UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA
CURSO DE ESPECIALIZAÇÃO EM WEB E SISTEMAS DE INFORMAÇÃO

ELIANARA CORCINI LIMA

Desenvolvimento de Sistemas em Multi-Camadas

Monografia de Conclusão de Curso apresentada como requisito parcial para a obtenção do grau de Especialista

Prof. Drª. Maria Lúcia Blanck Lisbôa
Orientadora

Prof. Dr. Carlos Alberto Heuser
Coordenador do Curso

Porto Alegre, dezembro de 2003

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Lima, Elianara Corcini
Desenvolvimento de Sistemas Multi-Camadas / Elianara Corcini Lima – Porto Alegre: Curso de Especialização em WEB e Sistemas de Informação, 2003.
37.:il.
Monografia (especialização) – Universidade Federal do Rio Grande do Sul. Curso de Especialização em WEB e Sistemas de Informação, Porto Alegre, BR-RS, 2003. Orientadora: Maria Lúcia Blanck Lisbôa.
1. Sistemas em Multi-Camadas. 2. Arquitetura de Software. 3. Padrões. I. Lisbôa, Maria Lúcia Blank
II. Título.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

CPD	Centro de Processamento de Dados
OSI	International Standardization Organization
SAI	Secretaria de Avaliação Institucional
TOTE	Test-Operate-Test-Exit
UFRGS	Universidade Federal do Rio Grande do Sul
UML	Linguagem de Modelagem Unificada

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho é uma proposta de uma arquitetura de sistema multi-camadas tendo como referência um estudo de caso.

Para introdução do tema faz um estudo sobre arquitetura de software e a importância de padrões de arquitetura e de projetos.

Descreve o estudo de caso e apresenta uma proposta de reestruturação utilizando uma arquitetura multi-camada.

Palavras-chave:
Sistema multi-camadas, Arquitetura de Software, Padrões

Multi-level System Development

ABSTRACT

This text focus on a multi-level system architecture proposal based on a case study.

It starts describing software architectures and the importances of design patterns and architecture patterns .

Finally it presents the case study and presents a restructuring proposed using multi-level.

Keywords:
Multi-level System, Software Architecture, Patterns

1 INTRODUÇÃO

Este trabalho tem como objetivo principal ser um momento de estudo sobre novas formas de desenvolvimento de sistemas. Um momento para parar, olhar e estudar novas alternativas.

A escolha do tema sistemas multi-camadas vem da experiência diária de desenvolvimento de sistemas. Os sistemas, ciclicamente, precisam ser reescritos, aperfeiçoados, melhorados, atualizados. Tarefa esta, normalmente, árdua, de alto custo econômico e de longo tempo de duração.

A tecnologia avança, modifica, transforma mas a lógica do negócio tende a permanecer a mesma na sua essência. É importante, então, que se tenha formas de preservar a essência e incorporar os avanços.

O tema escolhido para estudo procura atender dois requisitos: ser simples para servir como base de estudo e ser avaliado para desenvolvimento em multi-camadas.

Este trabalho apresenta, no Capítulo 2, conceituação sobre arquiteturas de software e enfatiza a importância do desenvolvimento de padrões. Expõe atributos de qualidade e sua relação com arquitetura de software. No Capítulo 3, apresenta o estudo de caso escolhido, descrevendo seu funcionamento e avaliação. No Capítulo 4, apresenta a proposta de reestruturação do sistema em uma arquitetura multi-camadas e avaliação da mesma.

2 ARQUITETURAS DE SOFTWARE

2.1 Conceitos e terminologia

A palavra arquitetura na sua definição está associada a área de construção de habitação ou espaço urbano. O conceito, segundo (FERRREIRA, 1986), " é a disposição das parte ou elementos de um edifício ou espaço urbano; os princípios, as normas, os materiais e as técnicas utilizadas para criar o espaço arquitetônico.".

Assim arquitetura é o projeto em um nível macro. Neste nível, é possível a visualização do todo, os componentes ou partes sem a preocupação de como será executado. Permite a compreensão do projeto sem uma especificação detalhada de cada componente mostrando como as partes se comunicam, interagem.

A disseminação do uso da computação e utilização em vários segmentos das atividades humanas, a integração de diferentes sistemas faz com que os softwares atuais sejam maiores e mais complexos do que no início da sua história. Estes sistemas devem responder mais facilmente às mudanças tecnológicas ou com a maior eficácia possível. Neste contexto a idéia de existir um estudo sobre a utilização do conceito arquitetura se desenvolve.

Arquitetura de Software é uma área de estudo na Engenharia de Software. Hoje, ela ainda não possui um conceito que seja aceito por todos. Os vários autores abordam a necessidade de se definir uma estrutura que permita conhecer o sistema como um todo e suas propriedades – e este conjunto definiria uma arquitetura de software. Apesar de não existir unanimidade conceitual, existe consenso da importância de seu estudo.

Na área de Engenharia de Software, as metodologias de projeto - como análise estruturada e orientação a objetos, entre outros - têm a sua importância reconhecida há muito tempo. A preocupação com a arquitetura de software é mais recente. Trabalhos relacionados a arquiteturas de domínio específico, linguagens de descrição de arquiteturas, linguagens de interface de módulos, padrões de projeto ('design patterns') são evidências desta preocupação com o estilo de arquitetura no processo de desenvolvimento de software (LISBÔA, 1997).

A definição, "a arquitetura é definida como a organização fundamental de um sistema, embutida em seus componentes, suas relações entre si e o ambiente e dos princípios que governam seu projeto e evolução. " (SEI, 2003, 2)

Esta definição tem como objetivo abranger a variedade de termos reconhecidos e seus elementos comuns.

Em resumo, a arquitetura de software se preocupa com a organização estrutural de um sistema como a composição de componentes, estruturas de controle globais, protocolos de comunicação, sincronização e acesso de dados. Também se preocupa com os elementos de funcionalidade do projeto: distribuição física, composição dos

elementos do projeto, escalonamento e execução, evolução e seleção de alternativas de projeto.

Os componentes de um sistema são, por exemplo, clientes e servidores, banco de dados, filtros e camadas em um sistema hierárquico. Interações podem ser simples como chamada de procedimentos e acesso a variáveis compartilhadas mas podem ser complexas e semanticamente ricas como protocolo cliente-servidor, protocolo de acesso a banco de dados e eventos assíncronos multi-cascata.

A arquitetura permite realizar correspondência entre as requisições e elementos de construção do sistema permitindo assim tomar decisões de projeto baseadas em racionalidade.

Os modelos ou padrões de arquitetura de sistema clareiam estruturas e diferenças semânticas entre componentes e interações. Estes podem ser usados para definir grandes sistemas. Idealmente, os elementos individuais de descrições de arquiteturas são definidas independentemente para que possam ser reutilizados em diversos contextos. A arquitetura estabelece especificações para esses elementos individuais que podem ser eles mesmos refinados como subsistemas ou implementados em uma linguagem de programação convencional.

2.2 Padrões de Arquitetura

Os padrões de arquitetura são modelos de construção de conhecimento utilizados pelo homem para resolução de problemas. Quando o homem se depara com um novo problema ele procura, em sua experiência anterior, a forma de resolve-lo. Se não encontra um padrão exato de solução ou ele faz adaptações de padrões existentes ou cria novos.

Para melhor entender a forma de resolução de problemas é apresentado como o sistema nervoso resolve problemas.

> O Test-Operate-Test-Exit(TOTE) "*é um modelo cibernético proposto por Karl Pribram e adotado pela Programação Neurolingüística, para explicar como trabalha o sistema nervoso. Consiste num circuito de realimentação que compara uma informação já gravada com a informação existente(recordada) com a informação esperada(construída) e gera operações destinadas a igualar ambas.*" (SPRITZER, 1994, p. 207)

Exemplificando, o arquiteto ou projetista tem um novo projeto para realizar. Primeiro, ele compara o projeto solicitado com os já desenvolvidos. "Esse é o primeiro teste. Ocorrendo diferenças, gera-se operações destinadas a tornar a representação existente compatível com a desejada. As operações podem ser variadas de acordo com as circunstâncias e com o modo de agir de cada um". Uma das alternativas é procurar entre os colegas experiências em casos parecidos. Outra alternativa pode ser buscar na literatura da área arquiteturas que se adaptem ao problema. Então ele compara novamente o caso encontrado com o caso inicial. Se o resultado da comparação satisfizer então o processo termina senão novas comparações e novas operações são realizadas até que as representações se igualem. SPRITZER (1994)

A figura 2.1 é uma representação de TOTE.

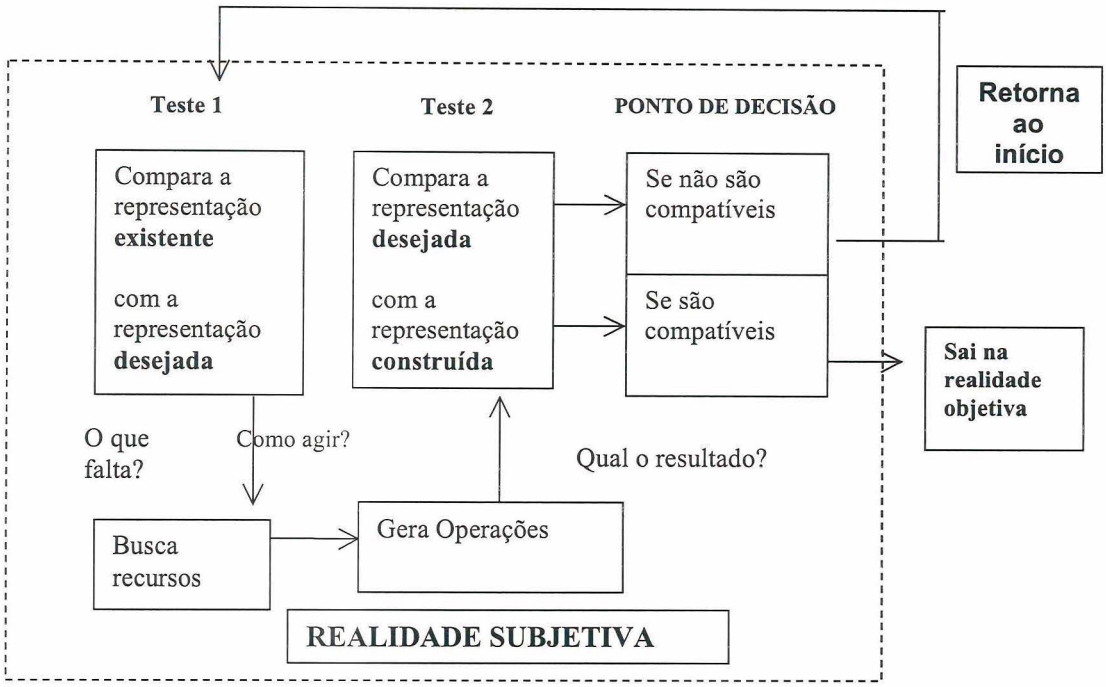

Figura 2.1: TOTE

Em várias áreas de conhecimento especialistas se utilizam de padrões de conhecimento coletivo para resolver problemas. Os padrões são utilizados como referência para solucionar problemas e não a solução dos problemas. O tema padrão é introduzido por *(BUSCHMANN et al., 1996, p. 2)* como segue:

> "*Quando especialistas trabalham em um problema particular, não é usual para eles usar um mecanismo para inventar uma solução que é completamente distinta de uma existente. Eles freqüentemente utilizam um problema similar que eles tenham resolvido e reutilizam a essência desta solução para resolver novos problemas. Este tipo de "comportamento expert", o pensamento do problema-solução, é comum para muitos diferentes domínios, como arquitetura [ALE79], economia[ETZ64] e engenharia de software[BJ94]. É um caminho natural de cópia com qualquer tipo de problema ou interação social[NS72].*".

Os padrões de arquitetura de software tem como objetivo definir esquemas de organização estrutural de sistemas computacionais. Eles fornecem um "conjunto de subsistemas predefinidos, especificando suas responsabilidades e incluem várias regras e linhas gerais para organização entre eles." (BUSCHMANN, 1996) .

Os padrões de arquiteturas estão agrupados em categorias que possuem propriedades similares e são as seguintes:

a) Da Desorganização para Estrutura (*From Mud to Structure*) - é o padrão que fornece elementos de análise para decomposição controlada de tarefas de um sistema completo em subtarefas cooperativas. Esta categoria inclui padrões como de camadas, padrões de canais e filtros.

b) Sistemas Distribuídos – esta categoria inclui um padrão *Broker* e refere dois padrões de outras categorias Microkernell e Canais e Filtros. O padrão *Broker* fornece uma infra-estrutura completa para aplicações distribuídas.

c) Sistemas Interativos – esta categoria compreende dois padrões: Modelo-Visão-Controlador e Controle-Abstração-Apresentação. Estes padrões fornecem elementos de análise para estruturação de sistemas de software de interação homem-computador.

d) Sistemas adaptáveis – esta categoria inclui o padrão Reflexão e o padrão Microkernel. Estes padrões suportam a extensão de suas aplicações e suas adaptações para desenvolvimento tecnológico e solicitações de trocas funcionais.

A escolha de um padrão de arquitetura depende das propriedades gerais do sistema. Para isso o sistema deve ser confrontado com os padrões definidos. Após análise, decidir qual ou quais padrões utilizar. Para alguns sistemas provavelmente será necessário a utilização de mais de um padrão. Não esquecendo sempre que os padrões devem ser utilizados como referência e não como arquitetura de software completa.

2.3 Arquiteturas Multi-camadas

" O padrão de arquitetura de camadas ajuda estruturar aplicações que podem ser decompostas em grupos de subtarefas e em que cada grupo de subtarefa é um nível particular de abstração" (BUSCHMANN, 1996)

A arquitetura em camadas é organizada hierarquicamente, onde cada camada fornece serviços para a camada de cima e é tratado como cliente da camada debaixo. Os conectores são definidos como protocolos que definem como as camadas interagem. Este conectores, usualmente, são chamadas de procedure. A restrição de topologia inclui limitação de interações entre camadas adjacentes.

A Figura 3.1, exemplifica a restrição de topologia mostrando como ocorre a comunicação entre as camadas. A Camada N se comunica com a Camada N-1 e assim por diante. Não existe uma interface de comunicação entre, por exemplo, a Camada N com a Camada 1. Assim, um dado que entre na Camada 1, com o objetivo de chegar na Camada N, deve passar por todas as camadas existentes entre a Camada 1 e a Camada N.

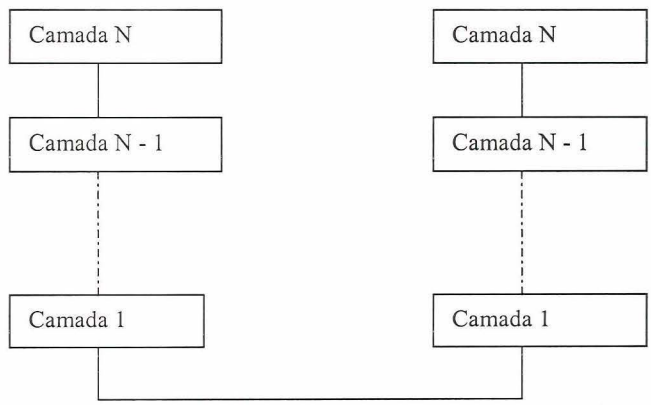

Figura 2.2: Exemplo de comunicação entre camadas

Os exemplos de sistemas mais conhecidos que utilizam esta estrutura são os de comunicação mas também utilizam este modelo os sistemas de banco de dados e sistemas operacionais.

A arquitetura do sistema operacional Windows 2000® é inspirada no modelo de micronúcleo e camadas. O sistema é dividido em módulos que são dispostos uns sobre os outros em camadas. O projeto implementa o modelo de camadas oferecendo um conjunto de serviços à camada superior e somente utiliza serviços fornecidos pela camada imediatamente inferior. (OLIVEIRA, 2001)

O exemplo mais conhecido deste modelo de arquitetura é o modelo OSI usado para comunicação, como mostra a Figura 3.2. A Figura mostra o número de camadas implementadas e a finalidade de cada uma. Exemplo extraído de (BUSCHMANN, 1996).

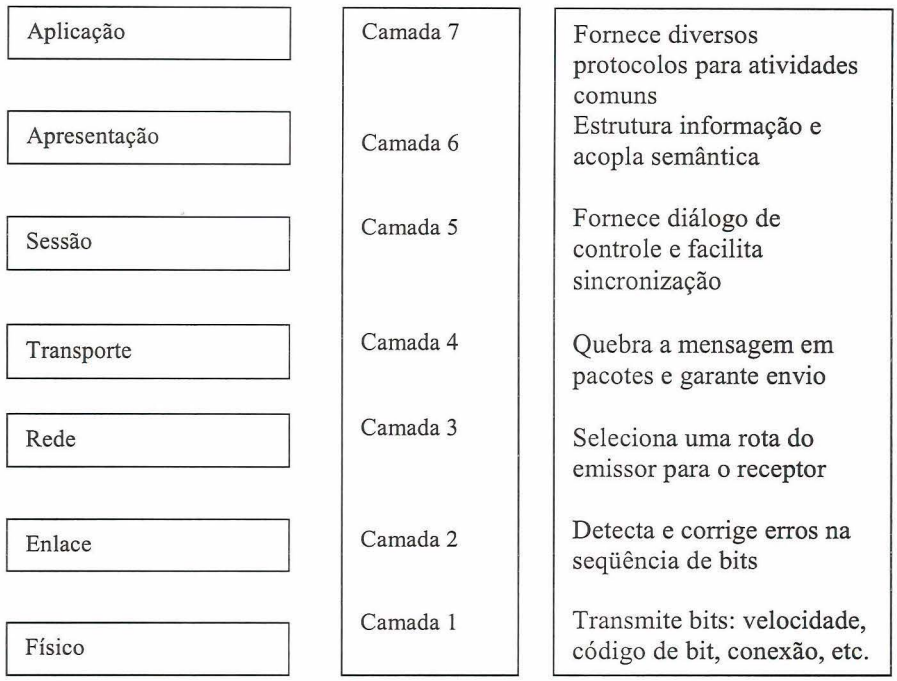

Figura 2.3: Modelo 7 camada OSI

Os seguintes passos descrevem uma forma de se definir a arquitetura de camadas de um sistema. Este passos nem sempre são válidos para todos os sistemas. Eles são apresentados como forma de referência de organização do conhecimento. Estes passos foram retirados de (BUSCHMANN, 1996).

Definir o critério de abstração para agrupar tarefas dentro de camadas : Este critério é freqüentemente a distância conceitual da plataforma. Algumas vezes você encontra outros paradigmas de abstração, por exemplo o grau de customização de domínios específicos ou o grau de complexidade conceitual.

Determinar o número de abstrações de níveis de acordo com seu critério de abstração; cada nível de abstração corresponde a um nível de camada. Algumas vezes este mapeamento não é óbvio. Muitas camadas podem impor uma sobrecarga desnecessária enquanto poucas camadas podem resultar em uma estrutura pobre.

Nomear as camadas e assinalar tarefas para cada uma delas: a tarefa de maior alto nível é a tarefa do sistema total, como percebido pelo cliente. A tarefa de todas as outras camadas são para ajudar a camada de mais alto-nível.

Especificar os serviços: o princípio mais importante de implementação é que as camadas estão estritamente separadas uma das outras, isto é, que nenhuma camada pode se espraiar para as demais.

Refinar a camada : iteração dos passos de 1 até 4. Não é possível definir um critério de abstração preciso antes de pensar sobre a implicação de camadas e seus serviços. Alternativamente, é usualmente errado definir componentes e serviços primeiro e mais tarde definir uma estrutura de camada de acordo com as suas relações(trocas de dados).

Então se faz necessário executar os primeiros passos verificando a independência entre as camadas.

Especificar uma interface para cada camada : isto permite que a camada possa ser vista como uma "caixa-preta", permitindo uma evolução melhor do sistema.

Estruturar as camadas individuais: o foco normalmente está na relação entre camadas, mas as vezes as camadas individualmente podem estar caóticas. Faz-se necessário verificar a complexidade da camada e, quando necessário, dividi-la em componentes separados.

Especificar a comunicação entre camadas adjacentes.

Separar camadas adjacentes.

Projetar uma estratégia de tratamento de erros: o tratamento de erros pode ser muito custoso para as camadas da arquitetura com respeito ao tempo de processamento e notável esforço de programação. Um erro pode ser tratado na camada onde foi encontrado ou passado para o próximo nível mais alto. No último caso, a camada mais baixa deve transformar um erro em um erro com significado para a camada mais alta.

As vantagens do uso de padrões de camadas, segundo (BUSCHMANN, 1996).

Incremento de níveis de abstração – permite ao projetista dividir um problema complexo em uma seqüência de passos incrementais

Reuso de camadas – a camada pode ser utilizada em mais de um contexto

Suporte para padronização – camadas claramente definidas permitem a padronização de tarefas e interfaces.

Dependências são armazenadas localmente – padronização de interfaces entre camadas usualmente confinam o efeito de troca na camada dentro dela mesma.

Adaptabilidade – implementação de camadas individuais podem ser trocadas por implementações semanticamente equivalentes sem afetar as demais.

As desvantagens do uso de padrões de camadas são , segundo (BUSCHMANN, 1996):

- Cascata de troca de procedimentos – um sério problema pode ocorrer quando os procedimentos de uma camada são alterados, isto é, quando uma alteração em uma camada pode ter forte impacto nas demais fazendo com que outras camadas necessitem ser trabalhadas.

- Baixa eficiência – uma arquitetura de camadas é usualmente menos eficiente que uma estrutura monolítica. A sua eficiência depende do número de camadas e do número de transformações que as mensagens sofrem em cada camada.

- Trabalho desnecessário – se alguns serviços executados por camadas baixas são excessivamente executados ou tem trabalho duplicados para o interesse da última camada, isto terá um impacto negativo na execução do sistema.

- Dificuldade de estabelecer a correta granularidade de camadas – a arquitetura de padrões com poucas camadas não explora todo o potencial para reutilização, trocabilidade e portabilidade. De outro lado, muitas camadas introduzem desnecessária

complexidade, sobrecarga na separação de camadas e a transformação de argumentos e retorno de valores.

A decisão sobre a granularidade das camadas e o assinalamento de tarefas para as camadas é difícil, mas é critico para a qualidade da arquitetura.

Segundo (TINDALL, 2000) existe um esforço da indústria para sair de um modelo monolítico para aplicações modulares. Essa experiência demonstra "que há um particionamento lógico de funcionalidade em grupos distintos conhecidos como camadas".

A divisão em camadas permite a distribuição de processamento através de múltiplas máquinas e também um alto nível de modularidade e capacidade de manutenção de código. A divisão em camadas ou níveis mais comuns de se encontrar em aplicações são: de apresentação, lógica da aplicação ou de negócio e camada de dados, como segue:

Camada de apresentação - a camada de apresentação é responsável por implementar componentes de interface com o usuário como navegação e apresentação. É através desta camada que o usuário interage com a aplicação. A interface pode ser num contexto de janelas ou ser através de um navegador ou a forma que permita dar a funcionalidade desejada.

Camada de lógica da aplicação ou de negócio - é a camada que contém as regras de negócios do sistema. Nesta camada está contida todo conhecimento do domínio para a qual a aplicação é desenvolvida. A camada de lógica da aplicação é considerada a parte central do sistema. Ela se comunica tanto com a camada de apresentação como a de dados.

Camada de dados - a camada de dados tem como função recuperar e armazenar dados da aplicação. Estes dados podem estar armazenados em banco dados, arquivos simples, XML ou qualquer outra forma de armazenamento de dados.

2.4 Arquitetura e sua relação com atributos de qualidade

A arquitetura de software define a organização geral do sistema. Para poder medir o comportamento do sistema diante de dificuldades e avaliar seu comportamento, o arquiteto tem a sua disposição os atributos de qualidade.

Os atributos de qualidade, também, são chamados de propriedades não funcionais. Estes atributos tem impacto sobre a qualidade de software.

A definição da arquitetura leva em conta os requisitos mostrados na figura. A Figura 2.4 foi extraído de (SEI 2003).

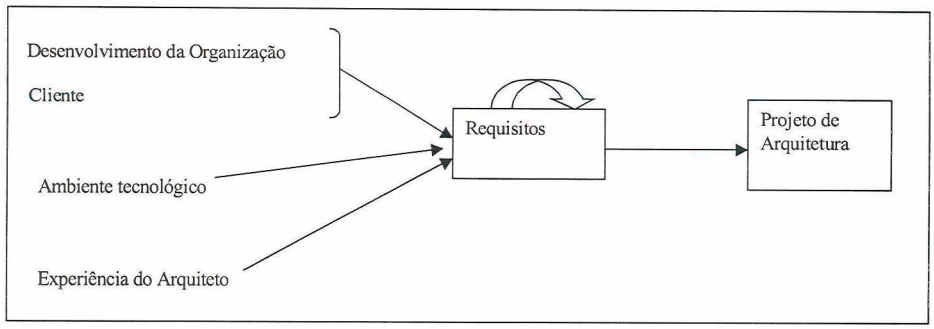

Figura 2.4: Requisitos para definição de projeto

Após apropriação do conhecimento dos requisitos levantados da análise o arquiteto tem condições de definir a arquitetura do sistema. O arquiteto pode analisar os padrões de arquitetura disponíveis e decidir pelo padrão mais adequado para seu projeto. Em alguns casos, o arquiteto poderá optar por ter em seu projeto mais de um padrão arquitetônico.

Para validação do modelo arquitetônico escolhido o arquiteto tem a sua disposição os atributos de qualidade. Estes atributos podem ser utilizados com objetivo de verificar se a escolha da arquitetura preenche da melhor maneira possível os requisitos desejados. Os atributos de qualidade, pela complexidade do problema, normalmente, poucos são atendidos. É importante procurar analisar e atender os atributos de qualidade que fortifiquem o objetivo do projeto. Estes objetivos podem ser de ter qualidade, de atender o negócio ou atender as pessoas que trabalham com o sistema. Estes objetivos nem sempre convergem.

O impacto dos atributos de qualidade no desenvolvimento de software verifica-se, por exemplo, no sistema operacional Windows 2000®. Os atributos ditam a arquitetura, o projeto e seu desenvolvimento. Os atributos orientadores do projeto são: primeiro, confiabilidade e robustez traduzidos no fato de que um sistema deve ter a capacidade de se proteger dos próprios erros e erros externos; segundo, adaptabilidade permitindo a evolução, adaptação as novas necessidade tanto de hardware como software; terceiro, portabilidade para permitir que um sistema seja empregado em diversos ambientes; quarto, interoperabilidade com outros sistemas operacionais. (OLIVEIRA, 2001)

Os atributos de qualidade descritos aqui são: adaptabilidade, interoperabilidade, eficiência, confiabilidade, testabilidade e reutilização. (BUSCHMANN, 1996)

2.4.1 Adaptabilidade

Os sistemas industriais e comerciais normalmente tem uma longa duração. Ao longo do tempo eles vão sofrendo alterações para se adaptar a novas exigências. Se faz necessário então pensar, no momento da construção do sistema escolher uma arquitetura que permita modificações e evolução.

Quatro são os aspectos da modificabilidade:

- Manuteniblidade – este aspecto tem a ver com determinação de problemas, reparando um erro ocorrido no sistema. Uma arquitetura de software definida considerando este aspecto tende a permitir que alterações sejam minimizadas para os demais componentes.
- Extensibilidade – este aspecto permite a extensão de novos componentes, bem como a troca de componentes não afetando ou afetando o mínimo possível a estrutura.

- Reestruturação – este aspecto permite a reorganização de componentes e a relação entre eles. A reestruturação de um sistema de software necessita um projeto cuidadoso de relações entre os componentes. Idealmente, o projeto deveria permitir a configuração de componentes sem afetar a maior parte dos componentes.
- Portabilidade – este aspecto tem haver com a adaptação do software as diversas plataformas de hardware, interfaces de usuário, sistemas operacionais, linguagens de programação ou compiladores.

2.4.2 Interoperabilidade

Um sistema de software não existe independentemente. Ele está interagindo com outros sistemas ou com seu próprio ambiente. Para ter interoperabilidade, uma arquitetura de software deve ser projetada para ter acesso bem definido com as estruturas externas e estruturas de dados. A interação de um programa com sistemas de software escritos em outras linguagens de programação é um aspecto de interoperabilidade que também impacta a arquitetura de software.

2.4.3 Eficiência

Eficiência é a avaliação de uso de recursos para a execução do software, o impacto do tempo de resposta, distribuição e consumo de armazenamento. A apropriada distribuição de responsabilidade dos componentes, bem com a acoplagem são características importantes para avaliação de eficiência.

2.4.4 Confiabilidade

Confiabilidade é a capacidade de um sistema de software manter a sua funcionalidade. Esta característica é importante tanto no tratamento de erros do sistema e situações inesperadas como no seu uso incorreto.

O sistema deve ser capaz de reconhecer a falha de um componente interno e utilizar estratégias para compensar a falta.

Dois aspectos de confiabilidade são importantes: tolerância a falha e robustez.

2.4.5 Testabilidade

O software que suporta testabilidade permite melhor detecção de falhas.
Com o aumento e complexidade dos sistemas está ficando difícil e caro testar sistemas. Uma arquitetura de software deve fornecer condições para avaliação de que está correto.
As arquiteturas de software não apresentam as mesmas condições para este atributo.

2.4.6 Reutilização

Reutilização tem dois grandes aspectos: desenvolvimento de software com reutilização ou desenvolvimento para reutilização.

Desenvolvimento de software com reutilização significa reutilizar componentes existentes e resultados de projetos anteriores ou bibliotecas comerciais, análise de projetos, especificações de projetos ou código de componentes. Estes artefatos reutilizados são integrados dentro de aplicações como são ou com modificações.

Desenvolvimento de software para reutilização focaliza a produção de componentes que são potencialmente reutilizáveis em futuros projetos como parte de desenvolvimento do projeto atual.

2.5 Padrões de projeto

Enquanto padrões de arquitetura oferecem soluções para a estruturação de sistema, padrões de projeto contribuem para a sua implementação, podendo assim ser definido: "Um padrão de projeto fornece um esquema para refinamento de subsistemas ou componentes de um sistema de software ou a relação entre eles. Ele descreve uma estrutura recorrente comum de comunicação dos componentes que resolve problemas gerais de projeto dentro de um contexto particular[GHJV95]" (BUSCHMANN, 1996).

Os padrões de projeto são menores que os padrões de arquitetura. Eles são independentes de uma linguagem de programação ou modelo de programação. O uso de um padrão de projeto não tem influência sobre a estrutura de um sistema mas tem grande influência sobre um subsistema.

O uso de padrões de projeto permite a reutilização de modelos já pensados e testados. Este reuso permite ao projetista entender melhor seu projeto e contribui para o desenvolvimento de software mais confiável.

Os padrões de projeto estão assim agrupados conforme (BUSCHMANN, 1996):

Decomposição estrutural – estes padrões permitem a decomposição de subsistemas e componentes em partes cooperativas.

Organização de trabalho – estes padrões permitem a definição de componentes colaborativos juntos para resolver um problema complexo.

Controle de acesso – padrões que guardam e controlam acesso para serviços ou componentes.

Gerenciamento – estes padrões que tratam coleções homogêneas de objetos, serviços e componentes em sua totalidade.

Comunicação – estes padrões que ajudam a organizar a comunicação entre componentes.

2.6 Projeto Orientado a Objeto

Padrões de projeto são importantes para a implementação de sistemas por permitirem um refinamento dos subsistemas ou componentes bem como a comunicação entre as partes envolvidas .

O modelo de orientação objeto descreve a estrutura de objetos de um sistema e o relacionamento entre eles. Um objeto tem definido dentro de seu escopo, seus atributos e funcionalidades.

A referência a objetos é o mundo real. O mundo real, como é percebido, é composto de objetos. Os objetos, como são conhecidos, tem suas características e funcionalidades próprias. Eles possuem identidade e funcionamento próprio. Através de suas funções se comunicam com outros objetos.

Segundo (RUMBAUGH, 1994) , definimos um objeto como um conceito, uma abstração, algo com limites nítidos e significado em relação ao problema em causa. Os objetos servem a dois objetivos: facilitar a compreensão do mundo real e oferecer uma base real para a implementação em computador. A decomposição de um problema em objetos depende do julgamento e da natureza do problema. Não existe apenas uma representação correta.

A vantagem deste modelo é o desenvolvimento de sistemas menores, através de reuso de componentes e maior estabilidade por estar mais próximo da realidade modelada (HEUSER, 2002).

3 ESTUDO DE CASO

3.1 Apresentação

Para análise de padrão de arquitetura e projeto foi escolhido o Sistema de Avaliação de Disciplina e Professor , desenvolvido pelo Centro de Processamento de Dados, por solicitação da Secretaria de Avaliação Institucional, órgãos da Universidade Federal do Rio Grande do Sul.

A avaliação da disciplina e do professor pelo aluno tem como objetivo fornecer informações ao curso para melhor planejamento de sua atividades e, também, uma reflexão sobre o ensino junto com os seus docentes.

O projeto surge quando a Faculdade de Agronomia entra em contato com a SAI solicitando o desenvolvimento de um sistema que automatizasse esta atividade. As dificuldades levantadas em manter um sistema manual foram o pequeno número de pessoas treinadas para realizar a atividade de organização, distribuição de formulários, recolhimento dos formulários, digitação das respostas, organização das respostas e geração de relatórios de análise. A SAI se diz favorável ao desenvolvimento do projeto ressalvando a necessidade de o modelo atender, não somente o curso de Agronomia mas sim a todos os cursos da UFRGS. Estabelecido o acordo, a SAI contata com o CPD, para o desenvolvimento de um projeto piloto.

O projeto inicia envolvendo mais cursos da UFRGS. Uma série de reuniões com os cursos, o CPD e a SAI é realizada. Logo no começo verifica-se a existência de cursos que realizam a avaliação e outros não. Os cursos que realizam avaliação têm, cada um, seus critérios de avaliação focados na sua realidade. Após algumas reuniões combina-se que o sistema deve permitir aos cursos terem o seu conjunto de critérios de avaliação próprios mas deve existir uma estrutura de avaliação comum a todos. Outra questão levantada é a não existência de uma escala de avaliação padrão. Decide-se utilizar uma escala de padrão única para todos os cursos. A utilização de uma escala única permite criar um sistema padrão para toda a Universidade, bem como, permitir a SAI ter um parâmetro de comparação entre os cursos.

A Figura 3.1 mostra o caso de uso do sistema permitindo uma visão geral dos atores envolvidos e dos processos.

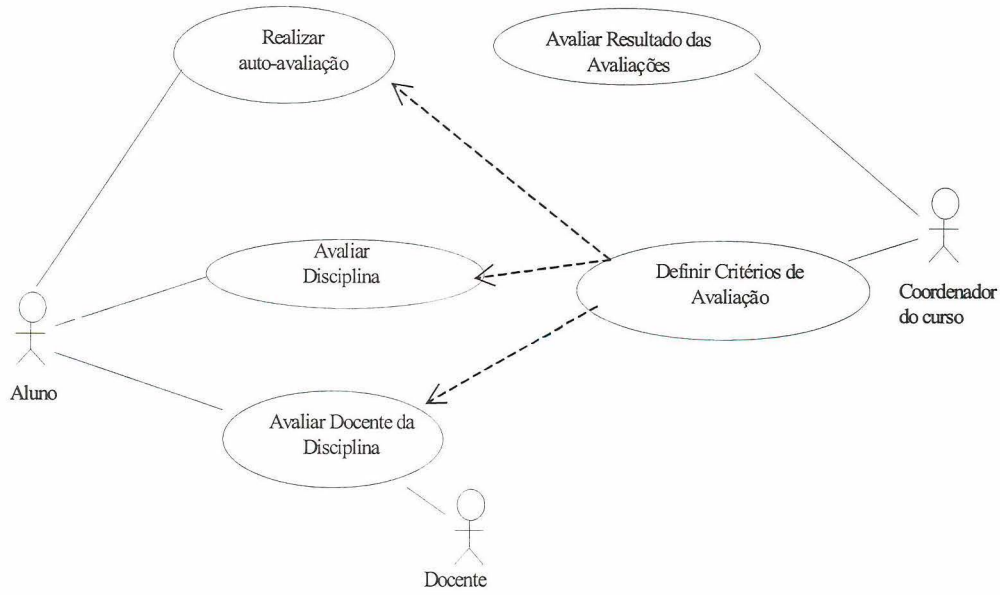

Figura 3.1: Caso de Uso – Visão geral

Tabela 4.1: Descrição Casos de Uso

Caso de Uso	Atores	Finalidade
Avaliar Disciplina	Aluno	O Aluno responde questões sobre a disciplina como planejamento, desenvolvimento. O Aluno somente avalia a disciplina/turma que esta assistindo.
Avaliar Docente da Disciplina	Aluno, Docente	O Aluno assiste disciplina ministrada pelo Docente. O Aluno avalia o Docente da disciplina que assiste. A disciplina pode ser ministrada por mais de um Docente. No caso de uma disciplina possuir mais de um Docente ministrando a disciplina, a avaliação deve ser feita individual, isto é, para cada um dos Docentes separados. O Aluno somente avalia a disciplina/turma que esta assistindo.
Realizar auto-avaliação	Aluno	O aluno faz uma avaliação de si mesmo sobre como foi seu aprendizado na disciplina. Os critérios são definidos pelo coordenador do curso.
Avaliar Resultado das Avaliações	Coordenador do curso	Relatórios estatísticos sobre as avaliações para o Coordenador do curso. Relatórios sobre a avaliação da disciplina, do docente, auto-avaliação.
Definir critérios de avaliação	Coordenador do curso	O Coordenador do curso define as questões de avaliação que são respondidas pelo aluno. Os critérios podem variar de semestre para semestre. Cada Coordenador de curso pode definir os critérios de seu curso.

O sistema deve atender os seguintes requisitos:

1. O formulário de avaliação deve conter os seguintes itens:
   - Escala de avaliação com valores de 1 a 5. Os valores eqüivalem a avaliação de discordo plenamente a concordo plenamente;
   - Identificação da disciplina permitindo ao aluno identificar a disciplina que está avaliando;
   - Apresentar o nome dos professores da disciplina que está sendo avaliada e permitir a identificação do professor que está sendo avaliado;
   - Critérios de avaliação do professor identificadas por um grupo de avaliação. Cada professor deve ser avaliado individualmente;
   - Critérios de avaliação da disciplina em geral como, por exemplo, planejamento, desenvolvimento, etc;
   - Avaliação da infra-estrutura;
   - Auto-avaliação do aluno para a coordenação saber como o ele se avalia em relação ao aprendizado proposto na disciplina; e
   - Um espaço aberto para o aluno poder tecer comentários considerado por ele importante e que não foram objeto de avaliação nos critérios definidos.
2. Um aluno responde um questionário por disciplina.
3. O aluno deve avaliar somente as disciplinas que está assistindo.
4. O aluno deve avaliar somente os professores das disciplinas que está matriculado.
5. O aluno deve ser estimulado a preencher todos os itens de avaliação.
6. Permitir a coordenação do curso ter seus critérios próprios de avaliação.
7. Emitir relatório estatístico sobre as respostas dos alunos.

3.2 Descrição

O sistema foi desenvolvido utilizando arquitetura multi-camada, cliente-servidor. A arquitetura cliente-servidor se baseia na relação entre dois programas de computador onde um faz uma solicitação, o cliente, e ou outro programa responde ao pedido, o servidor. O conceito se baseia na idéia de que existe um lado, o cliente, que usa os serviços e o outro lado que fornece os serviços, servidor.

A Figura 3.2 mostra como a arquitetura foi implementada.

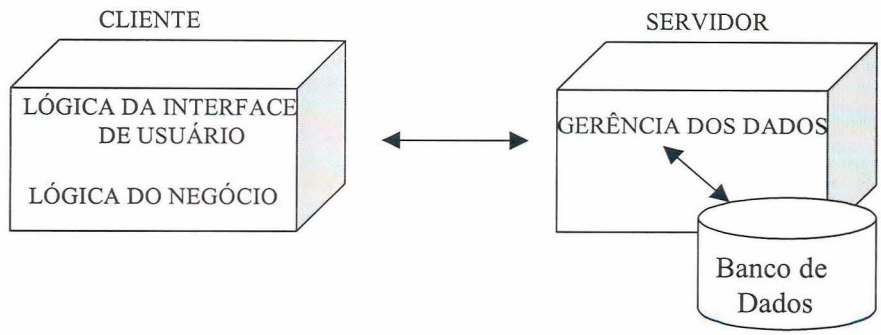

Figura 3.2: Arquitetura cliente-servidor

O sistema usa banco de dados relacional . A interface com o aluno é através de formulário eletrônico. A conexão com o banco de dados faz parte do código do formulário.

A metodologia de trabalho utilizada é fortemente centrada na organização dos dados. Primeiro se realiza a modelagem do dados, definindo as tabelas e seus relacionamentos. Após a definição das funcionalidade do sistema.

O diagrama de atividades mostra graficamente as etapas percorridas pelo sistema para que a avaliação seja executada integralmente.

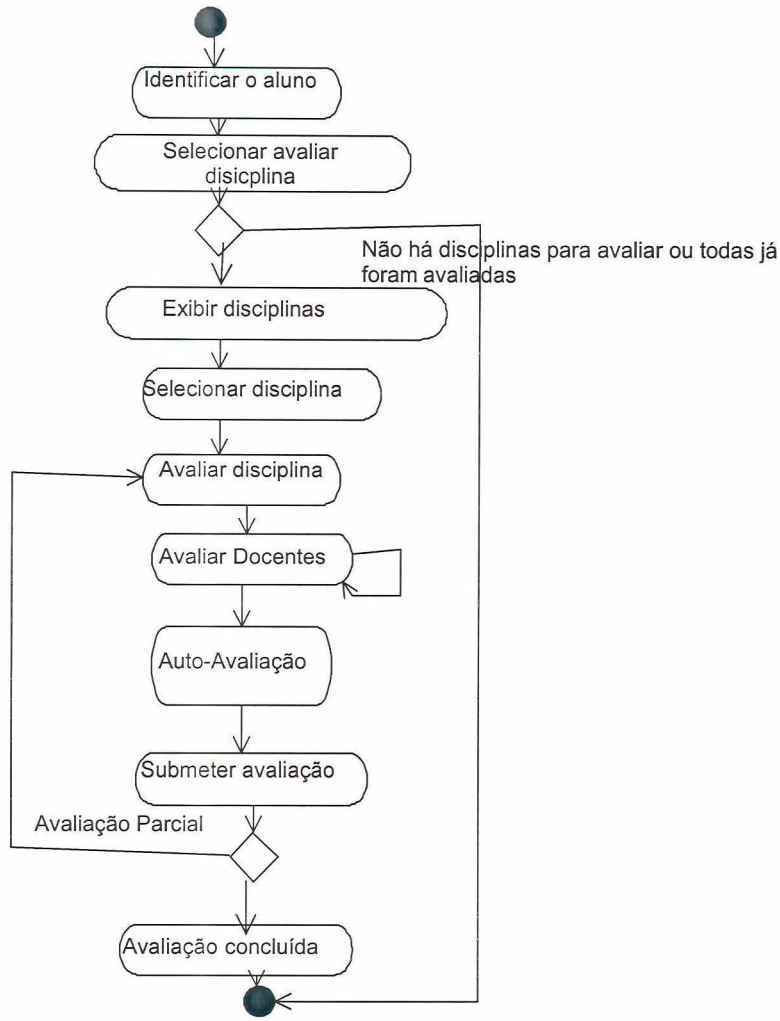

Figura 3.3: Diagrama de atividades

Após a identificação do aluno são apresentados os serviços oferecidos pelo Portal do Aluno. Ele deve, então, escolher Avaliação de Disciplina. Na seqüência são exibidas as disciplinas possíveis de avaliação pelo aluno. As disciplinas possíveis de avaliação são todas aquelas que o aluno está vinculado. As disciplinas já avaliadas não são passíveis de escolha pelo aluno.

A seqüência de formulários representa graficamente o que está descrito no diagrama de atividades, Figura 3.3.

O Formulário Avaliação é apresentado a seguir permitindo melhor entendimento de sua estrutura.

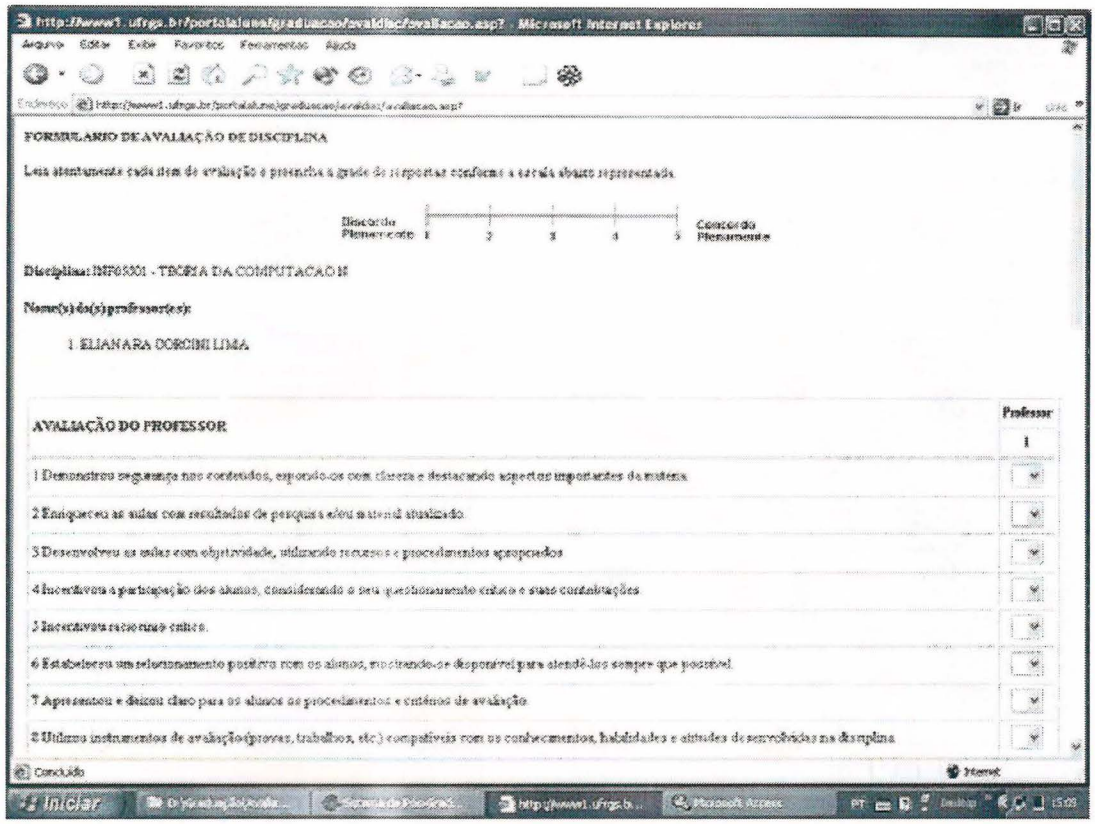

Figura 3.4: Formulário de disciplinas a avaliar

A Figura 3.4 mostra o Formulário de Avaliação apresentado ao aluno para ser preenchido. Ele contém a escala de avaliação, os professores a serem avaliados. São apresentados todos os professores que ministram a disciplina. Os itens de avaliação, Avaliação do Professor, Avaliação da Disciplina, Avaliação da Infra-estrutura e Auto-avaliação são definidos pelo Coordenador do Curso. Isto quer dizer que o Coordenador da Matemática pode definir critérios próprios e podem ser diferentes dos definidos pelo Coordenador do Curso de Física. Os critérios podem ser modificados ao longo do tempo. O sistema mantém história dos critérios definidos.

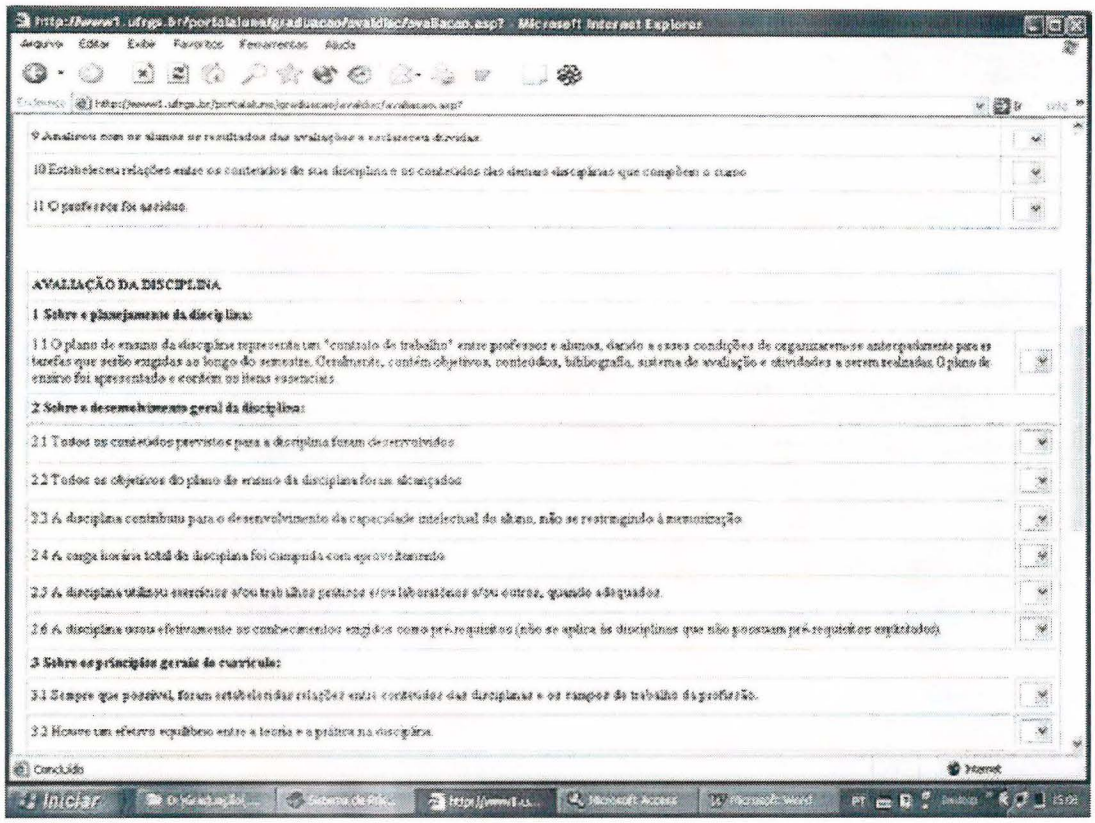

Figura 3.5: Formulário de disciplinas a avaliar (continuação)

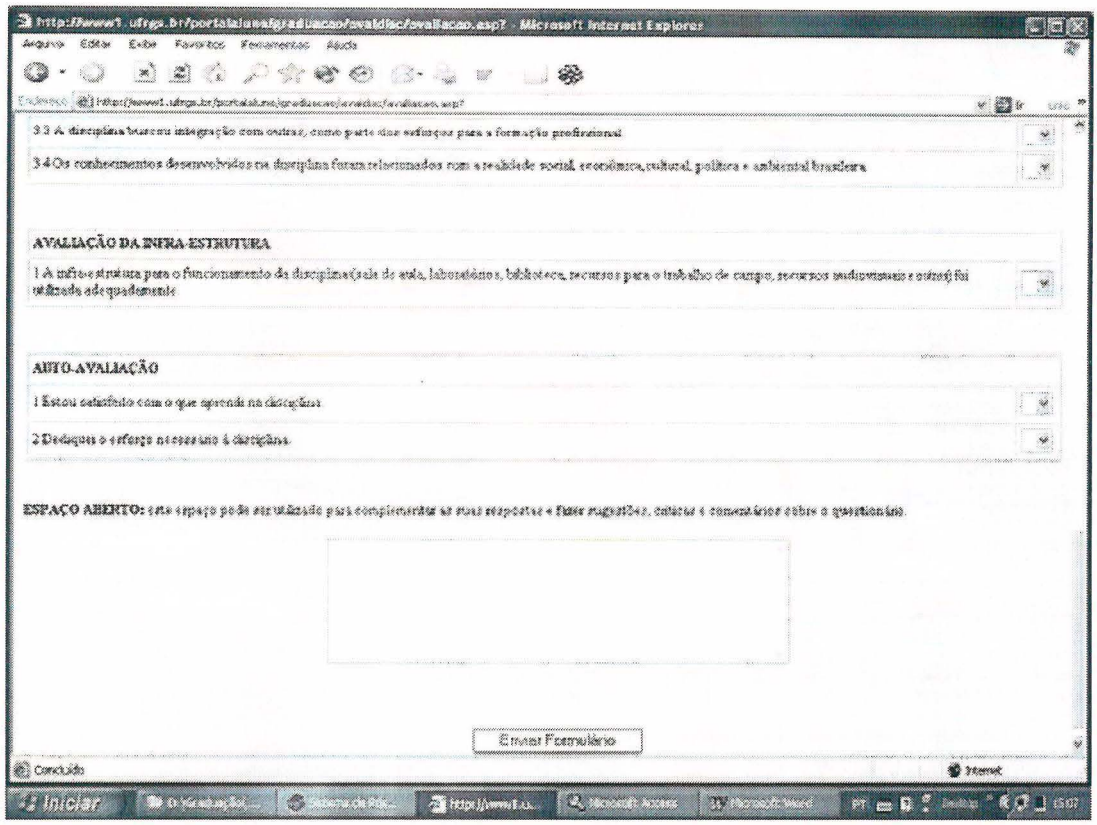

Figura 3.6: Formulário de disciplinas a avaliar (continuação)

Modelo de dados representa a organização dos dados, em forma de tabelas, no banco de dados.

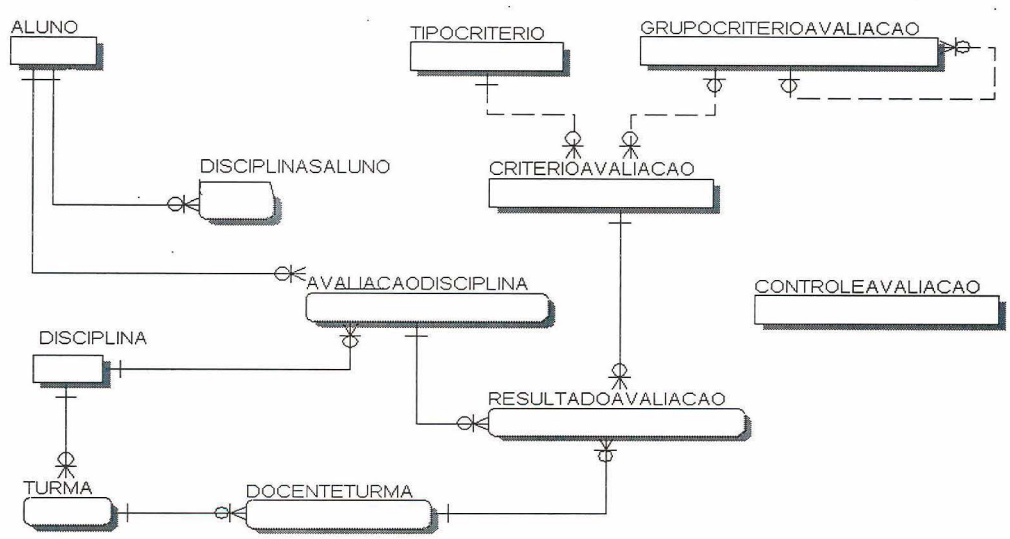

Figura 3.7: Modelo de dados

Pelo modelo de dados pode-se observar a organização dos dados. As tabelas TIPOCRITERIO, GRUPOCRITERIO e CRITERIOAVALIACAO contém os dados dos critérios de avaliação e sua organização em grupos como Avaliação da Disciplina, Avaliação do Docente, Auto-Avaliação e Infra-Estrutura.

As tabelas ALUNO, DISCIPLINASALUNO, DISCIPLINA, TURMA, DOCENTETURMA permitem saber as disciplinas cursadas pelo aluno bem como os seus docentes.

AVALIAÇÃODISCIPLINA e RESULTADOAVALIACAO são tabelas responsáveis por manter as respostas das avaliações realizadas pelos alunos e controle das avaliações permitindo saber se a avaliação foi total ou parcial.

3.3 Avaliação

As vantagens desta implementação são o domínio por parte dos projetistas e dos desenvolvedores do modelo e a utilização de técnicas e ferramentas conhecidas. O conhecimento das técnicas e ferramentas permite uma a implementação mais rápida.

A desvantagem desta implementação é o acoplamento existente entre interface do usuário e a regra de negócios. Este acoplamento pode causar:

1. Falha de segurança pois permite que existam múltiplos pontos de acesso ao banco de dados. Para resolver esta falha seria vantajoso estabelecer os acessos a partir de um ponto fixo, uma classe. Esta classe possibilitaria fornecer e controlar todo o acesso ao banco. A existência de um ponto comum de acesso torna mais fácil controlar os acessos e garantir a consistência das consultas.

2. Dificuldade de padronização no acesso. As páginas de acesso podem ser implementadas por diferentes desenvolvedores. Nesse caso, cada desenvolvedor pode decidir a forma de estabelecer a conexão, quando fechar a sessão, sobre a consistência dos dados. Uma forma de resolver pode ser por políticas de padronização mas que são muito difíceis de manter .

3. Cada página estabelece uma conexão com o banco. O problema que surge é um número elevado de conexões abertas gerando uma carga alta no servidor de banco de dados.

Partindo das considerações feitas anteriormente pode-se pensar em elaborar uma estratégia de reestruturação. A estratégia de reestruturação deve contemplar os aspectos:

1. A divisão em camadas proporcionará uma abstração maior impossibilitando acessos diretos ao banco de dados.

2. A camada de acesso aos dados fornece métodos que devem ser utilizados para a obtenção dos dados.

3. Como a camada de acesso centraliza o acesso ao banco, é mais fácil controlar quem tem acesso e a quais dados (questão de segurança), controlar a conexão com o banco de dados permitindo manter uma única conexão aberta desafogando o servidor de banco de dados.

4. Maior facilidade de programação pois os desenvolvedores não necessitarão conhecer a estrutura como os dados estão armazenados nas tabelas

5. Maior flexibilidade nas atualizações pois há uma interface comum entre cada camada. Dessa forma, alterações na estrutura como os dados estão armazenados, na forma propriamente dita, ou seja, em uma base de dados relacional, orientada a objetos, XML, arquivo de texto etc, não serão sentidas pelas camadas superiores.

4 PROPOSTA DE REESTRUTURAÇÃO

4.1 Arquitetura proposta

A proposta de reestruturação do sistema é uma arquitetura multi-camada, entendendo um sistema com mais do que duas camadas.

Um sistema de informação tem os seguintes elementos: a interface com o usuário, a lógica do negócio, o armazenamento dos dados, o tratamento de falhas.

Em uma arquitetura multi-camada cliente-servidor existe a separação em duas camadas, lógica do negócio e armazenamento de dados. Na camada lógica do negócio tem-se a interface com o usuário, a lógica do negócio e o tratamento de falhas.

A proposta é uma arquitetura que possa tratar estes elementos separados dando um tratamento diferenciado para cada uma das especificidades. A arquitetura inicialmente definida pode ser conforme a figura 4.1.

A Figura 4.1 mostra uma primeira arquitetura proposta visando contemplar o que foi explanado acima.

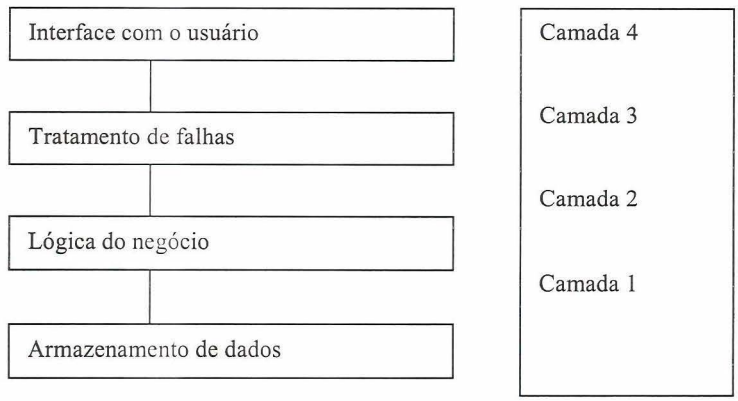

Figura 4.1: Arquitetura em quatro camadas, primeira proposta

A arquitetura em quatro camadas reflete os elementos ou componentes de um sistema de informação pensado em utilizar como armazenamento de dados banco de dados relacional e mantendo uma estreita relação entre a lógica de negócio e os dados.

A proposta deste trabalho é a utilização de orientação a objetos o que remete a necessidade de inserção, na arquitetura, de mais uma camada. Esta camada tem como funções: realizar o mapeamento de objetos para a forma de armazenamento de dados

que for utilizada e tornar transparente para as camadas superiores a forma como os dados são armazenados. A independência deve ser tal que qualquer modificação na base de dados não afete as demais. No exemplo que está sendo utilizado é um mapeamento para banco de dados relacional. A adição dessa camada permitiria trabalhar com uma base de dados em qualquer formato, banco de dados ou não. O segundo ponto a considerar que cada componente tenha como responsabilidade tratar informações que tenham a ver como o seu objetivo. Assim mais relevante se torna a existência de uma camada que logicamente faça a separação entre a lógica do usuário e o armazenamento de dados.

A arquitetura final proposta é apresentada pela figura 4.2.

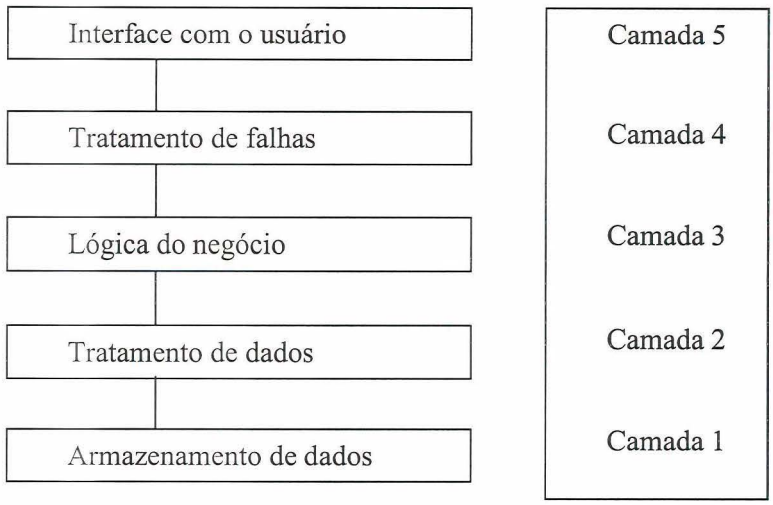

Figura 4.2: Arquitetura proposta final

A divisão nestas camadas permite a distribuição de processamento através de múltiplas máquinas e também um alto nível de modularidade e capacidade de manutenção de código. A tecnologia, ao longo do tempo especializou-se no tratamento de cada uma das áreas propostas como camada. Assim esta divisão, ainda, permite que se possa agregar o conhecimento desenvolvido específico em cada parte.

As responsabilidades de cada camada são como segue:

Camada de interface com o usuário ou camada de apresentação - a camada de apresentação é responsável por implementar componentes de interface com o usuário como navegação e apresentação. É através desta camada que o usuário interage com a aplicação. A interface pode ser num contexto de janelas ou ser através de um navegador ou a forma que permita dar a funcionalidade desejada. (*NAJMI, 2003*)

Camada de tratamento de falhas – a camada é responsável por tratar falhas ocorridas nas demais camadas permitindo ao usuário ter um software mais amigável e livrando da sensação de não saber fazer com os erros. Além disso, auxilia na navegação permitindo ao usuário acessar somente áreas que tenha permissão.

Camada de lógica do negócio ou aplicação - é a camada que contém as regras de negócios do sistema. Nesta camada está contida todo conhecimento do domínio para a qual a aplicação é desenvolvida. A camada de lógica da aplicação é considerada a parte central do sistema. (*NAJMI, 2003*) Ela se comunica tanto com a camada de apresentação como a de tratamento de dados.

Camada de tratamento de dados - a camada de tratamento de dados tem como objetivo separar a lógica do negócio do armazenamento dos dados. Assim uma mesma aplicação pode ter diferentes formas de armazenamento de dados. Ela se comunica tanto com a camada de lógica do negócio como a de camada de dados.

Camada de dados - a camada de dados tem como função recuperar e armazenar dados da aplicação. Estes dados podem estar armazenados em banco dados, arquivos simples, XML ou qualquer outra forma de armazenamento de dados. (*NAJMI, 2003*)

4.2 Avaliação da proposta

A atual implementação do sistema cliente-servidor do estudo de caso apresenta duas camadas, uma camada de armazenamento de dados e a outra camada com a lógica do negócio.

Neste caso, a lógica do negócio, a interface com o usuário, tratamento de falhas estão fortemente acopladas. Tem que ser considerado a relação entre as duas camadas. A camada cliente necessita conhecer a organização dos dados pois os acessos são feitos diretamente sobre o banco de dados, neste caso. Assim, modificações realizadas em qualquer parte do sistema pode repercutir no todo o que se faz necessário uma verificação no todo da repercussão da mudança.

Sistemas em camada são sistemas mais lentos e complexos. O sistema agrega além da programação para atingir o objetivo proposto a comunicação e o conseqüente tratamento que os dados devem ter em cada camada.

Este acréscimo de processamento é possível nos dias de hoje pelo alto índice de desenvolvimento adquirido a cada ano, não só a nível de microprocessadores como também a nível de comunicação de dados através de redes de computadores.

A proposta de multi-níveis tem como objetivo tirar proveito do desenvolvimento dos microprocessadores, redes de computadores e também das diversas áreas da ciência da computação.

O rápido desenvolvimento da tecnologia na área da ciência computação nos últimos trinta anos torna difícil predizer como será o futuro. O desenvolvimento de sistemas camadas permite tirar o proveito do desenvolvimento da tecnologia em cada área diminuindo para as empresas o custo de refazer o sistema inteiro, ganhando em tempo de desenvolvimento e custo, pois melhorias, alterações, avanços somente seriam estudadas e executadas nas camadas necessárias.

Os Sistemas de Informação têm como característica a atualização constante, principalmente o caso apresentado. O Sistema de Avaliação de Disciplina e Docente informatizado é uma aplicação nova na Universidade. Dessa forma ela apresenta um grande potencial de desenvolvimento e com isso a possibilidade de receber manutenção e incremento de novas funcionalidades é elevada. A organização em multi-camadas apresenta como a vantagem a possibilidade de um desenvolvimento e crescimento organizado. As diferentes áreas do sistema estão divididas em camadas. Esta organização permite avaliar o impacto das mudanças em cada camada separadamente e avaliar o impacto nas demais com mais segurança e controle.

Importante ressaltar que o avanço da tecnologia faz necessário que cada vez mais as instituições, empresas tenham pessoal qualificado em diferentes áreas. É necessário ter

pessoas que entendam da lógica do negócio, projetistas de interface, especialistas em tratamento de falhas, armazenamento de dados e todos mais a tecnologia apresentar.

5 CONCLUSÃO

Durante o desenvolvimento do trabalho é mostrado a importância do conhecimento de padrões existentes. Este conhecimento possibilita desenvolver softwares mais eficientes e mais rápidos pois há utilização de um conhecimento pré existente e, possivelmente, testado na sua eficácia.

O desenvolvimento de sistemas multi-camadas demonstra sua viabilidade e importância no atual momento em que se encontra a tecnologia na área da computação. Cada vez mais existirão novos equipamentos acessando sistemas e cada um com suas peculiaridades. Não só novos equipamentos acessam os sistemas mas também é maior o número de pessoas, pessoas de diferentes culturas. Torna-se, importante, existir formas de controle de acesso para segurança dos dados mas sem ferir a variedade de tecnologias existentes e que contemple as diferentes pessoas. Mas para o desenvolvimento de sistemas multi-camadas, para melhor benefício de seu uso e eficácia, torna-se prudente a existência de um grupo de pessoas de diferentes especialidades. O nível de desenvolvimento tecnológico atingido na área dificulta que uma pessoa tenha o domínio da amplidão de conhecimento existente.

Este trabalho limita-se a fazer um estudo teórico do tema proposto mas considera muito interessante poder desenvolver e aplicar a solução proposta para uma melhor avaliação e clareza das vantagens e desvantagens da solução proposta.

REFERÊNCIAS

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML Guia do Usuário**. Rio de Janeiro: Campus, 2000.

BUSCHMANN ET AL **Pattern-Oriented Software Architecture A System of Patterns**. Chichester, Englad: John Wiley & Sons, 1996.

FERREIRA, A. B. H. **Novo Dicionário da Língua Portuguesa**. Rio de Janeiro: Nova Fronteira, 1986.

GAMA, et al. **Padrões de Projetos Soluções Reutilizáveis de Software Orientado a Objetos**. Porto Alegre, Bookman, 2000.

HEUSER, C. **Projeto Orientado a Objetos** – Apostila da disciplina: Desenvolvimento Orientados a Objetos, UFRGS, Porto Alegre, 2003.

LEN, B. **Reasoning about Software Architectures**. Disponível em: <www.seicmu.edu/ata/reasoning_about.html>. Acesso em: jul. 2003.

LEN, B.; KAZMAN, R. **Desenvolvimento Baseado em Arquitetura**. Disponível em: <www.sei.cmu.edu/architecture/definitions.html >. Acesso em: jul. 2003.

LISBÔA, M.L.B. **Tutorial: arquiteturas de meta-nível**. Fortaleza: UFC, 1997. 35p. Tutorial apresentado no XI Simpósio Brasileiro de Engenharia de Software, tutorial.

YOURDON, E. **Análise e Projeto Orientados a Objetos**. São Paulo, Makron Books, Brasil, 1999.

OLIVEIRA, R.; CARISSIMI, A; TOSCANI,S. **Sistemas Operacionais.** Porto Alegre: Sagra Luzzato, 2001.

MARTIN, J.; ODELL, J. **Análise e Projeto Orientados a Objeto**. São Paulo: Makron Books, Brasil, 1996.

NAJMI: N-Tier Application Architecture. Disponível em: <www.techiwarehouse.com /Articles/2003-03-31.html > Acesso em: ago. 2003.

PLATT , M. **Microsoft Architecture Overview** . Disponível em: <http://msdn.microsoft.com/architecture> Acesso em: ago. 2003.

RUMBAUGH, J. et al. **Modelagem e Projetos Baseados em Objetos**. Rio de Janeiro, Campus, 2003.

SHAW, M..; GARLA.N. **Software Architecture – Perspectives on Emerging Discipline**, [S.l.] Prentice Hall, 1996.

SEI - **Como definir Arquitetura de Software**. Disponível em <www.sei.cmu.edu/architecture/definitions.html >. Acesso em: ago. 2003.

SPRITZER, S. **A Espiral de mudanças – Comunicação Efetiva e Neurolingüística aplicadas a mudanças pessoas e organizacionais**. Porto Alegre, Ortiz, 1994.

TINDALL. **Desenvolvendo aplicações corporativas com Visual Basic, MTS, IIS, SQL Server e XML**. Tradução de Edson Furmankiewicz, Joana Figueiredo. Rio de Janeiro, Campus, 2000.