# O Modelo de Integração de Ferramentas da Estação TABA

Guilherme Horta Travassos

**TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.**

**Aprovada por:**

**Profª. Ana Regina Cavalcanti da Rocha, D.Sc.**
**(Presidente)**

**Prof. Jano Moreira de Souza, Ph.D.**

**Profª. Claudia Maria Lima Werner, D.Sc.**

**Profª. Regina Célia de Souza Pereira, D.Sc.**

**Prof. Caetano Traina Júnior, D.Sc.**

**RIO DE JANEIRO, RJ - BRASIL**
**MARÇO DE 1994**

TRAVASSOS, GUILHERME HORTA

O Modelo de Integração de Ferramentas da Estação TABA [Rio de Janeiro] 1994.

VII, 230 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 1994)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Engenharia de Software

I. COPPE/UFRJ II. Título(série)

*Para Liliam e Rodrigo*

# Agradecimentos

À Professora Ana Regina Cavalcanti da Rocha, pela sua acolhida em seu projeto de pesquisa, sua correta orientação e por seus estímulos constantes. Não teria sido possível realizar este trabalho sem seu apoio.

À Liliam, pelo seu amor, compreensão e estímulos constantes, na esperança de que todo este esforço não tenha sido em vão.

Ao Rodrigo que, com sua inocência e pureza, mesmo não conseguindo entender minhas ausências, me alegrava nos poucos momentos em que conseguimos estar juntos.

Aos meus pais, Caio e Miriam, e ao meu irmão Gustavo, pelo apoio e estímulo constante.

À Dra. Maria H. Penedo pelo estímulo e o material bibliográfico enviado que permitiu o enriquecimento deste trabalho.

Ao Prof. Jano Moreira de Souza pelas idéias geniais, o apoio constante ao longo destes anos e por participar de minha banca de tese.

À Prof. Claudia Maria Lima Werner pelo apoio e o estímulo no momento certo, e por me dar a honra de participar de minha banca de tese.

À Prof. Regina Célia Pereira por sua paciência e compreensão e por participar da minha banca de tese.

Ao Prof. Caetano Traina Júnior por participar da minha banca de tese.

Aos colegas de batalha Xexeo, Monte, Trotta, Marta, Washington, Eliseu, Vera, Blaschek e Lygia pelo apoio e a compreensão nos momentos de insatisfação e nervosismo.

Ao Edu, Julinho, Adilson, Claudia, Ana Paula, Denise e Ari pelo apoio, sempre na hora certa, e pelos momentos descontraídos.

Aos professores e amigos do Programa de Sistemas da COPPE que sempre tiveram paciência para aguentar meus nervosismos e me apoiavam no momento necessário.

Aos amigos de Juiz de Fora por aceitarem as justificativas para as ausências constantes e me mostrarem que a amizade é eterna.

A Ruth, Márcia, Júnior e Daniel que com seu apoio e carinho permitiram a solidão necessária nos momentos certos.

Aos amigos do Rio de Janeiro, Carlos, José Augusto e Teresa, Luiz Mondego e Bernadete que sempre estiveram por perto nos momentos de angústia e solidão.

Ao CNPq e a CAPES pelo apoio financeiro.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

# O Modelo de Integração de Ferramentas da Estação TABA

Guilherme Horta Travassos

Março de 1994

Orientador: Profª. Ana Regina Cavalcanti da Rocha
Programa: Engenharia de Sistemas e Computação

O projeto TABA, em desenvolvimento no Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ, tem como objetivo a construção de uma Estação de Trabalho (Estação TABA) configurável para Engenharia de Software. Esta Estação permite, dentre outras atividades, a instanciação de Ambientes de Desenvolvimento, e a construção, quando for o caso, de ferramentas (CASE) integradas nestes ambientes. Este trabalho apresenta o modelo de integração de ferramentas utilizado na Estação TABA, descrevendo a estrutura de construção do meta-ambiente TABA e a forma de organização dos ambientes instanciados TABA e suas ferramentas integradas.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Doctor of Sciences (D.Sc.).

# The TABA Workstation's Tools Integration Model

Guilherme Horta Travassos

March, 1994

Thesis Supervisor: Ana Regina Cavalcanti da Rocha
Department: Systems and Computing Engineering

The TABA Project under development at Systems and Computing Engineering Program of the COPPE/UFRJ has the objective of building a configurable workstation (The TABA Workstation) for software engineering.

This workstation allows to create instances of Development Environments and to build integrated CASE tools. This work presents the Tools Integration Model to be used in the TABA Workstation, describing the architecture of the TABA meta-environment and the organization of the TABA environment instances and corresponding tools.

# ÍNDICE

Capítulo IV

Ambientes de Desenvolvimento de Software e suas propostas de Integração de Ferramentas

# Capítulo I

# Introdução

## I.1 Ambientes de Desenvolvimento de Software e Integração de Ferramentas

O conceito de Ambientes para Engenharia de Software surgiu ao se buscar combinar técnicas, métodos e ferramentas com o objetivo de prover um meio através do qual o Engenheiro de Software pudesse ter um apoio ao construir produtos de software [Penedo88].

Inicialmente estes ambientes se concentravam, apenas, nas fases de codificação e projeto detalhado. No final dos anos 70 houve uma evolução no entendimento do processo de desenvolvimento de software. O foco se moveu da fase de codificação para as fases de especificação e projeto.

Nos anos 80 viu-se a introdução dos ambientes automatizados e de ferramentas que visavam tornar mais prático e efetivo o uso dos métodos para desenvolvimento de software. Esta tecnologia, conhecida como CASE ("Computer Aided Software Engineering") tornou possível que os desenvolvedores de software desenvolvessem e validassem seus sistemas utilizando de forma automatizada os métodos antes de uso manual.

Entretanto ferramentas isoladas podem oferecer apenas soluções parciais. Surge, então, o conceito de Ambientes de Desenvolvimento de Software (ADS) que, aliado à necessidade de atender a todas as etapas do processo de desenvolvimento, provocou o surgimento de diferentes ferramentas, que deveriam operar juntas num ADS.

Esta necessidade de se ter várias ferramentas operando juntas tem provocado uma série de pesquisas na área de integração de ferramentas. É neste contexto que se situa este trabalho.

## I.2 Contexto do Trabalho: O Projeto TABA

Este trabalho insere-se no contexto do Projeto TABA, em desenvolvimento no Programa de Engenharia de Sistemas da COPPE.

O Projeto TABA tem como objetivo a construção de uma Estação de Trabalho para Desenvolvimento de Software Configurável para atender às características de diferentes domínios de aplicação. A motivação para realização do projeto TABA está na constatação de que domínios de aplicação diferentes têm características diferentes e que estas devem incidir nos ambientes de desenvolvimento através dos quais os engenheiros de software desenvolverão as aplicações.

Para atingir seu objetivo a Estação TABA contêm um Meta-Ambiente que tem as seguintes funções:

a) especificar o ambiente de desenvolvimento de software mais adequado a um projeto num determinado domínio de aplicação;

b) instanciar e tornar operacional o ambiente especificado.

A especificação de ambientes é feita através de XAMÃ [Aguiar92], [Massolar93], um assistente baseado em conhecimento que apoia o engenheiro de software nesta tarefa.

Este trabalho tem como objetivo resolver o problema da instanciação e de tornar operacional o ambiente especificado pelo XAMÃ. Esta instanciação implica na integração de ferramentas internas (desenvolvidas segundo a filosofia da Estação) ou externas (desenvolvidas em outros contextos).

Outra questão a ser resolvida, através deste trabalho, é a construção de novas ferramentas necessárias aos ambientes especificados.

## I.3 Organização da Tese

Este trabalho, além desta Introdução, contêm mais seis capítulos e dois anexos.

No capítulo II - Ambientes de Desenvolvimento de Software -- são descritos aspectos gerais relativos a ambientes e ferramentas CASE.

O capítulo III - A Questão da Integração de Ferramentas em Ambientes de Desenvolvimento de Software - discute os diversos enfoques presentes na literatura para tratamento do problema de integração em ambientes.

O capítulo IV - Ambientes de Desenvolvimento de Software e suas propostas de Integração de Ferramentas - descreve como a questão da integração de ferramentas é tratada em ambientes de desenvolvimento de software atualmente disponíveis.

No capítulo V - Integração de Ferramentas na Estação TABA - descreve-se a solução para construção e integração de ferramentas na Estação TABA.

O capítulo VI - O Protótipo da Estação TABA - contém a descrição da implementação da Estação com base na solução descrita no capítulo V.

O capítulo VII - Conclusões e Perspectivas Futuras - contém as conclusões do trabalho evidenciando suas contribuições e futuras pesquisas que devem se derivar deste trabalho.

O anexo A contém os diagramas do modelo de análise orientada a objetos para a Estação TABA.

O anexo B contém exemplos do uso de uma Linguagem para Descrição e Manipulação de Métodos, definida neste trabalho e aplicada ao contexto da Estação TABA.

# Capítulo II

## Ambientes de Desenvolvimento de Software

## II.1 Considerações Iniciais

O conceito de Ambientes para Engenharia de Software surgiu nos anos 70 buscando combinar técnicas, métodos e ferramentas com o objetivo de prover um meio através do qual o Engenheiro de Software pudesse ter um apoio ao construir produtos de software [Penedo88]. Esta forma de tratar o processo de desenvolvimento, como se entende hoje, envolve o suporte a atividades individuais e ao trabalho em grupo, ao gerenciamento do projeto, ao aumento da qualidade geral dos produtos, ao aumento da produtividade, permitindo ao Engenheiro de Software acompanhar o trabalho e medir, através de informações obtidas ao longo do desenvolvimento, a evolução dos trabalhos.

Inicialmente estes ambientes se concentravam, apenas, nas fases de codificação e projeto detalhado, não possuindo suporte às fases iniciais do ciclo de desenvolvimento [Mitze81], [Donahue81], [Habermann80], [Rothkind75], [Dolotta78], [Goldberg83]. Esta característica dos ambientes pode ser vista como uma consequência natural do processo de desenvolvimento então vigente. Não existia, neste momento, a preocupação em atender às fases iniciais do processo de desenvolvimento, muito embora já estivesse presente a preocupação com a idéia de ciclo de vida e existisse clareza quanto aos objetivos , atividades e questões a serem solucionadas em cada uma das etapas do ciclo.

No final dos anos 70 viu-se uma evolução no entendimento do processo de desenvolvimento de software. O foco se moveu da fase de codificação para as fases de especificação e projeto e começou-se a entender que desenvolver software era uma ciência e não uma arte e que carecia de processos mais sistemáticos e disciplinados de trabalho [Smith90]. Surge, então, um conjunto de métodos de

apoio às atividades de análise e projeto: Análise e Projeto Estruturado [Gane82], [DeMarco79], [Yourdon78], SADT [Ross77], PSL/PSA [Teichroew77], SREM [Alford85 ], entre outros.

Com a popularização dos computadores de uso pessoal, na metade dos anos 80, viu-se a introdução dos ambientes automatizados e de ferramentas que visavam tornar mais prático e efetivo o uso dos métodos para o desenvolvimento de software. Esta tecnologia, conhecida como CASE ("Computer Aided Software Engineering") tornou possível, principalmente hoje com as estações de trabalho, que os desenvolvedores de software desenvolvam e validem seus sistemas utilizando de forma automatizada os métodos para desenvolvimento de software, propostos para o uso manual nos anos 70 e 80 [Chikovsky88], [Mosley92], [Vessey92].

Embora estas ferramentas, em geral, pudessem resolver o problema específico para o qual foram construídas [Mosley92], [Vessey92], elas não se propunham de início, na maioria dos casos, a tratar o processo de desenvolvimento como um todo. Atualmente estamos num estágio onde, após ter-se verificado que ferramentas isoladas podem oferecer apenas soluções parciais, o que se deseja é utilizar ferramentas de apoio ao desenvolvimento de software ao longo de todo o seu processo de desenvolvimento [Smith90]. Surge então, o conceito de Ambientes de Desenvolvimento de Software (ADS) como o entendemos hoje:

> "*um sistema que fornece suporte a qualquer atividade executada por engenheiros de software* " [Saito89]

> "- *um ciclo de vida que define as etapas do processo de desenvolvimento e as atividades a serem realizadas em cada etapa;*
> - *um conjunto de métodos, usados para organizar o pensamento e o trabalho do usuário ao longo do processo de desenvolvimento, e;*
> - *um conjunto de ferramentas que automatizam os métodos*" (Projeto TABA, (Rocha87b))

Nomes equivalentes para Ambientes de Desenvolvimento de Software são encontrados na literatura: IPSE ("Integrated Project Support Environment"), I-CASE ("Integrated Computer-Aided Software Engineering"), SDE ("Software Development Environment"), SEE ("Software Engineering Environment"), ISEE ("Integrated Software Engineering Environment") e ISF ("Information System Factory"). [Ecma90]

O desejo de atender a todas as etapas do desenvolvimento levou à necessidade de se terem diferentes ferramentas, cada uma delas construída para atender às especificidades de uma atividade ou etapa do processo, operando juntas para a construção do produto. Uma medida da praticidade e maturidade de uma dada ferramenta será, então, sua capacidade para ser integrada a outras ferramentas [Smith90].

Integrar ferramentas é uma tarefa complexa e apresenta uma série de problemas. Os esforços por resolvê-los estão longe de uma resposta final [Normam91], [Meyers91], [Norman92]. Por ser o objeto principal deste trabalho, a questão da integração de ferramentas será abordada, detalhadamente, nos capítulos III, IV e V.

Nas próximas seções descrevemos, brevemente, a evolução histórica das ferramentas CASE e de ADSs (seção II.2), os requisitos para ambientes de desenvolvimento de software (seção II.3), o processo de construção de ambientes (seção II.4) , e as estruturas de ADSs propostas na literatura (seção II.5)

## II.2 Evolução Histórica de Ferramentas e Ambientes

Em [Brown92c], [Brown93], [Ghezzi91] podemos encontrar uma descrição da evolução de ferramentas CASE e ambientes de desenvolvimento de software. A

partir destas referências, pode-se estabelecer uma classificação considerando-se seis estágios:

1º) ferramentas isoladas;
2º) grupos de ferramentas;
3º) ambientes integrados primitivos;
4º) ambientes influenciados pela linguagem ADA;
5º) ferramentas CASE, e;
6º) ambientes abertos e fechados.

**Ferramentas isoladas** são quase tão antigas quanto a própria computação. Estas ferramentas são representadas, por exemplo, por compiladores, montadores, ligadores e depuradores e fornecem suporte apenas à fase de programação. Estas ferramentas evoluiram para tratar outros aspectos do processo de desenvolvimento e, a esta evolução, seguiu-se o uso de **grupos de ferramentas**. O trabalho com **ambientes integrados primitivos** foi uma tentativa de se aliar diferentes técnicas ao enfoque de desenvolvimento, tais como Inteligência Artificial e Orientação a Objetos. Embora a existência de ADA possa ter ocorrido ao mesmo tempo em que se construiam os ambientes primitivos, é inegável sua influência nos sistemas atuais. **Ambientes fechados e abertos** fazem parte de uma discussão recente e a tendência a se desenvolver sistemas abertos reflete as características de projetos recentes [DEC92], [Shu93] e as aspirações dos desenvolvedores de ADSs [Meyers91], [Brown93].

Penedo [Penedo93d] trata a evolução dos ADSs considerando três estágios. Num primeiro estágio, referente à década de 70, havia a preocupação com a definição de métodos de desenvolvimento de software. As ferramentas suportavam métodos particulares e funcionavam de forma isolada.

Num segundo estágio (década de 80) surge a preocupação com a integração de ferramentas. Neste ponto começaram os primeiros estudos para se definirem estruturas ("frameworks"[1]) de ADSs. Esquemas de interface com o

[1] Um "framework" é um projeto de alto nível, ou arquitetura de aplicação, e consiste em um conjunto de classes que são especificamente projetadas para serem refinadas, ou especializadas, e utilizadas como um grupo [Wirfs-Brock90].

usuário passaram a ser padronizadas, possibilitando o desenvolvimento de ferramentas consistentes a nível de interface. Atividades maiores no ciclo de vida passaram a ser consideradas. Este conjunto de características levaram às primeiras definições de mecanismos de integração e, junto a isto, começou a preocupação com a distribuição do ADS.

No terceiro estágio, referente a década de 90 onde nos encontramos, o enfoque é para ADSs orientados ao processo[2] de desenvolvimento. A preocupação é com o modelo de processo de desenvolvimento e que atividades devem ser desenvolvidas ao longo deste processo, bem como com a forma em que estas atividades são gerenciadas e controladas dentro da organização . Padrões de interface são fortemente utilizados, baseados nos modelos definidos no estágio anterior. Repositórios integrados, embora distribuídos, passam a ser utilizados nos ADSs.

A partir destes estágios Penedo classifica ADSs da seguinte forma:

- **ADS de 1ª Geração: Sistemas de Ferramentas Isoladas** que correspondem às ferramentas que foram utilizadas dos anos 60 aos anos 80 e não satisfazem às necessidades atuais de portatilidade[3] e integração. Uma relação das ferramentas CASE correspondentes a esta etapa pode ser encontrada em [Travassos90] e [Dart87].

- **ADS de 2ª Geração: Ambientes baseados em Estrutura** ("framework") que começaram a aparecer nos anos 80 e início de 90 e que foram projetados para suportar a integração, portatilidade e interoperabilidade[4] de ferramentas a partir da utilização de integração de dados, através de esquemas de banco de dados. Podemos entender uma estrutura ("framework") como sendo "os componentes incorporados ao ADS e que

---

[2] Processos podem ser definidos como sendo "o conjunto de atividades, métodos e práticas que direcionam pessoas(com suas ferramentas de software) na produção do software" [Penedo93c]

[3] Portatilidade é a capacidade de uma ferramenta poder ser utilizada em diferentes ambientes e arquiteturas.

[4] Interoperabilidade é a capacidade de ferramentas poderem trabalhar em conjunto e harmonicamente.

são tipicamente parte de algum ambiente para um projeto específico e provêm a infra-estrutura sobre a qual as capacidades funcionais do ambiente, para o suporte das atividades do ciclo de vida, são construídas" [Penedo93d]. Uma outra definição para esta estrutura é dada por Zelkowitz [Zelkowitz93] que diz que "uma estrutura fornece um conjunto de serviços comuns necessários aos programas de suporte à aplicação, escritos para o domínio do ambiente". Trabalhos nestes sistemas se deslocam para tratar questões relativas ao processo e reuso, que não foram geralmente os objetivos de projeto no desenvolvimento convencional. Em [Penedo92], [Brown92c] e [Moura92] podem ser encontradas descrições de ambientes que adotam esta organização.

- **ADS de 3ª Geração: Ambientes Orientados a Processo,** que contêm conhecimento da organização, projeto e processos do usuário. A tendência é que estes ambientes possuam uma arquitetura aberta, isto é, que permitam que ferramentas diversas sejam incorporadas e integradas ao ambiente. Prevê-se que estejam disponíveis a partir de 1995.

## II.3 Requisitos de Ambientes de Desenvolvimento de Software

A evolução das ferramentas CASE e dos ambientes de desenvolvimento de software foi mostrando que não é qualquer ferramenta ou ambiente que é realmente útil ao usuário desenvolvedor de software.

Para ser de real utilidade, um ADS deve possuir as seguintes características:

- **ser customizável**: o ambiente deve permitir adaptações que o tornem mais adequado a um determinado projeto e às preferências de seus usuários [Rocha87b], [Rocha90], [Penedo93d], [Chikofsky88a], [Sorenson88], [Chikofsky88b], [Vessey91], [Norman91], [Cybulski92], [McGrath93];

- **oferecer suporte à construção de ferramentas** [Rocha87b], [Rocha90], [Sorenson88], [Norman91], [Moura92], [Travassos92a].

- **ser independente de hardware**: o ambiente deve poder ser utilizado em diferentes plataformas de hardware [Penedo93d], [Chikofsky88].

- **reutilizabilidade de componentes internos**: os componentes que determinam a estrutura do ambiente devem poder ser reaproveitados em outros trabalhos e atividades [Penedo93d].

- **possuir uma base de dados central**: o ambiente deve manter um repositório com as informações relevantes ao desenvolvimento do produto, que possa ser utilizado por todas as ferramentas que compõem o ambiente [Penedo93d], [Pressman92], [Takahashi90],[Norman91], [McGrath93].

- **fornecer apoio para controle de versões e gerenciamento total de configuração** [Forte89].

- **ser de uso amigável** [Penedo93d] , [Brown92c], [Miyoshi93].

- **possuir suporte à prototipagem**: o ambiente deve permitir, ao longo do processo de desenvolvimento e onde for necessário, que o usuário possa desenvolver protótipos, de forma a validar idéias e verificar o entendimento do problema [Puncello88], [Penedo93d].

- **poder ser amplamente aplicado**: o ambiente deve poder ser utilizado no desenvolvimento de um vasto domínio de aplicações e, ainda, suportar o desenvolvimento de software tanto em pequena como em larga escala [Rocha87b], [Rocha90], [Penedo93d], [Saito89], [Norman91].

- **oferecer suporte a diferentes tipos de usuários**: o ambiente deve ser capaz de diferenciar entre os vários tipos de usuários (por exemplo, distinguir usuários

novatos de experientes) e adaptar-se de forma a atender as especificidades de cada um [Falcão92], [Penedo93d], [Vessey92], [Brown92c].

- **oferecer suporte a todas as atividades do processo de desenvolvimento**: o ambiente deve apoiar no desenvolvimento do produto ao longo de todo o processo de desenvolvimento, e não apenas em etapas específicas [Rocha87b], [Rocha90], [Penedo93d], [Ghezzi91], [Norman91].

- **oferecer suporte ao trabalho cooperativo**: o ambiente deve suportar tarefas de coordenação e cooperação entre os membros da equipe de desenvolvimento [Takahashi90], [Saito89], [Miyoshi93], [Ghezzi91], [Norman91], [Travassos93b], [McGrath93].

- **possuir uma interface gráfica consistente**: a interação do usuário com o ambiente deve ser feita de forma consistente, tendo-se uma apresentação uniforme e utilizar-se de recursos gráficos, preferencialmente, os mais avançados possíveis [Takahashi90], [Brown92c], [Thomas92], [Zelkowitz93].

- **possuir uma interface interna consistente** de forma a permitir que uma ferramenta possa se comunicar perfeitamente com outra, possibilitando a passagem de informações entre elas, e permitindo também que ferramentas possam ser facilmente introduzidas e/ou substituídas no ambiente [Zelkowitz93], [Pressman92], [Thomas92].

- **oferecer suporte para as atividades comuns** do processo de desenvolvimento, que envolvem, por exemplo, a documentação [Moura92], [Saito89], [Cybulsky92].

- **oferecer suporte para a avaliação da qualidade** ao longo do processo de desenvolvimento [Rocha87b], [Cardoso90], [Karimi88], [Moura92], [Rocha92], [Norman91].
- **oferecer suporte à gerência do projeto**, apoiando o acompanhamento do projeto, verificando se as metas foram atingidas, os objetivos estão sendo observados, etc. [Rocha87b] , [Moura92], [Sardinha93].

- **oferecer suporte baseado em conhecimento**, sobre diferentes domínios de aplicação [Puncello88], [Moura92].

- **suportar, e não impor, um nível de formalidade no desenvolvimento**, permitindo que o nível de formalidade desejado seja decidido pelo usuário do ambiente [Ghezzi91].

- **apoiar a evolução do produto**, permitindo que o usuário, à medida que vai avançando no desenvolvimento de uma atividade, incorpore livremente mais detalhes, e não lhe impondo limitações [Ghezzi91], [Miyoshi93].

- **ser extensível**, isto é, o ambiente deve ser aberto, permitindo que novas ferramentas lhe sejam incorporadas na medida em que forem necessárias. Assim sendo, o ambiente não deve possuir uma coleção fixa de ferramentas e sim permitir um enriquecimento progressivo a partir da inclusão de novas ferramentas ao longo do tempo [Rocha87b], [Rocha90], [Ghezzi91], [Moura92], [Penedo93d].

- **fornecer treinamento para o usuário**: o ambiente deve fornecer apoio ao usuário no sentido de auxiliá-lo no uso do ambiente como um todo e das ferramentas que o compõem, sob a forma de tutores e/ou assistentes baseados em conhecimento [Ghezzi91], [Falcão92], [Moura92], [Penedo88], [Puncello88], [Karimi88], [Brown92c].

- **ser robusto**, possuindo mecanismos que possibilitem ao usuário a recuperação de erros e a manutenção da consistência de suas informações ao longo do desenvolvimento [Brown92c];

- **não inibir a criatividade**, permitindo ao usuário utilizar sua criatividade e inteligência no processo de desenvolvimento, não inibindo soluções e não limitando as formas de trabalho do usuário [Brown92c], [Falcão92];

- **possuir utilitários para quantificação de métricas**: permitindo o acompanhamento do desenvolvimento, possibilitando com isto a verificação de metas, objetivos e a medição da produtividade e da qualidade [Wasserman81], [Brown92c], [Karimi88], [Norman91], [McGrath93].

## II.4 Processo de Construção de Ambientes de Desenvolvimento de Software

Uma questão importante a ser considerada é o processo de construção de ambientes. Brown [Brown92c] considera que esta questão pode ser tratada através das seguintes atividades, cujo conjunto pode ser visto como um modelo de ciclo de vida para ADSs (figura II.1):

1. **Estruturação do ADS**: consiste na definição e construção dos serviços que serão o núcleo do ADS. Estes serviços podem ser, por exemplo, facilidades de armazenamento, coordenação da interface com o usuário, serviço de mensagens para interconexão de ferramentas. Neste ponto, embora já possuindo as funcionalidades básicas, o ADS ainda não suporta o desenvolvimento de uma aplicação pois não está povoado, isto é, não contém ferramentas.

2. **População do ADS**: consiste na introdução de ferramentas ao ADS. Estas ferramentas podem ter sido especialmente desenvolvidas para o ambiente e, portanto, para utilizar os serviços definidos na estrutura do ADS, ou podem ser, também, ferramentas já existentes que necessitem ser modificadas para utilizar os serviços, ou, até mesmo, encapsuladas de forma a poderem compor o ADS.

3. **Adaptação do ADS**: consiste na adaptação do ADS de forma que os serviços e ferramentas existentes sejam adequados às restrições de desenvolvimento, procedimentos e estilo de uma determinada organização e/ou projeto.

4. **Uso do ADS**: consiste na utilização do ADS para o desenvolvimento de uma aplicação particular.

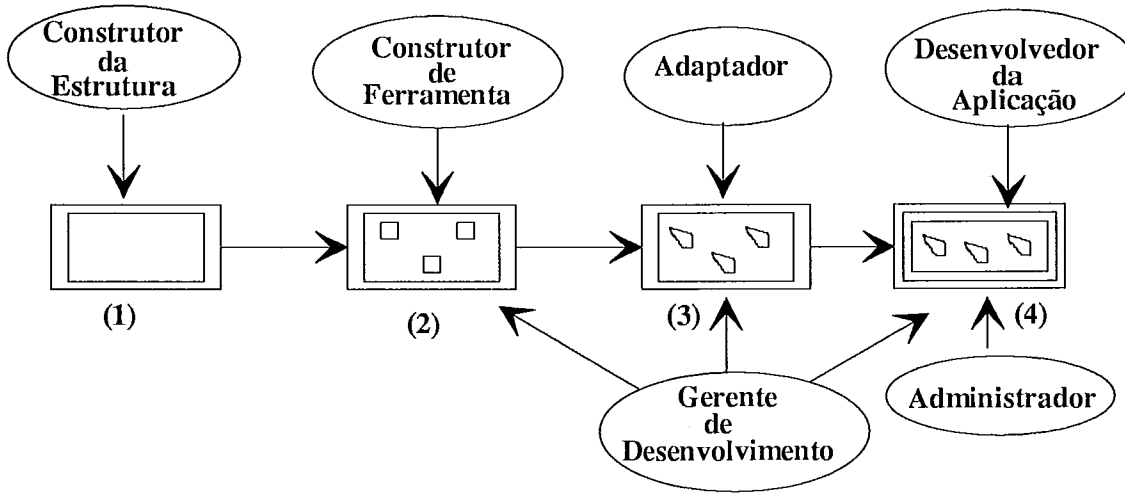



Figura II.1 - Um Modelo de Ciclo de Vida para ADSs [Brown92c]

Neste contexto são identificados, para cada atividade do ciclo, os usuários do ADS. Cada um destes usuários trata e percebe o ADS de forma distinta. Estes usuários, conforme a figura II.1, são:

- **o construtor da estrutura do ADS** responsável pela especificação, projeto e/ou implementação da estrutura do ADS e seus serviços associados. Para ele

um ADS pode ser definido como " um conjunto de serviços comuns que permitem que, ferramentas desenvolvidas ou em desenvolvimento, sejam mais facilmente integradas do que elas seriam num sistema operacional convencional, com o principal objetivo de suportar o desenvolvimento de software";

- **o desenvolvedor de ferramentas**, responsável pelo desenvolvimento de ferramentas específicas para um determinado ADS, ou então, ferramentas genéricas para classes de ambientes. Sob este ponto de vista um ADS é definido como "um conjunto de serviços comuns que fazem a produção e integração de ferramentas de software mais fáceis e frutíferas para o benefício dos usuários finais que irão desenvolver software utilizando estas ferramentas";

- **o construtor de ambientes**, responsável por preparar um ADS, incorporando as ferramentas necessárias para um desenvolvimento em particular. Para um construtor de ambientes, um ADS é "um conjunto de serviços comuns e um conjunto de ferramentas que necessitam adaptação, alinhamento e extensão para incorporarem vários conjuntos específicos de processos de desenvolvimento de software de forma a serem utilizadas pelos desenvolvedores de software dentro de uma organização em particular";

- **o desenvolvedor de aplicações** que usa o ADS na construção de um determinado produto de software. Segundo estes usuários um ADS é "um conjunto de serviços comuns que possibilita o desenvolvedor projetar, desenvolver e manter funcionando, sistemas de software manuteníveis que atendam aos requisitos, dentro das restrições impostas pelo negócio da organização do usuário que contratou o software";

- **o gerente de desenvolvimento**, responsável pela decisão de quais ADSs e ferramentas devem ser utilizados, e pela definição dos procedimentos e estilos que devem ser suportados num ADS. Para o gerente de desenvolvimento um ADS é "um conjunto de serviços comuns que facilita desenvolvedores na produção de software com alta qualidade e produtividade, e que permite que novas ferramentas e serviços lhe sejam incorporados de forma que o ambiente

possa evoluir para suportar novos métodos e técnicas de desenvolvimento de software";

- **o administrador do ADS**, responsável por controlar e gerenciar as operações diárias do ADS, recomendando possíveis alterações e evoluções no ADS. Neste contexto um ADS é "um conjunto de serviços e ferramentas de suporte ao desenvolvimento de software que deve ser mantido, estendido, sintonizado e continuamente monitorado para assegurar níveis aceitáveis de segurança, disponibilidade e desempenho requisitados pelos desenvolvedores de software que utilizem o ambiente";

- **o pesquisador de ADS**, responsável pelas perspectivas futuras de desenvolvimento de ADSs, investigando novas arquiteturas que tratem melhor as necessidades de diferentes classes de usuários do ADS. Para o pesquisador um ADS é "uma facilidade para o desenvolvimento de software que satisfaz as necessidades atuais e projetadas das diferentes classes de potenciais usuários do ADS".

Outro enfoque a se considerar, ao tratar da construção de ADSs, são os meta-sistemas, cujo objetivo é a construção automática de partes de um ADS particular [Sorenson88], [Borras88]. Estes sistemas, através de um nível meta, são capazes de gerar (configurar) ambientes específicos que atendam a métodos de desenvolvimento e linguagens particulares.

Neste contexto podemos situar, também, a proposta do Projeto TABA [Rocha87], [Rocha90], cujo objetivo é a construção de uma Estação de Trabalho Configurável para Desenvolvimento de Software. O Projeto TABA é descrito com mais detalhes, no capítulo V.

## II.5 Estruturas de Ambientes de Desenvolvimento de Software

Encontramos na literatura [Penedo88], [Penedo92], [Penedo93b], [Ecma90], [Brown92c] descrições das estruturas, normalmente utilizadas para

descrever Ambientes de Desenvolvimento de Software. O que estas estruturas pretendem mostrar diz respeito aos componentes existentes nos Ambientes de Desenvolvimento de Software necessários para atender aos requisitos funcionais estabelecidos, e que irão determinar a arquitetura final do ambiente.

Segundo Perry [Perry92] não existem nomes para arquiteturas de software. O que existe é alguma intuição de que ocorrem diferentes tipos de arquitetura de software, mas que não foram ainda formalizadas ou institucionalizadas. Estes diferentes tipos de arquitetura tendem a se concentrar em aspectos relacionados a arquitetura de hardware, de rede de computadores e de construção do software em si.

Penedo [Penedo88][Penedo92][Penedo93b] representa a estrutura de um ambiente a partir da identificação de um conjunto de funcionalidades que devem existir nestes ambientes e que foram descritas na seção II.2. A forma de representação pode ser vista na figura II.2. A estrutura em camadas utilizada organiza a arquitetura de um ambiente em cinco níveis. A forma de representação indica, também, que existe um relacionamento *usa* implícito entre as camadas. Camadas de nível mais alto podem se utilizar dos serviços de camadas anteriores.

A **camada de hardware e sistema operacional nativo**, de mais baixo nível, corresponde ao nível físico, e pode ser representada por um conjunto de máquinas heterogêneas, em alguns casos distribuídas e interligadas por sistemas de rede que, aliadas ao sistema operacional, fornecem o conjunto de funcionalidades básicas necessárias para o funcionamento do hardware.

A **camada de estruturação** fornece a infraestrutura básica que deve estar presente na implementação de um ADS. Esta infraestrutura é que deve ser utilizada para a construção das capacidades funcionais, sendo seu principal objetivo facilitar a construção de novos componentes para o ambiente. Ela pode ser usada, também, para propósitos de integração. Nesta camada encontramos a separação de três funcionalidades distintas, que são um sistema de gerenciamento de objetos, um sistema de gerenciamento de interface com o usuário e um gerente do ambiente,

aliadas à possibilidade da existência de um grupo de serviços e capacidades, denominado sistema operacional virtual, que permitem a portatilidade do ambiente.

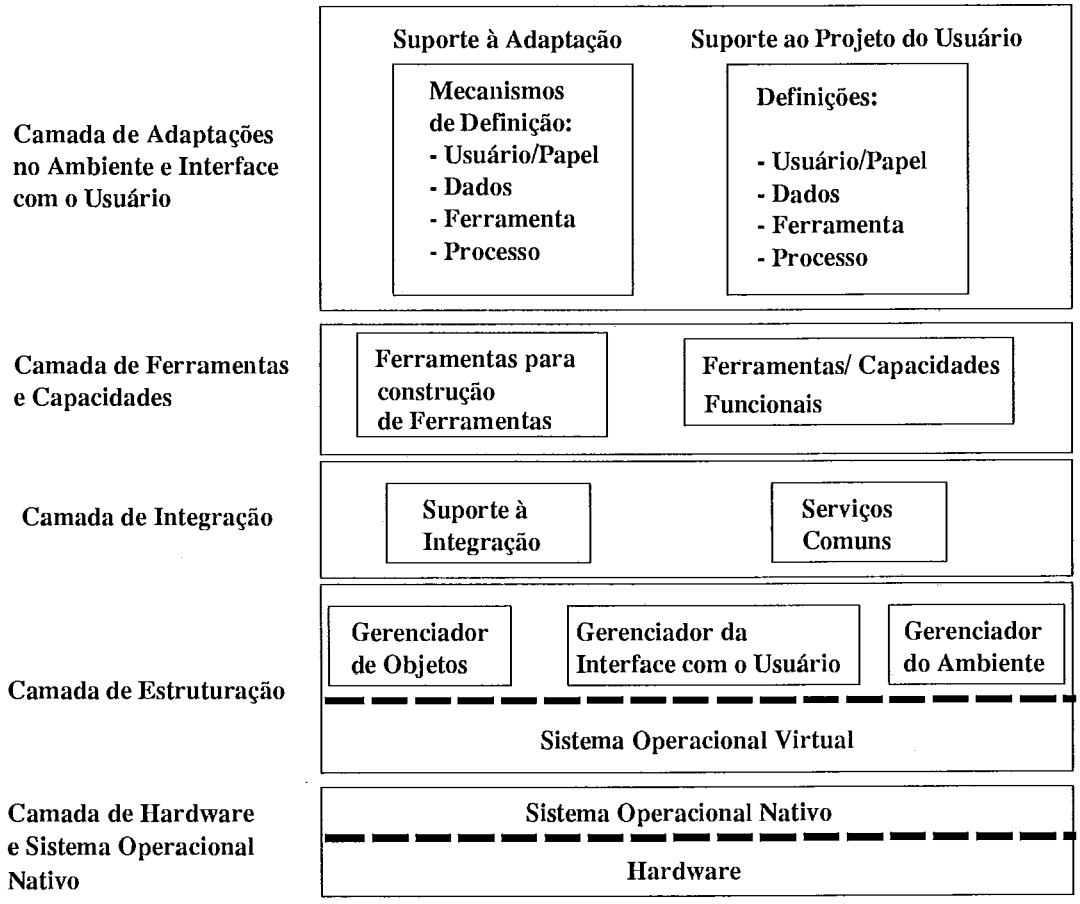



Figura II.2 - Estrutura utilizada por Penedo [Penedo92]

A **camada de integração** contém a definição das regras e diretrizes que vão governar o uso do ADS num projeto específico. Estas regras suportam a combinação estática ou dinâmica de ferramentas, de forma que elas possam cooperar harmonicamente, e são representadas pelo tratamento de integração de dados, controle e apresentação. De forma a proporcionar um maior controle sobre a integração e a prover aos níveis mais altos serviços que auxiliem em atividades

rotineiras, como por exemplo consultas e documentação, encontramos nesta camada um conjunto de serviços comuns a serem utilizados no ambiente.

A **camada de ferramentas e capacidades** contém os componentes que suportam as atividades do ciclo de desenvolvimento e, portanto, determinam a funcionalidade do ambiente. Normalmente, nesta camada encontram-se conjuntos distintos de ferramentas que devem ser selecionadas para compor um ambiente para um projeto específico. Estas ferramentas são divididas em duas categorias: ferramentas/capacidades funcionais e ferramentas para construção de ferramentas. Capacidades funcionais relacionam-se com a automação dos vários métodos e técnicas usados no suporte ao processo de desenvolvimento. Estas capacidades podem ser exemplificadas através de capacidades de reutilização de componentes e integração e teste de software. Ferramentas para construção de ferramentas, por sua vez, são utilizadas, principalmente mas não exclusivamente, pelos desenvolvedores de ambientes para desenvolver, ou gerar, componentes para o ambiente. Exemplos de ferramentas deste tipo são sistemas de desenvolvimento de interface e navegadores ("browser") genéricos.

A **camada de adaptações no ambiente e interface com o usuário** fornece dois conjuntos diferentes de funcionalidades, dependendo de que usuário irá utilizá-la: adaptador do ambiente ou usuário final. O adaptador do ambiente usa mecanismos de suporte à adaptação para definir e gerar ambientes específicos de um projeto. O usuário do ambiente utiliza as funcionalidades de suporte ao usuário para ajustar o ambiente de um projeto específico. Estes ajustes, que descrevem propriedades do ambiente, definem para o ambiente, características de seu comportamento e utilização, como por exemplo, a forma dos esquemas de ajuda e manipulação de janelas.

Uma outra forma usada para representar a estrutura de ambientes é considerá-la não como uma estrutura em camadas, mas diretamente a partir da funcionalidade de seus componentes, denominados neste caso como *serviços* que devem estar contidos no ambiente [Brown92c][Ecma90].

Nesta representação os *serviços* são fortemente relacionados, levando-se em consideração suas características e funcionalidades que permite que estes serviços sejam representados através de grupos bem definidos. O diagrama da figura II.3 representa esta estrutura. Não existe, nesta representação, a intenção de representar algum nível de hierarquia entre os agrupamentos de *serviços*. Pelo contrário, existe uma comunicação constante entre eles, que é permitida através de mensagens que são controladas pelo serviço de mensagens.

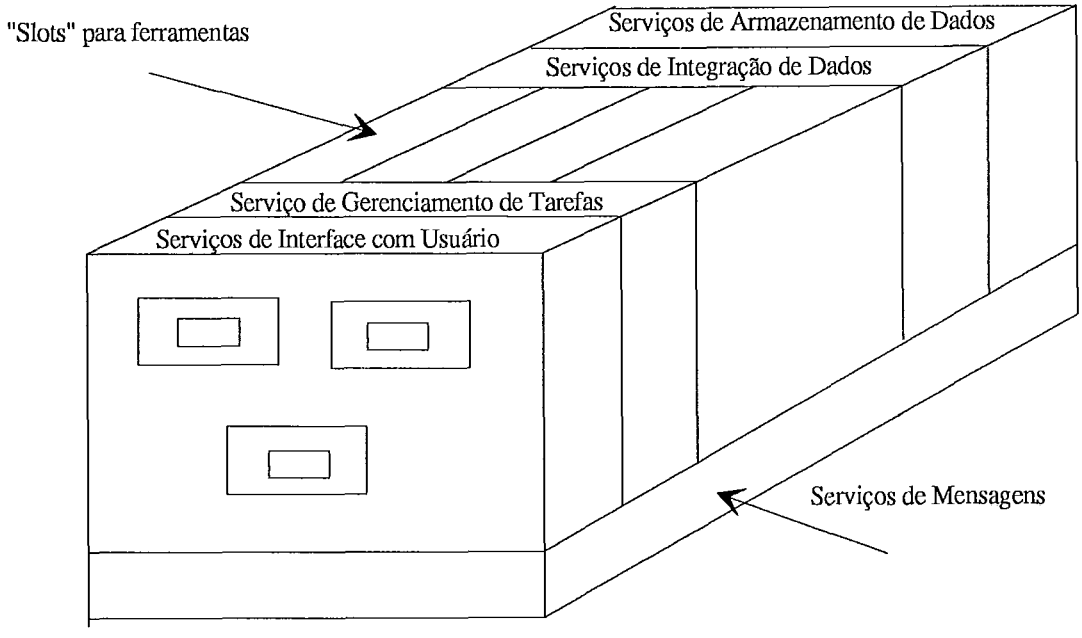



Figura II.3 - Estrutura utilizada por Brown [Brown92c]

Os serviços de mensagens permitem que comunicações ocorram entre serviços, entre ferramentas, e entre ferramentas e serviços. Os "slots", ou aberturas de conexão, existentes no modelo, são os locais onde é possível a integração de novas ferramentas ao ambiente e permitem que a funcionalidade do ambiente seja estendida, a partir da incorporação de novas ferramentas. Esta forma de

representação não implica que a arquitetura final do ambiente seja representada da mesma forma, ou seja, ambientes podem ser construídos contendo estas funcionalidades básicas embora não sejam estruturados desta maneira.

Os serviços de armazenamento de dados provêm facilidades para manutenção, gerenciamento e identificação de entidades ou objetos, e os relacionamentos entre estas entidades e objetos. Toda a funcionalidade básica de armazenamento e gerenciamento de informação é representada por este grupo. Considerações sobre a execução e o controle de processos, que podem estar fisicamente distribuídos junto com suas informações, também estão associados a este grupo.

Os serviços básicos existentes no grupo de armazenamento de dados são: Armazenamento de Dados (representam os meios e os locais para o armazenamento de informações); Relacionamento (permite a definição e a manutenção de relacionamentos existentes entre os objetos e entidades); Identificação (realiza e mantém associações de nomes externos para identificadores internos dos objetos e entidades); Localização (permite que as informações do ADS possam ser armazenadas, e posteriormente recuperadas, mesmo estando organizadas de forma distribuída); Transação de Dados (permite o controle de transações[5] no contexto de um ADS); Concorrência (estende a funcionalidade do armazenamento, permitindo que mais de uma transação possa ocorrer ao mesmo tempo); Suporte ao Processo (provê uma forma de comunicação e controle entre a estrutura de desenvolvimento do ambiente e o ambiente alvo, que pode ser representado por outras máquinas ou mesmo por um sistema computacional distribuído); Arquivamento (permite a manutenção de dados de projetos que não estejam sendo utilizados), e, Cópias de Segurança (permite a manutenção de cópias de segurança das informações geradas ao longo do desenvolvimento e sua total recuperação, no caso de uma eventual necessidade).

[5] Transação: É uma unidade de trabalho e uma unidade de recuperação realizadas a partir de uma sequência de operações atômicas. [Brown92c]

Os serviços de integração de dados aumentam a capacidade dos serviços de armazenamento de dados no sentido que provêm semânticas de alto nível e operações com as quais se consegue tratar os dados armazenados no repositório.

No grupo de integração de dados, encontram-se os seguintes serviços: Controle de Versões (permite a manutenção e armazenamento da evolução ocorrida nas informações ao longo do desenvolvimento); Configuração (permite a composição de objetos e entidades); Consultas (permite o acesso a informações pré-estabelecidas e que satisfaçam a determinadas condições); MetaDados (permite o controle e a manutenção de metadados[6]); Monitoração de Estados (permite a definição ou especificação dos estados do banco de dados e transformações para estes estados); Sub-Ambiente (permite a definição de conjuntos de dados e operações associados que podem ser utilizados no desenvolvimento), e, Troca de Dados (permite o mapeamento entre as informações existentes no repositório e a forma de apresentação utilizada pelo usuário).

Os serviços de gerenciamento de tarefas isolam o usuário de detalhes de utilização de ferramentas no ambiente, permitindo que o mesmo se abstraia e concentre-se na tarefa que tem que realizar, e não em como realizar a ativação das ferramentas que precisa para executar a tarefa. Um nome alternativo para este serviço é Gerenciamento do Processo de Software.

Os serviços existentes no grupo de gerenciamento de tarefas são: Definição de Tarefas (permite a definição das tarefas existentes no ambiente); Execução de Tarefas (prove a funcionalidade necessária para o controle e suporte a execução das tarefas); Transação de Tarefas (permite o controle da execução das tarefas no ambiente); Histórico das Tarefas (permite o acompanhamento de execuções de atividades passadas); Monitoração de Eventos (suporta a definição de eventos e ações a serem tomadas na ocorrência destes eventos); Auditoria e Contabilidade (permitem o acompanhamento da contabilidade do desenvolvimento, indicando o que foi feito e utilizado até o presente), e, Gerenciamento Funcional (permite a

[6] Metadados: são dados sobre dados. É um conjunto de informações que descrevem informações.

associação entre pessoas e as atividades que elas podem desempenhar no contexto do desenvolvimento).

Os serviços de mensagens tem como objetivo principal prover um padrão de comunicação entre serviços que podem ser utilizados para a comunicação entre ferramentas ou entre serviços.

Neste grupo, relacionado a mensagens, encontram-se os seguintes serviços básicos: Liberação de Mensagens (permite o controle e gerenciamento das mensagens que são trocadas entre os componentes do ambiente), e, Registro de Ferramentas (provê mecanismos que permite tornar uma ferramenta conhecida para o serviço de liberação de mensagens).

Os serviços de interface com o usuário tem como objetivo fornecer para o usuário um padrão consistente de utilização em todo o ambiente.

Os modelos de representação e estruturação apresentados mostram características fundamentais e necessárias em um ADS. No primeiro modelo encontramos uma organização onde o relacionamento entre níveis é feito a partir de uma determinada hierarquia, implicando numa propagação de esforços para a execução de uma determinada funcionalidade. Esta forma de representação implica numa visão muito mais funcional de cada nível do que propriamente das informações que são manuseadas em cada uma delas. No segundo modelo, por sua vez, funcionalidades são compartilhadas, independente de qualquer hierarquia, através de um esforço de gerenciamento e controle de mensagens, o que provoca um tráfego intenso de informações entre os serviços de cada grupo. Além disso, esta forma de representação impõe uma sobrecarga de atividades entre um serviço e outro, o que muitas vezes dificulta a visualização do que é realmente o serviço, ocultando serviços essenciais como a representação do conhecimento do processo, a reutilização de componentes e a forma participativa do trabalho.

Neste capítulo foi dada uma visão geral sobre Ambientes de Desenvolvimento de Software, seus requisitos e aspectos relacionados à sua construção.

Nos próximos capítulos, ADSs serão tratados focalizando, em detalhes, os aspectos relacionados à integração de ferramentas, objetivo principal deste trabalho.

# Capítulo III

# A Questão da Integração de Ferramentas em Ambientes de Desenvolvimento de Software

## III.1. Introdução

Ferramentas CASE isoladas tem se demonstrado inadequadas [Mosley92], [Vessey92]. O esforço dispendido no uso destas ferramentas, levando o usuário a gastar muito tempo em seu aprendizado [Miyoshi93], aliado ao problema de não conseguirem garantir a consistência e qualidade do produto ao longo do desenvolvimento, apontam para a necessidade de construção de ferramentas integradas que garantam um tratamento adequado do produto ao longo de seu ciclo de vida.

De acordo com Thomas [Thomas92] integração não é propriedade de uma simples ferramenta, mas de seus relacionamentos com outros elementos no ambiente. Estes relacionamentos demonstram de que forma as ferramentas se integram ao processo de desenvolvimento e à plataforma de execução. Para que estes relacionamentos possam existir, é preciso que se considere aspectos relevantes e interferentes na questão da integração.

A necessidade de se construir ferramentas CASE integradas está associada à evolução do processo de desenvolvimento. Esta evolução do processo de desenvolvimento e das aplicações aconteceu num período de tempo relativamente curto. Conforme descrito em alguns trabalhos [Norman92], [Brown92c], [Penedo93d], aplicações "batch" eram desenvolvidas nos anos 70, escritas em linguagens de 3ª geração, e utilizavam no máximo de editores de texto, compiladores e depuradores. O surgimento de aplicações "on-line", suportadas por sistemas de banco de dados, fez com que aplicações mais complexas fossem

desenvolvidas. Com esta tecnologia foi possível construir sistemas de apoio a decisão que ajudavam, de forma interativa, a análise de dados. Nos anos 80, junto com o crescimento de aplicações em tempo-real para dispositivos de controle e comunicações, foi criada a linguagem *Ada*. Ainda nos anos 80 começaram a surgir preocupações com o desenvolvimento de sistemas baseados em conhecimento, e já no final da década, começou-se a utilização de sistemas de informação estratégicos, que permitiam a obtenção de informações relevantes ao negócio e facilitavam,. desta forma, que gerentes acompanhassem a evolução de seus trabalhos.

Nos anos 90 os produtos de software estão incorporando as características descritas acima, de forma a permitir que facilitem um acompanhamento total do negócio. A construção destes sistemas requer a utilização de um conjunto considerável de tecnologia, que impõe um aumento ainda maior na complexidade do desenvolvimento, como por exemplo bases de dados distribuídas, arquitetura cliente-servidor [Gabel94], recursos de interface gráficos e multimídia, para que se consiga construí-los dentro dos prazos, com a qualidade garantida ao longo do desenvolvimento e dentro dos custos estimados. Este crescimento da complexidade é, segundo Norman [Norman92], o maior incentivo para o desenvolvimento de Ambientes de Desenvolvimento de Software integrados.

Além da questão da construção destas aplicações, é preciso considerar, também, aspectos relativos à manutenção destes produtos. Inerente a qualquer produto de software, manutenções devem ocorrer de forma a manter o produto adequado ao uso. Segundo Jarke [Jarke92], mesmo existindo no mercado uma grande quantidade de sistemas de alta complexidade e larga abrangência eles ainda são difíceis de manter e reutilizar. A principal causa desta dificuldade é a perda de integração. As etapas do processo de desenvolvimento normalmente não são integradas e os desenvolvedores, possuindo ferramentas CASE que apoiam atividades isoladas do desenvolvimento, não obtêm uma integração formal entre as etapas do desenvolvimento, entre o sistema e seu ambiente, e muito menos uma integração entre as tarefas do desenvolvimento. Chen [Chen92] afirma que embora as ferramentas CASE tenham afetado significativamente a prática de desenvolvimento de sistemas, ainda estão limitadas quanto a sua potencialidade

devido às dificuldades envolvidas em integrar estas ferramentas em ambientes coesos.

Para tentar solucionar, pelo menos em parte, a questão da integração de ferramentas em ambientes, alguns padrões (AD/Cycle [Mercurio90], PCTE [Thomas89b], ATIS [DEC92], IRDS [ANSI88], CAIS [Cais89], HP-SoftBench [Jenings89], NSE [Miller89], PMDB [Penedo84] ) foram definidos com o objetivo de proporcionar uma estrutura básica segundo a qual ferramentas podem ser construídas e agregadas, de forma a passar a constituir um Ambiente de Desenvolvimento de Software. Estes padrões tratam, de forma privilegiada, a questão específica do armazenamento e gerenciamento de informações, não considerando que o conceito de integração em ambientes de desenvolvimento de software relaciona-se com questões mais abrangentes que precisam ser consideradas.

Nas próximas seções trataremos a questão da integração de ferramentas. Partindo de diferentes definições de integração (seção III.2), analisamos as diversas abordagens propostas para resolver esta questão (seção III.2.1). Finalmente aborda-se, na seção III.3, a influência de integração em Ambientes de Desenvolvimento de Software.

## III. 2. Definições de Integração

Discutir integração, num contexto de Ambientes de Desenvolvimento de Software, e considerar apenas a questão do armazenamento e compartilhamento de informações não satisfaz os requisitos atuais dos ADS. A evolução destes ambientes demonstra a necessidade de considerar aspectos inerentes ao processo de desenvolvimento do software.

Existem diferentes visões sobre o conceito de integração e sua aplicabilidade na construção, e utilização, de um ADS. Autores como Brown

[Brown92a] [Brown92b] [Brown92c], Wasserman [Wasserman90], Thomas [Thomas92], Chen [Chen92], Mi [Mi92], Meyers [Meyers91], Garlan [Garlan87] , Yang [Yang93], Penedo [Penedo88] [Penedo92] [Penedo93b] [Penedo93d] e Pressman [Pressman92] tratam integração de forma ampla, cada um dando relevância a aspectos que julgam mais importantes, e considerando que existem dois usuários distintos do ADS, que são o desenvolvedor da aplicação (usuário do ambiente) e o construtor do ambiente.

Para o usuário do ambiente, segundo Thomas [Thomas92], a percepção da integração ocorre na interface com o ambiente, pois o mesmo espera que o ambiente seja composto por ferramentas que se apresentem de forma homogênea e facilitem o desenvolvimento do produto. Para o construtor do ambiente, que é o responsável pela construção e integração das ferramentas num ADS, esta percepção se dá pela facilidade e possibilidade de, com pouco esforço, conseguir fornecer ao usuário esta percepção de integração.

Estes autores tratam a questão da integração de diferentes formas.

Uma visão defendida por Brown [Brown92a] [Brown92b] [Brown92c], trata a questão da integração buscando identificar de que forma as ferramentas interagem entre si no ambiente, e sob que abrangência deve ser feito o tratamento das informações dentro do ambiente. Para Brown a questão da integração não está sendo tratada corretamente devido as pesquisas discutirem mecanismos de integração e não a semântica da integração.

Numa outra visão, Wasserman [Wasserman90], Chen [Chen92] e Thomas [Thomas92] tratam integração visualizando vários aspectos que devem ser considerados num contexto de ADS e quais os relacionamentos necessários entre as ferramentas para que se obtenha um uso efetivo e integrado das mesmas.

Como um terceiro enfoque, Mi [Mi92], Meyers [Meyers91], Garlan [Garlan87], Yang [Yang93], Penedo [Penedo88] [Penedo92] [Penedo93b] [Penedo93d] e Pressman [Pressman92] consideram que a integração deve ser tratada segundo uma abordagem centrada em um determinado tipo de mecanismo,

que seria o responsável por prover a integração no ADS.

A seguir descreveremos com mais detalhes estes três enfoques.

## III.2.1 Diferentes Enfoques para Integração

### III.2.1.1 Enfoque Ortogonal

Brown [Brown92a] [Brown92b] [Brown92c] trata a questão da integração em ADSs considerando que os grandes problemas, atualmente existentes, ocorrem devido às pesquisas se preocuparem muito mais com mecanismos de integração do que com a semântica da integração. Para Brown existe um forte relacionamento entre integração e reutilização de ferramentas. Afirma que é muito difícil, e custoso, construir ferramentas, e que, sendo assim, é fundamental a reutilização de ferramentas já existentes no contexto de um ADS.

Ainda segundo Brown, o conceito de integração só pode ser entendido se considerarmos os aspectos relacionados abaixo de uma forma conjunta, pois nenhum deles captura sozinho o conceito de integração:

- Integração de Interface: neste caso as ferramentas possuem as mesmas características de interface, utilizando as mesmas funcionalidades para tratar aspectos semelhantes. Este tipo de integração facilita ao usuário a utilização das ferramentas que compõem o ADS, criando uma estrutura de apresentação familiar ao usuário.

- Integração de Gerenciamento: neste caso facilidades de controle e gerenciamento devem estar disponíveis no ADS de forma a permitir o acompanhamento do processo em curso. A informação do gerenciamento deve ser derivada da informação técnica produzida pelos engenheiros de software.

- Integração Técnica: neste caso, que às vezes é confundido com integração do processo, o que se quer é permitir que o ADS seja construído para suportar

uma metodologia de desenvolvimento simples e coerente.

- Integração de Ferramentas: neste caso, considera-se que as ferramentas precisam compartilhar informações e o ADS deve fornecer facilidades que tornem isto possível, através do compartilhamento de um formato de dados comum que é utilizado para armazenar as informações num repositório;

- Integração de Equipes: neste caso, considera-se que o ADS deve facilitar o trabalho em equipe, assegurando uma boa comunicação entre os participantes, colaborando para que o trabalho seja realizado através da difusão e controle da informação, e de forma que não haja interferência direta de um membro da equipe no trabalho de outro.

Do ponto de vista das ferramentas, a integração é discutida com a preocupação de como as informações serão armazenadas e tratadas por elas. A abordagem utilizada neste ponto, determinará o nível de sofisticação com que as ferramentas irão trocar informações. Do ponto de vista do gerenciamento, falar em integração sintetiza a preocupação em controlar e gerenciar o uso das ferramentas. Quanto ao ponto de vista do processo, a preocupação está relacionada com a ordem em que as ferramentas devem ser utilizadas e para que informações elas devem ser ativadas.

Um aspecto interessante a ser destacado é que estas visões, segundo Brown, são ortogonais, ou seja, podemos ter um ADS bem integrado com relação às ferramentas mas não tê-lo integrado segundo os processos , ou gerenciamento.

Embora destacando a importância de se descrever a semântica da integração sob as dimensões descritas acima, apenas a questão da integração de ferramentas é tratada com mais detalhe pelo autor. Neste caso, são identificados níveis segundo os quais se poderia ter este tipo de integração. Esta classificação (figura III.1) tenta mostrar o grau de sofisticação do ADS, segundo o aspecto específico das ferramentas, e traz embutida, em sua definição, aspectos estruturais do ambiente.

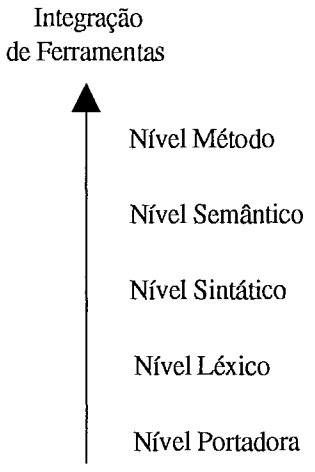



Figura III.1 - Níveis de Integração de Ferramentas [Brown92c]

O **nível Portadora**[1] é o mais baixo e primitivo. O nome Portadora é dado no sentido de representar a forma mais básica de transferência de informações entre as ferramentas. As ferramentas integradas neste nível não possuem nenhuma cooperação formal e trocam informações através da passagem de sequências de bytes, ou "streams", entre si. Ferramentas integradas, neste nível, dependem totalmente da plataforma em que o ADS está funcionando, tendo em vista a não existência de padrões de armazenamento de arquivos e tratamento de informações entre os sistemas. Um exemplo de ambiente que adota esta forma de integração é o sistema Unix padrão.

O **nível Léxico** é uma evolução do nível anterior. Neste nível é possível que conjuntos de ferramentas compartilhem formatos de dados e convenções de operações, permitindo que as ferramentas, que compõem o conjunto, possam estar fortemente integradas. Este tipo de integração é parcial, dado que apenas famílias de ferramentas compartilham e compreendem as convenções léxicas comuns às ferramentas. Mesmo ocorrendo este compartilhamento entre ferramentas, ainda não existe uma forte cooperação entre elas pois não existe comunicação direta

[1] "Carrier Level" no original, vem da área de comunicações

entre as mesmas, devendo toda modificação ser sentida através das informações. Desta forma uma ferramenta não conhece e não compreende as operações executadas por outra. Um exemplo de ambiente com este nível de integração é Unix-PWB [Dolotta78].

O **nível Sintático** possue uma força de representação maior que os níveis Portadora e Léxico devido as ferramentas utilizarem um conjunto de estruturas de dados e suas regras de formação. Este nível de integração, evita duplicação de esforços entre as ferramentas. Consegue-se construir este tipo de integração em ADSs através da definição de um esquema de banco de dados, que as ferramentas utilizarão para tratar as informações existentes no ambiente. O nível de detalhe, ou granularidade, do compartilhamento do esquema do banco de dados entre as ferramentas, determinará o grau da integração. Grande parte dos ADSs adotam este nível de integração de ferramentas. Alguns exemplos são PMDB [Penedo84], SIPS [Jino85], SLCSE [Strelich88], GRASPIN [Manucci89], AD/Cycle [Mercurio90] e FEGRES [Travassos90].

O **nível Semântico** aumenta o grau de integração das ferramentas através da representação da sintaxe das estruturas de informação, em conjunto com sua semântica. Isto significa que estão disponíveis, para as ferramentas, um conjunto de informações descrevendo as estruturas de dados e o significado das operações associadas a estas estruturas de dados. Para se obter este nível de integração podem ser utilizados dois tipos de abordagem. Uma é a implementação, nas ferramentas, das estruturas e operações previamente definidas. Cada ferramenta tem, então, uma cópia destas descrições e pode trabalhar sobre as informações. Numa segunda abordagem, ao invés de duplicar estas descrições pelas ferramentas, utiliza-se um banco de dados para armazenamento das descrições. Esta forma de tratamento, que é muito mais flexível, permite uma maior extensibilidade do ambiente. Um exemplo de ambiente que está próximo do nível semântico é o ambiente ADAGE [Giavitto89a].

O **nível Método** realiza a integração das ferramentas considerando que ferramentas devem ter conhecimento sobre o processo de desenvolvimento, e devem compartilhar este conhecimento com outras ferramentas. Este nível de

integração pode ser visto como a integração do processo. É preciso que as ferramentas, além de obterem a sintaxe e a semântica das informações envolvidas no processo de desenvolvimento, conheçam o processo no qual estão inseridas. Alguns trabalhos tratam a questão da integração e controle do processo em ADSs [Mi92], [Penedo93a], [Massolar93], [Oquendo93] embora ainda não considerem totalmente a forma de armazenamento e gerenciamento da informação.

### III.2.1.2 Enfoque Inter-Relacional

Um outro enfoque para a questão da integração em ADSs é dado por Wasserman [Wasserman90], Chen [Chen92] e Thomas [Thomas92]. Inicialmente Wasserman identificou cinco tipos de integração que deveriam ocorrer em ADSs, que posteriormente foram estendidos por Chen [Chen92] e Thomas [Thomas92]. A preocupação fundamental, neste tipo de tratamento, é tentar identificar os relacionamentos existentes entre as ferramentas do ambiente. Estes relacionamentos estão associados aos seguintes aspectos:

- **Plataforma**: relaciona-se aos serviços básicos que devem existir nos ADSs e que são utilizados pelas ferramentas para a execução de suas tarefas. Para que o ADS possa lançar mão destes serviços é preciso que exista disponibilidade de hardware e software básico, e que os serviços tenham sido construídos para utilizar esta estrutura em comum;

- **Apresentação**: trata a questão da interação do usuário com o ambiente, melhorando a eficiência e a efetividade da comunicação, através da diminuição da carga cognitiva necessária para realizar a interação. Neste contexto foram identificadas duas características associadas. A primeira trata a questão da aparência e comportamento da interação e indica a facilidade que o usuário terá no uso das ferramentas, ou seja, se a experiência adquirida pelo usuário na utilização de uma ferramenta poderá ser utilizada em outras ferramentas do ambiente. A segunda verifica o paradigma de interação e indica o quanto as ferramentas se utilizam dos mesmos modelos mentais e metáforas, para

representar situações semelhantes em seu uso;

- **Dados**: demonstra como deve ser a utilização das informações pelas ferramentas e assegura que todas as informações, no ambiente, são gerenciadas de forma consistente, independente da parte que está sendo utilizada e transformada. A integração de dados preocupa-se com a interoperabilidade das ferramentas, ou seja, descreve o que deve ser feito para que os dados gerados por uma ferramenta possam ser visualizados como uma informação consistente por outra ferramenta. Quanto menor o esforço necessário para fazer com que uma ferramenta utilize as informações geradas por outra ferramenta, maior será a interoperabilidade. A redundância, que deve ser evitada, indica o quanto de informação deve ser duplicado, ou derivado, entre as ferramentas para que elas possam utilizá-las. Pouca duplicação, ou derivação de dados, indica que ferramentas são bem integradas em relação a este aspecto. A consistência dos dados é outro fator que pode indicar o grau de integração. Ferramentas devem indicar suas ações e efeitos sobre os dados que manipulam e que também são utilizados por outras ferramentas. A troca de dados, entre ferramentas, indica quão bem uma ferramenta pode aproveitar, diretamente, os dados gerados por outra ferramenta. Estas características aplicam-se às informações persistentes, ou seja, aquelas que devem permanecer armazenadas no ambiente. Entretanto, ao longo do processo de desenvolvimento, as ferramentas criam informações intermediárias e temporárias, que também devem ser consideradas. Estas informações temporárias podem, eventualmente, ser utilizadas por outras ferramentas e isto provoca uma problema de sincronização, que indica como as ferramentas conseguem comunicar às outras sobre modificações e ações associadas às informações não persistentes e manipuladas no ambiente.

- **Controle**: relaciona-se com a comunicação e interoperabilidade das ferramentas e é responsável por permitir a flexibilidade de combinações de funcionalidades que o ambiente suporta, de acordo com as preferências de projeto, e seguindo a linha de processo adotada no ambiente. Esta flexibilidade de combinações pode ser obtida se as ferramentas compartilharem funcionalidade. A princípio, espera-se que todas as funções suportadas por uma ferramenta estejam disponíveis para as outras ferramentas pertencentes ao

ambiente. Neste aspecto de integração são destacados dois pontos importantes: a **provisão** e o **uso**. A provisão refere-se a quanto serviço uma ferramenta oferece às outras ferramentas existentes no ambiente, ou seja, que capacidade de exportação de serviços as ferramentas possuem. O uso refere-se a como as ferramentas conseguem utilizar os serviços e funcionalidades oferecidos por outras ferramentas.

- **Processo**: trata do papel de cada ferramenta no contexto do desenvolvimento, assegurando que as ferramentas interagirão, de forma efetiva, para o suporte ao processo de desenvolvimento definido. Para considerar a integração de processos é preciso tratar a decomposição (ou passos) do processo, os eventos e as restrições associadas. Ferramentas são ditas bem integradas na decomposição do processo se seus objetivos são coerentes com a decomposição em si. Quanto aos eventos, quando existe uma cooperação na ocorrência dos eventos, isto é, a notificação de um evento relativo a um processo por uma ferramenta deve ser respondida pelo tratamento deste evento por outra ferramenta, elas são bem integradas . Quanto às restrições, ferramentas são bem integradas se existe cooperação no sentido de garantir que as restrições associadas ao processo serão garantidas e observadas.

Na figura III.2 podemos visualizar a integração em ambientes seguindo a linha de Wasserman e Thomas. Nesta figura a integração de plataforma não é considerada por Thomas, devido ao foco de seu trabalho ser o relacionamento entre as ferramentas e não a estrutura básica do ambiente segundo a qual as ferramentas serão construídas.

### III.2.1.3 O enfoque Estrutural

Mi [Mi92], Meyers [Meyers91], Garlan [Garlan87], Yang [Yang93], Penedo [Penedo88] [Penedo92] [Penedo93b] [Penedo93d] e Pressman [Pressman92] tratam a questão segundo uma abordagem centrada numa estrutura de integração, tendo como ponto principal o armazenamento e compartilhamento

de informações. Considerações são feitas em relação a aspectos de integração de apresentação e controle, embora um detalhamento maior não seja realizado. A visualização destas estruturas se dá a nível dos serviços existentes no ambiente, descritos a partir de alguma estrutura básica.

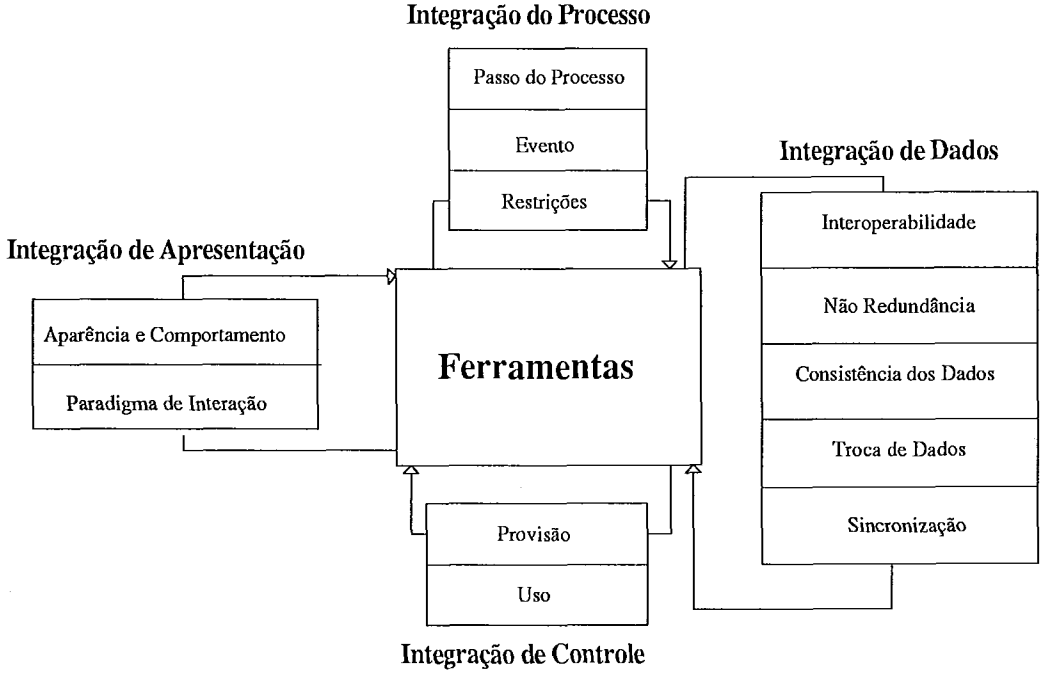



Figura III.2 - Visualização da Integração [Thomas92]

No enfoque estrutural podem ser identificados três tipos de integração: **integração de dados,** suportada por uma base de dados e serviços de integração de dados; **integração de controle,** suportada pelo gerenciamento do processo e serviços de mensagens entre as ferramentas; **integração de apresentação,** suportada pelos serviços de interface com o usuário.

Segundo Chen [Chen92], a capacidade de compartilhar informações de projeto é a chave para a integração de ferramentas. Esta necessidade de compartilhamento das informações, que surge durante o desenvolvimento de um

produto de software, mantendo-as acessíveis e inteligíveis durante todas as fases do ciclo de desenvolvimento, provocou o aparecimento de algumas abordagens para o tratamento e armazenamento destas informações.

Num contexto de ADS, informação é um conjunto de dados que possibilita a compreensão e a descrição do que está sendo desenvolvido. Esta definição possibilita o aparecimento de uma variedade de tipos de dados compondo a descrição do produto, e que devem ser tratados e entendidos durante todo o ciclo de desenvolvimento. Somada à informação pertinente ao produto em si, devemos considerar, neste ponto, o conjunto das informações e conhecimentos necessários ao ambiente para uma perfeita utilização dos métodos empregados no desenvolvimento do produto.

O tratamento destas informações pode ser realizado através de abordagens tradicionais, que se utilizam de arquiteturas computacionais bem definidas, ou através de enfoques diferenciados, construídos sob arquiteturas computacionais existentes mas que aumentem seu poder de representação de forma a proporcionar recursos sintáticos e semânticos no tratamento das informações.

Dentre as abordagens normalmente utilizadas para a representação e utilização das informações, podemos destacar [Garlan87], [Meyers91], [Chen92]:

- **Encadeamento de Tarefas**

Neste tipo de abordagem as ferramentas que compõem o ambiente são encadeadas, provocando que a saída de uma ferramenta seja a entrada da outra (a próxima a executar). Isto define um modelo sequencial, conforme apresentado na figura III.3 , onde as ferramentas se comunicam em estágios através de interfaces muito bem definidas. Exemplos deste tipo de ambiente são o UNIX e INTERLISP [Orth89].

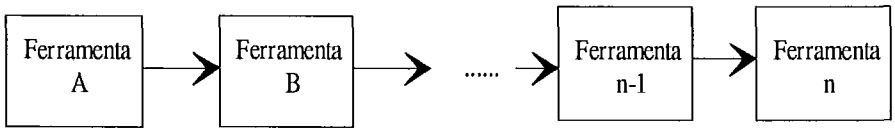



Figura III.3 - Integração a partir de Sequencialização de Tarefas [Meyers91]

A utilização da abordagem sequencial traz algumas vantagens para o construtor do ambiente entre as quais destacam-se:

1) Representação apropriada da informação: como o ambiente é formado a partir de módulos bem definidos, estes módulos podem ser construídos para tratar especificamente um determinado tipo de informação, provendo recursos de entrada e saída que satisfaçam aos requisitos do ambiente;

2) Encapsulamento: cada módulo realiza uma função específica em tipos de dados pré-definidos;

3) Abstração: devido ao módulo já estar definido e construído, ele pode ser utilizado diretamente sem preocupação com sua representação interna;

4) Proteção: consegue-se garantir que um determinado módulo não interferirá em outras informações que não sejam aquelas para as quais foi projetado;

5) Evolução: é assegurada através da construção de novos módulos que desempenhem as funções necessárias;

6) Reutilizabilidade: o mesmo módulo pode ser aproveitado para compor atividades diferentes;

7) Composição: a partir de módulos já existentes, é possível a construção de diferentes ferramentas alterando-se a ordem de execução dos módulos.

Apesar das vantagens apresentadas, o encadeamento de informações traz consigo alguns problemas que não podem ser esquecidos ao se analisar esta estrutura de execução. Podem ser destacados:

1) Estrutura sequencial do ambiente: o fato de, nesta abordagem, ser feito um encadeamento de módulos, em ordem bem definida, de forma a se construir uma ferramenta, e ao mesmo tempo existir uma forte dependência de informações entre elas, obriga ao ambiente possuir uma estrutura sequencial de processamento, não permitindo ao usuário liberdade na utilização das ferramentas;

2) Alta redundância de dados: o encadeamento e a reutilização dos módulos provoca redundância de dados, que é aumentada, ainda mais, pela própria estrutura utilizada para se realizar o encadeamento;

3) Alto custo de transformação da informação: como não existem esquemas de armazenamento temporário que permitam às ferramentas o aproveitamento de resultados parciais gerados por outras ferramentas, toda informação que já tenha sido tratada e transformada, embora não faça parte do encadeamento, necessita ser novamente tratada, o que aumenta bastante o custo de transformação da informação;

4) Unidirecionalidade da informação: a estrutura do ambiente só permite que se caminhe num único sentido, não permitindo ao ambiente suportar que o processamento seja feito em qualquer ponto.

- **Arquivos Compartilhados:**

Neste tipo de estrutura (figura III.4) as informações geradas a partir das ferramentas são armazenadas em arquivos compartilhados. Esta abordagem define um modelo concorrente, onde a informação é compartilhada entre todas as ferramentas que compõem o ambiente. Isto provoca um ambiente influenciado diretamente pela estrutura de informação dos arquivos, cujo funcionamento é

fortemente dependente do modelo de dados utilizado, além de dificultar bastante a construção das ferramentas que compõem o ambiente, dada a necessidade de adaptação à estrutura de dados e não ao método.

Apesar do desenvolvimento do ambiente ser direcionado de acordo com a estrutura de dados, este tipo de enfoque apresenta algumas vantagens que necessitam ser consideradas, entre as quais destacam-se:

1) Cooperação entre ferramentas: o fato do ambiente possuir um conjunto de arquivos único, possível de ser acessado por todas as ferramentas componentes do ambiente, possibilita uma forte cooperação entre as ferramentas, permitindo o aproveitamento de qualquer transformação intermediária sofrida pela informação que seja armazenada nos arquivos.

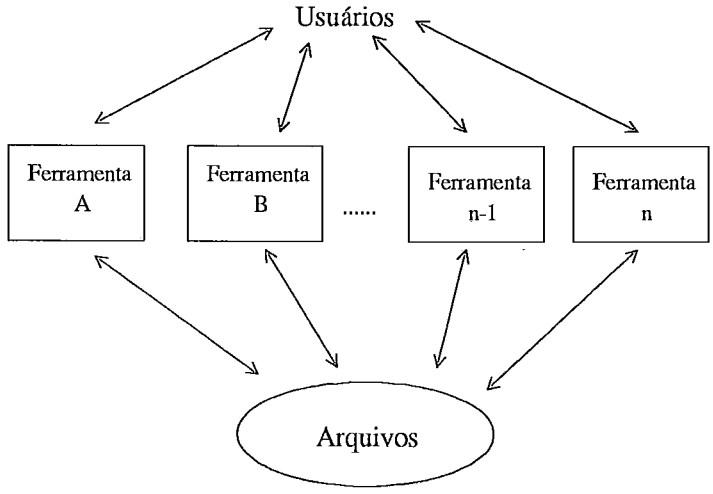



Figura III.4 - Integração a partir de arquivos compartilhados [Meyers91]

2) Compartilhamento das informações: a base de dados do ambiente, representada pelos arquivos, é a principal responsável pelo compartilhamento de informações entre as ferramentas. Desta forma é possível que a execução do ambiente se realize a partir de qualquer ponto do ciclo de desenvolvimento, tornando as ferramentas capazes de tratar e transformar a informação de acordo com suas necessidades e a disponibilidade destas informações nos arquivos.

3) Núcleo de suporte para as ferramentas: a unicidade da base de dados faz com que possa existir no ambiente um conjunto de atividades básicas responsáveis pelo gerenciamento das informações nestes arquivos, simplificando o processo de construção e manutenção das ferramentas.

Analisando este enfoque na sua forma mais pura, onde não são consideradas facilidades adicionais construídas para suportar atividades específicas das ferramentas tais como interface com o usuário, troca de mensagens e demais mecanismos convencionais de programação, podemos destacar as seguintes desvantagens:

1) Não suporta abstração: o fato da construção do ambiente estar diretamente ligada à estrutura de dados compartilhada pelas ferramentas não permite ao desenvolvedor se utilizar de abstração no desenvolvimento e construção de novas ferramentas.

2) Permite uma representação imprópria da informação: a utilização de uma estrutura de dados diferente da utilizada pelo ambiente pode provocar erros e inconsistências de tratamento, só detectáveis na execução da ferramenta, o que pode levar o ambiente a modificar as informações existentes, introduzindo erros durante o processo de desenvolvimento do produto, e até mesmo destruindo as informações existentes, tornando todo o trabalho realizado perdido e inutilizado.

3) A evolução do ambiente é restrita: toda nova ferramenta deve ser construída de forma a se enquadrar no modelo e estrutura existente no ambiente.

4) Reutilização limitada: ferramentas podem ser reutilizadas apenas se forem utilizadas em novos ambientes que possuam o mesmo tipo de estrutura de dados para armazenamento.

Embora o compartilhamento de informações apresente estas desvantagens, ele é utilizado em alguns ambientes já desenvolvidos. O uso do compartilhamento de informações é encorajado, na maioria das vezes, pela disponibilidade de

ferramentas e bibliotecas que diminuem o custo de desenvolvimento e facilitam sua construção. Exemplos de ambientes desenvolvidos segundo esta estrutura são DREAM [Hunke81], SIPS [Jino85], FEGRES [Travassos90].

- **Difusão de Mensagens por Seleção**

A idéia básica desta estrutura de integração (figura III.5) é fazer com que as ferramentas possam se comunicar através da troca de mensagens. Estas mensagens servem para controlar e coordenar a interação entre as ferramentas e comunicar as modificações associadas com uma determinada informação. Para que isto possa ocorrer é preciso que exista um elemento entre as ferramentas que seja responsável pela coordenação do envio das mensagens. Este elemento, chamado servidor de mensagens, regula o tráfego de mensagens entre as ferramentas, selecionando e direcionando as mensagens específicas de cada ferramenta, evitando com isto que ferramentas recebam mensagens que não lhes interesse. Um ambiente que utiliza esta estrutura de integração é Field [Reiss90].

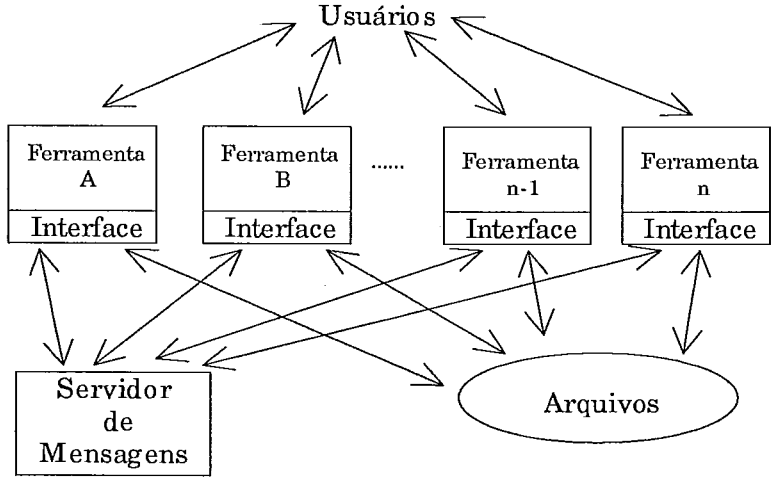



Figura III.5 - Integração de Ferramentas a partir de Difusão de Mensagens [Meyers91]

A utilização desta abordagem para realizar a integração de ferramentas em

ambientes provoca uma complicação adicional às abordagens anteriores. É preciso que se construa o servidor de mensagens e que seja definido um protocolo de comunicação básico entre as ferramentas. Este protocolo de comunicação deve permitir que as ferramentas possam registar novas mensagens para o servidor, enviar mensagens para o servidor direcionadas para alguma ferramenta e receber, na medida do possível, as mensagens que o servidor esta enviando para a ferramenta. Num ADS, ferramentas podem ser incorporadas dinamicamente, cada uma contendo um conjunto próprio de mensagens que consegue tratar. Isto provoca uma modificação constante no protocolo de comunicação. Da mesma forma a incorporação de novas ferramentas no ambiente pode provocar a necessidade de adaptação das ferramentas já existentes, de forma a poder compreender as mensagens que a nova ferramenta irá enviar.

- **Banco de Dados Simples**

Neste tipo de abordagem, que é utilizado por grande parte dos ADS existentes, o que se pretende é definir um modelo de dados tal que possa ser utilizado e compartilhado por todas as ferramentas do ambiente. Embora seja este o objetivo, nem sempre se consegue definir modelos que satisfaçam a todos os requisitos das ferramentas.

Na figura III.6 podemos ver a estrutura deste enfoque de integração. Não existe nesta abordagem formas de comunicação entre as ferramentas, exceto quando é possível no banco de dados a definição de "triggers"[2]. Mesmo neste caso, pouca comunicação é obtida entre as ferramentas, tendo em vista que se consegue detectar apenas os eventos que ocorreram no banco de dados, e não os que ocorrem internamente na ferramenta.

A vantagem desta abordagem é a possibilidade da manutenção da consistência da informação através do banco de dados.

---

[2] eventos que provocam o acontecimento de alguma ação

As desvantagens são relativas à construção de novas ferramentas para o ambiente. Estas ferramentas precisam conhecer o modelo de dados como um todo, não conseguindo se dedicar especificamente às informações que são de seu interesse.

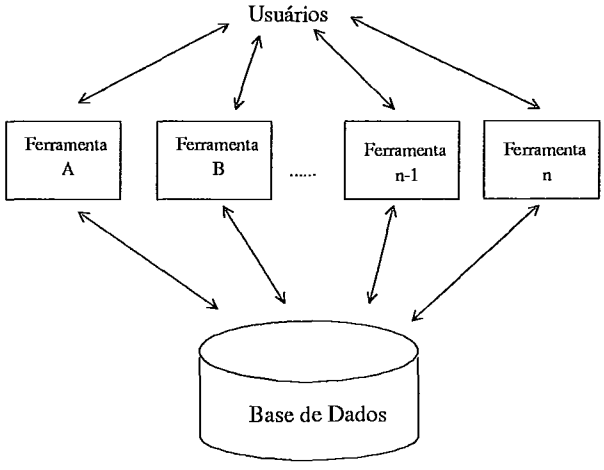



Figura III.6 - Integração de Ferramentas a partir de uma base de dados [Meyers91]

Alguns ambientes que utilizam esta abordagem são PMDB [Penedo84], SLCSE [Strelich88], GRASPIN [Manucci89], AD/Cycle [Mercurio90].

- **Modelo Orientado a Visões**

Nesta abordagem (figura III.7), que é uma mistura dos modelos de sequenciamento e compartilhamento de informações, a informação existente passa a ser tratada como um conjunto de objetos bem definidos, armazenados numa base de dados orientada a objetos, comum a todas as ferramentas, e que é acessada através de uma interface chamada Visão. Isto proporciona uma representação mais robusta das informações existentes e permite que as ferramentas se utilizem de objetos de forma a não interferirem entre si durante a execução de suas tarefas. A estrutura do ambiente suportando a orientação a objetos faz com que uma série de características do paradigma de orientação a objetos seja aproveitada pelas

ferramentas, tais como troca de mensagens, polimorfismo, reutilização, encapsulamento e abstração.

A vantagem desta forma de construção é uma união das vantagens do enfoque por Encadeamento de Tarefas e do Armazenamento de Informações.

A principal desvantagem deste modelo é que a liberdade de criação de visões pelas ferramentas para objetos compartilhados por todo o ambiente deve ser restringida, de forma a permitir que o ambiente possa misturar e controlar as visões definidas e associadas a cada ferramenta do objeto em questão. Uma descrição detalhada deste enfoque pode ser encontrada em [Garlan87].

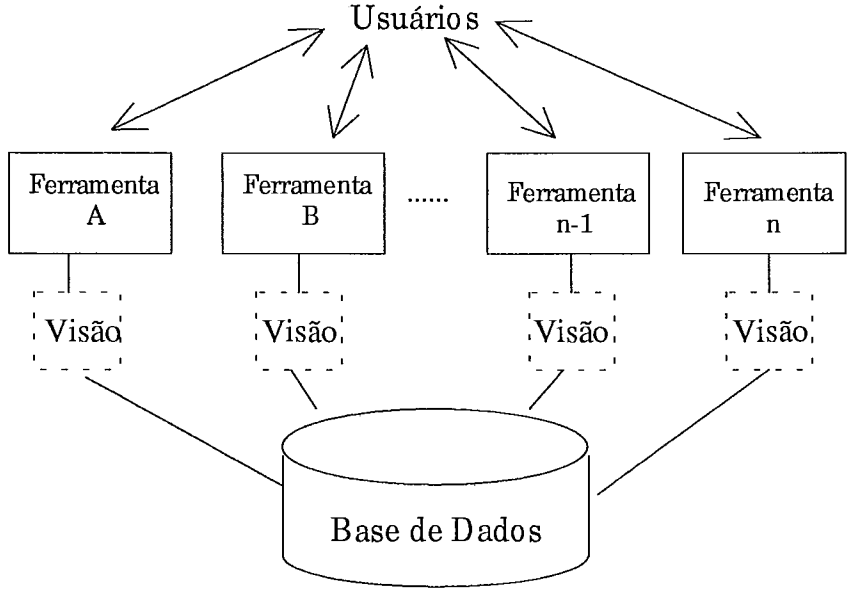



Figura III.7 - Integração de ferramentas a partir de Visões [Meyers91]

- **Modelo Canônico**

O uso de um modelo canônico (figura III.8) sintetiza a idéia do ADS possuir uma representação das informações, utilizando uma base de dados para armazenamento, que possua generalidade e força de representação suficiente para

tratar as informações manuseadas pelas ferramentas. Nesta abordagem tem-se a semântica e a sintaxe descritas pela estrutura de armazenamento e toda parte de controle e sincronização das ferramentas executadas a partir de modificações nesta estrutura. O compartilhamento de uma estrutura única, e básica, de informação simplificaria a construção de novas ferramentas para o ambiente.

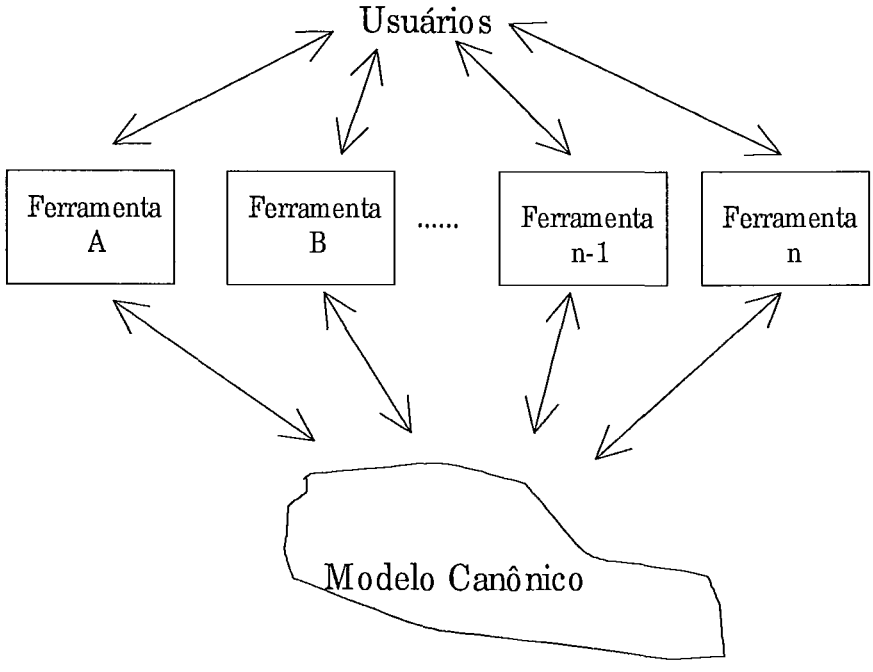



Figura III.8 - Integração de Ferramentas a partir de um modelo canônico [Meyers91]

Ainda não foi determinada uma estrutura de informação que possibilite sua utilização por todas as ferramentas. Entretanto, trabalhos têm sido desenvolvidos no sentido de prover mecanismos que permitam o desenvolvimento destas estruturas, como por exemplo GRAS [Kiesel93] e HyperCASE [Cybulsky92].

## III.3 Influência da Integração em Ambientes de Desenvolvimento de Software

Um ADS deve, naturalmente, ser adequado para o usuário. Isto quer dizer que ele deve reduzir o esforço e não complicar o trabalho do usuário [Hubert89]. Uma forma de atingir esta característica é permitir ao usuário a possibilidade de integrar ao ambiente ferramentas de sua preferência. Desta forma a arquitetura do ambiente deve permitir que ferramentas possam cooperar umas com as outras, que ferramentas possam ser conectadas ao ambiente e que o ambiente forneça suporte metodológico ao desenvolvimento.

Conforme pode ser observado na seção III.2 a forma mais usada para integração é a utilização de um modelo de dados comum, armazenado a partir de uma base de dados e compartilhado pelas ferramentas do ambiente. Esta abordagem de integração não permite a obtenção de muita flexibilidade na integração de ferramentas, tendo em vista a necessidade de fornecer, para todas as ferramentas, uma descrição completa do modelo de dados e das operações necessárias para tratá-lo. Se a ferramenta for desenvolvida desde o início voltada para ser utilizada no ambiente, a integração pode ser realizada de forma mais fácil. Mas estas ferramentas, inevitavelmente, não cobrirão todas as atividades e preferências do usuário. A utilização de ferramentas externas torna-se uma necessidade e surge com isto um problema para a reutilização de ferramentas que já existam e estejam disponíveis para o usuário.

### III.3.1 Ferramentas Internas x Ferramentas Externas

A integração de ferramentas construídas desde o início para serem utilizadas no ambiente não é problema. Em sua construção, a princípio, foram utilizados os conceitos existentes para a integração no ambiente. Denominamos estas ferramentas de *ferramentas internas* ao ambiente. Ferramentas internas

usufruem de toda a potencialidade do ambiente e apresentam para o usuário, se tiverem sido bem construídas, uma sensação de integração muito forte.

Por outro lado ferramentas que são desenvolvidas sem levar em consideração a arquitetura e a disponibilidade de serviços do ambiente são denominadas *ferramentas externas*. Estas ferramentas apesar de não adotarem os padrões de integração do ambiente, ainda assim, podem ser utilizadas no ambiente de forma a permitir ao usuário configurar suas preferências de trabalho. Não só o usuário dita suas preferências, mas o ambiente, em alguns casos, pode precisar utilizar uma ferramenta externa no sentido de complementar alguma atividade do desenvolvimento para a qual ferramentas internas não tenham ainda sido desenvolvidas.

Yang [Yang93] identifica maneiras de se integrar ferramentas externas ao ambiente. Estas formas, que definem enfoques ou paradigmas de acoplamento, mostram que tipo de relacionamento pode ser esperado de ferramentas que são incorporadas ao ambiente.

Ferramentas podem ser fortemente acopladas ao ambiente. Isto implica que a ferramenta utiliza-se de recursos e serviços do ambiente. Ferramentas acopladas desta maneira são consideradas, neste caso, ferramentas internas, tendo em vista que a integração com o ambiente é total. Exemplos de ambientes que utilizam somente este esquema de acoplamento são Gandalf [Ellison85] e Centaur [Borras88].

Entretanto, ferramentas podem não ser acopladas ao ambiente. É a forma mais simples de integração. Neste caso toda a comunicação com o ambiente será feita a partir do tratamento dos arquivos gerados pelas ferramentas. Não existe nenhum controle da comunicação entre as ferramentas, exceto o normalmente existente a nível de sistema operacional. Para realizar este tipo de acoplamento é preciso que sejam construídos dispositivos de transferência, ou conversores de informação, permitindo tanto ao ambiente como à ferramenta entender o que está sendo realizado no contexto das informações pertencentes à ferramenta. Um ambiente que adota este esquema é o Unix-PWB[Dolotta78]

Ferramentas podem, ainda, ser fracamente acopladas. Isto é um nível intermediário entre não estar acoplada e ser uma ferramenta interna. Neste tipo de acoplamento de alguma forma a ferramenta consegue se comunicar com o ambiente e utiliza algum tipo de serviço. Neste acoplamento o usuário tem a percepção de que a ferramenta está integrada ao ambiente, embora esta situação de integração, internamente, não seja totalmente verdade. Este tipo de acoplamento pode ser feito a partir da utilização de uma base de dados que concentre as informações e permita sua utilização por ferramentas externas. Ambientes que adotam esta forma de integração são PMDB [Penedo84], Adage [Giavitto89a], AD/Cycle [Mercurio90], PCTE [Gallo86] e FEGRES [Travassos90]. Outra forma de abordar este acoplamento fraco é através da troca de mensagens entre as ferramentas, que pode ser implementada a partir de um mecanismo que controle o tráfego de mensagens entre elas. Exemplos de ambiente deste tipo são Field [Reiss90] e Fegres/X [Vilanova92].

### III.3.2 Integração e Flexibilidade de Ambientes de Desenvolvimento de Software

A forma como o ADS trata a integração de ferramentas, permitindo ou não que ferramentas externas sejam incorporadas, determina a flexibilidade e extensibilidade do ambiente. Segundo Ghezzi [Ghezzi91] o ambiente deve ser altamente integrado, mas altamente evolutível e aberto. Ambientes de Desenvolvimento de Software que não suportam o acoplamento de ferramentas externas tendem a não atender completamente às preferências do usuário.

O aproveitamento de ferramentas externas relaciona-se diretamente com a questão da reutilização de ferramentas no ambiente. A falta de padronização no desenvolvimento de ferramentas faz com que seja necessária a definição de mecanismos para a incorporação de novas ferramentas ao ambiente.

Ambientes de Desenvolvimento que suportam apenas a utilização de ferramentas internas são pouco flexíveis. Normalmente são aplicados a atividades específicas do ciclo de vida e não permitem ao usuário ajustá-lo às suas preferências de trabalho. Além disso, é muito difícil a incorporação de novas ferramentas a estes ambientes. Exemplos de ambientes deste tipo são Gandalf [Ellison85] e Centaur [Borras88]. Embora possuam uma força de representação da informação considerável, a extensão destes ambientes só é possível se construirmos novas ferramentas utilizando totalmente a estrutura do ambiente.

Por outro lado ADSs que permitem a incorporação de ferramentas externas são mais flexíveis e extensíveis, embora esta flexibilidade imponha algumas restrições quanto à força de representação das informações representadas e manuseadas pelo ambiente.

A forma como é tratada a informação e a maneira como as ferramentas são incorporadas ao ambiente varia de acordo com as soluções de projeto. Alguns destes ambientes se utilizam de um modelo de dados comum, compartilhado entre as ferramentas e armazenado numa base de dados. Alguns ambientes que trabalham desta forma são PMDB [Penedo84], PACT [Thomas89a], Graspin [Mannuci89] e AD/Cycle [Mercurio90]. Para que as ferramentas externas sejam incorporadas ao ambiente é preciso que elas entendam as informações manipuladas pelo ambiente. Ferramentas nestes ambientes estão fortemente acopladas, ou no pior caso, fracamente acopladas. Eles não provêm mecanismos de tradução para informações geradas por ferramentas externas.

Outros ambientes se utilizam de um mecanismo de troca de mensagens entre as ferramentas, e a incorporação de ferramentas externas ao ambiente é feita a partir da conexão da ferramenta com um mecanismo de comunicação. Este enfoque só permite ferramentas fracamente acopladas, tendo em vista que o único compromisso das ferramentas é com o protocolo de comunicação, e não com o formato da informação. Esta abordagem provoca o aparecimento de ambientes flexíveis e extensíveis, tendo em vista que, a priori, é possível dotar ferramentas (pelo menos a nível de sistema operacional) de meios de comunicação primitivos

com outras ferramentas. Um ambiente que utiliza esta forma de integração é Field [Reiss90].

Uma evolução deste conceito é a utilização de um barramento de software. Segundo Beach [Beach92] um "software bus"[3] é um mecanismo para conexão de componentes de software que provê um protocolo de comunicação padrão que permite a comunicação entre componentes , que utilizem este protocolo. Alguns ambientes que utilizam este conceito são Kernel/1 [Shu93], Kernel/2r [Shu93] e ETHOS [Takahashi90]. A flexibilidade encontrada neste tipo de integração é semelhante ao mecanismo de comunicação convencional, embora neste caso seja aplicado a ambientes distribuídos.

### III.3.3 Integração x Reutilização

Independente do mecanismo utilizado, no ambiente, para realizar a integração e que determinará sua flexibilidade e extensibilidade, existe um problema na identificação dos componentes, ou ferramentas, que podem ser reutilizados na construção do ambiente. Como atesta Ghezzi [Ghezzi91], existe muita dificuldade na determinação de quais componentes deveriam ser padrões, como especificar os componentes padrão, como construí-los, como classificá-los e recuperá-los de catálogos.

Embora se encontre na literatura algumas propostas de padronização para a construção de ferramentas, como por exemplo ESF [Penedo92] e PCTE[Boudier88], não são encontradas muitas ferramentas que satisfaçam completamente a estes padrões, dificultando seu aproveitamento em ambientes pela impossibilidade de identificar sua forma de construção e tratamento de informações que permitam sua incorporação em ambientes.

Basili [Basili92] afirma que reuso é um conceito simples e implica na

[3] Na definição de Penedo [Penedo93c] o termo "software bus" descreve uma classe de arquitetura que compartilha um componente comum, chamado "software bus", que implementa um modelo de comunicação abstrato para a interoperação de componentes num ambiente distribuído.

utilização de alguma coisa mais de uma vez. Num contexto de software, reuso significa o uso de componentes de software existentes para a construção de novos sistemas [Prieto-Diaz93a]. Segundo Freeman [Freeman83] reutilização se aplica não somente a fragmentos de código, mas a todo produto gerado em fases intermediárias do ciclo de desenvolvimento, tais como especificações, estruturas de projeto e qualquer informação que o usuário precisa para criar software. Numa evolução desta definição Basili [Basili88] estendeu-a como sendo o uso de tudo associado com um projeto de software, incluindo o conhecimento.

Segundo Brown [Brown92c], para que se possa efetivamente reutilizar ferramentas, é preciso que se tenha uma compatibilidade interna entre as ferramentas. É difícil a obtenção de compatibilidade interna em ferramentas que tenham sido desenvolvidas isoladamente, não considerando nenhuma estrutura de ambiente, exceto os serviços existentes a nível de sistema operacional. Isto seria equivalente a níveis mais baixos de integração de dados, segundo a classificação da seção III.2.1.1, que seriam os níveis portadora e léxico.

Podemos visualizar ferramentas como módulos, aos quais se aplicam os conceitos de coesão e acoplamento tratados por Myers [Myers75] e posteriormente estendidos por Buxton [Buxton91]. A coesão, que mostra uma característica interna ao módulo e mede o quanto o módulo cumpre o seu papel, é classificada nos níveis coincidental, lógico, temporal, funcional e abstrato.

Níveis de coesão funcional e abstrato são os mais interessantes e aqueles que normalmente se procura obter no processo de construção do módulo. O nível funcional representa que o módulo contém, ou realiza, uma única e bem definida função. O nível abstrato representa, num contexto de orientação a objetos, que as funções relacionadas a uma classe estão totalmente encapsuladas nesta classe. O acoplamento, que mostra uma característica entre módulos e mede o grau de dependência entre eles, é classificado nos níveis direto, comum, estruturado a bloco, em rede e abstrato. Módulos com baixa dependência e boa qualidade devem ser construídos com acoplamento em rede ou abstrato. O nível em rede implica que o módulo tem bem definidas suas interfaces de entrada e saída. O nível abstrato, num contexto de orientação a objetos, implica que o módulo possui procedimentos

passíveis de verificação na ativação dos serviços entre os módulos.

Brown [Brown92c] afirma que, para ferramentas poderem ser reutilizadas elas precisam ter alta coesão e baixo acoplamento. Por outro lado, para que se consiga um conjunto de ferramentas bem integradas é preciso que se tenha alto acoplamento, e provavelmente, baixa coesão. Sob este ponto de vista reutilização está em conflito direto com a integração. Entretanto, pode ser possível obter reutilização e integração de ferramentas se forem observados os níveis certos de granularidade. É possível se trabalhar com conjuntos de ferramentas integrados para uma determinada fase, e é possível reutilizar este conjunto de ferramentas em outros ambientes, desde que se tenha uma integração de conjuntos que satisfaçam a fases específicas do ciclo de desenvolvimento realizada de forma um pouco mais relaxada.

# Capítulo IV

# Ambientes de Desenvolvimento de Software e suas propostas de Integração de Ferramentas

## IV.1. Introdução

Existem, atualmente, ou encontram-se em desenvolvimento, vários Ambientes de Desenvolvimento de Software. Estes ambientes têm sido construídos segundo diversas abordagens e perspectivas de utilização, que acabam determinando o tipo de arquitetura e os mecanismos de integração de ferramentas adotados. Não é finalidade deste capítulo apresentar uma lista completa dos ambientes existentes, mas representar aqueles que, de alguma forma, apresentam características marcantes com relação aos aspectos de integração. Nesta apresentação será dada uma maior ênfase para os mecanismos de integração existentes em detrimento de outras características dos ambientes.

## IV. 2 Modelos de Ambientes

A identificação de características e propriedades que permitam classificar ADSs tem sido tratada de forma distinta por Dart [Dart87], Penedo [Penedo88], Perry [Perry88], Lubars [Lubars89], Moura [Moura92] e Miyoshi [Miyoshi93].

Dart [Dart87] e Penedo [Penedo88] classificam ambientes segundo sua arquitetura e composição de serviços, que determinam qual o objetivo básico do ambiente. Perry [Perry88] classifica ambientes conforme sua constituição, porte e capacidade de trabalho. Lubars [Lubars89] visualiza ambientes sob o ponto de

vista do suporte ao usuário. Moura [Moura92] estendeu a classificação de Dart, Penedo, Perry e Lubars e analisou ambientes de acordo com sua funcionalidade, porte e suporte ao usuário, acrescentando novos aspectos para contemplar características de ambientes de desenvolvimento de software mais modernos. Miyoshi [Miyoshi93] trata a classificação de ambientes a partir da identificação de requisitos de qualidade essenciais para ADSs e os classifica a partir da identificação destes requisitos nos ambientes.

Na classificação de Moura [Moura92] um ADS pode ser classificado considerando-se quatro aspectos fundamentais: Arquitetura, Suporte ao Usuário, Escala e Características gerais.

A classificação baseada na Arquitetura considera cinco tipos distintos de ambientes. *Ambientes Centrados em Sistemas Operacionais* são aqueles influenciados diretamente pelo sistema operacional e controlados pelo usuário através de uma linguagem de comandos. *Ambientes Centrados em Linguagens de Programação* fornecem suporte através de uma linguagem de programação particular e possuem um conjunto de ferramentas apropriadas para o uso desta linguagem. *Ambientes Centrados em Métodos* privilegiam métodos específicos para o desenvolvimento e mantêm um nível de gerenciamento para estes métodos. *Ambientes Centrados em Configuração* permitem sua configuração pelo usuário para atender necessidades específicas de um dado contexto. *Ambientes Centrados em Trabalho Cooperativo* suportam o desenvolvimento cooperativo de software.

Na classificação baseada no suporte oferecido ao usuário devem ser enfocados aspectos relativos à construção do produto, avaliação da qualidade do produto ao longo do desenvolvimento e gerência do processo de desenvolvimento. A construção do produto envolve suporte para todo o ciclo de vida, atividades "clericais"[1], reutilização, inteligência localizada nas ferramentas e re-engenharia. A avaliação da qualidade do produto envolve a existência de suporte para o estabelecimento de metas de qualidade, avaliação da qualidade ao longo do projeto, testes, estimação e medição da confiabilidade do produto. Quanto a gerência do processo, envolve a existência de suporte para o planejamento,

[1] atividades inerentes a qualqwuer processo de desenvolvimento (Por exemplo: edição de textos, formatação de diagramas, etc.)

acompanhamento do projeto e gerência de configuração.

A classificação baseada na escala, estendendo a proposta de Perry, tratando aspectos relacionais do ambiente e determinando o tipo de ambiente a partir de cinco tipos distintos: Indivíduo, Família, Cidade, Estado e Comunidade de Estados. Ambientes do tipo *Indivíduo* são pequenos e fornecem um conjunto de ferramentas para o desenvolvimento. Normalmente são ambientes de programação. Ambientes do tipo *Família* além de possuirem características do modelo Indivíduo, possuem características que permitem a interação de pequenos grupos de programadores. O *Modelo Cidade* estende as características do modelo Família, de forma a permitir que grupos maiores de desenvolvedores possam interagir. Ambientes do tipo *Estado* são ambientes de desenvolvimento genéricos que podem ser instanciados de acordo com a necessidade de um dado projeto. Ambientes *Comunidade de Estados* incluem as facilidades do modelo Estado e contêm facilidades que permitem a integração de ferramentas externas.

Finalmente, na classificação a partir das características gerais de ambientes são considerados aspectos relacionados à facilidade de uso, facilidade de aprendizado, suporte a usuários inexperientes, grau de automação, valor do ambiente e inteligência global.

A seguir serão descritos alguns ambientes de desenvolvimento, destacando-se os seus conceitos de integração de ferramentas. (Estes ambientes são classificados segundo a taxonomia proposta por Moura)

## IV.2.1 O Modelo ADAGE

### IV.2.1.1 Visão Geral

ADAGE (ADAptable Graph-based Environment) [Giavitto89a] [Giavitto89b] [Giavitto90] é um ambiente de desenvolvimento de software que se utiliza do conceito de grafos para a representação das informações. O meta

ambiente ADAGE deve ser instanciado para produzir modelos de ambientes de desenvolvimento de software que podem ser completados com ferramentas específicas, de forma a se obter o ambiente final. Este ambiente instanciado e completo poderá, então, ser utilizado pelo usuário final para o desenvolvimento do produto.

O meta-ambiente ADAGE se utiliza de um enfoque orientado a objetos e foi desenvolvido em sistema Unix. Executa em SUN 3 e SUN 4 sobre OS 3.4 ou 4.0 e estações DEC 3100 sobre Ultrix.

ADAGE faz parte do projeto ESPRIT(Meteor). Um primeiro protótipo foi concluído em 1988 no laboratório de Marcoussis, CGE Corporate Research Center, em Marcoussis, França.

### IV.2.1.2 Representação da Informação e Integração das Ferramentas

O meta-modelo ADAGE é baseado no conceito de grafos estruturados tipados tratados através de uma linguagem específica, denominada GDL (Graph Description Language). Esta característica torna o meta ambiente ADAGE configurável, pois através da GDL consegue-se instanciar o meta-modelo de forma a obter um ambiente definido para um domínio de aplicação específico (ou para suportar um determinado método); extensível, no sentido de que é possível ao meta-modelo refletir as alterações e evoluções dos métodos representados através de novas descrições em GDL; e flexível, devido ao meta-modelo suportar a integração de estruturas de informação e organizações existentes.

O meta-modelo ADAGE é suportado pela GDL. O propósito de uma definição GDL é modelar o significado dado pelo usuário final aos dados gerenciados pelo ambiente. Na realidade este significado é uma abstração do mundo real, onde o usuário está mapeando seus objetos no conjunto de recursos básicos oferecidos pelo sistema.

O componente básico da GDL é um nó. Um nó inclui um grafo, o qual representa sua interação com o ambiente. Além disto, um nó possui propriedades

básicas que são descritas através de seus atributos. Podemos observar, neste ponto, que um nó é formado por duas regiões: uma contendo seus atributos, descrevendo suas propriedades básicas, e uma outra contendo ligações com outros nós, região esta que formará o grafo propriamente dito.

A estrutura de um nó é descrita através de um *tipo*. O tipo de um nó descreve os atributos ligados a este nó, os vértices que podem aparecer no grafo e as relações entre estes vértices. No meta-modelo ADAGE, o *tipo* é representado também por um nó, definido como um nó-tipo, permitindo que um nó-tipo seja derivado de um nó. Esta definição permite o aparecimento de especialização, ou herança simples. Nós são utilizados pelo ADAGE para representar qualquer tipo de objeto de software e abstração.

O mecanismo de estruturação das entidades de ADAGE é realizado a partir da consideração de que os vértices de um grafo são nós. Este tipo de estruturação se torna mais geral do que uma simples relação hierárquica: ela pode ser usada para representar uma decomposição funcional ou para representar relacionamentos de agregação mais complexos, além de possibilitar que um mesmo nó possa aparecer em mais de um grafo.

Para gerenciar e controlar este modelo de representação o meta ambiente ADAGE conta com algumas ferramentas auxiliares e bibliotecas específicas, das quais são retirados serviços básicos para a parte que trata da interface com o usuário e a parte que trata do sistema de informação. Estes serviços, cuja estrutura pode ser vista na figura IV.1, são classificados segundo o padrão abaixo:

. Bibliotecas Genéricas: utilizadas no nível mais baixo. Neste ponto encontramos PRESTO [Giavitto89b], que introduz o conceito de persistência na linguagem C++ e SpokeWindows, que assegura um padrão gráfico baseado em X-Windows ou em SunView;

- Bibliotecas Centrais: um nível acima das bibliotecas genéricas, provocam o tratamento das entidades de ADAGE. L3 (Low Level Library) trata da implementação das entidades GDL em objetos manuseados por PRESTO,

tais como armazenamento de atributos e de grafos. Assegura também que algumas restrições sejam satisfeitas, como por exemplo controlando a integridade referencial: quando um vértice é retirado de um grafo, todos os relacionamentos dependentes deste vértice são também retirados. ADAGIO (Adage Input Output) acrescenta à SpokeWindows funções que permitem ao ADAGE representar seu modelo de grafos na forma gráfica;

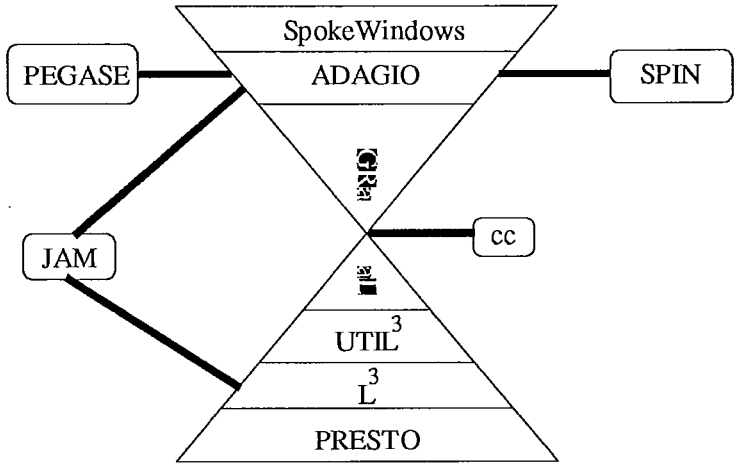



Figura IV.1 - Estrutura do Ambiente ADAGE [Giavitto90]

- Compiladores: são responsáveis pela transição das informações dos níveis mais altos para os mais baixos. O compilador incremental GDL transforma uma descrição textual de entidades ADAGE em dados manuseados pelo ambiente, enquanto UTIL$^3$ (User-action Triggering In L3) controla as restrições de integridade das entidades;

- Ferramentas Genéricas: são as ferramentas de mais alto nível e formam o meio de comunicação com o ambiente ADAGE. *Graal* é a shell padrão do ADAGE e é utilizada para gerenciar os dados e prover uma interface gráfica com o usuário. *JaM* é uma ferramenta utilizada para gerenciar espaços de trabalho. Estes espaços são definidos em ADAGE como sendo o conjunto de entidades envolvidas numa tarefa.

Podemos incorporar ao ambiente ADAGE ferramentas pré-existentes, ou seja, ferramentas que já estejam disponíveis mas que não foram construídas especificamente para o ADAGE, ou então utilizar os serviços disponíveis no ambiente para construir novas ferramentas.

As ferramentas pré-existentes são acrescentadas ao ambiente através da construção de um tradutor que possibilite executar um mapeamento das informações necessárias a ferramenta. Este tradutor pode ser construído a partir de Graal e UTIL3.

Quanto às novas ferramentas, elas devem adotar a representação de dados do ADAGE, onde a integração é conseguida através do compartilhamento da representação. Isto pode ser feito utilizando-se L3 , UTIL3 e ADAGIO. O desenvolvimento destas ferramentas, caso a eficiência não seja um fator crítico, pode ser realizado através de GRaal.

O ambiente gerado pelo meta-modelo ADAGE é construído a partir da manipulação dos dados descritos através da GDL. Neste ambiente são incorporados serviços oferecidos pelo meta-modelo que farão o trabalho de manter e gerenciar os objetos ao longo da utilização do ambiente.

### IV.2.1.3 Comentários a respeito do ambiente ADAGE

Podemos observar que o meta-ambiente ADAGE possui características que se enquadram tanto no enfoque de Armazenamento de Informações (o processo de incorporação de novas ferramentas ao ambiente tem como ponto fundamental o compartilhamento de informações) quanto no Modelo Orientado a Vistas (as informações são agrupadas em conjuntos de entidades bem definidas denominados espaços de trabalho), descritos na seção II.2. Além disto, observa-se que o meta-ambiente foi construído utilizando-se o paradigma da orientação a objetos, utilizando a linguagem C++, e incorporando novas características à linguagem (como exemplo principal a persistência dos objetos em C++).

Deve ser ressaltado o processo de desenvolvimento realizado no ambiente ADAGE: a partir de produtos existentes foi realizado um processo de transformação, de tal forma que as características necessárias à construção do ambiente fossem incorporadas a estes produtos. Se por um lado o processo de desenvolvimento pode, a primeira vista, parecer amarrado à estrutura do ambiente computacional em que o mesmo está sendo realizado, verifica-se que pelo montante do trabalho realizado aliado com os refinamentos técnicos necessários ao desenvolvimento destas atividades, conseguiu-se obter resultados satisfatórios em um tempo relativamente pequeno, segundo relata [Giavitto90].

## IV. 2.2 O Modelo SLCSE

### IV. 2.2.1 Visão Geral

O SLCSE ( Software Life Cycle Support Environment) [Strelich88] é um ADS construído para ser utilizado em ambientes VAX/VMS, a partir de uma interface com o usuário comum e consistente que acessa um amplo conjunto de ferramentas para desenvolvimento, construídas de forma a suportar as atividades específicas de cada fase do ciclo de vida. A idéia principal do SLCSE é utilizar os recursos existentes e disponíveis no ambiente operacional no sentido de se obter um conjunto bem definido destes recursos e dispositivos e aplicá-los na construção de ambientes específicos para o desenvolvimento de um produto em particular.

Segundo [Strelich88], se nos basearmos na classificação de [Dart87], podemos considerar o SLCSE como um quinto tipo de ambiente, definido por ele como sendo uma estrutura de ambiente, devido ao fato de podermos ajustar cada ambiente de desenvolvimento às necessidades de um projeto de software.

### IV.2.2.2 Representação da Informação e Integração das Ferramentas

A representação das informações e a integração das ferramentas no SLCSE é realizada através das estruturas existentes no ambiente operacional VAX/VMS.

Todo o processo de utilização das ferramentas existentes é feito pelo usuário, diretamente, através da interface com o usuário. Neste ponto, o usuário deve assumir funções bem definidas no escopo da fase do ciclo de desenvolvimento em que a ferramenta se aplica. Estas funções refletem a atividade que o usuário está desempenhando e são elas que classificam, ou separam, as ferramentas em categorias. Esta classificação restringe ao usuário o acesso a informações que não pertençam à fase de desenvolvimento em questão e provocam o uso disciplinado das ferramentas, evitando que o usuário viole as limitações e restrições impostas a cada fase do processo de desenvolvimento.

Para o armazenamento das informações e integração das ferramentas o ambiente SLCSE utiliza uma base de dados baseada no modelo de Entidade-Relacionamento. Esta base de dados é vista pelo SLCSE como um local comum de armazenamento das informações pertencentes a um certo projeto em desenvolvimento, e como meio principal de troca de informações entre as ferramentas. O modelo de dados é desenvolvido a partir do modelo de Entidade-Relacionamentos e a descrição dos dados juntamente com as restrições de integridade que fornecem (principalmente, o comportamento das entidades existentes) é feita através da construção de sub-esquemas.

Ferramentas podem ser acrescentadas ao SLCSE sob duas formas: através da construção de ferramentas que se utilizem diretamente dos recursos disponíveis no ambiente, ou então, através da construção de interfaces para ferramentas já existentes.

A construção de ferramentas para o SLCSE utiliza-se de padrões de interface com o usuário pré-definidos, denominados MOO e Winnie. MOO (Menu Operations Organizer) proporciona capacidade de controle de janelas através da definição da estrutura e formas de percurso em cardápios baseado nas entradas do usuário. Winnie é um conjunto de procedimentos que suportam a definição e o gerenciamento de janelas para sistemas de terminais orientados a caracteres, como aqueles compatíveis com os terminais VT100.

A base de dados do SLCSE é construída sobre um sistema de

gerenciamento de base de dados disponível comercialmente, o SMARTSTAR. Sua utilização se dá através da definição do modelo de Entidades-Relacionamentos da base de dados de projeto em SDL (Schema Definition Language). A linguagem SDL é baseada em Ada e permite a definição de sub-esquemas, tipos de entidades, relacionamentos e atributos. As definições do modelo escritas em SDL são transladadas para SQL (Structured Query Language) e, posteriormente, processadas pelo SMARTSTAR. O interfaceamento das ferramentas pode ser feito diretamente, através de um conjunto de procedimentos escritos em SDL e denominados ERIF (Entity-Relationship Interface), ou então a partir da utilização da DCL (Digital Command Language), linguagem disponível no ambiente operacional. Ferramentas que se utilizam de ERIF são conhecidas como ferramentas diretamente acopladas. Ferramentas que utilizam a DCL são denominadas indiretamente acopladas.

Esta base de dados é representada por dois repositórios distintos que armazenam informações relevantes ao ambiente. A base de dados de Projeto é o repositório central das informações obtidas ao longo do ciclo de vida. Ele serve como um mecanismo de integração de dados para as ferramentas, permitindo que elas possam se comunicar e trocar informações. A base de dados de infraestrutura, que é privada ao SLCSE, contém as informações necessárias à operação e controle do ambiente.

Ferramentas podem ser incorporadas ao ambiente considerando-se os seguintes aspectos [Penedo92]:

- Diretamente acopladas: ferramentas que acessam diretamente o banco de dados através da interface ERIF. Ferramentas desenvolvidas voltadas para o modelo representado são fortes candidatas a serem diretamente acopladas;

- Indiretamente acopladas: ferramentas que devem utilizar um interpretador de comandos DCL para acessar a base de dados;

- Fracamente acopladas: ferramentas que não utilizam a base de dados de projeto do SLCSE, mas ainda assim são utilizadas no ambiente;

- Infra-acopladas: ferramentas que utilizam a base de dados de infraestrutura do SLCSE;

- Não infra-acopladas: ferramentas que não utilizam a base de dados de infraestrutura do SLCSE;

- Dependentes: ferramentas que são totalmente dependentes da base de dados de projeto e a base de dados de infra-estrutura para manter seus dados, e não possui nenhuma base de dados particular.

### IV.2.2.3 Comentários a respeito do ambiente SLCSE

O SLCSE é a tentativa de padronização da construção de ambientes de desenvolvimento de software em ambientes VAX/VMS. As soluções propostas e adotadas são desenvolvidas dentro de um ambiente operacional específico, provocando uma forte dependência do hardware .

A utilização de um enfoque tradicional para o armazenamento e tratamento das informações impede que algumas características inerentes a ambientes de desenvolvimento sejam representadas, como por exemplo o comportamento semântico dos objetos, o tratamento dos objetos como entidades que representam no domínio de um determinado método e o armazenamento de objetos complexos.

Os padrões adotados para interface com o usuário deixam a desejar no instante em que são desenvolvidos para a utilização de terminais específicos, sem capacidade gráfica e dispositivos de interação apropriados. Além disto, o padrão de interface é quebrado com a incorporação de ferramentas que não pertençam ao ambiente e que possuam um forte relacionamento interativo com o usuário. Esta solução é, também, adotada pelo ambiente de interface MS-Windows e não apresenta bons resultados.

## IV.2.3 O SIPS

### IV.2.3.1 Visão Geral

O SIPS ( Sistema Integrado para Produção de Software) [Tsukumo85] é caracterizado como um meta-sistema de especificação de sistemas, ou seja, permite ao usuário definir Linguagens de Especificação de Sistemas (LES) adequadas às diversas áreas de aplicação. O SIPS é um sistema extensível para produção de software, cujo objetivo é suportar o desenvolvimento integrado de software. A idéia central do ambiente SIPS é definida através da utilização do Modelo de Representação de Objetos (MRO) [Traina86], que possibilita a definição e a modelagem dos objetos envolvidos num determinado método, permitindo ao ambiente reconhecer e tratar estas informações adequadamente, formando um conjunto de definições armazenadas numa base de dados compartilhada e tratadas pelas várias ferramentas existentes no ambiente.

O ambiente SIPS foi desenvolvido no Centro Tecnológico para Informática - CTI, em Campinas, em conjunto com a UNICAMP e a USP e conta com um protótipo, disponível em ambiente PC compatível e ambiente VAX.

### IV.2.3.2 Representação da Informação e Integração das Ferramentas

O ambiente SIPS pode ser considerado como sendo composto de dois subsistemas principais: SIPS-META e o SIPS-SISTEMA. Na figura IV.2 podemos visualizar a estrutura do ambiente. O SIPS-META permite a um especialista em métodos definir as características básicas que descrevem o método gerando Linguagens de Especificação de Sistemas (LES) que são armazenadas numa base de dados denominada Base-Meta. O SIPS-SISTEMA permite que um desenvolvedor da aplicação utilize o SIPS para desenvolver seu produto, a partir dos métodos descritos e armazenados na Base-Meta. As informações geradas por este processo de desenvolvimento são armazenadas na Base-Sistema e podem ser compartilhadas pelas ferramentas existentes para suportar as fases de desenvolvimento.

Os dados para o ambiente SIPS são modelados através do Modelo de Representação de Objetos (MRO) [Traina86]. Este modelo baseia-se na teoria dos grafos. Seus elementos básicos são nós e ligações, compondo um dígrafo. Os nós que compõem o dígrafo são identificados e categorizados, formando assim um conjunto onde seus elementos podem ser identificados e classificados dentro de uma única categoria de um conjunto de categorias existentes. Às ligações, por sua vez, são associados pesos, cujos valores permitirão separá-las segundo um determinado conjunto de classes. O dígrafo obtido com estas extensões satisfaz as necessidades de representação de informação do SIPS. [Jino85]

Uma associação é feita entre os dados que um dígrafo conterá e seus elementos: os nós representam objetos do mundo real e os arcos representam associações entre estes objetos. A cada objeto existente no modelo é associado um identificador e um tipo, de forma que seja possível classificar estes objetos. A cada associação é atribuído um peso e um tipo de relacionamento. Devido a esta caracterização objetos e relacionamentos são agrupados de acordo com seus tipos. Isto permite que todos os relacionamentos e objetos sejam armazenados em apenas um arquivo de relacionamentos e um arquivo de objetos, devido a uniformidade de sua estrutura interna que é independente do tipo da informação armazenada.

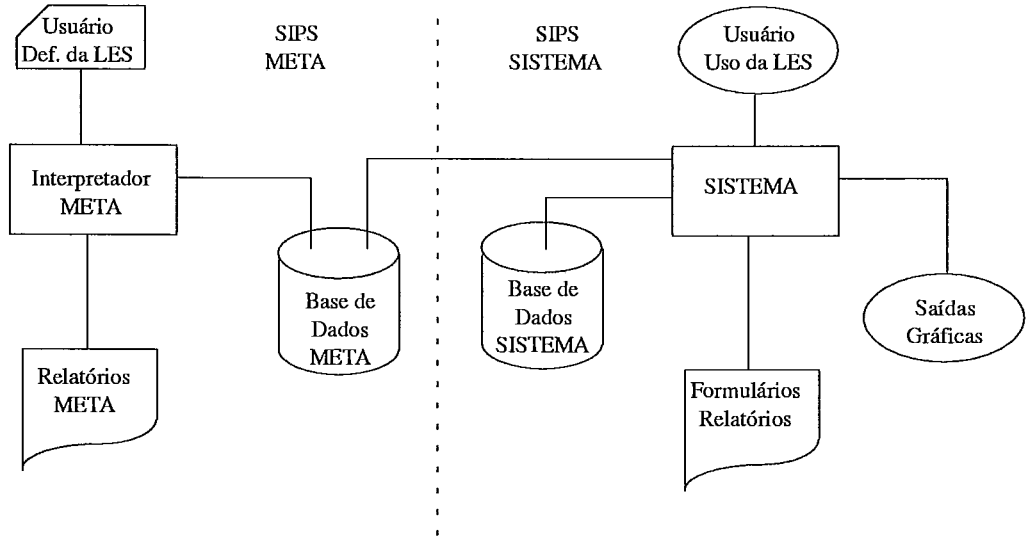



Figura IV.2 - Estrutura contextual do SIPS (Adaptado de [Tsukumo85])

O controle e gerenciamento deste modelo é feito a partir de um conjunto de rotinas primitivas responsáveis por todo o acesso às bases Meta e Sistema. Este conjunto de primitivas torna uniforme o acesso à informação e facilita a incorporação de novos métodos ao ambiente. A incorporação de novas ferramentas ao ambiente é feita a partir da descrição do método a que pertencem, através da meta linguagem do SIPS. Esta linguagem provê comandos para descrever tipos de objetos, tipos de relacionamentos, tipos de propriedades, juntamente com a sintaxe pertencente a estes elementos. A partir desta definição, uma ferramenta pode ser construída através da escrita de um procedimento que chama as rotinas primitivas e as rotinas que implementam a interface com o usuário. A informação semântica responsável pelo funcionamento da ferramenta de acordo com o método, é obtida através das primitivas que acessam a Base-Meta.

### IV.2.3.3 Comentários a respeito do ambiente SIPS

O ambiente SIPS utiliza-se de um banco de dados compartilhado. Todo o processo de troca de informações é centralizado na base de dados do ambiente, no caso representada por duas bases distintas: a Base-Meta e Base-Sistema. É importante notar que embora na nomenclatura de definição do ambiente se utilize de palavras como tipo de objeto, objeto, classe e o próprio modelo de representação ser chamado de Modelo de Representação de Objetos, o ambiente SIPS não utiliza o conceito de orientação a objetos em sua construção e utilização.

Uma questão que deve ser levantada aqui é a respeito da representação e modelagem dos métodos a partir do modelo proposto. Conforme descrito em [Jino85] e [Tsukumo85], o método Análise Estruturada é modelado conforme a proposta do ambiente. Embora o MRO consiga representar mais informaçõe semânticas do que o MER, não se observa a representação de restrições de integridade dos objetos (estas restrições representam o comportamento do objeto no método em questão) e muito menos de informações sintáticas específicas do método que interferem nos objetos existentes. Este tipo de deficiência foi detectado na modelagem de FEGRES [Trotta90] e solucionado através da adoção de novas características ao MER para que o mesmo pudesse representar estes

comportamentos. Além disso, a representação e o tratamento destas informações ficam dependentes da própria estrutura das bases de dados, baseadas no modelo relacional.

## IV.2.4 O Modelo Centaur

### IV.2.1 Visão Geral

CENTAUR [Borras88] é um ambiente interativo genérico utilizado para a configuração de ambientes específicos de apoio ao desenvolvimento em uma dada linguagem de programação. A partir da descrição da sintaxe e da semântica da linguagem que se deseja trabalhar, CENTAUR gera um conjunto de ferramentas para suporte ao desenvolvimento nesta linguagem, incluindo tradutores, editores e depuradores.

CENTAUR compõe o projeto ESPRIT-348 e foi escrito, essencialmente, em LISP e PROLOG, utilizando como padrão de interface com o usuário, para o meta-ambiente e o ambiente gerado, o X-Windows.

O ambiente CENTAUR é composto por três partes:

- uma base de dados de componentes, que provê uma padronização de representação e acesso aos objetos formais e persistentes;

- uma máquina lógica que é utilizada para executar as especificações formais descritas no ambiente;

- uma interface com o usuário orientada a objetos que facilita ao usuário o acesso à funcionalidade do ambiente.

### IV.2.4.2 Representação da Informação e Integração das Ferramentas

A arquitetura do ambiente CENTAUR é estruturada a partir de três classes de funções. Estas funções são responsáveis, cada uma delas, por determinadas características do ambiente. Funções existentes no núcleo tratam da representação e manipulação de objetos estruturados. Num nível intermediário, encontramos funções responsáveis por tratar a sintaxe e a semântica das linguagens que serão descritas. Para que possa ocorrer uma efetiva comunicação do sistema com o usuário, existem funções responsáveis por tratar esta comunicação.

As funções do núcleo do ambiente lidam com objetos estruturados, representados por árvores de sintaxe abstrata. O núcleo é construído através de um máquina abstrata, chamada VTP ("Virtual Tree Processor") que trata os aspectos sintáticos. A semântica é tratada por uma máquina lógica e é representada por um interpretador PROLOG. Estes componentes do núcleo de CENTAUR atuam como co-rotinas e são conectados a partir de esquemas de interface que contém serviços específicos para realizar sua interconexão. Primitivas de controle são utilizadas para realizar a comunicação, em ambos os sentidos, entre VTP e o Prolog. Primitivas de transformação são utilizadas para realizar a transformação[2] de tipos entre VTP e o Prolog.

O nível de especificação provê a capacidade de se descrever a sintaxe concreta e abstrata, juntamente com a semântica das linguagens. Existem dois níveis de descrição. Segundo Borras [Borras88], a sintaxe é descrita em METAL , que é um formalismo desenvolvido para o sistema MENTOR, e representada por uma coleção de regras gramaticais, com anotações que especificam que árvore sintática deve ser sintetizada. A semântica da linguagem de programação, que poderia ser representada a partir de diversos formalismos, é escrita em TYPOL e compilada em PROLOG.

O controle da interação e apresentação de informações é realizado pelas funções de interface com o usuário. Além dos objetos normalmente encontrados

---

[2] Coercion, no original

em sistemas de interface, uma característica interessante de CENTAUR é possuir "frameworks" para a definição de novos elementos de interface. Estes objetos são definidos a partir da descrição de sua aparência e comportamento. A aparência dos objetos é descrita a partir de um conjunto de primitivas em Le-LISP, chamado AIDA. Objetos gráficos, se precisarem, podem ter seu comportamento descrito em ESTEREL, que gera descrições em LISP.

#### IV.2.4.3 Comentários à respeito do ambiente CENTAUR

O ambiente CENTAUR permite a obtenção de ambientes integrados de apoio à programação. Utiliza uma representação formal para a descrição das linguagens e esta representação é utilizada pelas ferramentas para executarem seu trabalho.

Como sua aplicação se dá apenas na fase de programação, não há como utilizá-lo em todo o ciclo de desenvolvimento, embora a partir da utilização do formalismo definido seja possível realizar a integração de CENTAUR com outros ambientes, tendo em vista que as informações geradas pelas ferramentas representam código fonte em uma determinada linguagem de programação, que podem ser aproveitados diretamente para a construção de uma aplicação.

### IV.2.5 O Modelo AD\Cycle

#### IV.2.5.1 Visão Geral

AD/Cycle [Hoffnagle90] representa o enfoque de desenvolvimento de aplicações da arquitetura de sistemas de aplicação (SAA) da IBM. Foi projetado para suportar o desenvolvimento de software de aplicação e ajudar na diminuição da fila de desenvolvimento dos produtos de software necessários, através da utilização de técnicas modernas de desenvolvimento automatizadas. AD/Cycle facilita a construção de ferramentas CASE, devido à definição de uma infra-

estrutura comum de desenvolvimento entre as fases do ciclo de vida, incluindo o gerenciamento do próprio ciclo, que incluem ferramentas integradas, facilidades de armazenamento e interface com o usuário. De acordo com Merlyn [Merlyn90], AD/Cycle é um conjunto de planos, interfaces e padrões, cujo objetivo é permitir a integração de ferramentas individuais num ambiente de desenvolvimento compreensível, que proveja definições para armazenamento, manipulação e reutilização de todas as informações relevantes para o processo de construção do produto de software.

Segundo Mercurio [Mercurio90], os objetivos principais do AD/Cycle são:

- fornecer aos desenvolvedores e clientes uma tecnologia avançada de desenvolvimento de produtos;
- permitir o controle e o gerenciamento centralizados das informações relativas ao desenvolvimento;
- estabelecer uma arquitetura aberta e um conjunto básico de serviços para integração de ferramentas ao longo do ciclo de vida;
- complementar e estender a arquitetura SAA.

### IV.2.5.2 Representação da Informação e Integração das Ferramentas

A arquitetura do AD/Cycle, que pode ser vista na figura IV.3, fornece serviços responsáveis por tratar os seguintes aspectos no contexto de desenvolvimento:

- Interface com o Usuário: as ferramentas possuem o mesmo padrão de interface com o usuário e mantêm a consistência e integridade de apresentação, a partir da utilização do Presentation Manager [Artim90];
- Gerenciamento do Trabalho: é possível controlar a ordem de execução das

ferramentas no contexto do desenvolvimento, permitindo ao usuário escolher, a partir de uma lista de possíveis atividades, qual ele deseja executar naquele momento. O gerenciamento do processo é controlado a partir do produto Application Development Project Support [Chroust90], que permite ao usuário definir um modelo do processo de desenvolvimento da aplicação, guardar este modelo, e passar a utilizá-lo como o guia do desenvolvimento;

- Ferramentas para Desenvolvimento: estão disponíveis no ambiente ferramentas que podem ser aplicadas a fases e atividades específicas do ciclo de desenvolvimento;

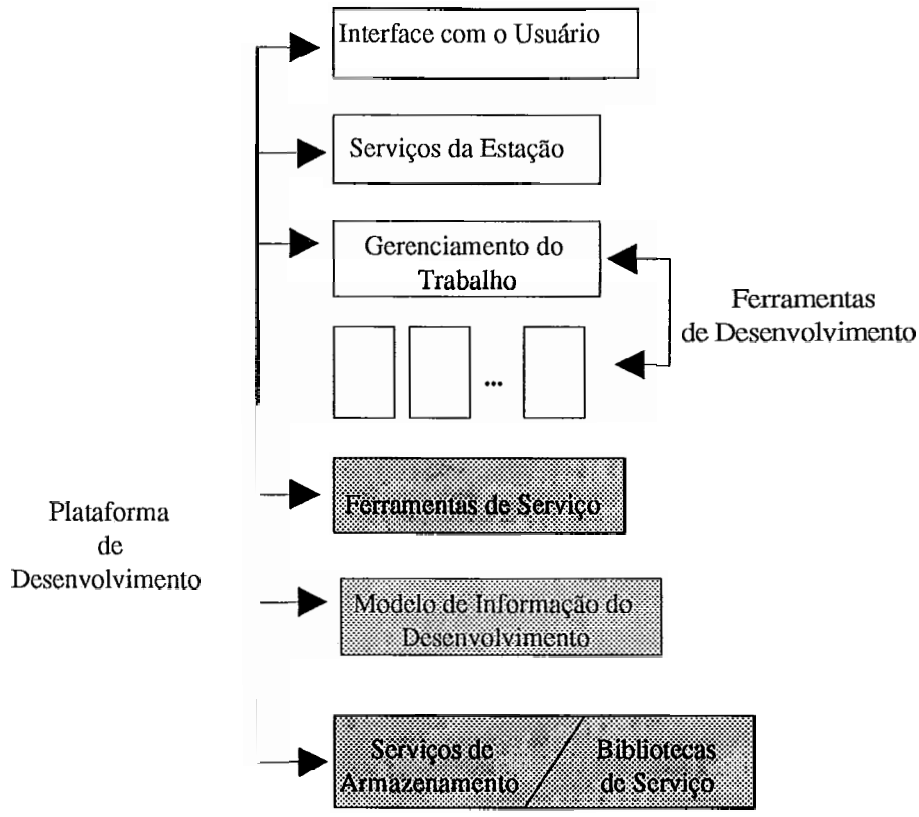



Figura IV.3 - Arquitetura do AD/Cycle [Mercurio90]

- Ferramentas de Serviços Gerais: ferramentas que possibilitam a execução de tarefas comuns a todas as fases do ciclo de desenvolvimento e ao contexto do próprio ambiente. Nesta categoria incluem-se ferramentas para tratamento de erros e envio de mensagens;

- Serviços de Armazenamento e Biblioteca de Serviços: no contexto do AD/Cycle todas as informações relevantes a um projeto são armazenadas numa base de dados centralizada, descrita segundo um modelo de entidades e seus relacionamentos e gerenciada pelo Repository Manager [Sagawa90]. Complementando o serviço de armazenamento, encontram-se disponíveis bibliotecas de serviços que provêm suporte para o versionamento e a construção automática de aplicações. Este serviço é fundamental para o modelo, pois através dele é que ferramentas poderão compartilhar informações.

### IV.2.5.3 Comentários a respeito do ambiente AD\Cycle

O ambiente AD/Cycle define um conjunto de diretrizes que deveriam ser seguidas no contexto do desenvolvimento, de forma a se obter um nível de qualidade e produtividade na arquitetura de desenvolvimento IBM. Esta talvez seja a característica fundamental do ambiente: definir uma filosofia de trabalho e oferecê-la aos desenvolvedores.

Embora exista um conjunto de funcionalidades disponíveis para o desenvolvimento de ferramentas e sua integração ao ambiente, a força do modelo reside no repositório de dados. É nele que todas as informações serão armazenadas e tratadas. Isto obriga que as ferramentas sejam construídas para entender o modelo de dados armazenado, e dificulta a integração de ferramentas que não tenham sido implementadas para utilizar os recursos disponíveis.

## IV.2.6 O Modelo Graspin

### IV.2.6.1 Visão Geral

GRASPIN [Manucci89] é um ADS com o objetivo de suportar as fases de análise e projeto do processo de desenvolvimento. Faz parte do projeto ESPRIT-125 e foi desenvolvido de 83 a 89. Oferece um conjunto completo de métodos e ferramentas CASE, com tratamento gráfico, associadas que conseguem abordar aspectos sintáticos e semânticos do desenvolvimento. Uma das características marcantes em GRASPIN é sua capacidade de descrição de formalismos e adaptação a novos enfoques de desenvolvimento. O protótipo do ambiente suporta formalismos para trabalhar em linguagem Ada, análise estruturada e modelo de Entidade-Relacionamento.

### IV.2.6.2 Representação da Informação e Integração das Ferramentas

GRASPIN considera que um ambiente integrado de desenvolvimento constrói e mantém produtos de software como uma coleção relacionada de objetos estruturados. Esta integração é obtida através da utilização de uma representação interna comum e única para os documentos [Manucci89], baseada numa estrutura de grafos. Grafos são compostos por nós e estes nós são ligados a outros nós, constituindo uma estrutura de relações significativas entre eles.

A customização do ambiente se dá a partir da descrição dos vários formalismos existentes no contexto do desenvolvimento. Considerando um esquema de desenvolvimento convencional, cada fase do processo de desenvolvimento possui um conjunto de atividades e linguagens bem definidas e, algumas vezes, disjuntas, como é o caso do desenvolvimento estruturado, que necessitam ser descritas. A descrição destes formalismos é feita a partir da definição da sintaxe concreta e abstrata, juntamente com a semântica. A sintaxe abstrata representa a estrutura de um programa e as relações existentes entre as partes que o compõem. A sintaxe concreta especifica a aparência dos programas. A semântica, que especifica o comportamento de um programa, pode ser

determinada a partir de três técnicas específicas, que são as rotinas de ação, gramática de atributos e semântica natural, é construída no modelo utilizando-se um enfoque baseado no modelo CENTAUR [Borras88]. Este tipo de abordagem, aliada a presença de ferramentas que permitem a descrição de novos formalismos, facilita o acréscimo de novas linguagens ao ambiente.

A arquitetura do ambiente, na qual as ferramentas se baseiam para funcionar, pode ser vista na figura IV.4. A máquina abstrata , conforme relata Manucci [Manucci89], define um tipo de dados abstrato que é representado por um grafo direcionado. Nesta máquina, encontram-se as rotinas primitivas necessárias à manipulação das estruturas. Estas estruturas são representadas internamente através de árvores sintáticas, acrescidas de informações gráficas e semânticas representando os atributos dos nós e as ligações entre eles.

O processador de comandos é responsável por traduzir os comandos do usuário. A associação entre os comandos e sua funcionalidade é feita por um documento interno, escrito a partir da estrutura de grafos definida acima. Este documento é chamado de árvore de comandos e permite a reconfiguração da funcionalidade do ambiente.

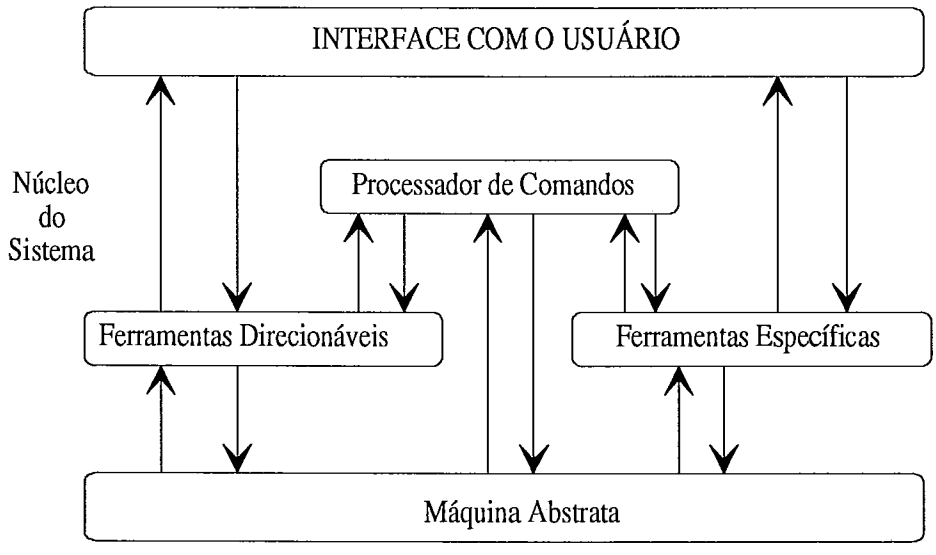



Figura IV.4 Arquitetura de GRASPIN [Manucci89]

As ferramentas, no contexto de GRASPIN, são todas elas implementadas levando-se em consideração as funções primitivas existentes na máquina abstrata. Estas ferramentas podem ser classificadas em específicas e direcionáveis. Ferramentas específicas são definidas pelo programador do sistema e se aplicam a documentos específicos. Ferramentas direcionáveis são de uso geral e permitem sua adaptação aos documentos.

O módulo de interface com o usuário permite a integração do ambiente sob o ponto de vista do usuário e garante a uniformidade e consistência da representação. As ferramentas que manipulam objetos gráficos possuem facilidades de layout automático dos diagramas, o que facilita bastante o trabalho de construção dos documentos.

### IV.2.6.3 Comentários a respeito do ambiente GRASPIN

Conforme pode ser observado, o ambiente GRASPIN [Manucci89] é um ADS que pode ser utilizado em fases distintas do ciclo de vida. Sua arquitetura e forma de tratamento e armazenamento da informação influenciam diretamente as ferramentas que serão incorporadas e integradas ao ambiente. Embora exista a definição de classes de ferramentas, estas classes não contemplam ferramentas que não tenham sido desenvolvidas para o modelo e, portanto, estas não podem ser incorporadas ao ambiente.

Os formalismos descritos são tratados como estruturas que possuem um conhecimento distribuído sobre seu comportamento. A descrição da semântica é feita através da associação de uma determinada função existente na máquina abstrata ao nó que necessite tal funcionalidade. Esta funcionalidade associada não pertence ao nó. É responsabilidade do ambiente controlar o comportamento destas estruturas e, inevitavelmente, esta responsabilidade é passada às ferramentas, que se tornam muito dependentes da forma de representação.

## IV.2.7 O Modelo ASPIS

### IV.2.7.1 Visão Geral

O sistema ASPIS ("Application Software Prototype Implementation System") [Puncello88] , projeto ESPRIT-401, explora a aplicação de técnicas de Inteligência Artificial a ambientes de desenvolvimento de software. Seu objetivo principal é permitir a especificação e projeto de produtos de software, facilitando a transição entre estas fases. Estão inseridas no ambiente, ferramentas que permitem a execução das atividades relativas às fases iniciais do desenvolvimento. Estas ferramentas, denominadas assistentes, são construídas de forma a possuir algum conhecimento incorporado, facilitando o trabalho nestas etapas.

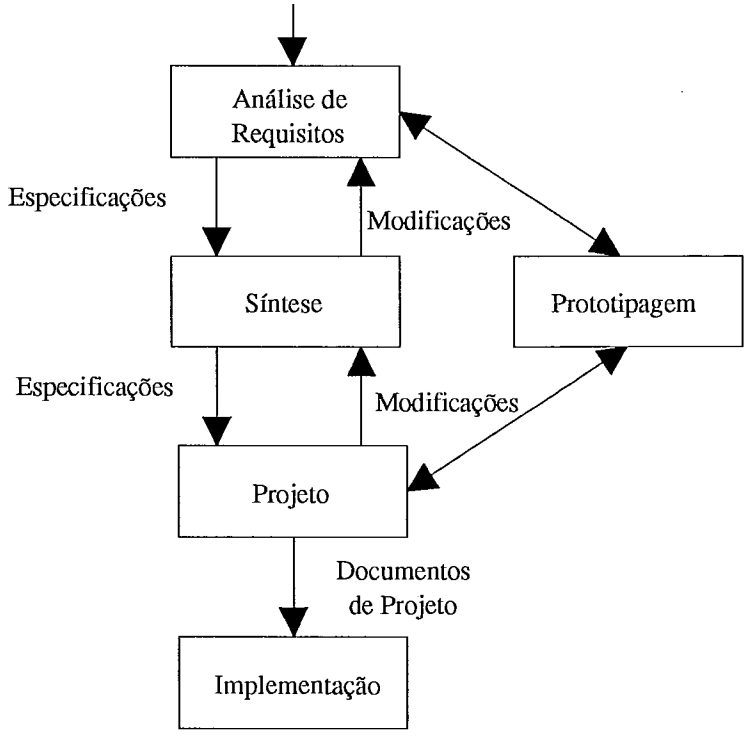



Figura IV.5 - Ciclo de Vida de ASPIS [Puncello88]

O enfoque de desenvolvimento de aplicações utilizando-se o ambiente baseia-se num modelo de ciclo de vida onde o uso intensivo dos assistentes, aliado a formalismos baseados em lógica, permitem a construção da especificação do

produto (figura IV.5) . A passagem da especificação para o projeto é precedida de um processo de transformação, a partir do formalismo adotado, das especificações em documentos de projeto. Este processo de transformação é chamado de síntese da especificação e pode ser realizado nos dois sentidos, permitindo que modificações solicitadas na fase de projeto sejam refletidas nos documentos de especificação.

### IV.2.7.2 Representação da Informação e Integração das Ferramentas

O modelo do ambiente ASPIS, segundo Puncello [Puncello88], possui quatro assistentes. Dois são baseados em conhecimento e podem ser utilizados diretamente pelos desenvolvedores da aplicação para a construção da especificação e o projeto. O conhecimento existente nestes dois ambientes descrevem o método utilizado para o desenvolvimento e as características do domínio da aplicação.

As especificações construídas pelos assistentes podem ser verificadas através do Assistente de Prototipagem, que permite ao desenvolvedor verificar a validade de suas especificações a partir da construção de protótipos.

Colaborando com o desenvolvimento, existe um quarto assistente que permite ao desenvolvedor reutilizar, para um determinado domínio de aplicação, especificações e projetos de outros aplicações já construídas em ASPIS.

O conhecimento associado aos ambientes de especificação e projeto podem ser divididos em duas categorias: conhecimento do método e conhecimento do domínio. O conhecimento sobre o método contém as regras de utilização de um determinado método de desenvolvimento. Nesta representação, encontramos a sintaxe do método, critérios de construção e heurísticas independentes do domínio de desenvolvimento. O conhecimento do domínio descreve como pode ser feita a utilização de um determinado método num domínio de aplicação, determinando as heurísticas a serem aplicadas. O conhecimento do domínio estende o conhecimento do método.

A estrutura básica de armazenamento do conhecimento é baseada em redes semânticas. Aos atributos dos nós desta rede são atribuídos sistemas de produção para representar o conhecimento associado a este nó. Esta forma de representação da informação é utilizada pelo ambiente para a integração das ferramentas, onde cada categoria de conhecimento é representada por uma rede semântica separada.

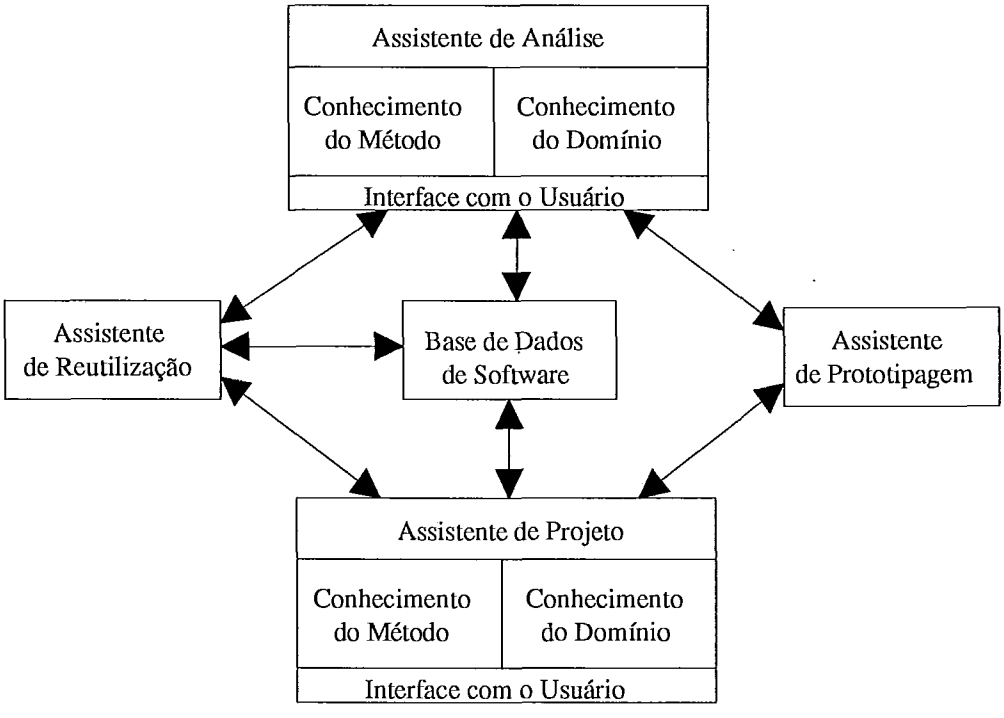



Figura IV.6 - Arquitetura de ASPIS [Puncello88]

ASPIS foi construído, inicialmente, para suportar especificação e projeto utilizando técnicas estruturadas e os diagramas gerados em cada atividade do desenvolvimento também são representados pelas redes semânticas.

O assistente de prototipagem utiliza as informações formais da especificação para poder gerar um protótipo para a aplicação. O assistente de reutilização utiliza estas informações para poder recuperar os documentos apropriados ao desenvolvimento. Este tratamento das informações formais de especificação dependem da possibilidade de se especificar formalmente os

documentos de análise e projeto.

### IV.2.7.3 Comentários a respeito do ambiente ASPIS

O ambiente ASPIS foi construído para apoiar as fases de especificação e projeto. O ciclo de vida do desenvolvimento é fixo e não é possível a inclusão de novas ferramentas ao ambiente sem o esforço de utilizar a estrutura de representação de conhecimento utilizada. A estrutura de representação do conhecimento e armazenamento das informações sobre o desenvolvimento foi modelada a partir da utilização de métodos estruturados e específicos, o que restringe seu uso para modelos de desenvolvimento que se utilizem de enfoques semelhantes de construção.

Por outro lado, a capacidade de reutilização e prototipagem do modelo está diretamente relacionada à capacidade de descrição dos documentos que exige um formalismo exagerado, o que pode dificultar o processo de desenvolvimento.

## IV.2.8 O Modelo IDeA

### IV.2.8.1 Visão Geral

IDeA [Lubars89] é um ambiente de desenvolvimento de software que integra diferentes aspectos do processo de desenvolvimento. O principal objetivo do ambiente é fornecer suporte baseado no conhecimento para reutilização de software e demais atividades inteligentes. A construção de IDeA é baseada em enfoques de projeto orientados a fluxo de dados e, devido a isto, metodologias estruturadas de análise e projeto são bem suportadas pelo ambiente.

O uso do ambiente é realizado pelo usuário a partir da definição da especificação de alto nível do sistema a ser construído. Esta especificação, normalmente, contém informações sobre as necessidades de entrada e saída da

aplicação e da função principal do produto. Partindo destas informações, IDeA procura no dicionário de dados definições de sistemas que melhor representem a descrição feita pelo usuário. Estas definições, juntamente com a descrição do usuário, permitem ao ambiente selecionar esquemas de projeto potenciais que são apresentados ao usuário, o qual escolhe aquele que melhor satisfaz ao projeto. O usuário tem liberdade de não utilizar este recurso do ambiente e partir para a especificação de um novo modelo de projeto, que será incorporado ao ambiente para posterior reutilização.

### IV.2.8.2 Representação da Informação e Integração das Ferramentas

O ambiente IDeA integra um conjunto de funcionalidades necessárias ao processo de desenvolvimento. Conforme pode ser visto na figura IV.7, encontram-se no ambiente um conjunto de funcionalidades que permitem o tratamento de atividades específicas e categorizadas de acordo com seu papel principal. Estas atividades, que estão integradas ao ambiente, podem ser classificadas, segundo Lubars [Lubars89], em:

- Suporte Burocrático ao Projeto: provê facilidades de edição e manipulação de representações gráficas de diagramas;

- Suporte de Interface com o Usuário: facilita a comunicação do usuário com o ambiente e possui recursos de comunicação tais como janelas, cardápios, gráficos e possibilidade de comunicação em linguagem natural;

- Suporte a Análise: suporta a avaliação da qualidade de projeto e controle de métricas, verificação e manutenção de consistência entre documentos e alguns procedimentos simples de validação;

- Suporte a Teste: facilita a construção de casos de teste, construção rápida de protótipos e avaliação de resultados de teste;

- Suporte Organizacional: acompanha o desenvolvimento do projeto e

verifica a observação das metas, objetivos e dependências de projeto;

- Suporte baseado em conhecimento: provê conhecimento especialista de projeto para desenvolvedores inexperientes, incluindo suporte para a localização e integração de fragmentos reutilizáveis ao projeto em desenvolvimento;

- Suporte Inteligente: acompanha a exploração e outras atividades criativas de projeto usando o planejamento realizado, refinamentos baseados em regras, propagação de restrições e transformações de projeto.

A comunicação do usuário com o ambiente é realizada a partir de quatro áreas distintas, ou janelas, de comunicação. Cada uma delas possui uma especificidade e representa aspectos diferenciados da informação. Estas janelas representam informações relativas à representação gráfica dos diagramas, transformações ocorridas no diagrama, definições existentes no diagrama e interação do usuário com o sistema.

IDeA possui um conjunto único de ferramentas para o desenvolvimento, dependentes da estrutura e arquitetura do ambiente. A consistência dos documentos é garantida ao longo do desenvolvimento devido a facilidades de propagação de modificações entre documentos pertencentes ao projeto em desenvolvimento.

A reutilização é bastante utilizada no ambiente, tanto a nível de desenvolvimento como na prototipação. A forma que o ambiente determina os elementos a serem reutilizados é através da verificação do domínio da aplicação ao qual o projeto se aplica. Estas informações são organizadas na base de conhecimento, a partir de uma estrutura hierárquica de representação que possui mecanismos de herança e compartilhamento de objetos comuns. Domínios de aplicação são agrupados segundo características e níveis de abstração semelhantes, o que permite que os mesmos sejam descritos por abstrações em comum. A distinção de um domínio para outro é feita a partir das características de cada domínio, que são agrupadas de forma semelhante. Esta forma de representação,

segundo Lubars [Lubars89], é semelhante as noções de facetas descritas em [Prieto-Diaz87].

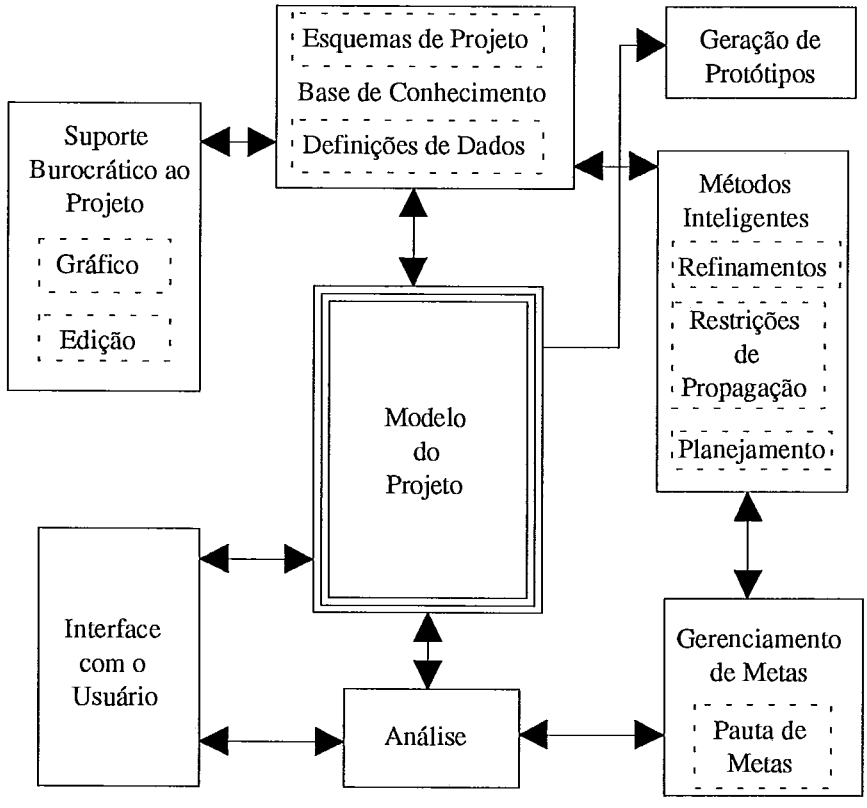



Figura IV.7 - Estrutura de IDeA [Lubars89]

### IV.2.8.3 Comentários a respeito do ambiente IDeA

O desenvolvimento de IDeA mostrou a viabilidade da incorporação de técnicas de inteligência artificial a outras características do desenvolvimento. Embora as ferramentas estejam integradas ao ambiente, não é possível ao usuário utilizar IDeA com outro paradigma de desenvolvimento, ou mesmo incorporar ao ambiente novas ferramentas para seu trabalho. Esta inclusão é dificultada ainda mais se for visualizada a estrutura de representação da informação do ambiente,

elaborada de forma a poder ser utilizada, essencialmente, pelas ferramentas inteligentes.

## IV.2.9 Outros Modelos de Ambientes

Em trabalho recente, Penedo [Penedo92] e Shu [Shu93] relacionaram um conjunto de ambientes de desenvolvimento de software e apresentaram suas características principais, com o objetivo de poder classificá-los a partir de um modelo comum. Estes ambientes, desenvolvidos de acordo com as mais variadas arquiteturas e enfoques , complementam a descrição de ambientes apresentada até aqui e demonstram algumas características adicionais no processo de construção e integração de ADS. Dos ambientes apresentados por Penedo [Penedo92] e Shu [Shu93], foram selecionados aqueles que representam o estado da arte no desenvolvimento de ambientes de desenvolvimento de software.

### IV.2.9.1 O modelo ATIS

ATIS [DEC92] ("A Tool Integration Standard") não é um ADS. Representa a proposta de um padrão cujo objetivo é facilitar o desenvolvimento de ambientes de desenvolvimento de software integrados. Sua arquitetura baseia-se num conjunto de serviços organizados em níveis específicos e especificados a partir de uma interface orientada a objetos. Os princípios fundamentais do projeto ATIS são:

- prover uma estrutura de integração orientada a objetos unificada que permita a integração de ferramentas de forma homogênea no ambiente, e,

- a padronização de um conjunto de serviços e interfaces, que aumentem a capacidade de integração e customização de ferramentas no ambiente.

A arquitetura proposta por ATIS para a construção de ambientes é

orientada a objetos, baseada num conjunto de classes pré-definidas que especificam um modelo de dados para o ambiente. Estas classes são organizadas numa hierarquia e os objetos instanciados, a partir destas classes, são chamados *elementos*. Os elementos, que pertencem a um determinado tipo, são manipulados através de mensagens. O mecanismo de herança suportado é herança simples e o conceito de meta-classes é utilizado, permitindo que o mesmo mecanismo de comunicação e troca de mensagens seja utilizado entre meta-classes, facilitando a extensão dinâmica do esquema de dados e do comportamento do ambiente.

ATIS considera que ferramentas podem estar integradas a um ambiente, desenvolvido de acordo com sua proposta de arquitetura, de quatro maneiras distintas:

- Não Integradas: representam as ferramentas isoladas do sistema e que não sofrerão nenhuma modificação para sua incorporação ao ambiente. Possuem comunicação restrita com as outras ferramentas e não podem manipular diretamente as informações existentes no contexto do desenvolvimento;

- Registradas: ferramentas registradas são reconhecidas pelo ambiente mas não podem criar elementos diferentes daqueles que foram descritos ao ambiente no momento de sua integração. Neste caso, o controle de entrada/saída pode ser realizado pelo ambiente e existe a possibilidade de comunicação da ferramenta com o ambiente, embora nenhuma modificação no código da ferramenta necessite ser realizada;

- Registradas e Encapsuladas: são ferramentas registradas no ambiente e encapsuladas numa shell, que servirá de meio de comunicação entre a ferramenta e o ambiente;

- Totalmente Integradas: são ferramentas desenvolvidas, ou que tiveram seu código modificado, para utilizar o ambiente. Podem utilizar diretamente os serviços do ambiente.

### IV.9.2 O modelo Atherton

O objetivo do modelo Atherton é oferecer um conjunto de ferramentas, construídas a partir da definição de estruturas[3] de integração e portatilidade, denominada Software BackPlane [Paseman89]. Esta estrutura é constituída por um sistema operacional genérico, um repositório de dados orientado a objetos e uma interface com o usuário padrão Apple-Macintosh.

Os objetivos fundamentais do modelo, que foi desenvolvido independente de ferramentas, metodologias, linguagens ou plataformas, são: prover um repositório de dados controlado e ordenado, permitir integração de qualquer tipo de ferramenta e suportar o controle do processo.

Software BackPlane provê uma interface orientada a objetos e um conjunto de classes pré-definidas que determinam a forma e a organização de sua base de dados. O enfoque de desenvolvimento é representado a partir de objetos que recebem mensagens. Um objeto é representado por uma entidade na base de dados. As mensagens representam a forma de comunicação entre os objetos. Tudo no ambiente é definido a partir das classes pré-definidas.

Encontram-se no modelo classes definidas para representar os seguintes aspectos:

- base de dados: contém a informação para um projeto completo, ou sobre projetos tratados pelo ambiente;

- tipos de objetos: definidos em termos das mensagens que recebe, os métodos que processam as mensagens recebidas e as variáveis de instância que possui;

- componente: um objeto de uma classe específica;

---

[3] Frameworks, no original

- coleção: um agrupamento de componentes;

- contexto: representa um subconjunto de uma base de dados que limita o escopo de utilização por parte de um usuário. Permite a definição de uma visão da base de dados;

- ferramenta: o código e o programa que pode operar num determinado objeto;

- método: um tipo de ferramenta associada com uma classe particular;

- relação: um relacionamento entre instâncias.

Ainda no modelo encontramos o conceito de mensagens genéricas. Estas mensagens são aplicadas a todos os objetos e permite aos usuários utilizar ferramentas similares usando o mesmo tipo de interface.

### IV.9.3 O modelo EUREKA

O modelo ESF ("Eureka Software Factory") [Schafer88][Eureka89] é um esforço de pesquisa europeu com o objetivo de desenvolver um ambiente capaz de ser configurado e que utilize tecnologias atuais de Engenharia de Software. A idéia central do ambiente, que define uma arquitetura diferenciada de ambientes, está centralizada em um *barramento de software*[4]. Este barramento permite a interconexão de ferramentas a partir do fornecimento de interfaces, baseadas em tipos abstratos de dados, entre os componentes de software e os usuários destes componentes.

O objetivo principal do projeto é fornecer uma plataforma segundo a qual possam ser geradas fábricas de software. Segundo Penedo [Penedo92], uma fábrica de software é um ambiente que tem seu foco voltado para o usuário e

[4] Software Bus, no original

permite o acompanhamento total do processo de desenvolvimento.

A arquitetura do modelo é baseada em duas características fundamentais, que tratam a questão da comunicação dos componentes e o modelo de execução do processo de desenvolvimento. Esta arquitetura está fortemente relacionada com a existência de três componentes fundamentais, que são os Componentes de Interação do Usuário, Componentes de Serviço e o Barramento de Software.

Componentes de interação com o usuário comunicam-se com o usuário e realizam requisições de serviços, através do barramento de software, aos componentes de serviços. Componentes de Interação podem não ter informações persistentes associadas. Componentes de serviços, por sua vez, podem não ter qualquer esquema de interface com o usuário. Um componente de interação que trabalhe em conjunto com um componente de serviço pode ser visualizado como uma ferramenta em ambientes convencionais.

O Barramento de Software é um canal de comunicação abstrato que oculta a distribuição e permite que dados de informação e controle sejam trocados entre os componentes. Este enfoque de construção permite que componentes sejam livremente conectados ao ambiente, independente de terem sido construídos para ele.

Os serviços básicos que devem existir numa fábrica de software denominam-se mecanismos centrais e são fornecidos parte pelo barramento de software e parte pelos componentes. O gerenciamento do modelo do processo é realizado a partir de sua descrição em uma linguagem de programação de processos e, no modelo Eureka, o programa do processo é construído nos tipos abstratos de dados fornecidos pelos componentes.

O modelo Eureka não utiliza um repositório central de informações como o elemento básico de integração de ferramentas. Cada componente, principalmente os componentes de serviço, pode utilizar internamente diferentes mecanismos de integração entre seus sub-componentes. Apesar desta característica, o uso de um modelo central de armazenamento é possível, desde que sejam descritos tipos abstratos de dados que possuam funcionalidades que permitam seu

compartilhamento entre os componentes.

Um componente torna disponível suas capacidades de trabalho a partir da exportação, para seus clientes, de um conjunto de tipos abstratos de dados. Cada tipo abstrato de dados modela, para a fábrica de software, um conceito de interesse que é necessário para o funcionamento da fábrica de software. Isto implica que a comunicação que ocorre no barramento de software, na realidade, consiste de chamadas às funções definidas e existentes nos tipos abstratos de dados.

O Barramento de Software é o mecanismo global de integração de uma fábrica de software construída de acordo com o modelo Eureka. A integração dos componentes é realizada a partir de dois mecanismos. Um mecanismo de conexão[5], utilizado no momento da instalação do componente, e um mecanismo de comunicação utilizado na execução do componente. O barramento de software é projetado para ocultar aspectos de distribuição e heterogeneidade dos serviços de comunicação. A integração de componentes ao barramento é feita baseada no conceito de ligações, que podem acontecer na instanciação, direcionamento, ajuste ou execução do ambiente.

Alguns ambientes Kernel/1r, Kernel/2r [Shu93] adotam este tipo de organização e arquitetura em sua construção.

### IV.9.4 O modelo HP-SoftBench

O modelo HP-SoftBench [Tatge89] é um ambiente de desenvolvimento de software que consiste em um conjunto de ferramentas integradas e uma plataforma para integração de ferramentas.

A integração das ferramentas é obtida através da troca de mensagens, controlada por um servidor de mensagens em rede, e o compartilhamento de estruturas de arquivo baseadas no modelo Unix. A filosofia de trabalho das ferramentas neste modelo diz que qualquer modificação realizada por uma

---

[5] Plug-in mechanism, no original

ferramenta, em alguma informação, deve ser comunicada às demais ferramentas do ambiente, através do servidor de mensagens.

Os principais objetivos do modelo são prover um conjunto mínimo de ferramentas altamente integradas e formas para a integração de ferramentas, possibilitando, assim, que formem um ambiente de desenvolvimento de software.

A arquitetura do ambiente está baseada em quatro componentes fundamentais:

- Difusor de Mensagens[6] : permite que as ferramentas comuniquem a execução de alguma tarefa e tomem conhecimento do que se passa no contexto do ambiente. Este servidor de mensagens realiza a integração de controle das ferramentas;

- Interface com o usuário: é fornecido às ferramentas um conjunto de funcionalidades de interface com o usuário baseado no modelo OSF/Motif, sobre X-Windows;

- Encapsulador: fornece um meio de integrar ferramentas ao ambiente. Este encapsulamento é realizado a partir da descrição da interface com o usuário e das necessidades de comunicação da ferramenta, através do difusor de mensagens, por uma linguagem de descrição de encapsulamento;

- Ferramentas Integradas: fornecem funções para o gerenciamento de configuração e programação, possuindo recursos para o controle das ferramentas do ambiente e acompanhamento das mensagens que estão trafegando pelo difusor de mensagens.

Neste modelo a preocupação com a integração se restringe às questões de controle e interface com o usuário. A integração de controle é realizada pelo difusor de mensagens que, através de seus mecanismos de gerenciamento de

---

[6] Broadcast Message Server, no original

mensagens, permite a ativação, monitoração dos eventos e comunicação entre as ferramentas. A integração da interface com o usuário é suportada desde que as ferramentas se utilizem de mecanismos padrão de interface, baseados no modelo X-Windows.

A integração de dados não é tratada diretamente e pode ser realizada através do sistema de arquivos Unix. Este enfoque de integração de dados não prejudica o modelo proposto, tendo em vista que não é um de seus objetivos realizar a integração de dados.

### IV.9.5 O modelo PCTE

O modelo PCTE ("Portable Common Tool Environment") [Boudier88][Thomas89b] é um conjunto de interfaces que define uma estrutura para o desenvolvimento de Ambientes de Desenvolvimento de Software. O objetivo do projeto é definir uma especificação, independente de linguagem e plataforma, de uma interface pública para ferramentas, que possibilite a portatilidade das ferramentas do ambiente entre diferentes plataformas, e que sirva como uma base para a integração de ferramentas.

No projeto do PCTE, cuja principal motivação foi a necessidade de prover portatilidade de ferramentas entre ADSs, considera-se os seguintes aspectos:

- independência de desenvolvedores e equipamentos;
- uma interface com funcionalidade suficiente para atender às necessidades dos usuários;
- suporte a múltiplas linguagens;
- facilidade de integração de ferramentas existentes;
- disponibilidade rápida de uma implementação do PCTE.

A arquitetura principal do modelo, embora não exclusiva, baseia-se num conjunto de estações de trabalho interligadas por uma rede local, comunicando-se através de facilidades, que se encontram definidas e disponíveis na interface pública, construídas para as ferramentas. Estas facilidades são representadas por três conjuntos básicos de serviços, que são:

- Sistema de Gerenciamento de Objetos: baseado no modelo de Entidades-Relacionamentos, estendido para suportar o conceito de herança múltipla de tipos;

- Controle do Processo: é possível, no PCTE, realizar a descrição e a modelagem dos processos. Esta descrição, denominada contexto estático dos processos, é utilizada para controlar a ativação dos processos. Após ativados, os processos obtêm as facilidades de comunicação a partir do Sistema de Gerenciamento de Objetos, de filas de mensagens, estruturas de encadeamento[7] e de sinalização;

- Gerenciamento da Interface com o Usuário: possui definido um conjunto de primitivas para o gerenciamento da interface com o usuário, embora considere que o desenvolvimento de um sistema gerenciador de interface com o usuário não faça parte do contexto do PCTE.

A integração das ferramentas pode ser realizada a partir dos serviços oferecidos pela interface pública. Apesar de fornecer estes serviços não está definida, nesta estrutura, qual deve ser a filosofia de integração no ambiente. O sistema gerenciador de objetos possui funcionalidades que podem ser utilizadas para efetivar a integração, mas não determina a forma normal de utilização destas facilidades.

---

[7] pipes, no original

### IV.9.6 O modelo NSE

NSE ("The Sun Network Software Environment") [Adams89][Miller89] é um gerenciador de objetos voltado para aplicação em rede e uma facilidade de integração de ferramentas. O gerenciador de objetos realiza o controle de versões e configurações de informações baseado em mecanismos de controle de concorrência, que permite às ferramentas, componentes de um ambiente, utilizarem informações em comum de modo compartilhado e não conflitante. A integração de ferramentas ao ambiente ocorre a partir da utilização de serviços do ambiente para controlar seus objetos, que são representados pelos arquivos utilizados pela ferramenta, e à sua própria incorporação, e disponibilidade, no ambiente. Seus principais objetivos são:

- ser invisível para o usuário, realizando a maior parte de seu trabalho em "background";
- suportar as ferramentas de desenvolvimento padrão do Unix;
- tornar a rede transparente, fazendo com que outros objetos e arquivos pareçam locais para o usuário, independente de sua localização;
- encorajar o desenvolvimento em paralelo a partir de utilização de um mecanismo de controle de concorrência otimista[8] sobre os objetos;
- permitir o ajuste e a extensão do ambiente.

Na arquitetura do ambiente encontramos, basicamente, serviços que permitem o gerenciamento de objetos e tarefas. Estes serviços fornecem quatro facilidades básicas, representadas da seguinte maneira:

- *objetos*, que identificam um local de armazenamento de dados, como um arquivo;

[8] Optimistic Concurrency Control, no original

- *ferramentas*, que apresentam ou trocam os conteúdos de um determinado objeto;

- *comandos*, que manipulam um objeto sem se preocupar com seu conteúdo;

- *ambientes*, que são espaços de trabalho isolados, nos quais o desenvolvedor pode modificar objetos sem a interferência de outros desenvolvedores que estejam modificando o mesmo objeto.

O elemento básico de manipulação no modelo é o objeto. Um objeto é tratado, em sua forma mais rudimentar de representação, como um arquivo. É possível realizar com um objeto uma série de operações básicas e primitivas. Dependendo da forma como estes objetos são organizados e utilizados eles podem ser visualizados de acordo com as seguintes características:

- *Arquivos*: representam o objeto básico e, na sua maioria, são encontrados na forma de arquivos Unix. Objetos podem ser editados, listados, compilados, copiados e movidos;

- *Alvo*: é um objeto composto, uma lista de outros objetos, que simplifica o gerenciamento de arquivos derivados que possuem muitas dependências. Normalmente correspondem aos arquivos executáveis;

- *Base de Dados de Ligações*: armazena conexões entre os objetos num ambiente, mostrando o relacionamento existente entre os objetos. Esta categoria de objetos pode ser visualizada como uma forma de hipertexto, onde é possível demonstrar a composição e o sentido de caminhamento num determinado componente;

- *Componente*: objeto de propósito geral que pode conter todos os tipos de objetos, incluindo outros componentes.

Para cada tipo de objeto pode existir uma, ou várias, ferramentas.

Ferramentas são específicas para um determinado tipo de objeto. Enquanto as ferramentas manipulam o conteúdo de objetos, os comandos do ambiente NSE gerenciam recipientes de objetos, sem se preocupar com seu conteúdo. Estes comandos realizam, na maioria dos casos, solicitações, realizadas pelo mecanismo de controle e gerenciamento de configuração, de cópias de objetos entre ambientes.

Um ambiente é identificado pelo NSE como sendo um local nomeado que contêm uma coleção de objetos, e em geral, são utilizados para separar tipos particulares de produtos e atividades de desenvolvimento.

O gerente de objetos do NSE não se configura como um gerenciador de propósito geral. Seu uso é semelhante ao uso de arquivos Unix, exceto pelo fato de existir um controle sobre estes arquivos. A principal diferença entre o sistema de arquivos convencional do Unix e o do NSE reside no suporte de um modelo particular de configuração e gerenciamento de versões. Este modelo é baseado num controle de concorrência otimista de trocas de objetos, que garante a livre utilização dos objetos pelos ambientes. Um ambiente nunca possui o objeto, mas sempre uma cópia dele. Trabalhos em paralelo podem ocorrer no ambiente com o mesmo objeto e as eventuais modificações ocorridas no objeto são mescladas a partir de serviços específicos, sem a necessidade de utilização de mecanismos de bloqueio.

## IV.10 Considerações sobre ADS

Neste capítulo foi apresentado um conjunto de Ambientes de Desenvolvimento de Software que possuem em sua constituição diferentes aspectos para abordar a questão da integração de ferramentas, conforme descrito no capítulo III.

Ambientes estão evoluindo a cada dia. Esta evolução, aliada à evolução do

processo de desenvolvimento e à disponibilidade de hardware que permite a construção de ADS cada vez mais poderosos, faz com que sejam incorporados requisitos essenciais que determinem características de apoio a trabalho cooperativo [Moura92] e o controle total do processo de desenvolvimento [Penedo93a], [Penedo93d]. Para que ADSs possam suportar estes requisitos é necessário que os mecanismos de integração levem em consideração as características de cooperação, ao longo do desenvolvimento, ao mesmo tempo que possuam o conhecimento sobre o processo de desenvolvimento. Embora alguns dos ambientes apresentados (NSE, ATIS, ATHERTON, AD/Cycle, Eureka) tratem, de forma branda, os dois requisitos apresentados, nenhum deles o faz de forma completa, ou seja, eles não contemplam os dois requisitos ao mesmo tempo.

Outro aspecto que deve ser considerado é quanto ao paradigma de desenvolvimento utilizado para a construção dos ADSs. A maioria dos ambientes utiliza o paradigma convencional de construção, onde se observa uma forte influência do pensamento estruturado aliado a modelos de relacionamentos entre as informações. Esta forma de construção não permite flexibilidade na representação das informações necessárias ao longo do desenvolvimento, e impõe às ferramentas, pertencentes ao ambiente, a execução de tarefas que não são de sua responsabilidade.

# Capítulo V

# Integração de Ferramentas na Estação TABA

## V.1 A Estação TABA

O projeto TABA [Rocha87b], [Rocha90], [Aguiar92], tem como objetivo a construção de uma Estação de Trabalho configurável para o desenvolvimento de software. A motivação para o projeto é prover desenvolvedores de software com ambientes de desenvolvimento de software que atendam às particularidades de domínios de aplicação e projetos específicos.

Para atender a este objetivo a estação TABA tem quatro funções:

I) auxiliar o engenheiro de software na especificação e instanciação do ambiente mais adequado ao desenvolvimento de um produto específico;

II) auxiliar o engenheiro de software a implementar as ferramentas necessárias ao ambiente definido em (I);

III) permitir aos desenvolvedores do produto (software) o uso da estação através do ambiente instanciado em (I) ;

IV) permitir a execução do software, dado que, eventualmente, o software produto poderá ser executado na própria estação para ele configurada (o que é sempre verdade pelo menos na fase de testes).

Estas funções desejadas para a Estação TABA fazem com que, na definição de sua estrutura, seja criado um conjunto de serviços, agrupados em ambientes distintos, responsáveis pela efetiva realização das funções. Desta forma tem-se quatro ambientes que são:

- **Ambiente Especificador e Instanciador de ADSs**

É o meta-ambiente TABA. Sua função é especificar o ADS mais adequado para o desenvolvimento de um produto de software num determinado domínio de aplicação e instanciar o ADS.

O instanciador é responsável por selecionar entre os componentes disponíveis na Estação aqueles que foram indicados para compor a instância de ADS, e tornar operacional o ambiente. Além de tornar operacional o ADS cabe ao instanciador agregar ao ADS instanciado um assistente baseado em conhecimento de auxílio à sua utilização.

A construção do especificador de ambientes, **XAMÃ** [Aguiar92], [Massolar93], foi realizada e está operacional em ambiente Sun.

- **Ambiente para Construção de Ferramentas**

Os componentes disponíveis na Estação e, portanto considerados por XAMÃ, foram, em algum momento, incorporados à mesma. Em algumas situações, entretanto, pode ocorrer que a especificação do ADS contenha requisições a componentes ainda não disponíveis no meta-ambiente. Neste caso, os componentes devem ser construídos, permitindo a instanciação completa do ADS especificado. Além disso, novas ferramentas podem, a qualquer momento, ser sugeridas (especificadas) pelo meta-usuário[1]. O Ambiente para Construção de

[1] Neste contexto o meta-usuário é o Engenheiro de Software

Ferramentas é o responsável por auxiliar na construção destas novas ferramentas e na sua incorporação ao meta-ambiente.

- **Ambiente de Desenvolvimento**: é o ADS que foi especificado e instanciado, através do meta-ambiente.

- **Ambiente de Execução**: é o local onde o produto de software poderá ser executado.

O objetivo desta tese é completar a definição e construção do meta-ambiente TABA, definindo um mecanismo para integração de ferramentas. Este mecanismo torna possível a instanciação dos ambientes especificados por XAMÃ, a construção de novas ferramentas na Estação e a integração de ferramentas externas desenvolvidas fora da Estação.

Nas próximas seções definimos os requisitos da Estação TABA (seção V.2), sua estrutura (seção V.3), a filosofia de integração (seção V.4), o modelo de integração TABA (seção V.5), descrevendo neste modelo considerações à respeito do meta-ambiente TABA e de seus ambientes instanciados.

## V.2 Requisitos da Estação TABA

Considerando o levantamento feito na literatura técnica sobre requisitos de ADSs (seção II.3) e a proposta do Projeto TABA [Rocha87b], [Rocha90], identificamos os seguintes requisitos para a Estação TABA, tanto a nível do meta-ambiente quanto dos ADSs instanciados na Estação:

**- Requisitos gerais da Estação TABA**

- **ser configurável**: a Estação TABA deve poder ser configurada para diferentes domínios de aplicação de forma a atender às especificidades do desenvolvimento de software para estes domínios;

- **possuir uma interface consistente**: a Estação TABA deve possuir mecanismos de interface com o usuário que permitam uma utilização consistente de seus recursos e ferramentas;

- **possuir um mecanismo de integração**: a Estação TABA deve permitir, e facilitar, a integração de ferramentas, sejam elas desenvolvidas com a tecnologia utilizada no próprio Projeto TABA (ferramentas internas), ou então, desenvolvidas por terceiros (ferramentas externas);

- **apoiar na construção de novas ferramentas:** a estação TABA deve possuir funcionalidades que permitam ao Engenheiro de Software construir, ou adaptar, novas ferramentas para a Estação;

- **possuir conhecimento sobre o processo e métodos de desenvolvimento**: a Estação TABA deve possuir conhecimento sobre o processo de desenvolvimento de software, as várias alternativas de modelos para o ciclo de desenvolvimento e sobre os métodos possíveis de serem utilizados nas várias atividades do processo, bem como sobre a adequabilidade de sua aplicação em diferentes contextos;

- **oferecer assistência inteligente ao usuário**: tanto a nível de assistência para o uso da Estação como um todo, quanto dos ADSs específicos e das ferramentas disponíveis;

- **possuir um modelo de armazenamento de dados comum**: as informações devem possuir uma forma de representação tal que possibilite às ferramentas compartilhar, e utilizar estas informações de forma natural e consistente;

- **possuir suporte à reutilização**: possuindo mecanismos que possibilitem a reutilização tanto a nível de código, quanto de especificação e projeto.

**- Requisitos específicos dos ADSs configurados pelo meta ambiente TABA**

- **possuir suporte para o controle e gerenciamento de versões**: os itens de software construídos ao longo do processo de desenvolvimento sofrem alterações durante este mesmo processo. É responsabilidade do ambiente controlar e gerenciar estas modificações, mantendo os documentos gerados disponíveis para os usuários em suas diferentes versões;

- **possuir suporte para a medição do produto**: este requisito considera que cabe ao ADS o papel de armazenar, ao longo do processo de desenvolvimento, informações relevantes que permitam fazer medidas sobre o processo de desenvolvimento. Estas métricas, estando disponíveis para futuros projetos, possibilitam aperfeiçoamentos no processo de desenvolvimento, consequentemente, melhorias na qualidade dos produtos;

- **possuir suporte para o gerenciamento de todas as atividades ao longo do processo de desenvolvimento**: a Estação TABA deve oferecer suporte para controle e gerencia do processo de desenvolvimento, verificando o andamento do trabalho, controlando a execução de atividades relacionadas e encadeadas, não permitindo que a ordem de execução das tarefas seja trocada, ou mesmo modificada, sem que haja uma intervenção do Engenheiro de Software;

- **apoiar o trabalho cooperativo:** dado que o desenvolvimento de um produto de software, principalmente quando se trata de desenvolvimento em larga escala, se dá através do trabalho de equipes. Isto traz a necessidade de se ter disponíveis controles de coordenação e interação, de forma que as atividades possam ser desenvolvidas corretamente. Além disso, a comunicação entre a equipe, ao longo do processo de desenvolvimento, ocorre frequentemente, obrigando o ambiente a definir protocolos de comunicação que possibilitem este relacionamento;

- **possuir interfaces customizáveis**: dado que cabe ao usuário, e não ao ADS, determinar como será a forma de apresentação da interface. É preciso que se

forneça ao usuário meios para que ele possa ajustar a interface apresentada às suas preferências pessoais;

- **possuir suporte para a avaliação do produto**: a Estação TABA deve oferecer suporte para a medição e avaliação da qualidade dos produtos nela desenvolvidos.

Estes requisitos, identificados para a Estação TABA, vem sendo explorados, validados e implementados pelos participantes do projeto através de uma série de trabalhos e sub-projetos.

Inicialmente, foram realizados trabalhos exploratórios onde se enfatizou a construção de ferramentas: [Vale88], [Nogueira88], [Bidarra88], [Cavalcante88], [Menezes88], [Angulo88], [Ribeiro89], [Travassos90], [Cardoso90], [Koblitz91], [Passos91], [Guttler91], [Faria91], [Robson91], [Vilanova92].

Num segundo momento, foi realizada uma série de trabalhos visando a construção do especificador de ambientes [Werneck90], [Duarte91], [Crispim91], [Chitman91], [Assis92], [Moura92]. Estes trabalhos culminaram na definição e implementação de XAMÃ, o especificador de ambientes da Estação TABA [Aguiar92], [Massolar93].

Paralelamente, foi iniciada uma série de trabalhos visando a definição de ambientes para diversos domínios de aplicação e paradigmas de desenvolvimento, alguns já concluídos e outros em andamento: software científico [Souza90], [Rocha89], [Rocha91], [Andrade91], [Bahia92], [Werner92], [Maidantchik92]; software educacional [Stähl92], [Campos93], [Breitman93], [FCampos93a], [FCampos93b], [Campos94]; software financeiro [Belchior92]; sistemas baseados em conhecimento [Neves94], [Werneck94]; sistemas orientados a objeto [Mattoso90], [Gonçalves93], [Rocha93].

Ainda neste contexto, foram iniciados trabalhos na direção de definir e construir os ambientes para a avaliação da qualidade [Rocha92], [Passos93],

[Mendonça93] e gerência de projetos [Sardinha93], que serão integrados aos ADSs configurados pelo meta ambiente.

Os requisitos relacionados a armazenamento de dados, gerenciamento de versões [Monte93], [Mattoso93], apoio ao trabalho cooperativo [Camargo92], [Vaz92], [Travassos93b] e suporte à reutilização [Werner92], [Xexeo94] vem sendo trabalhados pelo grupo de Banco de Dados.

Este trabalho tem como objetivo garantir que a Estação TABA atinja os seus requisitos de **possuir um mecanismo de integração** e **apoiar na construção de novas ferramentas**. Nas próximas seções detalharemos como são atingidos estes requisitos.

## V.3 A Estrutura da Estação TABA

Na seção II.5 foram descritas as principais estruturas para ADSs propostas na literatura. Tendo em vista as dificuldades apresentadas por estas estruturas e a necessidade de se modelar a estrutura de um ADS, realçando sua constituição e mostrando que, na realidade, os serviços existentes não interagem aos pares, mas sim como um todo, adotaremos para a Estação TABA a estrutura mostrada na figura V.1.

Nesta estrutura, a idéia principal que se pretende passar é que um ADS está inserido no contexto de um produto de software e é o responsável, com seu conjunto de funcionalidades e informações associadas, que aqui chamaremos de *componentes*, por traduzir (segundo a necessidade e desejo do usuário) uma representação do mundo real para o mundo computacional, que é o próprio produto de software. O que um ADS tem que fazer então, observados os requisitos funcionais estabelecidos, é diminuir a distância existente entre o problema do mundo real e o mundo computacional, facilitando para o usuário do ADS a construção de produtos de software.

Entretanto, um ADS também é um produto de software. E como todo produto de software sua utilização só é possível a partir de um computador. Esta utilização traz para o usuário a sensação de que ele está manuseando um elemento do seu mundo real. Este elemento, ou software, responde aos estímulos do usuário de acordo com as funcionalidades que foram previstas em sua construção e que estão disponíveis para uso. Estas funcionalidades estão presentes no produto, mas em alguns casos fica difícil para o usuário percebê-las separadamente. Elas têm que funcionar de forma harmônica, integradas e, o que é fundamental, independentes mas não livres. Esta forma de estruturação é possível quando se consegue tratar as funcionalidades como componentes separados, contendo controle sobre a sua existência, suas informações e estados associados, e possuindo um conjunto de funcionalidades básicas associadas, que são estimuladas através de uma comunicação organizada entre o usuário e o sistema.

Podemos, então, visualizar esta situação através da representação da figura V.1. Podemos considerar um ADS, sob o ponto de vista do usuário, como um sólido no seu mundo real. Este ADS real, quando apresentado no mundo computacional, precisa ser representado por seus componentes básicos. A projeção deste sólido, que é o ADS, num plano bidimensional, que representa a noção do computador, indica que os componentes, aparentemente disjuntos e isolados, na realidade, possuem um ponto de contato que é representado pelo centro do círculo, e que representa a conjunção das funcionalidades do ADS, e portanto, o próprio ADS. Estas funcionalidades, ou componentes, do ADS representam a forma segundo a qual ele consegue atender aos requisitos especificados para sua construção.

Uma representação semelhante foi utilizada por Miyoshi [Miyoshi93] para ilustrar a avaliação da qualidade de diversos ambientes de desenvolvimento de software. Em seu trabalho Miyoshi usou esta estrutura para mostrar, através de características de qualidade previamente definidas, o quanto determinados ambientes satisfaziam, ou não, aos requisitos estabelecidos. Apesar de visualmente semelhante, a representação utilizada, aqui, possui outro significado.

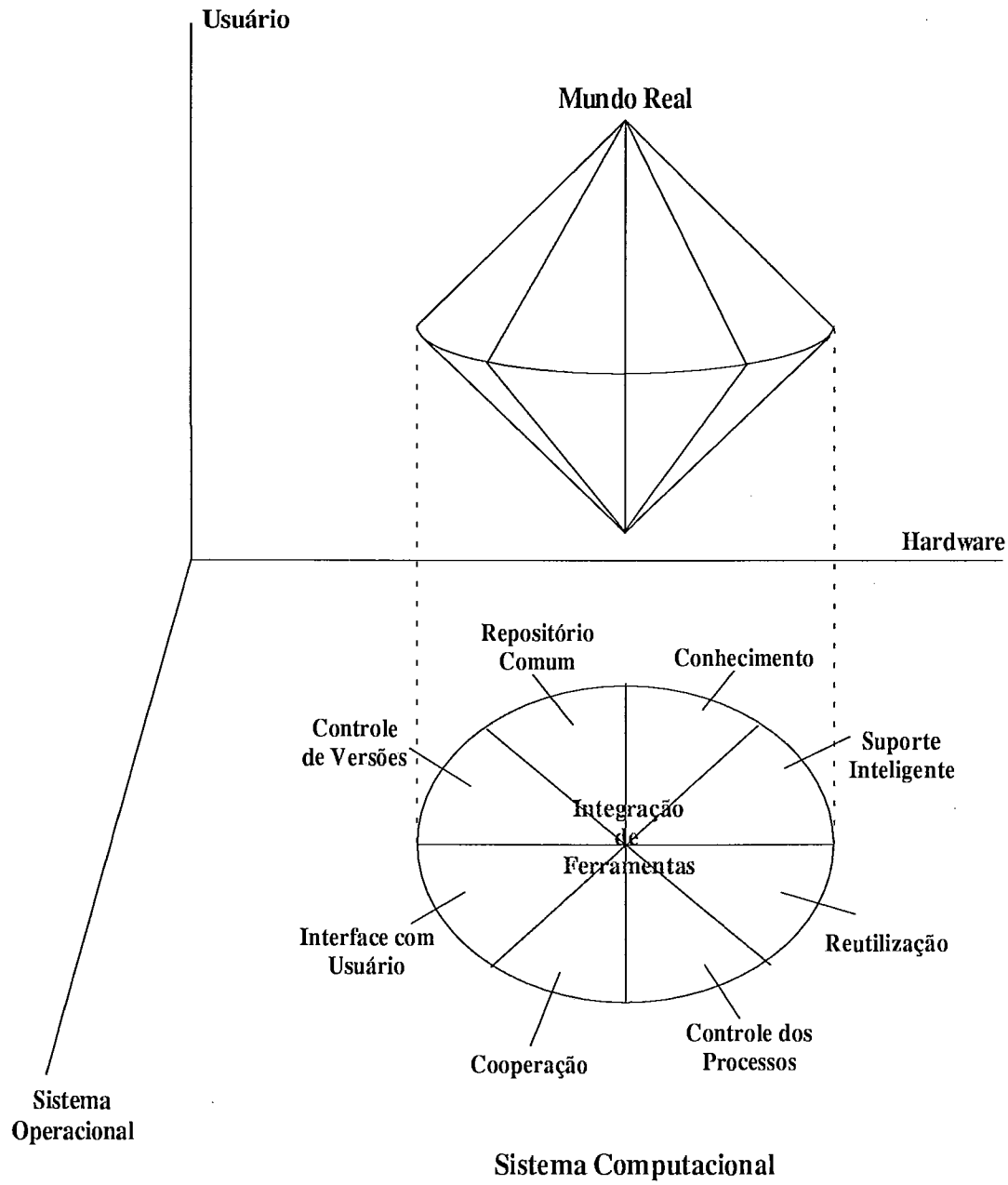



Figura V.1 - Estrutura de um ADS

De acordo com os requisitos descritos na seção II.3, os componentes presentes na Estação TABA são:

- O **Sistema Computacional**, representado por uma dimensão de hardware e outra do sistema operacional;

- O **componente de Interface com o Usuário,** que é o responsável pela gerência e controle da interação do usuário com o ADS. É de sua responsabilidade garantir a consistência da apresentação das ferramentas e permitir, na medida do possível, que o usuário ajuste a aparência do ADS às suas preferências pessoais;

- O **componente Cooperação** trata da comunicação entre os usuários e suas necessidades de interação com outros membros da equipe. Possui definições de protocolos de comunicação que devem ser utilizados no convívio no ADS. Este componente precisa auxiliar o componente de Interface com o Usuário no sentido de prover funcionalidades e recursos que o estendam de forma a suportar a interface de grupo;

- O **componente Controle dos Processos** é o responsável pelo controle e gerenciamento do processo de desenvolvimento. Identifica atividades, gerencia a execução das ferramentas e controla os papéis e atividades dos usuários, levantando dados sobre a realização do processo para o meta-ambiente.

- O **componente Suporte Inteligente**, que fornece inteligência global ao ambiente, assistindo o usuário na utilização do meta-ambiente e dos ADSs, isto é, sobre o uso de suas ferramentas constituintes e os métodos que as mesmas automatizam;

- O **componente Reutilização** provê o ADS de mecanismos que possibilitam o reaproveitamento de trabalhos anteriores. A reutilização se dá a nível da especificação do ADS, de componentes para a construção de ferramentas, bem como de componentes de todo tipo para a construção da aplicação. Assim sendo, este é um componente muito importante no meta-ambiente pois

possibilita ao instanciador e ao construtor de ferramentas o acesso a itens de software e ferramentas previamente desenvolvidos;

- O **componente Conhecimento** incorpora mecanismos de inferência que possibilitam a integração do conhecimento armazenado no ADS. O meta-ambiente se utiliza deste componente para a obtenção de características de ADS a serem instanciados. Ferramentas também usam este componente para obterem informações sobre o significado dos itens de software que estão manuseando. Embora possa parecer, à primeira vista, que este componente possui as mesmas funcionalidades do componente Suporte Inteligente, na realidade, o que ocorre é um forte relacionamento entre eles. Neste componente existem mecanismos que permitem o gerenciamento e armazenamento do conhecimento. O componente funciona, para o ambiente, auxiliando outros componentes em seu funcionamento, ao passo que no anterior, o objetivo principal é apoiar o usuário na utilização do ADS;

- O **componente Repositório Comum do ADS** é o responsável pelo controle, gerenciamento e armazenamento dos objetos manuseados pelo ambiente e meta-ambiente. É de sua responsabilidade manter a consistência e a integridade das informações;

- O **componente Controle de Versões**, que está muito relacionado com o repositório comum, controla e gerencia versões de documentos e itens de software produzidos. A partir dele o usuário tem condição de obter informações sobre um determinado momento no desenvolvimento.

A utilização, em conjunto, destes componentes permite a integração de ferramentas ao ambiente. Estes componentes definem políticas de armazenamento, a forma de interface com o usuário, embora não interferindo em sua funcionalidade, provêm recursos para a incorporação de ferramentas externas ao ambiente, e, de forma mais específica, definem a **filosofia** de integração da Estação TABA e de seus ambientes instanciados.

## V.4 A Filosofia de Integração da Estação TABA

Nos capítulos III e IV mostramos vários enfoques e preocupações, existentes na literatura, para tratar a questão da integração em Ambientes de Desenvolvimento de Software.

Ferramentas em ADS devem suportar algum método de desenvolvimento, e ainda, possibilitar que as informações a elas associadas estejam disponíveis para todo o ambiente, facilitando que outras ferramentas tomem conhecimento sobre o processo de desenvolvimento e o significado das informações existentes, e sobre a forma em que estas informações podem ser utilizadas.

Para que isto seja possível entendemos que é necessário considerar, além da integração de dados, controle e apresentação, conforme proposto por diferentes autores e em diversos ambientes [Brown92a] [Brown92b] [Brown92c], [Wasserman90], [Thomas92], [Chen92], [Mi92], [Meyers91], [Garlan87], [Yang93], [Penedo88], [Penedo92], [Penedo93b], [Penedo93d], [Pressman92], **a integração do conhecimento.**

Este conhecimento deve estar disponível no ambiente, de forma a poder ser reutilizado por outras ferramentas no contexto do desenvolvimento. A integração de conhecimento possibilita às ferramentas um melhor entendimento da semântica das informações e dos métodos existentes para tratar a informação. Além deste aspecto é considerado, também, que integração é uma filosofia de construção e como tal deve ser considerada como um todo, levando-se em consideração que os próprios mecanismos de integração tem que estar integrados. Na figura V.2 tem-se uma representação para este enfoque de integração.

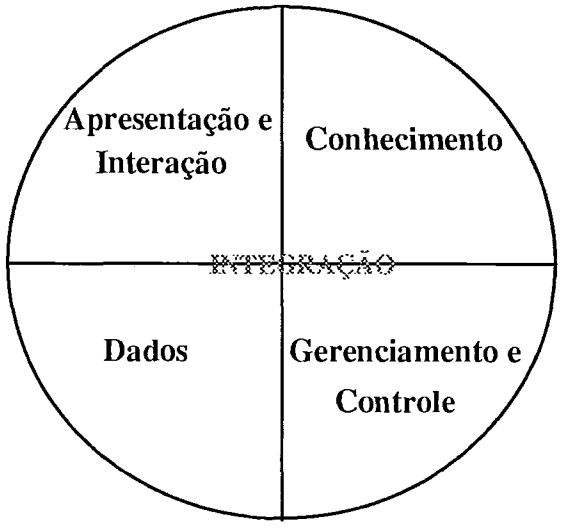



Figura V.2 - Enfoque de Integração da Estação TABA

## ▪ Integração de Apresentação e Interação

A integração de apresentação e interação é responsável por proporcionar ao usuário a sensação de integração no ambiente. É através dela que será possível a homogeneização das formas de apresentação da informação e das técnicas de interação do usuário. Neste aspecto, concordamos com o enfoque adotado por Thomas [Thomas92], descrito na seção III.2.1.2, acrescentando uma terceira característica que é a questão da **customização da interface**.

É importante que os mecanismos de interação e apresentação possibilitem ao usuário recursos para ajustes da interface, de forma a torná-la o mais agradável e adequada possível, às características e desejos do usuário. Ajustes feitos numa ferramenta, e que modifiquem elementos básicos e fundamentais da interação, devem refletir diretamente no ambiente, influenciando a apresentação das demais ferramentas que o compõem. No enfoque TABA, este tipo de integração é responsabilidade do componente Interface com o Usuário.

## ▪ Integração de Gerenciamento e Controle

A integração de gerenciamento e controle é responsável por prover, às ferramentas e ao ambiente, serviços e funcionalidades básicas que permitam o funcionamento de forma organizada, ao longo do processo de desenvolvimento.

No enfoque TABA, este tipo de integração é responsabilidade dos componentes Cooperação, Controle dos Processos, Reutilização e Controle de Versões.

O componente Cooperação é o responsável pela comunicação, colaboração e coordenação dos usuários nas atividades de desenvolvimento que devem ser realizadas em grupo [Kling91], [DNorman91], [Ellis91], [Travassos93b]. A comunicação trata de como será realizada a comunicação entre os usuários. A colaboração trata da forma como os usuários compartilharão as informações. A coordenação aumenta a eficiência da comunicação e da colaboração no sentido de que seu principal objetivo é evitar conflitos entre os usuários.

O componente Controle dos Processos controla e gerencia o processo de desenvolvimento, identificando as atividades, gerenciando a execução das ferramentas, não permitindo, por exemplo, que uma ferramenta, para uma determinada fase do processo, seja executada antes que as informações necessárias para sua execução estejam disponíveis. É responsabilidade do componente Controle dos Processos o acompanhamento do processo de desenvolvimento e a manutenção dos dados relativos à execução das atividades para o meta-ambiente.

O componente Controle de Versões é responsável por controlar os documentos gerados num determinado momento do processo de desenvolvimento, e torná-los acessíveis ao usuário.

O componente Reutilização é responsável por manter as informações geradas em outros ambientes e processos de desenvolvimento, que possam ser reutilizadas no contexto do desenvolvimento atual.

## ▪ Integração de Dados

A integração de dados estabelece a forma como as ferramentas da Estação TABA realizarão o tratamento das informações.

Este aspecto da integração deve prover os serviços básicos de armazenamento e gerenciamento de estruturas de informação (responsáveis por descrever os documentos gerados durante o processo de desenvolvimento) obtidas a partir das ferramentas componentes do ambiente.

No enfoque TABA este tipo de integração é responsabilidade do componente Repositório Comum do ADS.

## ▪ Integração de Conhecimento

A integração do conhecimento torna disponíveis os serviços básicos de armazenamento, gerenciamento e utilização do conhecimento descrito e adquirido ao longo do processo de desenvolvimento.

O conhecimento compartilhado pelas ferramentas descreve o significado das informações construídas ao longo do processo de desenvolvimento, e permite a descrição da sintaxe apropriada para a apresentação destas informações por parte das ferramentas.

Este tipo de integração é obtido no enfoque TABA a partir da utilização dos componentes Suporte Inteligente e Conhecimento.

O componente Conhecimento contém o conhecimento necessário ao funcionamento do ADS, e permite, a partir de mecanismos de inferência, a utilização deste conhecimento, e com isto, sua integração ao ADS.

O componente Suporte Inteligente é responsável por fornecer a inteligência global ao ambiente, assistindo o usuário no uso do ADS e de suas ferramentas associadas.

## V. 5. O MODELO DE INTEGRAÇÃO TABA

Considerando os requisitos da Estação TABA, especificados na seção V.1, podemos identificar duas questões relativas à Estação TABA.

A primeira diz respeito ao meta modelo TABA, e trata, especificamente, do meta ambiente e suas necessidades funcionais. A segunda se apresenta no momento em que, a partir da utilização da Estação TABA, ambientes são instanciados e tornam-se passíveis de utilização para o desenvolvimento de produtos de software. Estas duas situações precisam ser consideradas no contexto da integração, pois é responsabilidade da Estação TABA a instanciação de Ambientes de Desenvolvimento de Software **integrados**.

Para a descrição e especificação dos dois contextos descritos acima (isto é, o meta-ambiente e os ambientes instanciados) será utilizado o método Análise Orientada a Objetos proposto por Coad e Yourdon [Coad92], cuja descrição sintética pode ser encontrada no apêndice B (seção B.2.1). A escolha deste método deveu-se à sua aplicabilidade ao paradigma orientado a objetos e à disponibilidade de ferramenta CASE de apoio a sua utilização.

### V.5.1 O META MODELO TABA

O principal objetivo da Estação TABA é permitir a especificação de ADSs adequados a diferentes contextos e sua instanciação como um ambiente integrado.

Esta instanciação se faz a partir de um conjunto de facilidades e funcionalidades que suportam o Engenheiro de Software na definição e construção de novos ambientes. O meta modelo da Estação TABA está representado no diagrama de assuntos da figura V.3. No apêndice A pode ser encontrado o diagrama detalhado do meta-ambiente da Estação TABA. Neste diagrama são encontrados os seguintes assuntos que representam, a partir das classes que os compõem, a funcionalidade da Estação:

- **Meta Ambiente**: este assunto representa a essência do modelo. É composto apenas pela classe *Estação TABA*. O objeto descrito por esta classe é o responsável pela coordenação das tarefas do meta ambiente, tornando disponíveis, quando necessário, conhecimento, padrões de interface com o usuário e estruturas para a construção de ADS's e ferramentas.

- **Apresentação e Interação**: este assunto reúne os elementos responsáveis pela comunicação do usuário com a Estação. É composto de uma classe única, *Interface com o Usuário*, representando todos os objetos de interação.

- **Conhecimento**: reúne os objetos responsáveis por fornecer conhecimento sobre os métodos, processo de desenvolvimento e domínios de aplicação para a Estação, ADS's e ferramentas. Existem duas classes compondo este assunto que são *Conhecimento* e *Assistente Especialista.*

- **Frameworks**: representa as estruturas básicas para a construção de ambientes e ferramentas, que devem estar presentes na Estação. Neste assunto são encontradas as classes *FrameworkAmbientes* e *FrameworkFerramentas*, com suas respectivas especializações.

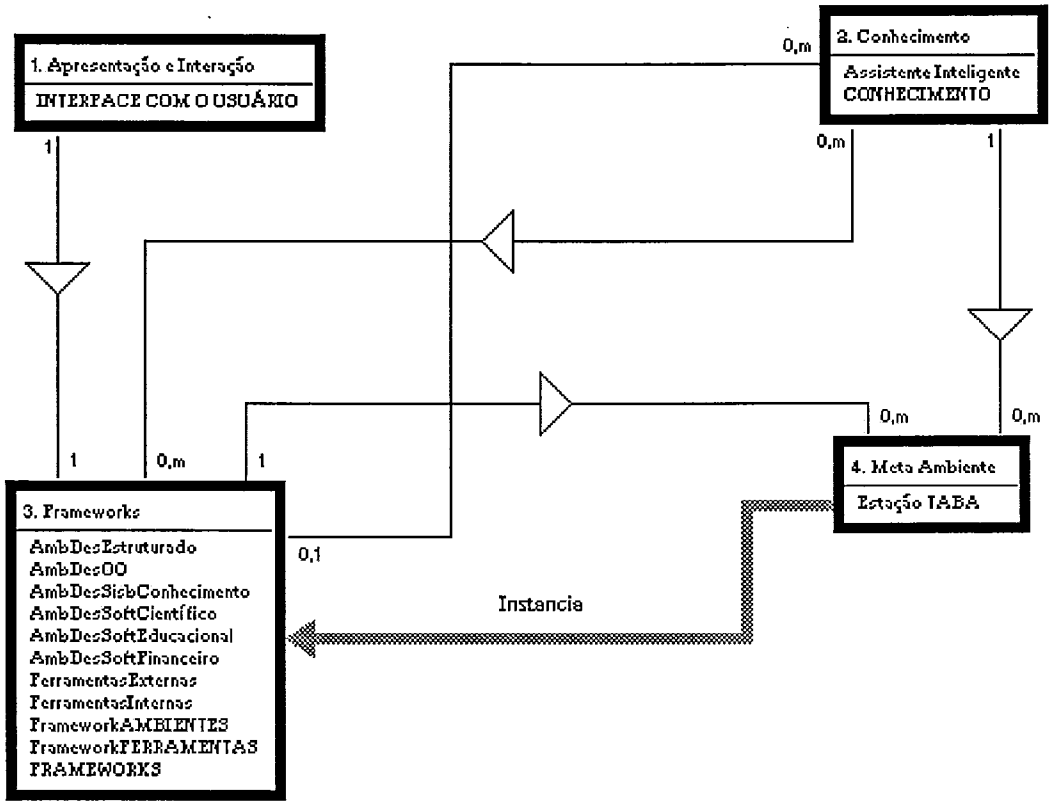



Figura V.3- Diagrama de Assuntos do Meta Ambiente TABA

A seguir serão descritas, com mais detalhe, as classes existentes nos assuntos do meta ambiente.

### V.5.1.1 A Classe Estação TABA

A classe Estação TABA representa o meta ambiente TABA, e é descrita a partir do conjunto de funcionalidades necessárias para o funcionamento da Estação. A instância da classe TABA é responsável pela coordenação dos trabalhos no meta ambiente e possui um conjunto de serviços associados.

- **Definir Ambiente**: este serviço realizado com o suporte do XAMÃ [Aguiar92], [Massolar93], permite a especificação de Ambientes de Desenvolvimento de Software que atendam às características de domínios de aplicação e projetos específicos.

- **Instanciar ambiente**: a partir da definição do ADS, realizada através de XAMÃ, instancia este novo ambiente de desenvolvimento de software e o torna disponível para o uso.

- **Testar Ambiente**: permite testar um ambiente instanciado e verificar se o mesmo atende aos requisitos estabelecidos.

- **Descrever Ambiente**: permite a descrição de novos ADSs para a Estação.

- **Sugerir Ferramenta**: permite ao Engenheiro de Software descrever uma nova ferramenta que deve, no futuro, ser desenvolvida, ou acrescentada, para compor a Estação TABA.

- **Construir Ferramentas Internas**: permite ao Engenheiro de Software a construção de ferramentas que tenham sido, anteriormente, sugeridas (descritas) para a Estação, e que irão utilizar, totalmente, o modelo de integração de ferramentas TABA.

- **Acrescentar Ferramenta Externa**: permite ao Engenheiro de Software tornar componentes da Estação TABA, ferramentas que tenham sido desenvolvidas sem utilizar sua filosofia de integração (ferramentas externas), mas que, mesmo assim, necessitam ser utilizadas num determinado ADS.

### V.5.1.2 A Classe Interface com o Usuário

A classe *Interface com o Usuário* descreve as características essenciais dos objetos responsáveis pela iteração do usuário com o ambiente e do usuário com as ferramentas integradas ao ADS.

O padrão de interface deve ser definido a partir de especializações de classes pré-determinadas, disponíveis no sistema computacional, e preferencialmente devem utilizar recursos de apresentação orientados a janelas, com disponibilidade de recursos gráficos.

### V.5.1.3 A Classe Conhecimento

A classe *Conhecimento* reúne os objetos que tem a capacidade de representar o conhecimento referente a métodos, processo de desenvolvimento e domínios de aplicação. É a partir destes objetos que as ferramentas obtêm as informações referentes aos elementos utilizados por cada método, de forma a poder realizar um mapeamento do mundo de objetos para o mundo do usuário.

A instanciação da classe Conhecimento é realizada a partir da representação e modelagem do conhecimento de cada método, através do preenchimento de uma estrutura de conhecimento, definida por Duarte, Travassos, Rocha e Pena [Duarte91a], [Duarte91b].

De acordo com esta abordagem, para expressar os métodos e suas características particulares, é necessário:

- estabelecer a forma de proceder do método, ou seja, como se origina e se deriva. Deve-se expressar o seu comportamento através de um conjunto de atitudes. Este tipo de conhecimento é denominado "Procedimentos";

- identificar todo o conhecimento que pode ser apreendido, ou seja, tudo aquilo que pode ser manipulado e perceptível para o entendimento do método. A partir daí, é necessário e possível expressar o tipo de associação entre esses conhecimentos identificados. Este tipo de conhecimento é denominado "Objetos/Relações";

- permitir se fazer a descrição dos objetos do método, ou seja, expor de forma circunstanciada e narrativa o objetivo e/ou característica do conhecimento identificado. Este tipo de conhecimento é denominado "Descrição";

- determinar a distribuição em classes e/ou grupos, segundo um sistema ou método de classificação, ou seja, os conhecimentos que representam categorias. Este tipo de conhecimento é denominado "Classificação", e,

- especificar verdades de formação irregular a partir de experiências de especialistas. Este tipo de conhecimento é denominado "Heurísticas".

Uma base de conhecimento para métodos de desenvolvimento de software deve conter os conhecimentos descritos acima, e deve ser construída utilizando um modelo misto de representação do conhecimento composto por redes semânticas, quadros e regras de produção.

Neste momento, apenas o conhecimento relativo a "Objetos/Relações" será considerado, e a estrutura para representação do conhecimento se restringirá a uma estrutura de redes semânticas.

Com as redes semânticas obtém-se uma visão macroscópica do conhecimento, de forma adequada e de fácil entendimento. Os nós da rede identificam e rotulam o conhecimento e os relacionamentos, e expressam as conexões apropriadas ao domínio tratado.

A construção da classe *Conhecimento* é realizada através da utilização de Estruturas de Conhecimento (EC). Uma estrutura de Conhecimento identifica e descreve um conhecimento representado por um nó na rede de conhecimento. Sua função organizacional permite uma otimização da rede, onde características particulares do conhecimento são representadas por ela.

A Estrutura de Conhecimento (EC) é formada por seis estruturas com características distintas. Cada estrutura possui atributos para representar, num

determinado momento, as informações das características do conhecimento representado pelo nó. As suas estruturas são:

- **Identificação** (EC(1)): contém os conhecimentos de identificação da rede e do domínio, para cada nó da rede de conhecimento (figura V.4);

**IDENTIFICAÇÃO**

**NÚMERO:** (Número de identificação do nó na rede de conhecimento)
**REDE:** (Nome do método representado pela rede de conhecimento)
**DOMÍNIO:** (Nome do domínio de classe que a rede pertence)
**NÓ_ESTRUT:** (Determina o número do nó para o qual devem ser consideradas as estruturas de conhecimento)

Figura V.4 - Estrutura de Conhecimento EC(1) (Fonte: [Duarte91a])

- **Descrição** (EC(2)): contém os conhecimentos necessários para o entendimento do nó e de suas características básicas (figura V.5);

- **Construção** (EC(3)): contém as informações relativas à construção da rede de conhecimento (figura V.6);

**DESCRIÇÃO**

**NOME:** (Nome de identificação do nó na rede de conhecimento)
**DESCRIÇÃO:** (Descrição geral e/ou objetivo e/ou características do nó)
**FUNÇÃO:** (Determina a função do nó na rede, se R=Rede ou Q=Quadro)
**TIPO_NÓ:** (Determina a característica do tipo de nó na rede, se A=Ativo ou F=Fraco)
**SEGMENTO:** (Determina a característica do nó na rede, se é ou não um nó segmentação, S=Sim e N=Não)
**CLASF:** (Determina a característica do nó na rede, se representa um conhecimento classificatório M=Método, I=Instumento, T=Técnica ou N=Nenhum, referente ao domínio a classe)

Figura V.5 - Estrutura de Conhecimento EC(2) (Fonte: [Duarte91a])

| CONSTRUÇÃO |
| --- |
| **TIPO_CONST:** (Determina se a construção do nó é relativa às conexões A=Anterior ou P=Posterior)<br>**QUANT_CONST:** (Determina a quantidade de conexões para o nó em referência)<br>**NÓS_CONST:** (Identifica os nós anterior ou posterior ao nó em referência)<br>**RELAC_CONST:** (Identifica os relacionamentos anterior ou posterior ao nó em referência)<br>**SENT_CONST:** (Identifica os sentidos de direção dos relacionamentos em relação ao nó em referência. Identificando com O para os nós origem e D para os nós destino) |

Figura V.6 - Estrutura de Conhecimento EC(3) (Fonte: [Duarte91a])

- **Estatística** (EC(4)): contém conhecimentos quantitativos resultantes de toda a rede (figura V.7);

| ESTATÍSTICA |
| --- |
| **COMEN_ESTAT:** (Descreve um objetivo e/ou característica para a estatística)<br>**NÓS_TOTAL:** (Determina a quantidade total de nós existentes na rede de conhecimento)<br>**NÓS_ATIVO:** (Determina a quantidade de nós do tipo ativo na rede)<br>**NÓS_FRACO:** (Determina a quantidade de nós do tipo fraco na rede)<br>**NÓS_REDE:** (Determina a quantidade de nós de função rede)<br>**NÓS_QUADRO:** (Determina a quantidade de nós de função quadro)<br>**NÓS_SEGMENTO:** (Determina a quantidade de nós de segmentação existente na rede)<br>**NÓS_CLASF:** (Determina a quantidade de nós de classificação existente na rede) |

Figura V.7 - Estrutura de Conhecimento EC(4) (Fonte: [Duarte91a])

- **Classificação** (EC(5)): identifica os nós da rede que tem conhecimento que se referem a classes de métodos, instrumentos e técnicas (figura V.8);

- **Especificação** (EC(6)): expressa os conhecimentos relativos à descrição de procedimentos, regras e heurísticas (figura V.9).

| CLASSIFICAÇÃO |
| --- |
| **REFER_CLASF:** (Identifica a classe a que se referem os nós relacionados. Se M para identificar a classe de método, se I para identificar a classe de instrumento e T para identificar a classe de técnica)<br>**QUANT_CLASF:** (Determina a quantidade de nós que classificam a estrutura referenciada)<br>**NÓS_CLASF:** (Identifica os nós da rede de conhecimento que classificam a estrutura referenciada) |

Figura V.8 - Estrutura de Conhecimento EC(5) (Fonte: [Duarte91a])

De forma a permitir a descrição, e consequentemente a criação, do conhecimento foi utilizada uma linguagem (LMC - Linguagem de Manipulação de Conhecimento)[Duarte91a]. Esta linguagem, embora cumpra seus objetivos principais, não consegue relacionar o conhecimento armazenado com o comportamento necessário à representação de elementos pertencentes ao método descrito.

Com o objetivo de permitir a instanciação de redes de conhecimento e, ao mesmo tempo, o acesso a este conhecimento pelas ferramentas existentes no ambiente, solucionando a questão do comportamento associado aos elementos, estabelecemos uma extensão à linguagem, então proposta, e que denominaremos **LDMM** - **L**inguagem para **D**escrição e **M**anipulação de **M**étodos.

| ESPECIFICAÇÃO |
| --- |
| **IDENT_ESPEC:** (Determina o nome de identificação da estrutura de procedimentos)<br>**COMEN_ESPEC:** (Descreve o objetivo e/ou característica da estrutura de procedimentos)<br>**TIPO_ESPEC:** (Determina o tipo do procedimento. Se P para um procedimento genérico, R para determinar um procedimento de regras pertinente ao método, ou H para determinar a especificação de regras traduzidas por um especialista)<br>**QUANT_FATOS:** (Determina a quantidade de fatos especificados para a estrutura)<br>**FATO(SEQUÊNCIA):** ( Especifica os fatos de interesse para a estrutura)<br>**QUANT_AÇÕES:** (Determina a quantidade de ações disponíveis na estrutura)<br>**AÇÃO(SEQUÊNCIA):** (especifica as ações de interesse para a estrutura)<br>**QUANT_REGRAS:** (Determina a quantidade de regras para atuarem sobre os fatos e ações disponíveis na estrutura)<br>**REGRA(SEQUÊNCIA):** (Especifica as regras, em pares condição-ação, relativos aos fatos e ações disponíveis na estrutura)<br>**PROX_ESPEC:** (Especifica a identificação de uma próxima estrutura de especificação) |

Figura V.9 - Estrutura de Conhecimento EC(6) (Fonte: [Duarte91a])

#### V.5.1.3.1 A *L*inguagem de *D*escrição e *M*anipulação de *M*étodos

A definição da LDMM foi realizada tomando-se por base métodos pertencentes a dois paradigmas distintos de desenvolvimento. Utilizou-se para sua construção os métodos SADT [Ross85] e SSA [Gane82], representando a classe de métodos pertencentes ao paradigma de desenvolvimento estruturado, cuja característica fundamental é serem baseados num enfoque de fluxo de dados, e os métodos OOA [Coad92] e Booch [ Booch91], representando a classe dos métodos orientados a objetos.

A partir do estudo destes métodos, foi possível identificar os papéis (funções) exercidos pelos elementos pertencentes a cada método em questão, que deveriam ser representados pela LDMM. Esta forma de representação permite que seja representada a semântica do documento gerado, e facilita, na medida do possível, o aproveitamento das informações geradas por uma ferramenta, num determinado momento do processo de desenvolvimento, pelo ambiente.

Foram identificadas, para os paradigmas estruturado e orientado a objetos, as seguintes funcionalidades para os elementos componentes dos métodos:

- **DESCRITIVO:** elemento que permite a descrição de alguma informação textual complementar que auxilie a compreensão do documento.

- **TRANSFORMADOR:** elemento responsável por receber um determinado conjunto de informações, transformá-las, de acordo com uma determinada lógica, e tornar estas informações transformadas disponíveis para outros elementos.

- **FONTE:** elemento responsável por fornecer informações para o sistema. A informação fornecida por uma fonte é sempre recebida por um transformador.

- **DESTINO:** elemento responsável por receber informações do sistema. A informação recebida por um destino é sempre fornecida por um transformador.

- **ESTADO:** elemento que permite a representação de algum estado em que o outro elemento, ou o sistema, se encontra.

- **ARMAZENADOR:** elemento responsável por guardar as informações relevantes ao sistema, e fornecê-las quando solicitado. Transformadores solicitam, aos armazenadores, a guarda ou recuperação das informações.

- **ORGANIZADOR:** elemento que permite a organização de outros elementos. Pode ser utilizado para representar elementos que devem estar agrupados e apresentados de forma homogênea. Organizadores servem para agrupar

encapsuladores e suas relações, salientando a característica, ou assunto principal, que eles representam.

- **ENCAPSULADOR:** elemento que possui propriedades de encapsular informações e funcionalidades.

- **HIERARQUIA:** elemento que permite mostrar a relação hierárquica existente entre elementos encapsuladores. Os encapsuladores envolvidos numa relação hierárquica não podem ser iguais.

- **COMPOSIÇÃO:** elemento que permite relacionar elementos encapsuladores, demonstrando uma relação de composição (estruturação) entre eles.

- **ESTÍMULO:** elemento que permite representar a ocorrência de algum evento (estímulo) que está acontecendo no sistema. Este estímulo pode ser representado por estímulos entre elementos encapsuladores, ou então, a partir de ocorrências de eventos que provoquem variações dos estados do sistema.

- **DEPENDÊNCIA:** elemento que permite representar uma relação de dependência entre elementos encapsuladores.

- **TRANSPORTADOR:** elemento responsável por transportar um conjunto de informações entre elementos transformadores, fonte, destino e armazenadores. Um transportador só pode transportar informações entre os seguintes elementos:

  fonte → transformador;
  transformador → destino;
  transformador → armazenador;
  armazenador → transformador;
  transformador → transformador.

No apêndice B encontramos uma descrição dos métodos utilizados para a realização deste estudo e suas respectivas descrições em LDMM.

A **LDMM** se compõe de dois blocos que são:

1) **Bloco de Definições**: permite a definição do conhecimento do método propriamente dito. É o responsável direto pela instanciação da classe Conhecimento;

2) **Bloco de Consultas**: permite a obtenção do conhecimento armazenado num objeto. É através dele que as ferramentas conseguem obter o conhecimento armazenado.

Na sintaxe da linguagem LDMM são encontradas as seguintes palavras reservadas:

```
Def_Método    Def_Doc       Def_Comp      Def_Função    Def_Forma
Sel_Método    Sel_Doc
Qual_Forma    Qual_Tipo     Qual_Comp
Importe
Fim_Def_Método        Fim_Def_Comp        Fim_Sel_Método        Fim_Sel_Doc
```

A definição da sintaxe, feita em BNF, onde o sinal ? representa o retorno de uma informação , é apresentada abaixo:

```
<Nome> ::= <Identificador> | <Identificador> <Nome>
<Identificador> ::= A | B | ... | Z
<Nome_Método> ::= <Nome>
<Nome_Documento> ::= <Nome>
<Nome_Componente> ::= <Nome>

<Método> ::=  Def_Método <Nome_Método>
                              <Definição_Documento>
                    Fim_Def_Método | &


<Definição_Documento> ::= Def_Doc <Nome_Documento>
                                    <Definições>
                              Fim_Def_Doc |  <Definição_Documento> | &
```

```
<Definições> ::=        Def_Comp <Nome_Componente>
                              <Define_função>
                              <Define_forma>
                        Fim_Def_Comp | <Definições> | &

<Define_função> ::=  Def_Função  <Função> | <Define_função> | &

<Define_forma> ::=   Def_Forma  <Forma>  | &

<Forma> ::= Importe <Nome_Primitiva> <Lista_Parâmetros>

<Nome_Primitiva> ::= <Nome>

<Lista_Parâmetros> ::= <Parâmetro> | <Parâmetro> <Lista_Parâmetros>

<Parâmetro> ::= <Nome> | &

<Função> ::=  DESCRITIVO | TRANSFORMADOR | FONTE | DESTINO | ESTADO |
ARMAZENADOR | ORGANIZADOR | ENCAPSULADOR | HIERARQUIA | COMPOSIÇÃO
| ESTÍMULO |  DEPENDÊNCIA | TRANSPORTADOR

<Seleções> ::=          Sel_Método <Nome_Método>
                              <Consultas>
                        Fim_Sel_Método | <Seleções> | &

<Consultas> ::=         Sel_Doc <Nome_Documento>
                              <Perguntas>
                        Fim_Sel_Doc | <Consultas> | &

<Perguntas>::= Qual_Forma {<Nome_Componente> | <Função>} ? <Forma> |
Qual_Função {<Nome_Componente> | <Forma>} ? <Função> | Qual_Comp
{<Função> | <Forma>} ? <Nome_Componente> | &

<Aplicação> ::= <Método> | <Consulta>
```

### V.5.1.4 A Classe Assistente Inteligente

A classe Assistente Inteligente reúne os objetos responsáveis por assistir o usuário na utilização do ADS e de suas ferramentas. Falcão [Falcão92] realizou um trabalho nesta área, construindo um Assistente Inteligente de Apoio ao Uso da Análise Estruturada. Na Estação TABA está prevista a existência de um Assistente Configurável a diferentes ciclos de vida, métodos e ferramentas.

### V.5.1.5 A Classe Frameworks

Segundo Wirfs-Brock [Wirfs-Brock90], a utilização de "frameworks" como elemento de definição e construção de modelos de interface com usuário tem sido realizada maciçamente a partir do trabalho de Krasner [Krasner88], que demonstrou a viabilidade de utilização de programação orientada a objetos para a construção da interface com o usuário. Nesta mesma linha de utilização encontramos outras definições [Schmucker86], [Palay88], [Linton89], [Vlissides89], [Weinand89] que enriqueceram o conjunto de "frameworks" disponíveis até então. A utilização de "frameworks", entretanto, não é restrita à área de interface com o usuário. Podem ser aplicados em qualquer área de projeto de software [Johnson88], [Gossain89], [Foote88], [Jindrich90], [Madany89], [Russo89], [Zweig90], [Thompson89] e permitem aumentar, nas linguagens orientadas a objetos, a capacidade de reutilização.

A formalização e conceituação de "frameworks" foi realizada no trabalho de Deutsch [Deutsch89] que destacou a importância da reutilização da interface dos "frameworks" e da especificação de seus componentes.

Um "framework" é projetado para ser refinado, ou especializado. Ele se constitui num conjunto de classes abstratas[2] e classes concretas[3], juntamente com seus relacionamentos, que representam o comportamento necessário para algum tipo de subsistema.

Ao longo de sua vida um "framework" pode evoluir, incorporando novas características e funcionalidades, a partir da inclusão de novas especializações na sua estrutura. A maturidade de um "framework" pode ser observada pelo número de classes concretas, especializadas a partir das classes abstratas, e incluídas na estrutura. Esta inclusão é bastante facilitada pela reutilização de toda a funcionalidade disponível nas classes abstratas superiores.

---

[2]classes abstratas são classes de alto nível que, a princípio, não permitem a instanciação de objetos

[3] classes concretas são classes passíveis de instanciação

Podem ser identificadas três atividades fundamentais quando se trabalha com "frameworks". Estas atividades, que determinam o tipo de pesquisa e trabalho a ser desenvolvido, relacionam-se a aspectos de projeto, uso e descrição de "frameworks".

Projetar "frameworks" envolve a identificação de características comuns, dentre determinados domínios de aplicação. Estas similaridades representam, na maioria dos casos, subsistemas que compartilham de funcionalidades e estruturas semelhantes. Um "framework" representa então uma generalização destes subsistemas, e permite que novos subsistemas sejam construídos a partir de sua definição.

Utilizar "frameworks" envolve a definição, no caso de necessidade[4], de novas classes que devam ser incorporadas à estrutura já existente, e na configuração de um conjunto de objetos a partir do fornecimento de parâmetros para cada um deles, permitindo sua posterior conexão ao modelo.

Descrever "frameworks" envolve a definição de formalismos que permitam a perfeita descrição dos modelos. As linguagens orientadas a objetos convencionais, em geral, não fornecem nenhum suporte direto que permita a formalização da descrição. Uma linguagem orientada a objetos que pode ser utilizada para a descrição de "frameworks" é EIFFEL [Meyer90], [Meyer92]. Visando definir maneiras de se descrever "frameworks", Profrock [Profrock89] descreveu um projeto onde um dos objetivos é a formalização de "frameworks".

Adotando esta conceituação, a classe Frameworks sintetiza a idéia de se obter uma representação uniforme, e reutilizável, de estruturas organizacionais que permitam a construção de ambientes e ferramentas, no contexto da Estação TABA.

---

[4] Idealmente nenhuma nova classe seria necessária num "framework"

Existem duas especializações abstratas desta classe. A primeira diz respeito aos ambientes de desenvolvimento e representa o conjunto de informações e funcionalidades necessários num ADS. É identificada no modelo com o nome de FrameworkAmbientes. A segunda diz respeito às ferramentas existentes e que possivelmente podem estar integradas num ADS. É representada, no modelo, pela classe abstrata FrameworkFerramentas.

As especializações de mais baixo nível representam os ambientes e as ferramentas, respectivamente, existentes na Estação TABA e passíveis de utilização. A utilização de estruturas deste tipo simplificam o processo de instanciação de ambientes e de construção de ferramentas, fornecendo um conjunto de funcionalidades e características básicas, associadas aos elementos internos que constituem a Estação, que podem ser reutilizadas a cada novo desenvolvimento.

#### V.5.1.5.1 Frameworks de Ambientes de Desenvolvimento

A classe FrameworkAmbientes contém a descrição das características comuns aos Ambientes de Desenvolvimento de Software. A identificação das especificidades foi realizada a partir da definição de um conjunto de ADS específicos para cada domínio de aplicação (seção V.2). A instanciação desta classe representa um ADS genérico, contendo um conjunto mínimo de funcionalidades que permita sua utilização. Isto é verdade se forem consideradas as atividades de planejamento e repetitivas[5], comuns a todo processo de desenvolvimento.

#### V.5.1.5.2 Frameworks de Ferramentas

A classe FrameworkFerramentas reúne as características básicas e necessárias para que ferramentas sejam integradas aos ambientes instanciados, de acordo com o modelo para integração de ferramentas do TABA.

---

[5] "clerical activities"

Na estação TABA são identificados dois tipos básicos de ferramentas: as ferramentas internas e as ferramentas externas.

As ferramentas internas são aquelas que foram construídas levando em consideração toda a estrutura TABA. Estas ferramentas estão incorporadas ao ambiente de forma natural, fazendo parte do mundo de objetos TABA e gerando informações que podem ser diretamente manuseadas no ambiente, e conforme a necessidade, reutilizadas por outras ferramentas. A forma de organização das informações será descrita na seção V.5.2 quando for discutido o modelo dos ambientes instanciados TABA.

As ferramentas externas representam aquelas ferramentas desenvolvidas sem utilizar a estrutura TABA. Quando é necessária sua integração a um ambiente, estas ferramentas são encapsuladas na estrutura TABA, e passam a fazer parte do mundo de objetos TABA[6]. O grau de integração destas ferramentas aos ambientes depende da forma como se consegue realizar o tratamento das informações geradas pela ferramenta no contexto do ambiente. Para isto pode ser utilizado na integração da ferramenta externa um tradutor ("driver"), que permita a transcrição das informações da ferramenta para o ambiente e do ambiente para a ferramenta. Todo o controle de ativação destas ferramentas passa a ser feito pelo ambiente instanciado, permitindo o acompanhamento de sua utilização e sua eventual participação nas atividades do processo de desenvolvimento.

## V.5.2 O MODELO DOS AMBIENTES INSTANCIADOS TABA

Na seção V.5.1 foi descrito o meta ambiente TABA que é o responsável pela definição e instanciação do ADS. Estes ambientes, por sua vez, são compostos de ferramentas, sejam elas internas ou externas, que necessitam

---

[6] Uma proposta semelhante, aplicada à reutilização de código fonte, é utilizada pelo sistema CAOS [Werner92].

trabalhar de forma integrada. Esta integração pode ser obtida conforme mostra o diagrama de assuntos apresentado na figura V.10 (no apêndice A pode ser encontrado o modelo completo dos ambientes instanciados na Estação TABA), que descreve a seguinte situação, onde as classes encontradas no modelo estão escritas em *itálico*:

> Um *ADS* existe para o desenvolvimento de algum *projeto*. Este *projeto*, por sua vez, pode ser desenvolvido por *equipes*, que são compostas por *pessoas*, responsáveis ou não, pelas *atividades* ao longo do processo de desenvolvimento. Cada *atividade*, ao longo do processo, gera um conjunto de *documentos*. Estes documentos, que representam informações específicas do processo, são compostos por *grafos* manuseados pelas *ferramentas*, construídas para terem *conhecimento* sobre um determinado método de desenvolvimento. A forma de comunicação destas ferramentas com o usuário é determinada pela *interface com o usuário*, que é comum a todas as *ferramentas*, e ao próprio *ADS*.
>
> O usuário pode solicitar, ao longo do processo de desenvolvimento, alguma assistência ao *ADS*, que possui um *assistente inteligente* específico para o domínio da aplicação considerado pelo ADS.

> A ordem de execução das *ferramentas* está relacionada à ordem de acontecimento das *atividades*, ao longo do processo de desenvolvimento. Cabe ao ADS certificar a ocorrência e término das *atividades*, e efetivar a manutenção das informações necessárias à realização de ensaios e medidas, a respeito da execução da atividade, que permitam um planejamento futuro mais adequado.

Adotando este enfoque de trabalho será realizada, a seguir, uma descrição das classes existentes no modelo de ADS TABA.

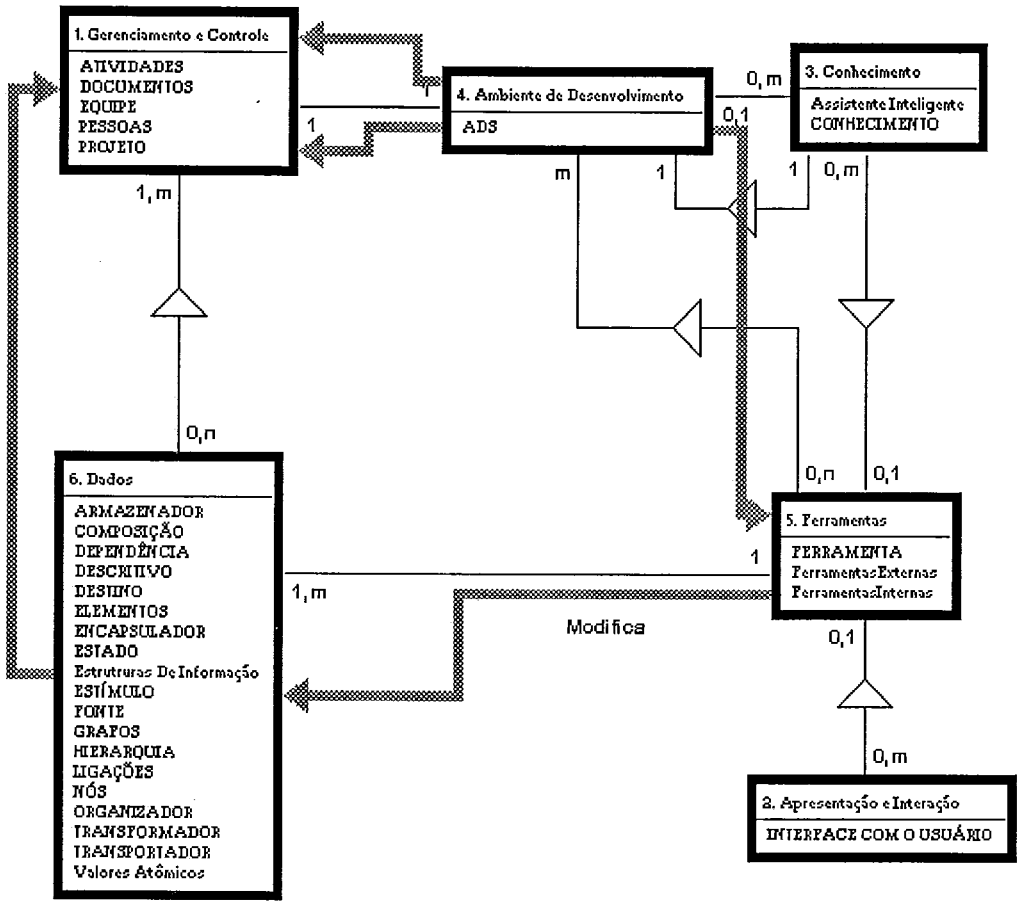



Figura V.10 - Diagrama de Assuntos dos Ambientes Instanciados TABA

### V.5.2.1 A classe GRAFOS

A classe GRAFOS representa a estrutura de representação da informação do modelo de integração para ferramentas da Estação TABA. Este tipo de organização da informação baseia-se num **modelo de grafos**, onde o conhecimento necessário para a descrição e representação de cada elemento pertencente a um determinado método é descrito a partir de uma linguagem específica e armazenado numa base de representação de conhecimento (seção V.5.1.3). A estrutura de grafos é representada através da utilização do paradigma de orientação a objetos, permitindo que o comportamento desta estrutura reflita as modificações sofridas

ao longo do processo de desenvolvimento de um produto de software. Outros ambientes [Jino85], [Giavitto89a], [Kiesel93] utilizam esta forma de organização da informação, sem se preocupar, entretanto, em representar o conhecimento associado.

Um grafo é composto por *Elementos*. Um elemento de um grafo é reconhecido no ambiente através da atribuição de seu **tipo** e **identificação.** A estrutura de tratamento de informação declarada e composta desta forma define então um **grafo tipado**. O **tipo** do elemento identifica o conjunto de funções primitivas necessárias à representação de seu comportamento, ao mesmo tempo que qualifica **subclasses** da classe primitiva *Elementos* que existam no ambiente. Os *Elementos* de um grafo podem ser de dois tipos: *nós* e *ligações*.

*Nós* representam os vértices do grafo e podem assumir os seguintes papéis, ou funções, representados por subclasses de nós:

- **DESCRITIVO:** nó que permite a descrição de alguma informação textual complementar que auxilie a compreensão do documento.

- **TRANSFORMADOR:** nó responsável por receber um determinado conjunto de informações, transformá-las, de acordo com uma determinada lógica, e tornar estas informações transformadas disponível para outros elementos.

- **FONTE:** nó responsável por fornecer informações para o sistema.A informação fornecida por uma fonte é sempre recebida por um transformador.

- **DESTINO:** nó responsável por receber informações do sistema.A informação recebida por um destino é sempre fornecida por um transformador.

- **ESTADO:** nó que permite a representação de algum estado que outro elemento, ou o sistema, se encontra.

- **ARMAZENADOR:** nó responsável por guardar as informações relevantes ao sistema, e fornecê-las, quando solicitado. Transformadores solicitam, aos armazenadores, a guarda ou recuperação das informações.

- **ENCAPSULADOR:** nó que possui propriedades de encapsular informações e funcionalidades.

- **ORGANIZADOR:** nó que permite a organização de outros elementos. Pode ser utilizado para representar elementos que devem estar agrupados e apresentados de forma homogênea. Organizadores servem para agrupar encapsuladores e suas relações, salientando a característica, ou assunto principal, que eles representam.

*Ligações* representam as arestas de um grafo e podem assumir os seguintes papéis, ou funções, representados por subclasses de ligações:

- **HIERARQUIA:** ligação que permite mostrar a relação hierárquica existente entre elementos encapsuladores. Os encapsuladores envolvidos numa relação hierárquica não podem ser iguais.

- **COMPOSIÇÃO:** ligação que permite relacionar elementos encapsuladores, demonstrando uma relação de composição (estruturação) entre eles.

- **ESTÍMULO:** ligação que permite representar a ocorrência de algum evento (estímulo) que está acontecendo no sistema. Pode ser representado por estímulos entre elementos encapsuladores ou, então, a partir de ocorrências de eventos que provoquem variações dos estados do sistema.

- **DEPENDÊNCIA:** ligação que permite representar uma relação de dependência entre elementos encapsuladores.

- **TRANSPORTADOR:** ligação responsável por transportar um conjunto de informações entre elementos transformadores, fonte, destino e armazenadores.

Um transportador só pode transportar informações entre os seguintes elementos:

fonte → transformador;
transformador → destino;
transformador → armazenador;
armazenador → transformador;
transformador → transformador.

Adotando esta estruturação, um grafo, segundo a visão do modelo para integração de ferramentas TABA, pode ser identificado por um conjunto **G**, definido como:

**G = {E}**
onde
**E = { N, L},**
e
**N = { N1, N2, ...., Ni } , i >= 1; e**
**L = { (N1, N2), ..... ,(Ni, Nj) }; i >=1, j >= 1 ; ou**
**L = { } ; i = 0, j = 0;**

onde **N** representa o conjunto não vazio formado pelos **nós Ni** e **L** representa as ligações **(Ni, Nj)** existentes entre o nó **Ni** e o nó **Nj**, nesta ordem. Este tipo de estrutura é denominado **grafo direcionado (dígrafo)**. A possibilidade da existência de uma **ligação** entre um mesmo nó (laço) é garantida através da condição **i = j** associada à ligação **(Ni, Nj)**. Além disto, pode-se observar que, a um **nó** é dada a possibilidade de se relacionar com mais de um **nó** a partir da condição **i >= 1, j >= 1**.

Considerando as extensões realizadas na estrutura de grafos definiremos a estrutura utilizada no modelo de integração como sendo um **dígrafo tipado direcionado configurável**. Adotaremos a partir deste ponto, apenas, o nome **grafo** para definir uma estrutura deste tipo.

Um documento **D**, produzido numa fase **f**, é composto pelos vários **grafos** gerados nesta fase. Assim podemos definir :

$$D_{i,f} = \mathbf{U}\ G_{k,f}\ ,\ f > i >= 0,\ k >= 0;$$

O conjunto de documentos pertencentes a uma determinada atividade do processo de desenvolvimento constitue o subproduto gerado nesta atividade e é identificado por:

$$P_f = \mathbf{U}\ D_{i,f},\ f > i >= 0.$$

Uma fase que ainda não tenha sido desenvolvida é satisfeita pela relação:

$$P_f = D_{0,f}.$$

Estas relações podem ser utilizadas pelo ADS para a verificação do desenvolvimento de alguma atividade e o controle de ativação das ferramentas para a execução de atividades subsequentes do processo de desenvolvimento.

### V.5.2.2 A classe EQUIPE

A classe *Equipe* descreve as características das equipes envolvidas no desenvolvimento de algum produto de software (projeto). Estas equipes são compostas por várias pessoas, com o objetivo de cumprir o melhor possível as estimativas de prazo, custo e qualidade atribuídas para um determinado projeto.

### V.5.2.3 A classe PESSOAS

A classe *Pessoas* representa os integrantes das equipes de desenvolvimento que participarão de algum projeto. De maneira geral, pessoas desempenham atividades ao longo do processo de desenvolvimento, podendo, em algumas

situações, assumir o papel de liderança, assumindo funções de coordenação e gerência.

### V.5.2.4 A classe PROJETO

A classe *Projeto* representa o produto que está sendo desenvolvido. Um projeto é desenvolvido através do trabalho de equipes de desenvolvimento, apoiadas por um Ambiente de Desenvolvimento de Software integrado. O processo de desenvolvimento ocorre com a execução de um conjunto de atividades, planejadas de acordo com o domínio da aplicação, o paradigma de desenvolvimento e o ciclo de vida adotados.

### V.5.2.5 A classe ATIVIDADES

A classe *Atividades* representa as atividades ao longo do processo de desenvolvimento. Estas atividades são, normalmente, realizadas por pessoas, integrantes de alguma equipe de desenvolvimento, e podem ter, como resultado, um conjunto de documentos associados. O acompanhamento da execução destas atividades, por parte do ADS, permite obter dados concretos sobre o tempo, esforço e custo, o que possibilita sua comparação com os valores estimados e, consequentemente, um melhor planejamento, no futuro, de processos de desenvolvimento de software.

### V.5.2.6 A classe DOCUMENTOS

A classe *Documentos* descreve as características dos documentos que devem ser gerados nas atividades do processo de desenvolvimento. Estes documentos são compostos por grafos, que por sua vez são construídos e manuseados pelas ferramentas, possuindo um formato associado. A partir deste formato, o Engenheiro de Software pode definir a forma de composição dos

documentos, garantindo, a partir da utilização de padrões pré-estabelecidos, a qualidade ao longo do processo de desenvolvimento.

### V.5.2.7 A classe ADS

A classe *ADS* representa um ambiente que tenha sido definido, instanciado e testado pelo meta-ambiente TABA. A descrição desta classe representa uma instância da classe *FrmAmbientes* (seção V.5.1.5.1)

### V.5.2.8 A classe FERRAMENTAS

A classe *Ferramentas* representa as ferramentas disponíveis na Estação TABA, e que se encontram integradas ao ADS. A descrição desta classe representa uma instância da classe *FrmFerramentas* (seção V.5.1.5.2).

### V.5.2.9 A classe CONHECIMENTO

A classe *Conhecimento* (seção V.5.1.2) reune os objetos que têm a capacidade de representar o conhecimento referente a métodos, processo de desenvolvimento e domínios de aplicação.

### V.5.2.10 A classe ASSISTENTE INTELIGENTE

A classe *Assistente Inteligente* (seção V.5.1.4) reune os objetos responsáveis por dar suporte ao usuário na utilização do ADS e de suas ferramentas.

### V.5.2.11 A classe INTERFACE COM O USUÁRIO

A classe *Interface com o Usuário* (seção V.5.1.2) descreve as características essenciais dos objetos responsáveis pela iteração do usuário com o ambiente e do usuário com as ferramentas integradas ao ADS.

### V.5.2.12 A classe ESTRUTURAS DE INFORMAÇÃO

A classe *Estruturas de Informação* representa as informações que são transportadas pelos *Transportadores*. Estas estruturas se prestam à descrição de agregados de informações tratados pelo produto de software que está sendo desenvolvido a partir da utilização do ADS. Na composição de estruturas de informação são utilizados *Valores Atômicos*, que representam as informações elementares no contexto do produto.

# Capítulo VI

# O Protótipo da Estação TABA

## VI.1 Introdução

De forma a tornar operacional a Estação TABA a partir da estrutura especificada no capítulo V, foram realizadas várias atividades.

Inicialmente foram construídos ambientes integrados que utilizavam enfoques diferenciados de integração. Em FEGRES [Travassos90], [Robson91] utilizou-se, como premissa para integração, a padronização da interface com o usuário, obtida através da utilização do ambiente MS-Windows [Microsoft88] e a integração de dados, obtida a partir da implementação de um gerenciador de informações que organiza as informações em tabelas pré-definidas [Trotta90] e que permite às ferramentas do ambiente compartilhar as informações geradas na especificação de um produto.

Posteriormente, realizou-se uma modificação no enfoque utilizado de forma a transportar FEGRES para o ambiente UNIX e utilizar novos padrões para a integração de ferramentas [Vilanova93]. A integração de interação e apresentação passou a ser realizada a partir do padrão Sun-OpenLook e a integração de dados realizada a partir de um gerenciador de dados, baseado no modelo relacional, acrescido de um servidor de mensagens que permite o atendimento simultâneo às ferramentas do ambiente.

Em paralelo a este desenvolvimento foram realizadas pesquisas no sentido de identificar qual paradigma e, consequentemente, que métodos e linguagens seriam utilizados na construção da Estação TABA. Os estudos realizados [Travassos91], [Duarte91a], [Duarte91b], [Gonçalves93], [Travassos92b], [Travassos93a] culminaram na estrutura apresentada no capítulo V.

Na seção VI.2 são descritas as características básicas do protótipo da Estação TABA. Na seção VI.3 é apresentado o protótipo e suas disponibilidades funcionais. Na seção VI.4 são apresentadas as soluções utilizadas para a implementação, das facilidades descritas na seção VI.3, no contexto do sistema gerenciador de banco de dados orientado a objetos $O_2$[$O_2$ 93].

## VI.2 Características Básicas do Protótipo da Estação TABA

Uma primeira implementação do protótipo foi realizada na linguagem Eiffel [Meyer88], [Meyer90], [Meyer92], utilizando a versão 2.3 do compilador. A linguagem Eiffel foi escolhida por ser a que melhor permitia, num conjunto de linguagens estudadas [Travassos92b], a implementação do modelo das classes que representam os objetos responsáveis pela integração das ferramentas na Estação TABA. Embora com estas qualidades, não foi possível a total adoção da linguagem Eiffel no modelo TABA, devido a deficiências nas classes que tratam da Interface com o Usuário, que exigiriam um esforço considerável do tempo de desenvolvimento para sua perfeita utilização (pelo menos na versão do compilador utilizada). Estas deficiências, aliada à necessidade de se obter recursos mais adequados de armazenamento e recuperação dos objetos[1], fizeram com que se buscasse a utilização de um ambiente capaz de resolver, pelo menos em parte, estes problemas.

---

[1] Eiifel possui recursos para persistência, mas não realiza nenhum tipo de gerenciamento nos objetos persistentes

A disponibilidade do sistema gerenciador de banco de dados orientado a objetos $O_2$ [$O_2$93] levou à sua experimentação, e posterior adoção, para o desenvolvimento do segundo protótipo da Estação TABA.

O sistema $O_2$ é um ambiente que possui diversos recursos de apoio ao desenvolvimento. Sua linguagem de programação, $O_2C$, contém as características fundamentais das linguagens orientadas a objetos, embora não possa ser considerada uma linguagem pura[2] permitindo a implementação (embora sem o rigor e a elegância propiciados pela linguagem Eiffel) dos requisitos da Estação TABA. Assim sendo, por propiciar a construção da interface com o usuário e ser um meio de armazenamento dos objetos, chegou-se à conclusão de sua adequação ao projeto.

Desta forma, em sua primeira versão operacional, a Estação TABA conta com um protótipo implementado no ambiente $O_2$, executável em estações de trabalho SunSparc, com pelo menos 32Mb de memória principal, 60Mb de espaço em disco disponível e dispositivo de apresentação com capacidade gráfica, sob o sistema operacional SunOs (Unix) versão 4.1.1 ou superior.

Os requisitos implementados neste protótipo são os descritos no capítulo V e representados pelos diagramas do apêndice A.

## VI.3 Funcionalidades do Protótipo da Estação TABA

O protótipo da Estação TABA possui funcionalidades que permitem ao Engenheiro de Software realizar a definição e a instanciação de um ADS, a partir das classes existentes, e implementadas, que estão descritas no modelo da arquitetura da estação (apêndice A).

---

[2] a linguagem de programação do ambiente $O_2$ foi construída a partir da linguagem C

A utilização da Estação TABA baseia-se na escolha de ações, por parte do Engenheiro de Software, que são indicadas através de cardápios específicos para cada assunto que deve ser tratado a nível do meta-ambiente. Ao inicializar a Estação uma janela de apresentação (figura VI.1) é apresentada, identificando a versão da Estação TABA e indicando que, a partir deste momento, a Estação está pronta para a execução das atividades de definição de ADSs.

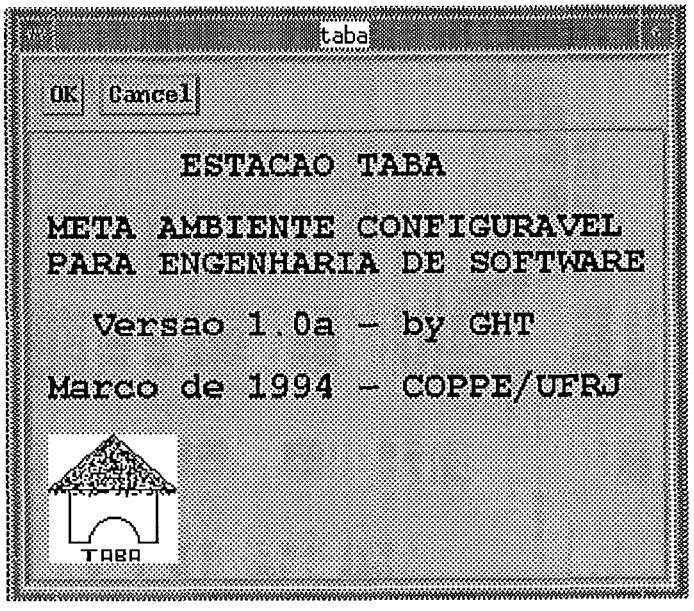



Figura VI.1 - Janela de apresentação da Estação TABA

Para que o Engenheiro de Software possa continuar seu trabalho, ele precisa confirmar, através do botão OK, a continuidade da execução. Após a confirmação é apresentado um cardápio principal (figura VI.2), que permite ao Engenheiro de Software escolher que tipo de atividade deseja executar no meta-ambiente. Estas atividades estão diretamente relacionadas aos serviços identificados para a classe Estação TABA, conforme descrito no capítulo V. Para cada atividade existe um cardápio associado, com recursos de ajuda descrevendo o que pode ser feito no cardápio em questão, que permite a execução das atividades relacionadas.

Neste ponto, existem opções para atividades relativas a:

- ambientes (figura VI.3), que permitem ao Engenheiro de Software definir, instanciar e testar um ADS no contexto TABA;

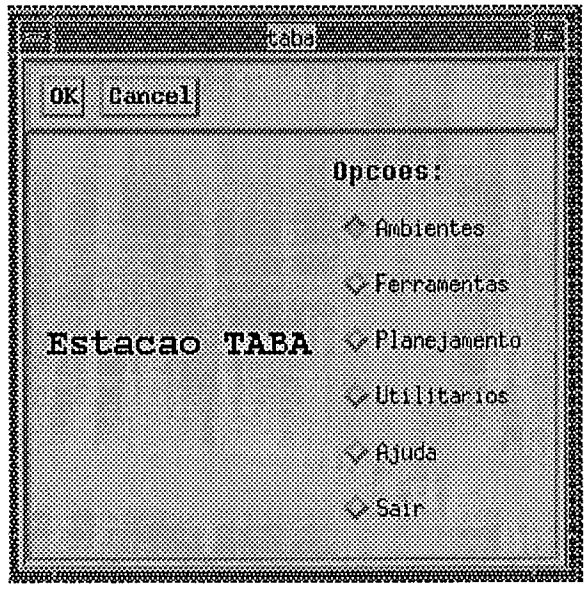



Figura VI.2 - Cardápio Principal da Estação TABA

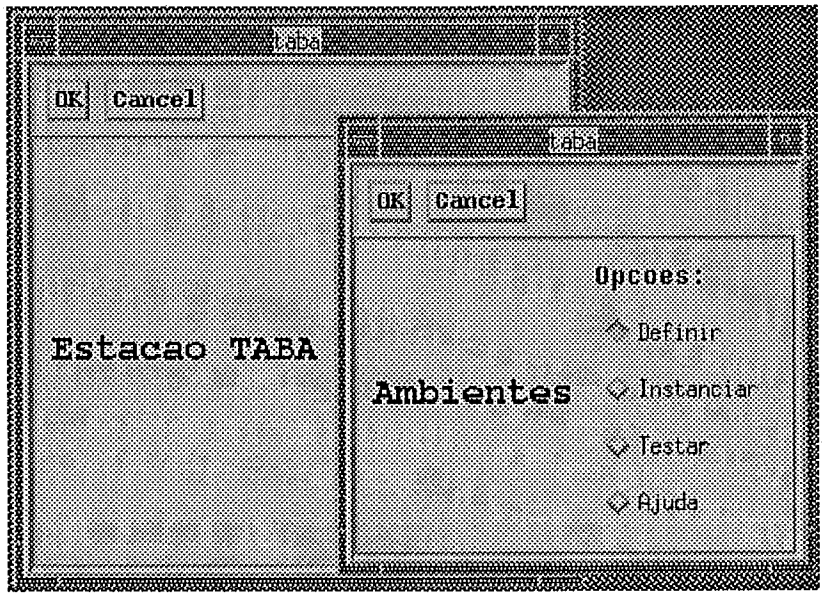



Figura VI.3 - Opção Ambientes do cardápio principal

- ferramentas (figura VI.4), que permitem ao Engenheiro de Software sugerir, construir ou acrescentar ferramentas na Estação TABA. Neste ponto ele pode, também, descrever um determinado método para o qual uma ferramenta tenha sido sugerida;

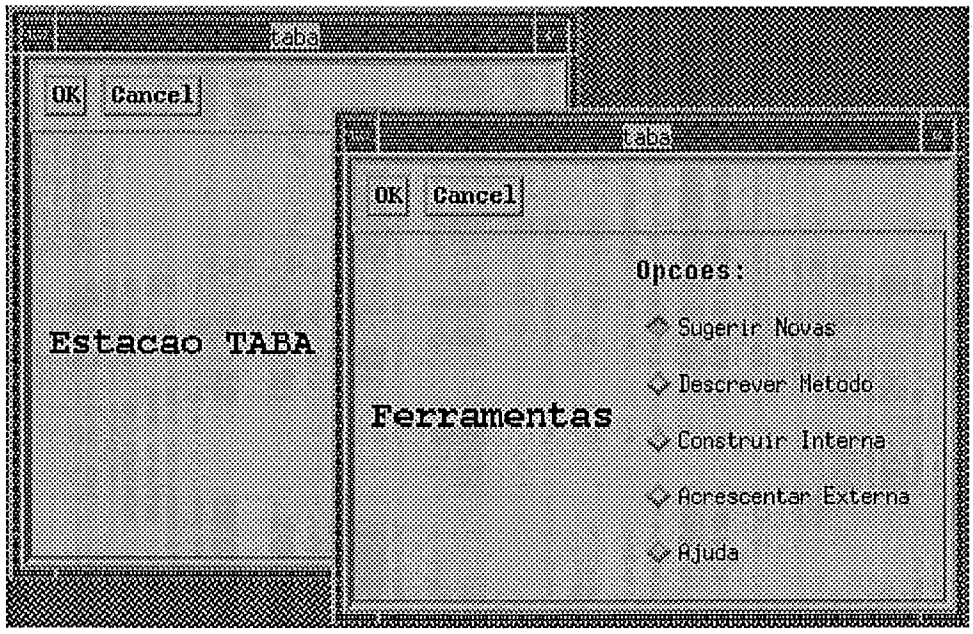



Figura VI.4 - Opção Ferramentas do cardápio principal

- planejamento (figura VI.5), que permite ao Engenheiro de Software realizar o planejamento das atividades, métricas e documentos que irão compor o ADS;

- utilitários (figura VI.6), que permite ao Engenheiro de Software realizar atividades repetitivas na Estação TABA;

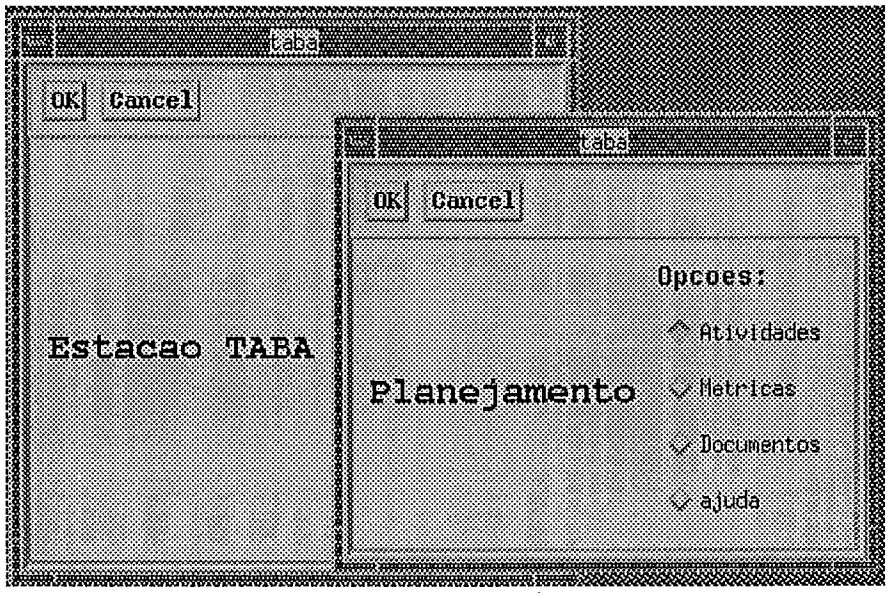



Figura VI.5 - Opção Planejamento do cardápio principal

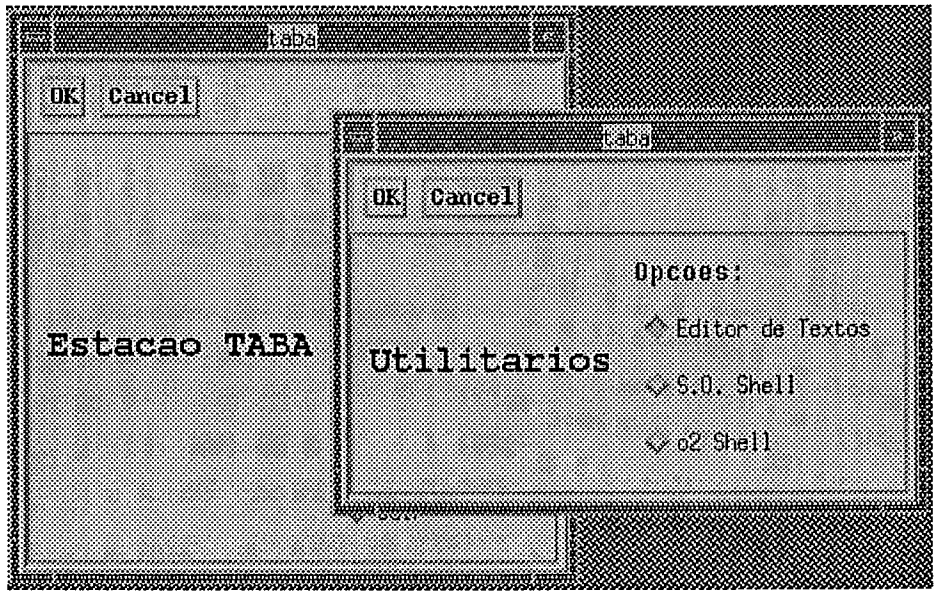



Figura VI.6 - Opção Utilitários do cardápio principal

O Engenheiro de Software deve utilizar a Estação TABA considerando os seguintes aspectos:

Para cada projeto, que pertence a um determinado domínio de aplicação, precisa ser definido um novo ADS, caso não exista algum ADS já definido para uma aplicação semelhante. Na definição do ADS serão sugeridas ferramentas à Estação TABA, por parte do XAMÃ ou do próprio Engenheiro de Software, o que posteriormente permitirá sua construção ou incorporação à Estação. Estas ferramentas suportam, na maioria das vezes, o trabalho em algum método de desenvolvimento de software específico. Para estes métodos, caso ainda não existia sua descrição na Estação TABA, esta precisa ser feita.

Cada nova ferramenta incluída na Estação TABA, seja ela externa ou interna, representa uma nova classe, especialização dos "frameworks" já existentes, passível de instanciação e utilização nos ambientes.

Para cada ADS definido, pode ser realizado um planejamento prévio de sua execução, por parte do Engenheiro de Software, que com isto facilita a utilização imediata do ADS instanciado, através da inclusão de planos para as atividades, métricas e documentos manuseados pelo ambiente. Estas características são incorporadas às classes que representam o ADS, para um determinado domínio de aplicação, já definidas nos "frameworks" existentes na Estação TABA.

Tendo realizado estas tarefas, o Engenheiro de Software pode, então, instanciar o ADS e, posteriormente, testá-lo, verificando se satisfaz aos requisitos estabelecidos.

A aprovação do ADS implica na criação de uma classe no meta-ambiente cujas instâncias representarão ambientes especificamente construídos para o tipo de projeto e o domínio da aplicação estabelecidos.

Este novo ambiente, em conjunto com as ferramentas incluídas para sua instanciação e seu planejamento, tornam-se disponíveis no meta-ambiente para serem reutilizados em instanciações de outros ambientes.

## VI.4 Abordagens de Implementação no ambiente $O_2$

O protótipo da Estação TABA foi implementado de forma a permitir que evoluções posteriores sejam facilmente incorporadas ao modelo. Esta flexibilidade garante à Estação TABA a possibilidade de acréscimo de melhorias nas atividades já existentes, e permite que novos trabalhos se utilizem, em sua elaboração, da estrutura já construída.

Na implementação do protótipo foram utilizados os recursos disponíveis no O2. Desta forma, todas as soluções utilizadas são descritas a partir dos recursos existentes.

### VI.4.1 Considerações a respeito da Interface com o Usuário

O padrão de Interface com o Usuário, disponível no ambiente, é o Motif (X-Windows). Os recursos de interface utilizados pela Estação TABA, ambientes instanciados e ferramentas estão padronizados. As técnicas de iteração e apresentação são consistentes ao longo de todos os objetos envolvidos na Estação TABA[3].

[3] Esta afirmação não inclui as ferramentas externas desenvolvidas sem levar em consideração o padrão X-Windows

Os cardápios utilizados são semelhantes ao apresentado na figura VI.2. As opções de utilização para o usuário aparecem na forma de botões, os quais serão escolhidos, com exclusividade, e confirmados a partir do botão OK. Toda escolha deve ser confirmada pelo usuário. O botão Cancela permite ao usuário desistir de utilizar o cardápio que esteja ativo no momento.

Existem três formas de comunicação da Estação TABA com o usuário: mensagens, diálogos e seleção de informação.

Caixas de mensagens (figura VI.7) são utilizadas todas vez que for necessário informar ao usuário acontecimentos, ou ações, que devam ser tomadas ao longo do processamento. O usuário deve, sempre, confirmar o recebimento da mensagem (botão OK) antes de prosseguir em seu trabalho.

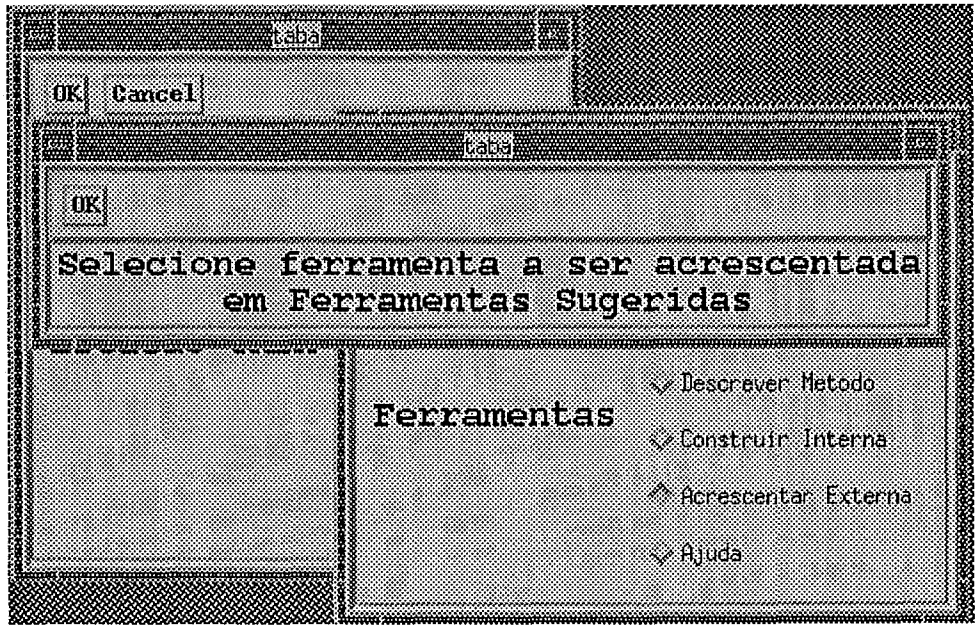



Figura VI.7 - Uma Caixa de Mensagem da Estação TABA

Diálogos (Figura VI.8) são utilizados pela Estação quando ocorre a necessidade de solicitar, ao usuário, alguma informação para o prosseguimento do processamento.

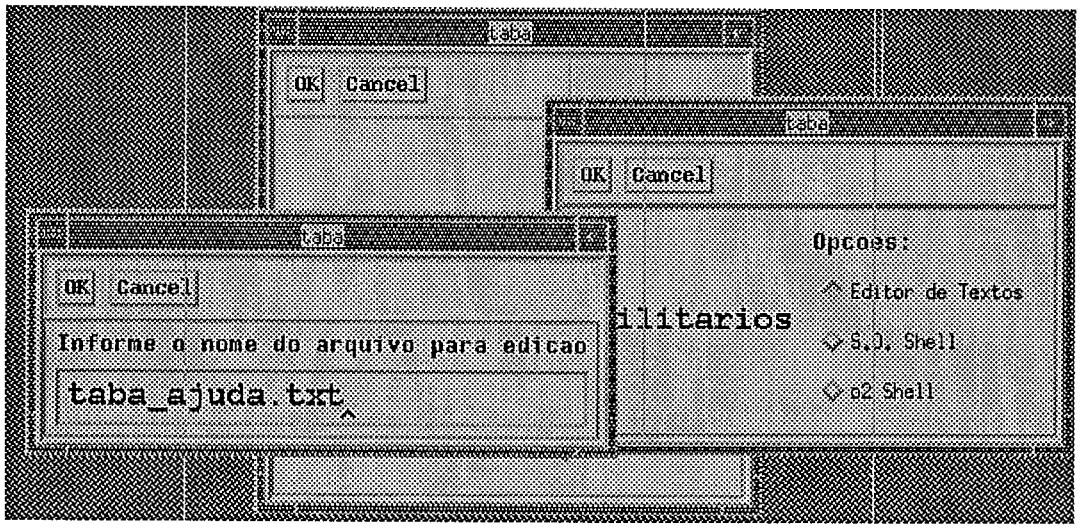



Figura VI.8 - Uma Caixa de Diálogo da Estação TABA

A seleção de informação (figura VI.9) é realizada pelo usuário a partir de um conjunto de informações apresentadas em uma janela. O usuário deve escolher a informação com o mouse e confirmar sua escolha, a partir do botão OK. Caso seja necessário para o usuário cancelar a operação, ele pode "clicar" com o mouse no botão Cancela.

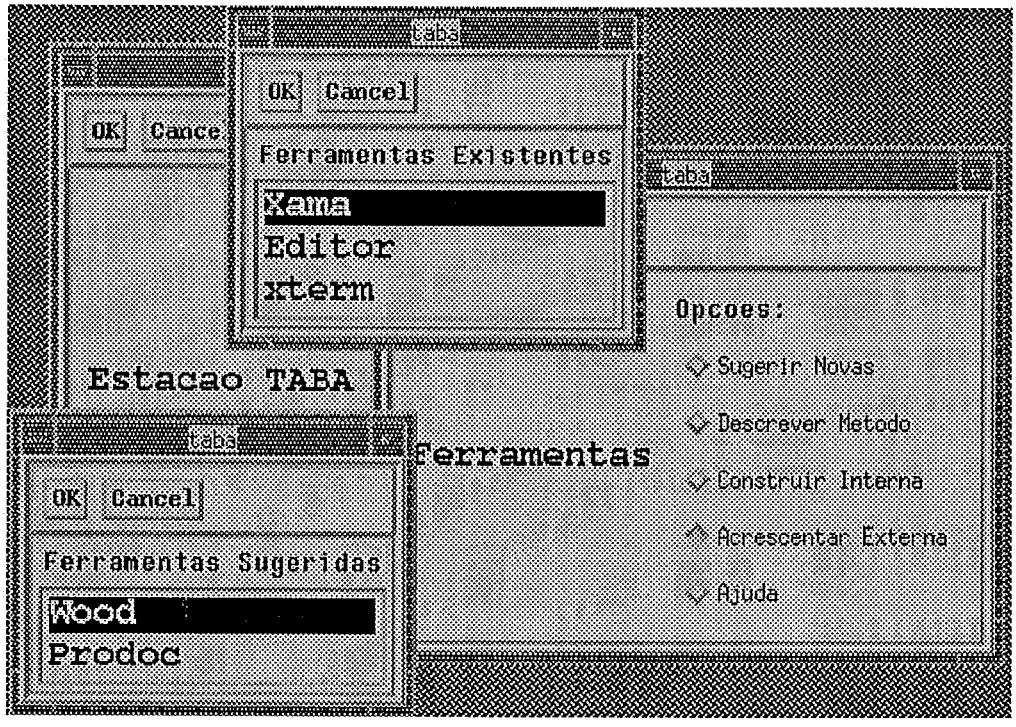



Figura VI.9 - Seleção de informações da Estação TABA

Para realizar a entrada e visualização das informações dos objetos existentes na Estação TABA, o sistema $O_2$ fornece padrões pré-definidos de diálogos, modelados de acordo com a estrutura do objeto que está sendo atualizado, ou consultado. Nestes diálogos são encontrados dois botões, um com o desenho de um lápis (botão OK), e outro com o desenho de uma borracha (botão Cancela) que servem para, respectivamente, confirmar ou cancelar a edição. Na figura VI.10 pode ser vista a caixa de edição do objeto

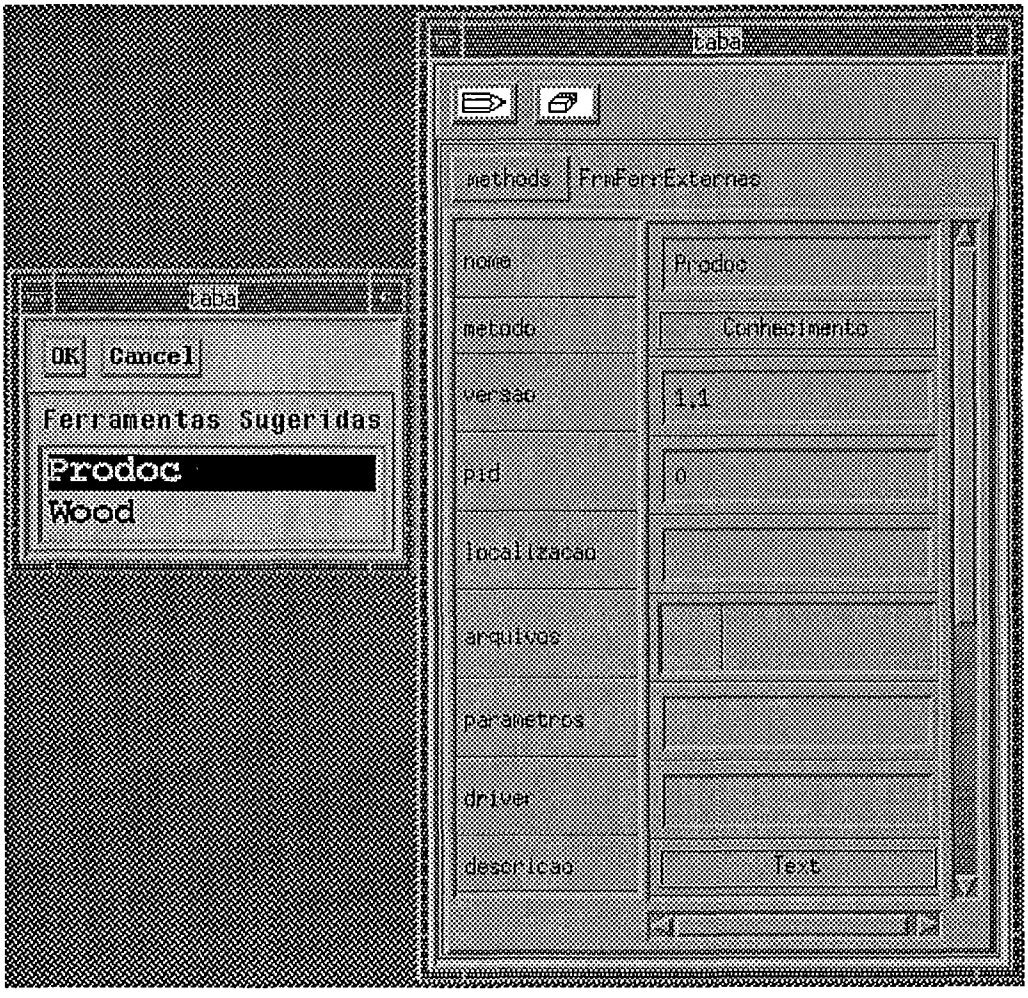



Figura VI.10 - Caixa de Edição do objeto FrmFerrExternas

Os recursos de interface com o usuário, utilizados nos ambientes instanciados e nas ferramentas internas, são semelhantes aos apresentados para a Estação TABA.

### VI.4.2 Considerações a respeito da Base de Dados

O sistema $O_2$ é um sistema gerenciador de banco de dados orientado a objetos, e como tal, permite o armazenamento e tratamento de objetos. Para definição do esquema (figura VI.11) e, consequentemente, das classes que o compõem, encontra-se disponível uma hierarquia que possui a classe *Object* como a mais primitiva. Esta hierarquia pode ser enriquecida se forem reutilizadas definições, existentes em outros esquemas previamente definidos, e que aumentam a disponibilidade de tipos de objetos (figura VI.12). As classes são definidas juntamente com a descrição do cabeçalho dos métodos associados (que definem a interface para os objetos descritos pela classe).

```
class EstacaoTABA inherit Object public type
   tuple(public ferramentas_sugeridas: set(FrmFerramentas),
         public ferramentas:           set(FrmFerramentas),
         public ambientes:             list(FrmAmbientes),
         public conhecimento:          set(Conhecimento))
 end;
method public finalizar in class EstacaoTABA;
method public acrescentar_externa(pid_pai: integer) in class EstacaoTABA;
method public descrever_metodo(pid_pai: integer) in class EstacaoTABA;
method public sugerir_ferramenta(pid_pai: integer) in class EstacaoTABA;
method private definir_ambiente in class EstacaoTABA;
method public editar_texto in class EstacaoTABA;
```

Figura VI.11 - Definição de Classe

```
import schema o2kit class Dialog_box;
import schema Meta_schema class Meta_definition;
```

Figura VI.12 - Reutilização de Classes

Os objetos, cujas classes devem estar descritas no esquema do banco de dados, podem ser determinados persistentes ou não. A descrição do esquema não implica, diretamente, na persistência dos objetos. A determinação da persistência dos objetos é feita a partir da nomeação[4] dos objetos que devem persistir. Existem formas diferenciadas de se realizar esta nomeação.

Quando existe a necessidade de que ocorra a persistência de apenas um objeto, descrito a partir de uma determinada classe, a nomeação é feita a partir de um nome e apenas uma possibilidade de instância (figura VI.13). Os objetos componentes do objeto persistente também se tornarão persistentes. Este é o caso do objeto Estação TABA. Apenas uma instância da classe Estação TABA (seção V.5.1.1 e anexo A) existe.

```
name Dialogos :Box;
name estacoes :EstacaoTABA;
```

Figura VI.13 - Nomeação de base com apenas uma instância

De outra maneira (figura VI.14), quando existe a necessidade de que um conjunto de objetos seja mantido persistente, a nomeação é realizada identificando-se um conjunto de objetos persistentes. As regras de persistência, válidas para o caso de apenas uma instância, continuam válidas neste caso para cada instância existente na base. Este é o caso da classe Documentos (seção V.5.2.6 e anexo A) existente no modelo dos ambientes instanciados TABA.

```
name ferramentas: set(FrmFerramentas);
name documentos:set(Documentos);
```

Figura VI.14 - Nomeação de base com várias instâncias

Outro aspecto, a ser considerado quanto à utilização das bases nomeadas de objetos, é a possibilidade da recuperação das instâncias existentes,

[4] esta nomeação pode representar uma composição de objetos

principalmente nas nomeações de conjuntos de objetos, a partir de comandos descritos em $O_2$Query (figura VI.15). Esta prática é utilizada ao longo da implementação do protótipo.

```
.....
/* Pega a classe na base de sugestoes */
resultado_consulta == set();
o2query(resultado_consulta,"select x from x in self->ferramentas_sugeridas
        where x->nome = $1", sugestoes[resposta]);
ferramenta = element(resultado_consulta);
.....
```

Figura VI.15 - Consulta em $O_2$Query

### VI.4.3 Considerações a respeito do Meta-Ambiente TABA

O meta-ambiente TABA, para funcionar, utiliza os recursos do meta esquema $O_2$ garantidos a partir da reutilização das classes disponíveis no sistema. Estas classes (figura VI.16) disponibilizam serviços para o tratamento de meta classes, responsáveis pela criação de novas classes no esquema. Desta forma, é possível realizar a evolução do esquema, em tempo de execução, e com isto garantir a existência de novas classes na Estação TABA.

```
import schema Meta_schema class Meta;
import schema Meta_schema class Meta_definition;
import schema Meta_schema class Meta_class;
import schema Meta_schema class Meta_type;
import schema Meta_schema class Meta_collection;
import schema Meta_schema class Meta_tuple;
import schema Meta_schema name Schema;
```

Figura VI.16 - Classes para o meta esquema

A utilização do meta esquema, por parte da Estação TABA, é que permite a definição, ao longo de sua existência, de novas especializações para os "frameworks" de ambientes e ferramentas, possibilitando a evolução constante do esquema e que se atinja, ao longo deste período, um grau de maturidade cada vez maior para os "frameworks". Quanto mais for utilizada a Estação TABA, maior será a quantidade de informação reutilizada na instanciação dos ambientes e, consequentemente, menor o esforço para esta instanciação. Na figura VI.17 pode ser visto o trecho do método da classe Estação TABA responsável por acrescentar uma ferramenta externa no esquema da Estação. Este enfoque foi utilizado pela Estação TABA, neste protótipo, para utilização do XAMÃ. O resultado da aplicação deste método pode ser visto na figura VI.18.

```
method body acrescentar_externa(pid_pai: integer) in class EstacaoTABA
{
        ......
        /* Pega todas as classes do esquema */
        todas_classes = Schema->classes;
        /* Encontra a Metaclasse desejada (FrmFerrExternas) */
        for (uma_classe in todas_classes)
                if (uma_classe.name == "FrmFerrExternas")
                        meta = uma_classe.class;
                /* Pega todas as Ferramentas Externas ja existentes */
        for(sub_classe in meta->subclasses)
                externas_existentes += list(sub_classe->name);
         /* Pega a classe na base de sugestoes */
        resultado_consulta == set();
        o2query(resultado_consulta,"select x from x in ferramentas_sugeridas where x->nome = $1", sugestoes[resposta]);
        ferramenta = element(resultado_consulta);
        /* Prepara nova ferramenta externa */
                nova_externa            = new FrmFerrExternas;
                nova_externa->nome      = ferramenta->nome;
        ......
        /* Prepara a criacao da nova classe com o mesmo nome da ferramenta */
        if(Schema->definition(nova_externa->nome) == nil){
        rc = Schema->command("create class " + nova_externa->nome + " inherit " + meta->name + " end");
         rc = Schema->command("create method public init in class " + nova_externa->nome);
        corpo  = "\n{\n";
        corpo += "self->nome      = \"" + nova_externa->nome + "\";\n";
        corpo += "self->autor     = \"" + nova_externa->autor + "\";\n";
        corpo += "self->parametros  = \"" + nova_externa->parametros + "\";\n";
        corpo += "self->localizacao = \"" + nova_externa->localizacao + "\";\n";
        if(nova_externa->shell)
                corpo += "self->shell   = true;\n";
        else
                 corpo += "self->shell   = false;\n";
        corpo += "self->funcao    = \"" + nova_externa->funcao + "\";\n";
        corpo += "self->fase      = \"" + nova_externa->fase + "\";\n";
        corpo += "};";
        rc = Schema->command("method body init in class " + nova_externa->nome + corpo);
        .......
```

Figura VI.17 - Utilização do meta esquema

```
class Xama inherit FrmFerrExternas public
end;
method private init in class Xama;
method body init in class Xama
{
        self->nome = "xama";
        self->autor = "Teresa Aguiar";
        self->parametros = " ";
        self->localizacao ="/oca/ES/guilherme/taba/meta/metabase/xama";
        self->shell = false;
        self->funcao = "Definir ADS";
        self->fase = "Meta Ambiente";
};
```

Figura VI.18 - Resultado da inclusão de uma Ferramenta Externa

A partir da inclusão desta nova classe é possível, no contexto da Estação TABA (figura VI.19), sua instanciação e, consequentemente, sua utilização por parte da própria Estação e de seus ambientes instanciados.

```
method body definir_ambiente in class EstacaoTABA
{
        o2 Xama xama = new Xama;
        self->ferramentas += set(xama);
        xama->inicia;
};
```

Figura VI.19 Utilização de uma classe criada pelo meta esquema

### VI.4.4 Organização das informações utilizadas pela Estação TABA

O uso da Estação TABA envolve, além da utilização da base de objetos, a manipulação de um conjunto de aplicações e informações que se encontram disponíveis em locais pré-determinados, organizados de forma a permitir, à Estação e aos ambientes e ferramentas integradas, um acesso rápido e facilitado.

Estas informações representam os arquivos texto necessários, para a apresentação de recursos de ajuda na Estação TABA e ambientes, o código

executável e os arquivos necessários para a execução das ferramentas externas e arquivos de ajuste da interface com o usuário. As informações estão organizadas de acordo com a hierarquia apresentada na figura VI.20.

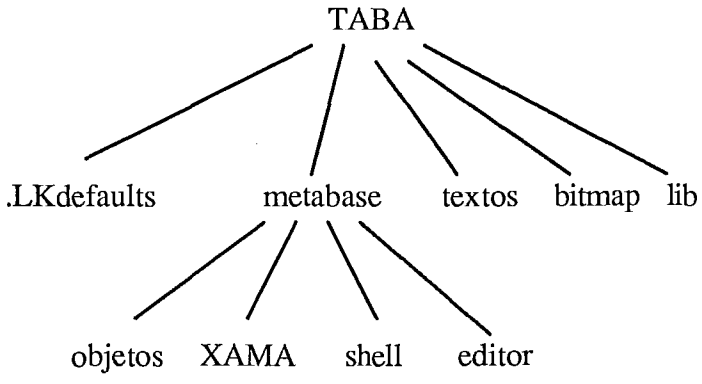



Figura VI.20 - Organização das Informações

O arquivo .LKdefaults (figura VI.21) contém a descrição dos recursos de interface com o usuário e permite, caso seja o desejo do usuário, ajustar a aparência da interface para a que melhor lhe agrade. Este ajuste pode ser feito a partir de um editor de textos convencional, disponível no cardápio de utilitários da Estação. A sintaxe detalhada do arquivo de descrição de interface com o usuário pode ser encontrada em [$O_2$93].

```
.....
*TABA.titleString:          Estacao TABA \n Meta Ambiente
*TABA.bitmap:               /oca/TABA/bitmap/taba.bitmap
*penType:                   STRING
*penString:                 Processar
*lockedPenType:             STRING
*lockedPenString:           Confirma
*eraserType:                STRING
*eraserString:              Terminar
*lockedEraserString:        Terminar
.....
```

Figura VI.21 - Arquivo .LKdefaults

No diretório *metabase* encontram-se as informações relativas aos objetos existentes na Estação TABA (ambientes e ferramentas internas) e os subdiretórios onde estão guardadas as ferramentas externas.

Os arquivos com extensão *.txt* (figura VI.21) representam os arquivos texto utilizados nas ações de ajuda, e se localizam no diretório *textos*. O nome destes arquivos deve representar, preferencialmente, que tipo de informação ele contêm.

```
./TABA/textos/taba_ajuda.txt
./TABA/textos/ambiente_ajuda.txt
./TABA/textos/ferramenta_ajuda.txt
./TABA/textos/planejamento_ajuda.txt
```

Figura VI.21 - Arquivos textos

O diretório *bitmap* contêm os bitmaps utilizados no contexto da Estação TABA e seus ambientes e ferramentas. Estes bitmaps representam ícones associados às ferramentas e ambientes, formas de representação para os componentes dos métodos de desenvolvimento cujo conhecimento tenha sido descrito na Estação TABA, e padrões de preenchimento para janelas e fundos de tela.

O diretório *lib* contém as bibliotecas de uso comum à Estação TABA e aos ambientes e ferramentas integradas, que tenham sido desenvolvidas em outras linguagens que não a normalmente utilizada no sistema O2 ($O_2C$).

### VI.4.5 Ativação da Estação TABA

O protótipo da Estação TABA necessita de algumas informações para executar. Estas informações, que devem existir no ambiente de execução do usuário, podem ser ajustadas a partir de modificações nos arquivos .cshrc (figura

VI.22) e .xinitrc (figura VI.23) existentes no diretório de trabalho (*home*). As seguintes modificações devem ser realizadas nestes arquivos:

```
setenv TABA /oca/ES/guilherme/taba
setenv O2SYSTEM taba
setenv O2SVBUFSIZE 3096
setenv O2BUFSIZE 3096
setenv O2SWAPSIZE 2048
set lpath = (/local/gnu /local/gnu/bin /oca/BD/o2/v4.3/X /oca/BD/o2/v4.3/bin)
```

Figura VI.22 - Modificações no arquivo .cshrc

```
# .xinitrc - OpenWindows startup script.
if [ -f $HOME/.Xdefaults ]; then
   xrdb $HOME/.Xdefaults &          # Load Users X11 resource database
else
   xrdb $OPENWINHOME/lib/Xdefaults &  # Load Default X11 resource database
fi
$OPENWINHOME/lib/openwin-sys &       # OpenWindows system initialization
# Install function key "F1" as an Open Look "Help" key
# This precludes its use by applications
# If your applications use F1 for anything else, comment out the following line
xmodmap -e 'keysym F1 = Help'
eval `svenv -env`                    # SunView binary compatibility
#olwm -3 &
mwm &              # OpenLook Window Manager (3-D look)
if [ -x $HOME/.openwin-init ]; then
   $HOME/.openwin-init              # Custom OpenWindows tools
else
   $OPENWINHOME/lib/openwin-init    # Default OpenWindows tools
fi
wait
```

Figura VI.23 - Modificações no arquivo .xinitrc

Após realizar estas modificações o usuário deve executar o comando *source .cshrc* para que estas informações passem a valer (isto só é necessário após

a modificação). Tendo realizado as modificações acima, o ambiente de execução está pronto para a utilização da Estação TABA. Para isto o usuário deve:

- ativar o sistema $O_2$: isto deve ser feito a partir da utilização do comando o2server&;

- ativar o sistema de interface $O_2$: através do comando o2 -toolsgraphic -L/oca/TABA/lib -ltaba

Neste capítulo foi descrito o protótipo da Estação TABA. Seu uso permitirá a população da Estação TABA com definições de ambientes e ferramentas que poderão, na medida do necessário, ser utilizadas no desenvolvimento de novos produtos de software.

# Capítulo VII

# Conclusões e Perspectivas Futuras

## VII.1 Considerações Finais

Neste trabalho foram apresentadas a evolução dos Ambientes de Desenvolvimento de Software (capítulo II), as preocupações e enfoques normalmente utilizados na construção de ADSs (capítulo III), descrições de ambientes construídos segundo os enfoques convencionais de desenvolvimento (capítulo IV) e uma solução para a construção e tratamento do problema da integração de ferramentas em ambientes (capítulo V), que permite o tratamento de todos os aspectos identificados, e necessários, para a obtenção de um ADS integrado.

Segundo Meyers [Meyers91] a solução do problema da integração de ferramentas em ambientes pode ser realizada a partir de um modelo canônico (seção III.2.1.3) o que simplificaria o processo de construção e integração de ferramentas para ambientes. O modelo construído para a Estação TABA (seção V.5) é um modelo canônico. Este modelo permite representar a sintaxe e a semântica das informações, facilitando ao ambiente, e consequentemente às ferramentas integradas, o controle e o gerenciamento das modificações nas informações ao longo do processo de desenvolvimento.

Neste trabalho foi identificada a necessidade de se tratar a integração de Ambientes de Desenvolvimento de Software de forma mais abrangente do que o usualmente proposto na literatura, permitindo o total compartilhamento das

características essenciais que determinam a integração. O enfoque de integração sendo feito a partir da consideração da homogeneidade dos serviços, possibilita uma melhor distribuição destes serviços entre as ferramentas, e não impõe nenhum tipo de sobrecarga na execução do ambiente e das ferramentas.

São contribuições deste trabalho:

- o modelo canônico para a representação de informações em ambientes;

- o enfoque de integração adotado, que considera de forma coesa quatro aspectos: dados, controle, interface e conhecimento. embora todos estes aspectos sejam tratados individualmente nos diferentes enfoques para a integração propostos na literatura, nenhum trabalho trata dos quatro em conjunto;

- a Linguagem para Descrição e Manipulação de Métodos (LDMM) que extende o trabalho anteriormente realizado com Duarte e Rocha [Duarte91a], [Duarte91b];

- o tratamento dado à questão da instanciação de ambientes e construção de ferramentas, a partir da utilização de "frameworks".

## VII.2 Perspectivas de Trabalho

O modelo proposto está, neste momento, construído de forma a representar métodos de desenvolvimento baseados no paradigma estruturado e no paradigma orientado a objetos. Devido a sua flexibilidade e a forma escolhida para construção da Estação TABA, que utiliza o paradigma orientado a objetos, é possível sua extensão, a partir da criação de novas especializações das classes existentes, para suportar outros paradigmas de desenvolvimento.

Algumas questões necessitam estudos adicionais para que se possa enriquecer a Estação TABA e, consequentemente, os ambientes instanciados na Estação.

O conhecimento, fator essencial para que ocorra a integração, necessita ser tratado de forma mais abrangente, tornando disponível o conhecimento sobre o processo de desenvolvimento para os ambientes instanciados, não apenas a nível de meta-ambiente. Este conhecimento pode ser utilizado pelo Engenheiro de Software para ajustes no ambiente instanciado, durante o processo de desenvolvimento do produto, não precisando que pequenas modificações impliquem na instanciação de um novo ambiente.

A reutilização é contemplada no modelo a partir da utilização de "frameworks" para ambientes de desenvolvimento e ferramentas. Os padrões definidos até o momento permitem a instanciação de ambientes e a construção de ferramentas específicas para os paradigmas utilizados em sua definição. Novas classes precisam ser acrescentadas a estes "frameworks", de forma a torná-los abrangentes o suficiente para representar o maior número possível de ambientes e ferramentas.

Um outro aspecto quanto à reutilização é o que diz respeito ao aproveitamento das informações já existentes a nível de processo de desenvolvimento. Embora a recuperação destas informações seja possível de ser feita diretamente da própria estrutura, é preciso que isto seja feito sob o controle do ambiente. A inclusão de serviços de recuperação, e escolha, na classe Documentos resolve esta questão e permite, nos ambientes instanciados, um aumento da produtividade no desenvolvimento.

O trabalho cooperativo é garantido, neste modelo, a partir da utilização de sistemas de armazenamento ($O_2$ e futuramente GeoTaba [Monte93],[Mattoso93]) que suportam a utilização das informações, de forma compartilhada, por vários usuários. Entretanto a definição de protocolos de comunicação, que estariam distribuídos entre as classes existentes no modelo, precisam ser realizados, e

tornados disponíveis na Estação TABA, de forma que os ambientes e ferramentas existentes possam utilizar estas características durante sua execução.

A interface com o usuário, que inicialmente se utiliza do padrão X-Windows, necessita ser definida de maneira mais específica para a Estação TABA e deve, também, permitir sua utilização em grupo.

O acompanhamento das atividades, ao longo do processo de desenvolvimento, permitirá o ajuste constante nas estimativas e poderá, em áreas ainda de difícil predição, como por exemplo orientação a objetos, fornecer informações que auxiliem o Engenheiro de Software a planejar, o mais próximo do real, novos processos de desenvolvimento.

A implementação da estrutura da Estação TABA, e de seus ambientes e ferramentas, deve ser realizada em linguagens que suportem conceitos modernos de Engenharia de Software. Neste sentido deve ser encorajada a utilização da linguagem Eiffel, em novas versões que resolvam os problemas atuais, por parte dos integrantes da equipe TABA. Esta linguagem reúne características marcantes do paradigma orientado a objetos, e deverá ser utilizada, no futuro, como a meta linguagem TABA para a definição de novos modelos de ambientes e ferramentas.

# REFERÊNCIAS BIBLIOGRÁFICAS


Abbott93 — B. Abbott, T. Bapty, C. Biegel et all; "Model-Based Software Synthesis", IEEE Software, Maio 1993

Adams89 — E.W.Adams, M. Honda and T.C. Miller; "Object Management in a CASE Environment", ACM, 11 ICSE, Pittsburg, USA, 1989

Aguiar91 — T.C. Aguiar, A.R.C. da Rocha; XAMÃ: Um Assistente Especialista de Apoio a Decisão na Especificação de Ambientes de Desenvolvimento de Software, Relatórios Técnicos do Projeto TABA, TABA-RT-5/91, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, 1991

Aguiar92 — T.C. Aguiar; Um Sistema Especialista de Suporte à Decisão para Planejamento de Ambientes de Desenvolvimento de Software, Tese de Doutorado, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, março 1992

Aiken91 — M.W.Aiken, O.R.L.Sheng and D.R.Vogel; "Integrating Expert Systems with Group Decision Support Systems" , ACM Transactions on Information Systems, Vol. 9, No. 1, Janeiro 1991

Akhras90 — F.N. Akhras, M.C.C. Costa; "Meta-Modelos para Ferramentas Genéricas Integráveis ao Ambiente SIPS", Centro Tecnológico para Informática, Instituto de Automação, Campinas, SP,1990

Alford85 — M. Alford; "SREM at the Age of Eight; The Distributed Computing Design System", IEEE Computer, vol. 18, no. 4, Abril 1985

Amaral93 — J.P. Amaral, J.F.B. Castro; "Uma Abordagem Orientada a Objetos para Construção de Ambientes Inteligentes" ,XIII Congresso da Sociedade Brasileira de Computação, XX SEMISH, Florianópolis, 1993

Angulo88 — E.D. Angulo; Um Dicionário de Dados para o Método DARTS; Tese de Mestrado, COPPE/UFRJ, dezembro de 1988

Andrade91 C.J. de Andrade; ANAFOR: um analisador de Programas FORTRAN, Tese de Mestrado, COPPE/UFRJ, abril de 1991

Ansi88 *Information Resource Dictionary Standard* (IRDS), ANSI, X3.I38-1988

Artim90 J.M. Artim, J.M. Harry and F.J. Spickhoff; "User Interface Services in AD/Cycle", IBM Systems Journal, vol 29, no. 2, 1990

Assis92 S.G. de Assis; Aquisição de Conhecimento: uma experiência em Engenharia de Software; Tese de Mestrado, COPPE/UFRJ, março de 1992

Bahia92 A.S. Bahia; Avaliação da Complexidade de Software Científico; Tese de Mestrado, COPPE/UFRJ, maio de 1992

Barnes91 B.H. Barnes and T. B. Bollinger; "Making Reuse Cost-Effective", IEEE Software, Janeiro 1991

Basili88 V.R. Basili and H.D. Rombach; "Toward A Comprehensive Framework for Reuse: A Reuse-Enabling Software Evolution Environment", Technical Report CS-TR-2158, CS Dept., University of Maryland, College Park, Md., 1988

Basili92 V.R. Basili , G. Caldiera and G. Cantone; "A Reference Architecture for the Component Factory", ACM Transactions on Software Engineering and Methodology", Vol. 1, No. 1 , Janeiro 1992

Batory92 D. Batory and S. O'Malley, " The Design and Implementation of Hierarchical Software Systems with Reusable Components", ACM - Transactions on Software Engineering and Methodology, Vol. 1 , No. 4, Outubro 92

Beach92 B. W. Beach; "Connecting Software Components with Declarative Glue", 14th International Conference on Software Engineering, Maio 11-15, 1992, Melbourn, Australy

Belchior92a A.D. Belchior; Qualidade de Software Financeiro; Tese de Mestrado, COPPE/UFRJ, junho de 1992

Belchior92b A. D. Belchior, A. R. C. da Rocha; " Características de Qualidade de Programas ", Relatórios Técnicos do Programa de Engenharia de Sistemas e Computação , ES-265/92 , Julho/1992, COPPE/UFRJ

Bidarra88 J. Bidarra; Um Analisador Estático de Programas Escritos em C; Tese de Mestrado, COPPE/UFRJ, abril de 1988

Biggerstaff87 T. Biggerstaff and C. Richter; "Reusability Framework, Assesment, and Directions", IEEE Software, vol. 4, no. 2, Março 1987

Booch91 G. Booch; *Object Oriented Design with Applications*, The Benjamin Cummings Publishing Company,Inc., 1991

Booch92a G. Booch; "The Booch Method: Notation, Part I", Computer Language, Setembro 1992

Booch92b G. Booch; "The Booch Method: Notation, Part II", Computer Language, Outubro 1992

Borras88 P. Borras and D. Clément; "CENTAUR: the system", ACM, 1988

Boudier88 G. Boudier, F. Gallo et all; "An Overview of PCTE and PCTE+", Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Development Environments, Boston, Novembro 1988

Boyd88 S. Boyd; "Software Engineering Environments", Dr. Dobb's Software Engineering SourceBook, Verão 1988

Breitman93 K.K. Breitman; HiperAutor: um método para especificação de aplicações multimidia; Tese de Mestrado, COPPE/UFRJ, setembro de 1993

Brown92a A.W. Brown and J.A.McDermid; "Learning from IPSE'S Mistakes", IEEE Software, Março 1992

Brown92b A.W. Brown, M.H. Penedo; "An Annotated Bibliography on Integration in Software Engineering Environments", ACM SIGSOFT - software engineering Notes, vol 17, no3, julho 1992

Brown92c A.W. Brown, A.N. Earl, J.A. McDermid; *Software Engineering Environments: Automated Support for Software Engineering*, McGraw-Hill Book Company, 1992

Brown93 A. W. Brown; "An Examination of the Current State of IPSE Technology", 15th International Conference on Software Engineering, Maio 17-21, 1993, Baltimore, Maryland

Budiargio93 E.K.Budiardjo; "A Custodial Application Generator for Use in a CASE Tool Environment", IEEE CASE'93 - Proceedings Sixth International Workshop on Computer-Aided Software Engineering, Singapore, Julho 19-23, 1993

Buxton91 J.N. Buxton and J.A. McDermid; "Architectural Design", in: The Software Enginerr's Reference Book, McDermid, J.A. (ed.), Butterworth-Heinemann, 1991

Cais89 Common ADA Programming support Environment (APSE) Interface Set. *Introduction to CAIS*, Setembro 1989. MIL-STD-1938A.

Camargo92 C. S. P. Camargo; FLECHA: Um Editor Gráfico Cooperativo para o Modelo de Objetos do GeoTaba, Tese de Mestrado, COPPE/UFRJ, Programa de Engenharia de Sistemas e Computação, Dezembro 1992

Campos93 G.H.B. de Campos, A.R.C. da Rocha; "Avalização da Qualidade de Software Educacional: uma possibilidade de estruturação de critérios"; Revista Informática Educativa; Bogotá, Colombia; Abril 1993

Campos94 G.H.B. de Campos, F.C.A. Campos, A.R.C. da Rocha; "A Computer Educational Environment: Paradigms, Life Cycle and Quality Evaluation", 11th International Conference on Technology and Education, University of Londres Institute of Education, Londres, england, United Kingdon, Março, 1994

Cardoso90 R.N. Cardoso; PRESTIME: Um Ambiente para predição, estimação e medição da confiabilidade de sistemas, Tese de Mestrado, COPPE/UFRJ, dezembro 1990

Cavalcante88 S.A. Cavalcante; Um Editor para o Método DARTS, Tese de Mestrado, COPPE/UFRJ, agosto de 1988

Charette86 R.N. Charette; *Software Engineering Environments: Concepts and Technology* , Intertext Publications Inc., McGraw-Hill, Inc. , NY, 1986

Chen92 M. Chen and R.J. Norman; "A Framework for Integrated CASE", IEEE Software, Março 1992

Chitman91 E. Chitman; Um Estudo sobre Modelos de Ciclo de Vida; Tese de Mestrado, COPPE/UFRJ, maio de 1991

Chroust90 G. Chroust, H. Goldman and O. Gschwandtner; "The Role of Work Mangement in Application Development", IBM Systems Journal, vol. 29, no 2, 1990

Coad92 P. Coad e E. Yourdon; *Análise Baseada em Objetos*, Editora Campus, 1992

Conradi93 R. Conradi, C. Fernstorm and A. Fugetta; "A Conceptual Framework for Evolving Software Processes", ACM SIGSOFT, vol 18, no 4, Outubro 93

Crispim88 M.E.H. Crispim; Definição de Termos em Ambientes para o Desenvolvimento de Software, Relatórios Técnicos do Programa de Engenharia de Sistemas e Computação - ES 175/88, COPPE/UFRJ, 1988

Crispim91 M.E.H. Crispim; Avaliação de Métodos de Desenvolvimento de Software; Tese de Mestrado, COPPE/UFRJ, maio de 1991

Crispim92 M.E.H. Crispim, A.R.C. da Rocha; Avaliação de Métodos para Desenvolvimento de Software, Relatórios Técnicos do Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ, ES-270/92, outubro 1992

Cybulski92 J.L.Cybulski and K. Reed; "A Hypertext Based Software-Engineering Environment", IEEE Software, Março 1992

Czejdo93 B.Czejdo, C.F. Eick and M. Taylor; "Integrating Sets, rules, and Data in an Object-Oriented Environment", IEEE Expert, Fevereiro 1993

Dart87 S.A.Dart, R.J.Ellison, P.H.Feiler and A.N. Haberman; "Software Development Environments", IEEE-Computer, Novembro 1987

Davis88 A. A. Davis et all., "A Strategy for Computing Alternative Software Development Life Cycle Models", IEEE Transactions on Software Engineering, Outubro 1988

DEC92 DEC. *Case Integration Services Base Document*, January 1992.

DeMarco79 T. DeMarco; *Structured Analysis and System Specification*, Prentice-Hall, 1979

Deutsch89 L.P. Deutsch; " Design reuse and frameworks in the Smalltalk-80 system", Software Reusability, Vol. II, T.J. Biggerstaff and A.J. Perlis, eds. ACM Press, 1989

Diaz-Herrera93 J.L. Diaz-Herrera; "The Importance of Static Structures in Software Construction", IEEE Software, Maio 1993

DNorman91 D. A. Norman; "Collaborative Computing: Collaboration First, Computing Second" , Communications of the ACM, volume 34, número 12, pp. 88-90, Dezembro 1991

Dolotta78 T.A. Dolotta, R.C. Haight and J.R. Mashey; "UNIX Time-Sharing System: The Programmer's Workbench", Bell Systemas Journal, vol. 57, no. 6, Julho-Agosto 1978

Donahue81 J. Donahue; "Cedar: An Environment for Experimental Programming", Integrated Project Support Environments, edited by J. McDermid, Londres: Peter Peregrinus, 1981

Duarte91a — M. A. Duarte; Um Sistema para representação do Conhecimentode Métodos de Desenvolvimento de Software, Tese de Mestrado, COPPE/UFRJ, Programa de Engenharia de Sistemas e Computação, Janeiro 1991

Duarte91b — M.A. Duarte, A.R.C. da Rocha, G.H. Travassos, J.C.C. Pena; Um Sistema de Representação do Conhecimento Adequado a Métodos de Desenvolvimento de Software, Relatórios Técnicos do Projeto TABA, TABA RT-6/91, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, 1991

Ecma90 — ECMA - European Computer Manufacturers Association, A Reference Model for Frameworks of Computer-Assisted software Engineering Environments, ECMA TR/55, Dezembro 1990

Ellis91 — C.A. Ellis, S.J. Gibbs, and G.L. Rein; "GROUPWARE: Some Issues and Experiences" , Communications of the ACM, Volume 34, número 1, Janeiro 1991

Ellison85 — R.J. Ellison and B.J. Staudt; "The evolution of the GANDALF System", The Journal of Systems and Software, volume 5, número 2, Maio 1985

Engels92 — G. Engels, C. Lewerentz et all; "Building Integrated software Development Environments Part I: Tool Specification", ACM Transactions on Software Engineering and Methodology, Vol. 1, No. 2, Abril 1992

Eureka89 — ESF - Eureka Software Factory. ESF Technical Reference Guide, 1989

Falcão92 — J.F.D. Falcão; Um Assistente Especialista de Apoio a Métodos de Desenvolvimento de Software; Tese de Mestrado, COPPE/UFRJ, maio 1992

Faria91 — E.C. Faria; MARTE: um Gerador de Aplicações baseado em Templates; Tese de Mestrado, COPPE/UFRJ, setembro de 1991

FCampos93a F. C. A. Campos, "Multimídia na Educação: Ambientes de Aprendizagem e Critérios de Avaliação", I Workshop Pesquisa de Tese em Informática na Educação, COPPE/UFRJ, novembro de 1993

FCampos93b F.C.A. Campos; G.H.B. de Campos e A.R.C. da Rocha; "Um Ambiente Educacional por Computador: Paradigmas, Ciclo de Vida e Avaliação da Qualidade", IV Simpósio Brasileiro de Informática na Educação, SBC, Recife, dezembro 1993

Fernström92 C. Fernström, K. Narfelt and L. Ohlsson; "Software Factory Principles, Architecture, and Experiments", IEEE Software, Março 1992

Fichman92 R. G. Fichman and C. F. Kemerer; "Object-Oriented and Conventional Analysis and Design Methodologies", IEEE Computer, Volume 25, número 10, Outubro 1992

Freeman83 P. Freeman; "Reusable Software Engineering: Concepts and Research Directions", Proc. Workshop on Reusability in Programming, Alna Peris, ed., ITT Programming, Newport, R.I., 1983, pp.2-16

Foote88 B. Foote; "Designing to facilitate change with object-oriented frameworks", Master's Thesis, University of Illimois at Urbana-Champaign, 1988

Forte89 G. Forte; "In Search of the Integrated Environment", CASE Outlook, Março/Abril 1989, pp 5-12

Gabel94 D.A. Gabel; "Software Engineering", IEEE Spectrum, Janeiro, 1994

Gallo86 F. Gallo, R. Minot et all; "The Object Management System of PCTE as a Software engineering Database Management System", 2nd ACM SIGSOFT/SIGPLAN software Engineering symposium on Practical Software Development Environments, 1986

Gane82 C. Gane and T. Sarson; *Structured Systems Analysis*, McDonnell Douglas, 1982

Garlan87 D. Garlan; Views for Tools in Integrated Environments, Carnegie-Mellon University, 1987

Ghezzi91 C. Ghezzi, M. Jazayeri and D. Mandrioli; *Fundamentals of Software Engineering*, Prentice-Hall International, Inc., 1991

Giavitto89a J. Giavitto, A. Devarenne,G. Rosuel,Y. Holvoat; "ADAGE: Utilisation de la Genericite pour Construire des Environnements Adaptables", Laboratories de Marcoussis, Centre de recherches de la CGE, Marcoussis, França, 1989

Giavitto89b J. Giavitto, A. Devarenne, G. Rosuel; "PRESTO: des objects C++ persistants pour le système d'information d'Adage", Laboratories de Marcoussis, CGE, 1989

Giavitto90 J. Giavitto, A. Devarenne, G. Rosuel, A. Mauboussin; " Design Decisions for the Incremental Adage Framework ", 1990, IEEE

Gibbs90 S.Gibbs, D. Tsichritzis et all; "Class Management for Software Communities", Communications of the ACM, vol. 33, no. 9, Setembro 1990

Goldberg83 A. Goldberg; *Smalltalk-80: The Interactive Programming Environment*, Reading Mass.: Addison-Wesley, 1983

Gonçalves93 M.V.M. Gonçalves, "Programação Orientada a Objetos - Estudo Comparativo de duas Linguagens de Programação", Trabalho de Final de Curso, Escola de Engenharia, Departamento de Eletrônica, UFRJ, Rio de Janeiro, RJ, junho/93

Gossain89 S. Gossain and D.B. Anderson; "Designing a class hierarchy for domain representation and reusability", Proceedings of Tools'89, Paris, França, Novembro 1989

Guttler91 E. Guttler; ESTIMA: uma Ferramenta para Estimação da Confiabilidade de Software; Tese de Mestrado, IME - Instituto Militar de Engenharia, julho de 1991

Haberman80 A.N. Haberman; "The Gandalf Research Project", Department of Computer Science Research Review 1978-1979, Carnegie-Mellon University, 1980

Hevner92 A.R. Hevner, S.A.Becker and L.B.Pedowitz; "Integrated CASE for Cleanroom Development", IEEE Software, Março 1992

Hoffnagle90 G.F. Hoffnagle; Preface in IBM Systems Journal, vol. 29, No.2 , 1990

Hunke81 H. Hunke; *Software Engineering Environments*, North-Holland, 1981

IGL82 IGL França, SADT Guide D'Auteur, Ref. 52 SADT, 1982

Jacobson92 I. Jacobson, M.Chisterson, P.Jonsson, G.Overgaard; *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, ACM-Press, 1992

Jarke92 M. Jarke; "Strategies for Integrating CASE Environments", IEEE Software, Março 1992

Jenings89 J.B.T. Jenings; The HP SoftBench Message Model: Concepts and Conventions used by the HP SoftBench Tools. SoftBench Technical Note Series SESD-89-21 Revision 1.2, Hewllet-Packard, Software Engineering Systems Division, Setembro 1989

Jindrich90 W.A. Jindrich; "FOIBLE: A framework for visual programming languages", Master's Thesis, University of Illinois at Urbana-Champaign, 1990

Jino85 M. Jino, et Alli; "SIPS - An Extensible and Integrated Environment for Software Development and Production", Centro Tecnológico para Informática, Instituto de Automação, 1985, Campinas, SP

Johnson88 R.F. Johnson et alli; "TS: An optimizing compiler for Smalltalk", Proceedings of OOPSLA'89, SIGPLAN Notes, Vol. 23, No. 11, San Diego, Califórnia, Setembro 1988

Kiesel92 N. Kiesel, A. Schürr, B. Westfechtel, "Design and Evaluation of GRAS, a Graph-Oriented Database System for Engineering Applications", Technical Report No. AIB 92-44, Technical University of Aachen, 1992

Kiesel93 N. Kiesel, A. Schürr, B. Westfechtel; "GRAS, a Graph-Oriented Database System for (Software) engineering Aplications", IEEE CASE'93 - Proceedings Sixth International Workshop on Computer-Aided Software Engineering, Singapore, Julho 19-23, 1993

Kling91 R. Kling; "Cooperation, Coordination and Control in Coputer-Supported Work", Communications of the ACM, volume 34, número 12, pp 83-88, Dezembro 1991

Klint93 P. Klint; "A Meta-Environment for Generating Programming Environments", ACM - Transactions on Software Engineering and Methodology, Vol.2, No. 2, Abril 93

Kobialka93 H. Kobialka and C. Meyke; "Views on an Object Oriented Software Engineering Environment", IEEE CASE'93 - Proceedings Sixth International Workshop on Computer-Aided Software Engineering, Singapore, Julho 19-23, 1993

Koblitz91 L.F. Koblitz; Um Sistema para Gerência de Configuração de Software; Tese de Mestrado, COPPE/UFRJ, abril de 1991

Krasner88 G.E. Krasner, S.T. Pope; "A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80", Journal of Object-Oriented Programming 1, 3 (Agosto/Setembro 1988)

Krueger92 C. W. Krueger; "Software Reuse", ACM Computing Surveys, Vol. 24, No. 2, Junho 1992

LewiOman90 T.G.Lewis and P.W.Oman; "The Challenge of Software Development", IEEE Software, Novembro 1990

Lindland93 O. I. Lindland and J. Krogstie; "Transformation in CASE Tools - A Compiler View", IEEE CASE'93 - Proceedings Sixth International Workshop on Computer-Aided Software Engineering, Singapore, Julho 19-23, 1993

Linton89 M.A. Linton, J.M. Vlissides and P.R. Calder; "Composing user interfaces with InterViews", IEEE Computer, 22, 2 , Fevereiro 1989

Lubars89 M. D. Lubars; "The IDeA Design Environment", 11th International Conference on Software Engineering, Pittsburgh, Pennsylvania, USA, Maio 1989

Lucena87 C. Lucena; *Inteligência Artificial e Engenharia de Software*, Publicações Acadêmico-Científicas, PUC-RJ/IBM Brasil, Jorge Zahar Editor, 1987

McGrath93 F. McGrath; "Checklist for Buyers of ICASE Products", IEEE Software, Novembro 1993

Machado90 J.A.C. Machado; A Medição da Produtividade no Desenvolvimento de Software; Tese de Mestrado, COPPE/UFRJ, maio de 1990

Madany89 P.W. Madany et alli; "A Class Hierarchy for Building Stream-Oriented File Systems", Proceedings of the 1989 European Conference on Object-Oriented Programming, Nottingham, UK, 1989

Maidantchik92 C.L.L Maidantchik; SIM: Um Gerador Semi-Automático de Documentos; Tese de Mestrado, COPPE/UFRJ, março de 1992

Mannucci89 S. Mannucci, B. Mojans et all; "Graspin: A Structured Development Environment for Analysis and Design", IEEE Software, Novembro 1989

Massolar93 J.L. Massolar; Modelagem do Processo de Desenvolvimento; Tese de Mestrado, COPPE/UFRJ, dezembro de 1993

Mattoso90 A.L.Q. Mattoso; TABA-OBJ: Um Ambiente para Desenvolvimento de Software com Orientação a Objetos; Tese de Mestrado, COPPE/UFRJ, agosto de 1990

Maurer91 J. Maurer; "Editorial Pointers", Communications of the ACM, volume 34, número 12, Dezembro 1991

McMenamim84 S. M. McMenamin & J. F. Palmer; *Essential Systems Analysis*, Yourdon Press, NY, 1984

Menezes88 I. Menezes; ESTIME: Uma Ferramenta para Estimativa de Custos de Desenvolvimento de Software; Tese de Mestrado, COPPE/UFRJ, setembro de 1988

Mercurio90 V.J. Mercurio, B.F. Meyers et all; "AD/Cycle strategy and architecture", IBM Systems Journal, vol. 29, no. 2, 1990

Merlyn90 V.Merlyn and G. Boone; "The Ins and Outs of AD/Cycle", Datamation, Março 1990

Meyer88 B. Meyer; *Object-Oriented Software Construction*, Prentice Hall International (UK) Ltd, 1988

Meyer90 B. Meyer; "Tools for the New Culture: Lessons from the Design of the Eiffel Libraries", Communications of the ACM, Vol.33,No.9, Setembro 1990

Meyer92 B. Meyer; "Applying "Design by Contract" ", IEEE-COMPUTER, Outubro 1992, pp.40-51

Meyers91 S. Meyers; "Difficulties in Integrating Multiview Development Systems", IEEE Software, January 1991

Mi92 P. Mi and W. Scacchi; "Process Integration in CASE Environments", IEEE Software, Março 1992

Microsoft88 MS-WINDOWS Programmer's Reference, Microsoft Corp., 1988

Miller89 T.C. Miller; "Configuration Management with the NSE", In 11th International Conference on Software Engineering, 1989

Miriyala93 K. Miriyala and R. Tamassia; "An Incremental Approach to Aesthetic Graph Layout", IEEE CASE'93 - Proceedings Sixth International Workshop on Computer-Aided Software Engineering, Singapore, Julho 19-23, 1993

Mitze81 R.M. Mitze; "The Unix System As a Software Engineering Environment", Software Engineering Environments, edited by H. Hünke, North-Holland, 1981

Miyoshi93 T. Miyoshi and M. Azuma; "A Empirical Study of Evaluating Software Development Environment Quality", IEEE Transactions on Software Engineering, vol. 19, no. 5, Maio 1993

Moura92 L.M.V. Moura; Taxonomia de Ambientes de Desenvolvimento de Software; Tese de Mestrado, COPPE/UFRJ, maio de 1992

Monte93 L.C.M. Monte; Um Modelo de Objetos para o Sistema de Objetos GEOTABA, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Tese de Doutorado, Maio de 1993

Myers75 G.J. Myers; "Reliable Software Trough Composite Design", Van Nostrand Reinhold, 1975

Neves94 L.S.B. Neves; "O Processo de Desenvolvimento de Sistemas Baseados em Conhecimento", Projeto de Final de Curso, Informática, Depto. Matemática, UFRJ, fevereiro de 1994

Nogueira88 D.L. Nogueira; Ferramentas Automatizadas para Apoio ao Projeto Estruturado; Tese de Mestrado, COPPE/UFRJ, abril de 1988

Norman91 R.J.Normam et alli; "CASE at the Start of the 1990's", International Conference on Software Engineering, 1991

Norman92 R.J. Norman and M. Chen; "Working Together to Integrate CASE", IEEE Software, Março 1992

Notkin85 D. Notkin; "The GANDALF Project", The Journal of Systems and Software, volume 5, número 2, Maio 1985

$O_2$93 $O_2$ Technology; The $O_2$ User Manual, Version 4.3, Released Julho 1993, Versailles Cedex, França

Oquendo93 F. Oquendo; "SCALE: software Process-centred CASE Environments for System Composition and Composition Reuse", IEEE CASE'93 - Proceedings Sixth International Workshop on Computer-Aided Software Engineering, Singapore, Julho 19-23, 1993

Orth89 A. I. Orth; Ambientes de Desenvolvimento de Software: Um Estudo, Relatórios Técnicos do Programa de Engenharia de Sistemas e Computação, ES-216/89, COPPE/UFRJ, Outubro de 1989

Palay88 A.J. Palay et alli; " The Andrew Toolkit - An Overwiew", USENIX Association winter Conference, Dallas, Texas, 1988

Paseman89 W. Paseman; "The Atherton Software BackPlane - An Architecture for tool integration", Unix rview 89, Abril 1989

Passos91 M.C.F. Passos; Uma Ferramenta para Construção e Avaliação de Gráficos de Estrutura; Tese de Mestrado, COPPE/UFRJ, maio de 1991

Passos93 M.C.S.F. Passos; "Especificador da Meta de Qualidade de Projetos", Exame de Qualificação, COPPE/UFRJ, 1993 (não publicado)

Penedo84 M.H. Penedo, E.D. Stuckle; "Integrated Project Master Database (PMDB) IR&D Final Report", Technical report TRW-84-SS-22, Dezembro 1984

Penedo88 M.H.Penedo and W.E.Riddle; "Guest Editors' Introduction Software Engineering Environement Architectures", IEEE Transactions on Software Engineering , vol. 14, no 6, Junho 1988

Penedo92 M.H. Penedo, A. Karrer, C.Shu; "A Survey of Software Engineering Environment Architectural Approaches", Technical Report Series, TRW, Arcadia-TRW-90-004-R1, Julho 1992

Penedo93a M.H.Penedo and W. Riddle; "Process-sensitive SEE Architecture (PSEEA) Workshop Summary", ACM SIGSOFT SEN, Julho 1993

Penedo93b M.H. Penedo; "Towards understanding Software Engineering Environments", Technical Report IMPSEE-TRW-93-003, Agosto 1993

Penedo93c M.H. Penedo and C. Shu; "SEE software Bus Attributes/Characteristics", version 1, Technical Report IMPSEE-TRW-93-006, Agosto 1993

Penedo93d M.H.Penedo; *Process-based Software Engineering Environments (PSEE)*, VII SBES Tutorial, Sociedade Brasileira de Computação, Outubro 1993

Perry88 D. E. Perry and G. E. Kaiser; "Models of Software Development Environments", IEEE , 1988

Perry89 D.E. Perry; "The Inscape Environment", Proceedings of the 11th International Conference on Software Engineering, Pensylvania, Maio 1989

Perry92 D.E.Perry and A.L.Wolf; "Foundations for the Study of Software Architecture", ACM SIGSOFT, vol 17, no 4, Outubro 1992

Poulin93 J. S. Poulin; "Integrated Support for Software Reuse in Computer-Aided Software Engineering (CASE)", ACM SIGSOFT, vol 18, no 4, Outubro 93

Pressman92 R. S. Pressman; *Software Engineering: A Practioner's Approach*, Third Edition, McGraw-Hill Internacional Editions, 1992

Prieto-Diaz87 R. Prieto-Diaz, P. Freeman; "Classifiyng Software for Reusability", IEEE Software, vol. 4, no. 1, 1987

Prieto-Diaz91 R. Prieto-Diaz; "Making Software Reuse Work: An Implementation Model", ACM SIGSOFT, vol.16, no. 3, Julho 1991

Prieto-Diaz93a R. Prieto-Diaz; "Status Report: Software Reusability", IEEE Software, Maio 1993

Prieto-Diaz93b R. Prieto-Diaz; "Software Reusability, Classification, and Domain Analysis", VII SBES Mini-Tutorial, Rio de Janeiro, Outubro 1993

Popovich91 S.S. Popovich, W.M. Schell and D.E.Perry; "Experiences with an Environment Generation System", IEEE, 1991

Puncello88 P.P. Puncello, P. Torrigiani et alli; "ASPIS: A Knowledge-Based CASE Environment", IEEE Software, Março 1988v

Redwine89 S.T.Redwine Jr., W.E. Riddle; "Software Reuse Processes", Proceedings of the 4th International Process Workshop, ACM SIGSOFT, vol 14, no 4, Junho 1989

Reiss90 S.P. Reiss; "Connecting Tools using Message Passing in the Field Program Development Environment", IEEE Software, Julho 1990

Ribeiro89 C.A.S. Ribeiro; Um Sistema Especialista para Apoiar o Avaliador de Custos; Tese de Mestrado, COPPE/UFRJ, dezembro de 1989

Rich92 C. Rich and R.C. Waters; "Knowledge Intensive Software Engineering Tools", IEEE Transactions on Knowledge and Data Engineering, Vol. 4 , No. 5, Outubro 1992

Rine92 D. C. Rine and B. Bhargava; "Object-Oriented Computing", IEEE-COMPUTER, Outubro 1992, pp 6-10

Robson91 A. D. Robson, "O Desenvolvimento do Módulo de Impressão do Dicionário de Dados do FEGRES", Trabalho de Final de Curso, Escola de Engenharia, Departamento de Eletrônica, UFRJ, Rio de Janeiro , RJ , março /1991

Rocha87a A. R. C. da Rocha; *Análise e Projeto Estruturado de Sistemas* , Editora Campus , 1987

Rocha87b A.R.C. da Rocha, T.C. Aguiar e J.R.S. Blaschek; Ambientes para Desenvolvimento de Software: Definição de Termos; Relatórios Técnicos do Programa de Engenharia de Sistemas e Computação - ES 137/87, COPPE/UFRJ, 1987

Rocha89 A.R.C. da Rocha, S. Palermo; "Software Quality Assurance in HEP"; Computer Physics Communications 57(1989), North-Holland

Rocha90 A.R.C. da Rocha; T.C. Aguiar; J.M. Souza; "TABA: a heuristic workstation for software development"; COMPEURO'90; Tel Aviv, Israel, maio 1990

Rocha91a A.R.C. da Rocha, S. Palermo; "An environment for software quality evaluation in HEP", Computing in High Energy Physics, Tsukuba, Japão, março 1991

Rocha91b A.R.C. da Rocha, J.M. Souza; O Projeto TABA, Relatórios Técnicos do Projeto TABA, TABA RJ-7/91, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, 1991

Rocha92 A.R.C. da Rocha et alli; "An environment for software quality evaluation", the 9th International Conference of the Israel Society for Quality Assurance, Jerusalem, Israel, novembro 1992

Rocha93 A.R.C. da Rocha et alli; "Ambiente de Desenvolvimento de Software para o Sistema de Planejamento Integrado", Convênio EMBRATEL/COPPE, 1993 (não publicado)

Rombach91 H.D. Rombach; "Software Reuse: a key to the maintenance problem", Information and Software Technology, vol. 33, no. 1, January/Fevereiro 1991

Ross77 D.T. Ross, S.J. Kenneth; "Structured Analysis for Requirements Definition" , IEEE Transactions on Software Engineering, vol. SE-3, No.1, January 1977

Ross85 D.T. Ross; "Aplications and Extensions of SADT", IEEE-Computer, Abril 1985

Rothkind75 M. Rothkind; "The Source Code Control System", IEEE Transactions on Software Engineering, vol. 1, no.14, Dezembro, 1975

Russo89 V. Russo and R.H. Campbell; "Virtual Memory and Backing Storage Management in Multiprocessor Operating Systems using Class Hierarquical Design", Proceedings of OOPSLA'89 SIGPLAN Notes, Vol. 24, No. 10, New Orleans, Louisiania, Setembro 1989

Sagawa90 J. M. Sagawa; "Repository Manager Technology", IBM Systems Journal, vol. 29, No. 2, 1990

Saito89 N. Saito; "The Software Engineering Environment", in Japanese Perspectives on Software Engineering, Matsumoto,Y.; Ohno,Y.(ed.); Addison-Wesley, 1989

Sardinha93 E. Sardinha; "Gerência de projetos na Estação TABA", Tese de Mestrado, USP-São Carlos, novembro 1993

Schafer88 W. Schafer and H. Weber; "The ESF-profile", Handbook of Computer Aided Software Engineering, Setembro 1988, Van Nostrand, Nova Iorque.

Schmucker86 K. Schmucker; *Object-Oriented Programming for the Macintosh*, Hayden, Hasbrouck Heights, New Jersey, 1986

Setliff93 D.Setliff, E.Kant and T. Cain; "Practical Software Synthesis", IEEE Software, Maio 1993

Shu93 C. Shu; "ESF Systems: Kernel/1 and Kernel/2r. An Addendum to A Survey of software Engineering Environment Architectural Approaches", Technical Report IMPSEE-TRW-93-004, Agosto 1993

Simmons93 D.B. Simmons, N.C. Ellis and T.D. Escamilla; "Manager Associate", IEEE Transactions on Knowledge and Data Engineering, Vol.5 , No. 3, Junho 1993

Smith90 D. Smith, P. Oman; "Software Tools in Context", IEEE Software, Maio 1990

Song93 X. Song, H. Fischer and J. Skeer; "Apply Meta-Models to Developing the Semantics Schema for ObjectMaker", IEEE CASE'93 - Proceedings Sixth International Workshop on Computer-Aided Software Engineering, Singapore, Julho 19-23, 1993

Sorenson88 P.G. Sorenson, J. Tremblay and A. J. McAllister; "The Metaview System for Many Specification Environments", IEEE Software, Março 1988

Souza90 J.M. Souza et alli; "Uma Estação de Trabalho para Desenvolvimento de Software", XVI Latin American Informatics Conference, CLEI'90, Assunção, Paraguai, 1990

Stähl92 M. Stähl, A.R.C. da Rocha; "Umboé: Ambiente de Desenvolvimento de Software Educacional na Estação TABA", congresso IberoAmericano de INformática Educativa; Santo Domingo, República Dominicana, Junho 1992

Stinson89 W. Stinson; "Views of Software Development Environments: Automation of Engineering and Engineering of Automation", ACM SIGSOFT, vol 14, no 6, Outubro 1989

Stonebraker92 M. Stonebraker; "The Integration of Rule Systems and Database Systems" , IEEE Transactions on Knowledge and Data Engineering, Vol. 4, No. 5, Outubro 1992

Strelich88 T. Strelich; "The Software Life Cycle Support Environment (SLCSE): A Computer Based Framework for Developing Software Systems", Communications of the ACM, vol. 11, No 35, Novembro 1988

Takahashi90 T. Takahashi e H.K.E. Liesenberg; *Programação Orientada a Objetos*, VII Escola de Computação, São Paulo, 1990

Tan93 K.P. Tan, G.H. Ong and P. Wong; "A Heuristics Approach to Automatic Data Flow Diagram Layout", IEEE CASE'93 - Proceedings Sixth International Workshop on Computer-Aided Software Engineering, Singapore, Julho 19-23, 1993

Tatge89 G. Tatche; "HP SoftBench", In FedCASE'89, 1989

Taylor89 R.N. Taylor, F.C. Belz et alli; "Foundations for the Arcadia Environment Architecture", ACM, 1989

Teichroew77 D. Teichroew and E. Hershey; "PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems", IEEE Transactions on Software Engineering, vol. 3, no. 1, January 1977

Thomas89a — I. Thomas; "Tool Integration in the Pact Environment", 11th International Conference on Sftware Engineering, Pittsburgh, 1989

Thomas89b — I. Thomas; "PCTE Interfaces: Supporting Tools in Software-Engineering Environments", IEEE Software, Novembro 1989

Thomas92 — I. Thomas and B.A. Nejmeh; "Definitions of Tool Integration for Environments", IEEE Software Março 1992

Thompson89 — T. Thompson; "The Next Step", Byte, Vol 14, No. 3, Março, 1989

Traina85 — C. T. Júnior et Alli; "Gerenciamento de Configuração de Software Usando o Modelo Entidade-Relacionamento", Centro Tecnológico para Informática, Instituto de Automação, Campinas, SP, 1985

Travassos90 — G.H. Travassos; FEGRES: Ferramentas Gráficas para Engenharia de Software, COPPE/UFRJ, Tese de Mestrado, Programa de Sistemas, Janeiro de 1990

Travassos91 — G. H. Travassos; OCA: Um Modelo para Representação das Informações Manuseadas pelas Ferramentas da Estação TABA, Exame de Qualificação para Doutorado, Coppe/Sistemas, UFRJ, 1991 (não publicado)

Travassos92a — G. H. Travassos, A.R.C. da Rocha; "Geração de Ferramentas na Estação TABA", Taller en Ingenieria de Sistemas; Santiago, Chile; julho 1992

Travassos92b — G. H. Travassos; Estudo Comparativo de Linguagens Orientadas a Objetos: Implementação do Modelo OCA , Exame de Qualificação para Doutorado, COPPE/UFRJ, Programa de Engenharia de Sistemas e Computação, Dezembro 1992 (não publicado)

Travassos93a — G.H. Travassos; "Um Modelo de Integração de Ferramentas CASE para a Estação TABA", II Workshop de Teses de Mestrado e Doutorado, Coppe/UFRJ - Sistemas, PUC-Rio, outubro 1993

Travassos93b G. H. Travassos; Utilização do Método de Coad/Yourdon em Trabalho Cooperativo: Estudo de Caso, Exame de Qualificação de Doutorado, COPPE/SISTEMAS, Dezembro de 1993 (não publicado)

Trotta90 C.N.F. Trotta, G.H. Travassos, J.M. Souza; "Modelagem de Dados de Aplicações não Convencionais", IV Simpósio Brasileiro de Engenharia de Software, Ouro Preto, MG, 1991

Tsichritzis90 D. Tsichritzis and S. Gibbs; "Towards Integrated Software Communities" in Object Management , Gestion d'Objets, Centre Universitarie D'Informatique, Université de Geneve, 1990

Tsukumo85 N.A. Tsukumo et Alli, Um Sistema Expansível para Produção de Software, Centro Tecnológico para Informática, Instituto de Automação, Campinas, SP

Vale88 M.A.O. Vale; PRODOC- Uma Ferramenta para Produzir Documentação de Produtos de Software, Tese de Mestrado, COPPE/UFRJ, março de 1988

Vessey92 I. Vessey et alli; "Evaluation of Vendors Products: CASE Tools as Methodology Companions", Communications of the ACM, vol. 35, no. 4, Abril 1992

Vilanova93 L. O. Vilanova, "Implantação de FEGRES/X no ambiente UNIX-OpenWindows", Trabalho de Final de Curso, Informática, Departamento de Matemática, UFRJ, Rio de Janeiro, RJ, fevereiro/93

Vlissides89 J.M. Vlissides and M.A. Linton; "Unidraw: A framework for building domain-specific graphical editors", Proceedings of the ACM User Interfaces Software and Technologies'89 Conference, Novembro 1989

Wasserman90 A. I. Wasserman; "Tool Integration in Software Engineering Environments", Software Engineering Environments: Proceedings of Int'l Workshop on Environments, F. Long Ed., Springer-Verlag, Berlin, 1990

Weinand89 A. Weinand, E. Gamma and R. Marty; "Design and Implementation of ET++, a seamless object-oriented application framework", Structured Program, vol. 10, No 2, 1989

Werneck90 V.M.B. Werneck; Taxonomia de Domínios de Aplicação; Tese de Mestrado, COPPE/UFRJ, novembro de 1990

Werneck94 V. M. B. Werneck, "Uma proposta para o TABA-BC", Exame de Qualificação, COPPE/UFRJ, 1994 (não publicado)

Werner92 C.M.L. Werner, "Reutilização de Software no Desenvolvimento de Software Científico", Tese de Doutorado, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, março de 1992

Willumsen93 G. Willumsen, O.I. Lindland et alli; "An Integrated Environment for Validating Conceptual Models", IEEE CASE'93 - Proceedings Sixth International Workshop on Computer-Aided Software Engineering, Singapore, Julho 19-23, 1993

Wirfs-Brock90 R.J. Wirfs-Brock, R.E. Johnson; "Surveying Current Research in Object-Oriented Design", Communications of the ACM, Vol. 33, No. 9, Setembro 1990

Xexeo94 G.B. Xexeo; "Um Sistema de Reutilização de Âmbito Global", Tese de Doutorado, COPPE/UFRJ, março de 1994

Yang93 Y.Yang, J.Welsh and W. Allison; "Supporting Multiple Tool Integration Paradigms within a Single Environment", IEEE CASE'93 - Proceedings Sixth International Workshop on Computer-Aided Software Engineering, Singapore, Julho 19-23, 1993

Young88 M. Young, R.N. Taylor and D.B. Troup; "Software Environment Architectures and User Interface Facilities", IEEE Transactions on Software Engineering , Vol. 14, No. 6, Junho 1988

Yourdon79 E. N. Yourdon and L. L. Constantine; *Strutuctured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Prentice-Hall, Englewood Cliffs (N.J.), 1979

Yourdon90 E. N. Yourdon; *Análise Estruturada Moderna*, Ed. Campus, 1990

Zelkowitz93 M. V. Zelkowits; "Use of an Environment Classification Model", 15th International Conference on Software Engineering, Maio 17-21, 1993, Baltimore, Maryland

Zweig90 J. Zweig and R. Johnson; "Conduits: A communication abstraction in C++", USENIX C++ Conference, 1990

# REFERÊNCIAS BIBLIOGRÁFICAS COMPLEMENTARES


Apple87 — Apple Computing Inc.; *Human Interface Guidelines: The Apple Desktop Guidelines*, Addison-Wesley Publishing Company, Inc., 1987

Bennett87 — J.L. Benett; "Collaboration of UIMS Designers and Human Factors Specialists", Computer Graphics, Vol. 21, No. 2, Abril, 1987

Bertino85 — E. Bertino; "Design Issues in Interactive User Interfaces", Interfaces in Computing, Vol.3, pp 37-53, 1985

Betts87 — B.Betts, D. Burlingame, G. Fischer et all; "Goals and Objectives for User Interface Software" , Computer Graphics, Vol. 21, No. 2, Abril, 1987

Catlin89 — T. Catlin, P. Bush, N. Yankelovich; "Internote: Extending a Hypermidia Framework to Support Annotative Collaboration", Hypertext'90 Proceedings, ACM Pittsburg, PA; Novembro 1989

Cox86 — B. J. Cox; *Object-Oriented Programming: An Evolutionary Approach*, Addison-Wesley, reading (Mass.), 1986

Cox90 — B. Cox; "There Is a Silver Bullet", BYTE Magazine, Outubro 1990

Dahl66 — O. Dahl and K. Nygaard; "SIMULA - An Algol-based Simulation Language",Communications of the ACM, vol. 9, nº 9, pp. 671-678, Sept. 1966

Davis92 — A. A. Davis; "Operational Prototyping: A New Development Approach" , IEEE Software, Setembro 1992

Ellis90 — M. Ellis and B. Stroustrup; *The Annotated C++ Reference Manual*, Addison-Wesley, 1990

Ezzell89 — B. Ezzell; *Object-Oriented Programming in Turbo Pascal 5.5*, Addison-Wesley, 1989

Fairley85 R. E. Fairley; *Software Engineering Concepts*, McGraw-Hill Book Company,1985

Fish88 R.S. Fish, R.E. Kraut, M.D.P. Leland, M. Cohen; "QUILT: A Collaborative Tool for Cooperative Writing", ACM SIGOIS Bulletin; Vol. 9. número 2&3, Março 1988

Francik91 E. Francik, S. E. Rudman, D. Cooper, S. Levine; "Putting Innovation to Work: Adoption Strategies for Multimedia Communication Systems", Communications of the ACM, Volume 34, número 12, Dezembro 1991

Greif87 I. Greif; Overview; in *Computer-supported Cooperative Work: A Book of Readings*, edited by Irene Greif, Morgan Kaufman Publishers, Inc. 1987; pp 5-12

Grudin92 J. Grudin; "Consistency, Standards, and Formal Approaches to Interface Development and Evaluation: A Note on wiecha, Bennett, Boies, gould , and Greene", ACM Transactions on Information Systems, Vol. 10, No. 1, January 1992

Halbert D. C. Halbert & P. D. O'Brien; "Using Types and Inheritance in Object-Oriented Languages", Digital Equipment Corporation, OOSystems Group

Ierusalimschy90 R. Ierusalimschy; *O=M: Uma Linguagem Orientada a Objetos para Desenvolvimento Rigoroso de Programas*, Dissertação de Mestrado, Depto. Informática PUC/RJ, Setembro/90

Ishii91 H. Ishii, N. Miyake; "Toward an Open Shared Workspace: Computer and Video Fusion Approach of TeamWorkstation", Communicatoins of the ACM, Volume 34, Número 12, Dezembro 1991

Johansen87 R. Johansen, C. Bullen; "Thinking Ahead: What to Expect from Teleconferencing", in *Computer-Supported Cooperative Work: A Book of Readings*, editado por Irene Greif, Morgan Kaufman Publishers, Inc. 1987

Jordan90 D. Jordan; "Implementations Benefits of C++ Language Mechanisms", Communications of the ACM, Vol.33,No.9, Setembro 1990

Kilian90 M. Kilian; "Trellis: Turning Designs into Programs", Communications of the ACM, Vol. 33, No.9, Setembro 1990

Knister90 M. J. Knister, A. Prakash; "DisEdit: A Distributed Toolkit for Supporting Multiple Group Editors", Proceedings of the Third Conference on Computer Supported Cooperative Work, Los Angeles, California, Outubro 1990

Kyng91 M. Kyng; "Designing for Cooperation: Cooperating in Design", Communications of the ACM, Vol. 34, Número 12, Dezembro 1991

Lewis88 B.T. Lewis; "Shared Books: Collaborative Publication Management for an Office Information System", ACM SIGOIS Bulletin, volume 9, número 2&3, Março 1988

Lippman89 S. B. Lippman; *C++ Primer*, AT&T Bell Laboratories, Addison-Wesley Publishing Company, 1989

MacLennan79 B.J. MacLennan; "Values and Objects in Programming Languages", SIGPLAN Notices, volume 17, Número 12, Dezembro 1982, pages 70-79

Marca91 D. A. Marca; "Augmenting SADT™ To Develop Computer Support for Cooperative Work", IEEE, 1991

Martins92 A. V. Martins; Análise de Domínios e suas relações com a Reutilização de Software, Monografia, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, 1992

McGregor90 J. D. McGregor and T. Korson; "Object-Oriented Design", Communications of the ACM, Volume 33, número 9, Setembro 1990

Neuwirth90 C.M. Neuwirth, D.S. Kaufer, R. Chandhok, J.H. Morris; "Issues in the Design of Computer Support for Co-authoring and Commenting", Proceedings of the Third Conference on Computer Supported Cooperative Work, Los Angeles, California, Outubro 1990

Nierstrasz O.M. Niertrasz; "What is The "OBJECT" in Object-Oriented Programming?", Centre Universitaire d'Informatique, University of Geneva, Switzerland, 1988

Niertrasz88 O.M. Niertrasz; "A Survey of Object-Oriented Concepts", Object-Oriented and Databases, Addison-Wesley, 1988

Page-Jones92 M. Page-Jones; "Comparing Techniques by Means of Encapsulation and Connascence" , Communications of the ACM, Setembro 1992, vol. 35, no. 9

Pascoe86 G. A. Pascoe; "Elements of Object-Oriented Programming", BYTE Publications Inc., Agosto 1986

Perin91 C. Perin; "Electronic Social Fields in Bureaucracies" , Communications of the ACM, volume 34, número 12, Dezembro 1991

Rentsch82 T. Rentsch; "Object Oriented Programming", SIGPLAN Notices, Volume 17, Número 9, Setembro 1982

Robson81 D. Robson; "Object-Oriented Software Systems" , BYTE Publications Inc., Agosto 1981

Rosen92 J. P. Rosen; " What Orientation Should Ada Objects Take ? " , Communications of the ACM, Novembro 1992, vol. 35, no. 11

Sarin87 S. Sarin, I. Greif; "Computer-based Real-Time Conferencing Systems",in *Computer-supported Cooperative Work: A Book of Readings*, edited by Irene Greif, Morgan Kaufman Publishers, Inc. 1987

Sathi87 A. Sathi, T. E. Morton and S. F. Roth; "Callisto: An Intelligent Project Management System", in *Computer-supported Cooperative Work: A Book of Readings*, edited by Irene Greif, Morgan Kaufman Publishers, Inc. 1987

Shneiderman83 B. Schneiderman; "Direct Manipulation: A Step Beyond Programming Languages" , IEEE Computer, Agosto 1983

Stefik88 M. Stefik, G. Foster, D. G. Bobrow, K.h Kahn, S. Lanning and L. Suchman; "Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings" in Computer-Supported Cooperative Work: A Book of Readings, Morgan Kaufmann Publishers, Inc., 1988

Stroustrup86 B. Stroustrup; *The C++ Programming Language*, Addison-Wesley, Menlo-Park(Calif.), 1986

Tang91 J.C. Tang, S.L. Minneman; "Video Draw: A Video Interface for Collaborative Drawing", ACM Transactions on Information Systems, vol. 9 número 2, Abril 1991

Tsichritzis92 D. Tsichritzis, O. Niertrasz and S. Gibbs; "Beyond Objects: Objects", Object Frameworks: Cadres d'Objets, Université de Genève, 1992

Ungar92 D. Ungar, R. B. Smith, Craig Cambers and U. Hölzle; "Object, Message, and Performance: How They Coexist in Self", IEEE-COMPUTER, Outubro 1992, pp. 53-64

Wegner87 P. Wegner; "Object-Oriented Classification", Research Directions in Object Oriented Programming, Shriver and Wegner editors, The MIT Press, 1987

Wegner92 Peter Wegner; "Dimensions of Object-Oriented Modeling", IEEE-COMPUTER, Outubro 1992, pp. 12-20

WhiteWater89 The WhiteWater Group; *ACTOR Language Manual* , The Whitewater Group Inc., 1989

Winograd87 T. Winograd; "A Language/Action Perspective on the Design of Cooperative Work", in *Computer-supported Cooperative Work: A Book of Readings*, edited by Irene Greif, Morgan Kaufman Publishers, Inc. 1987

# Apêndice A

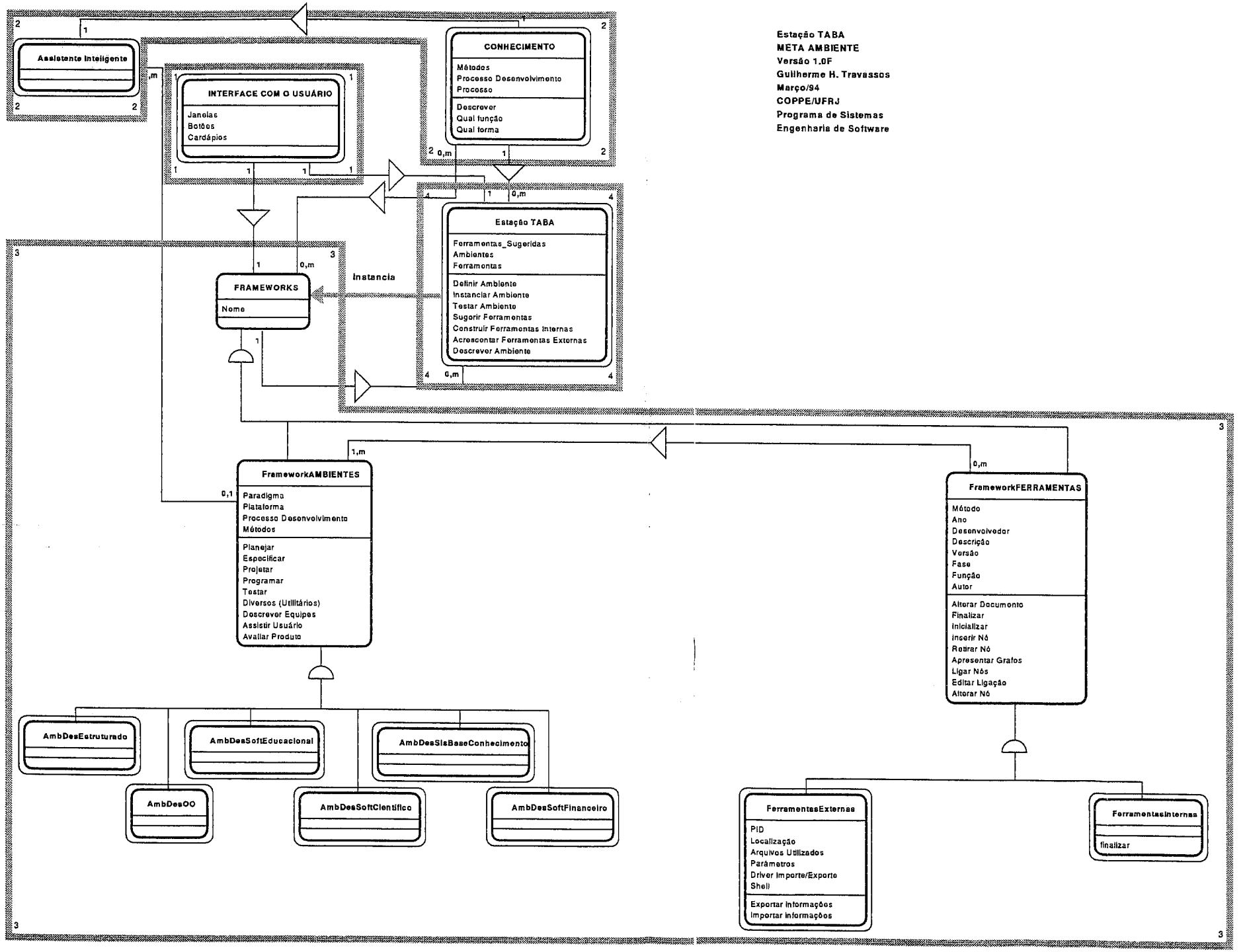
# Organização das Classes Componentes da Estação TABA

## A.1 Diagrama de Classes do Meta-Ambiente

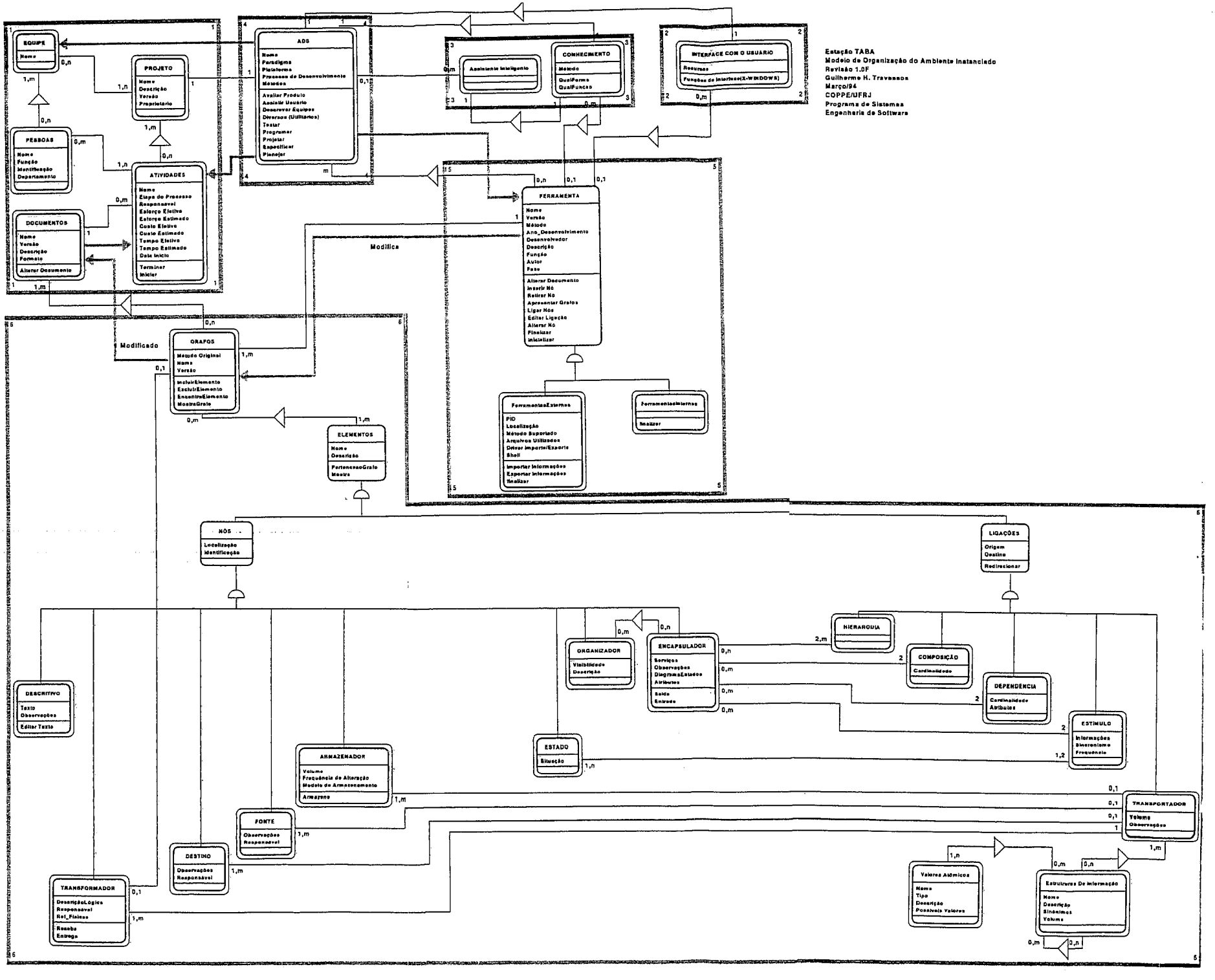
No diagrama da figura A.1 podemos encontrar a organização das classes que compõem o meta-ambiente TABA, cuja especificação está descrita na seção V.5.1.

## A.2 Diagrama de Classes dos Ambientes Instanciados TABA

No diagrama da figura A.2 podemos encontrar a organização das classes que compõem os ambientes instanciados TABA, cuja especificação está descrita na seção V.5.2.




**Estação TABA**
**META AMBIENTE**
**Versão 1.0F**
**Guilherme H. Travassos**
**Março/94**
**COPPE/UFRJ**
**Programa de Sistemas**
**Engenharia de Software**


Estação TABA
Modelo de Organização do Ambiente Instanciado
Revisão 1.0F
Guilherme H. Travassos
Março/94
COPPE/UFRJ
Programa de Sistemas
Engenharia de Software
EQUIPE
Nome
PROJETO
Nome
Descrição
Versão
Proprietario
PESSOAS
Nome
Função
Identificação
Departamento
ATIVIDADES
Nome
Etapa do Processo
Responsavel
Esforço Efetivo
Esforço Estimado
Custo Efetivo
Custo Estimado
Tempo Efetivo
Tempo Estimado
Data Inicio
Terminar
Iniciar
DOCUMENTOS
Nome
Versão
Descrição
Formato
Alterar Documento
ADS
Nome
Paradigma
Plataforma
Processos de Desenvolvimento
Métodos
Avaliar Produto
Assistir Usuário
Descrever Equipes
Diversos (Utilitários)
Testar
Programar
Projetar
Especificar
Planejar
Assistente Inteligente
CONHECIMENTO
Método
QualForma
QualFunção
INTERFACE COM O USUARIO
Recursos
Funções de Interface(X-WINDOWS)
FERRAMENTA
Nome
Versão
Método
Ano_Desenvolvimento
Desenvolvedor
Descrição
Função
Autor
Fase
Alterar Documento
Inserir Nó
Retirar Nó
Apresentar Grafos
Ligar Nós
Editar Ligação
Alterar Nó
Finalizar
Inicializar
Modifica
Modificado
FerramentasExternas
PID
Localização
Método Suportado
Arquivos Utilizados
Driver Importe/Exporte
Shell
Importar Informações
Exportar Informações
Finalizar
FerramentasInternas
finalizar
GRAFOS
Método Original
Nome
Versão
IncluirElemento
EscluirElemento
EncontreElemento
MostraGrafo
ELEMENTOS
Nome
Descrição
PertenecaaGrafo
Mostra
NÓS
Localização
Identificação
LIGAÇÕES
Origem
Destino
Redirecionar
DESCRITIVO
Texto
Observações
Editar Texto
ORGANIZADOR
Visibilidade
Descrição
ENCAPSULADOR
Serviços
Observações
DiagramaEstados
Atributos
Saida
Entrada
HIERARQUIA
COMPOSIÇÃO
Cardinalidade
DEPENDENCIA
Cardinalidade
Atributos
ESTIMULO
Informações
Sinonimo
Frequencia
ESTADO
Situação
ARMAZENADOR
Volume
Frequência de Alteração
Modelo de Armazenamento
Armazena
FONTE
Observações
Responsavel
DESTINO
Observações
Responsavel
TRANSFORMADOR
DescriçãoLógica
Responsavel
Rel_Finais
Recebe
Entrega
TRANSPORTADOR
Volume
Observações
Valores Atômicos
Nome
Tipo
Descrição
Possiveis Valores
Estruturas De Informação
Nome
Descrição
Sinônimos
Volume

# Apêndice B

# Descrição de Métodos utilizando o Modelo de Integração da Estação TABA

## B.1 Introdução

Este anexo contém a descrição de alguns métodos de desenvolvimento de software utilizando a modelo de integração da Estação TABA.

Para uma melhor exemplificação do modelo foram escolhidos, para modelagem em LDMM, quatro métodos de desenvolvimento. Destes métodos, dois seguem o paradigma do desenvolvimento estruturado (SADT e Análise Estruturada) e os outros dois, o paradigma de orientação a objetos (Análise Orientada a Objetos de Coad/Yourdon e o Método de Booch).

Para um melhor entendimento da modelagem, iniciamos por uma breve descrição de cada método.

## B.2 Descrição de Métodos representantes do Paradigma Estruturado

### B.2.1. O método SADT

O método SADT ( Structured Analysis and Design Technique) foi proposto por Ross [Ross77] e tem sido usado na especificação de simples aplicações comerciais até sistemas mais complexos, destacando-se aqueles aplicados a sistemas de defesa e monitoração de processos. Além disto, SADT tem sido utilizado como um gerador de informações para outros tipos de métodos, tais como Jackson, Hipo, PSL/PSA, dentre outros. [Ross85].

O modelo construído em **SADT** se constitui de duas decomposições que representam dois aspectos da aplicação que se está projetando: o que diz respeito às atividades (ou funções) e o que diz respeito aos dados pertinentes a aplicação. Estas decomposições são apresentadas na forma de diagramas e se constituem nos componentes principais do modelo. A abordagem utilizada na construção destes diagramas é "top-down", implicando no aparecimento de decomposições funcionais para um maior detalhamento das informações.

Um modelo **SADT** se compõem de:

- Diagramas funcionais representando o conjunto de atividades do sistema, que mostram as diversas ações que ocorrem para execução da aplicação. São denominados **Diagramas de Atividades ou "Actgram"**;

- Diagramas representando o conjunto de dados ou informações envolvidas na aplicação que se está construindo, que mostram o fluxo das informações dentro da aplicação, apresentando as diversas transformações que a informação sofre dentro da aplicação. São denominados **Diagramas de Dados ou "Datagram"**;

- Um texto que descreve os diagramas e os elementos envolvidos;

- Um glossário de termos específicos da aplicação;

- Um conjunto de informações suplementares, utilizadas para maior clareza da descrição;

- Uma relação hierárquica apresentando a interdependência entre os elementos.

Apesar dos próprios diagramas já apresentarem uma relação hierárquica, entre os elementos, através da descrição gráfica dos relacionamentos, o modelo SADT encoraja o projetista a especificar de maneira precisa a hierarquia existente entre estes elementos.

O primeiro diagrama é chamado Diagrama de Contexto Geral e apresenta o sistema juntamente com suas interfaces para o mundo externo. A partir daí, cada diagrama é um detalhamento das funções existentes e que se está representando para a aplicação. Os diagramas são identificados através do nível em que estão descritos e numerados, a partir de um número de nó, que é construído da seguinte forma:

letra ::= A|D;
número ::=0|1|2|3|4|5|6|7|8|9;
dígito ::= número | dígito número;
identificador ::= letra-"0" | letra<dígito>;

O nível de contexto é identificado por A-0 ,no caso de diagramas de atividades, ou D-0, no caso de diagramas de dados. O próximo nível é identificado por A1, A2, e assim sucessivamente.

O método **SADT** utiliza uma linguagem gráfica para construção dos diagramas que se constitui de caixas e flechas para interligação das caixas. As caixas são desenhadas como um retângulo e possuem, no seu interior, uma identificação do que estão representando. A identificação da caixa varia conforme o tipo de diagrama a que esta pertence. Nos diagramas de atividades, esta identificação assume a forma de um verbo no imperativo, associando a caixa à função, ou atividade, que realiza. Nos diagramas de dados, a identificação se torna um nome qualificador, ou substantivo, que identifica a informação que a mesma representa. Na figura B.1 pode ser encontrada a representação de uma caixa em SADT.

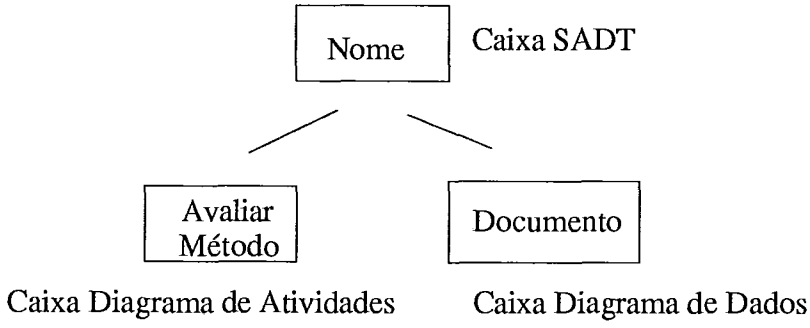



Figura B.1: Caixas em SADT (fonte: [IGL82])

As flechas realizam a conexão entre as caixas e provêm um meio de comunicação e troca de informações. São os canais por onde flui a informação. Cada flecha possui um significado específico e tem um identificador associado, que deve ser apresentado junto a ela, conforme mostra a figura B.2. Flechas podem entrar e sair de uma caixa e, dependendo do tipo de informação que carregam e da posição de contato na caixa, classificam-se em mecanismos, entrada, controle e saída. As flechas entrada, controle e saída são denominadas flechas de interface e podem entrar e sair de uma caixa. A flecha mecanismos representa uma flecha de suporte e só pode entrar na caixa.

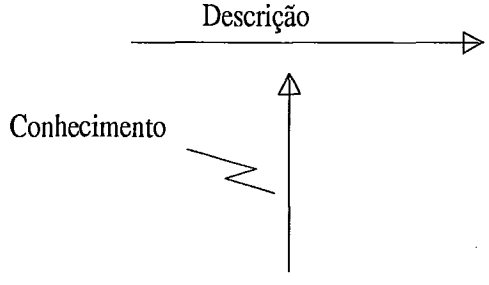



Figura B.2: Representação das Flechas (Fonte: [IGL82])

A representação do significado das flechas depende do diagrama em que as mesmas são apresentadas. Nos diagramas de atividades as flechas se relacionam com os dados e processos, enquanto nos diagramas de dados elas se relacionam com atividades e dispositivos de armazenamento. Isto implica em uma

interpretação diferenciada para cada tipo de diagrama. Esta interpretação pode ser feita da seguinte forma:

### Diagramas de Atividades

- Entradas: dados recebidos pela atividade;
- Saída: dados gerados ou transformados pela atividade;
- Controle: dados cuja utilização influem no processo de transformação de entrada e saída;
- Mecanismo: o processador que realiza, ou desempenha, a atividade (pessoa, um método, etc.).

### Diagramas de Dados

- Entrada: atividade que cria os dados;
- Saída: atividade que utiliza os dados;
- Controle: atividade que faz a transformação do dado;
- Mecanismo: dispositivo de armazenamento dos dados.

Na figura B.3 pode ser encontrada a representação das caixas com as flechas associadas em diagramas de atividades e diagramas de dados.

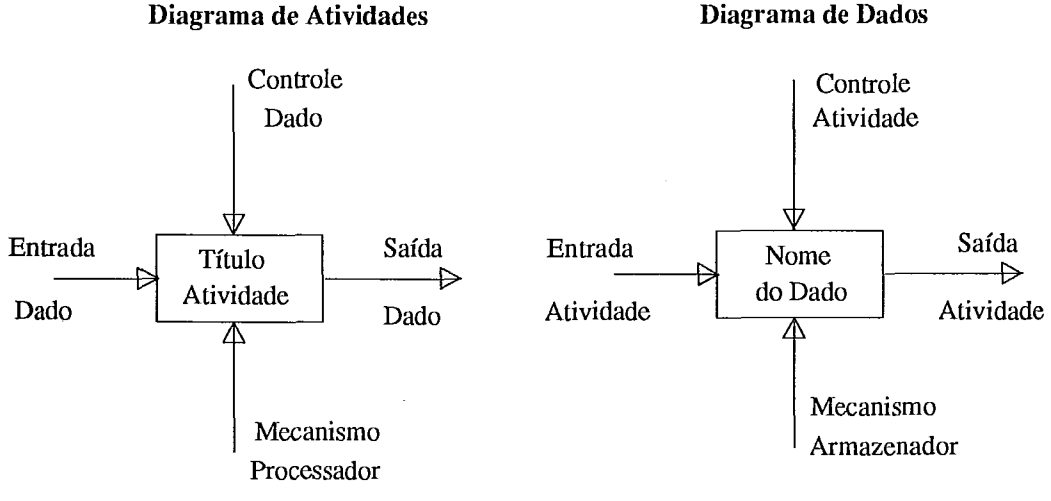



Figura B.3: Caixas e Flechas (Fonte: [IGL82])

Os diagramas SADT, bem como toda a especificação, são construídos em formulários específicos, que permitem um controle apropriado da informação que está sendo gerada. Não existe um rigor exagerado para a construção dos diagramas, mas é proposto um conjunto de regras no sentido de se aumentar a legibilidade e padronizar os documentos. Apenas uma regra tem sua observação fortemente recomendada: os diagramas devem conter de 3 a 6 caixas para que seja obtida uma maior clareza na compreensão do desenho.

As flechas podem se ramificar ou se reagrupar, conforme a necessidade, seguindo as convenções apresentadas na figura B.4. Além disto pode ocorrer a necessidade de representar as flechas em dois sentidos quando ocorrem a entrada recíproca e o controle recíproco. Estas situações são apresentadas na figura B.5.

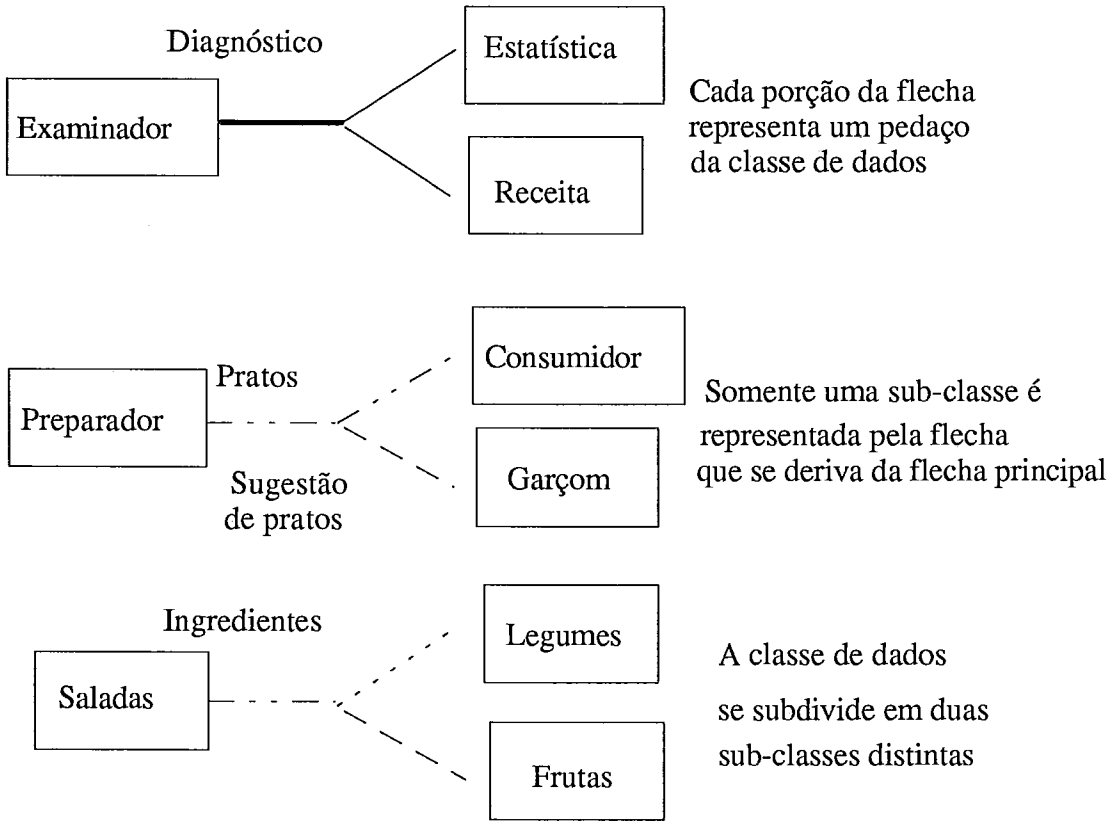



Figura B.4: Ramificação/Reagrupamento de Flechas (Fonte: [IGL82])

Um outro aspecto que deve ser observado é quanto à continuidade dos diagramas. O aparecimento da decomposição funcional implica na necessidade de existir um dispositivo que permita realizar a ligação entre um diagrama e seu detalhamento. Esta ligação é realizada através das flechas que são enviadas ou recebidas para o nível correspondente, associando a estas flechas um identificador que representa o tipo da flecha e seu número. Esta nomenclatura é conhecida como MECS (Mecanismos, Entrada, Controle e Saída) e deve ser feita conforme mostra a figura B.6.

Como pode ser observado, apesar de possuir uma representação relativamente simples, o método se apresenta como uma forma poderosa de construção da especificação de um produto de software. Um outro aspecto importante que podemos destacar é a dualidade dos diagramas de atividades e diagramas de dados. A importância desta característica é a possibilidade de detalhamento tanto da parte funcional, como da relativa aos dados, através do mesmo modelo.

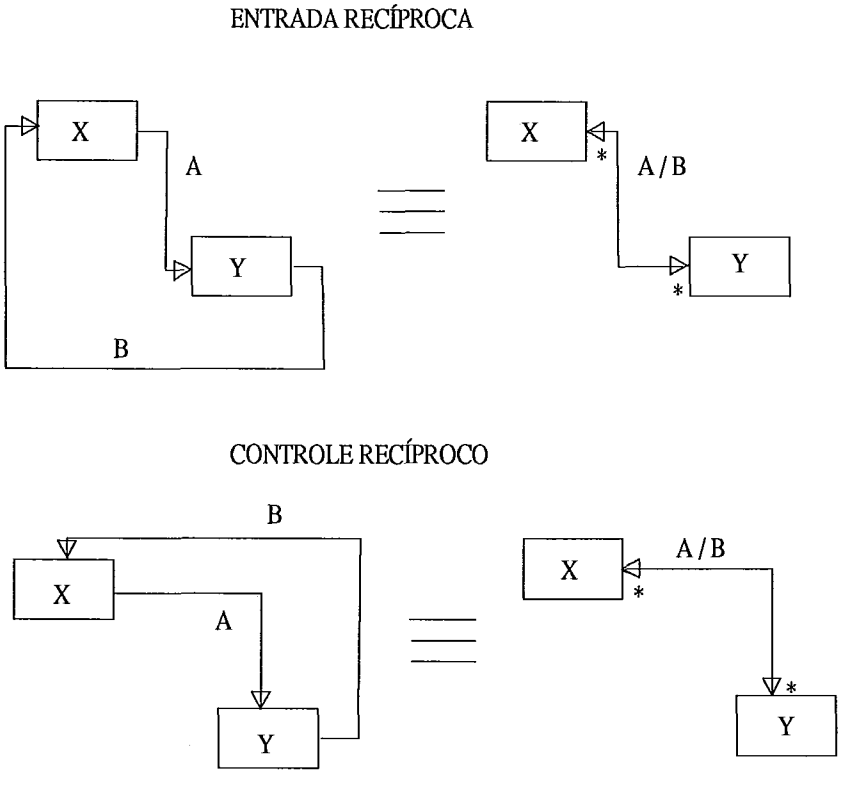



Figura B.5: Reciprocidade das Flechas (Fonte: [IGL82])

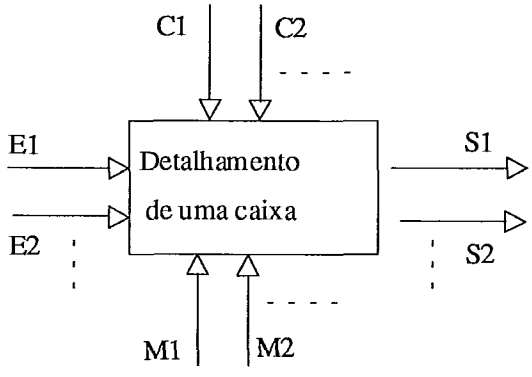



Figura B.6: Nomenclatura MECS (Fonte:[IGL82])

### B.2.2 O método Análise Estruturada

O método Análise Estruturada foi proposto por Gane [Gane82] e tem sido utilizado com grande sucesso na construção da especificação de um produto de software. Compõe-se de quatro ferramentas: Diagrama de Fluxo de Dados, o Dicionário de Dados, a Descrição da Lógica dos Processos e o Diagrama de Acesso Imediato a Dados. Existem variações da Análise Estruturada [Yourdon92] e ambientes [Travassos90a], que incorporam características que permitem representar com mais clareza e recursos o modelo de dados do sistema juntamente com a representação da variação de estados sofridas pelo sistema. Neste trabalho estamos apenas interessados em tratar a Análise Estruturada na sua forma padrão e consideraremos, para efeito de descrição, apenas o Diagrama de Fluxo de Dados.

#### B.2.2.1 Diagramas de Fluxos de Dados

A principal finalidade dos Diagramas de Fluxo de Dados (DFD) é representar a especificação funcional do sistema. Através deles é possível representar as interfaces do sistema com o ambiente externo, sua necessidade de informações e qual o caminho que estas informações devem seguir de forma que possam ser

transformadas e apresentadas, segundo os requisitos e objetivos especificados para o produto.

Um DFD é composto por quatro elementos , representados de forma gráfica, que são a entidade externa, o processo, o depósito de dados e o fluxo de dados.

Entidades externas representam os elementos que interagem com o sistema e não pertencem a ele. Podem fornecer informações para o sistema ou receber informações do sistema. Normalmente representam algum objeto, uma pessoa, um departamento ou até mesmo um outro sistema. São identificadas a partir de um código e um nome. Este código é formado, normalmente, por uma letra e seguida de um número. O nome deve associar a entidade externa com o objeto que a mesma representa no mundo externo.

Processos representam os elementos transformadores da informação. Estes elementos devem desempenhar as atividades essenciais do sistema de forma que o mesmo possa atingir os requisitos e objetivos especificados. Os processos são identificados a partir de dois identificadores principais (um código e um nome) e um secundário (local onde se realiza a atividade). O código é formado por um número sequencial, fornecido de forma a representar a ordem de execução do processo no contexto do sistema. O nome é construído a partir de um verbo no imperativo seguido de substantivos que representem a atividade que o processo desempenha. O local, embora opcional, representa o local onde a atividade é realizada.

Depósitos de Dados representam os elementos responsáveis pelo armazenamento da informação no sistema. São identificados através de um código e um nome. O código é formado pela associação da letra D seguida por um número sequencial. O nome é um substantivo representando o tipo de informação que o depósito é responsável por guardar.

Fluxos de Dados são os elementos responsáveis pelo transporte das informações utilizadas pelo sistemA. Podem ser visualizados como dutos por onde

flui um conjunto de informações. São identificados por sua origem e destino, e possuem um nome descrevendo as informações que transportam.

A figura B.7 mostra a representação dos elementos componentes de um diagrama de fluxo de dados.

Um DFD é construído segundo uma abordagem "Top-Down", onde a medida em que caminhamos ao longo de seus níveis obtêm-se maiores informações a respeito de um determinado processo.

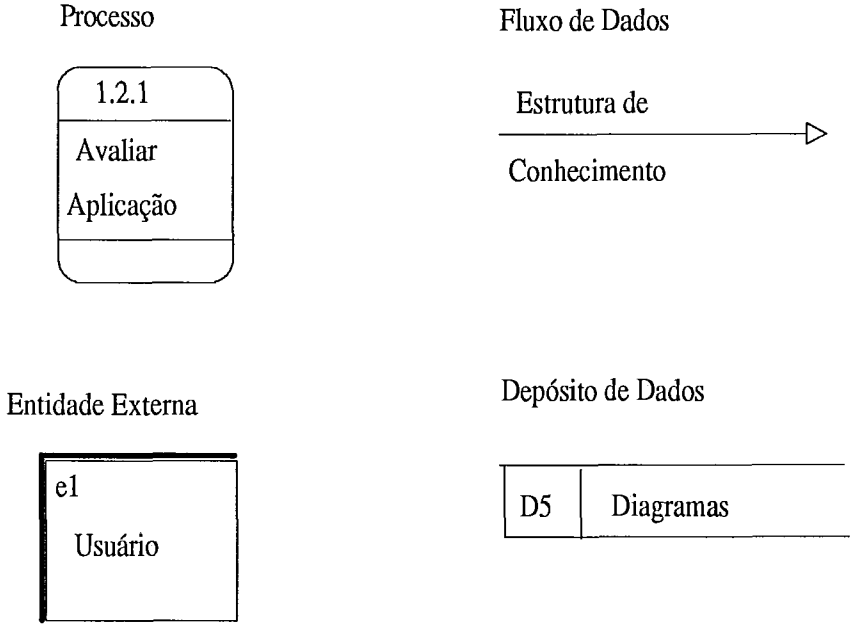



Figura B.7: Elementos de um DFD (Fonte: [Gane82])

O trabalho de construção de um diagrama de fluxo de dados inicia-se através da representação de um processo principal, representando o sistema como um todo, e suas interfaces com o meio externo. Este primeiro nível é denominado de Diagrama de Contexto, ou nível 0, e apresenta as Entidades Externas que se

relacionam com o sistema juntamente com as informações que as mesmas recebem ou enviam para o sistemA.

No nível 1, que é um detalhamento do Diagrama de Contexto, são apresentadas as macro atividades do sistema, mostrando os relacionamentos entre estas atividades e, também, os locais onde as informações ficarão armazenadas durante a vida do sistema, que são representados pelos depósitos de dados.

A cada atividade existente no nível 0, representadas pelos processos, é associado um identificador numérico, que deve indicar a ordem de execução desta atividade de forma a facilitar o entendimento do funcionamento do sistemA.

A partir do nível 1, a decomposição funcional dos processos provoca o aparecimento de novos diagramas derivados dos processos existentes no nível anterior. Este detalhamento dos processos é conseguido a partir da explosão dos processos. Estes novos diagramas são filhos dos processos explodidos e a partir deste ponto definem dois níveis de trabalho: o nível interno e o nível externo para um diagrama. O nível interno implica em toda a região envolvida pelo processo explodido, e identificada pelo envoltório existente e pertencente ao processo. O nível externo é tudo aquilo que não está nesta região.

Algumas regras devem ser observadas para a construção da explosão dos diagramas:

- Entidades Externas não podem aparecer dentro da região do processo explodido;

- Depósitos de dados definidos externamente não podem ser inseridos no nível interno, ao mesmo tempo em que depósitos de dados definidos na explosão são visíveis apenas para os diagramas filhos desta explosão e a própria explosão;

- Fluxos de dados que entram ou saem do processo pai tem de continuar entrando e saindo no diagrama explodido, e sua origem ou seu destino tem de

ser sempre um processo, e pode ocorrer a formação de laços (sair e voltar para um mesmo elemento).

### B.2.3 Correspondências entre SADT e Análise Estruturada

Os métodos de Análise Estruturada e SADT pertencem a uma mesma classe de métodos. Ambos tratam seus diagramas de forma estruturada e preocupam-se com o mesmo tipo de informação. Existem similaridades entre os dois métodos que devem ser consideradas.

Diagramas de Fluxo de Dados são similares a Diagramas de atividades, porém falta aos DFD's a representação de mecanismos e controles, além de utilizarem uma notação adicional (o depósito de dados) para representar o armazenamento de informações. Se considerarmos também os Diagramas de Dados a construção de um DFD a partir de um modelo SADT pode ser feita diretamente. A construção de um modelo SADT a partir de um DFD não pode ser realizada diretamente sem que sejam feitas as definições precisas dos mecanismos e controles envolvidos em cada atividade. Esta restrição torna-se mais forte se o método Análise Estruturada tiver sido usado para especificar uma aplicação que exija recursos de controle mais elaborados. A identificação nos processos que compõem o DFD do local onde as atividades são desempenhadas pode em alguns casos, identificar também os mecanismos para um Diagrama de Atividades.

Realizando uma comparação simplificada podemos destacar as associações apresentadas na tabela B.1.

É difícil dizer qual o modelo mais completo sem definir, precisamente, qual o tipo de problema que se deseja modelar. Conforme pode ser observado na descrição dos métodos, enquanto SADT consegue representar mecanismos de controle que possibilitam o monitoramento da aplicação, Análise Estruturada possibilita a descrição de informações adicionais que permitem detalhar a lógica envolvida nos vários processos existentes. Uma afirmação pode ser feita: é possível

representar as informações geradas pelos dois métodos de forma a possibilitar um mapeamento automático entre eles.

| DFD | Diagrama de Atividades | Diagrama de Dados |
|---|---|---|
| Processo | Atividades | Controle/Entrada/Saída |
| Entidades Externas | Entradas/Saídas sem origem/destino | *** |
| Fluxos | Entradas/Saídas | *** |
| Depósitos | *** | Mecanismos |
| *** | *** | Dados |

(*** - não existe correspondência)

Tabela B.1- Relação entre elementos Análise Estruturada/SADT

### B.2.4 Descrição dos métodos estruturados utilizando o Modelo de Integração TABA

A descrição dos métodos, apresentada nas seções anteriores será tomada como base para a representação do conhecimento no Modelo de Integração TABA. O conhecimento necessário para a representação das regras para validação não será levantado devido a que, nessa primeira versão, o modelo não considerar o aspecto da validação. Embora estas informações se reflitam, diretamente, na modelagem completa dos métodos sua falta não invalida a utilização do modelo, pois a arquitetura de funcionamento do Modelo de Integração TABA possui flexibilidade suficiente para permitir a inclusão destas características no momento adequado.

#### B.2.4.1 O método SADT descrito em LDMM

A descrição de SADT em LDMM implica na construção de modelos para os diagramas de atividades e diagramas de dados Conforme já observado, a visão da

especificação por parte do usuário torna-se completa quando analisados e visualizados os dois diagramas. Estes diagramas enquadram-se perfeitamente numa estrutura de grafos, que permite sua manipulação diretamente.

Os objetos existentes nos diagramas de atividades e diagramas de dados podem ser definidos, conforme o conceito apresentado, com as seguintes funções:

**Diagrama de Atividades**

- Atividade: é um objeto **Transformador**;
- Entrada sem origem: é uma **Fonte**;
- Saída sem Destino: é um **Destino**;
- Entrada/Saída: é um **Transportador**;
- Mecanismo: não possui associação direta;
- Controle: é um **Estímulo**;

**Diagrama de Dados**

- Dados: é um objeto **Descritor**;
- Entrada sem origem: é um **Descritor**;
- Saída sem Destino: é um **Descritor**;
- Entrada/Saída: é um **Transformador**;
- Mecanismo: é um **Armazenador** ;
- Controle: é um **Transformador**.

A descrição pode ser construída como:

```
Def_Método SADT
    Def_Doc Actgram
        Def_Comp  Atividade
            Def_Função Transformador
            Def_Forma Importe DesCaixa
        Fim_Def_Comp
```

```
        Def_Comp  Entrada
                Def_Função Fonte
                Def_Função Transportador
                Def_Forma Importe DesSeta
        Fim_Def_Comp
        Def_Comp  Saída
                Def_Função  Destino
                Def_Função  Transportador
                Def_Forma Importe DesSeta
        Fim_Def_Comp
        Def_Comp Mecanismo
                Def_Forma  Importe DesSeta
        Fim_Def_Comp
        Def_Comp Controle
                Def_Função  Evento
                Def_Forma  Importe DesSeta
        Fim_Def_Comp
Fim_Def_Doc

Def_Doc Datagram
        Def_Comp  Dado
                Def_Função Descritor
                Def_Forma Importe DesCaixa
        Fim_Def_Comp
        Def_Comp  Entrada
                Def_Função Descritor
                Def_Função Transformador
                Def_Forma Importe DesSeta
        Fim_Def_Comp
        Def_Comp  Saída
                Def_Função  Descritor
                Def_Função  Transformador
                Def_Forma Importe DesSeta
        Fim_Def_Comp
```

```
        Def_Comp Mecanismo
            Def_Função Armazenador
            Def_Forma  Importe DesSeta
        Fim_Def_Comp
        Def_Comp Controle
            Def_Função  Transformador
            Def_Forma  Importe DesSeta
        Fim_Def_Comp
    Fim_Def_Doc
Fim_Def_Método
```

### B.2.4.2 O método Análise Estruturada descrito em LDMM

A descrição da Análise Estruturada a partir da LDMM é mais simples do que a descrição anterior feita para SADT. Os objetos existentes na Análise Estruturada são associados às seguintes funções:

- Processo: é um objeto **Transformador**;
- Entidade Externa: é uma **Fonte** ou **Destino**;
- Depósito de Dados: é um objeto **Armazenador**;
- Fluxo de Dados: é um objeto **Transportador**.

A descrição pode ser então construída da seguinte forma:

```
Def_Método SA
    Def_Doc DFD
        Def_Comp  Processo
            Def_Função  Transformador
            Def_Forma  Importe DesRndRect
        Fim_Def_Comp
        Def_Comp  Entidade Externa
            Def_Função  Fonte
```

Def_Função Destino
Def_Forma Importe DesCaixa
Fim_Def_Comp
Def_Comp Depósito de Dados
Def_Função Armazenador
Def_Forma Importe DesDepósito
Fim_Def_Comp
Def_Comp Fluxo de Dados
Def_Função Transportador
Def_Forma Importe DesSeta
Fim_Def_Comp
Fim_Def_Doc
Fim_Def_Método

## B.3 Descrição de Métodos representantes do Paradigma Orientado a Objetos

### B.3.1 O método de Coad/Yourdon

O método de Coad/Yourdon, conforme descrito em [Coad92], realiza a representação do problema a partir das classes e objetos existentes no domínio do problema. Na figura B.8 podem ser encontradas as várias estruturas de representação existentes no método.

Uma classe é representada a partir de um retângulo arredondado que possui três áreas separadas. A primeira contém o nome da classe, a segunda os atributos da classe e a terceira os serviços associados à classe. Uma classe pode ser instanciada ou não. A forma que se tem para representar a instanciação de classes é o contorno externo da representação da classe. Classes instanciáveis definem uma estrutura *Classe-&-Objeto*.

As estruturas Classe-&-Objeto podem ser organizadas de acordo com suas especializações e as relações existentes entre elas. A estrutura utilizada para a representação de uma relação hierárquica entre as classes é a estrutura generalização-especialização (*Gen-Spec*). Esta estrutura representa, no modelo, a apresentação de herança entre classes. Uma variação da estrutura Gen-Spec é a que possibilita entrelaçar estruturas Classe-&-Objeto. O que esta estrutura nos permite é representar uma relação hierárquica de uma Classe-&-Objeto com mais de um superior. É a forma fornecida pelo método para representação de herança múltipla.

Em alguns casos é possível que Objetos sejam compostos por outros objetos sem representação de uma relação hereditária entre eles. Neste caso não é possível a utilização de uma estrutura Gen-Spec. Para isto existe uma estrutura de composição, chamada *Todo-Parte*, que indica a relação estrutural entre objetos.

A dependência entre objetos é representada através de *conexões de instâncias*. Estas conexões de instância demonstram o grau de relacionamento e interdependência entre objetos. Esta representação fortalece o modelo e permite a visualização direta da relação entre os objetos.

A comunicação entre os objetos é representada através de uma estrutura de *conexão de mensagem*. A partir desta estrutura é possível representar quais estímulos serão aplicados aos objetos e por qual objeto este estímulo será disparado. Uma conexão de mensagem existe no sentido de que um objeto fará uma pergunta a outro objeto e aguardará uma resposta.

A apresentação das estruturas Classe-&-Objeto, seus relacionamentos e suas comunicações, se faz através de diagramas contendo estas características separadamente. Pode ser interessante, por questões de clareza e organização, separar as estruturas Classe-&-Objeto por *assunto*. Neste caso os assuntos devem estar devidamente delimitados, de forma a proporcionar um melhor entendimento do modelo.

A especificação das estruturas Classe-&-Objeto se faz a partir de descrições mais detalhadas das informações contidas no diagrama. Esta descrição pode ser realizada utilizando-se o modelo apresentado a seguir:

especificação
- atributo
- atributo
- Entrada Externa
- Saída Externa
- Diagrama De Estado De Objeto
- Especificações Adicionais
- notas
- serviço<nome & Diagrama de Serviço>
- serviço<nome & Diagrama de Serviço>
- e, na medida do necessário,
  - Códigos de acompanhamento
  - Códigos de Estado Aplicáveis
  - Requisitos de tempo
  - Requisitos de memória

A descrição dos estados do objeto pode ser feita através da utilização de diagramas de transição de estados. A forma destes diagramas proposta em [Coad92] pode ser vista na figura B.9.

Os serviços, que representam o comportamento dos objetos, são descritos através de diagramas de serviço, que se apresentam sobre um conjunto de símbolos associados a elementos convencionais de descrição de algoritmos, ou sob a forma de pseudocódigo. Estas formas de representação podem ser vistas na figura B.10.

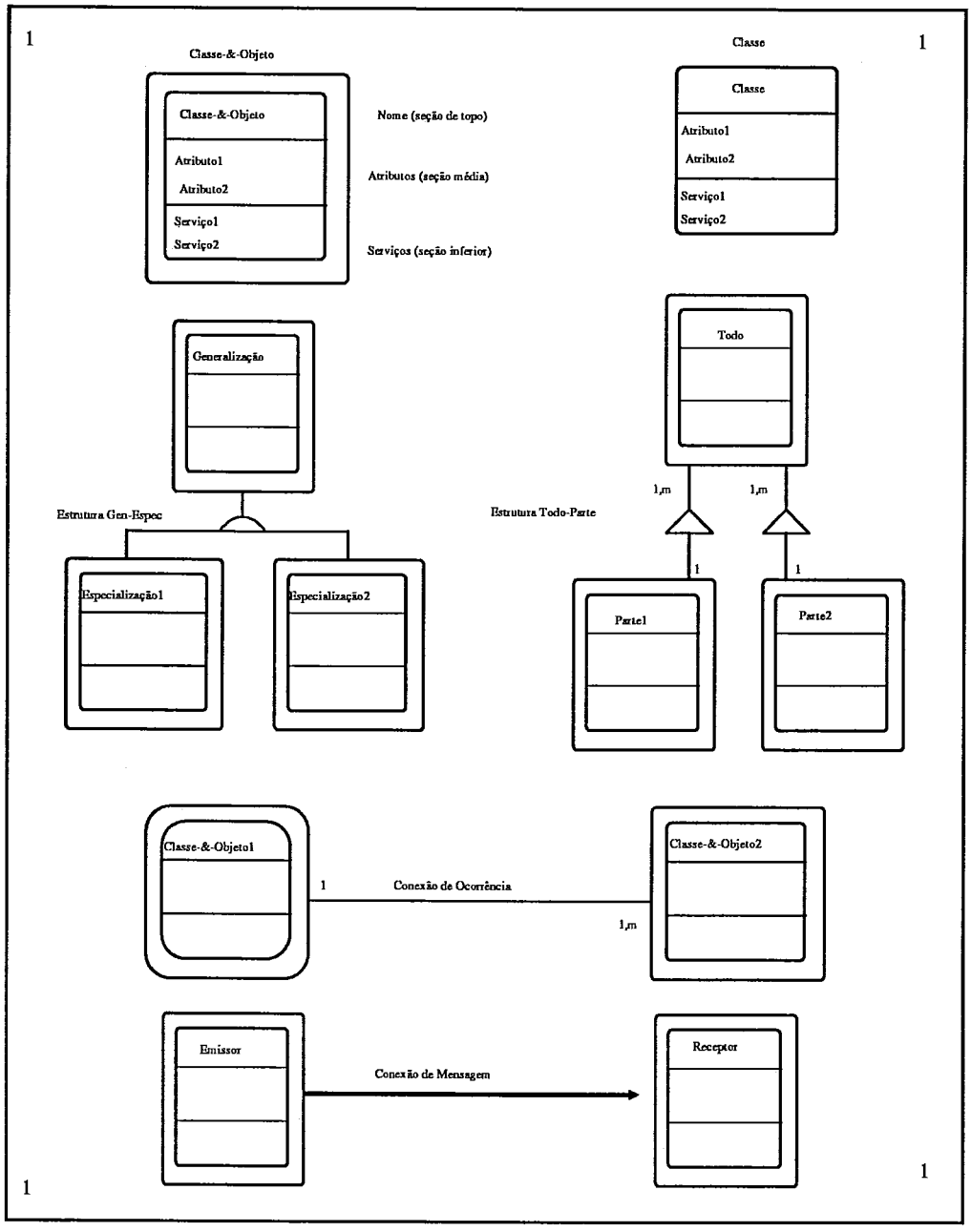



Figura B.8 - Estruturas OOA (Fonte: [Coad92])

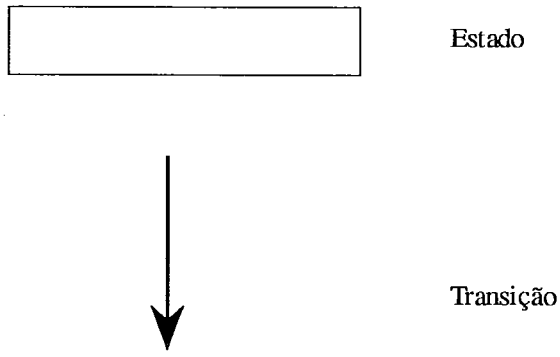



Figura B.9 - Diagramas de Transição de Estados: notação (Fonte:[Coad92])

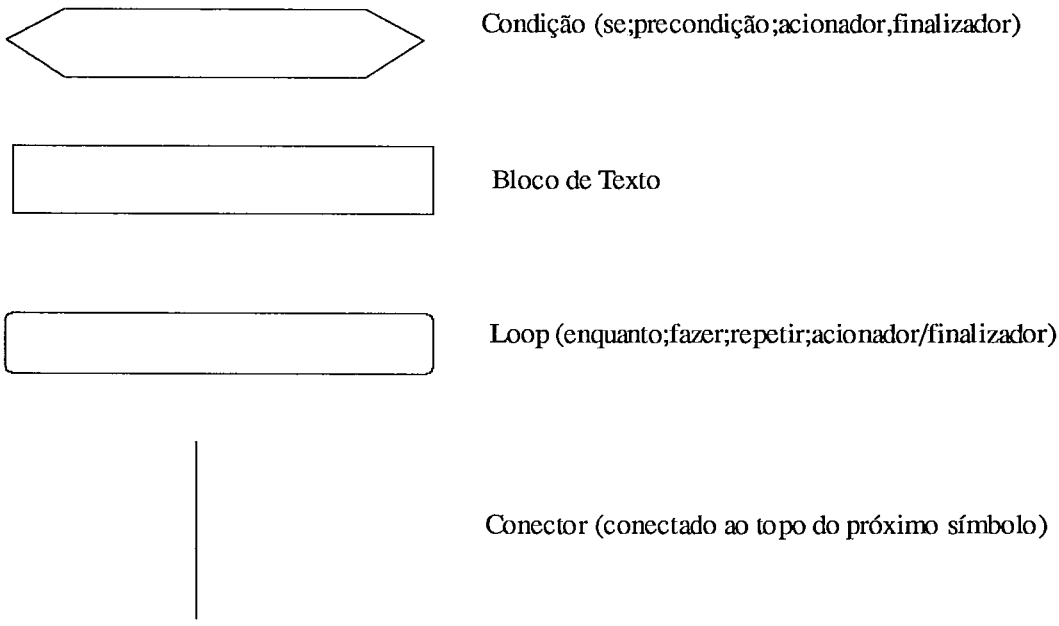



Figura B.10 - Representação dos Serviços: notação (Fonte: [Coad92])

### B.3.2 O Método de Booch

O método de Booch, conforme descrito em [Booch91], [Booch92a] e [Booch92b], realiza a representação do problema a partir da identificação das classes e objetos e sua posterior apresentação em diagramas específicos. A utilização do método leva em consideração que cada projeto deve ser representando de forma balanceada, equacionando o grau de formalidade e informalidade necessários para sua descrição, ao longo do processo de desenvolvimento. Para isto encontram-se disponíveis formas de representação que podem ser aplicadas ao longo de todo o processo e que devem ser, para cada fase do ciclo de vida, selecionadas de maneira a se poder manter o equilíbrio de representação.

O objetivo do método é ser o mais abrangente possível, possuindo representações que atendem a grande parte dos requisitos de modelagem em orientação a objetos. Esta grande disponibilidade de recursos traz, em alguns casos, a impossibilidade de sua utilização direta na linguagem de programação que será usada para implementação do produto. Nestes casos é recomendável que estas características não sejam utilizadas na modelagem.

Para representar o modelo do problema o método de Booch utiliza-se de quatro tipos de diagramas. Estes diagramas, que retratam o modelo do problema sob pontos de vistas diferenciados, devem ser considerados em conjunto para uma perfeita compreensão do que se está modelando. Podem ser construídos os seguintes diagramas :

- Diagrama de Classes: permite a representação de informações de muito alto nível lógico. É utilizado para mostrar a existência das classes e de seus relacionamentos no modelo lógico do sistema.

- Diagrama de Objetos: permitem a representação da estrutura estática dos objetos juntamente com seu comportamento, indicando a organização dos

objetos e suas necessidades de comunicação com outros objetos existentes no modelo.

- Diagrama de Módulos: permite a separação das classes em módulos bem definidos, denominados subsistemas.

- Diagrama de Processos: permite a ilustração da forma de distribuição de processos entre os processadores de um sistema.

As classes no método de Booch são descritas através dos diagramas de classe. Toda classe deve possuir um nome associado e, se possível, a relação e descrição de seus membros[1]. O nome dos membros da classe deve ser único entre todos os membros da classe. Membros, que identificam atributos, são equivalentes a uma associação "possui-por-valor"[2] com outra classe, indicando que atributos podem ser objetos de outras classes.

Classes relacionam-se com outras classes. Estes relacionamentos permitem a representação de relações de hierarquia, composição e dependência entre classes. Estes relacionamentos podem ser "por-valor" ou "por-referência". Relacionamentos "por-valor" são utilizados para representar hierarquias de agregação. Relacionamentos por-referência são utilizados para representar acoplamento indireto entre instâncias da classe. Na figura B.11 encontramos a notação utilizada pelo método de Booch para representação do modelo do sistema.

As classes, e mesmo o sistema como um todo, podem ter associado um diagrama de transição de estados para indicar como os eventos externos podem influenciar no comportamento dos objetos descritos por ela. O aspecto dos símbolos utilizados neste diagrama estão apresentados na figura B.12.

---

[1] Membros da classe: atributos e funções da classe

[2] has-by-value , no original

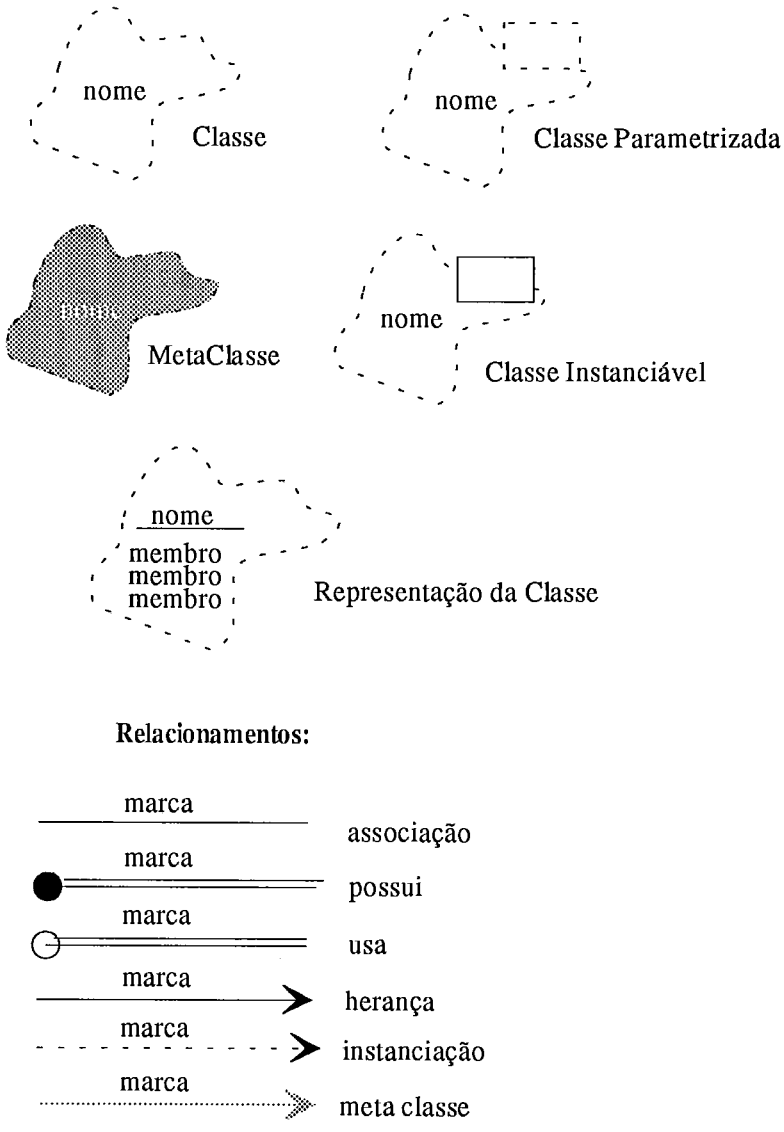



Figura B.11 - Notação de Classes (Fonte: [Booch92a])

Os objetos que aparecem nos diagramas foram criados a partir das classes instanciáveis. Para diferenciar a representação, objetos são representados com o mesmo símbolo da classe, porém com a linha contínua, realçando o aspecto de sua existência. Estes objetos, por sua vez, podem se comunicar através de mensagens. Na figura B.13 pode ser observada a notação para objetos e mensagens, do método.

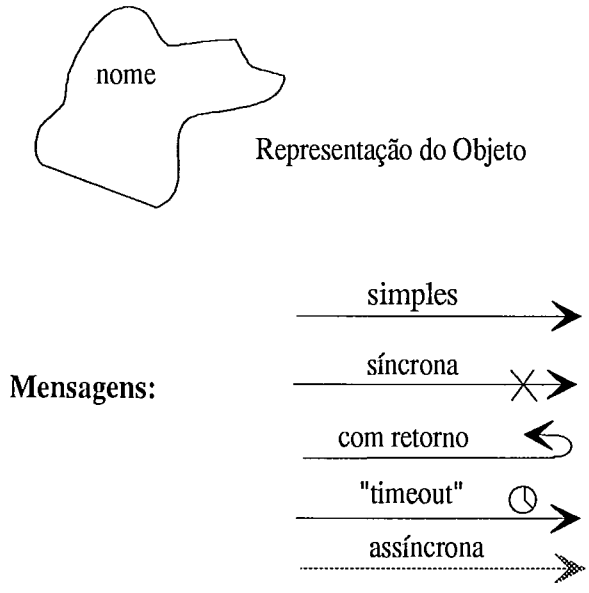



Figura B.12 - Representação do Objeto (Fonte: [Booch92b])

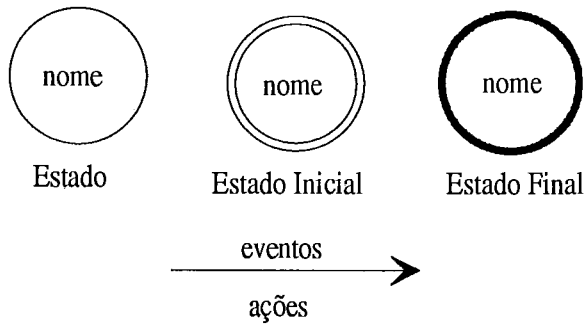



Figura B.13 - Símbolos para o diagrama de transição de estados (Fonte: [Booch92b])

Cada elemento que aparece nos diagramas deve ser descrito através de esquemas pré determinados, que possuem informações mínimas para

documentação do modelo. A documentação dos diagramas é realizada a partir do conjunto de informações apresentados nas tabelas B.2, B.3 e B.4:

| **Especificação de Classe** |
|---|
| **Nome: Identificador** |
| *Responsabilidades*: texto |
| *Qualificação*: texto |
| *Acesso*: exportada ou local |
| *Cardinalidade*: cardinalidade da classe |
| *Parâmetros Genéricos*: lista de parâmetros |
| *Meta Classe*: nome da classe |
| *SuperClasse*: lista de nomes de classes |
| *Membros*: lista de relacionamentos e declaração de operações |
| *Complexidade Espacial*: expressão |
| *Persistência*: transiente ou persistente |
| *Concorrência*: sequencial, protegido ou síncrono |
| *Máquina de Estados Finitos*: diagrama de transição de estados |

Tabela B.2 - Informações para Especificação de Classe (Fonte: [Booch92a])

| **Especificação de Relacionamentos** |
|---|
| **Tipo: Não refinado, uso, é-um, herança, instanciação, meta classe** |
| *Responsabilidades*: texto |
| *Marca*: texto |
| *Fonte*: Nome |
| *Destino*: Nome |
| *Adereços*: adereços |

Tabela B.3 -Especificação de Relacionamentos (Fonte: [Booch92a])

| Especificação de Operações |
|---|
| **Nome: Identificador** |
| *Responsabilidades*: texto |
| *Classificação*: texto |
| *Acesso*: público, protegido, privado ou implementação |
| *Qualificação*: texto |
| *Tipo do resultado*: tipo |
| *Parâmetros formais*: lista de declaração de parâmetros |
| *Exceções*: lista de declaração de exceções |
| *Pré Condições*: texto, código, Linguagem de descrição de processo ou diagrama de objetos |
| *Semântica*: texto, código, Linguagem de descrição de processo ou diagrama de objetos |
| *Pós Condições*: texto, código, Linguagem de descrição de processo ou diagrama de objetos |
| *Complexidade temporal*: expressão |
| *Complexidade Espacial*: expressão |
| *Concorrência*: sequencial, protegido ou síncrono |

Tabela B.4 - Informações para Especificação de Operações (Fonte: [Booch92a])

### B.3.3 Correspondências entre os métodos de Coad/Yourdon e Booch

Os métodos de Coad/Yourdon e Booch possuem características em comum. Por representarem métodos que suportam o mesmo paradigma de desenvolvimento seus elementos constituintes expressam os mesmos conceitos embora existam níveis diferentes de detalhamento das informações entre eles.

O conceito de classe, e dos possíveis objetos existentes a partir destas classes, é realizado de forma mais simplificada no método de Coad, através da estrutura Classe&Objeto. Esta estrutura diferencia se a classe é instanciável ou não. No método de Booch as classes são representadas de forma mais abrangente, mostrando a existência de classes instanciáveis, parametrizáveis, abstratas e possíveis meta classes. É permitido, em ambos os métodos, realizar o relacionamento entre as classes, que serão, inevitavelmente, relações de hierarquia, composição e dependência.

A dinâmica do mundo de objetos pode ser modelada com os dois métodos. Enquanto no de Coad o dinamismo é representado no próprio diagrama de classes (a partir da representação da comunicação entre os objetos existentes no modelo) no método de Booch existe um documento específico, denominado diagrama de objetos, que permite este tipo de representação. Esta representação da comunicação se faz de maneira mais simples no método de Coad. As mensagens, neste método, representam a necessidade de comunicação entre dois objetos, não possuindo mecanismos que permitam a especificação das dependências temporais de sua ocorrência. No método de Booch, ao contrário do que ocorre no método de Coad, mensagens podem ser representadas levando-se em consideração em que tempo elas devem ocorrer. A diferenciação do tipo da mensagem é feita por sua representação gráfica.

Diagramas de transição de estados existem nos dois métodos, embora a representação gráfica seja diferente. A correspondência entre os elementos existentes nos métodos de Coad e de Booch pode ser vista na tabela B.5.

| Método de Coad/Yourdon | Método de Booch |
|---|---|
| Classe&Objeto (não instanciável) | Classe |
| Classe&Objeto (não instanciável) | Classe Parametrizada |
| Classe&Objeto (não instanciável) | Meta-Classe |
| Classe&Objeto | Classe Instanciável |
| Conexão de ocorrência | Relacionamento associação |
| Estrutura Todo-Parte | Relacionamento possui |
| Conexão de ocorrência | Relacionamento usa |
| Estrutura Gen-Spec | Relacionamento herança |
| Mensagem | Relacionamento instanciação |
| Estrutura Gen-Spec | Relacionamento meta-classe |
| Classe&Objeto | Objeto |
| Mensagem | Mensagem simples |
| Mensagem | Mensagem síncrona |
| Mensagem | Mensagem com retorno |
| Mensagem | Mensagem com "timeout" |
| Mensagem | Mensagem assíncrona |
| Estado | Estado inicial |
| Estado | Estado |
| Estado | Estado final |
| Evento | Evento |
| Assunto | Diagrama de Categoria de Classes |

Tabela B.5 - Correspondência entre os Métodos de Coad/Yourdon e de Booch

### B.3.4 O método de Coad/Yourdon descrito em LDMM

Para realizar a descrição do método de Coad/Yourdon em LDMM é necessária a identificação de seus elementos em conjunto com seus papéis, ou funções, na representação. A modificação do paradigma leva à necessidade de

existirem conceitos diferentes daqueles utilizados para a representação dos métodos estruturados. Neste caso encontramos as seguintes representações:

- Classe&Objeto: é um **Encapsulador**
- Gen-Spec: é uma **Hierarquia**
- Conexão de Ocorrência: é uma **Dependência**
- Todo-Parte: é uma **Composição**
- Mensagem: é um **Estímulo**
- Assunto: é um **Organizador**
- Estado: é um **Estado**
- Evento: é um **Estímulo**

A descrição pode ser, então, construída da seguinte forma:

```
Def_Método Coad/Yourdon
    Def_Doc Diagrama
        Def_Comp  Classe&Objeto
            Def_Função  Encapsulador
            Def_Forma  Importe RoundRect
        Fim_Def_Comp
        Def_Comp  Gen-Spec
            Def_Função  Hierarquia
            Def_Forma Importe DesLinhacomMeiaLua
        Fim_Def_Comp
        Def_Comp ConexãodeOcorrência
            Def_Função Dependência
            Def_Forma Importe DesLinha
        Fim_Def_Comp
        Def_Comp Todo-Parte
            Def_Função Composição
            Def_Forma  Importe DesSeta com Triângulo
        Fim_Def_Comp
```

```
            Def_Comp  Mensagem
                  Def_Função Estímulo
                  Def_Forma  Importe DesSeta
            Fim_Def_Comp
            Def_Comp  Assunto
                  Def_Função Organizador
                  Def_Forma  Importe DesLinhaLarga
            Fim_Def_Comp
            Def_Comp  Estado
                  Def_Função Estado
                  Def_Forma  Importe DesCaixa
            Fim_Def_Comp
            Def_Comp  Evento
                  Def_Função Estímulo
                  Def_Forma  Importe DesLinha
            Fim_Def_Comp
      Fim_Def_Doc
Fim_Def_Método
```

### B.3.5 O método de Booch descrito em LDMM

Para realizar a descrição do método de Booch em LDMM é necessária a identificação de seus elementos em conjunto com seus papéis, ou funções, na representação. A modificação do paradigma leva à necessidade de existirem conceitos diferentes daqueles utilizados para a representação dos métodos estruturados. Neste caso encontramos as seguintes representações:

- Classe: é um **Encapsulador**
- Classe Parametrizada: é um **Encapsulador**
- Meta-Classe: é um **Encapsulador**
- Classe Instanciável: é um **Encapsulador**

- Relacionamento associação: é uma **Dependência**
- Relacionamento possui: é uma **Composição**
- Relacionamento usa: é uma **Dependência**
- Relacionamento herança: é uma **Hierarquia**
- Relacionamento instanciação: é um **Estímulo**
- Relacionamento meta-classe: é uma **Hierarquia**
- Objeto: é um **Encapsulador**
- Mensagem simples: é um **Estímulo**
- Mensagem síncrona: é um **Estímulo**
- Mensagem com retorno: é um **Estímulo**
- Mensagem com "timeout": é um **Estímulo**
- Mensagem assíncrona: é um **Estímulo**
- Estado inicial: é um **Estado**
- Estado: é um **Estado**
- Estado final: é um **Estado**
- Evento: é um **Estímulo**
- Categoria de Classes: é um **Organizador**

A descrição pode ser, então, construída da seguinte forma:

```
Def_Método Booch
    Def_Doc Diagrama
        Def_Comp  Classe
            Def_Função  Encapsulador
            Def_Forma  Importe "Nuvem tracejada"
        Fim_Def_Comp
        Def_Comp  Classe_Parametrizada
            Def_Função  Encapsulador
            Def_Forma Importe "Nuvem tracejada" + retângulo
                                    tracejado
        Fim_Def_Comp
        Def_Comp Meta_Classe
            Def_Função Encapsulador
            Def_Forma Importe "Nuvem tracejada pintada
```

```
Fim_Def_Comp
Def_Comp Classe_Instanciável
        Def_Função Encapsulador
        Def_Forma  Importe "Nuvem tracejada" + retângulo cheio
Fim_Def_Comp
Def_Comp  Rel_Associação
        Def_Função Dependência
        Def_Forma  Importe Deslinha
Fim_Def_Comp
Def_Comp  Rel_Possui
        Def_Função Composição
        Def_Forma  Importe "Linha dupla" + círculo cheio
Fim_Def_Comp
Def_Comp  Rel_Usa
        Def_Função Composição
        Def_Forma  Importe "Linha dupla" + círculo vazio
Fim_Def_Comp
Def_Comp  Rel_Herança
        Def_Função Hierarquia
        Def_Forma  Importe Linha com seta
Fim_Def_Comp
Def_Comp  Rel_Instanciação
        Def_Função Estímulo
        Def_Forma  Importe "Linha tracejada" com seta
Fim_Def_Comp
Def_Comp  Rel_Possui
        Def_Função Rel_Meta-Classe
        Def_Forma  Importe Linha com seta colorida
Fim_Def_Comp
Def_Comp  Objeto
        Def_Função Encapsulador
        Def_Forma  Importe "Nuvem linha cheia"
Fim_Def_Comp
Def_Comp  Msg_Simples
```

```
        Def_Função Estímulo
        Def_Forma  Importe Linha com seta
Fim_Def_Comp
Def_Comp  Msg_Síncrona
        Def_Função Estímulo
        Def_Forma  Importe Linha com seta + "x"
Fim_Def_Comp
Def_Comp  Msg_Com_Retorno
        Def_Função Estímulo
        Def_Forma  Importe Linha com seta voltando
Fim_Def_Comp
Def_Comp  Msg_Timeout
        Def_Função Estímulo
        Def_Forma  Importe Linha com seta + relógio
Fim_Def_Comp
Def_Comp  Msg_Assíncrona
        Def_Função Estímulo
        Def_Forma  Importe Linha com seta colorida
Fim_Def_Comp
Def_Comp  Estado_Inicial
        Def_Função Estado
        Def_Forma  Importe Círculo com borda dupla
Fim_Def_Comp
Def_Comp  Estado
        Def_Função Estado
        Def_Forma  Importe Círculo
Fim_Def_Comp
Def_Comp  Estado_Final
        Def_Função Estado
        Def_Forma  Importe Círculo com borda larga
Fim_Def_Comp
Def_Comp  Evento
        Def_Função Estímulo
        Def_Forma  Importe Linha com seta
```

```
            Fim_Def_Comp
            Def_Comp  Categoria_Classes
                  Def_Função Organizador
                  Def_Forma  Importe Linha Larga
            Fim_Def_Comp
      Fim_Def_Doc
Fim_Def_Método
```