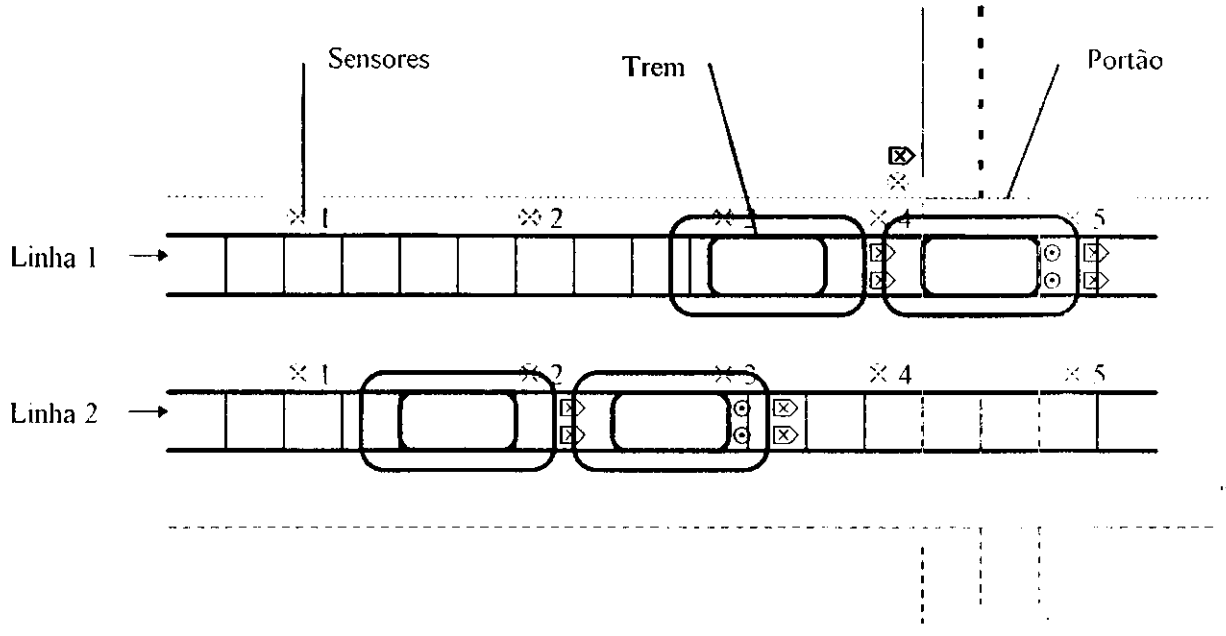

Figura A.1. ***Exemplo do controle do portão da linha do trem***

B. Representação dos Conceitos em STL

O objetivo deste apêndice é ilustrar como STL poderá representar os conceitos de *classe*, *herança* e *objeto* contribuindo para complementar a discussão do Capitulo 4.

Os conceitos dos métodos estudados podem ser representados através de uma combinação dos conceitos STL. Com os conceitos pertinentes a esta linguagem é possível representar um conceito de um dos métodos orientados a objetos usando mais de um forma de representação em STL. Neste sentido, existe pesquisa em andamento com o objetivo de construir um metamodelo que elimine esta forma de representação ambígua em STL.

Esta ambigüidade está relacionada com a necessidade de se descrever os pacotes de informação a serem transferidos a um nível de detalhe que permita a implementação. Resultados parciais no sentido de resolver esta questão pode ser visto em "Using New STL Constructs to Compare Object-Oriented Software Specification" [MEDE 96] que especifica uma proposta para representação dos principais conceitos dos métodos estudados. A Figura B.1 ilustra como os novos conceitos de *action*, *data*, *device e event* podem ser combinados para representar objeto, classe e o modelo comportamental dos métodos.

Note que um objeto pode ser o resultado de encapsulamento de dados e ações. As ações manipulam os dados. Dados que precisam de interface com o mundo externo utilizam uma interface chamada *device*. Já os eventos estão associados aos dados (através da interface *device*) e às ações. O mecanismo de envio e recebimentos de mensagens para objetos dos métodos pode ser mapeado em função de eventos.

Usou-se o relacionamento *contain* para definir a herança entre classes e o relacionamento *related to* para possibilitar relacionamento entre classes. A cardinalidade destes relacionamentos vai permitir o mapeamento de herança simples e múltipla, agregação e associação.

Finalizando, ao se instanciar uma classe é feita a associação entre as informações sobre os conceitos STL básicos e os conceitos dos cinco métodos estudados.

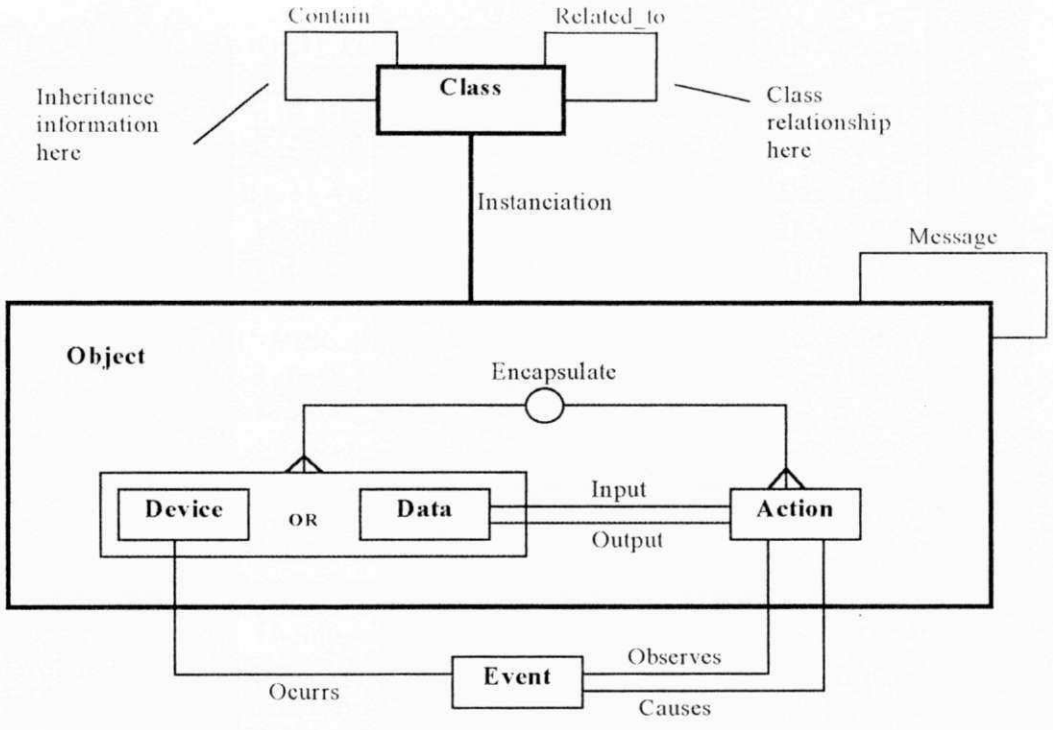

Figure B. 1: Definição dos conceitos orientados a objetos em STL

Referências Bibliográficas

[BOOC 86] BOOCH, Grady. Object-Oriented Development. IEEE Transactions on Software Engineering. v.12, n.2, Feb. 1986.

[BOOC 94] BOOCH, Grady. Object-Oriented Design with Applications. 2.ed. Redwood City: The Benjamin/Cummings Publishing Company, Inc., 1994. 376p.

[BOOC 95a] BOOCH, Grady, RUMBAUGH, J. Introduction to the Unified Method: Unifying the Booch & OMT Methods. SIGPLAN Tutorial 30, OOPSLA '95, Austing, Oct. 1995. 37p.

[BIND 95] BINDER, Robert V. Testing Object-Oriented Systems: A Status Report. Software Engineering Technology. p. 16-19. Apri. 1995.

[COAD 91] COAD, P., YOURDON, E. Object-Oriented Analysis. 2.ed. Englewood Cliffs: Yourdon Press, 1991. 233p.

[CONG 94] CONGER, Sue. The New Software Engineering. Ed. International Thompson Publishing, 1994;

[GANE 90] GANE, Chris. CASE O Relatório Gane. 1.ed. São Cristovão: Livros Técnicos e Científicos Editora Ltda, 1990. 260p.

[GRAH 94] GRAHAM, Ian. Object Oriented Methods. 2.ed. London: Addison-Wesley Publishing Company, 1994, 473p.

[JACO 93] JACOBSON, Ivar. Is Object Technology Software's Industrial Plataform?. IEEE Software. Jan.1993.

[IEEE 92] IEEE Trial-Use Standard Reference Model for Computing System Tool Interconnections, IEEE Std 1175, Pub. August 17, 1992.

[JACK 96] JACKSON, Daniel. Requirements Need Form, Maybe Fromality. IEEE Software. v.13, n.2, p.21-22. Mar.1996.

[JONE 91] JONES, Capers. Produtividade no Desenvolvimento de Software. São Paulo: MAKRON Books, 1991. 370p.

[JORG 94] JORGENSEN, Paul C., ERICSON, Carl - *Object-Oriented Integration Testing* - Communication of the ACM, v.7, n. 9, p.30-38, Sep.1994.

[JORG 95] JORGENSEN, Paul C. Software Testing A Craftsman's Approach. Boca Raton: CRC Press, 1995. 254p.

[KOZA 93] KOZACZYNSKI, Wojtk, KUNTZMANN-COMBELLS, Annie - What it Takes to Make OO Work - IEEE Software, Jan.1993.

[LAWR 96] LAWRENCE, Brian; Do You Really Need Formal Requirements?, IEEE Software, v.13, n.2, p.20-22, Mar.1996.

[LOY 90] LOY, Patrick H. A Comparison of Object-Oriented and Structured Development Methods. ACM Sigsoft, Software Engineering Notes, v.15, n.1, p.44-48, Jan.1990.

[MART 95] MARTIN, J; ODELL, J.J. Object-Oriented Methods A Foundation. Englewood Cliffs: P T R Prentice Hall, 1995. 412p.

[MART 95] MARTINS, Luiz M. F. ; MOURA, J. Antão B.; MEDEIROS, Álvaro F. C.; RCYCLE: Um Molde para o Processo de Produção, Disponibilização e Evolução de Software- IX Simpósio Brasileiro de Engenharia de Software. Recife. Ago.95.

[MEDE 94] MEDEIROS, A. F. C.; SAÚVE, J. P.; MOURA, J. A. B.; NICOLETI, P. S. Aumentando a Produtividade e Qualidade em Sistemas Abertos: Guia Avançado para Ambientes *UNIX*. São Paulo: Makron Books, 1994. 394p.

[MEDE 96] MEDEIROS, A.F.C.; JORGENSEN, P. C.; MOURA, A. B.; MEDEIROS, I..M.S. New STL Constructs To Compare Object-Oriented Development Software Specification Methodologies. International Conference on Information Systems, Analisis and Sysntesis. Orlando -USA. Jul.1996.

[NAUR 60] NAUR, P. Revisited Report on the Algorithmic Language Algol 60. v.6, n.1, p.1-17,Jan.1960.

[RUMB 91] RUMBAUGH, J. BLAMA, M., PREMERLANI, W., EDDY, F., LORENSEN, W. Object-Oriented Modeling and Design. Englewood Cliffs: Prentice Hall, 1991. 500p.

[RUMB 95] RUMBAUGH, J, GRADY, B.Unified Method. Rational Software Corporation, Jul.1995. 43p.

[SHLA 88] SHLAER, S., MELLOR, S. Object-Oriented System Analysis: Modeling the World in Data. Englewood Cliffs: Yourdon Press, 1988. 144p.

[SHLA 91] SHLAER, S., MELLOR, S. Object Life Cycle. Englewood Cliffs: Yourdon Press, 1991. 251p.

[SNYD 93] SNYDER, Alan . The Essence of Objects: Concepts and Terms. IEEE Software, Jan.1993.

[SOMM 89] SOMMERVILE, Ian. Software Engineering. 3.ed. New York: Addison -Wesley. 1989.

[SOMM 96] SOMMERVILE, Ian. Software Engineering. 15.ed. New York: Addison - Wesley, 1996. 742p.

[TOPP 94] TOPPER, Adrew, OUELLETTE, Daniel; JORGENSEN, Paul. Structured Methods: Merging Models Techniques and Case. McGraw-Hill, 1994

[WIRF 90] WIRFS-BROCK, R., WILKERSON, B., WIENER, L. Designing Object-Oriented Software. Englewood Cliffs: Prentice Hall, 1990. 341p.

[WOLF 89] WOLF, Wayne. A Practical Comparison of Two Object-Oriented Languages. IEEE Software, p.61, Set.1989.

[YORD 90] YORDON, E. Análise Estruturada Moderna. Editora Campus Ltda, 1990.