# Sumário

# Capítulo 1

# Introdução

Um processo de software pode ser definido como o conjunto de todas as atividades relacionadas ao ciclo de vida do software, incluindo-se determinação e especificação dos requisitos de software, controle de qualidade, entre outras atividades. Este processo pode ser representado por um modelo de processo de software.

Um ambiente de engenharia de software é reponsável pelo suporte ao funcionamento integrado de ferramentas CASE (*Computer Aided Software Engineering*) que deverão trabalhar de forma cooperativa para dar suporte a todo o processo de software. Um tipo particular destas aplicações são os PSEEs, ou ambientes de engenharia de software orientados a processo. Nestes sistemas a ênfase sai da integração de ferramentas CASE e vai para a definição e controle do processo de software. Paralelamente a isto existem os sistemas gerenciadores de *workflow* (WfMS), que gerenciam processos de negócios de maneira geral. Os PSEEs e os WfMSs tem um grande número de componentes em comum, o que encoraja uma tentativa de união destas tecnologias.

As novas tecnologias de desenvolvimento para a *Web*, como CORBA, *Servlets* e JavaScript vem trazendo grande flexibilidade aos sistemas de informação na Internet, assim surge uma nova tendência com relação a PSEEs: Ambientes de Engenharia de Software baseados em *Web*.

Este relatório apresenta os resultados obtidos nos 6 primeiros meses do projeto ¨**Uma Arquitetura Baseada em Web para um Ambiente de Engenharia de Software Orientado a Processos**¨, que tem por objetivo elaborar o projeto de uma arquitetura baseada em *Web* de um ambiente de desenvolvimento de software orientado a processos e construir o protótipo correspondente.

O relatório está estruturado da seguinte forma: o capítulo 2 apresenta os objetivos do projeto, os materiais e conceitos envolvidos estão nos capítulos 3, 4 e 5 que abordam respectivamente Ambientes de Desenvolvimento de Software, Arquitetura de Software e Internet. O capítulo 6 apresenta os principais resultados obtidos e finalmente as conclusões parciais encontram-se no capítulo 8.

No apêndice encomtram-se a descrição do modelo de processo e os artigos referentes a trabalhos apresentados ou submetidos.

# Capítulo 2

# Objetivos

O principal objetivo deste trabalho é estudar e propor mecanismos que viabilizem a construção de ambientes de engenharia de software orientados a processo na *Web*. Outros objetivos específicos, porém não menos importantes, incluem:

- Elaborar a especificação da arquitetura baseada em *Web* de um ambiente de desenvolvimento de software orientado a processos;
- Avaliar padrões de projeto e *frameworks* já existentes que possam ser reutilizados na construção do ambiente.
- Avaliar os recursos necessários, para construção de interfaces, em ambientes baseados em *Web*;
- Avaliar mecanismos de comunicação baseados em Objetos Distribuídos na *Web*;
- Construir o protótipo de um ambiente, baseado em *Web*, com a arquitetura proposta.

# Capítulo 3

# Ambientes de Engenharia de Software

## 3.1 Introdução

Desde o início dos anos 70, quando aconteceu a chamada crise do software [Pre96], tem se desenvolvido cada vez mais a disciplina de engenharia de software. Grande parte deste desenvolvimento está relacionado à utilização de recursos computacionais no auxilio do processo de desenvolvimento de software, surgindo assim as ferramentas CASE (*Computer Aided Software Engineering*).

Um tipo muito especial de ferramenta CASE são os ambientes de engenharia de software, que agrupam outras ferramentas CASE através de algum mecanismo de integração a fim de apoiar todo o processo de desenvolvimento.

## 3.2 Ambientes de Engenharia de Software

Uma ferramenta CASE oferece um conjunto de serviços fortemente relacionados para apoiar uma ou mais atividades do processo de software[1]. Estas ferramentas são de importância crucial no desenvolvimento de produtos de software, porém elas são geralmente construídas com a finalidade principal de apoiar atividades específicas do processo (como por exemplo projeto ou teste) e não o processo como um todo. Logo para o completo suporte à execução de todo o processo de software faz-se necessário um conjunto (geralmente grande) de ferramentas CASE, que de modo geral são construídas sem levar em consideração a integração entre elas.

Na busca por melhorar a integração entre estas ferramentas surgiram os Ambientes de Engenharia de Software (SEE - *Software Engineering Environment*), que nada mais são que um conjunto de ferramentas CASE que se comunicam através de algum mecanismo de integração.

[1] Entenda-se por proceso de software o conjunto de atividades necessárias para se produzir e manter um software de qualidade [Gim94]

A figura 3.1 apresenta um ambiente de engenharia de software genérico e seus principais componentes.

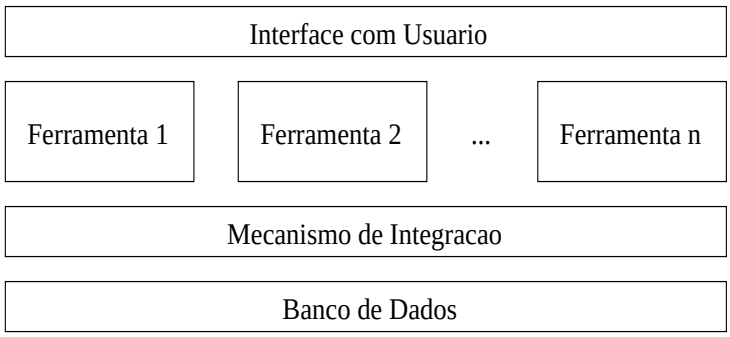


**Figura 3.1**: SEE genérico

Pela figura 3.1 vê-se que além das ferramentas CASE e do mecanismo de integração, um SEE deve conter também uma base de dados e uma interface com o usuário consistente.

Além disso, um SEE deve possuir algumas características básicas [Wer94] [Gim94]:

- ser customizável: o ambiente deve permitir adaptações que o tornem mais adequado a um determinado projeto e as preferências de seus usuários;
- permitir reuso de componentes internos: os componentes que determinam a estrutura interna do ambiente devem poder ser reutilizados em outros processo;
- prover um módulo que seja responsável por controlar o acesso e a evolução de objetos compartilhados;
- fornecer apoio para o controle de versões e configurações;
- permitir a comunicação entre pessoas: o pessoal envolvido no desenvolvimento de software deve ter acesso a mecanismos de comunicação como e-mail e teleconferência;
- poder ser amplamente utilizado: o ambiente deve poder ser utilizado no desenvolvimento de um vasto domínio de aplicações e suportar o desenvolvimento de software em diferentes escalas;
- possuir uma interface gráfica consistente, com apresentação uniforme e com utilização de técnicas e recursos avançados;
- possuir uma interface interna consistente de forma que uma ferramenta possa se comunicar perfeitamente com outra, possibilitando a passagem de informações entre elas, e permitindo também que ferramentas possam ser introduzidas e/ou substituídas ao ambiente;

- ser extensível: o ambiente deve ser aberto, permitindo que novas ferramentas sejam incorporadas na medida que forem requisitadas;
- oferecer suporte a todas as atividades do processo de desenvolvimento: o ambiente deve apoiar o desenvolvimento do produto ao longo de todo o processo de software e não apenas em determinadas etapas;
- suportar múltiplos usuários: o sistema deve ter mecanismos para atender diferentes usuários ao mesmo tempo, requisitando tarefas que podem ser concorrentes, ainda assim devendo manter a consistência do ambiente.

Apesar de todas estas facilidades os ambientes de engenharia de software ainda não são largamente utilizados, principalmente devido a sua complexidade e problemas na construção do sistema de integração.

## 3.3 Ambientes de Engenharia de Software Orientados a Processo

Em um ambiente de engenharia de software orientado a processo (PSEE - *Process-centered Software Engineering Environment*), a preocupação principal sai da integração das ferramentas CASE e vai para a definição, validação, e execução do processo de software.

Estes ambientes visam não só fornecer uma bancada de ferramentas altamente customizável para que engenheiros de software possam trabalhar nas várias fases de um projeto, mas um completo ambiente que apoie a definição e o gerenciamento de processos, se possível, de maneira formal. Assim, além dos componentes apresentados na figura 3.1, este tipo de ambiente deve conter um gerenciador de processos de software, que nada mais é do que um conjunto de mecanismos para apoiar a definição e a execução de processos de software.

Nos PSEEs existem os objetos de software (artefatos produzidos nas várias fases do processo) e os operadores de software (ferramentas utilizadas nas várias fases do processo de software). Os objetos de software podem ser vistos como váriaveis do processo de software e os operadores de software como as ferramentas que transformam estes objetos até se atingir o objetivo do projeto (normalmente um software operacional).

A visualização de objetos e operadores de software em um PSEE nos leva à inclusão de uma série de novos tipos de entidades que devem ser gerenciadas. Dentre estas destacam-se:

- **Artefatos:** São os objetos de software propriamente ditos;

- **Atores:** São uma abstração do usuário do sistema dentro do gerenciador de processos;
- **Feramentas:** Representações das ferramentas CASE como operadores de software;
- **Tarefas:** Representações de atividades do processo de software.

Estes novos elementos, que devem dar suporte ao controle do processo, são em geral manipulados por gerenciadores internos ao gerenciador de processos. Por exemplo, para a manipulação de ferramentas, utiliza-se o gerenciador de ferramentas. Esta abordagem é útil pois diminui a complexidade do gerenciador de processos criando domínios de gerenciamento separados para cada tipo de objeto e operador presente no processo de software [Gim98].

Com esta nova visão os PSEEs passaram a incorporar o conceito de programas de processo, que nada mais são do que descrições detalhadas de processos que visam aplicar maior rigor no controle das atividades envolvidas na produção e manutenção de software de qualidade [Gim94].

Dentre os principais benefícios da utilização de PSEEs destacam-se um maior controle e monitoramento do processo de software, levando-se em consideração sempre questões cruciais sobre o sucesso de projetos como custos e prazos.

## 3.4 Sistemas Gerenciadores de *Workflow*

Segundo a WfMC (*Workflow Management Coalition*) [Wfm95] podemos definir *workflow* como sendo uma coleção de atividades organizadas para realizar um processo de negócios, como por exemplo, o processamento de uma nota fiscal ou a avaliação de um artigo científico. Sendo assim, um *workflow* estabelece a ordem de execução das atividades e as condições em que cada atividade pode ser iniciada, assim como a sincronização de atividades e o fluxo de informações [Tan99].

Os WfMS (*Workflow Management Systems*) são sistemas que permitem a definição, a execução e a monitoração de processos de *workflow*. Estes sistemas utilizam uma ou mais máquinas de *workflow* que estão aptas a coordenar os processos definidos de maneira automática (i.e. executar os *workflows*), invocando outros sistemas de software quando necessário e interagindo com agentes humanos e computacionais.

A figura 3.2 apresenta a arquitetura básica de um WfMS [Wfm95].

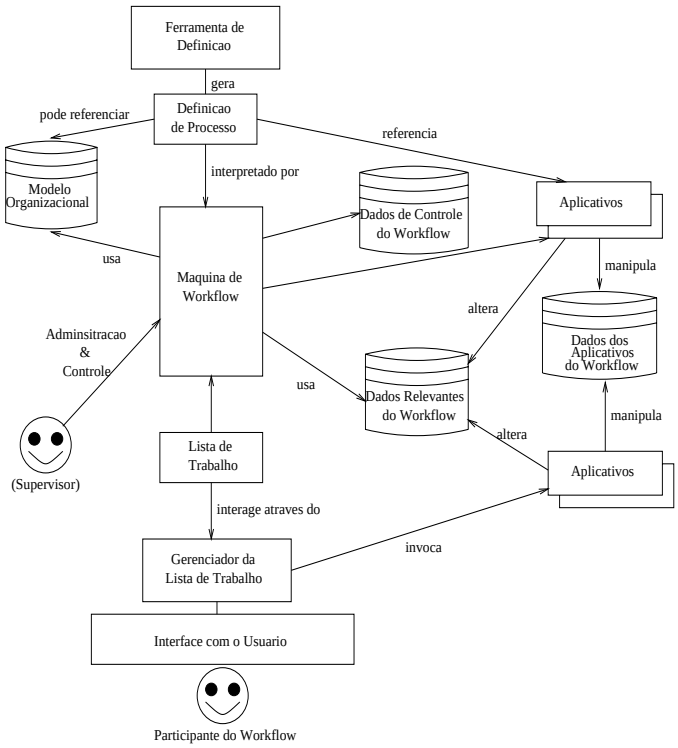


**Figura 3.2:** Arquitetura genérica para um sistema de de *Workflow*

Nesta figura 3.2 podemos identificar vários componentes do WfMS:

- **Ferramenta de Definição:** Esta ferramenta é utilizada para transformar um processo de negócio em uma representação computacional capaz de ser executada pela máquina de *workflow*. Este componente deve dar suporte á definição dos participantes do *workflow* bem como das ferramentas a serem executadas por estes.

- **Definição do Processo:** Modelo computacional do processo de negócio. Esta representação é gerada pela ferramenta de definição de processos e deve ser implementada pela máquina de *workflow*.

- **Modelo Organizacional:** Esta base de dados define a estrutura interna da instituição em que o *workflow* será executado.

- **Máquina de *Workflow*:** É o componente de software responsável pela execução dos processos de *workflow*.

- **Dados de Controle do *Workflow*:** A máquina de *workflow* mantém neste repositório uma série de dados de controle do processo sendo executado.

- **Aplicações:** São as aplicações (ou ferramentas) que a máquina de *workflow* ou o gerenciador de lista de trabalho podem invocar para a realização de alguma atividade dentro do processo corrente.

- **Dados dos Aplicativos do *Workflow:*** Estes dados são produzidos e utilizados pelas aplicações durante a execução dos processo. A máquina de *workflow* não tém acesso a estes dados diretamente, mas sim a uma representação destes. Ex: Diversas aplicações tem acesso a um documento de 20 páginas, entretanto a máquina de *workflow* o enxerga apenas como um documento e seus atributos (autor, data de criação, quem produziu, versão, etc...).

- **Dados relevantes do *Worklfow*:** São os dados que a máquina de *workflow* tem acesso diretamente e analisa para tomar decisões a respeito do processo sendo executado. Um exemplo de dado relevante ao processo seria a representação de um documento.

- **Lista de Trabalho:** Cada participante do *workflow* tem sua própria lista de trabalho e nela estão representadas todas as tarefas designadas a ele, bem como as ferramentas que serão utilizadas nestas tarefas.

- **Gerenciador de Lista de Trabalho:** É o componente do sistema responsável pelo gerenciamento das listas de trabalhos dos participantes do *workflow*.

- **Interface com o Usuário:** Usualmente temos uma agenda onde são apresentadas todas as tarefas que o mesmo deve executar, bem como as suas especificações e seus respectivos estados.

Analisando os componentes dos WfMSs e os componentes dos PSEEs, verificamos várias semelhanças entre os dois tipos de sistemas.

A tabela 3.1 mostra algumas semelhanças entre os dois tipos de ambientes:

| PSEEs | WfMS |
|---|---|
| Gerenciador de Processos | Máquina de *Workflow* |
| Gerenciador de Tarefas | Gerenciador de Lista de Trabalho |
| Repositório de Artefatos | Dados das Aplicações |
| Gerenciador de Ferramentas | Invocação de Aplicativos |
| Ferramentas CASE | Aplicativos Invocados |
| Gerenciador de Artefatos | Dados de controle e dados relevantes ao *Workflow* |

**Tabela 3.1:** Comparação dos PSEEs com os WfMSs

Além das semelhanças apresentadas na tabela 3.1, temos também alguns componentes idênticos nos dois sistemas como por exemplo a agenda, e o fato de ambos os sistemas terem como objetivo apoiar a definição e a execução de processos (processos de software nos PSEEs e processos de negócio[2] nos WfMSs).

[2]Note que o processo de software também pode ser visto como um processo de negócios.

## 3.5 Considerações Finais

Neste capítulo foram apresentados os principais conceitos relacionados a Ambientes de Engenharia de Software, tanto convencionais quanto orientados a processo. Também foram apresentados os conceitos de *workflow* e de WfMS. Assim destacamos as semelhanças entre os WfMSs e os PSEEs através da identificação de componentes equivalentes nos mesmos.

Apesar da característica dinâmica do processo de software impedir sua correta manipulação pela maioria dos WfMSs, os PSEEs tém se mostrado bastante robustos quando utilizados para modelagem de processos de negócios. Desta forma podemos afirmar que os PSEEs são na verdade máquinas de *workflow* estendidas capazes de executar processos de software.

A construção de uma arquitetura para a *Web* de um ambiente de engenharia de software proposta neste trabalho foi embasada nos conceitos apresentados neste capítulo, bem como na idéia de se construir um ambiente baseado na terminologia dos sistemas de *workflow*, a fim de que este possa ser usado para manipulação não só de processos de sofware, mas também de processos de negócio em geral.

# Capítulo 4

# Arquitetura de Software

## 4.1 Introdução

Desde os primórdios do desenvolvimento de software os sistemas são divididos em sub-sistemas relacionados, onde definem-se os módulos (sub-sistemas) e seus relacionamentos, mesmo que informalmente. Esta definição nada mais é do que a **Arquitetura do Software** do sistema.

Segundo Bushmann [Bus96], a arquitetura de software de um sistema consiste na descrição dos sub-sistemas, dos componentes e dos relacionamentos entre eles. Sub-sistemas e componentes são especificados tipicamente sob diferentes visões para mostrar a relevância das propriedades funcionais ou não funcionais de um sistema de software. A arquitetura de software é o resultado da atividade do projeto de software.

Existem várias outras definições possíveis para o termo arquitetura de software [Gam95] [Sha96], entretanto, o ponto em que todas convergem é no fato de que esta área está relacionada à descrição em alto-nível da organização geral dos sistemas e de seus componentes.

Nesta capítulo serão analisados alguns estilos arquiteturais, padrões, e *frameworks* de interesse na construção de PSEEs.

## 4.2 Estilos Arquiteturais

O uso de padrões e estilos arquiteturais já é pratica comum em várias disciplinas de engenharia, estes estilos representam maneiras bem definidas de se lidar com determinados tipos de problemas, e possuem um vocabulário e uma série de regras associadas a eles. O produto de software também tem uma organização e, existem vários estilos já bastante difundidos e utilizados tais como camadas, baseado em eventos, programa principal/subrotinas entre outros. Assim, um **estilo arquitetural** (no contexto de sistemas de software) define uma família de sistemas em termos de organização estrutural, ex-

pressa componentes e os relacionamentos entre eles bem como as regras de composição associadas a estes [Bus96].

A seguir são descritos alguns estilos arquiteturais, para maiores informações a respeito consulte [Sha96].

### 4.2.1 *Pipes and Filters*

Neste estilo arquitetural cada componente tem uma série de entradas e saídas. Um componente lê dados de suas entradas e produz dados em suas saídas. Esta operação normalmente é conseguida aplicando-se algum tipo de transformação nos dados de entrada. Assim os componentes são chamados *filters* e os conectores que transportam dados da saída de um componente para a entrada de outro chamam-se *pipes* [Sha96]. A figura 4.1 apresenta uma arquitetura de *pipes* and *filters*.

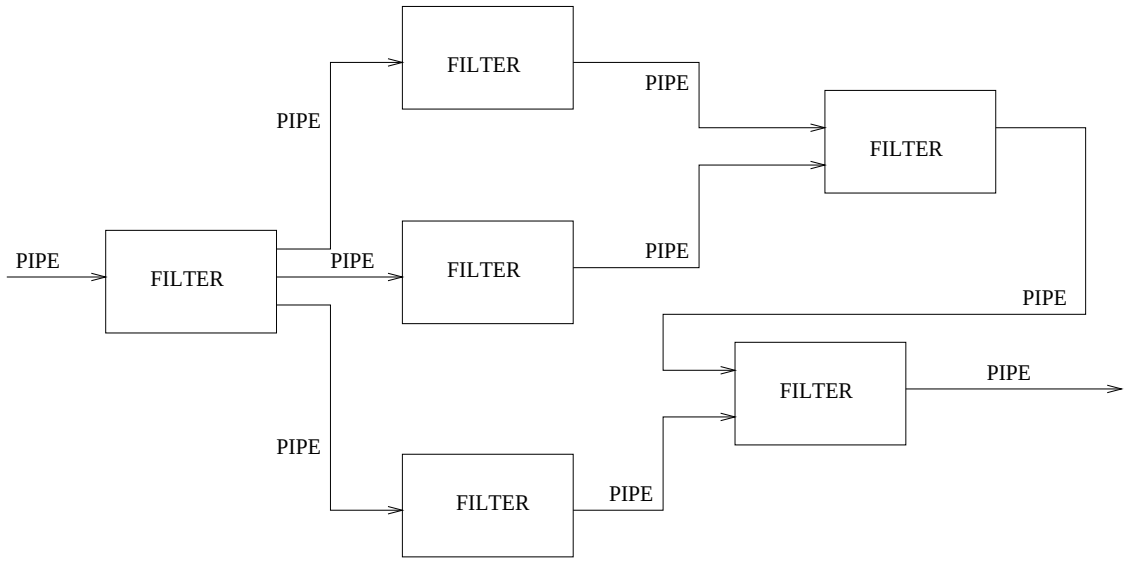


**Figura 4.1:** *Pipes & Filters*

O uso mais frequente deste estilo arquitetural é na organização interna de compiladores, onde cada fase da compilação é modelada como um *filter* e as transferências de dados entre estas fases são modeladas como *pipes*. A figura 4.2 ilustra esta idéia.

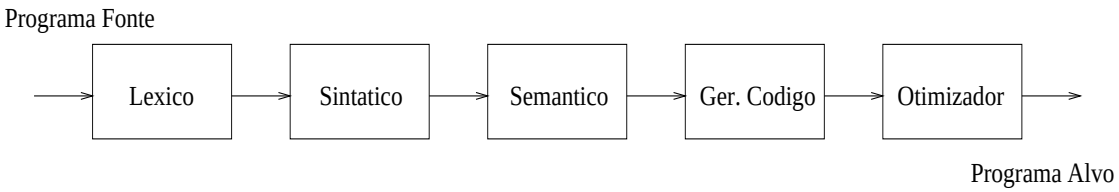


**Figura 4.2:** Arquitetura de um Compilador

### 4.2.2 Orientação a Objetos

Neste estilo arquitetural, a representação de dados e as operações básicas associadas a este dados estão encapsuladas em um objeto. Assim os objetos são entidades que contém um estado (dados) e podem realizar determinadas ações sobre este estado (operações). Os objetos interagem entre si através de invocações de operações e seus parâmetros (também denominados métodos ou mensagens).

### 4.2.3 Baseado em Eventos (Invocação Implícita)

Também conhecido como procedimentos implícitos ou integração reativa, o estilo baseado em eventos baseia-se na idéia de ao invés de invocar um procedimento diretamente, um componente pode anunciar um ou mais eventos e outros componentes no sistema podem registar interesse nestes eventos, associando um procedimento à ocorrência destes. Assim, quando o evento é disparado, o sistema faz a invocação de todos os procedimentos associados a este. Logo o anúncio de um evento causa implicitamente a invocação de vários outros procedimentos [Sha96].

Os componentes neste estilo arquitetural são módulos que apresentam em suas interfaces uma série de operações e eventos. Além das operações poderem ser chamadas de maneira usual, um componente pode associar uma operação a um evento do sistema.

O maior benefício dos sistemas baseados em eventos é o suporte a reuso: um componente pode ser integrado ao sistema apenas registrando interesse (associando um procedimento) em um certo tipo de evento.

Um bom exemplo de sistema baseado em eventos é Java *Swing* [Sun99a], um *framework* para a construção de interfaces altamente customizadas para Java. Neste sistema, cada ação do usuário dá origem a um evento (um *click* com o *mouse*, ou o pressionamento da tecla *Enter*) que é respondido por algum componente do sistema que registrou seu interesse no mesmo.

### 4.2.4 Em Camadas

Em um sistema organizado em camadas, ocorre uma organizacao hierarquica dos diversos níveis de abstração, onde as camadas mais altas utilizam os serviços implementados pelas camadas mais baixas (ou vice-versa). Os componentes do sistema são os níveis ou camadas, enquanto os conectores são os protocolos por onde estas se comunicam.

A utilização mais conhecida deste estilo arquitetural é na implementação de pilhas de protocolo como TCP/IP.

### 4.2.5 Interpretadores

Numa organização de interpretador, uma máquina virtual é construída em software [Sha96]. Assim o interpretador compreende o pseudo-programa a ser interpretado e a máquina de interpretação. A figura 4.3 mostra a arquitetura básica de um interpretador.

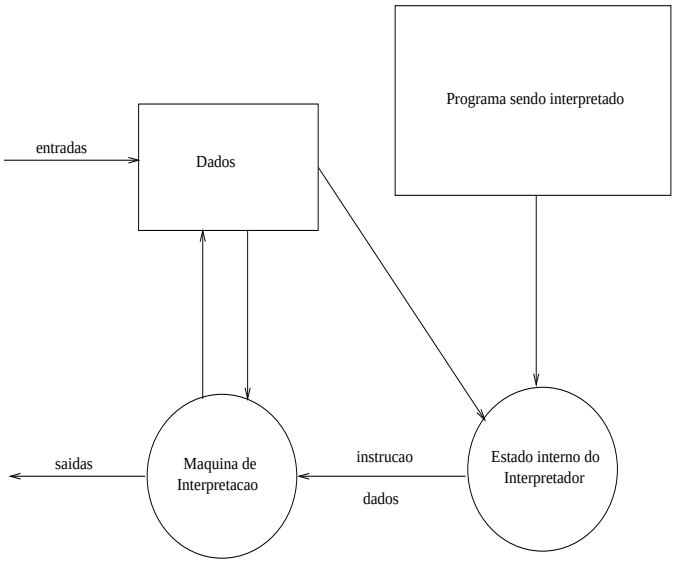


**Figura 4.3:** Arquitetura básica de um Interpretador

Pela figura 4.3 podemos identificar quatro componentes básicos: uma máquina de interpretação que interpreta propriamente o código, a memória que contém o pseudo-código a ser interpretado, a representação do estado interno do interpretador e os dados do programa que está sendo interpretado.

Uma utilização bastante difundida desta arquitetura são os *shells* para a linguagem de *scripts* Tcl/Tk [Ous94].

### 4.2.6 Cliente/Servidor

Neste estilo arquitetural muito utilizado na construção de sistemas distribuídos temos dois tipos de componentes: O Servidor - que provê serviços e o Cliente - que utiliza os serviços oferecidos pelo servidor. Nos sistemas com este estilo arquitetural podemos ter vários clientes conectados á vários servidores.

O PSEE é um exemplo de sistema cliente/servidor.

## 4.3 Linguagens de Descrição de Arquiteturas

As descrições de arquiteturas de software apresentadas até o momento são bastante informais (diagramas de blocos sem grandes regras semânticas), o que dificulta bastante a comunicação e o reaproveitamento de uma arquitetura por outros desenvolvedores.

Outro problema destes diagramas é a carência de uma descrição mais concreta dos componentes e relacionamentos do sistema.

Para suportar um maior formalismo e uma especificação mais direta das arquiteturas de software foram criadas as linguagens de descrição de arquiteturas ou ADLs (*Architectural Description Languages*). Uma ADL contém uma sintaxe e uma semântica formais, de tal modo que a arquitetura a ser descrita seja livre de ambiguidades e tenha um nível de detalhamaneto mais próximo da implementação do sistema de software.

Uma ADL deve possuir algumas características ou propriedades como [Bar97]:

- **Composição**: um sistema deve ser descrito através da composição de seus componentes e conectores;
- **Abstração**: componentes e interações devem identificar suas funções no sistema;
- **Reusabilidade**: componentes e conectores devem ser reutilizáveis;
- **Configuração**: as descrições devem ser capazes de localizar a estrutura de um sistema e permitir a reconfiguração dinâmica;
- **Heterogenidade**: deve ser possível combinar diferentes estilos arquiteturais e também componentes escritos em diferentes linguagens;
- **Análise**: a descrição deve ser capaz de permitir diversos tipos de análise.

Um exemplo de ADL, é a *Wright Model of Architectural Description* [Sha96].

## 4.4 Reusabilidade

Um dos principais objetivos do estudo da engenharia de software atualmente reside na reutilização sistemática de todos os artefatos resultantes do processo de software, como idéias, projetos, componentes e o código em si. Desta forma, todas as etapas do processo de desenvolvimento podem ser beneficiadas com a reusabilidade de artefatos e soluções já utilizadas com sucesso em projetos semelhantes.

A reusabilidade de software proporciona algumas vantagens ao software, a saber:

- **Melhor Qualidade:** A reutilização de componentes e idéias que já foram estensivamente testadas e colocadas à prova em sistemas reais permite a minimização da ocorrência de erros no programa.
- **Minimização do tempo de Desenvolvimento:** Obviamente a reutilização de componentes em todas as fases do processo de software diminui drasticamente o tempo dispendido neste.

Assim, não somente o código pode ser reutilizado mas sim todos os artefatos produzidos durante o processo de desenvolvimento. Desta forma, pesquisas no campo da arquitetura de software, de padrões de software (*Software Patterns*), e de *frameworks* estão sendo realizadas a fim de permitir a reutilização e reciclagem de soluções existentes, na forma de idéias, conceitos, projetos e produtos similares aplicáveis ao processo de desenvolvimento de software, sob diferentes níveis de abstração [Wei98].

## 4.5 Padrões de Software

A aplicação de padrões no desenvolvimento de software tém surgido como uma das mais promissoras abordagens para a melhoria da qualidade de software.

Um padrão de software representa uma solução tanto em nível de projeto detalhado, quanto de arquitetura ou implementação para um problema no contexto do processo de software[1].

Cada padrão descreve um problema que se repete em um ambiente, juntamente com sua solução, de tal modo que se possa reutilizar esta solução sempre que o problema reaparecer [Ale77].

Segundo Tanaka [Tan99], os padrões de software nos ajudam a construir sistemas baseados no conhecimento e experiência coletiva de engenheiros de software mais experientes.

Desde que Gamma [Gam95] aplicou o conceito de padrão no contexto do desenvolvimento de software, diversos cursos, artigos, livros e conferências sobre este assunto têm emergido. Destacam-se entre estes os catálogos de padrões, que contém uma série de padrões de software desenvolvidos e documentados por especialistas que podem ser reutilizados por qualquer engenheiro de software.

Padrões de software normalmente são representados em três partes: Contexto-Problema-Solução.

- **Contexto** descreve o domínio onde o problema ocorre. Ex: *¨Partição do trabalho em sub-tarefas semânticamente iguais¨*[2] .
- O **problema** é a questão que ocorre repetidamente dentro do contexto e que deve ser solucionada pelo padrão.
- A **solução** é a parte do padrão que mostra como resolver o problema. Normalmente a solução é apresentada como uma estrutura de software com a descrição de seus componentes e relacionamentos.

[1] O problema aqui está sempre relacionado a transformar os requisistos do software em um sistema de software operacional.

Existem três categorias básicas de padrões de software [Bus96]: Padrões Arquiteturais, Padrões de Projeto e Padrões Idiomáticos (Idiomas). As próximas sessões abordam estes tipos de padrões.

### 4.5.1 Padrões Arquiteturais (*Architectural Pattern*)

Os padrões arquiteturais provêm um conjunto de sub-sistemas, especificando suas responsabilidades e incluindo regras e informações para organizar os relacionamentos entre eles [Bus96].

Padrões arquiteturais representam os padrões de mais alto nível, eles são úteis na especificação dos componentes básicos do sistema e nos relacionamentos entre eles. Além disso estes padrões nos fornecem uma descrição da estrutura geral do sistema, de tal forma que a toda atividade de desenvolvimento posterior a definição do padrão arquitetural é dirigida por essa estrutura.

Um exemplo de padrão estrutural é o ***Process Manager*** [Wei98], apresentado na sessão 4.7.1.

### 4.5.2 Padrões de Projeto (*Design Pattern*)

Um padrão de projeto descreve uma estrutura de componentes comumente utilizada que resolve um problema de projeto em um contexto [Gam95].

Padrões de projeto possuem uma descrição mais refinada dos sub-sistemas, componentes e seus relacionamentos. Neste nível, os mecanismos de cooperação entre componentes são descritos e definidos para encontrar soluções de questões de projeto, em um dado dominío ou contexto [Tan99]. Os padrões de projeto tratam das estruturas e funcionalidades do sistema, não da implementação destas, o que os torna independentes do domínio.

Um exemplo de padrão de projeto é o ***CLTool*** [Bes99b], apresentado na sessão 4.7.3.

### 4.5.3 Idiomas (*Idioms*)

Idiomas são padrões de software de baixo nível específicos de uma linguagem de programação. Um idioma descreve como implementar um aspecto em especial dos componentes ou dos relacionamentos entre eles com as características de uma dada linguagem [Bus96].

Diferentemente dos padrões de projeto, que resolvem problemas estruturais do software, os idiomas tratam de problemas específicos de implementação em uma determinada linguagem.

[2] *Master-Slave Pattern* [Bus96].

Um exemplo de idioma é o ***Counted Pointer*** [Cop92], que ajuda no gerenciamento de memória através de ponteiros compartilhados em C++.

## 4.6 *Frameworks*

Um *framework* é um projeto genérico em um domínio que pode ser adaptado a aplicações específicas, servindo como molde para a construção de aplicações no domínio especificado [Tan99]. Assim, a grosso modo, um *framework* pode ser visto como uma estrutura de software pré-fabricada onde vários componentes são acoplados para se formar o software operacional.

Estruturalmente falando, um framework é uma arquitetura de software incompleta para um determinado domínio.

Geralmente um *framework* é composto por um conjunto de classes e interfaces, bem como os seus relacionamentos de tal forma que algumas destas classes possam ser instânciadas ou estendidas e algumas destas interfaces possam ser implementadas para formar a aplicação.

Os *framework* não são padrões de projeto, embora partes de um framework possam ser determinados e documentados com a ajuda de padrões [Tan99]. Padrões de software são muito mais abstratos que um *framework*.

Um exemplo de *framework* é o ***VeryHot*** [Yan99], descrito na sessão 4.7.5.

## 4.7 Padrões e *Frameworks* para o Desenvolvimento de PSEEs

A construção de uma arquitetura para PSEE na *Web* não é, de forma alguma, uma tarefa trivial, vários são os componentes e diversos são os mecanismos que os conectam. Somese a isto a complexidade de certos componentes que individualmente requerem uma subarquitetura completa. Felizmente o arquiteto de software tem à sua disposição vários catálogos de componentes, padrões e *frameworks* que podem lhe ser útil na construção de uma arquitetura complexa, que obviamente, involve outras arquiteturas.

A seguir analisaremos alguns padrões e *frameworks* de grande utilidade na construção de uma arquitetura de PSEEs para Web.

### 4.7.1 *Process Manager* [Wei98]

Este padrão arquitetural descreve o conjunto de componentes necessários, bem como suas relações, para se construir um gerenciador de processos de software. Estes componentes se dividem em três grupos básicos:

1. **Modelo de Processo**: são os objetos que descrevem o processo de software;
2. **Meta-Modelo de Processo**: são os objetos que descrevem as arquiteturas dos processos de software;
3. **Gerenciadores**: componentes que gerenciam os objetos do processo de software durante todo o seu cilo de vida.

O padrão segue a descrição proposta por Bushman [Bus96], e têm como principal exemplo de utilização o ambiente ExPSEE [Gim99].

### 4.7.2 *Microkernel* [Bus96]

A arquitetura de *microkernel* tem sido usada com sucesso em diversos sistemas operacionias [Tan95], entretanto sua utilização em outros tipos de sistemas, como por exemplo sistemas de informação, ainda é pequena se comparado às vantagens em termos de flexibilidade e portabilidade que esta arquitetura oferece. A figura 4.4 apresenta a arqutetura geral de um sistema baseado em *microkernel*.

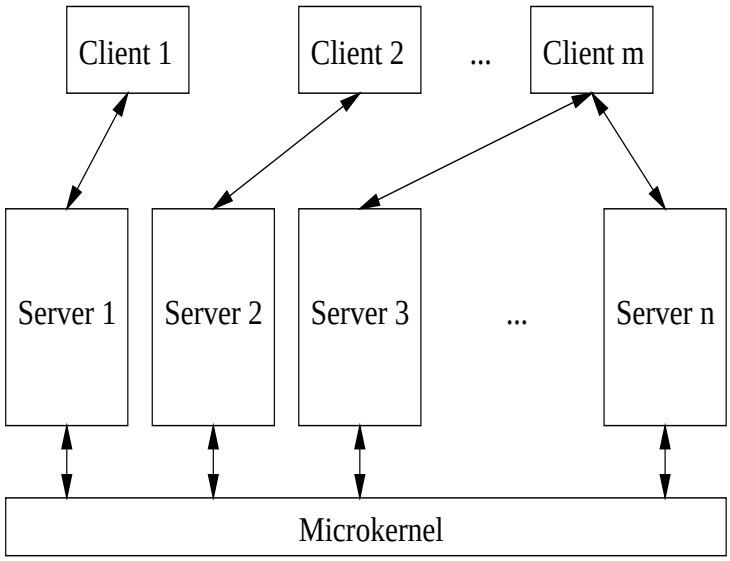


**Figura 4.4**: Arquitetura básica de *Microkernel*

A arquitetura é dividida em 3 camadas principais: *microkernel*, servidores e clientes. O *microkernel* implementa as funcionalidades básicas do sistema, em particular aquelas utilizadas por todos os servidores. Os servidores utilizam-se da infraestrutura oferecida pelo *microkernel* para implementar os serviços necessários ao sistema. Os clientes utilizam-se dos serviços oferecidos pelos servidores para implementar as funcionalidades do sistema em nível de usuário. Além destes componentes básicos, a arquitetura necessita de algum tipo de mecanismo de troca de mensagens para realizar a comunicação cliente/servidor e servidor/*microkernel*.

Um exemplo de sistema que utiliza este padrão é o sistema operacional BeOS [Be00].

### 4.7.3 *CLTool* [Bes99b]

O padrão de projeto **CLTool**, foi desenvolvido com base em experiências com integração de ferramentas a PSEEs. Este padrão descreve um modo de se integrar ferramentas de linha de comando a um software de tal forma que as funcionalidades implementadas nestas ferramentas possam ser aproveitadas pelo software.

O padrão é bastante simples, e contém apenas 1 classe base (*CLTool*) e mais tantas quantas forem necessárias de acordo com o número de ferramentas do sistema ativadas por linha de comando a serem utilizadas (uma sub-classe de *CLTool* para cada ferramenta).

Um exemplo de sistema que pode utilizar este padrão são os sistemas de gerenciamento de rede nos sistemais operacionais Unix e ambientes de engenharia de software que permite ter acesso a ferramentas ativadas por linha de comando (como compiladores) remotamente.

### 4.7.4 *VeryHot* [Yan99]

Este *framework* gráfico é utilizado para construir interfaces complexas, que fazem analogias com o mundo real.

Existem três componentes básicos no framework:

1. **Painel de Desenho**: é a classe onde são desenhados os objetos, ela dá suporte a uma série de eventos e faz o gerenciamento de figuras. Esta classe pode ser instanciada ou estendida;

2. **Figuras**: é um conjunto de classes abstratas que definem os métodos e atributos de todas as figuras que são utilizadas no *framework*. Todo objeto que quiser ser desenhado no painel de desenho deve estender uma destas classes e implementar alguns métodos abstratos.

3. **Ferramentas**: são as classes responsáveis pela criação de figuras e gerenciamento do painel de desenho. Num dado momento, apenas uma ferramenta pode estar ativa no painel de desenho. Para se criar ou manipular determinado tipo de objeto é necessária a selecão da ferramenta correspondente a fim de que esta tome conta do painel e responda pelos eventos gerados pelo usuário. Toda ferramenta criada no ambiente deve estender uma destas ferramentas e implementar os métodos abstratos definidos.

O *VeryHot* têm sido aplicado com sucesso na definição de modelos formais de processos de software [Yan99] e na implementação de sistemas de simulação de robôs [Mel00].

## 4.8 Relacionamento entre Arquitetura de Software, Padrões e *Frameworks*

Existem várias técnicas e tecnologias que nos ajudam a construir uma arquitetura de software melhor e mais robusta em menor tempo. Entretanto é importante entender como elas se relacionam afim de que se possa tirar o máximo proveito destas tecnologias. A figura 4.5 mostra o relacionamento entre padrões, *frameworks*, arquitetura de software, estilo arquitetural, classes e objetos [Sch97].

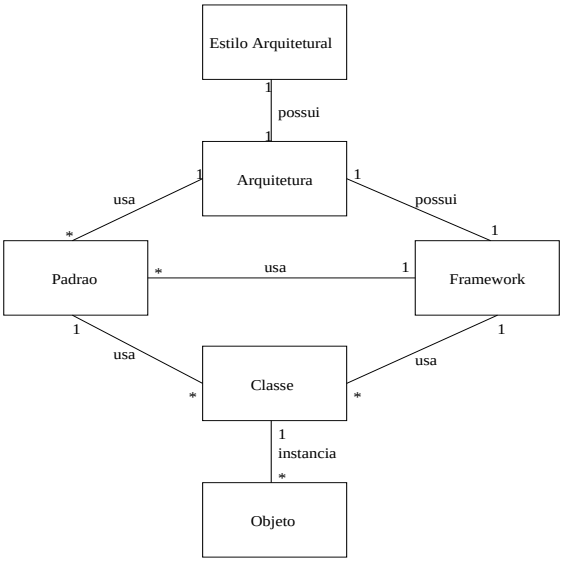


**Figura** 4.5**:** Relacionamento entre Arquitetura de Software, Padrões e Frameworks

Pela figura 4.5 verificamos que toda arquitetura tém um estilo arquitetural, e todo *framework* representa uma arquitetura. Um *framework* pode utilizar muitos padrões em sua estrutura, que por sua vez utiliza muitas classes em sua documentação e implementação.

## 4.9 Considerações Finais

Nesta capítulo foram apresentadas descrições sucintas de alguns estilos arquiteturais, padrões de software e *frameworks* utilizdos na construção da arquitetura proposta nesta trabalho. O capítulo 6 descreve desta arquitetura.

Um ponto primordial quando se fala de arquitetura de software e técnicas de reusabilidade é a documentação. Todo e qualquer tipo de padrão, arquitetura, componente ou *framework* para ser reutilizado, deve conter uma documentação vasta e completa. Só assim, o principal objetivo da reusabilidade pode ser alcançado: minimização do tempo de execução do processo de software.

# Capítulo 5

# Aplicações na Internet

## 5.1 Introdução

Nos últimos anos o mundo vem passando por uma nova revolução, o computador está cada vez mais invadindo a vida das pessoas e uma nova palavra vem mudando conceitos pré-estabelecidos há anos: Internet.

A Internet, que há alguns anos era apenas uma rede acadêmica, atualmente tornou-se uma mídia altamente interativa, que vem sendo utilizada para os mais diversos fins, desde entretenimento até negócios. Atualmente pessoas compram, conversam, fazem amizades, estudam e trabalham via Internet. Ou seja, a Internet está mudando o modo de vida das pessoas. Só para se ter um idéia da revolução que vem acontecendo pequenas empresas da Internet com menos de 5 anos de atividade valem mais do que grandes corporações centenárias.

A origem da Internet está estritamente ligada à ARPANET, uma rede concebida pelo Departamento de Defesa (DoD - *Departament of Defense*) dos Estados Unidos em meados dos anos 60 com o intuito de interligar o país mesmo no caso de uma guerra nuclear[1]. Esta rede interligava centros de pesquisa e universidades que tivessem contratos de pesquisa ligados ao DoD. Nos anos 70 o número de máquinas conectadas à ARPANET cresceu enormemente, principalmente depois que o protocolo TCP/IP tornou-se o único protocolo oficial. Em 1983 ela foi interligada a NSFNET[2] e a partir daí, o crescimento em termos de número de *hosts* tornou-se exponencial [Tan96]. Assim esta nova rede começou a interligar outras redes de várias partes do mundo, tornando-se uma inter-rede.

No final dos anos 80, foi criada uma nova aplicação para a Internet: o WWW (*World Wide Web*). Esta aplicação visava prover um meio simples e intuitivo de compartilhar informações entre os pesquisadores do CERN (*European Organization for Nuclear Research*) que estavam localizados em diferentes países [Ber96]. A arquitetura básica da

[1] Esta era a época aúrea da guerra fria.

[2] Uma outra rede de pesquisadores americanos financiada pela NSF (*National Science Foundation*).

WWW, proposta por Tim Berners-Lee, baseia-se em hipertextos que utilizam a Internet como infra-estrutura de comunicação.

Aproximadamente uma década depois de seu surgimento, a *World Wide Web*, uma WAN (*Wide Area Network*) [Tan96] praticamente universal e com baixos custos de conexão para o usuário final, apresenta-se como catalisador de uma nova revolução nos sistemas de informação [Alc98].

Atualmente a WWW vem sendo utilizada não somente para a apresentação de informações, mas também como um *backbone* para os mais variados sistemas de informação, indo desde vendas *on-line* até cadastros de pesquisadores e projetos.

Neste capítulo serão analisados aspectos técnicos da Internet, enfatizando-se particularmente a WWW e algumas tecnologias para contrução de complexos sistemas de informação, tais como os PSEEs, na *Web*.

## 5.2 Aspectos Técnicos do Funcionamento da Internet

A Internet é uma grande rede de computadores com centenas de milhares de *hosts* conectados a ela. Estes *hosts* obviamente não são semelhantes, máquinas de diferentes arquiteturas operando com sistemas operacionais diversos comunicam-se através da Internet.

O suporte a toda esta interoperabilidade entre arquiteturas depende fundamentalmente de uma pilha de protocolos implementados no software e hardware de rede. A arquitetura da Internet está fundamentada em dois protocolos básicos: o TCP (*Transmission Control Protocol*) e o IP (*Internet Protocol*).

### 5.2.1 A Pilha de Protocolos TCP/IP

O modelo de referência TCP/IP foi uma solução encontrada para conectar as várias redes diferentes que vinham se juntando à ARPANET. O nome do modelo vém de seus dois principais protocolos, o TCP e o IP.

A figura 5.1 mostra a pilha de protocolos TCP/IP [Tan96].

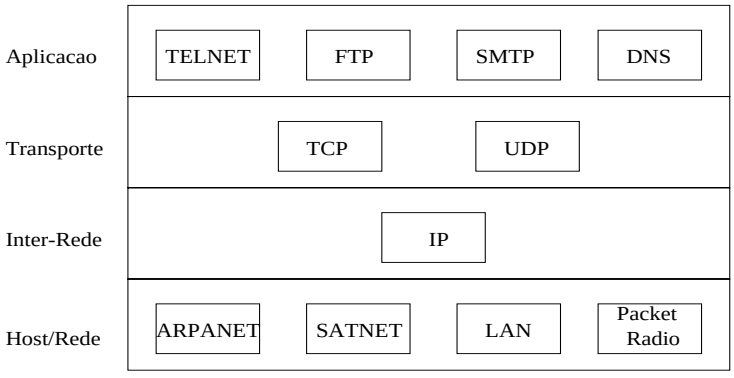


**Figura 5.1:** Modelo de Referência TCP/IP

A figura 5.1 apresenta as 4 camadas básicas do modelo TCP/IP juntamente com os protocolos utilizados nestas camadas.

- **Aplicação:** Nesta camada estão os protocolos de alto nível, utilizados por aplicações específicas da Internet como o TELNET (*login* remoto e terminal virtual), FTP (transferência de arquivos), SMTP (*e-mail*) e o HTTP (usado para buscar hipertextos na WWW).

- **Transporte:** A finalidade desta camada é permitir que pares de *hosts* de origem e destino possam manter uma conversação. Para isso, a camada de transporte do modelo TCP/IP define dois protocolo básicos: O TCP (*Transmission Control Protocol*) - protocolo orientado a conexão confiável - e o UDP (*User Datagram Protocol*) - um protocolo sem conexão e não confiável. O TCP é largamente utilizado em aplicações que necessitam de confiabilidade e que não podem perder dados durante a transmissão, como por exemplo transferência de arquivos. Dentre suas principais funções destacam-se o controle de fluxo e a garantia da entrega de mensagens. O UDP é mais utilizado em aplicações que requerem mais velocidade do que confiabilidade, um exemplo destas aplicações é a transmissão de voz.

- **Inter-Rede:** Esta camada tém por função básica a integração de todas os tipos de redes em um protocolo baseado em pacotes. Sua tarefa é permitir que os *hosts* injetem pacotes em qualquer rede e permitir que eles sejam transmitidos independentemente do destino (que pode ser em outra rede) [Tan96]. A camada inter-redes IP define um um formato de pacotes e um protocolo para o envio destes. Além de ter de entregar pacotes IP onde quer que seja necessário é também tarefa desta camada tratar questões relacionadas a roteamento de pacotes e congestionamento de linhas.

- **Host/Rede:** Esta é a camada física da rede, nela encontra-se o hardware de interconexão. O modelo não define padrões neste nível, a única coisa necessária é que a camada de Host/Rede utilize algum protocolo de baixo nível para que seja possível enviar pacotes IP. Na figura 5.1 vemos quatro dos inumeros tipos de redes que podem ser utilizados nesta camada.

Apesar do modelo TCP/IP ser o mais utilizado no mundo, ele tem alguns problemas, principalmente relacionados à documentação e organização dos protocolos [Tan96].

### 5.2.2 WWW - *World Wide Web*

O WWW[3] utiliza a arquitetura cliente/servidor e baseia-se em três tecnologias básicas: o protocolo HTTP (*HyperText Transfer Protocol*), para comunicação entre clientes e servi-

[3]Também chamado *Web*.

dores, a linguagem HTML (*HiperText Markup Language*), usada para criar documentos hipertexto e URLs (*Uniform Resource Locators*), que possibilitam a identificação dos documentos na rede [Alc98].

Existem dois tipos básicos de software para a Web: os *browsers* e os servidores HTTP. Os browsers são os aplicativos clientes que solicitam documentos aos servidores HTTP.

A figura 5.2 apresenta o modelo básico de funcionamento da Web.

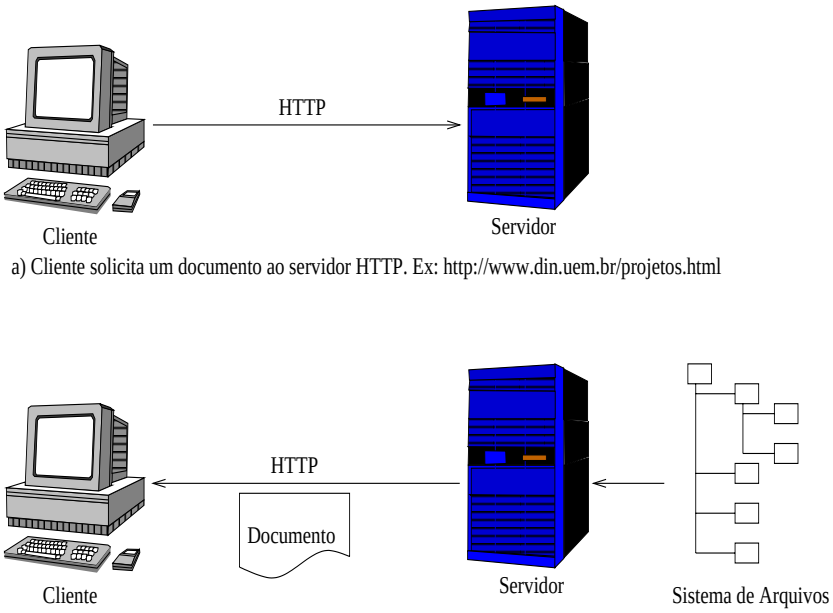


**Figura 5.2:** Modelo básico de funcionamento da *Web*

Existem muitos outros serviços realizados pelos *browsers* e pelos servidores, entretanto, num modelo clássico, são apenas programas clientes e servidores, respectivamente, para a apresentação de páginas HTML.

Os endereções dos recursos na *Web* são dados pelas URLs. Um exemplo de URL é o mostrado na figura 5.2: *http://www.din.uem.br/projetos.html*.

Atualmente os documentos acessados através da WWW não são mais apenas hipertextos e figuras. Já existem tecnologias que permitem a construção dos mais diversos sistemas na *Web*, utilizando recursos avançados como som, animações e acesso a grandes bancos de dados corporativos.

## 5.3 Tipos de Sistema na *Web*

Como visto anteriormente, desde a criação da *Web*, houve uma grande evolução no que diz respeito aos tipos de documentos apresentados na rede. As páginas estáticas de outrora agora se transformaram em complexas animações e formulários altamente interativos,

que possibilitam a criação dos mais diversos sistemas para a *Web*.

Alcântara [Alc98] divide os sistemas de informação da *Web,* ou WbIS (*Web Based Information System*) e 4 classes, conforme descrito a seguir.

### 5.3.1 Os sites de presença institucional e de conteúdo

Esta classe representa os sites essencialmente estáticos e contruídos com base no paradigma de páginas, isto é, um conjunto de páginas é construído e armazenado no sistema de arquivos do servidor *Web*. O grande objetivo desta classe de sistemas é apresentar informações ao usuário de maneira agradável. Um bom exemplo de site institucional é o do Departamento de Informática da UEM[4].

### 5.3.2 Os sites de serviços

Os sites de serviços representam o meio termo entre os sites que usam o paradigma de página e os que usam o paradigma de programação. Nestes sites é possível a criação dinâmica de páginas de acordo com os parâmetros fornecidos pelos usuários. Um exmplo de site desta classe é o serviço de busca Cadê[5].

### 5.3.3 WbIS transacionais

Os WbIS transacionais utilizam o paradigma de programação e provêm uma grande interação com SGBDs (Sistemas Gerenciadores de Banco de Dados) através da *Web* [Alc98].

Os sites que utilizam este paradigma assemelham-se aos sistemas cliente/servidor tradicionais, entretanto as facilidades em termos de implantação, manutenção e acesso que a *Web* oferece os tornam únicos em termos de número de usuários. Um exemplo bastante representativo deste tipo de sistemas são as lojas virtuais, que mantém um cadastro de clientes e produtos e permitem a compra através da *Web*.

### 5.3.4 Aplicações OCSI - *Object-oriented Client/Server Internet-based*

Esta ultima classe de aplicações combina a tecnologia *Web* com a de objetos distribuídos [Orf96] sobre um *middleware* cliente/servidor, para construção de sistemas com uma interface padrão (com o uso de navegadores), alto grau de reusabilidade (através da orientação a objetos e utilização de componentes) e distribuição facilitada peslo recursos da *Web* [Alc98].

[4] Acessível em *http://www.din.uem.br.*
[5] Acessível em *http://www.cade.com.br.*

Estas aplicações, geralmente são sistemas distribuídos [Col94] complexos que demandam um grande número de recursos tanto de clientes quanto de servidores, que podem ser inclusive móveis.

Um exemplo de sistemas deste tipo seriam os PSEEs na *Web*.

## 5.4 Tecnologias para Desenvolvimento na *Web*

Tendo em vista os vários tipos de sistemas que podem ser construídos na *Web*, é fácil perceber que diferentes tecnologias dão suporte à construção de diferentes classes de sistemas. A seguir existe uma breve descrição de algumas tecnologias utilizadas na construção de WbIS.

### 5.4.1 HTML - *HyperText Markup Language*

HTML é a linguagem para criação de hipertextos que deu origem a WWW como a se conhece atualmente. Sua sintaxe é bastante simples e intuitiva, o que facilitou em muito sua grande utilização pela *Web*.

A HTML, como o próprio nome diz, é uma linguagem de marcação, ou seja, comandos são inseridos, no texto para definir certos comportamentos dos mesmos. TEX e troff são outros exemplos muito conhecidos de linguagens de marcação.

Um arquivo HTML contém, além das informações a serem apresentadas, comandos que definem a formatação da página, *links* que a ligam a outras páginas e que indicam as figuras a serem apresentadas, além de tabelas e formulários.

Sem dúvida nenhuma, HTML é a tecnologia de desenvolvimento para *Web* mais utilizada atualmente, praticamente todas as páginas da *Web* contém pelo menos alguma linha de código em HTML.

### 5.4.2 CGI - *Common Gateway Interface*

A linguagem HTML permite a criação de formulários completos, entretanto, por ser uma linguagem apenas de marcação não permite a manipulação e processamento dos dados presentes nestes formulários. Entretanto, existe um padrão para o tratamento destes dados, ele se chama CGI.

Um CGI é um programa que geralmente é executado no servidor HTTP e responde a determinadas requisições de clientes. Os CGIs são normalmente usados para a construção de sites de serviços que necessitam de mecanismos que lhes permitam ter acesso a poderosos bancos de dados.

Por exemplo, suponha que exista uma página de busca por determinados tipos de carros. Na página principal há um formulário onde o usuário preenche alguns campos

especificando o modelo e detalhes a respeito do carro, e no fim da página existe um botão ¨procurar¨. Quando pressionado este botão, o *browser* manda os dados do formulário para o servidor HTTP que procura pelo CGI especificado pelo cliente HTTP. Se este existir, ele é executado com os parâmetros passados pelo servidor. Nesta execução do CGI, ele faz uma busca no banco de dados e com os dados encontrados constroi uma página HTML contendo informações sobre os carros que interessam ao usuário. Esta página criada é então enviada de volta ao *browser* para que este a interprete e apresente ao usuário.

O exemplo apresentado representa apenas uma pequena demonstração do que um CGI pode fazer pelos WbIS, existem vários tipos de CGIs que são utilizados para os mais diversos fins, entretanto a maior utilidade desta tecnologia é facilitar o acesso a bancos de dados. Dentre as principais linguagens para a criação de CGis destacam-se *Perl*, C e as linguagens de script do UNIX (como as utilizadas nos *shells sh* ou *bash*).

Apesar dos CGIs serem os mecanismos de acesso a bancos de dados mais utilizados, eles são também o de pior desempenho, pois cada vez que os CGIs são chamados pelos clientes HTTP, um novo processo é criado no servidor para atender a requisição. A criação de processos é uma tarefa cara para os sistemas operacionais, implicando em um tempo relativamente longo para a execução dos CGIs. Outro problema sério que os CGIs enfrentam é o fato de que a cada execução eles devem abrir e fechar conexões com o banco de dados. Esta operação de estabelecimento de conexão consome muitos recursos da máquina servidora, pois além de ser muito lenta, requer uma grande quantidade de memória [Alc98].

### 5.4.3 Java *Applets*

Mesmo com os CGIs apresentados na sessão anterior, ainda é impossível criar qualquer tipo de interação rápida em páginas HTML, isto devido ao fato de CGIs ficarem em servidores e, de modo geral, terem uma péssima performance. Para a criação de páginas verdadeiramente dinâmicas é necessária a utilização de *applets* escritos em Java [Gos99].

A linguagem Java foi criada pela Sun Microsystems como uma linguagem independente de plataforma para utilização em aparelhos de consumo orientados a informação. Mais tarde, devido à portabilidade da linguagem, ela foi levada a *Web*, onde obteve enorme sucesso. Uma importante característica da linguagem Java é o fato desta não ser usualmente traduzida para linguagem de máquina (como acontece com as linguagens não interpretadas), mas sim para *bytecodes*, uma espécie de ¨lingugem de máquina¨ independente de máquina que é interpretada pela JVM (*Java Virtual Machine*) [Lin96], uma máquina construída em software que executa estes *bytecodes*.

A sintaxe da linguagem Java é bastante semelhante à da linguagem C++, entretanto Java provê facilidades de acesso a uma série de recursos como *threads*, *networking* e

segurança.

Os aplicativos Java utilizados na Internet são chamados *applets*. Estes aplicativos são transferidos juntamente com as páginas HTML e ao chegarem no *browser*, são interpretados por JVMs locais.

Por razões de segurança, os *applets* têm uma série de restrições [Sun99a], que são apresentadas a seguir.

- Um *applet* não pode carregar bibliotecas nem definir métodos nativos;
- Não pode ler ou escrever arquivos no *host* que o está executando;
- Não pode estabelecer conexões de rede com outros *hosts* diferentes do que ele veio (em geral o servidor HTTP);
- Não pode executar nenhum programa no *host* que está excutando;
- Não pode ler certas propriedades do sistema;
- As janelas que um *applet* cria tem aparência diferente das janelas usuais do sistema.

Apesar destas restrições, os *applets* ainda têm uma série de capacidades [Sun99a]:

- *Applets* usualmente podem estabelecer conexões com os *host* de onde vieram;
- *Applets* podem facilmente apresentar documentos HTML;
- *Applets* podem invocar métodos públicos de outros *applets* na mesma página;
- *Applets* carregados da mesma máquina em que estão executando não obedecem a nenhuma das restrições impostas;
- Apesar de muitos *applets* interromperem sua execução quando o usuário deixa a página em que estavam, eles não tem de fazer isto.

Apesar dos grandes benefícios, a utilização de *applets* Java enfrenta um grave problema: sua transferência e interpretação são um processo um tanto quanto demorado, o que causa a navegação por páginas que contém *applets* bastante lenta.

A utilização de *applets* é recomendada somente quando faz-se necessária a disponibilização de aplicativos completos em páginas HTML, e sempre buscando diminuir o tamanho dos *applets*, para consequentemente diminuir o *overhead* na rede.

### 5.4.4 JavaScript

JavaScript é uma linguagem criada pela Netscape como uma extensão do HTML para dar mais poder aos *browsers* [ipl00]. Ela emprega elementos da linguagem Java, como sintaxe básica, componentes de interface e modelo de eventos, em uma linguagem de *script*[6] elegante e leve.

JavaScript é utilizada para criar páginas que contém aspectos dinâmicos, ou seja, que realizam algum tipo de interação com o usuário ou mesmo alteram seu estado. Note que com o HTML puro é impossível incluir qualquer tipo de atividade dinâmica às páginas, haja vista que esta é apenas uma linguagem de marcação.

A introdução de JavaScript nos WbIS proporciona dois grandes benefícios ao sistema como um todo:

1. **Menor tráfego na rede:** A verificação de campos e outros tipos de processamentos mais leves podem agora ser feitos no cliente, eliminando assim a necessidade de se enviar dados para CGIs no servidor a fim de que sejam realizado processamento irrisório;

2. **Mais sofisticação na interface com o usuário:** JavaScript permite ao desenvolvedor incluir um sem número de recursos gráficos avançados em suas páginas, tornando desta forma a navegação do usuário mais agradável.

Outro aspecto relevante a respeito do JavaScript é o fato dele ser bem mais leve que os *applets* Java, desta forma interfaces que antes só eram possíveis através de pesados *applets*, agora podem ser construídas incluindo-se algumas linhas de código JavaScript no arquivo HTML da página. Vale ressaltar entretanto, que o JavaScript apesar de muito útil na confecção de interfaces para *Web* não tem tantos recursos quanto a linguagem Java e seus pesados *applets*.

### 5.4.5 Outras Tecnologias

Existem várias outras tecnologias na *Web*: ASP, JSP, Servlets, VBScript, Flash, XML, entre outras. Entretanto, grande parte destas ainda carece de amadurecimento no mercado.

## 5.5 Arquiteturas de Objetos Distribuídos

Como já abordado, os PSEEs na *Web* são essencialmente aplicações OCSI, estas aplicações utilizam um modelo de componentes ¨*plug and play*¨ para o desenvolvimento de

[6] Linguagens de script diferem das tradicionais devido ao fato de serem interpretadas e não compiladas.

software. Estas aplicações utilizam o conceito de objetos distribuídos, que podem estar espalhados pela rede e se comunicar através de uma interface que todos entendem.

As arquiteturas de objetos distribuídos baseadas na *Web*, utilizam os navegadores como interface padrão para objetos que podem interagir com outros objetos espalhados por toda rede. Este tipo de arquitetura normalmente necessita de um *middleware* sofisticado, que permita a objetos com localizações distintas se comunicarem.

Nesta sessão serão abordadas algumas arquiteturas de objetos distribuídos embasadas em duas palavras: Interoperabilidade e Java.

### 5.5.1 CORBA - *Common Object Request Broker Architecture*

CORBA [Omg99a] é um padrão aberto definido pela OMG (*Object Management Group*), cujo principal objetivo é definir uma arquitetura capaz de simplificar o acesso a recursos remotos em ambientes heterogêneos. O padrão CORBA especifica uma arquitetura que define alguns componentes, serviços e facilidades para a construção de aplicações distribuidas heterogêneas orientadas a objeto.

Várias são as características da arquitetura CORBA. A seguir temos algumas [Omg99a, Orf96]:

- Independente de lingugem de programação;
- Independente de plataforma[7] ;
- Centrada no paradigma da orientação a objetos;
- Forte separação entre interface e implementação de objetos.

O padrão CORBA se baseia no conceito de *middleware*: uma camada intermediária que esconde a complexidade de se estabelecer comunicações com máquinas remotas [Bes99a]. Portanto, CORBA é um conjunto de componentes de software que permitem a construção e comunicação entre objetos remotos como se eles estivessem na mesma máquina, desta forma não há nenhuma diferença entre a sintaxe de uma chamada de método de um objeto local e de um objeto remoto [Bes99a].

A figura 5.3 apresenta a arquitetura básica do padrão CORBA [Sch00].

[7] Sistema Operacional e Hardware.

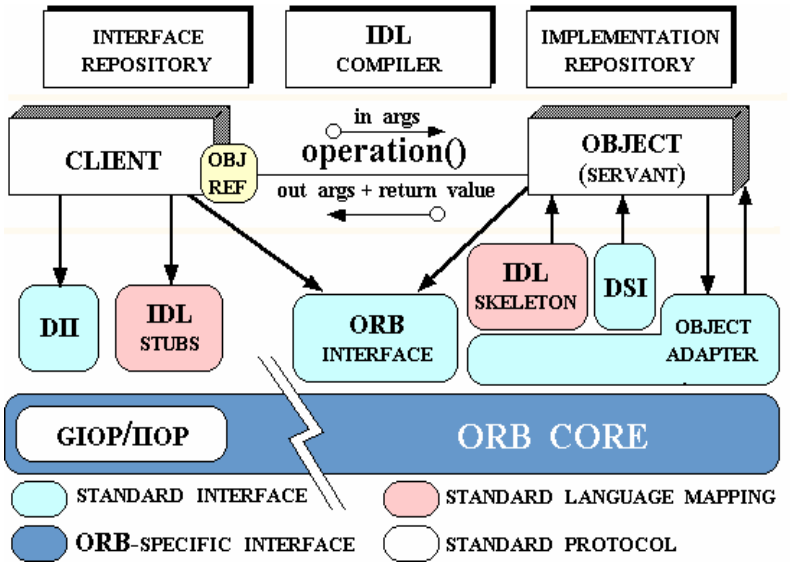


**Figura 5.3:** Arquitetura Geral do padrão CORBA

Na figura 5.3 podemos identificar todos os componentes básicos definidos pela arquitetura CORBA. Dentre estes descam-se alguns, a saber:

- **Compilador IDL:** IDL (*Interface Definition Language*) é a linguagem utilizada para definir as interfaces dos componentes da arquitetura CORBA, esta lingugem se faz necessária pelo fato da arquitetura ser independente de linguagem de programação, desta forma, descreve-se a interface de uma maneira comum, e esta é traduzida para uma linguagem convencional utilizando um compilador de IDLs. Para realizar esta tradução, o compilador deve implementar o mapeamento proposto pela OMG, que especifica como a sintaxe e a semântica da IDL serão representadas em uma determinada linguagem. Já existem mapeamentos para as linguagens C, C++, Java, Smaltalk, ADA, entre outras.

- **Objeto (Servo):** É a entidade de software que implementa a interface definida em IDL. Conhecido também como objeto CORBA.

- **Cliente:** O cliente é a parte do sistema que tem acesso ao objeto remoto CORBA.

- **ORB (*Object Request Broker*):** O ORB provê mecanismos para a comunicação transparente entre os clientes e servidores. Entre outras funções, o ORB é responsável por fazer com que a chamada de métodos no clinete se pareça local, desta forma simplificando a vida do engenheiro de sistemas distribuídos.

- **IDL *Stub* e *Skeleton*:** Os *stubs* e *skeletons* são, repectivamente, as camadas de software que ligam o cliente e o servidor de maneira transparente ao ORB. Os *stubs* e *skeletons* são normalmente gerados pelos compiladores de IDL.

Além da arquitetura básica, a OMG define mais dois conjuntos de padrões relacionados ao CORBA: *CORBA Services* [Omg99b] - que definem serviços úteis a todos os tipos de objetos CORBA como por exemplo segurança e persistência e *CORBA Facilities* - que definem padrões para determindos domínios como telecomunicações [Omg98] ou Medicina [Omg99c].

O padrão CORBA contém um *framework* muito mais complexo do que o mostrado neste texto, para maiores informações consulte [Omg99a, Orf96].

### 5.5.2 RMI - *Remote Method Invocation*

O RMI pode ser descrito como um *framework* para a construção de sistemas baseados em objetos distribuídos escritos em Java. Basicamente, o RMI permite que um objeto localizado em uma máquina virtual Java possa chamar métodos de objetos localizados em JVMs remotas como se estes fossem locais [Sun99d].

Apesar do RMI também reforçar a separação entre interface e implementação de objetos, não há nenhuma IDL no RMI, pois o fato de ter sido projetado para aplicações em Java puro, permite que a descrição das interfaces dos objetos remotos seja feita nesta linguagem.

O RMI é muito mais simples que o CORBA, porém não oferece o grande número de funcionalidades deste, mesmo assim tem conseguido um grande número de adeptos, principalmente na comunidade Java, onde sua simplicidade é especialmente apreciada.

Para maiores informações a respeito do RMI consulte [Sun98, Sun99a, Sun99d].

### 5.5.3 Jini

Jini é um sistema distribuido baseado na idéia de grupos federados[8] de usuários e dos recursos requeridos por estes usuários. O objetivo principal da arquitetura Jini é transformar a rede em uma ferramenta simples e flexivel onde recursos podem ser encontrados por clientes humanos ou computacionais [Sun99b]. Desta forma levando ao extremo o *slogan* da Sun *¨the network is the computer¨*[9].

Os serviços que a arquitetura Jini provê podem tanto ser implementados em dispositivos de hardware, como impressoras, quanto em programas de software. Acima de tudo, a idéia é prover às redes um maior dinamismo.

A arquitetura Jini consiste dos seguintes componentes [Sun99]:

- Um conjunto de componentes que provêem uma infra-estrutura para serviços federados em um sistema distribuido;

[8] Grupos fechados de objetos, onde todos tem acesso a todos.
[9] A rede é o computador.

- Um modelo de programação que suporte e encorage a produção de serviços distribuídos.

- Serviços que podem fazer parte de sistemas federados Jini e que podem oferecer funcionalidades para qualquer outro membro de sua federação.

O ambiente Jini estende o ambiente Java para suportar uma rede de JVMs. O ambiente de programação Java é bastante propício para a computação distribuida devido ao fato de suportar transferência tanto de dados, quanto de código[10], e também define mecanismos capazes de garantir a segurança em ambientes distribuídos.

A arquitetura Jini é fundamentada nos conceitos de Clientes, Serviços e Eventos.

- **Clientes:** são os componentes da federação que necessitam, de algum serviço;

- **Serviços:** são os componentes da federação que fornecem funcionalidades a todos os outros membros da mesma. Um serviço pode ser também um cliente de outro serviço da federação;

- **Eventos:** Jini estende o modelo de eventos da lingugem Java para suportar eventos distribuídos. Assim eventos que ocorrem em uma JVM podem ser percebidos em todas as outras da rede, desde que algum objeto executando nestas registre interesse no evento.

Um serviço muito especial do Jini é o *Lookup Service* [Sun99b]. Ele é responsável pela entrada e saída de elementos em uma federação de maneira transparente e automática.

Para ilustar o funcionamento de um sistema Jini, considere uma impressora que implementa um serviço Jini. Ao ser colocada na rede ele faz um *multicast* à procura de um *Lookup Service*, assim que ela obtém resposta se registra no mesmo, e seus serviços ficam disponíveis à federação. Considere que após algum tempo um outro membro da federação necessita de um serviço de impressão. Ele procura no *Lookup Service* um serviço de impressão, assim que o encontra ele obtém uma referência ao serviço e estabelece uma comunicação direta com o serviço, possívelmente passando a este o documento a ser impresso.

A arquitetura Jini é ainda muito nova, e sente ainda a falta de maturidade, entretanto é inegável seu potencial e robustez como sistema distribuido dinâmico. Várias empresas estão investindo na tecnologia, entretanto estudos mais aprofundados devem ser realizados a fim de atestar sua robustez na implementação de ambientes de engenharia de software.

[10] Através de RMI e Serialização de Objetos.

## 5.6 Considerações Finais

Neste capítulo foram apresentados os aspectos técnicos da internet e tecnologias para a construção de sistemas na WWW, bem como uma classificação dos sistemas de informação na Web (ou WbIS). Uma classe em especial de WbIS mereceu maior atenção: as aplicações OCSI. Estas aplicações utilizam tecnologias bastante complexas, que ainda são muito novas e sentem a falta de maturidade. Foram apresentadas três arquiteturas para objetos distribuídos: CORBA, RMI e Jini, dando ênfase maior ao padrão CORBA, que será utilizado neste trabalho.

A experiência em [Gim99, Bes99b] mostra que os PSEEs são, por excelência, aplicações OCSI, desta forma, toda a definição da arquitetura do PSEE proposto neste trabalho foi embasada em um sistema de objetos distribuídos.

# Capítulo 6

# Resultados Parciais

## 6.1 Introdução

Tendo em vista os estudos realizados, foi possível coletar os requisitos necessários à construção de uma primeira versão da arquitetura do ambiente proposto neste trabalho, a esta arquitetura foram aplicados padrões de software e *frameworks*, a fim de acrescentar experiências bem sucedidas ao projeto. Após a definição da arquitetura, iniciou-se o processo de implementação de um dos módulos da interface do ambiente, sempre tendo em vista as tecnologias utilizadas na *Web* e os requisitos coletados.

Neste capítulo serão apresentadas a versão atual da arquitetura de software proposta e uma breve descrição dos componentes e relacionamentos desta arquitetura, bem como os primeiros resultados referentes a implementação da interface do ambiente.

## 6.2 WPSEE - *Web-based Process-centered Software Engineering Environment*

O nome escolhido para o protótipo do ambiente em desenvolvimento foi WPSEE (*Web-based Process-centerd Software Engineering Environment*), ou Ambiente de Engenharia de Software centrado em Processo e baseado na *Web*.

O princípio básico do projeto foi construir um ambiente que tenha todas as características de um PSEE mas que utilize a terminologia relacionada aos sistemas gerenciadores de *workflow*, para que o WPSEE possa ser definido como uma máquina de *workflow* que suporte processos de software.

Um segundo princípio do projeto foi a capacidade de integração com a *Web*. A interface do sistema deve ser implementada levando em consideração sua integração com *browsers* e com o servidor do ambiente (gerenciador de processos).

### 6.2.1 Arquitetura Geral

Conforme descrito no capítulo 4, a arquitetura de um software descreve os componentes do sistema e os relacionamentos entre eles, uma primeira definição para a arquitetura do ambiente foi definida em um nível mais alto de abstração, procurando identificar os componentes básicos do sistema e as interações necessárias entre eles.

Tanto so PSEEs quanto so WfMSs tem dois módulos básicos de interface, um para definição de processos e recursos (participantes, ferramentas, etc) e outro para acesso por parte dos usários às tarefas agendadas a eles (agenda). Estes componentes da interface devem ser acessados pela internet através de um *browser* e também devem se conectar ao gerenciador de processos do sistema, que esta localizado na mesma rede do servidor HTTP[1].

A abordagem utilizada no gerenciador de processos do sistema é a proposta em [Huz00]: um gerenciador de processos distribuído baseado em uma arquitetura de *microkernel*. Nesta abordagem, cada sub-gerenciador é um servidor do sistema, partindo assim o gerenciador de processos em uma série de servidores específicos dedicados a determinadas tarefas.

A versão atual da arquitetura do ambiente WPSEE é apresentada na figura 6.1 [Huz00].

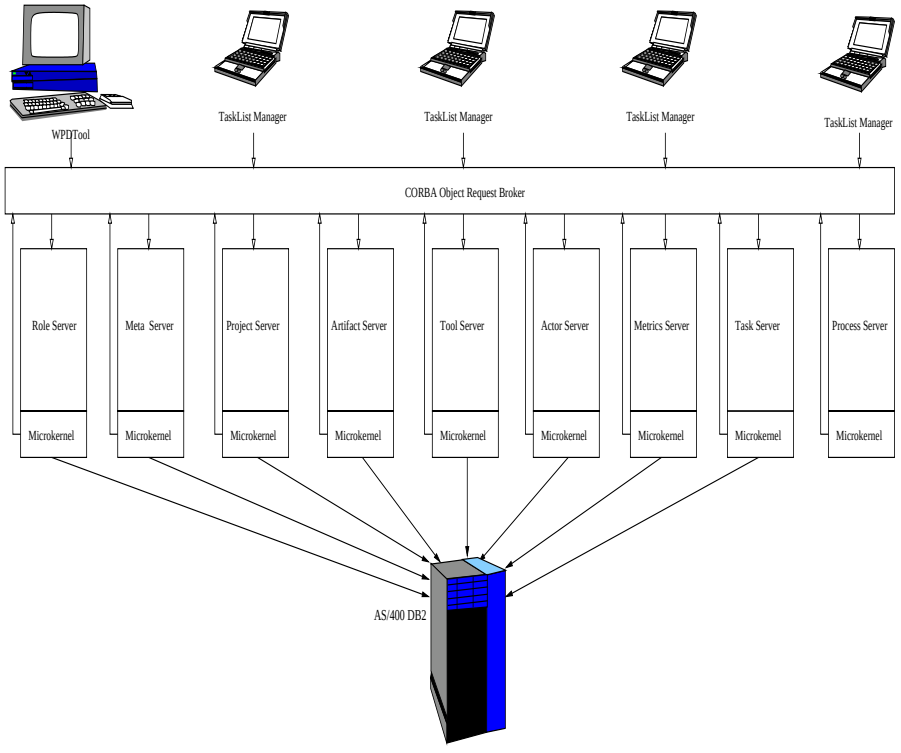


**Figura 6.1:** Arquitetura do Ambiente WPSEE

[1]Esta é uma questão importante pois grande parte dos sites utilizam-se de algum tipo de proteção contra invasores (como firewalls), o que dificulta a implantação de clientes que se comunicam com servidores atrás do servidor HTTP. Felizmente grande parte das implementações do padrão CORBA contém ferramentas que resolvem este problema.

Na arquitetura apresentada na figura 6.1 pode-se identificar uma série de componentes, divididos em quatro categorias básicas - interface (WPDTool e WAgenda), servidores, banco de dados e mecanismo de integração. As próximas sessões abordam estas categorias, bem como seus componentes.

### 6.2.2 WPDTool - WPSEE *Process Definition Tool*

WPDTool é a ferramenta utilizada pelos gerentes de projetos para cadastrar novos participantes dos processos (chamados de atores), novas ferramentas, novos cargos, e principalmente para a definição de arquiteturas de processos e instanciação das mesmas.

Uma arquitetura de processo é uma meta-estrutura geral de processo que descreve os diversos tipos dos elementos de processo: tipo de tarefa (*TaskType*) para tarefas, tipo de artefato (*ArtifactType*) para artefatos, tipo de ferramenta (*ToolType*) para ferramentas e um cargo ou tipo de ator (*Role*) para atores.

Assim a definição de uma arquitetura de processos compreende a definição dos tipos de passos a serem executados, por determinados tipos de pessoas e utilizando determinados tipos de ferramentas para se executar um tipo de processo.

Definida a arquitetura de processo, é possível instanciá-la como um processo alocando recursos para sua execução. Os recursos de possível alocação são atores ou participantes (*Actor*), ferramentas (*Tool*) e artefatos (*Artifacts*).

O modelo de processos utilizado pelo WPSEE é uma versão simplificada do modelo apresentado pelo padrão *Process Manager* (sessão 4.7.1)[Wei98].

A representação gráfica de processos fornecida pela ferramenta utiliza também outros elementos de processo, entretanto sua inclusão visa apenas reforçar a representatividade semântica do modelo desenvolvido.

Uma vantagem relevante da utilização do conceito de arquitetura de processos é a reusabilidade que ele proporciona, pois definida uma arquitetura, esta pode ser instanciada várias vezes para processos semelhantes. Por exemplo, uma arquitetura de processos de compra de materiais em uma repartição publica poderia ser tanto instanciada para comprar uma caixa de canetas, quanto para a compra de um computador.

A sessão 6.4 descreve a implementação da ferramenta WPDTool.

### 6.2.3 WAgenda - WPSEE Agenda

A agenda[2] é o componente da interface do ambiente que deve interagir com os participantes do processo. O principal objetivo da agenda é ler os dados referentes às tarefas que determinados usuários tem de executar e apresentá-las a ele.

[2]Também chamada *Task List Handler*.

O ambiente ExPSEE [Gim99], fornece também um ambiente cooperativo de programação de tarefas, assim cada tarefa pode ser programada pelo gerente de projeto. Estes programas de processo podem ser acessados pelos seus executores (os atores designados para a tarefa) através de suas agendas.

Além de apresentar tarefas ao usuário, é esperado que uma agenda leve em consideração o trabalho em grupo (*groupware*) e de suporte à cooperação de pessoas na execução de tarefas. Este suporte pode se dar através de um serviço de *e-mail*, *chat*, e teleconferência entre outros.

A principal vantagem de se ter um PSEE integrado a *Web* é facilitar o acesso dos usuários ao seu ambiente de trabalho, pois grande parte das máquinas atualmente tem acesso à Internet. Assim, o acesso à agenda independe da localização do usuário, basta apenas um *browser* e o acesso à rede para que este tenha acesso as suas tarefas, de tal forma que possa executá-las.

Outra vantagem de uma agenda integrada a *Web* é a facilidade de suporta as atividades de cooperação que a Internet oferece: a grande maioria dos *browsers* conta com um cliente de *e-mail*, e serviços de *chat* podem ser facilmente implementados na Internet.

### 6.2.4 Gerenciador de Processos Distribuído

Um gerenciador de processos distribuído constitui-se em um conjunto de servidores específicos a certos domínios que são executados em processos diferentes (possívelmente em *hosts* diferentes) e um serviço de integração que permita a comunicação entre eles.

A utilização desta abordagem distribuída tem como principal vantagem a escalabilidade - novos gerenciadores podem ser adicionados ou subtraídos ao sistema e os outros domínios de gerenciamento não serão afetados[3]. Outra vantagem pertinente é o aumento da disponibilidade do ambiente, pois ao invés de cada transação ocupar o gerenciador de processos como um todo, ocupa apenas o servidor do domínio em questão. Por exemplo, se a WPDTool requisitar a inserção de um novo grupo de atores no ambiente, ela não deverá chamar o gerenciador de processos como um todo, mas sim o servidor de atores que de posse deste grupo saberá o que fazer.

Uma desvantagem desta abordagem é a pior performance no tratamento de transações que envolvem vários gerenciadores, visto que a comunicação entre eles é uma tarefa relativamente custosa. Um exemplo de tarefa deste tipo é a instanciação de uma arquitetura de processo, visto que esta operaçõa exige a participação de quase todos os gerenciadores do sistema.

A nível de ilustração figura 6.2(a) apresenta a arquitetura de um gerenciador de processos convencional, ou monolítico (como o utilizado atualmente no ExPSEE), já a

[3]Não serão afetados desde que não haja uma dependência entre os mesmos.

figura 6.2(b) apresenta a arquitetura do gerenciador de processos distribuido, como o proposto para o WPSEE.

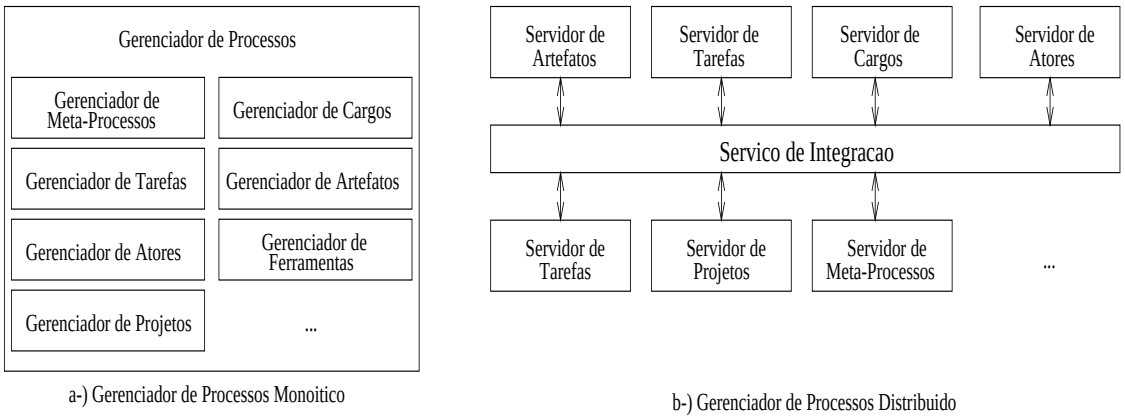


**Figura 6.2:** Gerenciadores de processos Monolítico(a) e Distribuído(b)

Como visto na figura 6.1, além do gerenciador de processos do WPSEE ser distribuído, ele também é baseado em uma arquitetura de microkernel (sessão 4.7.2), constituindo-se assim em um sistema baseado em *microkernel* distribuído.

Em um sistema baseado em *microkernel* distribuído, todos os componentes tém acesso a um conjunto básico de funcionalidades que estão implementadas no *microkernel*. Desta forma todos os servidores do WPSEE agregam um componente que define as principais rotinas de comunicação com os outros servidores e com o banco de dados, bem como políticas de reaproveitamento de conexões com o banco de dados e *cache* de informações.

### 6.2.5 Base de Dados

A base de dados especificada para o WPSEE é o sistema gerenciador de banco de dados DB2 for AS/400 da IBM. Este SGBD foi escolhido por algumas razões como:

- SGBD largamente utilizado pela indústria e comércio;
- Alta performance e confiabilidade;
- Existência de *drivers* JDBC (*Java Data Base Conectivity*) que facilitam o acesso de aplicativos Java ao sistema;
- É um SGBD objeto-relacional;
- Está disponível ao laboratório de engenharia de software sem ônus adicional devido a um convenio UEM - IBM.

Além das razões supracitadas pode-se destacar ainda a alta confiabilidade da plataforma AS/400, onde o SGBD está instalado.

### 6.2.6 Mecanismo de Integração

Baseando-se nos estudos realizados em [Bes99a, Bes99b] escolheu-se o padrão CORBA como mecanismo de integração para o ambiente. Desta forma todos os comoonentes do WPSEE (com excessão do SGBD) tem ligação com o ORB, e assim podem acessar todos os outros.

## 6.3 Implementação do Modelo de Processos

Apesar do cronograma inicial deste trabalho não prever estudos a respeito de modelos de processo, no decorrer da implementação da ferramenta de definição de processos (WPDTool) surgiu a necessidade de se definir um modelo de processos e implementá-lo, a fim de que a ferramenta pudesse criar arquiteturas de processo e instancia-las. Desta forma definiu-se o modelo de processos discutido na sessão 6.2.1 e utilizando IDL e Java implementou-se este modelo.

A figura 6.3 apresenta um diagrama de classes simplificado do modelo implementado atualmente.

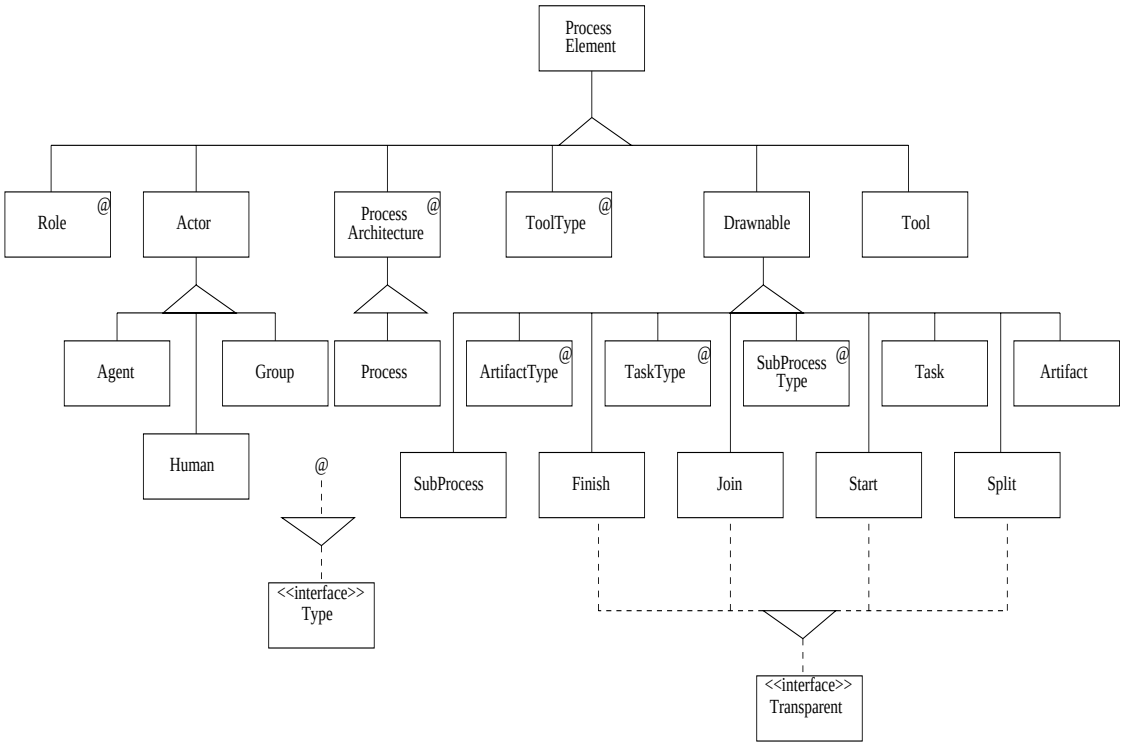


**Figura 6.3:** Diagrama de classes do modelo de processo do WPSEE

O diagrama de classes da figura 6.3 apresenta as classes e interfaces definidas no modelo de processo.

- ***ProcessElement:*** Esta é a classe base de todas as outras classes do modelo de processos. Nela estão definidos atributos importantes como id, nome, versão e

descrição;

- ***Drawnable:*** Todas as classes que podem ser desenhadas pela WPDTool estendem esta interface, nela estão contidas informações a respeito da posição do objeto no painel da ferramenta;

- ***Actor:*** Superclasse de todos os tipos de atores. Todo ator tem pelo menos uma lista de cargos que ele suporta;

- ***Agent*****:** Tipo especial de ator. A classe *Agent* descreve um agente de software capaz de realizar algum tipo de tarefa. Nesta classe estão armazenadas informações sobre como ativar o agente;

- ***Human:*** Um ator humano. Nesta classe são armazenadas várias informações a respeito do ator, como por exemplo a senha de acesso ao sistema, e a quantidade de horas disponíveis de trabalho por semana;

- ***Group:*** Uma tarefa também pode ser executada por um grupo, por isso existe a classe *Group* como sendo uma estensão da classe *Actor*. Nela estão definidos o lider e a lista de membros do grupo;

- ***Type:*** A interface *Type* define métodos para a instanciação de tipos referentes a uma arquitetura. Ela é implementada por todos os elementos pertencentes à arquitetura de processos. Por razões de estética as classes que implementam *Type* na figura 6.3 tém um @ no canto superior direito;

- ***Role:*** Um cargo (ou papel) definido na organização. Todo ator suporta pelo menos um cargo;

- ***ToolType:*** Tipo de ferramenta. Toda a ferramenta tem um tipo;

- ***ArtifactType:*** Tipo de artefato. Artefato é todo produto ou entrada de uma tarefa. Ex: relatório de testes, código fonte, diagrama de classes, etc.

- ***TaskType:*** Tipo de tarefa. Esta classe define pré e pós condições para a execução deste tipo de tarefa, bem como uma lista de tipos de ferramentas utilizadas em sua execução e cargos aptos a executá-la. Outra informação relevante que *TaskType* armazena são os tipos de artefatos requeridos para a execução de tarefas deste tipo e os tipos de artefatos produzidos;

- ***ProcessArchitecture:*** Classe que representa uma arquitetura de processos como um todo. Nesta classe definem-se algumas informações auxiliares e a lista de objetos que compõem a arquitetura de processo;

- ***SubProcessType:*** Esta classe encapsula uma arquitetura de processo como se fosse uma tarefa. Desta forma, pode-se formar novas arquiteturas a partir da composição de outras. O conceito de sub-processo é útil tambem na implementação de *loops*, pois um sub-processo pode ser repetido várias vezes[4] até que sua pós condição seja satisfeita;

- ***Process:*** Esta classe representa um processo tanto de software, quanto de negócios. Ele armazena vários dados referentes à administração e gerenciamento do processo, como por exemplo o ator responsável, o estado e a prioridade do processo. Esta classe estende a arquitetura de processo devido ao grande número de atributos em comum a ela, entretanto semânticamente esta especialização está incorreta;

- ***Task:*** Uma tarefa do processo. Nela estão armazenadas referências aos recursos utilizados em sua execução (atores e ferramentas), os artefatos requeridos e os produzidos para seu início e término respectivamente bem como as pré e pós condições;

- ***Artifact:*** Além das informações triviais (herdadas de *ProcessElement*), a classe artefato armazena referências para quem a produziu e quem necessita dela;

- ***SubProcess:*** Representa um sub-processo e portanto encapsula um processo. Para que um processo possa ser encapsulado em um sub-processo os artefatos necessários para a inicialização do processo devem ter o mesmo tipo dos artefatos necessários para o início do sub-processo. Esta regra aplica-se também aos tipos produzidos pelo processo e pelo sub-processo (devem ser de tipos iguais).

- ***Start:*** Representa o início do processo. O elemento início foi incluido apenas para acrescentar poder semântico à representação gráfica de processo definida pela WPDTool;

- ***Finish:*** Mesmo caso de *Start*. Este elemento representa o fim de um processo.

- ***Split:*** O elemento *Split* representa uma réplica de artefatos. Ex: o relatório 1 produzido pela tarefa 1 é necessário na tarefas 2 e 3, assim a saída da tarefa 1 (relatório) passa por um Split e transforma-se em dois outros objetos semelhantes ao primeiro (relatório 2 e relatório 3), que servem de entrada para as tarefas 2 e 3. A classe *Split* armazena referências do artefato de entrada e dos artefatos de saída;

- ***Join:*** O *Join* é uma composição de artefatos. Os artefatos de entrada são agrupados em um único artefato de saída. Esta classe armazena várias referências aos artefatos de entrada e uma referência ao artefato composto de saída;

- ***Transparent:*** Esta interface é implementada por todos os elementos do processo cuja semântica pode ser representada por uma pré ou pós condições de seus objetos adjacentes. Assim, esta interface é implementada por todos os objetos incluídos no modelo para dar mais poder semântico a representação gráfica implementada pela WPDTool (Ex: *Start*, *Finish*, *Join*, ...). *Transparent* define métodos que verificam se o elemento está sendo utilizado em uma arquitetura de processo ou em um processo própriamente dito, entre outros.

Note que o diagrama da figura 6.3 é uma representação simplificada do diagrama de classes do modelo de processos e não define os relacionamentos entre os objetos (a não ser *extends* e *implements*[5]) nem os atributos e métodos dos mesmos.

A implementação do modelo de processos foi feita utilizando-se a IDL e na tradução das descrições para Java utilizou-se o compilador de IDls do *ORBacus 4.0 beta*, uma implementação do padrão CORBA 2.3. Esta implementação foi escolhida por ser gratuíta e implementar *valuetypes*[6].

Para uma descrição mais elaborada do modelo de processos ver a definição em IDL presente no apêndice A.

## 6.4 Implementação da WPDTool

Como já apresentado, o principal objetivo da ferramenta WPDTool é fornecer ao gerente de processos uma interface para definição, controle e manutenção de arquiteturas e processos. Assim está sendo implementada uma interface gráfica de manipulação direta para à definição de arquiteturas de processos e sua instanciação.

A figura 6.4 apresenta a interface da ferramenta no estado atual.

[4]Entretanto, diferentes versões de suas tarefas são executadas.

[5]Uma classe implementa uma interface.

[6]Objetos que podem ser passados por valor como argumentos de métodos, ou seja, podem ser transferidos pela rede.

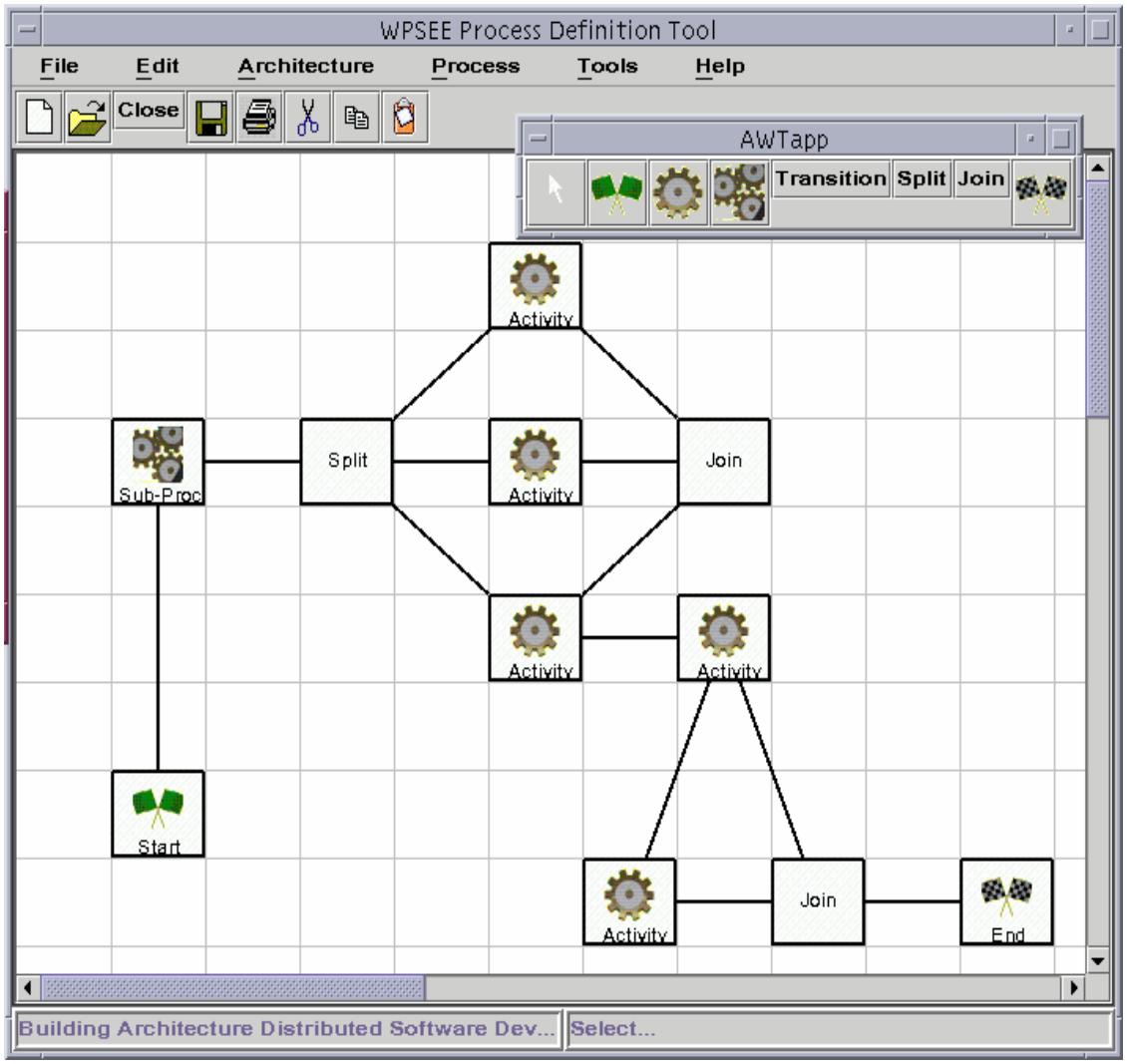


**Figura 6.4:** Interface atual da WPDTool

A janela apresentada na figura 6.4 mostra uma arquitetura de processo sendo definida, note que todos os elementos de processos estão representados nesta arquitetura[7]. A janela que está sobreposta à principal, com o título *AWTapp* contém os elementos de processo que podem ser inseridos na arquitetura do painel[8].

A integração do modelo de processos à ferramenta se dá através da inclusão em cada figura representada no painel de seu correspondente no modelo de processos, e a cada operação realizada sobre a arquitetura, um algoritmo verifica se o modelo está de acordo com o grafo apresentado no painel, assim todas as possíveis inconsistências são corrigidas antes dos dados da arquitetura (ou processo) serem utilizados.

A WPDTool está sendo implementada utilizando-se a linguagem Java, particularmente o kit de desenvolvimento jdk1.2.2 da Sun Microsystems. A interface construída foi implementada utilizando-se o *framework Swing* [Sun99a]. A interface manipulável onde se apresentam os elementos de processo (painel central da ferramenta) foi impl-

[7] Os artefatos são representados pelas linhas que conectam outros elementos da arquitetura de processo.
[8] Alguns elementos ainda não tem seus ícones.

mentada utilizando-se a Java2D API [Sun99c] e o *framework VeryHot*.

O programador mais atento notará que esta interface não está integrada a *Web* ainda, isto se deve a dois fatos:

1. Os *browsers* atualmente em uso não suportam Java 1.2, e os *plug-ins*[9] nem sempre funcionam bem;

2. A execução da ferramenta é bastante pesada, portanto a inclusão da mesma em um *applet* na fase de desenvolvimento diminuiria a agilidade do processo de codificação/teste.

Apesar desta não integração com a *Web*, todas as restrições para *applets* estão sendo respeitadas a fim de fazer com que o processo de transformação da aplicação em um *applet* seja trivial.

## 6.5 Considerações Finais

Neste capítulo foram apresentadas a versão atual da arquitetura integrada à *Web* de um PSEE e as implementações já realizadas desta arquitetura. O ambiente proposto, chamado WPSEE, foi dividido em quatro camadas básicas: interface com o usuário, gerenciadores, mecanismo de integração e banco de dados.

Atualmente a ênfase tem sido dada para a definição de um modelo de processos que possa ser transportado via CORBA dos clientes (WPDTool e WAgenda) para o gerenciador de processos distribuído e vice versa.

Grande parte das funcionalidades da ferramenta de definição de processos (WPDTool) já está implementada, principalmente no que se refere à definição de arquiteturas de processo. A próxima fase visa principlamente completar a integração da ferramenta com o modelo proposto e sua integração com a *Web* através de um *applet*.

[9] Bibliotecas que são adicionadas ao browser para manipulação de arquivos diferentes de HTML.

# Capítulo 7

# Conclusões Parciais

Com base nos estudos realizados, de tópicos como ambientes de engenharia de software, sistemas de *workflow*, arquitetura de software e Internet, bem como na experiência dos autores no desenvolvimento de aplicações baseadas em objetos distribuídos, foi definida a arquitetura do ambiente de engenharia de software orientado a processo integrado a *Web* (WPSEE).

No decorrer dos estudos realizados, decidiu-se iniciar a implementação da ferramenta de definição de processos, para testar a passagem do modelo definido na ferramenta para o servidor. O padrão CORBA até pouco tempo não suportava a passagem de objetos por valor (assim como Java não suporta esta funcionalidade para objetos simples), e por isso, inicialmente foi considerada a opção de se traduzir os objetos definidos na interface em registros de dados que pudessem ser transmitidos como *structs*. Entretanto as experiências realizadas neste sentido mostraram que a abordagem, além de ser bastante complexa (por exigir algoritmos de dificil implementação) não se constitui em uma solução viável. A solução adotada foi utilizar um novo conceito definido no padrão CORBA 2.3: *valuetypes*. Com *valuetypes* foi possível implementar a transferência de maneira direta, apenas agregando as implementações do modelo de processo em suas respectivas figuras.

Outro ponto interessante de ser reportado a respeito da implementação da WPDTool é a dificuldade de se implementar uma interface complexa com a linguagem Java. Apesar da API do *Swing* ser muito customizável, a complexidade associada ao desenvolvimento de interfaces que utilizam alguns de seus recursos mais avançados é muito grande se comparado às ferramentas RAD (*Rapid Aplication Development*), como Delphi ou Visual Basic.

Entre as próximas tarefas a serem realizadas temos:

- Conclusão da implementação da WPDTool;
- Integração da ferramenta WPDTool à *Web*;

- Implementação da WAgenda integrada a *Web*;
- Realização de testes com a WPDTool e a WAgenda;
- Revisão da arquitetura do WPSEE, com base nas experiências realizadas, a fim de se garantir a viabilidade do projeto.

Grande parte do trabalho a ser realizado caracteriza-se como implementação de um sistema para a *Web* em Java, o know how para o desenvolvimento deste tipo de aplicativo já foi adquirido nos estudos realizados e em andamento, portanto acredita-se que o trabalho será finalizado em prazo hábil.

# Referências Bibliográficas


[1] [Alc98] Alcântara, Andreia A. Desenvolvimento de Sistemas de Informação Baseados na Web - Um Abordagem utilizando o Ciclo de Vida Espiral. Dissertação de Mestrado - DI - UFPE. 1998.

[2] [Ale77] Alexander, Christopher et al. *A Pattern Language: Towns, Buildings, Constructions*. Oxford University Press. USA. 1977.

[3] [Bar97] Barroca L. *Using Frameworks to Describe Software Architectures*. Dept. of Computing, The Open University. Inglaterra. 1997.

[4] [Be00] Be Inc. *BeOS - The Media OS*. Disponível em *http://www.be.com/products/freebeos/beoswhitepaper.html*. Acessado em 30/03/2000.

[5] [Ber96] Berners-Lee, Tim. *The World Wide Web: Past, Present and Future*. Acessível em *http://www.w3.org/People/Berners-Lee/1996/ppf.html*. Acessado em 27/02/2000.

[6] [Bes99a] Bessani, Alysson N. & Huzita, Elisa H.M. Utilização do Padrão CORBA como Mecanismo de Integração por Controle em Ambientes de Engenharia de Software. Relatório semestral de projeto de iniciação ciêntifica PIBIC-CNPq. Universidade Estadual de Maringá. Maringá. 1999.

[7] [Bes99b]Bessani, Alysson N. & Huzita, Elisa H.M. Utilização do Padrão CORBA como Mecanismo de Integração por Controle em Ambientes de Engenharia de Software. Relatório final de projeto de iniciação ciêntifica PIBIC-CNPq. Universidade Estadual de Maringá. Maringá. 1999.

[8] [Bus96] Buschmann, Frank et al. *A System of Patterns: Pattern - Oriented Software Architecture*. Wiley Compuer Publishing. Inglaterra. 1996.

[9] [Col94] Coulouris, G. et. al. *Distributed Systems: Concepts and Design*. Addison -Wesley. 1992.

[10] [Cop92] Coplien J.O. *Advanced C++ - Programing Styles and Idioms*. Addison-Weslley. 1992.

[11] [Fow97] Fowler, Martin & Scott, Kendal. *UML Destiled*. Addison-Wesley. USA. 1997.

[12] [Gam95] Gamma, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. 1995.

[13] [Gim94] Gimenes, I. M. S. Uma Introdução ao Processo de Engenharia de Software. XIII Jornada de Atualização em Informática. Caxambu - MG. 1994

[14] [Gim98] Gimenes, I. M. S. et Al. FORMLAB: Um ambiente Integrado de apoio aos métodos formais para o desenvolvimento de software. IDEAS 98. Torres -Brasil. 1998.

[15] [Gim99] Gimenes, I. M. S., Fantinato, M. e Carnielo A. ExPSEE: Um Ambiente Experimental de Engenharia de Software Orientado a Processo. Relatório final de projeto - CNPq. Universidade Estadual de Maringá. Maringá. 1999.

[16] [Gos99] Gosling, James et al. *The Java Language Specification*. Disponível em *http://java.sun.com/docs/books/jls/html/index.html*. Acessado em 22/03/2000.

[17] [Huz00] Huzita, E. H. M., Bessani, A. N. B. & Gimenes, I. M. S. *A Distributed Workflow System Based on Microkernel Architecture*. Artigo submetido ao PDPTA2000. 2000.

[18] [Ipl99] iPlanet - e-commerce solutions. *Client Side JavaScript Guide.* Acessível em *http://docs.iplanet.com/docs/manuals/js/client/jsguide/index.html*. Acessado em 26/03/2000.

[19] [Lin96] Lindholm, Tim & Yellin, Frank. *The Java Virtual Machine Specification*. Disponível em *http://java.sun.com/docs/books/vmspec/html/VMSpecTOC.doc.html*. Acessado em 22/03/2000.

[20] [Mel00] Melo, Wilson S. & Figueiredo, M.F. Desenvolvimento de Sistemas Neurais Dotados de Memória de Curta Duração e Capazes de Planejamento Aplicados a Navegação Autonôma de Robos. Relatório Semestral de Projeto - CNPq. Universidade Estadual de Maringá. Maringá. 2000.

[21] [Omg98] Object Management Group. CORBA Telecom Specification. Disponível em *http://www.omg.org*. 1998.

[22] [Omg99a] Object Management Group. The Common Object Request Broker Architecture And Specification, Version 2.3. Disponível em *http://www.omg.org*. 1999.

[23] [Omg99b] Object Management Group. CORBA Services Specification. Disponível em *http://www.omg.org*. 1999.

[24] [Omg99c] Object Management Group. CORBA Med Specification. Disponível em *http://www.omg.org.* 1999.

[25] [Orf96] Orfali, R. et al. *The Essetial Distributed Objects Survival Guide*. Wiley Compuer Publishing. 1996.

[26] [Pre96] Pressman, Roger S. Software Engineering: A Practitioners Approach. MacGraw Hill Colege Div. USA. 1996.

[27] [Sch97] Schmitt D. Software Architecture, Frameworks and Patterns. PhD Progress Report. The Open University. 1998.

[28] [Sch00] Schmidt, Douglas. *Overview of CORBA*. Disponível em *http://www.cs.wustl.edu/ ∼shmidt/corba-overview.html*. Acessado em 26/03/2000.

[29] [Sha96] Shaw, M. & Garlan, *D. Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall. 1996.

[30] [Sun98] Sun Microsystems. *The Java Remote Method Invocation Specification*. Disponível em *ftp://ftp.javasoft.com/docs/jdk1.2/rmi-spec-JDK1.2.ps*. Acessado em 25/03/2000.

[31] [Sun99a] Sun Microsystems. *The Java Tutorial*. Disponível em *http://web2.java.sun.com/docs/books/tutorial/*. Acessado em 22/02/2000.

[32] [Sun99b] Sun Microsystems. *The Jini Architectural Overview*. Disponível em *http://www.sun.com/jini/*. 1999.

[33] [Sun99c] Sun Microsystems. *Programer s Guide to the Java 2D API - Enhanced Graphics and Imaging for Java*. Disponível em *http://java.sun.com/products/jdk/1.2/docs/guide/2d/spec/j2d-title.fm.html*. Acessado em 27/03/2000.

[34] [Sun99d] Sun Microsystems. *The Java Remote Method Invocation (RMI)*. Disponível em *http://java.sun.com/products/jdk/1.2/docs/guide/rmi/index.html*. Acessado em 20/03/2000.

[35] [Tan95] Tanenbaum, A. S. *Distributed Operating Systems*. Prentice Hall. 1995.

[36] [Tan96] Tanembaum, A. S. *Computer Networks*. Prentice Hall. 1996.

[37] [Tan99] Tanaka, S. A. & Gimenes I. M. Um *Framework* para Desenvolvimento de Gerenciadores de Workflow. TI 857 CPGCC-UFRGS. Universidade Federal do Rio Grande do Sul. Porto Alegre. 1999.

[38] [Yan99] Yanaga E. & Gimenes I. M. S. Implementação do Protóripo do Módulo de Simulação de Processos (P/pSim) de uma Ferramenta para Coleta e Avaliação de Métricas Baseadas em Modelos para Processos de Software (P/pTool).Relatório Final de Projeto - CNPq. Universidade Estadual de Maringá. Maringá 1998.

[39] [Wer94] Werner, C. M. L. et al. O Uso de Frameworks como Arquiteturas Reutilizáveis na Construção de Ambientes de Desenvolvimento de Software. VIII Simpósio Brasileiro de Engenharia de Software. Curitiba, Brasil. 1994.

[40] [Wfm95] Workflow Management Coalition. *The Workflow Reference Model*. Disponível em *http://www.wfmc.org* . 1995.

# Apêndice A

Definição em IDL do modelo de processo do WPSEE.

```
/*
                        ProcessData.idl

Autor: Alysson Neves Bessani
Data: 28/02/2000
Versao: beta1
Obs: Dados sobre processos e arquiteturas de processos
*/

#include "Util.idl"

module WPSEE{
module ProcessData{

 /*
  * Classe base de todos os elementos do processo
  */
 valuetype ProcessElement{
   //identificador unico do objeto
   private unsigned long id;
   //nome do elemento
   public string name;
   //versao do elemento
   public string version;
   //descricao do elemento do processo
   public string description;
   //retorna o id do processo
   unsigned long getId();
 };
```

```
typedef sequence<ProcessElement> ProcessElementList;

/*
 * Classe basica para todos os objetos desenhados
 */
valuetype Drawnable: ProcessElement{
  //Posicao que o objeto esta sendo desenhado
  public WPSEE::Point position;
};

typedef sequence<Drawnable> DrawnableList;

/*
 * Metodos que todos os tipos devem implementar
 */
interface Type{
  //Instanciacao de um objeto
  ProcessElement instantiate(in unsigned long flag);
};

/*
 * Cargo de um ator
 */
valuetype Role : ProcessElement supports Type{};

typedef sequence<Role> RoleList;

/*
 * Tipo de ferramenta
 */
valuetype ToolType: ProcessElement supports Type{};

typedef sequence<ToolType> ToolTypeList;

/*
 * Tipo de artefato
 */
valuetype ArtifactType: Drawnable supports Type{
  public Drawnable producedBy;
```

```
  public Drawnable neededBy;
};

typedef sequence<ArtifactType> ArtifactTypeList;

/*
 * Tipo de tarefa
 */
valuetype TaskType: Drawnable supports Type{
  public string preCondition;
  public string posCondition;
  public RoleList executerRole;
  public ToolTypeList toolsTypes;
  public ArtifactTypeList requiredArtifactsTypes;
  public ArtifactTypeList producedArtifactsTypes;
};

/*
 * Arquitetura de processo
 */
valuetype ProcessArchitecture: ProcessElement supports Type{
  public string author;
  public string classification;
  public string documentation;
  public DrawnableList elements;
};

/*
 * Tipo de sub-processo
 */
valuetype SubProcessType: Drawnable supports Type{
  public ProcessArchitecture subProcess;
  public boolean isLoop;
  public string preCondition;
  public string posCondition;
  public ArtifactTypeList requiredArtifactsTypes;
  public ArtifactTypeList producedArtifactsTypes;
};
```

```
/*
 * Artefato
 */
valuetype Artifact: Drawnable{
  public Drawnable producedBy;
  public Drawnable neededBy;
};

typedef sequence<Artifact> ArtifactList;

/*
 * Ferramenta que ajuda na execucao do processo
 */
valuetype Tool: ProcessElement{
  //nome do fabricante da ferramenta
  public string vendor;
  //comando de invocacao
  public string invocationCommand;
  //parametros a serem passados para a ferramenta
  public string parameters;
  //numero de licensas, eh util para se saber quantas tarefas podem usar a
  //mesma ferramenta simultaneamente
  public unsigned short numberOfLicenses;
  //plataforma em que a ferramenta pode ser usada
  public WPSEE::Plataform plataform;
};

typedef sequence<Tool> ToolList;

/*
 * Ator
 */
valuetype Actor: ProcessElement{
  public RoleList roles;
};

typedef sequence<Actor> ActorList;

/*
```

```
 * Ator humano
 */
valuetype Human: Actor{
  public string password;
  public string email;
  public unsigned short totalFreeHours;
  public unsigned short actualFreeHours;
};

/*
 * Ator automatico, Agente de software ou robo
 */
valuetype Agent: Actor{
  public string homeHost;
  public string initCommand;
};

/*
 * Grupo de atores
 */
valuetype Group: Actor{
  private Actor leader;
  private ActorList members;
};

/*
 * Tarefa
 */
valuetype Task: Drawnable{
  public string preConditions;
  public string posConditions;
  public WPSEE::TaskState state;
  public ActorList executers;
  public ToolList tools;
  public ArtifactList requiredArtifacts;
  public ArtifactList producedArtifacts;
};

/*
```

```
 * Processo
 */
valuetype Process: ProcessArchitecture{
  public Actor responsible;
  public WPSEE::ProcessStatus status;
  public WPSEE::Date validFrom;
  public WPSEE::Date validTo;
  public unsigned short priority;
  public unsigned long timeLimit;
};

/*
 * Sub Processo
 */
valuetype SubProcess: Drawnable{
  //processo de software que sera usado como sub-processo
  public Process subProc;
  //indica se o sub processo eh ou nao um loop
  public boolean loop;
  //pre condicoes para a execucao do processo
  public string preCondition;
  //pos condicoes para a execucao do processo
  public string posCondition;
  //lista de artefatos de entrada para o sub-processo
  public ArtifactList requiredArtifacts;
  //lista de artefatos produzidos pelo sub-processo
  public ArtifactList producedArtifacts;
};

/*
 * Objetos do processo que sao transparentes na execucao e tem a mesma
 * semantica tanto na arquitetura do processo, quanto no processo em si
 */
interface Transparent{
  //verifica se o objeto esta sendo executado como processo, ou a nivel de
  //arquitetura
  boolean isProcess();
  //pega a lista com os proximos artefatos produzidos no processo, se existir
  boolean getNext(out ProcessElementList atl);
```

```
};

/*
 * Inicio do processo, pode ou nao ter um artefatoque o leva a(s) tarefa(s)
 * inicial(ais)
 */
valuetype Start: Drawnable supports Transparent{
  //artefatos que dao inicio ao processo, normalmente eh algum tipo de
  //introducao ou requisicao ao processo a ser executado. Este artefato serve
  //de ponto de partida para a primeira tarefa
  public ProcessElementList initialArtifacts;
};

/*
 * Fim do processo, o artefato que leva a este objeto normalmente eh o produto
 * final do processo, ou produtos finais do processo
 */
valuetype Finish: Drawnable supports Transparent{
  //Lista com os artefatos cuja a producao leva ao fim do processo, note que
  //para que o processo termine eh necessario que todos estes artefatos
  //existam
  public ProcessElementList finalArtifacts;
};

/*
 * Divide o fluxo de dados, 1 artefato de entrada transforma-se em N
 * artefatos que sao distribuidos para outros objetos do processo
 */
valuetype Split: Drawnable supports Transparent{
  //artefato de entrada no split
  public ProcessElement enterArtifact;
  //artefatos copias do artefato de entrada, que serao distribuidos para os N
  //outros objetos do processo
  public ProcessElementList exitArtifacts;
};

/*
 * Sincroniza o fluxo de dados, N artefatos entram e a partir do momento que
 * todos estao prontos eh criado um novo artefato que representa a composicao
```

```
   * destes.
   */
  valuetype Join: Drawnable supports Transparent{
    //artefatos de entrada
    public ProcessElementList enterArtifacts;
    //artefato de saida, composto de todos os outros artefatos de entrada
    public ProcessElement exitArtifact;
  };
};
};
```