



UNIVERSIDADE FEDERAL DE PERNAMBUCO

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CENTRO DE INFORMÁTICA

ANÁLISE DE TÉCNICAS DE ENGENHARIA DE SOFTWARE PARA A INTEGRAÇÃO E PADRONIZAÇÃO DE PROJETOS DE SOFTWARE

Autor

Albérico de Lima Pena Júnior

Orientador

Prof. Alexandre Marcos Lins de Vasconcelos

Co-orientador

Sandro Ronaldo Bezerra Oliveira

Recife, Fevereiro 2006

UNIVERSIDADE FEDERAL DE PERNAMBUCO

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

2005.2

ALBÉRICO DE LIMA PENA JÚNIOR

ANÁLISE DE TÉCNICAS DE ENGENHARIA DE SOFTWARE PARA A INTEGRAÇÃO E PADRONIZAÇÃO DE PROJETOS DE SOFTWARE

ESTE TRABALHO FOI APRESENTADO À GRADUAÇÃO EM CIÊNCIA DA
COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE
FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO
DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.
ORIENTADOR: PROF. ALEXANDRE MARCOS LINS DE VASCONCELOS
CO-ORIENTADOR: SANDRO RONALDO BEZERRA OLIVEIRA

Recife, fevereiro 2006

A

AGRADECIMENTOS

Aos meus pais, Albérico e Rose, à minha irmã, Catarina por todos esses anos de carinho, atenção e apoio.

Agradeço a Daniela por todo o apoio, compreensão e carinho.

A todos os professores que me fizeram chegar até a Universidade Federal de Pernambuco.

Aos professores da Universidade Federal de Pernambuco por me ensinarem tanta coisa até me transformar em um Bacharel em Ciência da Computação.

Ao aluno de doutorando Sandro Bezerra pela amizade conquistada e pela co-orientação deste trabalho. Despendendo várias horas para ajudar no que fosse necessário.

Ao professor Alexandre Vasconcelos, pelas aulas ministradas durante o curso, pela orientação e validação do meu trabalho.

A todos os meus amigos que tanto me ajudaram neste trabalho e fizeram da minha vida acadêmica mais alegre.

Albérico de Lima Pena Júnior

SUMÁRIO

INTRODUÇÃO	7
SISTEMAS DE SOFTWARE: VISÃO GERAL	9
2.1 A Importância do Software	9
2.1.1 O Papel Evolutivo do Software	9
2.2 Software	11
2.2.1 Uma Crise no Horizonte	11
2.2.2 Projetos de Software	11
2.3 Padronização de Software	12
2.3.1 Categorias de padrões de software	13
2.3.2 Características	14
2.4 Integração de Software	16
2.4.1 Categorias e Níveis de Integração	17
2.5 Considerações Finais	23
AMBIENTE DE IMPLEMENTAÇÃO DE PROCESSO DE SOFTWARE	24
3.1 Processo de Software	24
3.2 PSEE - Ambientes de Desenvolvimento de Software Centrado no Processo	27
3.3 ImPProS – Ambiente de Implementação Progressiva de Processo de Software	31
3.4 Considerações Finais	33
UM MODELO DE PADRONIZAÇÃO E INTEGRAÇÃO DE FERRAMENTAS NO CONTEXTO DO IMPPROS	34
4.1 Contexto do ImPProS	34
4.2 Sugestão de um Modelo de Padronização	35
4.2.1 Arquitetura do Projeto	35
4.2.2 Especificação dos Requisitos	39
4.2.3 Padrão de Interfaces Gráficas	42
4.2.4 Modelagem dos Dados	43
4.2.5 Padrão de Codificação	44
4.2.6 Padrão de Diretórios	54
4.3 Integração no ImPProS	55
4.3.1 Modelo de Integração do ImPProS	56
4.3.2 Mecanismos para Integração no ImPProS	60
4.3.3 Integração do ImPProS e suas Ferramentas de Suporte ao ProKnowledge	63
4.4 Considerações Finais	65
VALIDAÇÃO DO MODELO	66
5.1 Ferramenta ProKnowledge: Visão Geral	66
5.2 Aplicação dos Modelos ao Proknowledge	67
5.2.1 Arquitetura de Projeto	68
5.2.2 Especificação de Requisitos	69
5.2.3 Padrão de Interfaces Gráficas	71
5.2.4 Modelagem de Dados	73
5.2.5 Padrões de Codificação	74
5.2.6 Padrão de Diretórios	81
5.3 Considerações Finais	81
TRABALHOS FUTUROS	83
BIBLIOGRAFIA	84

ÍNDICE DE FIGURAS

Figura 1. A evolução do software.	10
Figura 2. Propriedades das Categorias de Integração.....	18
Figura 3. Níveis de Integração de Ferramentas [Reis00].....	20
Figura 4. Arquitetura do Ambiente ImPProS	32
Figura 5. Arquitetura de Implementação do ImPProS.....	37
Figura 6. Janela de Manutenção do Usuário.....	43
Figura 7. Estruturação dos Diretórios do ImPProS e das Ferramentas de Suporte.....	54
Figura 8. Nível de Integração do ImPProS.....	57
Figura 9. Estrutura de Integração do ImPProS com base no PCTE	59
Figura 10. Funcionamento do Kernel do ImPProS.....	59
Figura 11. Estrutura do Arquivo de Configuração do ImPProS.....	61
Figura 13. Link com o ProKnowledge.....	64
Figura 14. Fluxo de Execução da Integração com o ProKnowledge	64
Figura 15. Estrutura do arquivo XML.....	65
Figura 16. Mapeamento entre uma pilha de objetos do ProKnowledge e a Arquitetura.....	68
Figura 17. Tela de login do sistema.....	71
Figura 18. Tela de Manutenção de Usuários	72
Figura 19. Manutenção de Conhecimento	72
Figura 20. Estruturação dos Diretórios do ProKnowledge e das Ferramentas de Suporte	81

ÍNDICE DE TABELAS

Tabela 1. Capacidades de integração de dados, controle e apresentação em cada nível.....	22
Tabela 2. Declarações dos componentes de uma classe.....	48
Tabela 3. Declarações das interfaces	49
Tabela 4. Métodos Padrões das Interfaces Gráficas	53
Tabela 5. Nome dos componentes de interface gráfica	54
Tabela 6. Descrição da Tabela Contexto do Proknowledge	73
Tabela 7. Descrição da Tabela Informação do Proknowledge.....	73

INTRODUÇÃO

Contexto do Trabalho

Devido à globalização, os projetos estão cada vez mais descentralizados. E com o aumento do investimento em tecnologia, os projetos estão cada vez maiores. Logo, é necessário analisar as diversas técnicas de engenharia de software para garantir a padronização e integração dos projetos.

Dentro deste contexto, um projeto de pesquisa oriundo de uma proposta de Doutorado do programa de pós-graduação do CIn/UFPE, cujo objetivo é o desenvolvimento do ImPProS – Ambiente de Implementação Progressiva de Processo de Software, concebeu o desenvolvimento de alguns módulos funcionais, os quais foram implementados separadamente por alunos de graduação. No entanto, a implementação trazia vários problemas de padronização de arquitetura e modelos, entre outros. Este trabalho de graduação vem, com base neste contexto, propor técnicas de engenharia de software para a padronização de integração destes módulos.

Motivação

Com a crescente demanda por projetos de grande porte, é comum ter equipes trabalhando em módulos diferentes em lugares diferentes. É importante o estudo de técnicas de engenharia de software que garantam a homogeneidade (padronização) dos módulos ao final do projeto. Além disso, os módulos precisam “conversar”, ou seja, os módulos precisam ter algum mecanismo que possam trocar e entender informações.

O estudo de técnicas de engenharia de software que garantam a padronização e integração de projetos de software está cada vez mais em

evidência. As grandes empresas usam cada vez mais técnicas que possam garantir o sucesso dos grandes projetos.

Metodologia

O desenvolvimento deste trabalho deu-se com a análise e aplicação de conceitos de engenharia de software, de conhecimento ao longo do curso de graduação, para a definição de um modelo que proveja a padronização e integração de sistemas, no contexto do ImPProS.

Foi estudada a especificação dos sistemas do ImPProS que serviram como aplicação do modelo, para que as informações pudessem ser processadas de acordo com a integração das funcionalidades ora propostas.

Estrutura do Trabalho

Além deste capítulo introdutório, este trabalho está dividido nos seguintes capítulos:

Capítulo 2 – Fala sobre projetos de software, Padronização e Integração, trazendo uma visão geral dos sistemas de software, passando pela sua história, trazendo fatos interessantes de várias épocas. Retrata a sua importância dentro da sociedade, toda a sua evolução.

Capítulo 3 – Este capítulo é dedicado ao contexto na qual o projeto está inserido. O ImPProS. É apresentada a base deste ambiente como também todas as suas características.

Capítulo 4 – Neste capítulo será definido a proposta de modelo de padronização e integração para sistemas no contexto do ImPProS. São discutidas boas práticas, modelagem de dados, arquitetura, padrões e uma estrutura de integração.

Capítulo 5 – Este capítulo apresenta evidências da aplicação do modelo proposto dentro do ambiente ImPProS, mas precisamente na ferramenta ProKnowledge.

SISTEMAS DE SOFTWARE: VISÃO GERAL

No início da década de 1980 era clara a preocupação em relação ao software. A revista *Business Week* estampou em sua primeira página a seguinte manchete: “Software: A Nova Força Propulsora”. Em meados da década de 1980, a revista *Fortune* lamentava “Uma Crescente Defasagem de Software”, e ao final da década a *Business Week* advertia os gerentes sobre a “Armadilha do Software – Automatizar ou Não”. Já no início da década de 1990, a *Newsweek* perguntava: “Podemos Confiar em Nosso Software?”, enquanto o *Wall Street Journal* comentava as dificuldades de uma grande empresa com um artigo de primeira página, intitulado “Criar Software Novo: Era uma Tarefa Agonizante...”. Manchetes como essas, e várias outras que eram vinculadas, eram o anúncio da importância que sendo atribuída ao software de computador – as oportunidades oferecidas por ele e os perigos que apresenta.

2.1 A Importância do Software

Durante as três primeiras décadas da era do computador, o principal desafio era reduzir o custo de processamento e armazenagem de dados. Durante a década de 1980 foram desenvolvidos avanços na microeletrônica que resultaram em maior poder de computação a um custo cada vez mais baixo. Atualmente, o grande problema é diferente. O principal desafio durante as décadas de 1990 e 2000 é melhorar a qualidade (e reduzir o custo) de soluções baseadas em computador.

2.1.1 O Papel Evolutivo do Software

Em livros populares sobre a “revolução do computador”, Osbourne [OSB79] caracterizou uma “nova revolução industrial”, Toffler [TOF80] chamou o advento de microeletrônica de “a terceira onda de mudança” na história humana e Naisbitt [NAI82] previu que a transformação de uma sociedade industrial em uma

“sociedade de informação” terá um profundo impacto sobre nossas vidas. Feigenbaum e McCorduck [FEI83] sugeriram que a informação e o conhecimento (controlados por computador) serão o foco principal do poder no século XXI, e Stoll [STO89] argumentou que a “comunidade eletrônica” criada por redes e software é a chave de troca de conhecimento em todo o mundo. No início da década de 1990, Toffler [TOF90] descreveu uma “mudança de poder”, em que as velhas estruturas de poder (governamental, educacional, industrial, econômico e militar) se desintegrarão enquanto os computadores e o software levarão a uma “democratização do conhecimento”.

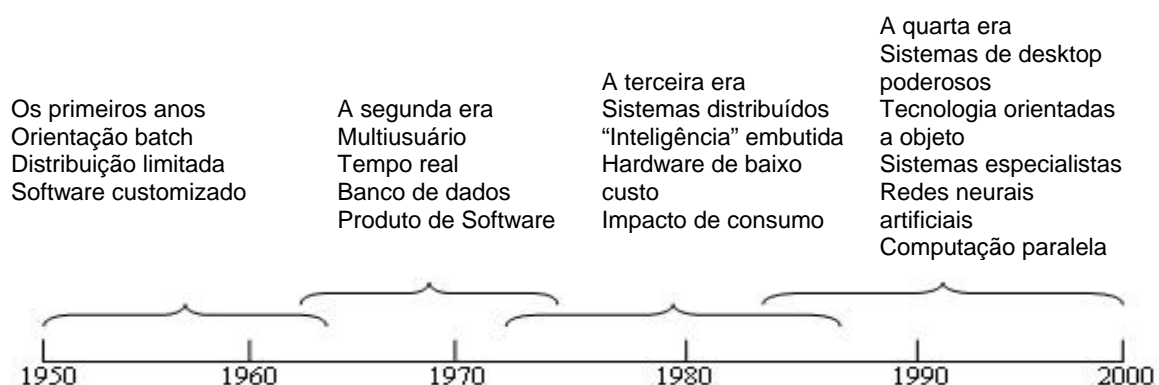


Figura 1. A evolução do software.

A caminho da quinta era, os problemas associados ao software continuam a se intensificar:

- A sofisticação do software ultrapassou nossa capacidade de construir um sistema que extraia o potencial do hardware;
- Nossa capacidade de construir programas não pode acompanhar o ritmo da demanda de novos programas;
- Nossa capacidade de manter programas existentes é ameaçada por projetos ruins e recursos inadequados.

Em resposta a esses problemas, estão sendo adotadas práticas e técnicas de engenharia de software.

2.2 Software

“Software são instruções (programas de computador) que, quando executadas, produzem a função e o desempenho desejado; estruturas de dados que possibilitam que os programas manipulem adequadamente a informação; e documentos que descrevem a operação e o uso dos programas” [PRE95].

2.2.1 Uma Crise no Horizonte

Muitos observadores da indústria de software caracterizavam os problemas associados ao desenvolvimento de software como uma “crise”. Contudo, o que temos pode ser algo bem diferente.

A palavra “crise” é definida como um ponto decisivo no curso de algo. Contudo, para o software, não tem havido nenhum “ponto decisivo”; somente uma mudança lenta e evolucionária. Na indústria de software, temos tido uma “crise” que nos acompanha há quase 40 anos.

Ainda temos que chegar ao estágio de crise no software de computador. O que se tem é uma aflição crônica. A palavra “aflição” é definida como “algo que causa dor ou sofrimento”. E o adjetivo “crônica”: “que dura um longo tempo ou retorna freqüentemente; que continua indefinidamente”. Então, é mais preciso afirmar que estamos enfrentando uma aflição crônica e não uma crise durante as três últimas décadas. Não há cura milagrosa, mas há muitas maneiras pelas quais podemos reduzir a dor enquanto lutamos para descobrir a cura.

2.2.2 Projetos de Software

As organizações executam trabalho. O trabalho envolve serviços continuados e/ou projetos, embora possa haver superposição entre os dois. Serviços continuados e projetos possuem muitas características comuns; por exemplo, ambos são:

- Executados por pessoas;
- Restringidos por recursos limitados;
- Planejados, executados e controlados.

Serviços continuados e projetos diferem principalmente porque enquanto os primeiros são contínuos e repetitivos, os projetos são temporários e únicos. Segundo o *Project Management Institute*, um projeto pode ser definido em termos de suas características distintas – *um projeto é um empreendimento temporário com o objetivo de criar um produto ou serviço único*. *Temporário* significa que cada projeto tem um começo e um fim bem definidos. *Único* significa que o produto ou serviço produzido é de alguma forma diferente de todos os outros produtos ou serviços semelhantes. Os projetos são desenvolvidos em todos os níveis da organização. Eles podem envolver uma única pessoa ou milhares delas. Podem requerer poucas horas de trabalho ou até milhares delas para se completarem. Os projetos podem atravessar as fronteiras organizacionais, como ocorre com consórcios e parcerias, ou envolver uma unidade isolada da organização. Os projetos são freqüentemente componentes críticos da estratégia de negócios da organização.

Pode-se citar como exemplos de projetos:

- Desenvolver um novo produto ou serviço;
- Implementar uma mudança organizacional em nível de estrutura, de pessoas ou de estilo gerencial;
- Planejar um novo veículo de transporte;
- Desenvolver ou adquirir um sistema de informação novo ou modificado;
- Construir um prédio ou instalações;
- Implementar um novo processo ou procedimento organizacional;

No contexto de software, *projeto* é usado com o sentido do ato de projetar um sistema ou serviço.

2.3 Padronização de Software

O ser humano convive com a padronização há milhares de anos e depende dela para a sua sobrevivência, mesmo que não tenha consciência disto.

Imagine como seriam as relações comerciais entre as nações se não existisse o Sistema Métrico para estabelecer uma linguagem comum? Ou então, como seria possível manter a ordem pública sem os sinais de trânsito?

No início da era automotiva, Henry Ford declarou que os norte-americanos poderiam ter seus carros na cor que desejassem – desde que fossem pretos. Apenas parte da declaração era ironia. Por trás dela estava um princípio básico da revolução na produção em massa que a manufatura de automóveis ajudou a inaugurar: o sucesso desse modelo depende do uso de componentes padrões que vêm em número limitado de estilos.

Os primeiros conceitos de padrões e linguagens de padrões tiveram origem no trabalho do arquiteto Christopher Alexander, que desenvolveu um conjunto de teorias sobre padrões para arquitetura. Foi a partir de suas idéias que o conceito de padrões passou a ser utilizado na área de desenvolvimento de software. Em [ALE77] Alexander apresenta uma definição para padrão: “Cada padrão descreve um problema que ocorre repetidamente no nosso ambiente, e então descreve a essência de uma solução para este problema, de forma que pode-se usar esta solução milhares de vezes, sem fazê-lo da mesma forma duas vezes”. Padrões captam a experiência comprovada no desenvolvimento de software. Um padrão refere-se a um problema de software recorrente que surge em uma situação de projeto específica, e apresenta um esquema genérico e comprovado para sua solução. Um desenvolvedor de software inexperiente pode aproveitar padrões criados por especialistas para resolver os problemas com os quais se depara, gerando um software de melhor qualidade.

A criação de padrões, além dos benefícios já mencionados, documenta a experiência existente e comprovada, e provê um vocabulário comum para os desenvolvedores de software.

2.3.1 Categorias de padrões de software

A utilização de padrões pode ser feita durante diversas etapas do desenvolvimento de um software. Para cada etapa há uma categoria diferente de padrões. As principais categorias de padrões verificadas na literatura são padrões de arquitetura, padrões de projeto e idiomas, todos estes descritos por Buschmann em [BUS 96], e padrões de análise, descritos por Fowler em [FOW 97].

Um padrão de arquitetura representa um modelo de estrutura básica para sistemas de software. Ele provê um conjunto de subsistemas pré-definidos,

especifica suas responsabilidades e inclui regras para organizar os relacionamentos entre eles. Padrões de arquitetura são modelos para arquiteturas de software concretas [BUS 96].

Um padrão de projeto provê um esquema para refinar os subsistemas ou componentes de um sistema de software, ou os relacionamentos entre eles. O padrão descreve uma estrutura de componentes relacionados que soluciona um problema de projeto em um contexto particular [BUS96]. Padrões de projeto estão mais próximos da implementação do que os padrões de arquitetura e padrões de análise, pois focam, principalmente, nos aspectos típicos de projeto, como por exemplo, interfaces homem-máquina, criação de objetos e propriedades estruturais básicas. Padrões de projeto podem ser usados em um número maior de aplicações do que outras categorias de padrões. Esta é a categoria de padrões mais encontrada na literatura. Vários livros descrevem padrões de projeto [BUS 96, GAM 94], em especial padrões orientados a objeto.

Idioma é um padrão baixo-nível, específico para uma linguagem de programação. Descreve como implementar aspectos particulares de componentes ou os relacionamentos entre eles usando características de determinada linguagem. Idiomas se referem a aspectos de projeto e implementação [BUS 96].

Um padrão de análise é uma parte de uma especificação de requisitos ou modelagem conceitual de dados que se origina em um projeto e pode ser reutilizada em diversos outros projetos. Um padrão de análise descreve um conjunto de objetos do mundo real, seus relacionamentos e as regras que definem seu comportamento. Para isto utiliza-se um modelo semântico de alto nível. Padrões de análise são dependentes da aplicação, pois sua semântica descreve aspectos específicos de algum domínio ou aplicação.

Além destas categorias de padrões, existem outras, tais como padrões de suporte [FOW 97], padrões organizacionais, padrões para planejamento de projeto [APP 00] etc.

2.3.2 Características

Considerando as definições apresentadas anteriormente, um padrão é uma entidade que documenta uma solução, um problema recorrente e o contexto em

que se deve aplicar tal solução. Para esta monografia, consideraremos esse conceito de padrões.

É importante salientar que o objetivo da comunidade de padrões de *software* é documentar e compartilhar soluções comprovadas de engenheiros de *software* experientes para problemas recorrentes em um determinado contexto, criando assim uma literatura que auxilie na adoção das boas práticas para o desenvolvimento de sistemas. Neste contexto, Vlissides [90] cita quatro benefícios importantes no reuso de padrões de *software*:

- Capturar experiências tornando-as acessíveis aos não experientes;
- Formar um vocabulário a fim de ajudar os desenvolvedores a se comunicarem melhor;
- Ajudar engenheiros de *software* a entender um sistema mais rapidamente quando ele está documentado com os padrões reutilizados;
- Facilitar a reestruturação de um sistema, tendo ele sido ou não projetado com padrões em mente.

Além dos benefícios citados, existem outras vantagens no reuso de padrões de projeto para o desenvolvimento de sistemas, como listada a seguir:

- Evolução de código;
- Modularidade;
- Desacoplamento entre áreas de responsabilidades de forma que as mudanças em uma área não ocasionem mudanças nas outras;
- Diminuição da complexidade do projeto e do código final;
- Facilidade na criação de *frameworks* contendo padrões de projeto já implementados a fim de facilitar o reuso dos padrões posteriormente;
- Estabilidade do código;
- Confiabilidade no reuso de padrões cujas soluções são comprovadas;
- Ganho de produtividade;
- Facilidade de repassar conhecimento entre os desenvolvedores experientes;
- Facilidade de aprendizado de novas áreas de conhecimento para uma equipe sem experiência na aplicação a ser desenvolvida.

O desenvolvimento de um software pode ser considerado um trabalho artesanal. Cada profissional soluciona os problemas que encontra usando seu conhecimento e criatividade. Com o uso de padrões isto sofre alterações, pois boas soluções passam a ser documentadas, transmitidas e adotadas por outros profissionais. Como consequência, é intensificada a criação de uma linguagem comum para a comunicação e troca de experiências entre os profissionais.

Por outro lado, a documentação do desenvolvimento do software, que é uma tarefa muitas vezes relegada a uma posição secundária, ganha mais destaque, alterando a cultura vigente.

A utilização de padrões também é uma forma eficaz de difusão do conhecimento, pois através do estudo de padrões pode-se aprender, com a experiência de especialistas do mundo inteiro, boas práticas de desenvolvimento de software.

2.4 Integração de Software

O desenvolvimento da área de ADS - Ambientes de Desenvolvimento de Software - na década de 80 estimulou o surgimento de diversos ambientes e teorias para melhorar o apoio à engenharia de software. Com o objetivo de aumentar o nível de integração e portabilidade das ferramentas, surgiram padrões de infra-estruturas (ou *frameworks*) de serviços comuns para ADSs. Porém, o entendimento e comparação dos novos ambientes e dos *frameworks* foi dificultado devido à diversidade de representação dos seus serviços e componentes.

Ambientes de desenvolvimento de software são compostos de ferramentas. Estas ferramentas deverão compartilhar informações sobre os vários projetos que os usuários necessitarão utilizar em uma ferramenta, informações geradas por outra. Portanto, o ADS deve prover uma estrutura unificadora que integre suas ferramentas.

O conceito de integração pode ser entendido de diferentes formas e possui várias classificações. De acordo com [Reis00], a integração de ferramentas CASE pode ser definida como uma propriedade dos relacionamentos entre as ferramentas que formam um ambiente e refere-se ao grau de “acordo” que existe entre essas ferramentas.

A integração é necessária para obter produtividade no desenvolvimento e aumentar a qualidade dos produtos de software, mas os custos associados à construção de ambientes integrados devem ser levados em consideração. A integração pode aumentar a consistência dos dados e reduzir a redundância através do compartilhamento de informações.

2.4.1 Categorias e Níveis de Integração

Apesar de existirem várias classificações de integração entre ferramentas, é visto que em geral as ferramentas integradas devem exibir a mesma aparência, compartilhar informações e invocar umas as outras quando apropriado. Essas capacidades correspondem às categorias gerais de integração de **dados**, **controle** e **apresentação**. As categorias de integração descrevem as áreas nas quais as ferramentas podem cooperar. Os níveis de integração para as ferramentas descrevem o quanto cada ferramenta coopera dentro de uma categoria de integração específica.

Segundo [Jorgensen94], as categorias de integração mais comumente encontradas na literatura são: dados, apresentação, controle e processo. A Figura 2. mostra as categorias de integração de ferramentas e suas propriedades.

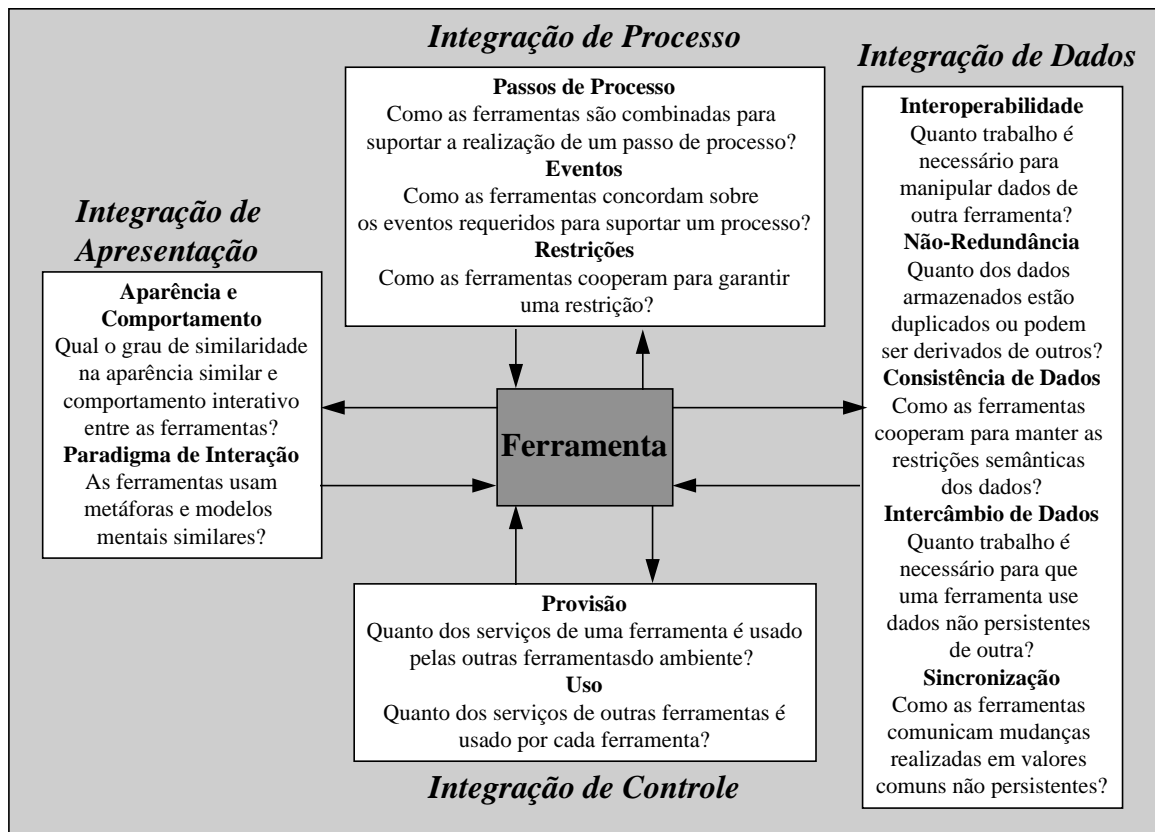


Figura 2. Propriedades das Categorias de Integração

As categorias descritas por [Jorgensen94] são descritas a seguir:

- Integração de Apresentação:** Esta categoria impõe consistência nas interfaces gráficas das ferramentas. Cada ferramenta possui o mesmo conjunto de construtores na interface com o usuário, ou seja, os usuários interagem com todas as ferramentas do ambiente da mesma forma. Isso reduz a necessidade do usuário aprender novos comandos para uma ferramenta recém integrada e permite que ele se concentre apenas na funcionalidade específica das ferramentas que ele utiliza. Para obter a integração de apresentação as ferramentas podem compartilhar a mesma biblioteca de interface com o usuário, por exemplo o padrão X Windows ou Motif ou Swing;

- **Integração de Dados:** As ferramentas compartilham informações através de um mesmo formato de dados. Os usuários podem trabalhar com o mesmo item de dados utilizando múltiplas ferramentas. Os métodos para prover integração de dados são: transferência direta de informação entre duas ferramentas (*pipes*), transferência baseada em arquivo (as ferramentas compartilham um arquivo), transferência baseada em comunicação (apropriada para sistemas abertos e ambientes distribuídos), e transferência baseada em repositório compartilhado (com serviços de armazenamento e gerência de objetos, controle de versões, configurações, segurança e transações). O repositório é o componente central da abordagem de integração de dados. O compartilhamento de dados é obtido através de esquemas comuns que definem a estrutura e comportamento dos dados;

- **Integração de Controle:** As ferramentas devem ser capazes de notificar eventos para outras ferramentas, ativar outras ferramentas e compartilhar funções de outra ferramenta, ou seja, exercer influência sobre outras. Os mecanismos de integração de controle incluem passagem explícita de mensagens, triggers ativados por tempo e por acesso e servidores de mensagens. Para obter integração de controle as ferramentas utilizam serviços de mensagem para prover três tipos de comunicação: ferramenta-ferramenta, ferramenta-serviço e serviço-serviço. Uma alta integração de controle pode aumentar o grau de automação do processo de desenvolvimento de software no ambiente;

- **Integração de Processo:** As ferramentas do ambiente devem ser utilizadas de acordo com um processo de desenvolvimento descrito formalmente através de uma linguagem de modelagem de processo e executado através de uma máquina de execução do processo. Esta categoria de integração é encontrada em ADS orientados a processos e será abordada com mais detalhes no próximo capítulo.

Já os níveis de integração são a forma que a integração deve acontecer. Por exemplo: carregador, léxico, sintático, semântico e de método [Reis00]. Esses

níveis não são estritamente hierárquicos, apenas progridem de baixo grau de integração a alto grau de integração.

Os fatores cruciais na avaliação do nível de integração entre ferramentas em um ADS são as formas de armazenamento, compartilhamento e transferência de informações entre as ferramentas. Os níveis são mostrados na Figura 2.3 e apresentados em seguida.

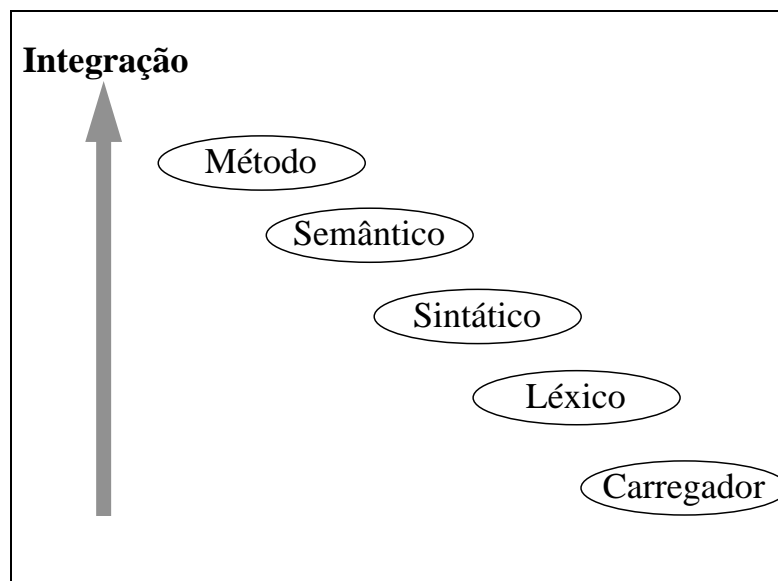


Figura 3. Níveis de Integração de Ferramentas [Reis00]

- **Nível Carregador:** Neste nível, cada ferramenta analisa seus dados de entrada e saída, pois não há entendimento comum sobre os dados que as ferramentas podem tratar. Um exemplo é o ambiente UNIX onde a integração é provida através da composição de ferramentas em um formato de arquivos simples e consistente. Cada ferramenta é responsável pelos seus dados e o compartilhamento de informações se dá pelo fato de todas as entradas e saídas de ferramentas serem na forma de um fluxo de bytes, não existindo nenhuma forma de cooperação mais profunda entre as mesmas;

- **Nível Léxico:** Neste nível, grupos de ferramentas compartilham formatos de dados e convenções de operações, o que permite sua interação. Porém, para adicionar uma nova ferramenta ao grupo, a comunicação e os formatos de dados dessa ferramenta devem ser compreendidos, e isto se torna bastante trabalhoso;
- **Nível Sintático:** Neste nível um conjunto de regras de formação de estruturas de dados é compreendido por todas as ferramentas do ambiente, não sendo necessário que cada ferramenta analise, valide e converta estrutura de dados de outras ferramentas. Este é o nível encontrado na maioria dos ADS, tipicamente na forma de um esquema de banco de dados que pode ser consultado por todas as ferramentas do ambiente;
- **Nível Semântico:** Para aumentar a integração, um entendimento das estruturas de dados deve ser acompanhado de uma definição da semântica dessas estruturas. Estão disponíveis as definições das estruturas de dados mais os significados das operações sobre essas estruturas. Bancos de dados orientados a objetos suportam este nível de compartilhamento;
- **Nível de Método:** Neste nível as ferramentas, além de estarem no nível semântico, são usadas no contexto do processo de software implicando que as ferramentas “entendam o processo”. Aqui, as ferramentas devem concordar com as estruturas de dados, operações e com o processo de desenvolvimento específico. Este nível somente pode ser obtido por um ADS orientado a processos.

A compreensão dos níveis de integração entre ferramentas fornece uma base para o entendimento da estrutura dos ambientes de desenvolvimento de software. A arquitetura do ambiente depende do nível de integração pretendido. Geralmente a integração de dados é obtida através do compartilhamento de todas as informações em um repositório comum. Essa característica aproxima muito a tecnologia de banco de dados da área de engenharia de software, pois observa-se

uma preocupação da área de banco de dados em prover suporte a aplicações não convencionais, como os ADSs [Reis.00].

A Tabela 1 apresenta as capacidades necessárias nos quatro primeiros níveis de integração para atingir as categorias de integração.

Categoria\Nível	Carregador	Léxico	Sintático	Semântico
Dados	Transferência de arquivos	Uso de arquivos compartilhados	Uso do mesmo banco de dados/ banco de objetos	Uso dos mesmos metadados
Controle	Remote Procedure Call (RPC)	Uso de <i>triggers</i>	Servidores de mensagens	Acordo sobre a semântica das mensagens
Apresentação	Uso do mesmo sistema de janelas de interface	Uso do mesmo gerente de interface	Uso da mesma toolkit	Semântica padrão para toolkit

Tabela 1. Capacidades de integração de dados, controle e apresentação em cada nível

Apesar do aumento da integração trazer benefícios, existem custos associados. Para aumentar o nível de integração, é necessário aumentar o nível de compartilhamento da informação. Isto requer que as ferramentas sejam desenvolvidas utilizando esquemas padronizados, ou que haja um esforço de programação para adaptar as ferramentas existentes, ou ainda que sejam criados mecanismos para interpretação de informações entre as ferramentas.

Podemos dizer que os problemas de integração de ferramentas existentes são, em geral, decorrentes da falta de consenso entre desenvolvedores de ferramentas sobre mecanismos apropriados de integração, níveis de integração e padrões para integração de dados, controle e apresentação.

2.5 Considerações Finais

Neste capítulo foram discutidas características de projetos de software. Padronização e Integração. Trazendo uma visão geral dos sistemas de software, passando pela sua história, trazendo fatos interessantes de várias épocas. Retrata a sua importância dentro da sociedade, toda a sua evolução.

AMBIENTE DE IMPLEMENTAÇÃO DE PROCESSO DE SOFTWARE

Neste capítulo serão apresentadas características e conceitos de processos de software bem como de ambientes de desenvolvimento de software, apresentando detalhes do ambiente de software ImPProS.

3.1 Processo de Software

Informalmente, o processo de software pode ser compreendido como um conjunto de todas as atividades necessárias para transformar os requisitos do usuário em software. Um processo de software é formado por um conjunto de passos de processo parcialmente ordenados, relacionados com conjuntos de artefatos, pessoas, recursos, estruturas organizacionais e restrições e tem como objetivo produzir e manter os produtos de software finais requeridos [Reis 1998].

Os passos do processo são atividades ou tarefas. **Tarefas** são passos de processo gerenciado e **atividades** são passos elementares, que conduzem à realização de uma tarefa, e não são gerenciados. Uma atividade é um passo de processo que produz mudanças de estado visíveis externamente no produto de software. Atividades incorporam e implementam procedimentos, regras e políticas, e têm como objetivo gerar ou modificar um dado conjunto de artefatos.

A atividade aloca recursos (por exemplo, máquinas e orçamento), é atribuída a desenvolvedores (agentes), e tem o objetivo de gerar ou modificar um dado conjunto de artefatos. Uma atividade também pode ser executada somente por ferramentas automatizadas, sem intervenção humana. Neste caso ela é considerada uma atividade automática. Toda atividade possui uma descrição, a qual pode especificar os artefatos necessários, as relações de dependência com outras atividades, as datas de início e fim planejadas, os recursos a serem alocados e os agentes responsáveis pela mesma.

Um **agente** está relacionado com as atividades de um processo e pode ser uma pessoa ou uma ferramenta automatizada (quando a atividade é automática).

Os agentes podem estar organizados em cargos e responsabilidades. Diferentes agentes têm diferentes percepções acerca do que acontece durante o processo de software. Um agente, por exemplo, perceberá os aspectos de controle e alocação de recursos e cronogramas para atividades, enquanto um desenvolvedor perceberá as suas atividade como atribuições que devem ser feitas para produzir um resultado.

Um **artefato** é um produto gerado ou modificado pelo processo, sendo resultado de uma atividade. Um artefato pode ser utilizado como matéria-prima para outras atividades do processo. Desta forma uma atividade pode consumir artefatos, de entrada, e produzir artefatos de saída.

A realização do processo é afetada pelas restrições, que podem atingir atividades, agentes, recursos, artefatos, papéis e seus relacionamentos. Uma **restrição** é uma condição definida que um passo de processo deve satisfazer antes ou depois de ser executado.

Um modelo de processo de software é uma descrição abstrata do processo de software, onde vários tipos de informação são integrados, sendo representada por uma linguagem de modelagem do processo de software, a qual deve oferecer recursos para descrever e manipular os passos dos processos.

Por sua vez, um projeto é a instância de um processo, com objetivos e restrições específicas, envolvendo uma estrutura organizacional, prazos, orçamentos, recursos e um processo de desenvolvimento. A gerência de projetos tem como responsabilidades o planejamento, controle e monitoração de um processo em execução, enquanto que a gerência de processos preocupa-se em construir, analisar e verificar modelos de processo.

3.1.1 Meta-Processo de Software

Existe um ciclo de vida para processo de software análogo ao ciclo de vida de produtos de software. As atividades do ciclo de vida de processo de software são chamadas de meta-atividades, e o processo de desenvolvimento e evolução de processo de software é denominado de meta-processo.

As fases do meta-processo são representadas em alto nível de abstração, de forma que cada fase pode ser decomposta nas seguintes subfases:

- **Provisão de Tecnologia:** a tecnologia de suporte à produção de software e de modelos de processo deve ser concebida, incluindo as linguagens de modelagem de processo, modelos de processo prontos para reutilização e ferramentas para aquisição, modelagem, análise, projeto, simulação, evolução, execução e monitoração de modelos de processo;
- **Análise de Requisitos do Processo:** nesta fase são identificados requisitos a atividades de projeto de um novo processo, ou novos requisitos para um processo existente. Os requisitos resultantes especificam os recursos e propriedades que o processo deve oferecer. Os métodos utilizados nesta fase podem ser os métodos convencionais de análise de sistemas de informação (por exemplo, SADT ou modelos orientados a objetos), ou até mesmo técnicas de aquisição de conhecimentos e métodos formais;
- **Projeto do Processo:** provê a arquitetura geral e detalhada do processo. Nesta etapa as linguagens de modelagem do processo são utilizadas, havendo necessidade que satisfaçam os requisitos;
- **Implementação ou Instanciação do Processo:** implementa a especificação do processo produzida pela atividade anterior. Nesta fase é gerado um modelo de processo instanciado, contendo informações detalhadas sobre prazos, agentes e recursos utilizados por cada atividade definida no processo;
- **Simulação do Processo:** a simulação ocupa um papel chave na verificação e validação dos processos definidos. A simulação é uma tarefa que geralmente é acompanhada tanto pelo projetista de processo quanto pelo gerente do desenvolvimento com o objetivo de antever o comportamento do projeto (execução do processo);
- **Execução do Modelo de Processo:** nesta etapa o modelo de processo instanciado é executado através da invocação de ferramentas para guiar e assistir a realização do processo no mundo real. Informações sobre o andamento do processo (*feedback*) são coletadas e analisadas durante a execução;
- **Avaliação do Processo:** provê informação quantitativa e qualitativa descrevendo o desempenho de todo o processo em execução. Esta fase

ocorre simultaneamente à execução do modelo de processo e as informações adquiridas são utilizadas na atividade de análise de requisitos. Nesta fase são usados métodos de avaliação do processo como o SCAMPI do SEI, assim como métricas de monitoração do processo e do produto.

No meta-processo é necessária a participação dos agentes humanos que operam na fase de execução do processo. Entretanto, para que o processo seja modelado entra em cena um projetista de processo, que é o responsável por descrever o processo a ser executado e um gerente do processo que deverá acompanhar a execução e a avaliação do processo, analisando seu desempenho [OLIVEIRA 2005].

3.2 PSEE - Ambientes de Desenvolvimento de Software Centrado no Processo

O surgimento da tecnologia CASE (Computer Aided Software Engineering) - Engenharia de Software Auxiliada por Computador, exerceu um enorme impacto sobre a área de engenharia de software. A idéia de utilizar software para auxiliar a produção de software foi bem recebida pelos desenvolvedores. As ferramentas CASE proporcionam uma sólida estrutura às metodologias e métodos de desenvolvimento de software.

A definição de Ambiente de Desenvolvimento de Software ou, simplesmente, ADS veio em decorrência ao reconhecimento de que a comunicação e a coordenação entre todas as ferramentas usadas no desenvolvimento e manutenção de software são essenciais para a produção eficiente de software de qualidade. Um dos pontos-chave desta área tem sido obter um bom entendimento sobre como as ferramentas são integradas, definidas, implementadas, adaptadas e desenvolvidas.

O principal objetivo de um ADS recai na necessidade de prover um ambiente pela qual os produtos de software de grande porte possam ser desenvolvidos através da integração de um conjunto de ferramentas que suportam métodos de desenvolvimento, apoiados por uma estrutura que permite a

comunicação e cooperação entre as ferramentas [REI2000a]. O conceito de ADS é considerado bem mais amplo que o de ferramentas CASE por prover uma estrutura unificadora de serviços onde várias ferramentas suportando diferentes métodos podem ser integradas.

Uma evolução significativa nos ADS foi conseguida com a tecnologia de processos de software. A automação do processo de software foi incorporada aos ADSs mais recentes tornando-os ADS centrados em processo (ou orientados a processo), também conhecidos na literatura como PSEEs - Process-Centered Software Engineering Environment. Estes ambientes constituem uma nova geração de ADS que suportam além da função de desenvolvimento de software, também as funções associadas a gerência e garantia da qualidade durante o ciclo de vida do software.

Um ADS centrado em processo baseia-se em uma definição explícita do processo de desenvolvimento de software. Por isso o processo de software utilizado na organização deve estar formalizado e ser obedecido. O ambiente, então, usa o modelo de processo de software descrito em uma linguagem de modelagem do processo para executar de forma independente aquelas tarefas que podem ser completamente automatizadas, coordenar tarefas mais complexas, e garantir informação atualizada para a gerência, eliminando cargas administrativas desnecessárias dos usuários.

Os ADS centrados em processo atuam de forma mais abrangente no desenvolvimento de software. As funções genéricas que podem ser suportadas por um ADS centrado em processo são, segundo [Reis 2000a]:

- **Engenharia de Processos:** definição e manutenção de modelos de processo de software. O ADS deve prover facilidades de definição, análise e simulação de processos;
- **Engenharia de Software:** desenvolvimento e manutenção de um produto de software através do seguimento de um processo de software;
- **Gerência de Projetos:** coordenação e monitoramento das atividades da engenharia de software a fim de garantir que o processo está sendo seguido.

Atualmente existem propostas que visam aumentar a facilidade de uso dos Ambientes de Desenvolvimento de Software e a qualidade do processo e do produto de software com o uso de outras tecnologias. Internet, sistemas multiagentes, lógica fuzzy, redes neurais, técnicas de interface com o usuário são alguns exemplos do esforço nesse sentido.

3.2.1 Características dos ADSs Atuais

Como foi visto na seção anterior, os Ambientes de Desenvolvimento de Software mais recentes são os orientados a processo. Através da observação dos ambientes propostos na literatura é possível destacar as características principais presentes nos mesmos. Cabe ressaltar que nem todos os ambientes orientados a processo possuem todas as características que serão apresentadas.

O uso de ambientes orientados a processos acaba resultando na definição rigorosa da execução do processo. Esta característica traz benefícios muito importantes, pois permite melhorar a comunicação entre as pessoas envolvidas e a consistência do que está sendo feito. Além de facilitar no treinamento de novos funcionários, prover informações que guiam o desenvolvedor a realizar seu trabalho com mais eficiência e fornecer informações sobre o processo quando necessário.

O fato de que as definições de processo podem ser reunidas em uma biblioteca para reutilização é uma das vantagens principais do uso de ambientes orientados a processo. Esta característica faz com que a organização não somente economize em recursos, mas também possa atingir o nível 3 (três) de maturidade do modelo CMMI – Capability Maturity Model Integration. Além disso, outro recurso de grande contribuição é a coleta automática de métricas que pode ser obtido com o uso de ambientes orientados a processo [Christie 1997]. As características de ambientes orientados a processo são apresentadas a seguir, segundo [Reis 2000a]:

- **Suporte a múltiplos usuários:** o sistema deve ter mecanismos para atender vários usuários ao mesmo tempo requisitando serviços do ambiente concorrentemente e mantendo a consistência do ambiente;
- **Gerência de objetos:** o ambiente deve prover um módulo que seja responsável por controlar o acesso e a evolução de objetos

compartilhados. Geralmente este módulo é conhecido como Repositório do ambiente e é implementado através de um sistema de gerência de banco de dados (SGBD). As versões dos documentos e produtos de software devem ser gerenciadas para permitir cooperação e consistência;

- **Gerência de comunicação entre pessoas:** as pessoas que estarão envolvidas no desenvolvimento de software devem ter acesso a mecanismos de comunicação, tais como mensagens eletrônicas e conferência eletrônica. Além disso, a interface disponível para comunicação entre pessoas, e entre pessoas e o ambiente deve ser adequada e dirigida a tarefas específicas para reduzir o excesso de informação contida no ambiente;
- **Gerência de cooperação:** a edição cooperativa de documentos e itens de software deve ser gerenciada pelo ambiente de forma que os usuários envolvidos obtenham comunicação síncrona sobre os produtos que estão manipulando. Esta característica vem do fato de alguns ambientes apoiarem o trabalho de equipes de desenvolvimento geograficamente dispersas;
- **Gerência de Processo:** o suporte à modelagem e execução do processo de software tem sua importância na gerência das atividades desenvolvidas pelas pessoas e das ferramentas durante a construção de software. Dentro desta característica, encontra-se a necessidade de um formalismo de modelagem de processo e de uma máquina de execução das definições de processo. Alguns ambientes também fornecem um módulo de simulação de processos de software, para que o modelo desenvolvido com a linguagem de modelagem possa ser validado e refinado antes de sua execução real, prevenindo, desta forma, problemas com o cronograma e orçamento do projeto;
- **Extensibilidade:** permitir extensão do ambiente através da inclusão de novas ferramentas, sejam elas de apoio ao desenvolvimento de software ou com outras funções. Esta característica implica a existência de um mecanismo de integração eficiente;

- **Integração entre todos os módulos:** todos os níveis de integração devem estar disponíveis para viabilizar as características apontadas. As ferramentas do ambiente devem concordar sobre os tipos de dados, operações, métodos utilizados e o processo de desenvolvimento sendo seguido.

3.3 ImPProS – Ambiente de Implementação Progressiva de Processo de Software

Os ambientes de desenvolvimento de software centrados no processo foram propostos com o intuito de apoiar a modelagem e a execução de processos de desenvolvimento de software. No entanto, em alguns casos, a sua implementação nem sempre retrata a realidade das características da organização ou do projeto desenvolvido por esta. Isto se deve ao fato de que os responsáveis pela definição do processo não dispõem de um guia contendo as suas reais necessidades de execução e estes, por si, indiquem as melhores práticas a serem instanciadas a partir de um processo padrão.

Assim, para ajudar uma organização na implementação progressiva de um processo de software, é útil fornecer apoio automatizado por meio de um ambiente capaz de suportar as fases que a literatura especializada propõe como necessárias. O termo “progressiva” decorre do fato de que a implementação do processo é aperfeiçoado com as experiências aprendidas na sua definição, simulação, execução e avaliação.

O ambiente ImPProS (Implementação Progressiva de Processo de Software), proposto por Sandro Oliveira, em seu projeto de Tese de Doutorado no CIn/UFPE[OLIVEIRA 2005], está sendo concebido com o objetivo principal de apoiar a implementação de um processo de software em uma organização. Dentro deste contexto podem ser caracterizados como seus objetivos específicos:

- Apoiar a definição de um processo de software para organização;
- Permitir a modelagem e instanciação deste processo;
- Especificar um meta-modelo de processo de software a fim de definir uma terminologia única entre os vários modelos de qualidade de processo de software existentes, para uso do ambiente em seus serviços providos;

- Permitir a simulação do processo a partir das características instanciadas para um projeto específico;
- Dar apoio à execução do processo de software tomando como base uma máquina de inferência;
- Possibilitar a avaliação dos critérios do processo de software;
- Apoiar a melhoria contínua do processo de software e o reuso através da realimentação e coleta das experiências aprendidas.

Para alcançar estes objetivos o ambiente foi concebido segundo a arquitetura apresentada na Figura 4.

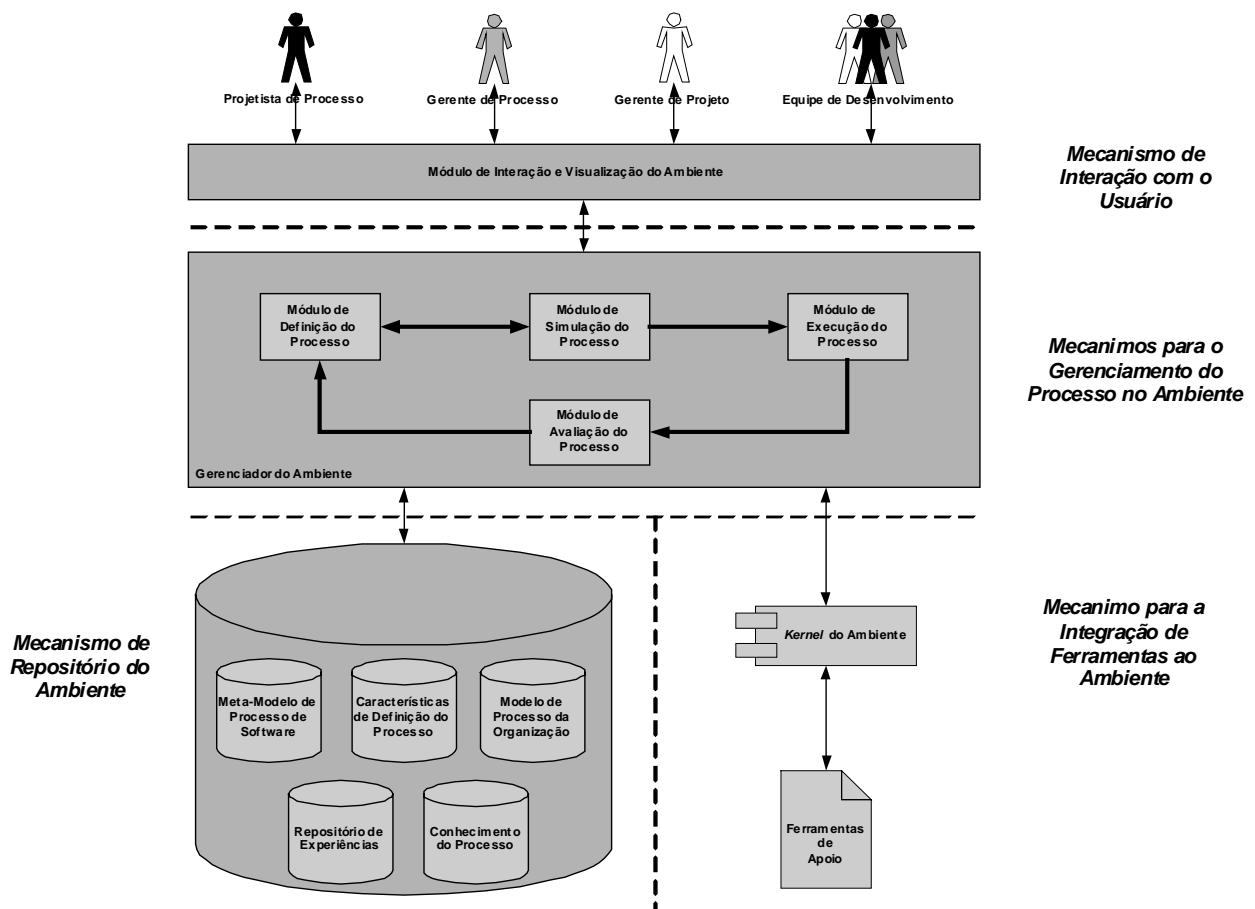


Figura 4. Arquitetura do Ambiente ImPProS

Pode-se notar que a arquitetura contempla quatro tipos de usuários para interação com o Ambiente:

- **Projetista do Processo:** responsável pela definição do processo, coleta e triagem de experiência acerca da execução de projetos. Este tipo de

usuário interage com o ambiente recebendo orientações e identificando melhorias para processos existentes ou em concepção;

- **Gerente de Processo:** este tipo de usuário acompanha a simulação e a avaliação do processo a fim de prover conhecimentos formal e informal (lições aprendidas) para possibilitar o reuso e a melhoria contínua dos processos de software;
- **Gerente de Projetos:** este usuário atua nas fases de instanciação do processo para um projeto específico, acompanhando a execução do processo e a sua avaliação para posterior coleta de experiências;
- **Equipe de Desenvolvimento:** agrupa todos os perfis relacionados à execução de um projeto de software (Gerentes, Analistas, Engenheiros de Software, Arquitetos, etc.).

A definição de cada um dos componentes definidos na arquitetura do ambiente pode ser encontrada em [OLIVEIRA 2005].

3.4 Considerações Finais

Este capítulo detalha o contexto na qual o projeto está inserido, o ImPProS. É apresentada a base deste ambiente como também todas as suas características.

UM MODELO DE PADRONIZAÇÃO E INTEGRAÇÃO DE FERRAMENTAS NO CONTEXTO DO IMPPRO S

4.1 Contexto do ImPProS

Como já mencionado, o ImPProS – Ambiente de Implementação Progressiva do Processo de Software, é um projeto de pesquisa do programa de Doutorado do CIn/UFPE, cujo interesse é a implementação progressiva do processo de software com foco no ciclo de vida de seu desenvolvimento: definição, simulação, execução e avaliação do processo.

A partir do projeto de pesquisa definido, especificado e aprovado, algumas frentes de trabalho foram propostas para tornar realidade o que o mesmo se propõe. Desta forma, procurou-se promover esta pesquisa com a ajuda de: alunos (nove alunos no total) concluintes do curso de Bacharelado em Ciência da Computação, a partir do desenvolvimento de seus Trabalhos de Graduação; alunos da disciplina Desenvolvimento de Sistemas, no período 2005.1, ministrada pelo Prof. Alexandre Vasconcelos; alunos de Iniciação Científica; e alunos do programa de Mestrado do CIn/UFPE.

Embora o foco do projeto seja a implementação de processos de software a partir da análise de características organizacionais, de projetos de software e de produtos de software, no início da implementação do projeto de pesquisa pouco se deu importância quanto ao modelo de padronização e integração do qual todos iriam fazer uso. Modelo este, extremamente necessário no contexto do projeto de pesquisa, visto que cada uma das equipes não tinha interação direta a fim de promover uma padronização do projeto como um todo. Assim, percebeu-se que, mesmo com o desenvolvimento eficaz das funcionalidades do projeto, o mesmo não tinha uma padronização e conseqüentemente a integração das partes desenvolvidas por cada equipe tornava-se difícil.

Assim, a partir deste interesse, detalhou-se um modelo de padronização e integração, o qual todos os membros das equipes do projeto de

pesquisa deveriam seguir a fim de promover a iteração mais eficiente dos módulos em desenvolvimento para que a integração fosse facilitada. As seções a seguir apresentam este modelo.

4.2 Sugestão de um Modelo de Padronização

Nesta seção será apresentada uma proposta de um Modelo de Padronização de Software.

4.2.1 Arquitetura do Projeto

A arquitetura do projeto visa especificar a estrutura lógica e física de implementação do projeto, possibilitando a definição das tecnologias usadas no modelo de implementação e os padrões de projeto usados para a sua composição.

4.2.1.1 Especificações da Implementação

As seguintes tecnologias foram selecionadas e adotadas para o desenvolvimento do projeto:

- **Plataforma de Desenvolvimento:** como linguagem de programação para implementação, foi escolhida Java (1.4), pois a mesma trata-se de uma plataforma livre, reconhecida pela sua qualidade, possuindo uma extensa gama de recursos e bibliotecas que auxiliam e facilitam o desenvolvimento, sendo adotada como padrão para o desenvolvimento do ambiente. Além de ser uma plataforma livre, o que atendia as restrições impostas, também existia uma boa experiência em seu uso, por parte dos envolvidos no projeto.
- **Ambiente de Desenvolvimento:** foi escolhido como IDE, para o desenvolvimento, o Eclipse, pois além de se tratar de um ambiente gratuito, possui perspectivas (Resource, Java, Debug, Team Synchronizing) que funcionam como uma poderosa ferramenta de auxílio, além de possuir integração com banco de dados e vários

plugins que contribuem para o aumento da produtividade. O desenvolvimento da GUI foi feito na ferramenta NetBeans.

- **Persistência de Dados:** foi adotado o banco de dados MySQL (Distrib 5.0.16, for Win32 (ia32)) e utilizado como ferramenta de gerência o MySQL Front, a qual além de ser gratuita, possui todas as funcionalidades necessárias para se manipular os dados armazenados.

4.2.1.2 Visão Geral da Arquitetura

Para o desenvolvimento do projeto foi adotada como padrão a arquitetura em três camadas, a fim de construir um sistema com baixo acoplamento.

Esta arquitetura possibilita que cada camada possa ser desenvolvida de forma independente dando possibilidades de cada especialista executar o melhor do seu trabalho, ou seja, cada camada (Banco de dados, Regras de Negócio e Interface com o Usuário) pode ser substituída ou novamente implementada sem que as outras camadas sofram manutenções. A arquitetura definida esta representada na Figura 5.

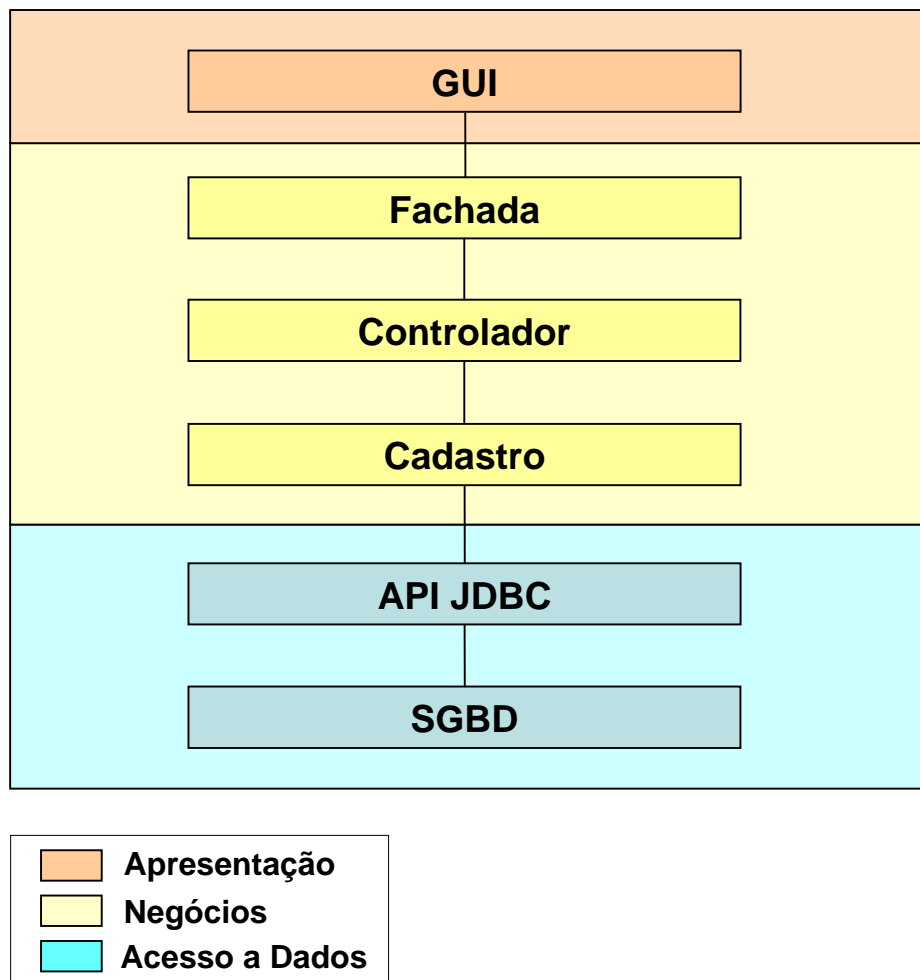


Figura 5. Arquitetura de Implementação do ImPProS

A seguir serão descritas as camadas da arquitetura bem como os elementos que compõem cada camada.

Camada de Apresentação

É a camada responsável por promover a interação entre o usuário e a aplicação. Capturando as entradas fornecidas e apresentando respostas.

- **GUI (*Graphic User Interface*):** é a porta de entrada do software, implementam as telas da aplicação e é responsável exibir as informações, ou seja, o resultado do processamento dos dados pelo sistema. Responsável por capturar as ações realizadas pelo usuário e exibir as respostas vindas do sistema. Entre as possibilidades de interfaces estão: comandos de texto e janelas de programas.

Camada de Negócios

Tem por objetivo encapsular a manipulação de dados e o funcionamento da aplicação, separando assim, as regras de negócios da interface gráfica e da persistência dos dados. Nesta camada estão armazenadas todas as regras relacionadas ao domínio da aplicação, necessárias para manter o sistema consistente do ponto de vista do negócio. É responsável por validar os dados fornecidos pelos usuários. É onde estão modeladas as seguintes entidades.

- **Fachada:** o sistema possui apenas um elemento desse tipo, tem a responsabilidade de fornecer uma entrada única do sistema para acesso às regras de negócio, tornando o sistema independente de interface gráfica. Implementado utilizando o padrão de projeto *Singleton* e *Facade*;
- **Controlador:** responsável por mapear e controlar as ações, servindo como uma camada intermediária entre a camada de apresentação e a camada lógica. Tem a finalidade de agrupar funcionalidades, fluxos de execução, semelhantes, provendo uma melhor organização lógica ao software. É responsável pela implementação das regras de negócio necessárias a execução dos fluxos de atividades;
- **Cadastro:** faz a comunicação entre a camada de negócios e a camada responsável pela persistência dos dados, tornando o sistema independente da implementação do acesso a dados.
- **Classes Básicas:** Modela coisas no mundo real em objetos.

Camada de Acesso a Dados

Nesta camada estão definidos os mecanismos para tornar persistentes os dados em processamento pelo sistema e a recuperação destes dados. Aqui se toma a decisão dos mecanismos de persistência que serão usados, ou seja, é responsável pelo armazenamento e recuperação dos dados. Esta camada é composta pelos seguintes elementos:

- **API JDBC:** responsável por manipular os mecanismos de persistência, definindo e realizando as operações (inclusão, alteração, exclusão e consulta) de acesso ao banco de dados e de materializar os resultados

providos pelo mesmo. Além disso possui a finalidade de prover a conexão do sistema ao Banco de Dados;

- **SGBD**: responsável por armazenar fisicamente os dados da aplicação, fornecendo também meios para recuperar as informações.

4.2.2 Especificação dos Requisitos

A especificação de requisitos é o conjunto de informações imperativas para a construção de um sistema. O modelo abaixo proposto tem como objetivo definir um padrão de documento visando o fácil entendimento dos casos de uso por diferentes equipes.

Cabeçalho

É importante que o documento possua em todas as páginas um cabeçalho com os seguintes dados:

<Nome do Projeto>	Versão : <Número>
Documento de Requisitos	Data da Versão: <Data>
<Identificador do Documento>	

Histórico de Revisões

Data	Versão	Descrição	Autor

O histórico de revisões serve para manter uma base histórica das alterações do documento.

Introdução

<O Analista de Sistemas deve fazer uma descrição breve da finalidade da elaboração deste documento para o projeto em desenvolvimento. Uma descrição sucinta para o que este documento se aplicará. O que será afetado e influenciado por este documento.>

Problema Identificado

<Aqui, o Analista de Sistemas deve fazer uma descrição sobre o problema que o sistema pretende solucionar.>

Visão Geral do Sistema

<Aqui, o Analista de Sistemas deve fazer uma descrição sucinta do objetivo do projeto, ressaltando principalmente os requisitos concebidos ao longo da fase de especificação a partir de reuniões e entrevistas com clientes.>

Referências

<Uma lista de documentos relacionados e que servem como referência desta Especificação dos Requisitos do Usuário.>

Requisitos Funcionais

Atores

<Adicione aqui o gráfico com a hierarquia entre os atores>

Após o gráfico deve-se colocar a lista de atores seguindo o modelo abaixo:

<Identificador> <Nome do Ator>

<Descrição da função do Ator no sistema.>

<Adicione aqui o gráfico com o Modelo de Casos de Uso definido>

Após o gráfico deve-se colocar a lista de requisitos funcionais seguindo o modelo abaixo:

<Identificador> <Nome do Requisito>

Identificação	Nome	Caso de Uso Relacionado
<Identificador do Requisito>	<Nome do Requisito>	<Identificador do Caso e Uso><Caso(s) de Uso relacionado(s) para a contemplação do Requisito>
Descrição		
<Descrição sucinta do Requisito do Sistema>		

Detalhamento dos Casos de Uso

<Nesta seção o Analista de Sistemas deve especificar detalhadamente todos os casos de uso identificados no modelo visualizado na seção anterior que são usados para detalhar os requisitos levantados do sistema.>

<Identificador> <Nome do Caso de Uso>

1. Descrição Sumária		
<Uma descrição detalhada de como se procede a execução do Requisito como um todo a partir do Diagrama anexado>		
2. Atores		
<O Analista de Sistemas deve especificar o ator que ativa a execução do Requisito>		
3. Prioridade		
<Com base nas especificações feitas no documento de Atributos dos Requisitos, deve-se especificar o grau de prioridade do requisito a ser especificado>		
() Essencial	() Importante	() Desejável
4. Requisitos Associados		
<O Analista de Sistemas deve listar quais os Requisitos do Sistema que os Casos de Uso representam>		
5. Entradas		
<Deve-se definir o que servirá como entrada para a execução funcional do requisito>		
6. Pré-Condições		
<O Analista de Sistemas deve analisar e especificar se há alguma pré-condição para a execução do requisito>		
7. Saídas		
<Deve-se definir o que teremos como retorno da execução do requisito>		
8. Pós-Condições		
<O Analista de Sistemas deve analisar e especificar se há alguma condição ao final da execução do requisito>		
9. Fluxo de Eventos		

9.1 Fluxo Básico

<O Analista de Sistemas deve listar como dar-se-á o fluxo de execução dos eventos para a realização do requisito>

9.2 Fluxos Alternativos

<O Analista de Sistemas deve listar, caso haja alguma exceção na execução, os eventos a serem realizados contrários a esta Falha ou outros fluxos>

Requisitos Não-Funcionais

< Nesta seção o Analista de Sistemas deve especificar algumas características que permitem intermediar o desenvolvimento do projeto de software. >

- Desempenho
- Usabilidade
- Manutenibilidade
- Confiabilidade/Robustez
- Segurança
- Restrições de Hardware e Software

4.2.3 Padrão de Interfaces Gráficas

Para garantir uma padronização da interface gráfica, foi definido um modelo que todas as telas deveriam seguir.

O sistema deve apresentar uma tela inicial, conhecida como *splash*, com as seguintes informações do sistema. Tal como o logotipo do nome da ferramenta, os autores, *copyright* e a instituição que o desenvolveu.

As janelas foram divididas em 3 importantes áreas: Descrição, Conteúdo e Ação.

O objetivo da Área de Descrição é deixar claro ao usuário qual o objetivo da tela. Uma descrição simples, porém completa deve ser exibida nesta área.

Na Área de Conteúdo, devem ficar todos os componentes que o usuário terá que interagir. Ou seja, o formulário que deve ser preenchido para satisfazer o caso de uso.

A Área de Ação tem como principal objetivo agrupar todas as ações disponíveis para o usuário.

A Figura 6. mostra uma tela com todas as ações que o usuário pode executar no momento. É possível verificar que existem ações não disponíveis, visto que o usuário está incluindo um novo membro.

ProKnowledge - Ferramenta de Aquisição de Conhecimento

Definindo Usuários do ProKnowledge

Usuários Perfis

Nome: Alexandre Vasconcelos

Email: amlv@cin.ufpe.br

Perfil: Administrador

Login: amlv

Senha: ****

Listar Cancelar Alterar Remover Inserir

Figura 6. Janela de Manutenção do Usuário

4.2.4 Modelagem dos Dados

Na Modelagem dos Dados, todas as tabelas deverão começar com o código do sistema ao qual elas pertencem. Esse código deve ser composto por 3 letras que seja simples identificar a qual sistema ela pertence. Esta regra serve também para as colunas das tabelas. Estas, por sua vez, deverão possuir nomes claros, separados por *underscores*(_) para facilitar sua leitura. No caso de palavras grandes com códigos ou identificadores conhecidos, pode-se optar pelo mesmo. Por exemplo, a palavra identificador, que tem como código as letras id.

É necessária a construção de uma tabela que descreva todas as colunas das tabelas. Os dados necessários são:

- Atributo
- Descrição
- Chave primária
- Chave Estrangeira
- Tipo

- Tamanho

4.2.5 Padrão de Codificação

O maior esforço no desenvolvimento de software é dedicado às atividades de manutenção, desta forma, quanto mais fácil for o entendimento do código do sistema, mais produtiva será a equipe de desenvolvimento. Normalmente nas grandes empresas é verificado que as pessoas que escrevem o código não são as mesmas que o mantêm e, quando são, geram uma dependência com o código desenvolvido que dificilmente é dissolvida e entendida por outras pessoas. Um padrão de codificação visa minimizar esses problemas, pois estabelece regras definindo como o código deve ser escrito para favorecer a impessoalidade do artefato. Sendo assim o seguinte padrão de codificação foi definido:

ARQUIVOS

Nomenclatura

Os arquivos devem ter, obrigatoriamente, o mesmo nome da classe pública que contêm. Não se define mais de uma classe por arquivo, exceto para as *Inner Classes* e classes auxiliares, declaradas privadas.

Documentação

Na documentação é altamente recomendado o uso do tipo de comentário definido no Javadoc [Javadoc.05] onde inicia-se o comentário com “/” e termina-se com “/”, o qual é usado para gerar documentação do código em páginas *HTML*.

Cada arquivo começa com um bloco de comentários contendo as seguintes informações: título do projeto, nome da classe ou interface, e informação de *copyright* relevante ao conteúdo do arquivo.

Se um arquivo possuir mais de um tipo (classe ou interface), é inserida uma lista com uma pequena descrição de cada tipo que compõe o arquivo. É importante destacar uma explicação que justifique a declaração de mais de um tipo por arquivo, pois Java só permite um tipo público por arquivo, dificultando a busca dos tipos não públicos.

Abaixo, segue o modelo de documentação de arquivo.

```

/* Projeto: <Título do projeto>
*
* Tipo1: <Nome da classe ou interface pública>
* Tipo2: <Nome da classe ou interface>, descrição: <descrição da classe
ou
* interface>
* ...
* TipoN: <Nome da classe ou interface>, descrição: <descrição da classe
ou
* interface>
*
* Informação de Copyright opcional
*/

```

Declaração

A classe ou interface pública é a primeira a ser declarada em um arquivo. A seguir, uma tabela apresenta a ordem das declarações em um arquivo.

Declaração de um arquivo

1. Comentários
2. Declaração do pacote
3. Declaração de classe e interface
4. A classe pública do arquivo é declarada no início do mesmo. Este item de declaração é repetido para cada interface ou classe que compõe o arquivo.

```

/*
 * JanelaAvaliacaoIndividual.java
 *
 * Created on 25 de Janeiro de 2006, 20:30
 */
package br.ufpe.cin.gui;

/**
 * Esta classe representa a janela de Avaliação Individual.
 *
 * Exemplo de uso:
 * <pre>
 * JanelaAvaliacaoIndividual win = new JanelaAvaliacaoIndividual();
 * win.setVisible(true);
 * </pre>
 *
 * Limitações: Uma janela não pode ser criada dentro de outra por...
 *
 * @author Albérico Pena Jr.
 * @see javax.swing.JFrame
 */
public class JanelaAvaliacaoIndividual extends javax.swing.JFrame {

    /** Creates new form JanelaAvaliacaoIndividual */
    public JanelaAvaliacaoIndividual() {
        /* Comentários sobre opções de
         * implementação invisíveis ao
         * Javadoc são feitos aqui */

        initComponents();
    }
}
..

```

```
}
```

PACOTES

Os identificadores que compõem o nome de um pacote – são separados por pontos e não há restrição quanto ao seu número; são escritos com letras minúsculas e não devem conter caracteres especiais, como *underscores*, ou caracteres específicos de uma língua.

Os pacotes são nomeados de acordo com os conceitos que agrupam.

```
package impros.proknowledge.basics;
```

Declaração

Após o comentário de início do arquivo, é declarada a sentença `package`, após isto, a lista de classes importadas é relacionada através da cláusula `import`.

As classes dos pacotes do núcleo (*core*) de Java, pacotes tipo “java.<nome do pacote>”, são listadas primeiro. Após estas, são listadas as classes das extensões de Java, pacotes tipo “ javax.<nome do pacote>”, e, por último, as classes específicas do sistema e outras APIs utilizadas.

CLASSES

Nomenclatura

Os nomes das classes não devem ser abreviados. As abreviações devem ser usadas, apenas, quando as abreviações são mais conhecidas que o seu nome completo, pois a economia de palavras na elaboração de um nome não compensa a perda de expressividade associada. Não usa-se artigos ou preposições para conectar substantivos e adjetivos, nem caracteres específicos da língua, por exemplo ç e ü. A primeira letra de cada palavra que compõe o nome de uma classe deve ser maiúscula.

O nome de uma classe representa um objeto da mesma e não o seu conjunto de objetos. Assim, o nome da classe é sempre no singular, exceto nos casos em que o próprio objeto represente uma coleção de outros objetos; nestes casos, usa-se a palavra que represente a coleção no singular e o nome dos objetos no plural como, por exemplo, `DAOInformacoes`. Onde `DAO` representa

Data Access Object, ou objeto de acesso a dados que representa uma coleção de objetos.

Toda classe que define uma exceção contém a palavra *Exception* no final de seu nome.

Documentação

Cada classe começa com um comentário “/**...*/” descrevendo o seu propósito, instruções de uso e, opcionalmente, alguns exemplos para facilitar o uso da mesma. Em seguida, têm-se lembretes sobre possíveis melhoramentos e defeitos existentes na classe. No final do comentário, adiciona-se o nome dos autores e referências úteis para o entendimento da classe.

Em seguida, tem-se a declaração do nome da classe e seus supertipos, se necessário, com algum comentário “/*...*/” que não deve aparecer no documento gerado pelo Javadoc.

Abaixo, segue o comentário de uma classe usando os marcadores comentados anteriormente, que devem aparecer na ordem ilustrada abaixo:

```
/**
 * Esta classe representa a janela de Avaliação Individual.
 *
 * Exemplo de uso:
 * <pre>
 * JanelaAvaliacaoIndividual win = new JanelaAvaliacaoIndividual();
 * win.setVisible(true);
 * </pre>
 *
 * Limitações: Uma janela não pode ser criada dentro de outra por...
 *
 * @author Albérico Pena Jr.
 * @see javax.swing.JFrame
 */
public class JanelaAvaliacaoIndividual extends javax.swing.JFrame {

    /** Creates new form JanelaAvaliacaoIndividual */
    public JanelaAvaliacaoIndividual() {
        /* Comentários sobre opções de
         * implementação invisíveis ao
         * Javadoc são feitos aqui */

        initComponents();
    }

    ...
}
```

Declaração

Após a declaração do nome da classe e seus supertipos, são declaradas as constantes, variáveis de classe, variáveis de instância, construtores, finalizador, métodos de classe e métodos de instância, nesta seqüência. Quanto aos

modificadores de acesso, primeiro declaram-se as variáveis públicas, depois as protegidas, as sem modificadores, e, por último, as privadas.

Os métodos são agrupados por funcionalidade e não pela forma de acesso ou sua condição de estático ou de instância.

A tabela 2. ilustra a ordem de declarações dos componentes de uma classe:

1. Comentários da classe (/**...*/)	
2. Sentença class	
3. Comentários de implementação da classe	Caso necessário.
4. Constantes	
5. Variáveis de classe	Na seguinte ordem: públicas, protegidas, sem modificadores (pacote), privadas.
6. Variáveis de instância	Na seguinte ordem: públicas, protegidas, sem modificadores (pacote), privadas.
7. Construtores	Na seguinte ordem: públicas, protegidos, sem modificadores (pacote), privados.
8. Métodos	Os métodos devem ser agrupados por funcionalidade e não por forma de acesso.

Tabela 2. Declarações dos componentes de uma classe

INTERFACES

Nomenclatura

O nome de uma interface é um adjetivo ou substantivo, e segue as mesmas regras para nomenclatura de classes precedido da sua abreviação Intf.

Abaixo, segue exemplos de possíveis nomes para interfaces:

- IntfDAOREpositorio
- IntfDAOAcessoDados

Documentação

A forma de documentação das interfaces é idêntica a das classes.

Declaração

As declarações de interface seguem a ordem apresentada na tabela abaixo:

1. Comentários da interface <code>"/**...*/"</code>	
2. Sentença interface	
3. Constantes	Na seguinte ordem: públicas, protegidas, sem modificadores (pacote), privadas
4. Métodos	Os métodos devem ser agrupados por funcionalidade

Tabela 3. Declarações das interfaces

Variáveis de classe, de instância e constantes

Nomenclatura

Os nomes das variáveis de classe e instância atendem aos seguintes requisitos abaixo:

- *Lower Camel Case* - A primeira letra das palavras, exceto da primeira palavra, é maiúscula. Ex: lowerCamelCase;

- Os nomes não são abreviados, exceto nos casos que a sua abreviação seja mais sugestiva que o nome completo, ou no caso de variáveis que armazenam componentes visuais;
- Não se mistura palavras de mais de uma língua;
- Não se utiliza nenhum caractere especial nem específico de uma língua.

Os nomes das constantes são compostos de palavras não abreviadas com todas as letras maiúsculas utilizando *underscores* como separadores.

Veja abaixo nomes de constantes e variáveis de classes e instância:

- VALOR_MAXIMO // constante
- btnOk // componente visual
- lblContexto // componente visual
- informacao // variável de instância
- conhecimento // variável de instância
- contador // variável de classe

Documentação

A documentação de uma variável ou constante é feita em uma linha, exceto nos casos em que não caiba em uma linha de código.

Faz parte da documentação a descrição do invariante que se aplique à variável como, por exemplo, o intervalo de 1 a 12 para a variável mês.

Há uma justificativa caso alguma variável ou constante não seja declarada privada.

Declaração

Apenas uma variável ou constante é declarada por linha de código.

MÉTODOS

Nomenclatura

Os nomes dos métodos atendem aos requisitos abaixo:

- *Lower Camel Case* - A primeira letra das palavras, exceto da primeira palavra, é maiúscula. Ex: `getInformacao()` ;
- Os nomes não são abreviados;

- Não se mistura palavras de mais de uma língua, exceto nos métodos de acesso a variáveis;
- Não se utiliza nenhum caractere especial;
- A primeira palavra deve ser um verbo no infinitivo representando a utilidade do método, com exceção dos métodos que retornam um `boolean`, que devem começar com um verbo no presente.

Apesar da aplicação deste padrão resultar em nomes maiores, necessitando digitação extra, o efeito da sua conformidade é um código mais fácil de compreender, pois o propósito do método já é esclarecido no seu nome.

Métodos de acesso a variáveis iniciam com *get* ou *set* e finalizam com o nome da variável tendo a primeira letra de cada palavra maiúscula.

Veja abaixo alguns exemplos de nomes de métodos:

- `consultarContexto(FiltroContexto filtroContexto)`
- `removerInformacao(Informacao informacao)`
- `isValido(ItemConhecimento itemConhecimento)`
- `getNome()` // método de acesso

Documentação

Todo método contém um cabeçalho que fornece informações suficientes para seu entendimento e uso adequado. Esse cabeçalho deve obedecer ao estilo de comentário definido no Javadoc [Javadoc.05]. Inicialmente, documenta-se o que o método faz e porque faz, após isto, relacionam-se todos os parâmetros necessários para chamar o método, sua cláusula de retorno, e as possíveis exceções que pode levantar. Caso a decisão de visibilidade do método possa ser questionada, documenta-se a razão pela qual foi tomada esta decisão. Se necessário, são declaradas ao final do comentário referências a outras classes e métodos, assim como, a data da criação do método.

Veja um exemplo de documentação de método:

```
/**
 *
 * @param pVOItemConhecimento Item de conhecimento com a chave primária
 * @param pLancarExcecaoSeRegistroNaoEncontrado Lançar exceção?
 * @return O item de conhecimento consultado
 * @throws ExcecaoGenerica
 */
```

```

public VOItemConhecimento consultarPorChavePrimaria(
    VOItemConhecimento pVOItemConhecimento,
    boolean pLancarExcecaoSeRegistroNaoEncontrado)
    throws ExcecaoGenerica {
    ...
}

```

Caso necessário, outros itens são acrescentados ao cabeçalho acima como, por exemplo, precondições e pós-condições, histórico de alterações do método, questões de concorrência, limitações e erros detectados no método.

O comentário no estilo C (`/*..*/`), apesar de ser bastante prático, pois têm o efeito de comentar todas as linhas entre o início e o fim da declaração, é utilizado em blocos isolados de código, caso contrário, pode surgir problemas quando comentários no mesmo estilo precisam ser feitos em escopo mais geral.

Este problema é apresentado abaixo, ilustrando a situação em que um comentário no estilo C é aninhado e o efeito causado por isto.

```

/*
    try{
        socket.open();
        ...
        while(true){
            ...
            socket.receive(packet);
        }*/

```

O comentário em linha é realizado ao lado de uma declaração ou comando caso o mesmo não exceda o número máximo de caracteres, caso contrário, é inserido acima do trecho de código sendo comentado.

Abaixo, segue um exemplo de uso deste comentário:

```

public VOMembro incluirMembro(VOMembro pVOMembro) throws ExcecaoGenerica
{
    Integer cdMembro = null;
    try {

        BancoDadosPROKNOWLEDGE.getInstancia().iniciarTransacao();

        cdMembro = SequencialMembro.getInstancia()
            .getProximoSequencialComoInteger(); //Pega o próximo id do membro

        pVOMembro.setIdMembro(cdMembro);

        ...
    }
}

```

Caso algum trecho de código possa ser melhorado, embora esteja funcionando corretamente, coloca-se um comentário no código informando a possível melhoria, utilizando-se o seguinte formato:

```
//TODO: CORRIGIR Comentário
```

Caso exista algum trecho de código que não esteja correto e que deve ser corrigido posteriormente, comenta-se utilizando o seguinte formato:

```
//TODO: REVISAR Comentário
```

Declaração

Todas as declarações de variáveis locais são realizadas no início do código do método e todas as variáveis devem ser inicializadas.

```
private void preencherColecoes() {  
    try {  
        FiltroManterItemConhecimento filtroManterItemConhecimento =  
            new FiltroManterItemConhecimento();  
  
        filtroManterItemConhecimento  
            .setDsSituacaoItemConhecimento("Aguardando Avaliação");  
  
        ...  
    }  
}
```

INTERFACE GRÁFICA

Métodos

No Proknowlege foram usados métodos padrões que representam ações corriqueiras na programação de interfaces gráficas.

Abaixo, segue uma lista dos métodos padrões utilizados e suas utilizações:

Método	Utilização
preencherColecoes ()	Inicializa os componentes que necessitam de uma coleção
limparTela()	Limpa os campos da tela
preencherTela(Object o)	Inicializa os campos da tela com os valores correspondentes no objeto

Tabela 4. Métodos Padrões das Interfaces Gráficas

Nomes para componentes de interface gráfica

Tipo do Componente	Abreviação do Tipo	Exemplo
List	list	listMembros
Button	btn	btnNotificar
Table	tb	tbItemConhecimento
Menu	me	meArquivo
MenuItem	mi	miIndexacao
ComboBox	cb	cbTipoInformacao
Textbox	txt	txtDescricao
Panel	panel	panelAcao

Tabela 5. Nome dos componentes de interface gráfica

4.2.6 Padrão de Diretórios

Para o desenvolvimento do projeto do ImPProS e suas ferramentas de suporte, a equipe de desenvolvimento deverá usar a estruturação definida na Figura 7, para manter os arquivos fontes gerados. Uma definição prévia do que deverá conter em cada diretório é feita a seguir.

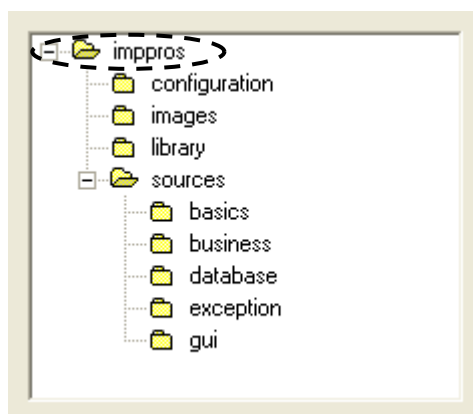


Figura 7. Estruturação dos Diretórios do ImPProS e das Ferramentas de Suporte

Na Figura 7 o diretório selecionado corresponde à raiz do projeto, o qual conterá o arquivo executável da ferramenta em desenvolvimento e os subdiretórios que organizam os arquivos fontes da automação do ImPProS e suas ferramentas de suporte. O nome deste diretório-raiz muda conforme o nome do projeto a ser implementado, podendo assumir: *proknowledge*, ferramenta de aquisição do conhecimento; *proreuse*, ferramenta de reuso de processo de software; *proimprove*, ferramenta de melhoria contínua do processo de software; entre outros. O conteúdo dos demais subdiretórios encontra-se definido a seguir:

- **configuration:** diretório que mantém os arquivos que proverão a troca de mensagens entre as ferramentas. Estes arquivos possuem a configuração que permite a integração das ferramentas a partir do nível de controle;
- **images:** diretório que mantém os arquivos de imagem do projeto;
- **library:** diretório que mantém as bibliotecas de controle e serviços necessários para a automação do ImPProS e suas ferramentas de suporte, por exemplo, controle de acesso ao BD;
- **sources:** diretório que armazena os códigos-fonte provenientes da automação do ImPProS e suas ferramentas de suporte;
- **basics:** diretório que mantém as classes básicas da aplicação, ou seja, classes que representam características comuns dos objetos (entidades manipuladas pela aplicação);
- **business:** diretório que mantém as classes de negócios da aplicação, que servem para prover validações dos serviços providos;
- **database:** diretório que mantém as classes de acesso ao BD a partir da execução dos scripts do BD;
- **exception:** diretório que mantém as classes de tratamento das exceções geradas pela aplicação;
- **gui:** diretório que mantém as classes de interface gráfica da aplicação

4.3 Integração no ImPProS

Como visto no capítulo anterior, o ImPProS é um ambiente de implementação de processo de software que tem com foco a definição, simulação, execução e avaliação do processo utilizando ferramentas de suporte ao processo de software e de apoio ao desenvolvimento de software. Ele deve prover em sua estrutura a integração dos serviços (funcionalidades) de ferramentas, a partir da disponibilização por parte do seu fornecedor, de um conector de comunicação entre o ImPProS e a ferramenta a ser integrada. Esta integração dá-se a partir da categoria de controle da integração das ferramentas [Jorgensen 1994], uma vez

que o ImPProS deve ser capaz de notificar eventos para outras ferramentas, ativar outras ferramentas e compartilhar funções de outra ferramenta, ou seja, exercer influência sobre outras.

O modelo proposto de integração das ferramentas ao ImPProS é descrito na subseção seguinte, abordando também aspectos de como se procederá a comunicação entre o ambiente e as ferramentas alocadas e integradas ao seu uso.

4.3.1 Modelo de Integração do ImPProS

Para o ImPProS foi definido um modelo de integração com as seguintes requisitos:

- Oferecer um mecanismo para compartilhar as informações (serviços) das ferramentas a serem gerenciadas por todas as funcionalidades do ImPProS relacionadas ao uso destes serviços e ativação (manipulação) das ferramentas;
- Permitir que uma mudança efetuada num item de informação (serviço da ferramenta) seja rastreada nas funcionalidades disponíveis no ImPProS;
- Permitir acesso direto, não seqüencial, a qualquer ferramenta contida no ImPProS;

Para que os requisitos de integração sejam atendidos, é necessário definir a categoria de integração das ferramentas e a maneira com que esta integração se efetiva, ou seja, que estrutura seguir para a sua realização [Brown 1994]. Visto que a área a ser cooperada entre o ambiente e as ferramentas diz respeito ao monitoramento dos serviços providos para a manipulação das ferramentas, a integração das ferramentas ao ImPProS é baseada na categoria de controle. Dessa forma, o ImPProS deve prover duas propriedades básicas: permitir o uso das funcionalidades (serviços) das ferramentas a partir de suas manipulações pelo ambiente; e permitir o controle de quanto dos serviços de uma ferramenta é usado quando da execução das atividades do ImPProS. Para tanto, as funcionalidades das ferramentas devem ser registradas e configuradas no ImPProS, a fim de que as mesmas tornem-se conhecidas para o seu uso.

Faz-se necessário, ainda, que seja descrita a estrutura de integração que define o quanto cada ferramenta coopera dentro da categoria de integração

de controle. Assim, o ImPProS adotará um nível de integração que baseia-se no acesso comum às ferramentas, ou seja, um nível, que de acordo com Pressman [Pressman 2004], permite que o usuário invoque uma série de diferentes ferramentas de maneira semelhante, por exemplo, a partir da ativação dos seus serviços no menu (*pull-down*) disponível pelo ImPProS.

Como é possível verificar na Figura 8 o intercâmbio de serviços ferramenta a ferramenta é efetuado mediante invocação do procedimento de tradução (Tradutor) entre o ImPProS e as ferramentas, o qual “filtra” os serviços que cada ferramenta torna disponível para uso no ImPProS e os executa conforme prévia solicitação do usuário a partir de uma interface comum. Assim, o uso deste tradutor preserva as informações contidas nas ferramentas, eliminando a necessidade de re-introduzir elementos existentes da especificação ou projeto e evitando-se que erros no uso dos serviços das ferramentas sejam introduzidos desnecessariamente.

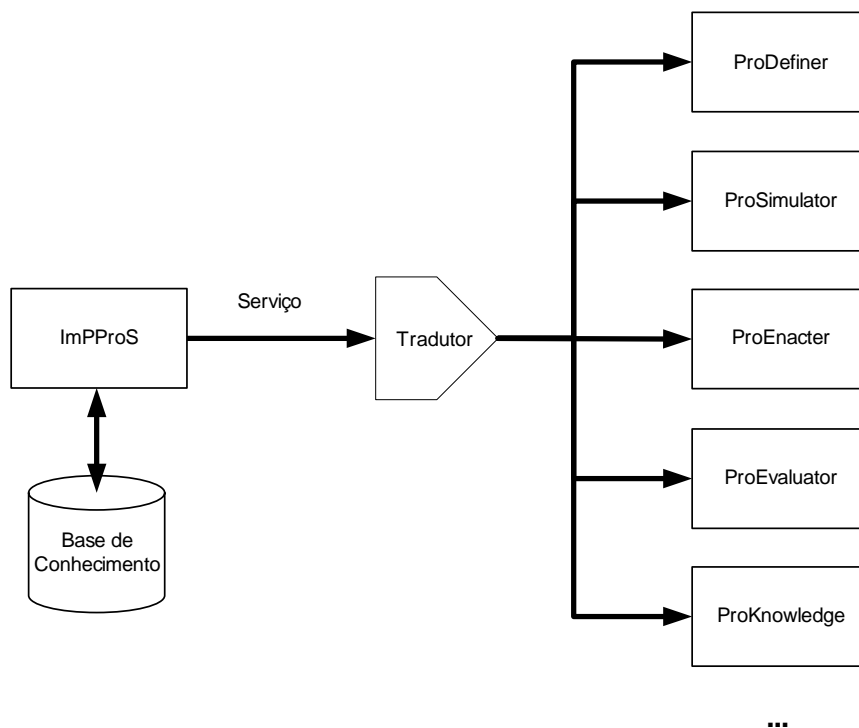


Figura 8. Nível de Integração do ImPProS

A Figura 8 apresenta, ainda, a existência de uma Base de Conhecimento que tem a função de manter as regras de implementação do processo de software no ImPProS. A sua principal finalidade é possibilitar um conhecimento mais

dinâmico à estrutura do ambiente acerca dos mecanismos que possibilitam com que o processo de software possa ser definido, simulado, executado e avaliado.

Para efetuar a estrutura de integração da Figura 8, é necessário que haja portabilidade e interoperabilidade por parte das ferramentas a fim de que se possa definir como as mesmas irão operar e interagir a partir do ImPProS e definir como os mecanismos de infra-estrutura (comunicação e execução) do ambiente irão ser usados. Assim, faz-se necessário o uso de primitivas de interface (conectores de integração, troca de mensagens, etc.) entre as ferramentas e o ImPProS para padronizar tal integração das ferramentas. Essas primitivas representam a Interface Pública de Ferramentas (PTI – *Public Tool Interface*) [Brown 1992], que também podem ser chamadas de serviços de infra-estrutura do gerenciador. Dessa forma, a PTI é tornada pública pelo ImPProS para permitir que seja usada pelas ferramentas disponíveis aos usuários.

O ImPProS, dentre as PTIs existentes no mercado, baseou-se nos conceitos de integração de controle disponíveis no PCTE (Portable Common Tool Environment) [Morris 1993], uma especificação de interface para a integração de ferramentas que adota funções (mecanismos de execução e comunicação) para manipular “entidades” que existem no contexto de desenvolvimento de software. Entre as entidades incluem-se tanto os objetos (por exemplo, dados, documentos) como as ferramentas que operam sobre os objetos. Esses mecanismos de execução proporcionam “um modo uniforme de iniciar um processo a partir de seu contexto estático, independente de ele ser um programa executável ou interpretável”, como descrito em [Pressman 2004], ou seja, as ferramentas serão executadas de forma única, a partir do ImPProS, independentemente de como as mesmas são ativadas. Já os mecanismos de comunicação gerenciam a comunicação inter-processo ao estabelecer filas de mensagens que possibilitam que diferentes ferramentas comuniquem-se com o ambiente.

Dessa forma, a estrutura do PCTE no mecanismo do ImPProS se propõe ser um super-conjunto deste ambiente, o que permite herdar funções como a passagem de mensagens e a gerência do seu processo. Tal estrutura pode ser visualizada conforme a Figura 9.

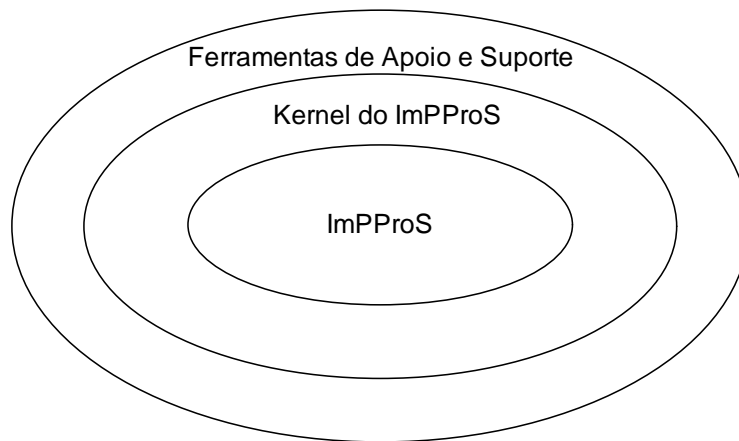


Figura 9. Estrutura de Integração do ImPProS com base no PCTE

Segundo a Figura 9, a estrutura do ImPProS com base no PCTE é composta do Kernel do ImPProS que nada mais é do que o componente que torna válido os serviços (funcionalidades) pré-definidos das ferramentas para uso no ImPProS. Assim, esta estrutura define o padrão de gerenciamento da interface do ambiente com as ferramentas dos fornecedores, permitindo assim que cada fornecedor possa adaptar sua ferramenta conforme o padrão especificado pelo PCTE no ImPProS. Este kernel terá o funcionamento conforme descrito na Figura 10.

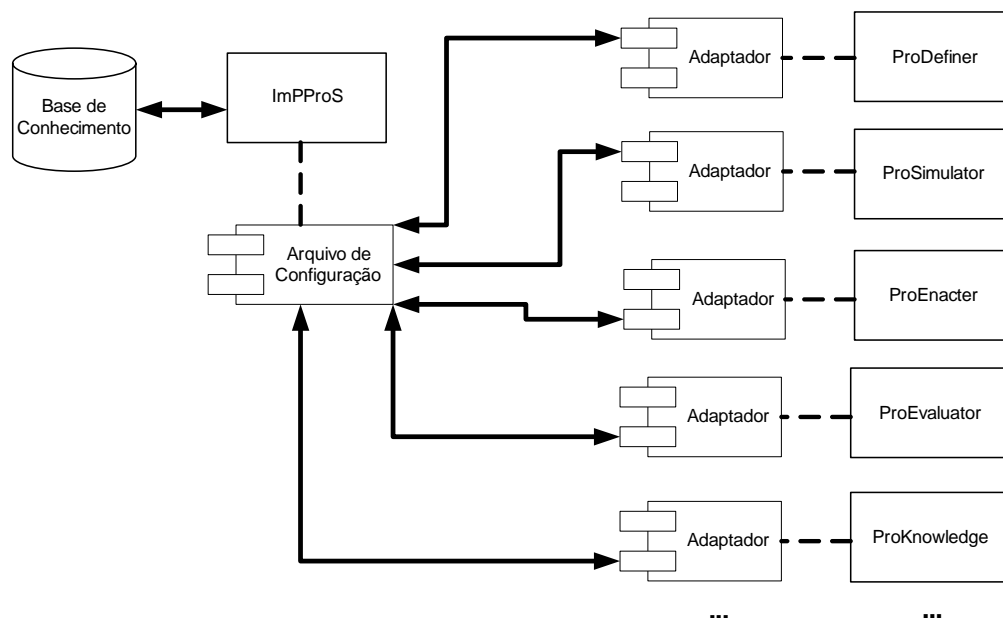


Figura 10. Funcionamento do Kernel do ImPProS

Conforme a Figura 10, o **Arquivo de Configuração** representa o componente que fará o mapeamento das funcionalidades das ferramentas integradas ao ImPProS com o nome dos métodos que as tornam acessíveis ao mesmo, ou seja, métodos que implementam suas execuções na ferramenta. **Adaptador** nada mais é que o componente que traz a implementação dos métodos que dão acesso às funcionalidades básicas das ferramentas requeridas ao ImPProS. Este último objeto fará parte do **Conector da Ferramenta**, que nada mais é do que o **Tradutor** visualizado na Figura 8, que proporciona de uma maneira geral a comunicação do ImPProS com suas ferramentas integradas e a execução dos serviços oferecidos por estas ferramentas a partir do ambiente. Já o arquivo de configuração diz respeito ao controle do ImPProS a fim de que o conector da ferramenta possa capturar as informações deste arquivo e efetivar as suas devidas execuções sobre esta ferramenta.

De acordo com o esquematizado na Figura 10, pode-se definir o funcionamento da seguinte forma: o ImPProS solicita que um determinado serviço do ProKnowledge, por exemplo, seja executado, então ele envia tal mensagem ao arquivo de configuração, este, por sua vez, verifica que método implementa o acesso a esta funcionalidade na ferramenta e qual o adaptador (conector de comunicação da ferramenta) que possui tal implementação. Posteriormente, ele envia a informação do método ao adaptador da ferramenta específica a fim de prover o acesso à funcionalidade solicitada.

Embora não exista um padrão comercial para o desenvolvimento destes conectores de integração de ferramentas, o conhecimento do mecanismo adotado para este desenvolvimento no ImPProS deve ser total uma vez que os fornecedores das ferramentas são os responsáveis por disponibilizá-los para efetivar a comunicação e execução dos serviços. Este mecanismo está definido na seção 4.3.2.

4.3.2 Mecanismos para Integração no ImPProS

O método de comunicação entre o ImPProS e as ferramentas viabiliza uma portabilidade quanto à integração de qualquer ferramenta, independente do seu fornecedor, pois não há a necessidade da implementação de um novo método de comunicação para cada ferramenta a ser integrada; e uma flexibilidade de

comunicação entre o ImPProS e os conectores das ferramentas, uma vez que a informação sobre o serviço a ser executado que é passada como parâmetro é configurada e capturada a partir do arquivo de configuração único do ImPProS. Este arquivo de configuração possui a estrutura definida na Figura 4.

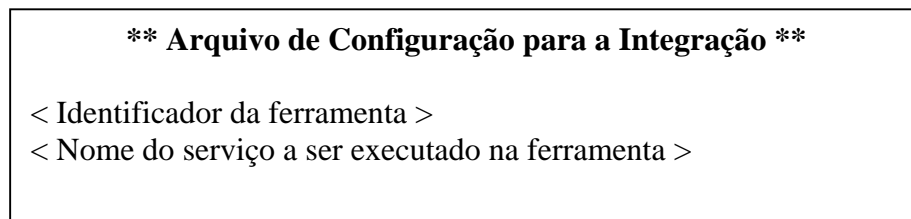


Figura 11. Estrutura do Arquivo de Configuração do ImPProS

As informações pertencentes à estrutura do arquivo de configuração do ImPProS, como visto na Figura 11, dizem respeito: ao identificador da ferramenta a ser requisitada a execução do serviço para que o conector certifique se tal requisição está sendo feita ao seu processamento, já que mais de um conector de diferentes ferramentas podem estar em execução; e ao nome do serviço configurado previamente ao ImPProS para ser executado na ferramenta.

Quando o conector já está em execução, este deve ativar a ferramenta a ser utilizada pelos usuários e certificar se há algum serviço da ferramenta no arquivo de configuração a ser executado. Caso haja, ele verifica o nome do método que implementa este serviço e o executa a partir da ferramenta. O conector deve permanecer ativo enquanto a ferramenta também estiver, a fim de prover o “canal” de comunicação do ImPProS com a ferramenta específica. Este conector, provido pelo fornecedor da ferramenta, deve obedecer à estrutura representada pela Figura 11 para a efetiva transferência de informações para os serviços das ferramentas integradas.

```

{Ler Arquivo de Configuração}
Ler ExecutarTool.ini;
{Verificar se o Identificador da Ferramenta é o Mesmo do Conector}
Se (IdentificadorFerramenta=Tool) então
Início
{Verificar se Ferramenta Encontra-se Ativa}
Se (IdentificadorFerramenta está ativa) então
Início
{Sair do Se}
Break;
Senão
{Executar Ferramenta}
Executar (Tool);
Fim-Se
{Identificar Método do Serviço da Ferramenta}
Caso NomeServico for
Início
Servico1: NomeMetodo:=Metodo1;
Servico2: NomeMetodo:=Metodo2;
....
Fim-Caso
{Executar Método do Serviço da Ferramenta}
Executar (NomeMetodo);
Fim-Se
Senão
{Esperar Próxima Requisição}
Esperar
Fim-Se

```

Figura 12. Estrutura do Conector de uma Ferramenta para a Integração

É importante lembrar, também, que o conector da ferramenta deve executar seu processamento, conforme definido na Figura 12, a cada intervalo de tempo que permita que a execução solicitada a partir do ImPProS seja efetivada.

A transferência dos objetos (mensagens) a serem tratados entre as ferramentas usando-se de padrões de frameworks são baseados em XML (*eXtensible Markup Language*), para uma melhor portabilidade quanto ao uso de inúmeras ferramentas de diferentes fornecedores e de uma flexibilidade na definição dos métodos (serviços) destas ferramentas integradas.

4.3.3 Integração do ImPProS e suas Ferramentas de Suporte ao ProKnowledge

O ProKnowledge é uma ferramenta de suporte ao ImPProS com o objetivo de manter e gerir o conhecimento adquirido ao longo da implementação de um processo de software, ou seja, um suporte que possibilita criar uma base de conhecimento e fazer uso desta durante a definição, simulação, execução e avaliação do processo de software no ImPProS. Desta forma, ele dá a possibilidade aos usuários do ImPProS e de suas ferramentas armazenarem e manterem o aprendizado obtido durante a execução das tarefas disponíveis sobre o seu controle.

Com esta possibilidade, cada tarefa do ImPProS e de suas ferramentas possui um link com o ProKnowledge garantindo a consulta de conhecimento adquirido a fim de ajudar na execução de tarefas uma vez realizadas e garantir que este conhecimento possa ser disseminado por todos que fazem uso dos serviços providos pelo ImPProS e pelas suas ferramentas. Cada uma destas tarefas possui uma palavra-chave para que o seu conhecimento possa ser indexado no ProKnowledge de acordo com o contexto do conhecimento (área do conhecimento mantida no ProKnowledge para agrupar o aprendizado adquirido, por exemplo: processo de software; resuo; melhoria contínua, etc.) e assim facilitar a consulta deste conhecimento. A Figura 13 exibe um exemplo de tela com o link com o ProKnowledge.



Figura 13. Link com o ProKnowledge

A partir da ativação deste link, o acesso ao conhecimento adquirido e mantido no ProKnowledge ocorre a partir do fluxo representado na Figura 14.

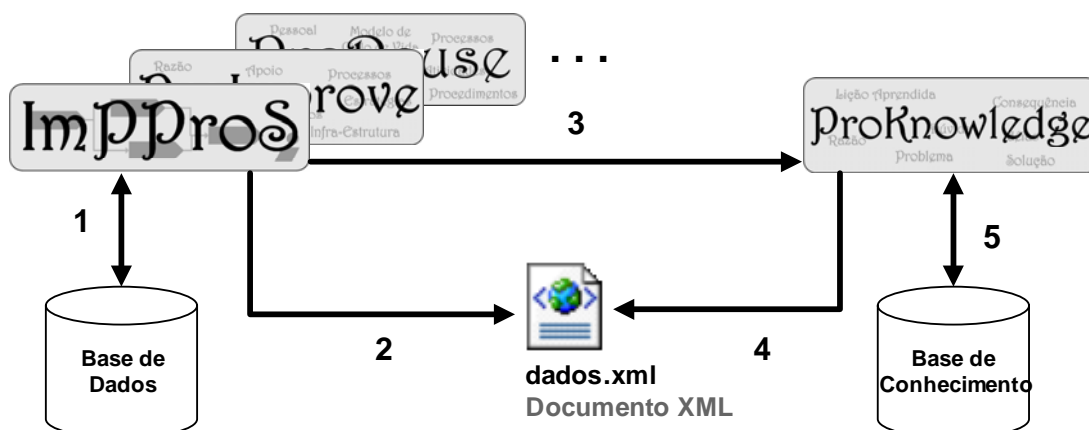


Figura 14. Fluxo de Execução da Integração com o ProKnowledge

A partir do apresentado na Figura 14, pode-se detalhar o fluxo da seguinte forma: inicialmente o ImPProS e/ou suas ferramentas de suporte acessam a Base de Dados a fim de pesquisar a palavra-chave que corresponde à tarefa para a qual foi solicitada a consulta do conhecimento; posteriormente o ImPProS e/ou suas ferramentas de suporte geram um arquivo XML (dados.xml, a Figura 15 possui um exemplo de seu uso e estruturação) contendo informações relevantes (função do ProKnowledge, contexto de conhecimento e palavra-chave inerente ao conhecimento) para a consulta do conhecimento; logo em seguida o ImPProS e/ou suas ferramentas de suporte ativam a execução do ProKnowledge, o qual lê e captura as informações mantidas no arquivo XML (dados.xml) e acessa a sua Base Conhecimento efetivando uma consulta acerca do contexto e da palavra-chave do conhecimento solicitado.

O arquivo XML deve ser gerado, segundo o padrão de diretórios definido para o ImPProS disponível no tópico Padrão de Diretórios, no subdiretório **configuration** constante da árvore de diretórios da ferramenta ProKnowledge. Antes de ser usado no ImPProS e/ou nas suas ferramentas de suporte, o ProKnowledge necessita ser configurado para uso a partir da captura do endereço físico (por exemplo: C:\proknowledge\proknowledge.jar) do seu arquivo executável onde o mesmo encontra-se instalado.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <proknowledge>
  <funcao>Consulta</funcao>
  <contexto>Processo de Software</contexto>
  <palavrachave>Modelo/Norma de Qualidade</palavrachave>
</proknowledge>
```

Figura 15. Estrutura do arquivo XML

4.4 Considerações Finais

Neste capítulo foi definida uma proposta de modelo de padronização e integração para sistemas no contexto do ImPProS. Foram discutidas boas práticas de codificação, modelagem de dados, arquitetura e padrões. Foi definida, também, uma estrutura de integração para o ImPProS.

V

ALIDAÇÃO DO MODELO

Este capítulo tem como objetivo apresentar a aplicação do modelo ao projeto Proknowledge, desenvolvido em paralelo a este documento.

5.1 Ferramenta ProKnowledge: Visão Geral

O Proknowledge é uma das ferramentas de apoio ao processo de software que compõe a estrutura do ImPProS (Ambiente de Implementação Progressiva de Processo de Software). Seu objetivo é prover a aquisição, manutenção e gerenciamento de todos os conhecimentos obtidos por uma organização em qualquer contexto (domínio) de aplicação.

Através da ferramenta, o usuário, que é um membro da empresa, pode criar um novo contexto. O contexto é a representação de um domínio da aplicação da empresa, para que os conhecimentos (experiências) obtidos possam ser categorizados. São definidos os membros que fazem parte, a área de aplicação e o usuário responsável pelo contexto.

O usuário pode associar palavras-chave ao contexto para facilitar sua busca por conhecimentos mantidos neste domínio. As palavras-chave também servem para indexar um item de conhecimento. Antes de falar sobre o item de conhecimento é necessário entender o conceito de tipo de item de conhecimento ou tipo de conhecimento. Um tipo de conhecimento pode ser uma dúvida, uma idéia ou até uma lição aprendida, ou seja, são categorias de conhecimento.

Todo item de conhecimento está associado a um tipo de conhecimento. Ou seja, para armazenar um novo item na base de conhecimento, ele terá que ter um tipo associado. O conhecimento é o ponto central do ProKnowledge. A ferramenta dá o apoio à sua criação, avaliação e validação ou remoção. Um conhecimento é composto por informações. Essas podem ser de vários tipos: título, problema, solução, causa do problema e etc.

Depois de cadastrado o conhecimento, a próxima etapa trata da sua avaliação. Neste momento, o responsável pelo contexto do conhecimento escolhe 3 membros (no mínimo e no máximo) do contexto para fazer a validação. Cada

um dos 3 membros recebe uma notificação via e-mail para efetuar a validação. Se depois de certo período de tempo (4 dias) os membros não tiverem feito a avaliação, é enviada outra notificação (e-mail de pendência). Cada membro deve fazer uma avaliação individual do conhecimento. São avaliados vários critérios, como: corretude, completude, abrangência, originalidade, relevância etc. Cada membro deverá atribuir um peso para cada critério e ao final deverá dar o seu parecer final: aceitação forte, fraca, neutro, rejeição fraca ou forte.

O responsável pela avaliação pode, a qualquer momento, acompanhar o andamento das avaliações individuais. Ao final das avaliações o responsável avaliará as respostas dos avaliadores escolhidos e tomará a decisão de manter o conhecimento na base de conhecimento ou removê-lo.

O Proknowledge permite também a consulta de todos os itens de conhecimento através do contexto e da palavra-chave, o que possibilita a sua integração ao ambiente ImPProS e todas as suas demais ferramentas de apoio à implementação do processo de software.

5.2 Aplicação dos Modelos ao Proknowledge

O objetivo deste tópico é validar os padrões definidos no capítulo 4. Para a validação foi desenvolvida uma ferramenta utilizando todos os padrões descritos. A ferramenta desenvolvida está descrita no tópico anterior, e como foi visto, chama-se Proknowledge.

A seguir seguem as evidências da aplicação do modelo apresentado.

5.2.1 Arquitetura de Projeto

A Figura 16 faz um paralelo entre a arquitetura sugerida e uma pilha de objetos do ProKnowledge, colocando cada um em sua camada dentro da arquitetura.

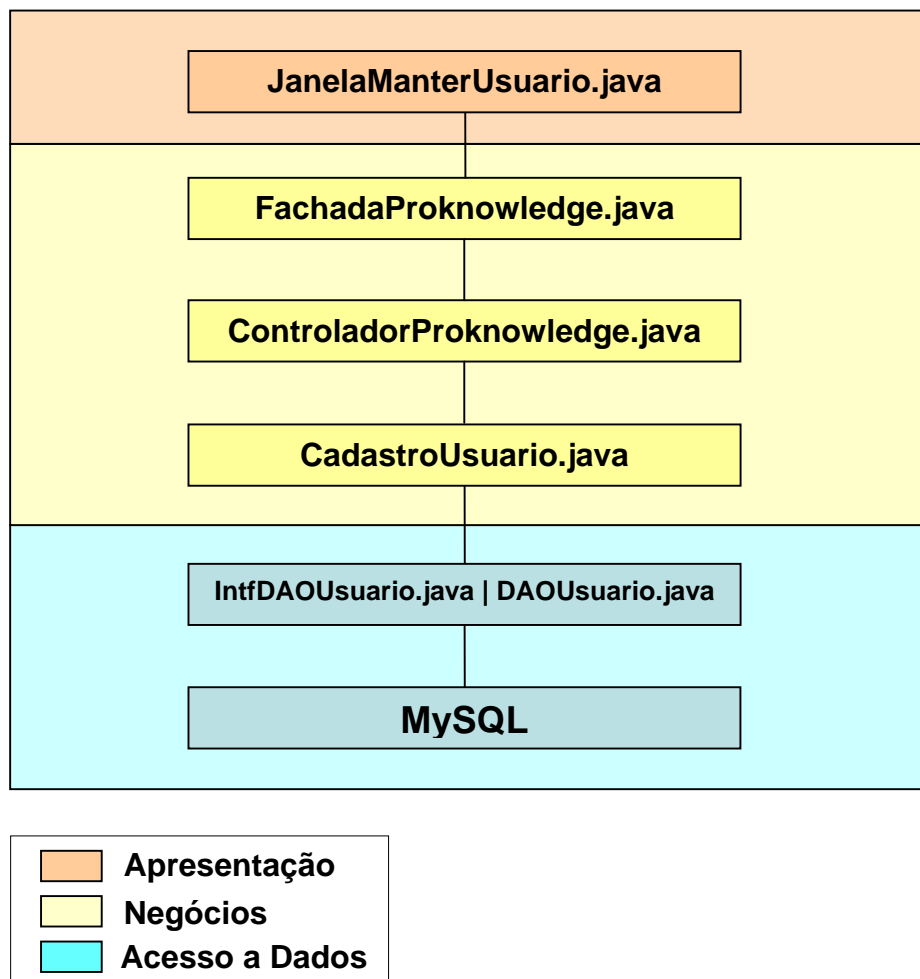


Figura 16. Mapeamento entre uma pilha de objetos do ProKnowledge e a Arquitetura

5.2.2 Especificação de Requisitos

Com a padronização dos casos de uso ficou mais fácil saber identificar todas as informações necessárias. Abaixo seguem 2 exemplos de casos de uso:

[UCF001] Cadastrar Palavras-Chave

1. Descrição Sumária		
Permite ao ator cadastrar Palavras-Chave		
2. Atores		
Usuário		
3. Prioridade		
(X) Essencial	() Importante	() Desejável
4. Requisitos Associados		
UCF002 Cadastrar Contexto		
5. Entradas		
<ul style="list-style-type: none">▪ Contexto aberto▪ Nome da palavra-chave▪ Descrição da palavra-chave		
6. Pré-Condições		
Existir um contexto aberto		
7. Saídas		
Confirmação da operação realizada. Mensagem: "Palavra-chave cadastrada com sucesso."		
8. Pós-Condições		
A palavra-chave deve estar cadastrada na base.		
9. Fluxo de Eventos		

9.1 Fluxo Básico

São preenchidos todos os campos e a palavra-chave é cadastrada com sucesso.

9.2 Fluxos Alternativos

1. *Campos não preenchidos*
Se o ator não preencheu todos os campos o sistema informa o campo que está faltando.
 2. *Cancelar a criação.*
Em qualquer momento o usuário pode cancelar a criação.
 3. *Erro no cadastro.*
O sistema informa que a operação não foi concluída com sucesso e o erro ocorrido. Se possível guiar o usuário a concluir a atividade com sucesso.
-

[UCF0010] Abrir Contexto

1. Descrição Sumária		
Permite ao ator abrir um Contexto		
2. Atores		
Usuário		
3. Prioridade		
(X) Essencial	() Importante	() Desejável
4. Requisitos Associados		
UCF002 Cadastrar Contexto		
5. Entradas		
▪ Contexto		
6. Pré-Condições		
O contexto a ser aberto está cadastrado		
7. Saídas		
8. Pós-Condições		
Alterar a mensagem de <i>status</i> da Janela Principal para Contexto: <Nome do contexto aberto>		
9. Fluxo de Eventos		

9.1 Fluxo Básico

É escolhido um Contexto a ser aberto.

9.2 Fluxos Alternativos

1. Campos não preenchidos

Se o ator não preencheu todos os campos, o sistema informa o campo que está faltando.

2. Cancelar a abertura.

Em qualquer momento o usuário pode cancelar a abertura.

5.2.3 Padrão de Interfaces Gráficas

Os exemplos a seguir demonstram o trabalho de padronização das telas do ProKnowledge. Esta padronização deverá ocorrer em todas as ferramentas do ImPProS.

O resultado obtido com interface gráfica foi muito bom, pois as telas são bastante intuitivas e amigáveis. As mensagens de erro deixaram de ser apenas de erro e mostram o que é preciso para executar a atividade corretamente.

É importante frisar que a padronização não se deu apenas no aspecto visual, ela transcende o comportamento das ações realizadas pelo usuário. Ou seja, as ações tomadas sempre terão resultados correspondentes. Por exemplo, independentemente da entidade trabalhada ao apertar o botão Listar, na Figura 18, uma nova janela se abrirá com todos os itens desta entidade.

Como foi visto no modelo, as telas são divididas em 3 áreas principais:

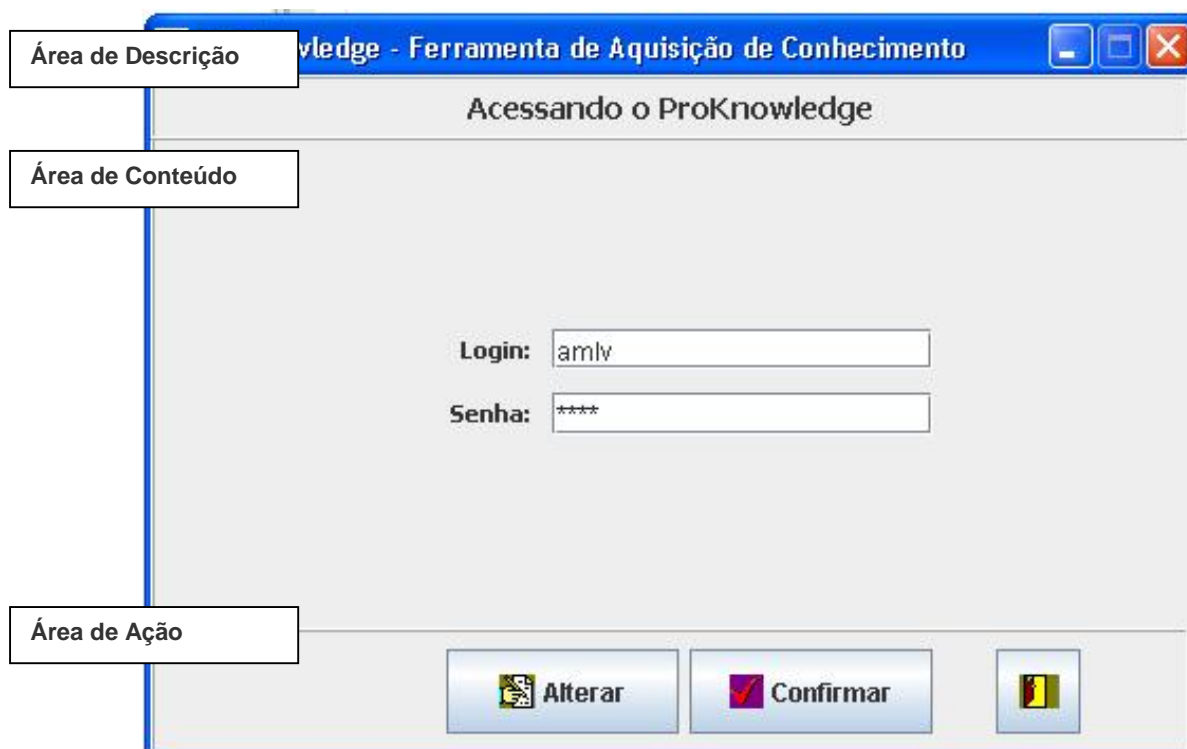


Figura 17. Tela de login do sistema

ProKnowledge - Ferramenta de Aquisição de Conhecimento

Definindo Usuários do ProKnowledge

Usuários | Perfis

Nome: Prof. Alexandre Vasconcelos

Email: amlv@cin.ufpe.br

Perfil: Gerente 2

Login: amlv

Senha: ****

Listar Cancelar Alterar Remover Inserir

Figura 18. Tela de Manutenção de Usuários

ProKnowledge - Ferramenta de Aquisição de Conhecimento

Definindo Itens de Conhecimento do Contexto

Conhecimento: teste

Tipo do Conhecimento:

Nome:

Descrição:

Situação: Aguardando Avaliação

Data do Registro: 09/02/2006

Listar Cancelar Alterar Remover Inserir

Figura 19. Manutenção de Conhecimento

5.2.4 Modelagem de Dados

Abaixo segue 2 exemplos de modelos de dados do ProKnowledge, segundo o modelo sugerido:

Tabela: PRK_CONTEXTO

Atributos	Descrição	Chave Primária	Chave Estrangeira	Tipo	Tamanho
PRK_ID_CONTEXTO	Identificador do Contexto	Sim	-	Integer	-
PRK_NOME_CONTEXTO	Nome do Contexto		-	Text	-
PRK_DESCRICAO_CONTEXTO	Descrição do Contexto		-	Text	-
PRK_APLICACAO_CONTEXTO	Aplicação do Contexto		-	Text	-
PRK_ID_MEMBRO_CONTEXTO	Membro Responsável pelo Contexto		Sim	Integer	-

Tabela 6. Descrição da Tabela Contexto do Proknowledge

Tabela: PRK_INFORMACAO

Atributos	Descrição	Chave Primária	Chave Estrangeira	Tipo	Tamanho
PRK_ID_INFORMACAO	Identificador da Informação	Sim	-	Integer	-
PRK_ROTULO_INFORMACAO	Rótulo da Informação		-	Text	-
PRK_DESCRICAO_INFORMACAO	Descrição da Informação		-	Text	-
PRK_ID_TIPO_INFORMACAO	Tipo da Informação		Sim	Integer	-
PRK_ID_ITEM_CONHECIMENTO	Item de Conhecimento da Informação		Sim	Integer	-

Tabela 7. Descrição da Tabela Informação do Proknowledge

5.2.5 Padrões de Codificação

ARQUIVOS

Abaixo um exemplo de um arquivo java, onde mostra o cabeçalho do arquivo com a declaração do pacote em seguida, seguido do comentário da classe e do construtor público.

JanelaAvaliacaoIndividual.java

```
/*
 * Projeto: ImPProS - ProKnowledge
 * JanelaAvaliacaoIndividual.java
 *
 * Created on 25 de Janeiro de 2006, 20:30
 */
package br.ufpe.cin.gui;

/**
 * Esta classe representa a janela de Avaliação Individual.
 *
 * Exemplo de uso:
 * <pre>
 * JanelaAvaliacaoIndividual win = new JanelaAvaliacaoIndividual();
 * win.setVisible(true);
 * </pre>
 *
 * Limitações: Uma janela não pode ser criada dentro de outra por...
 *
 * @author Albérico Pena Jr.
 * @see javax.swing.JFrame
 */
public class JanelaAvaliacaoIndividual extends javax.swing.JFrame {

    /** Creates new form JanelaAvaliacaoIndividual */
    public JanelaAvaliacaoIndividual() {
        /* Comentários sobre opções de
         * implementação invisíveis ao
         * Javadoc são feitos aqui */

        initComponents();
    }
    ..
}
```

CLASSES

Seguem exemplos da construção de uma classe segundo o modelo proposto:

```

/*
 * Projeto: ImPProS - ProKnowledge
 * JanelaAvaliacaoIndividual.java
 *
 * Created on 25 de Janeiro de 2006, 20:30
 */
package br.ufpe.cin.gui;

/**
 * Esta classe representa a janela de Avaliação Individual.
 *
 * Exemplo de uso:
 * <pre>
 * JanelaAvaliacaoIndividual win = new JanelaAvaliacaoIndividual();
 * win.setVisible(true);
 * </pre>
 *
 * Limitações: Uma janela não pode ser criada dentro de outra por...
 *
 * @author Albérico Pena Jr.
 * @see javax.swing.JFrame
 */
public class JanelaAvaliacaoIndividual extends javax.swing.JFrame {

    /** Creates new form JanelaAvaliacaoIndividual */
    public JanelaAvaliacaoIndividual() {
        /* Comentários sobre opções de
         * implementação invisíveis ao
         * Javadoc são feitos aqui */

        initComponents();
    }
    ...
}

```

```

/*
 * Projeto: ImPProS - ProKnowledge
 * VOPerfil.java
 *
 * Created on 25 de Janeiro de 2006, 20:30
 */
package br.ufpe.cin.basics;

import br.ufpe.cin.framework.excecao.ExcecaoAtributoNaoEncontradoVOGenerico;
import br.ufpe.cin.framework.excecao.ExcecaoAtributoObrigatorioNaoPreenchido;
import br.ufpe.cin.framework.excecao.ExcecaoTamanhoAtributoInvalido;
import br.ufpe.cin.framework.excecao.ExcecaoValorNumericoInvalido;
import br.ufpe.cin.framework.vo.VOEntidade;
import br.ufpe.cin.framework.vo.VOGenerico;
import br.ufpe.cin.util.ConstantesPROKNOWLEDGE;

public class VOPerfil extends VOEntidade {

    public static final String NM_ENTIDADE =
        ConstantesPROKNOWLEDGE.NM_SCHEMA_PROKNOWLEDGE
        + ConstantesPROKNOWLEDGE.CD_SEPARADOR_BD + "perfil";

    public static final String DS_ENTIDADE = "Perfil de um Membro";

    public static final String NM_ATR_NM_PERFIL = "NOME";

    public static final String NM_ATR_DS_PERFIL = "DESCRICAO";

    public static final String NM_ATR_ID_PERFIL = "IDPERFIL";
}

```

```

    public static final String DS_ATR_NM_PERFIL = "Nome do Perfil";

    public static final String DS_ATR_DS_PERFIL = "Descrição do Perfil";

    public static final String DS_ATR_ID_PERFIL = "Identificador do Perfil";

    private String nmPerfil;

    private boolean isNmPerfilAlterado = false;

    private String dsPerfil;

    private boolean isDsPerfilAlterado = false;

    private Integer idPerfil;

    private boolean isIdPerfilAlterado = false;

    public VOPerfil(Integer pIdPerfil) throws ExcecaoValorNumericoInvalido,
        ExcecaoTamanhoAtributoInvalido,
        ExcecaoAtributoObrigatorioNaoPreenchido {

        this.setIdPerfil(pIdPerfil);
    }

    public VOPerfil() throws ExcecaoValorNumericoInvalido,
        ExcecaoTamanhoAtributoInvalido,
        ExcecaoAtributoObrigatorioNaoPreenchido {
    }

    ...

    ...

    /**
     * @return nmPerfil Nome do Perfil
     */
    public String getNmPerfil() {
        return this.nmPerfil;
    }

    /**
     * @param pNmPerfil Nome do Perfil
     */
    public void setNmPerfil(String pNmPerfil)
        throws ExcecaoTamanhoAtributoInvalido,
        ExcecaoAtributoObrigatorioNaoPreenchido {

        if (pNmPerfil == null)
            throw new ExcecaoAtributoObrigatorioNaoPreenchido(
                VOPerfil.NM_ATR_NM_PERFIL,
                VOPerfil.DS_ATR_NM_PERFIL);
        pNmPerfil = pNmPerfil.trim();

        if (pNmPerfil.length() < 1 || pNmPerfil.length() > 50)
            throw new
                ExcecaoTamanhoAtributoInvalido(VOPerfil.NM_ATR_NM_PERFIL,
                    VOPerfil.DS_ATR_NM_PERFIL, pNmPerfil, 1, 50);

        this.nmPerfil = pNmPerfil;
        this.isNmPerfilAlterado = true;
    }
}

```

INTERFACES

Seguem exemplos da construção de uma interface segundo o modelo proposto. É possível verificar a semelhança com a construção de uma classe.

```
/*
 * Projeto: ImPProS - ProKnowledge
 * IntfDAOAvaliacaoIndividual.java
 *
 * Created on 25 de Janeiro de 2006, 20:30
 */

package br.ufpe.cin.database.dao;

import java.util.Collection;

import br.ufpe.cin.basics.VOAvaliacaoIndividual;
import br.ufpe.cin.database.filtro.FiltroManterAvaliacaoIndividual;
import br.ufpe.cin.framework.dao.InterfaceDAO;
import br.ufpe.cin.framework.excecao.ExcecaoGenerica;

/**
 * Esta interface representa todas as operações que
 * o Repositório(DAO) oferece
 *
 * Exemplo de uso:
 * <pre>
 * DAOAvaliacaoIndividual implements IntfDAOAvaliacaoIndividual
 * </pre>
 *
 * @author Albérico Pena Jr.
 * @see br.ufpe.cin.framework.dao.InterfaceDAO
 */
public interface IntfDAOAvaliacaoIndividual extends InterfaceDAO {

    public VOAvaliacaoIndividual consultarPorChavePrimaria(
        VOAvaliacaoIndividual pVOAvaliacaoIndividual)
        throws ExcecaoGenerica;

    public void incluir(VOAvaliacaoIndividual pVOAvaliacaoIndividual)
        throws ExcecaoGenerica;

    public void alterar(VOAvaliacaoIndividual pVOAvaliacaoIndividual)
        throws ExcecaoGenerica;
}
```

```

        public void excluir(VOAvaliacaoIndividual pVOAvaliacaoIndividual)
            throws ExcecaoGenerica;

        public Collection consultar(FiltroManterAvaliacaoIndividual pFiltro)
            throws ExcecaoGenerica;
    }

```

Variáveis de classe, de instância e constantes

Os exemplos abaixo mostram exemplos das variáveis de classe, de instância e as constantes.

FiltroManterAvaliacao.java

```

...
public static final String DS_ATR_ID_AVALIACAO = "Identificador da Avaliação";
public static final String DS_ATR_DT_ALERTA = "Data de Alerta";
public static final String DS_ATR_ID_ITEM_CONHECIMENTO =
    "Identificador do Item de Conhecimento";
public static final String DS_ATR_DS_PARECER = "Parecer do Avaliador";
public static final String DS_ATR_DS_COMENTARIO =
    "Comentário do Avaliador";

private Integer idAvaliacao;
private boolean isIdAvaliacaoAlterado = false;
private Date dtAlerta;
private boolean isDtAlertaAlterado = false;
private Integer idItemConhecimento;
private boolean isIdItemConhecimentoAlterado = false;
private String dsParecer;
private boolean isDsParecerAlterado = false;
...

```

MÉTODOS

Abaixo segue a descrição da construção de métodos:

ControladorProknowledge.java

```

/**
 *
 * @param pVOItemConhecimento Item de conhecimento com a chave primária
 * @param pLancarExcecaoSeRegistroNaoEncontrado Lançar exceção?
 * @return O item de conhecimento consultado
 * @throws ExcecaoGenerica

```

```

*/
public VOItemConhecimento consultarPorChavePrimaria(
    VOItemConhecimento pVOItemConhecimento,
    boolean pLancarExcecaoSeRegistroNaoEncontrado)
    throws ExcecaoGenerica {
    ...
}

```

ControladorProknowledge.java

```

public VOMembro incluirMembro(VOMembro pVOMembro) throws ExcecaoGenerica
{
    Integer cdMembro = null;
    try {

        BancoDadosPROKNOWLEDGE.getInstancia().iniciarTransacao();

        cdMembro = SequencialMembro.getInstancia()
            .getProximoSequencialComoInteger(); //Pega o próximo id do membro

        pVOMembro.setIdMembro(cdMembro);

        ...
    }
}

```

JanelaEscolherAvaliadores.java

```

private void preencherColecoes() {
    try {
        FiltroManterItemConhecimento filtroManterItemConhecimento =
            new FiltroManterItemConhecimento();

        filtroManterItemConhecimento
            .setDsSituacaoItemConhecimento("Aguardando Avaliação");

        ...
    }
}

```

INTERFACE GRÁFICA

JanelaEscolherAvaliadores.java

```

private javax.swing.JButton btnExcluir;

private javax.swing.JButton btnNotificar;

private javax.swing.JButton btnSair;

private javax.swing.JScrollPane spListaFonte;

private javax.swing.JScrollPane spListaFontel;

private javax.swing.JScrollPane spItemConhecimento;

private javax.swing.JTable tbItensConhecimento;

private javax.swing.JPanel panelDescricao;

private javax.swing.JPanel panelUsuario;

private javax.swing.JLabel lblAvaliadoresDisponivel;

```

JanelaEscolherAvaliadores.java

```
private void preencherColecoes() {  
  
    try {  
        FiltroManterItemConhecimento filtroManterItemConhecimento =  
new FiltroManterItemConhecimento();  
  
        filtroManterItemConhecimento  
            .setDsSituacaoItemConhecimento("Aguardando " +  
                "Avaliação");  
  
        ArrayList clItensConhecimento =  
(ArrayList) FachadaPROKNOWLEDGE.getInstancia().  
consultarItemConhecimento(filtroManterItemConhecimento);  
  
        Object[] itens = new Object[3];  
        VOItemConhecimento voItemConhecimento = null;  
  
        model.addColumn("Item Conhecimento");  
        model.addColumn("Data do Registro");  
        model.addColumn("Situação");  
  
        ...  
    }  
}
```

JanelaManterIndexacao.java

```
private void limparTela() {  
    this.cbItemInformacao.setSelectedIndex(-1);  
    this.lblTipoConhecimentoValor.setText("");  
    this.listTipoInformacao.setListData(new Vector());  
  
    this.listaDestino.setListData(new Vector());  
}
```


5.2.6 Padrão de Diretórios

A Figura 20 ilustra a estrutura de diretório atual do ProKnowledge, onde na pasta *configuration* estão todos os arquivos de configuração; na pasta *images* todas as imagens dos botões tal qual os logotipos e ícone do sistema. Na pasta *library* estão todas as bibliotecas utilizadas, como por exemplo, as bibliotecas de envio de email. E na pasta *source* está localizado todo o código fonte do ProKnowledge.

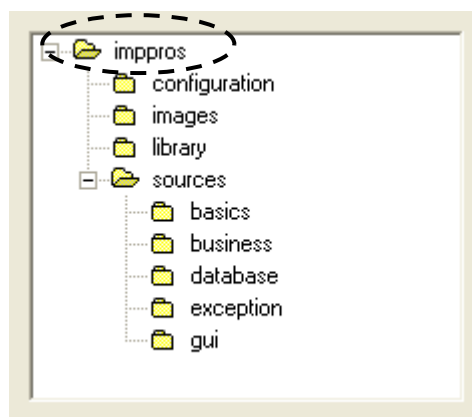


Figura 20. Estruturação dos Diretórios do ProKnowledge e das Ferramentas de Suporte

5.3 Considerações Finais

Foram mostradas evidências da aplicação do modelo proposto dentro do ambiente ImPProS, mas precisamente na ferramenta ProKnowledge. Validando o modelo proposto através de exemplos práticos.

C

ONCLUSÃO

Neste trabalho foram apresentadas as características do software, sua história, passando pela a sua importância conquistada perante a sociedade. Passando pela “crise” do software onde a preocupação pela qualidade começou a ser levada a prática.

Foram discutidos aspectos de engenharia de software: padrões de software, arquitetura, modelagem de dados, boas práticas e etc., dentro do contexto do ImPProS.

O ImPProS foi apresentado de forma superficial, visto que é um ambiente muito grande e complexo. Mas foram descritas as principais características e idéias que estão por trás dele.

Foi apresentado a fundo técnicas de padronização e integração de sistemas. Onde foram mostradas várias atividades simples para se obter um bom grau de padronização e que ajudam e muito o dia-a-dia do desenvolvedor de software, trazendo grande qualidade ao produto final e satisfação do cliente.

A ferramenta ProKnowledge que foi base para a validação da proposta e foi desenvolvida em paralelo com a escrita deste trabalho trouxe resultados melhores que o esperado. A ferramenta foi concebida com bastante qualidade trazendo muita alegria. Com um bom grau de padronização facilitando a vida de futuros colaboradores do ImPProS como também, dos futuros clientes, que terão uma ferramenta de qualidade, robusta, com interface simples, amigável e intuitiva.

Foram encontradas algumas dificuldades durante todo o projeto que foram sendo ultrapassadas com estudo.

TRABALHOS FUTUROS

Foi definida uma proposta de modelo de padronização e integração. Esta proposta foi validada em uma ferramenta grande porém não apresenta todas as características possíveis de um sistema.

Como trabalho futuro é interessante o refinamento desta estrutura obedecendo aspectos organizacionais, processos da empresa. Deixando o modelo o mais flexível possível.

A aplicação do modelo às ferramentas do ImPProS.

BIBLIOGRAFIA

[APP 00] APPLETON, B. Patterns and Software: Essential Concepts and Terminology. <http://www.enteract.com/~bradapp/>

[ALE 77] ALEXANDER, C. et al. A Pattern Language. New York: Oxford University, 1977.

[BUS 96] BUSCHMANN, F. et al. Pattern-Oriented Software Architecture: A System of Patterns. New York: John Wiley & Sons, 1996.

[FOW 97] FOWLER, M. Analysis Patterns: Reusable Object Models. Menlo Park, CA: Addison Wesley Longman, 1997.

[GAM 94] GAMMA, E. et al. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison Wesley, 1994.

[PRE 95] PREE, W. Design Patterns for Object-Oriented Software Development. Reading, MA: Addison Wesley, 1995.

[TOF90] Toffler, A., Powershift, Bantan, 1990.

[TOF80] Toffler, A., The Third Wave, Morrow, 1980.

[STO89] Stoll, C., The Cuckoo's Egg, Doubleday, 1989.

[OSB79] Osbourne, A., Running Wild – The Next Industrial Revolution, Osborne/McGraw-Hill, 1979.

[FEI83] Feigenbaum, E. A. e B. McCorduck, The Fifth Generation, Addison-Wesley, 1983.

[Javadoc.05] Javadoc Tool, <http://java.sun.com/j2se/javadoc/index.jsp>

Brown, A., Earl, A., Mcdermid, J. (1992) "Software Engineering Environments: Automated Support for Software Engineering", London: McGraw-Hill;

Brown, A. W., Carney, D. J., Morris, E. J., Smith, D. B, Zarrella, P. F. (1994) "Principles of CASE Tool Integration", Software Engineering Institute, Oxford University Press;

Jorgensen, S. A. (1994) "An Object-Oriented Approach to Tool Integration in an Integrated CASE Environment", M.S. Thesis Electrical and Computer Engineering, University of New Mexico;

Morris, E., Long, F. (1993) "An Overview of PCTE: A Basis for a Portable Common Tool Environment", Technical Report CMU/SEI-93-TR-1, Software Engineering Institute, Pittsburg, USA;

Oliveira, S. R. B, Vasconcelos, A. M. L., Rouiller, A. C. (2002) "The Integration Philosophy of ToolManager: a Tool for Management of CASE Tools" published in CSITeA-02 conference proceedings - ACIS International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications, Foz do Iguaçu-PR;

Object Management Group (1998) "XML Metadata Interchange (XMI) Specification", OMG Document ad/98-10-05. Framingham, MA: Object Management Group;

Object Management Group (2000) "Meta Object Facility (MOF) Specification", Version 1.3. Framingham, MA;

Pressman, R. S. (2004) "Software Engineering: A Practitioner's Approach", 6th edition, MacGraw-Hill International Edition;