

1 Introdução

Atualmente, os processos de negócios de uma corporação estão firmemente ligados a aplicações de computador. Embora os seres humanos sejam os responsáveis pela maior parte do trabalho, os sistemas de computador são projetados para dar suporte à estrutura organizacional e aos processos existentes [CUMMINS, 2002]. Estes sistemas são baseados nos processos de negócios da corporação e desenvolvidos na tecnologia atual, porém com o passar dos anos eles não contemplam mais todas as necessidades das empresas e passam a pertencer a uma tecnologia obsoleta, constituindo então os conhecidos sistemas legados.

A evolução tecnológica é um fator que contribui para a fragmentação de sistemas, pois quando novos sistemas são desenvolvidos, é utilizada a tecnologia mais recente. Os sistemas mais antigos simplesmente não podem ser reimplantados utilizando-se a mesma tecnologia; eles precisam de um retrabalho ou redesenvolvimento substancial para serem embutidos na nova tecnologia. Caso este retrabalho não seja feito, os antigos sistemas vão se desatualizando e se tornando de difícil operação e integração.

Quando são feitas tentativas para alterar processos de negócios, normalmente as aplicações também precisam passar por alterações associadas. E, em geral, essas alterações são mais difíceis do que demonstram ser. Conseqüentemente, as alterações nos processos de negócios exigem uma implementação longa que envolve muitos profissionais, muitas horas de trabalho e, em geral, soluções baseadas em tentativas e erros.

Em geral, as alterações feitas no sistema e nos processos de negócios não precisam ser iniciadas do zero, pois normalmente a empresa não é tão alterada a ponto de não haver mais qualquer funcionalidade relevante disponível nos sistemas que já existiam. Infelizmente, as

funções de negócios costumam estar tão entrelaçadas a mecanismos computacionais que é difícil separar o que pode ser preservado do que precisa ser alterado [CUMMINS, 2002].

Um outro aspecto importante foi o crescimento em massa de usuários na Internet, pois devido a este crescimento, surgiu a necessidade de as empresas desenvolverem sistemas adaptados à tecnologia *Web*, a fim de atender uma nova demanda de mercado. Conseqüentemente, surgiu também a necessidade de integrar alguns sistemas legados da empresa com as aplicações que irão executar na *Web*.

Pelos motivos expostos acima, a arquitetura de integração corporativa deve fornecer uma estrutura em que sistemas legados eficientes possam continuar a funcionar quando essas novas tecnologias e as mudanças de negócios correspondentes se integrarem à operação da empresa. Várias soluções de integração se esforçam para dar suporte a esta arquitetura, porém estas soluções mudam constantemente, fazendo com que o acompanhamento não seja uma tarefa fácil.

Analisando as diversas áreas da engenharia, percebe-se que o problema da falta de integração entre sistemas legados também é facilmente encontrado, como em qualquer outra área. Em se tratando dos sistemas legados da engenharia elétrica, em especial aqueles voltados à educação, é muito comum encontrar nas universidades sistemas que operam isoladamente, não havendo integração alguma entre os mesmos. **Portanto, a proposta desta dissertação é apresentar um estudo de caso relacionado a aplicações educacionais direcionadas para a engenharia elétrica.** Nesse estudo de caso são apresentados os aspectos fundamentais referentes à integração de três sistemas legados e à execução dos mesmos via *Web*.

Os três sistemas legados em questão têm a função de realizar algumas tarefas referentes ao processo de planejamento de sistemas móveis celulares. Esses sistemas, que são os módulos de um sistema maior chamado CELLP, foram desenvolvidos pelos alunos do curso de pós-graduação em engenharia elétrica da UFPA, e foram implementados utilizando diferentes linguagens de programação e sistemas operacionais.

Como nesta dissertação serão tratados sistemas a serem aplicados na área de educação, não se pode deixar de pensar em estimular cada vez mais o uso da educação a distância, já que este modelo de educação talvez seja o único que se adapta à nova realidade empresarial, que define o bom trabalhador como sendo aquele que aprende de forma não-convencional e que sabe trabalhar cooperativamente para gerar soluções inovadoras e competitivas. Analisando o cenário empresarial, percebe-se que o ensino convencional não tem dado condições aos indivíduos de se capacitarem conforme as novas exigências de mercado.

Percebe-se que hoje a educação e o treinamento ainda são, na grande maioria, baseados fisicamente nas instituições e restritos a cronogramas pré-determinados de cursos. Essa estrutura implica em deslocamento de treinandos e professores, e na necessidade de interromper temporariamente o trabalho para se submeter a uma formação [MAIA, 2000]. Portanto, faz-se necessário uma nova abordagem na formação e na atualização profissional, fazendo com que as pessoas retomem o trabalho, mas continuem sempre se capacitando.

A solução é um novo modelo de formação: aprendizado cooperativo em ambientes ligados em rede e tempos virtuais. Aprender deixa de ser um processo que ocorre num período definido, para ser uma atividade com a duração direcionada pelo aluno. A tecnologia dos ambientes em rede é um meio para facilitar a interação social, viabilizar a aprendizagem

individual, através das interações com um grupo e para possibilitar a criação coletiva de um conhecimento compartilhado [MAIA, 2000].

Tendo em vista os aspectos apresentados anteriormente, decidiu-se que esta dissertação terá como objeto final um site, o qual conterá um material teórico onde serão explicados de forma prática e didática, todos os componentes e passos referentes ao processo de planejamento de sistemas móveis celulares. E, a partir desse site, o usuário será capaz de executar os três sistemas mencionados anteriormente, com o propósito de pôr em prática os conhecimentos adquiridos.

Este site servirá de apoio para as disciplinas que tratam de Sistemas Móveis Celulares no Curso de Engenharia Elétrica da UFPA, que hoje são ministradas de forma presencial. E para suprir as inúmeras desvantagens do ensino presencial, é que surgiu este ambiente, o qual tem como seus principais objetivos:

- Facilitar o aprendizado, através de uma *interface* interativa que reforça os conhecimentos da sala de aula tradicional;
- Disponibilizar o conhecimento para aprendizado e revisões complementares das disciplinas envolvidas;
- Disponibilizar um protótipo para o uso da comunidade acadêmica, deixando-o preparado para o acoplamento de novos módulos;
- Possibilitar, por um período de tempo integral, a utilização do ambiente, levando em conta a disponibilidade e o ritmo do aluno;
- Avaliar a integração de sistemas que foram desenvolvidos em linguagens de programação diferentes.

Para que isso fosse possível, houve a necessidade de fazer a engenharia reversa do CELLP, a fim de facilitar o processo de integração dos três sistemas e facilitar também a fatoração dos mesmos em alguns objetos distribuídos. Estes processos, posteriormente, agilizaram as modificações necessárias no código, para enfim possibilitar a integração e execução via *Web*. Em função dessa nova capacidade do CELLP, decidiu-se então que o mesmo deve ser chamado de WebCELLP.

Para obter estes resultados fez-se uso das seguintes tecnologias: i) UML (*Unified Modeling Language*), para fazer a engenharia reversa do CELLP; ii) MVC (*Model View Controller*), para facilitar o particionamento do CELLP em três camadas (camada cliente, *middleware* e camada servidora); iii) CORBA, para tratar a heterogeneidade, permitindo assim a integração dos módulos do CELLP; iv) SOAP (*Simple Object Access Protocol*), para tornar o sistema flexível o bastante a ponto de operar sobre o protocolo HTTP (*HiperText Transfer Protocol*), e v) os ambientes de programação Delphi e Kylix, para realizar as modificações necessárias na codificação do CELLP.

1.1 Estrutura do Trabalho

No capítulo 2 é feita uma breve apresentação sobre a educação a distância, abordando os recursos que vêm sendo utilizados por ela ao longo do tempo, dando maior ênfase ao ensino através da Internet.

O capítulo 3 aborda a computação distribuída, explorando os problemas enfrentados em sistemas distribuídos, como por exemplo, a heterogeneidade. São abordados também dois *middlewares* como solução para tratar a heterogeneidade e a integração de sistemas legados.

No capítulo 4 são apresentadas as principais características das arquiteturas de uma, duas e três camadas. É apresentado também, cada um dos componentes do padrão de projeto *Model View Controller* (MVC).

No capítulo 5 são tratadas as tecnologias envolvidas no desenvolvimento do WebCELLP, além de ser apresentada a modelagem do mesmo, através de alguns diagramas da UML, como Diagrama de Caso de Uso, Diagrama de Classes e Diagrama de Sequência.

E no capítulo 6 se encontram as conclusões obtidas no trabalho, bem como o elenco de trabalhos futuros que poderão surgir.

2 Educação a Distância (EAD)

2.1 Introdução

Analisando o cenário do mercado de trabalho, percebe-se que as empresas modernas demandam por profissionais com constante atualização dos conhecimentos e com habilidade para resolver problemas complexos através da cooperação com outros profissionais, independentemente dos seus locais de trabalho. Entretanto, as instituições de educação e treinamento usualmente preparam pessoas para trabalharem isoladamente e competitivamente em assuntos relacionados com um domínio de aplicação. Portanto, é necessário criar condições para formar profissionais que saibam aprender e trabalhar cooperativamente.

A educação a distância enquadra-se perfeitamente com a necessidade de capacitação contínua exigida pelas empresas, pois é uma modalidade não tradicional típica da era industrial e tecnológica, cobrindo distintas formas de ensino-aprendizagem, dispondo de métodos, técnicas e recursos, postos à disposição da sociedade.

2.2 O novo paradigma da educação

O modelo de educação tradicional está apoiado quase que exclusivamente na prática da aula expositiva, cuja forma praticamente não sofreu modificações, apesar da evolução dos meios de comunicação e do aparecimento de novos recursos de auxílio ao professor no ambiente da sala de aula.

De acordo com [KURI, 1998], o ensino apoiado exclusivamente na aula expositiva cumpriu seu papel de forma aceitável durante uma época em que os conhecimentos a serem transmitidos eram relativamente estáveis, e o conjunto de conhecimentos a ser adquirido pelo aluno não necessitava ser renovado durante sua vida profissional. No entanto, com a rapidez das mudanças científicas e tecnológicas, os objetivos formativos tornaram-se prioritários em relação aos informativos na formação dos novos profissionais.

No novo paradigma, o aluno deve desenvolver a habilidade de aprender a aprender, ou seja, ter capacidade de atualização contínua, de apropriar-se do conhecimento, sabendo de quais fontes obter a informação e como filtrá-la. Assim, o aluno estará mais bem preparado para lidar com o crescimento exponencial da informação e com a evolução cada vez mais rápida do conhecimento científico e tecnológico, situações que fatalmente enfrentará durante a sua vida profissional.

O educador, por sua vez, deixa de ser o “provedor” da informação e do conhecimento e passa a desempenhar papel de “facilitador” da aprendizagem, orientando e fornecendo oportunidades para que o próprio aluno busque a informação onde quer que ela se encontre e transforme-a em conhecimento, dentro de uma postura ativa, reflexiva e criativa.

Portanto, faz-se necessário um novo paradigma educacional que acompanhe a alta velocidade em que o mundo está mudando, sendo necessário que ele atenda entre outros requisitos, a:

- Otimizar as atividades do aluno;
- Melhorar a qualidade das informações transmitidas;
- Atualizar constantemente o currículo-base da formação desejada;
- Adequar a velocidade de aprendizado à capacidade do aluno.

Para atingir essas metas, é necessário incrementar a qualidade e a quantidade de interações entre os estudantes. O uso de sistemas computadorizados, utilizados como suporte na geração e transmissão das bases de conhecimento e experiência, servirão para efetivamente tornar este novo paradigma uma realidade disponível ao nosso alcance [MAIA, 2000]. A figura 2.1 representa o novo modelo de educação.

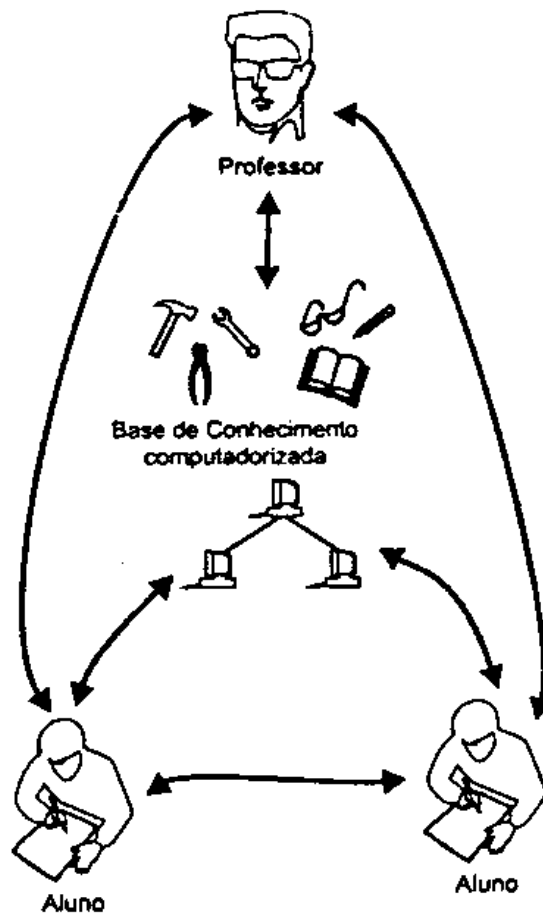


Figura 2.1: O novo paradigma da educação.

2.3 Educação a Distância

Tratar de educação a distância significa trabalhar com um tema que representa romper com um ciclo determinado há muito tempo. Este rompimento, no entanto, não pode ter em sua base a substituição de sistemas presenciais por sistemas a distância.

A educação a distância tem em sua base a idéia de democratização e facilitação do acesso à escola, não a idéia de suplência ao sistema regular estabelecido, nem tampouco a implantação de sistemas provisórios. Mas sim, em sistemas fundados na educação permanente, demanda que a sociedade impõe hoje como forma de superação de problemas relativos ao desenvolvimento econômico e tecnológico que vivenciamos [PRETI, 1996].

A idéia da educação a distância é estabelecer um processo educativo sistemático e organizado através de uma comunicação bidirecional entre aluno e professor, estando estes geograficamente afastados. Além de criar um canal de comunicação, os sistemas de EAD buscam enfatizar outras características, como: a criação de um processo continuado, a organização da informação de uma maneira mais crítica e construtiva contribuindo para a democratização do saber e a utilização de meios ou multimeios na estratégia de comunicação [NUNES, 1994].

A educação a distância não é nova, ela é utilizada há muito tempo na educação formal e não-formal valendo-se de mídias convencionais, como material impresso, rádio e televisão. A primeira forma de EAD conhecida e relatada na literatura foram os cursos por correspondência, que foram muito utilizados desde o início até meados do século passado, quando o rádio e a televisão instrucional tornaram-se populares. Hoje, vivemos a era dos satélites, fibras óticas, computadores, CD-ROM, tele e videoconferências, Internet, correio eletrônico, cursos on-line, etc.

A EAD, nesta nova perspectiva, se caracteriza pela independência de tempo e espaço, pelo controle do aprendizado realizado mais intensamente pelo aluno do que pelo professor não-presencial e pela comunicação entre alunos e professores mediada por documentos na *Web* ou alguma outra forma, como videoconferência. Os encontros síncronos e assíncronos agora fazem parte da terminologia utilizada na educação.

No início da introdução das tecnologias da informação e comunicação na área educacional, muitas pessoas passaram a crer que estes instrumentos iriam solucionar os problemas educacionais, podendo até substituir os professores. Até o presente instante, no entanto, não foi isso que se verificou de fato. Constatou-se sim a possibilidade de utilizar esses instrumentos para sistematizar os processos e a organização educacional e para redefinir o papel do professor.

Como a utilização da informática na área educacional é recente, muitos se questionam sobre a sua utilização. Segundo [TAJRA, 2000], a informática deve ser utilizada na educação, pois não se trata apenas de um instrumento com fins limitados, mas apresenta várias possibilidades, tais como: pesquisas, simulações, comunicações ou entretenimento. Cabe a quem irá utilizá-la para fins educacionais definir qual objetivo se quer atingir, pois mesmo a sua utilização restrita pode ter um importante valor. Veremos a seguir como a Internet pode auxiliar o processo de ensino-aprendizagem a distância.

2.4 Internet na Educação a Distância

No Brasil a Internet esteve confinada ao circuito acadêmico, na chamada rede nacional de pesquisa (RNP), até meados de 1995, quando o governo criou o grupo gestor de Internet Brasil, incentivou a criação de backbones comerciais e viabilizou o surgimento de provedores

de acesso [DAMSKI, 1996]. A Internet teve um crescimento estrondoso nos últimos anos, possibilitando o contato de milhares de pessoas no mundo inteiro, ultrapassando as barreiras da distância, da língua e da cultura.

A sua característica democrática que permite que qualquer pessoa faça parte dela, aliada a sua interface gráfica que possibilita o uso de imagens estáticas e animadas, vídeo, som e interatividade com o usuário, fazem com que a Internet tenha um alto poder de sedução. Este fato fez com que a Internet seja hoje uma das principais ferramentas utilizadas por alunos, professores e tutores em EAD, pois oferece a chance de esclarecer suas dúvidas a distância, promovendo ainda, o estudo em grupo com estudantes separados geograficamente, permitindo-lhes a discussão de temas do mesmo interesse.

Com esta tecnologia, o aluno sai do isolamento, enriquecendo seu conhecimento de forma individual ou grupal, fazendo perguntas, manifestando idéias e opiniões, realizando também uma leitura de mundo mais global. Isto cria uma nova dinâmica pedagógica interativa, contribuindo para a formação moderna dos alunos. A aprendizagem torna-se colaborativa, pois os estudantes podem trabalhar com alunos de outras culturas, permitindo assim ampliar seu conhecimento.

A educação a distância mediada pela Internet guarda uma certa semelhança com as salas de aulas tradicionais. Na agora chamada escola virtual, seus componentes são basicamente os mesmos da escola tradicional: aluno, professor, facilitador, monitor, suporte técnico, suporte administrativo, administradores, conteúdo didático, sistemas de suporte ao material didático, sistemas de gerenciamento de aprendizagem e mídia.

2.4.1 Vantagens da Internet na Educação

As vantagens da utilização da Internet em um ambiente educacional podem variar de acordo com os objetivos que o professor definir para cada atividade. É importante, no entanto, que quando forem utilizar a Internet, professores e alunos estejam dispostos a enfrentar os novos desafios desta ferramenta.

Dentre as inúmeras vantagens da utilização da Internet na educação a distância, pode-se destacar o fato dela ser:

- Flexível: a qualquer hora e a partir de qualquer lugar pode-se acessar o curso, desde que haja os recursos mínimos, como computador conectado à rede e um *browser*;
- Dinâmica: é de fácil atualização e possibilita o contato direto a qualquer momento e por razões imediatas, seja este contato com professores, tutores, equipe de apoio ao curso ou outros colegas;
- Aberta: pois além do ambiente virtual criado para o curso, têm-se a possibilidade de pesquisar em diferentes sites na Internet, ampliando conceitos e informações oferecidas na estrutura do curso e possibilitando que os alunos percorram bibliotecas e sites internacionais, sem custos adicionais, desde que não existam barreiras lingüísticas;
- Sem fronteiras geográficas: podem se atingir pessoas presentes em qualquer parte do mundo, desde que não haja obstáculos da língua, para colaborar no esclarecimento de dúvidas, participação em fóruns de debates, etc;
- Amigável: pois requer do aluno mínimos conhecimentos de navegação, como a manipulação de um *browser* e familiaridade com os recursos comunicativos da Internet;
- Adaptável às necessidades do aluno: a educação a distância on-line adequa-se à formação continuada de profissionais que não podem interromper suas atividades de trabalho e também não podem se deslocar para participar de cursos presenciais.

Entretanto, nem tudo na Internet é perfeito, podem-se identificar alguns problemas como: informações sem fidedignidade, facilidade de dispersão durante a navegação e lentidão de acesso aos sites. Porém, estes problemas dificilmente superarão as inúmeras vantagens que a Internet proporciona para a educação a distância.

3 Computação Distribuída

3.1 Introdução

Neste capítulo serão abordados alguns conceitos da computação distribuída bem, como algumas tecnologias inerentes à mesma, dando maior ênfase nos aspectos relacionados a aplicações distribuídas, para não fugir do escopo desse trabalho. O capítulo inicia com uma breve descrição dos sistemas distribuídos, que causaram a emersão do paradigma de objetos distribuídos como solução para diversos problemas encontrados no desenvolvimento tradicional de sistemas de computação. Em seguida, será feita uma descrição detalhada sobre objetos distribuídos e também a conceituação de *Middleware*, o qual é o componente responsável pela comunicação entre os objetos distribuídos. Este capítulo será finalizado com a apresentação das duas especificações de *Middleware* mais utilizadas no mercado, que são os padrões CORBA e Java/RMI. Serão apresentadas também algumas considerações finais sobre o assunto.

3.2 Sistemas Distribuídos

O aparecimento de redes de computadores permitiu a utilização de um novo paradigma computacional, que se mostrou, com o passar do tempo, extremamente poderoso. Este paradigma se refere à possibilidade de distribuição do processamento entre computadores diferentes. Mais do que a simples sub-divisão de tarefas, este paradigma permite a repartição e a especialização das tarefas computacionais conforme a natureza da função de cada computador. Um exemplo típico é a chamada arquitetura cliente/servidor, onde muitos computadores clientes se comunicam com computadores servidores que nada mais são do que

processos especializados na execução de certas tarefas, como cuidar de arquivos ou administrar bancos de dados.

Para [COULOURIS, 2001], sistema distribuído é um sistema no qual componentes de hardware ou software localizados em uma rede de computadores se comunicam e são coordenados somente através da passagem de mensagens. Estes sistemas têm as seguintes características:

- Concorrência: Em uma rede de computadores, execução de programas concorrentes é uma norma. Dois usuários podem estar trabalhando em máquinas diferentes, e compartilhando recursos como páginas *Web* ou arquivos quando necessário. A capacidade do sistema manusear recursos compartilhados pode ser aumentada através da adição de mais recursos na rede. Um outro aspecto importante é a coordenação da execução de programas concorrentes, que também compartilham recursos;
- Ausência de relógio global: Quando programas precisam cooperar, eles coordenam suas ações através da troca de mensagens. Coordenação perfeita, sempre depende de uma idéia compartilhada do tempo no qual as ações dos programas ocorrem. Mas na verdade existem limites para a correta sincronização entre relógios de computadores de rede, não existe uma noção global do tempo correto. Isto é uma consequência direta do fato de que a única comunicação na rede é feita através da troca de mensagens;
- Falhas independentes: Todos os sistemas de computadores podem falhar, e é responsabilidade de quem os projeta, planejar as consequências de possíveis falhas. Falhas na rede resultam no isolamento de computadores que estão conectados, mas isto não significa que eles pararam de funcionar. Na verdade os programas que estão sendo executados nestes computadores, nem podem detectar se a rede

apresenta alguma falha ou está inexplicavelmente lenta. Similarmente, a falha de um computador ou a terminação inesperada de um programa em alguma parte do sistema, não é imediatamente reconhecida por um outro componente com o qual se comunica. Cada componente do sistema pode falhar independentemente, deixando os outros ainda funcionando.

A motivação para construir e usar sistemas distribuídos originou do desejo de compartilhar recursos. Esses recursos, como impressoras, arquivos, programas, dentre outros, são disponibilizados para os clientes através de serviços. Cada serviço tem por finalidade gerenciar de forma eficiente o acesso aos recursos de que ele dispõe. Os serviços podem ser implementados como objetos, que oferecem seus recursos para os clientes externos. Este é o fundamento da tecnologia de Objetos Distribuídos.

Segundo [COULOURIS, 2001], existem sete desafios que o projeto de um bom sistema distribuído deve satisfazer. São eles: heterogeneidade, abertura, segurança, escalabilidade, gerenciamento a falhas, concorrência e transparência. Porém, a heterogeneidade será o único desafio detalhado neste trabalho, embora os outros desafios sejam de suma importância e possam ser encontrados em [COULOURIS, 2001].

3.2.1 Heterogeneidade

Segundo [COULOURIS, 2001], a Internet permite que usuários acessem serviços e executem aplicações através de uma coleção heterogênea de computadores e redes. A heterogeneidade aplica-se para os seguintes itens:

- Redes;

- Hardware de computador;
- Sistemas Operacionais;
- Implementações de diferentes desenvolvedores.

Embora a Internet consista de diferentes tipos de redes, essas diferenças são mascaradas pelo fato de todos computadores conectados a ela usarem os protocolos da Internet para se comunicar um com o outro. Por exemplo, um computador conectado a uma rede *Ethernet* utiliza uma implementação de protocolos da Internet específicos para *Ethernet*, enquanto que um computador em um tipo diferente de rede irá precisar de uma implementação dos protocolos da Internet específicos daquela rede.

Tipos de dados como inteiros podem ser representados de formas diferentes em diferentes tipos de hardware. Por exemplo, não há apenas uma forma para ordenar bytes inteiros. Estas diferenças de representações têm que ser tratadas na questão de troca de mensagens entre programas rodando em diferentes tipos de hardware.

Embora sistemas operacionais de computadores na Internet precisem incluir uma implementação dos protocolos da Internet, eles não necessitam prover a mesma interface de programação para estes protocolos. Por exemplo, as chamadas para troca de mensagens em Unix são diferentes das chamadas para Windows 2000.

Diferentes linguagens de programação usam diferentes representações para caracteres e estruturas de dados como *arrays* e *records*. Estas diferenças precisam ser endereçadas de tal forma que programas escritos em linguagens diferentes possam se comunicar uns com os outros.

Programas escritos por diferentes desenvolvedores não podem se comunicar uns com os outros, a não ser que eles usem padrões em comum. Ou seja, para essa comunicação acontecer, padrões precisam ser concordados e adotados – como são os protocolos da Internet.

3.3 Objetos Distribuídos

Apesar da grande expressividade que a programação orientada a objeto possui quando utiliza conceitos como classe e objetos para implementar conceitos concretos e abstratos do mundo real, ela só tem sentido quando submetida a um compilador da linguagem na qual foi escrita. Este compilador por sua vez, é certamente projetado para lidar com características de determinada plataforma (sistema operacional / processador), sem contar que os objetos das classes que virtualmente possam existir só poderão comunicar-se se estiverem no mesmo espaço de endereçamento.

Percebe-se que aplicações desenvolvidas neste modelo são fortemente acopladas, o que impede uma evolução das mesmas para um ambiente distribuído. Contudo, é neste contexto que os objetos distribuídos encaixam-se perfeitamente. Eles possuem as mesmas características dos objetos tradicionais: encapsulamento, polimorfismo, herança, só que com a grande vantagem de poderem ser executados em ambientes totalmente diferentes.

Como visto anteriormente os aplicativos distribuídos são aqueles aplicativos que são compostos de programas que cooperam entre si e rodam em vários processos diferentes. Segundo [COULOURIS, 2001], para alcançar este propósito, alguns modelos de programação precisam ser estendidos para serem aplicados a aplicativos distribuídos. A principal e mais conhecida extensão foi à extensão do modelo convencional de chamada de procedimento para

o modelo de chamada de procedimento remoto, o qual permite que programas clientes invoquem procedimentos que estão rodando em programas servidores, que geralmente estão em um computador diferente do cliente.

Recentemente o modelo de programação baseada em objetos foi estendido para permitir que objetos de processos diferentes se comuniquem uns com os outros por meio de RMI (*Remote Method Invocation*). RMI é uma extensão de invocação de métodos locais que permite a um objeto vivendo em um processo invocar métodos de outro objeto vivendo em um processo diferente. Está sendo usado o termo RMI para referenciar invocação de métodos remotos de uma forma genérica. Isto não deve ser confundido com exemplos particulares de invocação de métodos remotos tal como Java/RMI.

Em RMI, a requisição do cliente para invocar um método de um objeto é mandada em uma mensagem para o servidor que gerencia o objeto. A invocação é carregada através da execução de um método de um objeto no servidor e o resultado é retornado para o cliente em uma outra mensagem. Para permitir uma cadeia de invocações relacionadas, objetos em servidores podem se tornar clientes de objetos em outros servidores [COULOURIS, 2001]. Estes objetos podem também ser replicados para obter os benefícios de disponibilidade, tolerância de falha e otimização de performance.

Para representar estes tipos de objetos tornou-se necessário criar novos modelos de objetos, agora adaptados para solucionar as questões de distribuição. Conceitos como troca de mensagens são claramente tratados de maneira particular neste ambiente. Neste mesmo contexto, adaptou-se de tecnologias anteriores o conceito de *Middleware*, o qual passou a lidar com todo o processo de comunicação de objetos remotos, bem como o gerenciamento de todo o ciclo de vida destes mesmos objetos.

Também é de suma importância que o *Middleware* dê suporte a coexistência de objetos distribuídos com aplicações legadas. Para tanto, a utilização de protocolos de comunicação comum entre diferentes plataformas torna-se ideal. Os fabricantes de aplicações legadas deverão suprir esta necessidade através da implantação de interfaces que utilizem esse protocolo para comunicação.

3.4 *Middleware*

O termo *Middleware* aplica-se para camada de software que providencia a abstração programável bem como o mascaramento da heterogeneidade das redes, hardwares, sistemas operacionais e linguagens de programação. CORBA (*Common Object Request Broker Architecture*), o qual será detalhado neste capítulo, é um exemplo. Alguns *Middleware*, como Java/RMI mascaram as diferenças de rede, porém não têm a capacidade de mascarar as diferenças entre as linguagens de programação. Mas todos os *Middlewares* lidam com as diferenças entre sistemas operacionais e entre hardwares.

Segundo [COULOURIS, 2001], alguns aspectos importantes que podem ser tratados pelos *Midlleswares* são:

- A transparência de localização: no RPC (*Remote Procedure Call*), o cliente que chama o procedimento não pode dizer se o procedimento roda no mesmo processo ou em um diferente, possivelmente em um diferente computador. Nem ainda o cliente precisa saber da localização do servidor. Similarmente, em RMI o objeto fazendo a invocação não pode dizer se o objeto invocado é local ou não e nem precisa saber sua localização;
- Os protocolos de comunicação: os protocolos que suportam a abstração do *Midllesware* são independentes dos protocolos de transportes. Por exemplo, o protocolo de

requerimento/resposta pode ser implementado sobre UDP (*User Datagram Protocol*) ou TCP (*Transmission Control Protocol*);

- O hardware de computador e os sistemas operacionais: as informações armazenadas em programas são representadas por estruturas de dados. Independente da forma de comunicação usada, a estrutura de dados pode ser convertida para uma sequência de bytes antes da transmissão e re-convertida na chegada. As atividades de *marshalling* e *unmarshalling* são realizadas pela camada *Middleware* sem nenhum envolvimento por parte do programador. A função destas atividades é esconder as diferenças provenientes da arquitetura de hardware tal como ordem de bytes, e esconder as diferenças entre os diversos sistemas operacionais;
- O uso de várias linguagens de programação: alguns *Middleware* são projetados para permitir que aplicativos distribuídos usem mais de uma linguagem de programação. Em particular, CORBA permite que clientes escritos em uma linguagem invoquem métodos em objetos que vivem em outro programa servidor escrito em outra linguagem. Isto é alcançado pelo uso da IDL (*Interface Definition Language*) que define as interfaces de comunicação.

Para resolver estes problemas de heterogeneidade, o *Middleware* providencia um modelo computacional uniforme para ser usados por programadores de aplicações distribuídas. Estes modelos incluem invocação remota de objetos, notificação de eventos remotos, acesso SQL remoto e processamento de transações distribuídas. Por exemplo, CORBA provê invocação de objetos remotos, no qual um objeto de um programa rodando em um computador invoca um método de outro objeto de um programa rodando em outro computador. Sua implementação esconde o fato das mensagens serem passadas através da rede para serem mandadas as invocações requeridas e suas respostas [COULOURIS, 2001].

Definiu-se anteriormente o *Middleware* como uma camada de software tendo como propósito mascarar a heterogeneidade e isolar o desenvolvedor dos detalhes de baixo nível envolvidos nos protocolos de comunicação em uma rede. *Middleware* é representado por processos ou objetos em um conjunto de computadores que interagem entre si para implementar comunicação e suportar compartilhamento de recursos para aplicações distribuídas, a figura 3.1 mostra a localização da camada *Middleware*.

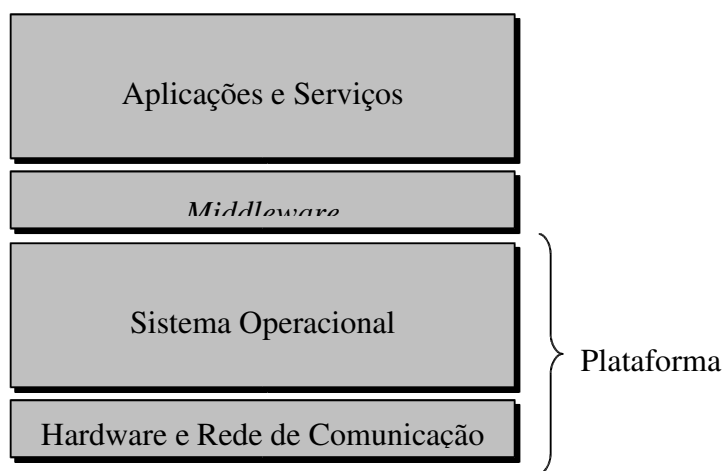


Figura 3.1: Camadas de software e hardware em sistemas distribuídos

O *Middleware* pode também prover serviços para serem usados por programas aplicativos. Estes são serviços infra-estruturais, totalmente ligados com o modelo de programação distribuída que o *Middleware* proporciona. Por exemplo, CORBA oferece uma variedade de serviços que proporcionam facilidades para aplicativos, aos quais incluem

nomeação, segurança, transações, persistência de armazenamento e notificação de eventos. Detalhes sobre estes serviços podem ser encontrados em [COULOURIS, 2001].

3.5 Java/RMI

A Invocação Remota de Métodos permite que os métodos de objetos Java sejam chamados a partir do código Java, que pode estar rodando em uma máquina virtual diferente, e que freqüentemente está em outro computador. Para utilizar RMI, deve-se fazer com que os métodos em seus objetos possam ser chamados a partir de outros objetos. Objetos que podem ter seus métodos chamados dessa maneira chamam-se de objetos remotos [CHAN, 1998].

Em Java/RMI interfaces remotas são definidas da mesma maneira de qualquer outra interface Java. Elas possuem as habilidades de interfaces remotas, simplesmente pelo fato estenderem de uma interface chamada *remote*. Ambos CORBA IDL e Java/RMI suportam herança múltipla de Interfaces. Isto é, uma interface pode ser estendida de uma ou mais interfaces [COULOURIS, 2001].

3.5.1 O Modelo Cliente/Servidor em Java

A linguagem Java oferece mecanismos conhecidos como *Sockets* e RMI para implementação do modelo cliente/servidor. *Sockets* abstraem, na forma dos objetos, as funcionalidades do software de rede que permitem a comunicação entre programas. Esta comunicação é realizada através de fluxos de dados puros, ou seja, tanto o cliente quanto o servidor enviam e recebem uma seqüência de bytes. Por conseguinte, o código deste tipo de aplicação deve tratar de questões como a codificação dos dados e protocolos de transmissão, elevando a complexidade de desenvolvimento do software.

Um outro mecanismo que Java utiliza para implementar o modelo cliente/servidor é o RMI. Este mecanismo é totalmente embutido em Java, isto quer dizer que todas as características inerentes à linguagem como a coleta de lixo automática, portabilidade, serialização de objetos, dentre outras serão também exploradas por RMI.

RMI pode ser considerado uma extensão orientada a objetos do mecanismo de RPC. Isto significa que um código cliente, neste caso um objeto cliente, pode invocar métodos exportados pela interface de determinado objeto servidor. A vantagem principal da utilização do modelo de objetos para a implementação RPC, é o fato dos objetos se adequarem perfeitamente ao contexto de distribuição.

A profunda integração de RMI com Java tem uma consequência direta na implementação de programas: tanto o programa cliente quanto o servidor devem ser codificados na mesma linguagem. Isso pode representar uma deficiência em comparação a outros protocolos como CORBA, visto que este permite uma independência de linguagem utilizada tanto no cliente quanto no servidor. Entretanto, RMI não precisa de adaptação para manipulação de objetos Java, enquanto que CORBA precisa implementar mapeamentos entre a IDL e a linguagem alvo, o que compromete o desempenho final do software.

O que pode ser notado, é que dependendo das circunstâncias que envolvem a escolha desta ou daquela tecnologia, a seleção de CORBA para implementar aplicações distribuídas em várias linguagens parece ser a melhor opção. Porém, segundo [CORNELL, 2001], se dois objetos que estão se comunicando forem escritos na linguagem de programação Java, então a generalidade e complexidade totais do CORBA não são exigidas. A melhor opção neste caso é usar RMI, já que foi desenvolvido especificamente para a comunicação entre aplicativos Java.

3.5.2 Arquitetura de RMI

O modelo arquitetural de comunicação utilizado por RMI, mostrado pela figura 3.2, é estruturado na forma de três camadas: a camada *stub/skeleton*, a camada de referência remota e a camada de transporte. Entre cada camada existe uma interface e um protocolo que regem a comunicação entre elas. Isso permite uma independência entre as camadas, sendo que a implementação de qualquer uma pode ser alterada sem que isto afete as demais.

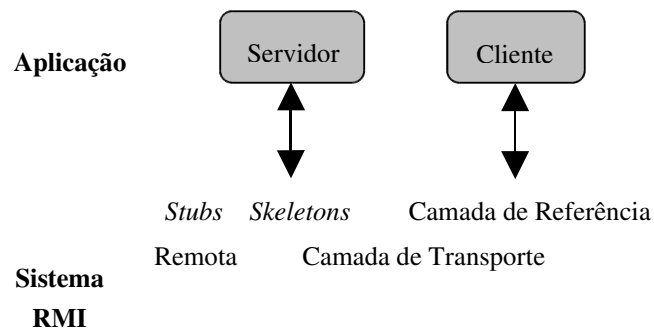


Figura 3.2 – Modelo de comunicação RMI

A implementação destas camadas, especialmente a camada *stub/skeleton*, utiliza duas técnicas que são fundamentais para garantir a transparência na invocação de métodos, que são: a Serialização de Objetos e o Carregamento Dinâmico de Classes. A primeira é utilizada no processo de *marshaling/unmarshaling* de parâmetros e a segunda é automaticamente acionada por RMI para garantir que um programa possa lidar com objetos remotos.

3.5.2.1 Camada Stub/Skeleton

Esta camada é a interface entre a camada de aplicação e o restante do sistema RMI. Ela é responsável por passar dados para a camada de referência na forma de fluxos *marshal*. RMI utiliza o mecanismo de serialização de objetos para efetuar o *marshaling/unmarshaling* de argumentos, para enviar objetos através de uma conexão de rede[CORNELL, 2001].

Em RMI, para que um cliente possa invocar métodos remotos é necessário que exista uma classe *stub* associada ao objeto remoto no contexto do cliente. Quando o código chama um método remoto, o *stub* faz um pacote que contém cópias de todos os valores de parâmetros e o envia para o servidor, usando o mecanismo de serialização de objetos para reunir os parâmetros [CORNELL, 2001]. Segundo a [SUN MICROSYSTEMS, s.d.], a classe *stub* basicamente executa as seguintes funções:

- Inicia uma chamada a um objeto remoto;
- Realiza o *marshaling* de argumentos;
- Informa a camada de referência que a chamada deve ser invocada;
- Realiza o *unmarshaling* do valor de retorno ou de uma exceção;
- Informa a camada de referência que a chamada está completa.

No lado do servidor existe um objeto *skeleton* que recebe o pacote enviado pelo *stub client*. O pacote enviado pelo *stub* geralmente consiste de um identificador do objeto remoto, um número de operação para descrever o método a ser chamado e os parâmetros depois do

processo de *marshaling* [CORNELL, 2001]. Segundo a [SUN MICROSYSTEMS, s.d.], de posse deste pacote, o *skeleton* executa as seguintes tarefas:

- Realiza o *unmarshaling* dos argumentos;
- Faz a chamada à implementação do objeto, que foi referenciada pelo identificador enviado na mensagem;
- Realiza o *marshaling* do valor de retorno da chamada ou de uma exceção, caso alguma tenha ocorrido.

3.5.2.2 Camada de referência remota

A segunda camada, denominada de camada de referência, tem a finalidade de especificar como a invocação de método irá transcorrer. Esta camada executa um protocolo específico que, dentre outras coisas, determina se o objeto servidor é simples ou replicado. De posse desta informação, por exemplo, se o objeto for replicado por vários servidores é possível determinar que o cliente encaminhe a invocação para cada réplica de servidor.

Além disso, esta camada também tem a tarefa de abstrair os diferentes modos de referenciar os objetos como, por exemplo, identificar se os servidores já estão executando em algumas máquinas, ou que só executarão quando invocados. Segundo a [SUN MICROSYSTEMS, s.d.], as principais funções executadas nesta camada são:

- Invocação ponto-a-ponto (*unicast*);
- Invocação para grupos de objetos replicados (*multicast*);
- Suporte para uma estratégia de replicação;

- Suporte para uma referência persistente a um objeto remoto (possibilitando a ativação de objetos remotos);
- Estratégia de re-conexão (se um objeto tornar-se inacessível).

3.5.2.3 Camada de Transporte

Esta camada é responsável basicamente pela configuração e gerenciamento da conexão entre espaços de endereçamento distintos. Na invocação remota de métodos, uma referência remota, na realidade, é representada por um *endpoint* e um identificador de objetos. Um *endpoint* consiste de uma abstração para um espaço de endereçamento ou máquina virtual. A camada de transporte do lado cliente utiliza o *endpoint* para ativar a conexão com o espaço de endereçamento do objeto remoto. No lado servidor, a camada de transporte utiliza o identificador de objetos para localizar a implementação de objetos para chamada. Segundo a [SUN MICROSYSTEMS, s.d.], esta camada é responsável pela:

- Configuração de conexões para espaços de endereçamento remoto;
- Gerenciamento de conexões;
- Monitoramento de conexões “liveness”;
- Atendimento de entrada de chamadas;
- Manutenção da tabela de objetos remotos que residem em um espaço de endereçamento;
- Configuração de uma conexão para uma entrada de chamada;
- Localização de um despachante para o alvo da chamada remota, e passagem da conexão para este despachante.

3.6 CORBA

A OMG (*Object Management Group*) foi formada em 1989 com objetivo de encorajar a adoção de sistemas de objetos distribuídos visando a aplicação dos benefícios da programação orientada a objetos para o desenvolvimento de software e fazer uso dos sistemas distribuídos, os quais estavam sendo cada vez mais difundidos. Para alcançar esses objetivos, a OMG defendeu o uso de sistemas abertos baseados no padrão de interfaces orientadas a objetos. Esses sistemas poderiam ser construídos utilizando hardwares heterogêneos, redes de computadores e sistemas operacionais diferentes e diversas linguagens de programação [COULOURIS, 2001].

Uma importante motivação foi permitir que objetos distribuídos pudessem ser implementados por qualquer linguagem de programação, e que esses objetos fossem capazes de se comunicar uns com os outros. Para atender esta motivação, foi projetada a IDL (*Interface Definition Language*), que é uma linguagem de interface independente de qualquer linguagem de implementação específica [COULOURIS, 2001].

Na primeira versão do CORBA, que foi a 1.1 surgida em 1991, definiu-se a IDL e a API (*Application Programming Interfaces*). Porém, a interoperabilidade entre os objetos desenvolvidos em linguagens de diferentes fabricantes, só veio em 1996 com a segunda versão, o CORBA 2.0, quando se implementou no mesmo o GIOP (*General Inter-ORB Protocol*). O GIOP pode ser implementado sobre qualquer camada de transporte com conexões. A implementação do GIOP para a Internet usa o protocolo TCP/IP e é chamado de IIOP (*Internet Inter-ORB Protocol*) [COULOURIS, 2001].

O componente mais importante desta arquitetura é o ORB (*Object Request Broker*), cujo papel fundamental é permitir que os clientes invoquem métodos que estão em objetos servidor, onde ambos clientes e servidores possam ser implementados por uma variedade de linguagem de programação [COULOURIS, 2001]. As principais funções que o ORB possui são: localização do objeto, ativação do objeto se necessário, envio da requisição do cliente para o objeto e envio da resposta do servidor para o cliente.

Além do ORB, existem outros componentes na arquitetura CORBA, que são de extrema importância. Estes componentes são mostrados na figura 3.3.

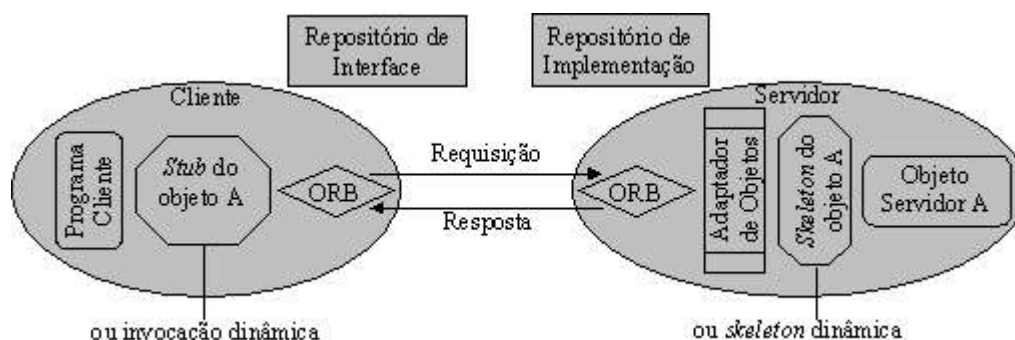


Figura 3.3 – Componentes da Arquitetura CORBA

Em CORBA, o programador define as interfaces IDL, que são as interfaces que os objetos remotos irão implementar, e depois usa um compilador de interface para produzir os correspondentes *stubs* e *skeletons*. Os *stubs* são gerados na linguagem do cliente e *skeletons* na linguagem do servidor. No modelo CORBA os clientes não são necessariamente objetos, eles podem ser qualquer programa que envia requisições para os objetos remotos e recebem as respostas.

Quando um cliente solicitar um serviço a um servidor, ele o fará através de invocação estática ou dinâmica. A invocação estática é usada quando a interface remota do objeto CORBA é conhecida na hora da compilação, capacitando *stubs* e *skeletons* para serem usados. Se a interface remota não é conhecida em tempo de compilação, invocações dinâmicas devem ser usadas. A maioria dos programadores prefere usar invocações estáticas porque elas providenciam um modelo mais natural de programação [COULOURIS, 2001].

3.6.1 Arquitetura de Gerenciamento de Objeto

A OMA (*Object Management Architecture*) foi criada pelo OMG, com o objetivo de integrar as aplicações baseadas em objetos distribuídos. Sua estrutura é baseada basicamente em dois elementos: o Modelo de Objeto, que é quem define o objeto que será distribuído pelo sistema heterogêneo e o Modelo de Referência, que é quem define as características da integração destes objetos.

O RFP (*Request for Proposal*) é utilizado para que os Modelos de Objetos e de Referência possam ser compatíveis com especificações anteriormente utilizadas, e com isso tornar possível a construção de sistemas de objetos distribuídos interoperáveis em sistemas heterogêneos.

O CORBA possui toda a sua estrutura baseada na arquitetura OMA, conforme mostra a figura 3.4. Esta define a comunicação entre os objetos distribuídos através de quatro elementos:

- **Objetos de Aplicação:** são os objetos criados pelo usuário, que possuem características definidas de acordo com os seus objetivos. Eles também são tidos como os usuários finais de toda a infraestrutura CORBA;
- **Facilidades Comuns CORBA:** são componentes definidos em IDL que fornecem serviços para o uso direto das aplicações de objetos. Estes estão divididos em duas categorias que são, horizontal e vertical. Eles definem regras de integração para que os componentes possam colaborar efetivamente;
- **Serviços Comuns CORBA:** são serviços (interfaces e objetos) que possibilitam a implementação e utilização de objetos. Eles definem uma estrutura de objetos de baixo nível, que ampliam as funcionalidades do ORB, sendo utilizados para criar um componente, nomeá-lo e introduzi-lo no ambiente;
- **ORB:** é o elemento que permite que objetos emitam requisições em um ambiente distribuído e heterogêneo, de forma transparente.

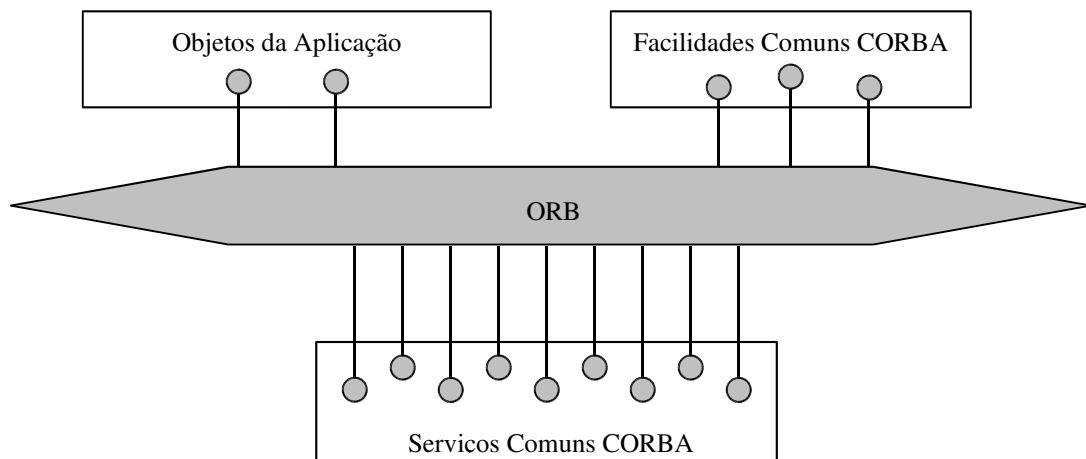


Figura 3.4 – Modelo de Referência OMA

3.6.2 ORB (*Object Request Broker*)

Como dito anteriormente, clientes e objetos que se comunicam através do CORBA, podem estar utilizando um sistema distribuído totalmente heterogêneo. Portanto, para mascarar esta heterogeneidade, é necessário uma interface com a qual o cliente tem contato direto, e esta com o objeto. Esta interface é denominada de ORB. Segundo [COULOURIS, 2001], o ORB provê um nível de abstração que permite que um cliente utilize um objeto independente da localização da sua implementação. Deste modo o ORB é responsável por todos os mecanismos necessários para:

- Encontrar a implementação de um objeto para o pedido de um cliente;
- Preparar a implementação do objeto para receber um pedido;
- Fazer toda a comunicação necessária.

Em soluções mais complexas, pode haver a necessidade de comunicação entre ORBs que podem estar separados em redes diferentes. Neste caso, uma requisição pode passar por vários ORBs, antes de chegar ao objeto. A comunicação entre ORBs, é possível através do uso do protocolo IIOP (*Internet Inter Orb Protocol*).

3.6.3 Linguagem de Definição de Interface

O propósito da IDL (*Interface Definition Language*) é permitir a interoperabilidade de aplicações, ela tem a função de fornecer uma interface de comunicação entre clientes e objetos, independentemente da localização física, do sistema operacional, da arquitetura de hardware e da linguagem de programação. A IDL é puramente declarativa, não fornecendo nenhum aspecto de implementação como definições de variáveis ou estruturas algorítmicas, portanto a codificação dos objetos é escrita em uma linguagem específica como Delphi, Java,

C++, SmallTalk, Cobol, etc, a figura 3.5 mostra o mapeamento da IDL em algumas linguagens de programação.

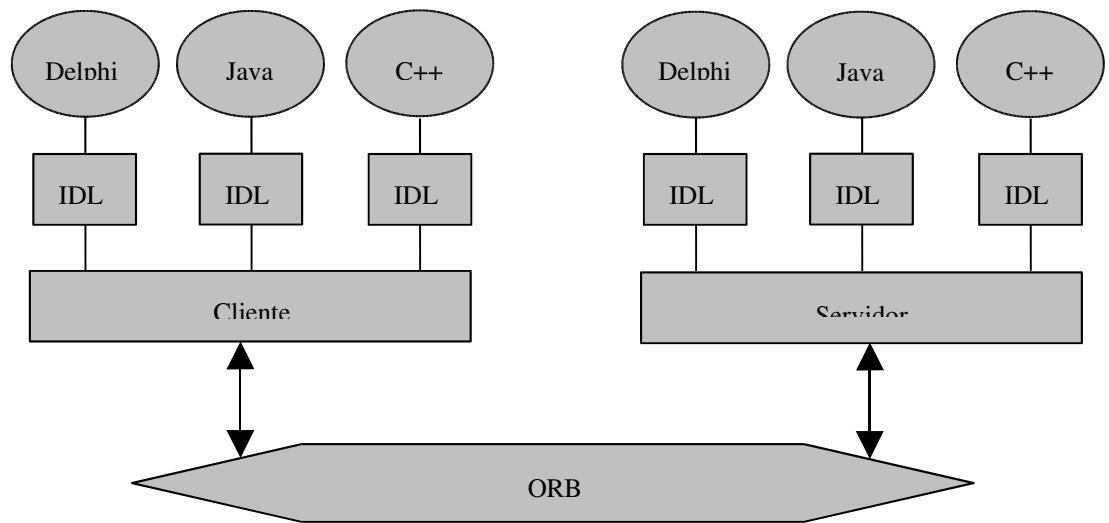


Figura 3.5 – Mapeamento da linguagem IDL em algumas linguagens de programação

A IDL é utilizada somente para expressar as operações que o objeto pode realizar, os seus atributos e parâmetros de entrada e saída, os eventos que o objeto pode emitir e suas exceções, ou seja, ela serve para especificar a interface do objeto. A IDL CORBA provê facilidades para definição de módulos, interfaces, tipos de dados, atributos, assinatura de métodos e exceções. Ela tem as mesmas regras lexicais da linguagem C++, acrescida de algumas palavras-chaves para suportar distribuição. As regras gramaticais da IDL são um conjunto das regras de ANSI C++, acrescidas de construções para suportar assinatura de métodos [COULOURIS, 2001].

3.6.4 Repositório de Interfaces

O papel do repositório de interface é prover para clientes e servidores, informações a respeito das interfaces IDL registradas [COULOURIS, 2001]. O *Interface Repository* (IR) armazena todas as definições IDL dos objetos CORBA disponíveis para clientes e possibilita que uma aplicação, em tempo de execução, possa descobrir informações sobre outros objetos sobre os quais desejam obter algum serviço. O IR é uma peça chave em sistemas de objetos distribuídos, que deve estar disponível não somente ao ORB, mas também aos clientes e implementações. Desse modo, podemos adicionar, excluir ou executar a depuração dessas interfaces, bem como traçar uma árvore de herança para definir o tipo de um objeto.

Para que o ORB possa interagir com objetos, ele precisa obter informações sobre estes e ter um local para armazená-las. Utilizando o IR, ele pode no momento de uma requisição (na invocação dinâmica que será vista posteriormente), verificar os tipos de parâmetros, as relações de herança entre interfaces e ajudar na interação de diferentes ORBs. Segundo [COULOURIS, 2001], as aplicações que usam invocações estáticas com *stubs* e *skeletons* (que também serão vistas posteriormente), não necessitam de repositório de interfaces. É importante ressaltar que nem todos os ORB's possuem um repositório de interface.

Uma vez que o IR permite que as aplicações descubram a informação de tipo em tempo de execução, pode-se afirmar que sua principal utilidade é a de dar suporte necessário a Invocação Dinâmica. Algumas operações que possibilitam o acesso aos elementos do IR são:

- *lookup_id()*: Procura o objeto no depósito de interfaces que possua o identificador passado como parâmetro da chamada;
- *contents()*: Usada para listar o conteúdo (constantes, definições de tipo, exceções, interfaces e módulos) do depósito de interfaces;
- *lookup_name()*: Procura elementos do depósito de interfaces que possuam o nome especificado; pode ser especificado o tipo de elemento que se deseja;

- *decribe_contents()* Usado para obter a descrição completa do conteúdo do depósito de interfaces, retornando o identificador e o nome de cada objeto contido.

3.6.5 Repositório de Implementação

Um repositório de implementação é responsável pela ativação de servidores registrados e pela localização de servidores que estão em execução. Através de um componente chamado Adaptador de Objetos, que será apresentado ainda neste capítulo, faz-se à referência de servidores quando eles estão sendo registrados e ativados.

Um repositório de implementação armazena o mapeamento dos nomes dos adaptadores de objetos para os caminhos (*pathnames*) dos arquivos que contêm as implementações de objetos. As implementações dos objetos e os nomes dos adaptadores de objetos são geralmente registrados no repositório de implementações quando programas servidores são instalados. Quando as implementações dos objetos são ativadas em servidores, o nome da máquina (*hostname*) e o número da porta do servidor são adicionados ao mapeamento [COULOURIS, 2001].

Portanto, os registros dos objetos no repositório de implementação são descritos da seguinte forma:

- Nome do adaptador de objetos;
- Caminho da implementação do objeto;
- *Hostname* e número da porta do servidor.

Nem todos os objetos CORBA precisam ser ativados em demanda. Alguns objetos, por exemplo, objetos de *callback* (retorno de chamada) criados por clientes, rodam apenas uma vez e depois são desativados porque não são mais necessários. Estes objetos não usam o repositório de implementações.

Um repositório de implementação geralmente permite o armazenamento de informações extras a respeito de cada servidor, como por exemplo, informação de controle de acesso que determina quem tem permissão de ativar ou invocar suas operações. Além disso, é possível replicar informações em repositórios de implementação para prover maior disponibilidade ou tolerância à falhas [COULOURIS, 2001].

3.6.6 STUBS E SKELETONS

Os *stubs* e os *skeletons* realizam funções correspondentes, sendo que o *stub* é utilizado no cliente e o *skeleton* no objeto. São eles que implementam a arquitetura cliente/servidor no CORBA. Tanto o *stub* quanto o *skeleton* são criados baseados na compilação da interface IDL do cliente e do objeto, respectivamente.

Devido a esta capacidade de localizar objetos-destinos, os *stubs* também são denominados de *proxies* ou procuradores. Os *stubs* são gerados a partir de uma interface IDL, através da compilação desta IDL pela linguagem do cliente. Os *stubs* empacotam os argumentos de uma invocação, para uma forma que possa ser trafegada pela rede, e desempacotam os resultados provenientes da mensagem de resposta.

Os *skeletons* são gerados a partir da mesma interface IDL em que serão gerados os *stubs*, porém desta vez a compilação será feita pela linguagem do servidor. Os *skeletons* têm

função parecida, com a diferença que eles irão receber requisições do ORB e, a partir de então, deverão desempacotar estas requisições e entregá-las à implementação do objeto. Caso haja resposta, eles irão empacotá-las e enviá-las aos *stubs*. O processo de requisição/resposta feito pelos *stubs/skeletons*, pode ser melhor compreendido na figura 3.6.

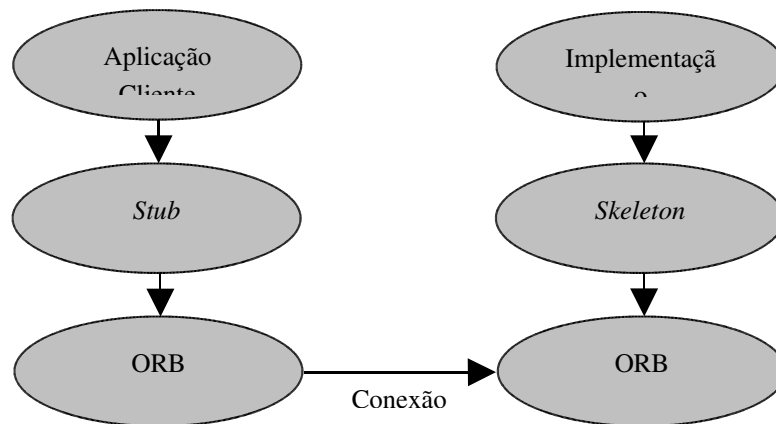


Figura 3.6 – Invocação Estática

É importante lembrar, que a invocação estática é utilizada quando se tem o conhecimento completo das interfaces OMG IDL dos objetos CORBA que serão invocados, capacitando então os *stubs* e os *skeletons* para serem usados.

3.6.7 Invocação Dinâmica

A invocação e envio dinâmico surgiu para solucionar o problema da inflexibilidade da invocação estática, que não permite que objetos criados após a compilação do *stub* ou *skeleton* sejam acessados. Ou seja, na invocação estática, para que novos objetos pudessem ser

acessados, era necessário parar o sistema e realizar nova compilação. Isto já não é mais necessário na invocação dinâmica.

As duas interfaces suportadas pelo CORBA para a invocação dinâmica são o DII e o DSI. O DII (Interface de Invocação Dinâmica) corresponde ao *stub* e o DSI (Interface *Skeleton* Dinâmico) corresponde ao *skeleton*.

Na invocação dinâmica, as interfaces, diferentemente da invocação estática, são fornecidas diretamente pelo ORB, portanto não há necessidade de cada cliente ou objeto possuir a sua. A vantagem da invocação dinâmica é a transparência, pois a implementação nunca sabe se a requisição está sendo feita por um *stub* estático ou dinâmico.

3.6.7.1 Interface de Invocação Dinâmica

Na interface de invocação dinâmica (DII), clientes podem emitir requisições a qualquer objeto, em tempo de execução, sem necessitar ter o conhecimento prévio das interfaces dos mesmos. Os objetos podem ser encontrados pelos serviços de nomes ou de *trading*. A importância desta interface é que, caso um objeto necessite ser modificado constantemente, o sistema não precisará ser parado e recompilado toda vez que isto acontecer, como na invocação estática.

Para que um cliente emita uma requisição a um objeto, ele precisa dizer quem é o objeto-destino (através de sua referência), qual é a operação que deseja realizar e quais são os parâmetros da operação. Tudo isto é especificado pela DII, que precisa realizar várias chamadas de rotinas de requisição, para que possa obter todos estes dados, que são recuperados nos IR específicos.

Estes acessos são transparentes, ou seja, não importa se o IR é local ou remoto. O problema é que pode ser acumulado muito *overhead*, caso vários acessos remotos necessitem ser feitos. Segundo [COULOURIS, 2001] o cliente pode obter do repositório de interfaces as informações necessárias a respeito de métodos disponíveis para um dado objeto CORBA. O cliente pode usar esta informação para construir as invocações com argumentos apropriados e mandá-las para o servidor, através das seguintes rotinas:

- *Create_request()*: Cria uma requisição ao ORB, que será invocada por uma chamada à rotina *invoke()*, ou utilizando *send()* e *get_response()*;
- *add_arg()*: Adiciona argumentos incrementalmente a uma requisição;
- *invoke()*: Chama o ORB, que faz a invocação do método apropriado, retornando o controle para o cliente após concluída a operação;
- *delete()*: Apaga uma requisição que estava sendo montada;
- *send()*: Inicia uma operação de modo assíncrono, retornando o controle para o cliente sem aguardar o término da operação;
- *send_multiple_request()*: Inicia várias requisições em paralelo, retornando o controle ao cliente sem aguardar que as operações sejam concluídas;
- *get_response()*: Informa se uma requisição específica foi completada; dependendo dos *flags* passados em sua chamada, pode retornar imediatamente ou guardar, caso a requisição ainda não tenha sido completada;
- *get_next_response()*: Informa se alguma requisição foi completada; dependendo dos *flags*, pode retornar imediatamente ou aguardar o fim de uma requisição, caso não haja nenhuma pendente.

Para que seja invocada, a requisição deverá primeiramente receber valores de argumentos (que poderão ser determinados a partir do IR), e então invocar operações

diretamente sobre o pseudo-objeto. Em suma, isto significa que um pseudo-objeto será criado e valores de argumentos atribuídos a ele. A partir de então, existem três formas de se efetuar uma invocação:

- **Invocação Síncrona:** ao emitir a requisição, o cliente bloqueia a espera até que a resposta chegue. Este é o tipo mais utilizado pelo CORBA, devido também ser suportado pelos *stubs* estáticos.
- **Invocação Síncrona Deferida:** podem ser emitidas várias requisições e as respostas recebidas na medida que forem chegando.
- **Invocação *Oneway*:** a requisição é emitida e não espera por uma resposta de volta.

3.6.7.2 Interface de *Skeleton* Dinâmico

Como visto anteriormente, a DII possibilita que os clientes emitam requisições sem a necessidade de acessar *stubs* estáticos. O mesmo acontece no lado dos servidores, porém com a interface de *skeleton* dinâmico (DSI). A DSI permite ao ORB distribuir requisição para uma implementação de objeto, mesmo que não sejam conhecidas em tempo de compilação informações como: o tipo de servidor e a sua implementação.

Isto permite que um objeto CORBA aceite invocações de uma interface na qual este não tem *skeleton* porque o tipo de sua interface não é conhecido em tempo de execução. Quando um *skeleton* dinâmico recebe uma invocação, ele inspeciona o conteúdo do requerimento para descobrir seu objeto alvo, o método a ser invocado e os argumentos. E então, o objeto alvo é invocado [COULOURIS, 2001].

3.6.8 Adaptador de Objetos

O adaptador de objetos é o elemento responsável pela interconexão do ORB com a implementação dos objetos servidores, um ORB pode ser interfaceado por mais de um adaptador de objetos ao mesmo tempo, sendo eles destinados a determinadas aplicações e com requisitos de performance diferentes.

Para atender uma grande quantidade de aplicações, tornando desnecessária a proliferação demasiada de adaptadores de objetos, a OMG propôs a especificação de um adaptador básico de objetos, chamado BOA (*Basic Adapter Object*), que deve estar disponível em todas as implementações de ORBs [OMG, 2002]. As funções de um adaptador de objetos são mostradas na figura 3.7, e são as seguintes:

- Ativar e registrar as implementações junto ao ORB;
- Ativar os objetos;
- Utilizar os *skeletons* IDL, para chamar os métodos do objeto;
- Interfacear os serviços providos pelo ORB.

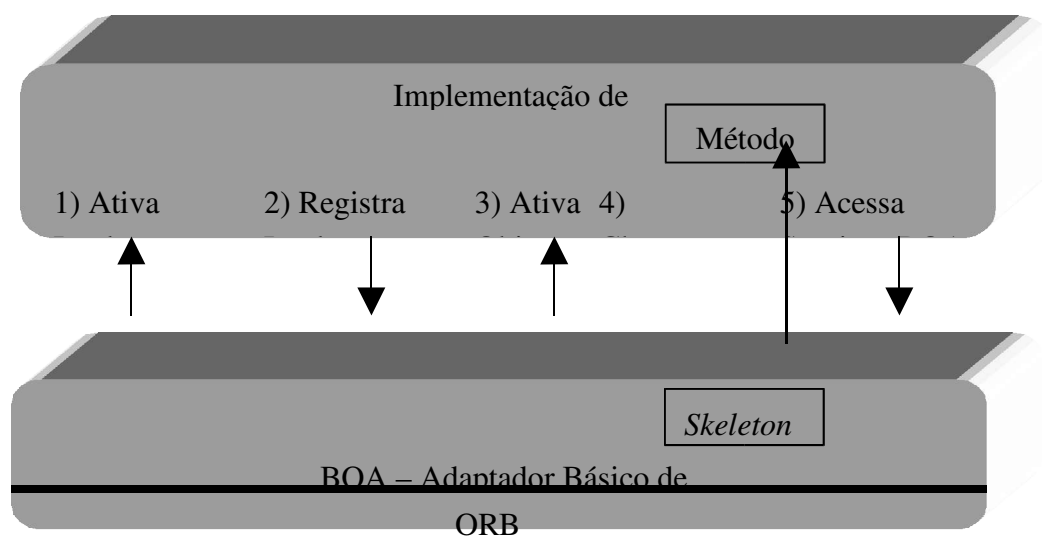


Figura 3.7 – Estrutura Operacional do BOA

Um adaptador de objetos fornece a cada objeto CORBA um nome único, o qual faz parte da sua referência de objeto remoto. O mesmo nome é usado cada vez que o objeto é ativado. O nome do objeto pode ser especificado pelo programa aplicativo ou gerado pelo adaptador de objetos. Todo objeto CORBA é registrado em um adaptador de objetos, o qual pode manter uma tabela de objetos remotos que mapeia os nomes dos objetos CORBA para as suas implementações [COULOURIS, 2001].

3.6.9 Interoperabilidade

Como previsto pela OMG, existe atualmente uma grande variedade de produtos que obedecem a especificação CORBA, entretanto devido à grande flexibilidade permitida, as implementações de cada ORB diferem, refletindo as soluções peculiares de cada fabricante, não só pela utilização de diferentes mecanismos para a obtenção das mesmas funcionalidades preconizadas, como também através do acréscimo de novas funções, consideradas importantes para o seu usuário final [OMG, 2002].

O CORBA para ser uma especificação completa, deveria possibilitar a interoperabilidade entre ORBs de diferentes fabricantes, porém a interconexão de ORBs só pode ser alcançada através de *bridges*. No CORBA, as fronteiras de interoperabilidade são determinadas por Domínios. Domínio foi a forma que o OMG criou para agrupar os objetos

que possuem características semelhantes, sejam elas relativas à implementação ou puramente administrativa.

Existem dois tipos de interoperabilidade entre ORBs no CORBA, a direta e a indireta. Na direta, os domínios estão ligados diretamente através de *bridges*. As *bridges* diretas ligam dois domínios, traduzindo de forma direta o protocolo de um para o protocolo do outro. Sua vantagem é que são bem rápidas e eficientes. Porém, são muito inflexíveis, pois deverá haver uma *bridge* para cada par de domínios.

Já as *bridges* indiretas têm-se mais flexibilidade, pois elas traduzem o protocolo de um domínio, primeiramente para um protocolo comum, deixando a cargo da *bridge* destinatária a função de traduzir para o protocolo de seu domínio. Este protocolo comum pode ser o IIOP (*Internet Inter-Orb Protocol*), que é um padrão especificado pelo OMG.

Um outro aspecto importante, que precisa ser comentado, é referente ao BOA, pois os fabricantes de ORBs desenvolveram suas próprias extensões proprietárias, devido à falta de algumas características no BOA, o que gerou um problema de interoperabilidade. Segundo [COULOURIS, 2001], para solucionar este problema foi criado no CORBA 2.2 um outro padrão de adaptadores de objetos, que é chamado de POA (*Portable Object Adapter*). Ele é chamado de portátil porque permite que aplicativos e objetos servidores sejam executados em ORBs produzidos por diferentes desenvolvedores.

Para que a interoperabilidade entre ORBs seja alcançada, o POA permite que programadores de sistemas penetrem no código interno do servidor CORBA, fazendo com que se possa controlar quase todos os aspectos do ambiente servidor.

3.6.10 Sistema Legado

Utilizando CORBA, pode-se reprojeter um sistema legado, de forma que ele possa ser adaptado à tecnologia de objetos distribuídos. Para isto, basta que uma parte do sistema legado seja criada como um objeto CORBA, através da definição de uma interface IDL, e que seja fornecida uma implementação de um adaptador de objeto e *skeleton*.

3.6.11 Protocolo IIOP

Usando o protocolo IIOP como padrão, torna-se possível descartar o uso do protocolo HTTP (*HiperText Transfer Protocol*), que é considerado uma tecnologia de baixo desempenho por não manter estado. No HTTP, para cada requisição é iniciada uma conexão, que é terminada quando os dados são recebidos. Isto é, para cada requisição é necessário um processo de abertura de conexão. Já com o IIOP, o cliente abre uma conexão TCP/IP com o servidor e envia uma ou mais requisições IIOP ao servidor na mesma conexão [RICCIONE, 2000].

3.6.12 Novidades no CORBA 3.0

Nos primeiros anos, CORBA concentrou-se na definição da infra-estrutura e sua integração com linguagens de programação como C++. Mais recentemente, o padrão passou a abordar também a inclusão das áreas verticais e a integração com novas tecnologias, como Java. Atualmente, o padrão está incorporando novidades como XML (*Extensible Markup Language*) e *Enterprise JavaBeans*. Segundo a [OMG, 2002], as principais novidades para o CORBA 3 deverão ser:

- Garantia de qualidade de serviço: o que inclui a adaptação de CORBA para sistemas de tempo real, *middlewares* orientados à mensagem e a definição de CORBA Mínimo para sistemas embutidos;
- Suporte à portabilidade de objetos: complementando a integração Java/RMI com IIOP;
- A definição do novo *Component Implementation Definition Language*: proporcionando mapeamento direto em *Enterprise JavaBeans*, e padronizando soluções semelhantes em outras linguagens.

3.6.12.1 Integração com Java/Internet

O mapeamento Java/RMI para CORBA permite que os objetos Java/RMI possam interoperar com a rede como objetos CORBA [OMG, 2002]. Isto faz com que agora os programadores Java possuam uma linguagem padrão sem conhecer IDL. O problema da nova abordagem é que os programadores não possuirão à disposição as vantagens disponibilizadas pela IDL, como mensagens assíncronas e qualidade de serviço, dentre outras funcionalidades que estão sendo disponibilizadas.

Os objetos Java, tornando-se objetos CORBA, farão com que os usuários de Java estejam entrando no uso de um ambiente que não representa apenas uma linguagem. O usuário vai escrever um objeto servidor Java que automaticamente vai gerar uma saída IDL, permitindo que se acesse métodos de clientes CORBA escritos em qualquer linguagem.

A especificação *Firewall* do CORBA 3 define a capacidade que o CORBA precisa para atravessar o *Firewalls* com segurança. Originalmente foi adotada no final de 1998 e foi

revisada durante o ano de 2002. Esta especificação tem sido adotada, mas ainda não foi submetida ao seu ciclo inicial de manutenção [OMG, 2002].

A referência do objeto CORBA é um ponto fundamental da arquitetura. Possuir a referência do objeto é a única maneira de instanciá-lo e invocá-lo. Mesmo que se soubesse onde o objeto está localizado, se não se conhecesse a referência nada adiantava. CORBA 3, define uma maneira fácil de buscar um objeto, através de um URL (*Unified Resource Locator*), definido como no exemplo: `iioploc://www.omg.org/NameService`. O `iioploc` é usado para localizar serviços definidos em determinada localização. Um segundo formato definido, `iiopname`, vai realizar a busca usando o nome fornecido pelo usuário existente na URL passada, retornando o IOR (*Interoperable Object Reference*) do objeto [OMG, 2002].

3.6.12.2 Gerência da Qualidade de Serviço

Dentro da gerência e qualidade de serviço estão definidos tópicos como a invocação e a troca mensagens assíncronas. Serão colocadas quatro extensões na arquitetura, onde os clientes poderão realizar requisições síncronas adiadas via IDL *stubs*, fazer requisições que terminam depois que o cliente termina, controlar a qualidade de serviço associada a uma dada requisição e ainda poderão controlar a ordem de múltiplas requisições enfileiradas. Com relação às chamadas assíncronas, os clientes poderão especificar, no momento da invocação, se desejam que o retorno seja feito via retorno do objeto, conhecido como *callback*, ou via *polling*.

Uma outra novidade é o CORBA Mínimo, que é uma arquitetura CORBA reduzida, proposta para ser usada em sistemas embutidos, como por exemplo, chips que são fixos e

possuem uma interação previsível com o mundo externo. Não possuem aspectos dinâmicos de CORBA, sendo eliminados os conceitos de DII (Dynamic Interface Invocations), DSI (Dynamic Skeleton Interface), Dynamic Any (DynAny), repósitório de interfaces, algumas características e políticas implementadas pelo POA (*Portable Object Adapter*) e todos os protocolos, com exceção do GIOP e IIOP. O “CORBA Mínimo” suporta todos os tipos de dados definidos na IDL [OMG, 2002]. Outros importantes aspectos, como o suporte para ambientes de tempo real e a tolerância a falhas, também são tratados no CORBA 3. Maiores informações sobre os itens apresentados podem ser encontradas no site da OMG.

3.6.12.3 Componentes Distribuídos

Segundo a [OMG, 2002], os componentes CORBA representam um avanço com vários benefícios para programadores, usuários e consumidores e componentes de software. As três partes mais significativas dos componentes CORBA são:

- Ambiente que empacota transacionalidade, segurança e persistência, e provê interface e resolução de eventos;
- Integração com *Enterprise JavaBeans* (EJB);
- Um formato de distribuição de software que permita independência de plataforma.

EJBs vão agir como componentes CORBA e podem ser instalados em um agregador de componentes CORBA. Ao contrário de EJBs, os componentes CORBA podem ser escritos em múltiplas linguagens e suportar múltiplas interfaces. A especificação define um formato de distribuição de software de multi-plataforma, incluindo um instalador e ferramenta XML para configuração. Uma especificação separada vai tratar de linguagens de script que irão mapear objetos CORBA, como JavaScript, Perl, Tcl e Python [OMG, 2002].

3.6.13 A Integração CORBA e Java/RMI

Ultimamente, têm surgido diversas propostas de padronização e implementação, principalmente dentro da OMG, no sentido de integrar CORBA e Java, trazendo a vantagens de ambas as tecnologias, principalmente no que se refere à utilização através da Web. Esta integração promete ser a grande novidade nas tecnologias de objetos distribuídos deste início de século. Na OMG, existem dois grupos trabalhando no sentido de especificar esta integração.

A primeira seria no sentido do mapeamento da IDL para Java, permitindo a geração do *stubs* e *skeletons* escritos em Java, seguindo a mesma idéia do mapeamento IDL, para outras linguagens de programação já padronizadas pela OMG (C, C++, COBOL, Ada, *Smalltalk*, etc). A segunda é uma idéia não menos interessante, e consiste no mapeamento de Java para IDL. Neste caso, o programador teria uma forma de construir aplicações distribuídas diretamente em Java, que se integraria com o resto do ambiente heterogêneo, via IIOP.

Através da geração de IDL a partir do código Java, muitas linguagens de programação teriam acesso a esses componentes escritos em Java, através do IIOP. A própria SUN tem feito estudos de viabilidade para o Java/RMI adotar o protocolo IIOP, e tudo indica que isso é bastante provável [RICCIONE, 2000].

3.6.14 Considerações Finais

Conforme dito anteriormente, o ORB permite que clientes escritos em uma determinada linguagem invoquem operações de objetos remotos (chamados de objetos

CORBA), escritos em uma linguagem diferente [COULOURIS, 2001]. CORBA trata também de outros aspectos de heterogeneidade, como segue:

- O GIOP inclui uma representação de dados externos chamado de CDR (*Common Data Representation*), o qual torna possível clientes e servidores se comunicarem independentemente dos seus hardwares. Ele também especifica uma forma padrão de referência a objetos remotos.
- GIOP também inclui uma especificação para as operações do protocolo de requisição/resposta, o qual pode ser usado independente do sistema operacional utilizado.
- O IIOP implementa o protocolo de requisição/resposta utilizando TCP/IP. As referências de objetos remotos do IIOP incluem o nome do domínio, o número da porta e o servidor.

Um objeto CORBA implementa as operações existentes em uma interface IDL. Tudo que o cliente precisa saber para acessar o objeto CORBA são as operações disponíveis em sua interface. O programa cliente acessa objetos CORBA via *stubs*, os quais são gerados automaticamente pela linguagem do cliente, baseados nas interfaces IDL. Os *skeletons* do servidor para objetos CORBA também são gerados automaticamente baseados nas interfaces IDL, porém na linguagem do servidor. O adaptador de objetos é um componente importante dos servidores CORBA. Seu papel é criar referências a objetos remotos e relacionar essas referências nas mensagens de requisições com as implementações dos objetos CORBA [COULOURIS, 2001].

A arquitetura CORBA permite que objetos CORBA sejam ativados sob demanda. Isto é conseguido através do repositório de implementação, o qual guarda um bloco de dados de implementações indexados pelos nomes de seus adaptadores de objetos. Quando o cliente

invoca um objeto CORBA, este pode ser ativado se necessário para atender a invocação [COULOURIS, 2001].

Um repositório de interface é um banco de dados de definições de interfaces IDL, ele pode ser usado para obter informações sobre os métodos disponíveis na interface dos objetos CORBA, para permitir as invocações dinâmicas dos métodos [COULOURIS, 2001].

4 Arquitetura Multicamadas e Projeto do Sistema

4.1 Introdução

Neste capítulo será feita uma breve descrição das arquiteturas com uma, duas e três camadas, dando maior ênfase em três camadas, já que esta será a arquitetura a ser utilizada no desenvolvimento do WebCELLP. Será também apresentado o padrão de projeto *Model View Controller* (MVC), onde será detalhado cada elemento desse padrão.

4.2 Arquitetura com Uma e Duas Camadas

Utilizando as ferramentas que qualquer linguagem de programação oferece, pode-se criar aplicações conhecidas como *Desktop* ou de uma camada, onde será adicionada toda a funcionalidade da aplicação (as regras de negócios) à interface do usuário, através dos manipuladores de eventos. Entretanto, esta é uma técnica que gera uma série de problemas como, por exemplo, baixa de performance e dificuldade de distribuição. Veremos logo a seguir como particionar uma aplicação em mais de uma camada, de forma que a mesma obtenha um melhor rendimento.

Quando particiona-se uma aplicação com uma camada em duas, questões de performance são a maior preocupação. Com tabelas locais, performance não é uma coisa crítica, uma vez que geralmente apenas um usuário está tentando acessar um banco de dados. Entretanto, em aplicações com duas camadas vários usuários tentam acessar o banco de dados baseados em um servidor, o que pode causar grandes problemas de performance. Eis porque otimização é algo tão crítico neste tipo de aplicação. Otimizar uma aplicação para realizar

exatamente as mesmas tarefas, com o mínimo de esforço de um banco de dados, pode melhorar a performance de uma maneira impressionante.

Uma forma muito eficiente de otimizar uma aplicação é armazenar as regras de negócios em *stored procedures*, que nada mais são do que procedimentos armazenados no servidor de banco de dados. *Stored procedure* normalmente constitui-se de instruções que são executadas repetidamente no próprio banco de dados, e muitas vezes os resultados da operação são passados à aplicação cliente. Com elas, além de simplificar e melhorar a performance da aplicação cliente, pode-se reduzir o tráfego na rede e o tempo de resposta ao usuário. A figura 4.1 mostra uma aplicação de duas camadas com as regras de negócios armazenadas no banco de dados.

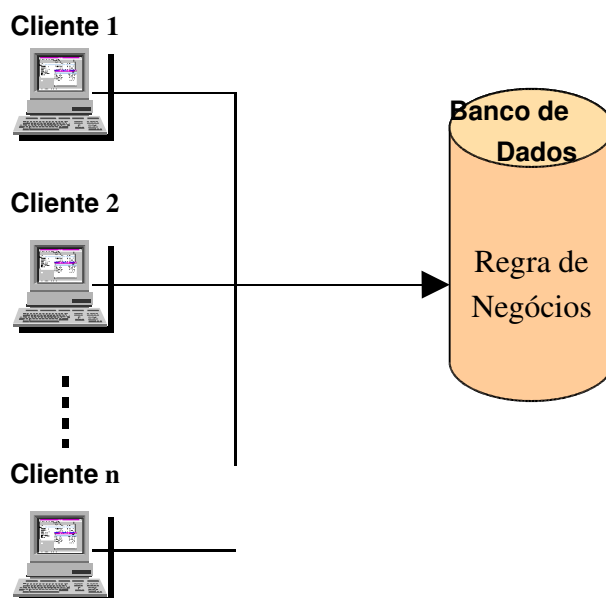


Figura 4.1 – Arquitetura de duas camadas

A maior parte das implementações de servidor de banco de dados suportam muitas características que podem isolar um aplicativo dos dados subjacentes sobre os quais atua. Este nível de isolamento torna mais fácil o desenvolvimento de consistentes metodologias de

acesso a dados, como por exemplo *stored procedure*, para uma gama de aplicativos e plataformas de desenvolvimento. Isto permite que o servidor de bancos de dados execute a validação de dados necessária e reforce as regras de negócios que definem como os dados serão acessados pelo aplicativo cliente, permitindo também que seja alterada livremente as regras sem interferir na interface do cliente [BORLAND SOFTWARE CORPORATION, 1997].

Estas regras, além de poderem ser colocadas na camada servidora, podem também ser colocadas na camada cliente, ou ainda podem ser divididas entre ambas. Porém, a decisão de onde posicionar as regras de negócios, deve ser muito bem investigada e designada durante a fase de projeto. Foi comentado anteriormente o problema que há quando se posicionam as regras de negócios no cliente. Tratando-se do posicionamento das regras no servidor, podem ser citados como vantagens e desvantagens, os seguintes aspectos:

Vantagens:

- A criação da integridade de referência realiza-se melhor ao nível de servidor, pois todos os Sistemas de Gerenciamento de Banco de Dados Relacionais (SGBDR), tem esta capacidade [BORLAND SOFTWARE CORPORATION, 1997];
- Como já dito anteriormente, o fato de manter as regras perto dos dados e em uma localização central permite que quaisquer alterações ou atualizações das regras que se façam necessárias possam ser realizadas, sem que isso interfira no acesso de dados do aplicativo cliente;
- Pode-se limitar o acesso de dados a indivíduos, de forma programada nos aplicativos clientes. Porém, isto não assegura que estes dados não serão acessados por outras ferramentas [BORLAND SOFTWARE CORPORATION, 1997].

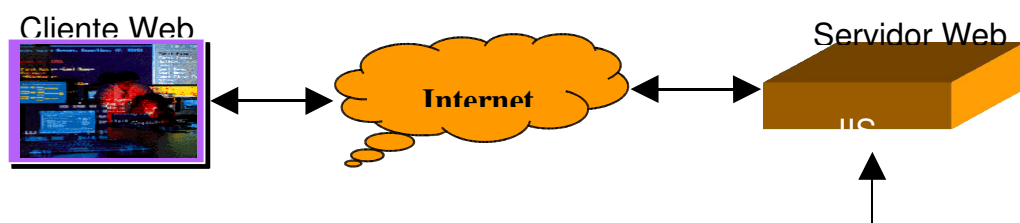
Desvantagens:

- Supondo que haja a necessidade de se aplicar um determinado conjunto de regras de negócios a um cliente em particular, e caso estas regras sejam posicionadas no servidor, elas estariam sendo impostas a todos os clientes;
- Há muitas operações que podem ser criadas no cliente mais facilmente e melhor, pois a linguagem utilizada no aplicativo cliente além de ser mais robusta, possui ferramenta para depurar e testar a aplicação;
- A validação de dados sendo feita no servidor levará um tempo maior do que se fosse feita no cliente, pois este processo fará com que o usuário espere que o servidor valide os dados [BORLAND SOFTWARE CORPORATION, 1997].

Fazendo uma análise das vantagens e desvantagens do posicionamento das regras de negócios no servidor, percebe-se claramente que, para a maior parte das aplicações, a escolha ideal é a divisão destas regras entre a interface do usuário e o servidor, pois quando esta divisão é bem aplicada, ela pode atender a várias situações e ambientes. No entanto, carrega os estigmas da dificuldade de distribuição e falta de portabilidade.

4.3 Arquitetura com Três Camadas

A arquitetura de três camadas teve como principal novidade a adição de uma nova camada, comumente chamada de servidor de aplicação, onde serão armazenadas as regras de negócios da aplicação. Desta forma, a aplicação cliente não faz mais as suas solicitações diretamente ao servidor de banco de dados, mas sim ao servidor de aplicação, que por sua vez repassa estas solicitações para o servidor de banco de dados, a figura 4.2 mostra arquitetura de três camadas.



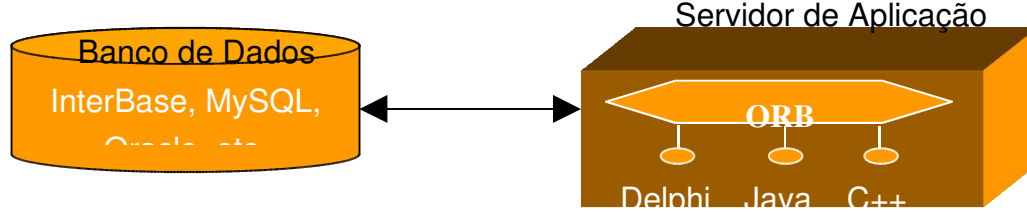


Figura 4.2 – Arquitetura de três camadas

Um exemplo prático desta arquitetura são as aplicações para *WEB* com concepção moderna, que são construídas utilizando arquitetura de projeto de três camadas [BROWN; et al., 2001], as quais apresentam as seguintes funções:

- Primeira camada: é a interface do usuário, como por exemplo, a interface gráfica da aplicação apresentada em páginas *Web*;
- Segunda Camada: reside dentro do servidor de aplicação, e é a mediadora da comunicação entre as camadas;
- Terceira Camada: é o *back-end* de dados ou transações, incluindo parte das regras de negócios para o processamento de transações, além do próprio banco de dados.

A nova camada, presente nesta arquitetura, acrescenta algumas vantagens em comparação com as arquiteturas de uma e duas camadas, e dentre elas podemos destacar:

- A centralização das regras de negócios, o que facilitará as futuras atualizações, já que qualquer modificação feita no servidor de aplicação não afetará a aplicação cliente;

- A possibilidade de escrever as regras de negócios através de classes em uma linguagem expressiva como o *Object Pascal*, ao invés de utilizar a linguagem SQL (*Structured Query Language*);
- A distribuição do trabalho de uma aplicação em várias máquinas, o que melhora bastante o desempenho por causa do balanceamento de carga;
- O fato de poder isolar funcionalidade em camadas intermediárias com restrições de acesso diferente, o que fornece níveis de segurança flexíveis e configuráveis, pois as camadas intermediárias podem limitar os pontos de entrada, permitindo controlar o acesso mais facilmente [ARAÚJO, 2000].

Pode ser citado como um exemplo do modelo com três camadas, o padrão de projeto *Model View Controller* – MVC, originalmente desenvolvido na década de 1970 pela Xerox PARC para a implementação de objetos *Interfaces Gráficas de Usuários* (GUI) [BROWN; et al., 2001], posteriormente foi adaptado para outros tipos de aplicações.

4.4 Projeto do Sistema

Projetar software orientado a objeto é uma tarefa difícil, mas projetar software reutilizável orientado a objeto é mais difícil ainda. Deve-se achar objetos pertinentes, fatorá-los em classes, definir as interfaces das classes e as hierarquias de herança e estabelecer as relações-chave entre elas. O projeto deve ser específico para o problema a resolver, mas também genérico o suficiente para atender futuros problemas e requisitos, pois se deve tentar evitar ou pelo menos minimizar o re-projeto [GAMMA; et al., 2000]. Criar um projeto reutilizável e flexível é difícil, se não impossível, de obter-se corretamente logo na primeira vez.

Desenvolver um projeto do zero tem uma demanda de tempo muito grande, portanto é comum reutilizar soluções que funcionaram bem no passado. Conseqüentemente, é comum encontrar padrões de classes e de comunicação entre objetos, que reaparecem freqüentemente em muitos sistemas orientados a objetos. Estes padrões resolvem problemas específicos de projetos e os tornam mais flexíveis e reutilizáveis.

Os padrões de projeto tornam mais fácil a tarefa de reutilizar projetos e arquiteturas bem sucedidas, eles podem melhorar a documentação e a manutenção do sistema ao fornecer uma especificação explícita de interações de classes e objetos e o seu objetivo, ou seja, ajuda a obter-se um projeto correto mais rapidamente. Os padrões de projeto tornam um sistema menos complexo, ao permitir falar sobre ele em um nível de abstração mais alto do que se fosse falar em uma linguagem de programação.

Um padrão de projeto nomeia, abstrai e identifica os aspectos-chave de uma estrutura de projeto comum, para torná-la útil para a criação de um projeto orientado a objetos reutilizável. O padrão de projeto identifica as classes e instâncias participantes, seus papéis, colaborações e a distribuição de responsabilidade[GAMMA; et al., 2000]. Cada padrão de projeto focaliza um problema ou tópico particular. Ele descreve quando pode ser aplicado, se pode ser aplicado em função de outras restrições de projeto e as conseqüências, custos e benefícios de sua utilização.

4.5 O Padrão de Projeto *Model View Controller* (MVC)

Como visto anteriormente, o MVC é um padrão que possibilita que uma aplicação seja dividida em três objetos, sendo eles: o Modelo, a Vista e o Controlador, que trocam

mensagens entre si, conforme ilustra a figura 4.3. O Modelo é o objeto da aplicação, a Vista é a interface do usuário que será apresentada na tela e o Controlador define a maneira que a interface do usuário irá reagir às suas intervenções. Antes do MVC, os projetos de interface para o usuário tendiam a agrupar esses objetos, porém a separação desses objetos ajuda a aumentar a flexibilidade e a reutilização.

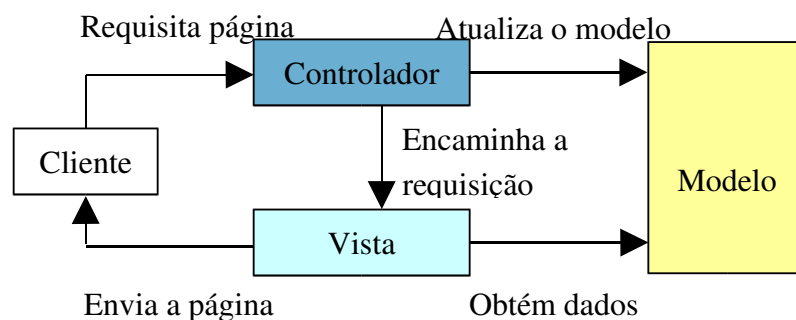


Figura 4.3 – Visão Geral do Modelo MVC

O efeito resultante do particionamento de um sistema em uma coleção de classes cooperantes é a necessidade de manter a consistência entre objetos relacionados. Esta consistência não deverá ser obtida com classes fortemente acopladas, porque isso reduz a capacidade de ser reutilizada.

Analisando a figura 4.3, percebe-se que a vista representa o estado do modelo em um dado instante, e uma forma ainda mais simples de representar o relacionamento entre modelo e vista é mostrada na figura 4.4. Esta figura contém um modelo e três visões (por simplificação, não estão sendo mostrados os controladores), o modelo contém alguns valores de dados, e as visões são representadas por uma planilha, um histograma e um gráfico de pizza, apresentando os dados do modelo de várias maneiras.



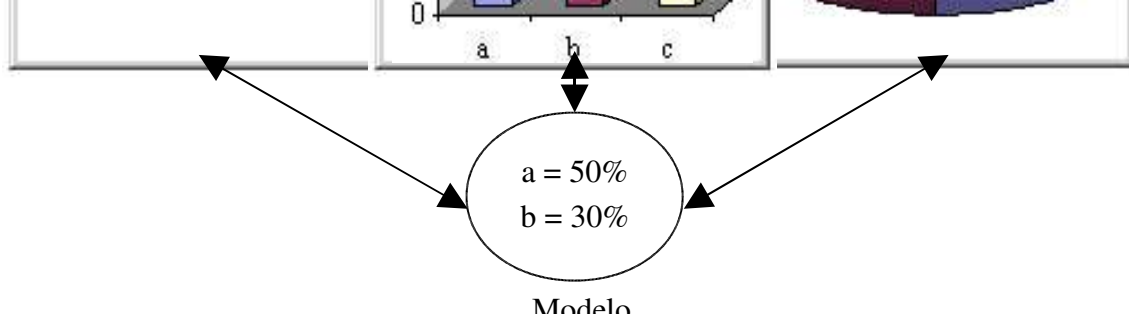


Figura 4.4 – Representação de três Visões baseadas em um único Modelo

O modelo permite uma clara separação entre apresentação e dados, encapsulando os dados e todos os métodos utilizados para acessá-los, possibilitando então um total isolamento entre as camadas e permitindo, portanto que qualquer alteração no modelo como modificações no banco de dados ou criação de novas regras de negócios não impliquem em modificações na interface do usuário.

Por outro lado, também podem ser construídas novas e diferentes *interfaces* sem que o banco de dados ou as regras de negócios sejam modificados. Esta filosofia de separação define regras transparentes entre uma equipe de projeto, visto que o responsável pelo designer não precisa se preocupar com a lógica de acesso ao banco de dados ou aos próprios dados, assim como o desenvolvedor pode criar suas rotinas e classes sem se preocupar com a *interface* do usuário final [GOMES, 2002].

4.6 Componentes da Arquitetura MVC

4.6.1 O Modelo (*Model*)

O Modelo é o objeto da aplicação, ou seja, ele representa os dados e as regras de negócios de uma aplicação. Conceitualmente, o Modelo é independente do Controlador e da Vista. Segundo [GOMES, 2002], o Modelo é constituído de componentes de estado e componentes de ações. Os componentes de estado definem o conjunto de valores atuais do modelo e incluem métodos para mudar esses valores.

Os componentes de ação definem as mudanças de estados em resposta aos eventos que ocorrem na aplicação. Numa arquitetura MVC, o Controlador é quem gerencia todos os eventos e faz as chamadas apropriadas aos componentes de ação.

Como a maioria das aplicações possui dados para serem armazenados em banco de dados, este banco pode ser mapeado para uma entidade separada onde a comunicação com o sistema é feita através do Modelo, como ilustra a figura 4.5.

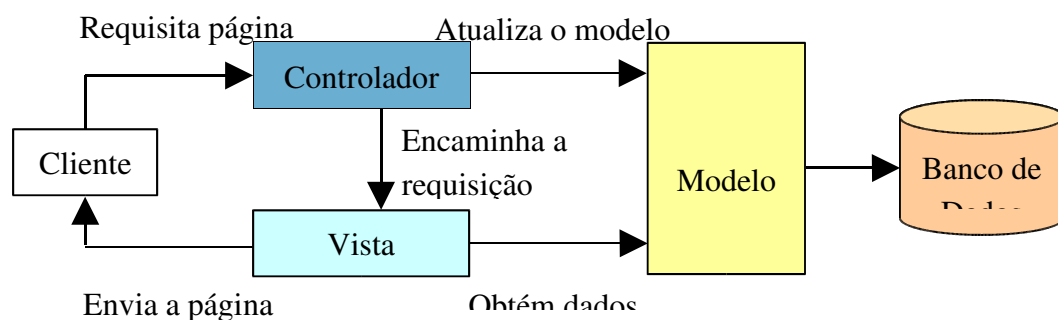


Figura 4.5 – Modelo MVC incorporando Banco de Dados

4.6.2 A Vista (*View*)

No MVC a Vista é a única parte visível da aplicação para o usuário, ela é responsável pela apresentação das informações ao mesmo, e deve também assegurar que estas informações reflitam o estado do Modelo em um dado momento. Isto pode ser feito de duas maneiras: uma seria feita pelo Modelo, que, de posse dos dados alterados, informaria à Vista as respectivas alterações para que ela se atualize; e a outra seria de responsabilidade da própria Vista, ou seja, ela é quem verificaria se está de acordo com as informações contidas no Modelo.

Como visto anteriormente, o fato de a Vista ser separada do Modelo permite que sejam ligadas múltiplas Visões a um Modelo para fornecer diferentes apresentações. Sendo assim, elas têm a responsabilidade de apresentar um conjunto específico de eventos que o usuário pode ativar em um dado momento.

4.6.3 O Controlador (*Controller*)

Segundo [GOMES, 2002], o Controlador age como o agregador do sistema. Ele realiza a gerência das entradas do usuário, ou seja, ele define a maneira com que a *interface* reage às intervenções do usuário e como se comunica com o Modelo. O Controlador recebe eventos, determina a página e/ou a ação apropriada ao evento e dispara a geração de uma resposta à solicitação. Nesta arquitetura o Controlador age também como um mensageiro e deve atender a certos requisitos como:

- Segurança: o Controlador deve garantir a realização de tarefas relacionadas à segurança, como autenticação;

- Identificação e processamento de eventos: deve identificar um evento específico ocorrido e enviar a solicitação para seu destino apropriado;
- Interação com o Modelo: deve garantir a disponibilidade dos componentes solicitados ao Modelo, como o instanciamento de classes, comunicação com banco de dados, etc;
- Tratamento de Erros: deve controlar qualquer evento de erro gerado. O que pode ser feito através do uso de exceções e geração de páginas de Erro;
- Geração de respostas: deve enviar o controle para o gerador de respostas.

5 Estudo de Caso

5.1 Introdução

Neste capítulo será apresentado um detalhamento sobre o WebCELLP que, como dito anteriormente, são compostos pelos seguintes módulos:

- **Predict:** desenvolvido principalmente para realizar estudos estatísticos comparativos dos modelos de predição. Tal comparação visa avaliar o desempenho dos mesmos frente aos dados de uma campanha de medições realizada em uma determinada região.
- **Radio Link:** desenvolvido para realizar o balanço de potência do sistema, determinando o raio máximo de cobertura de um determinado modelo para os dois enlaces básicos (descida e subida).
- **Traffic Design:** desenvolvido para realizar o projeto de tráfego a partir de um mapa de tráfego produzido em qualquer formato gráfico usual (jpg, jpeg, bmp, emf ou wmf).

Serão apresentados os diagramas da UML que foram utilizados durante a fase de modelagem do WebCELLP, e também serão apresentadas as tecnologias envolvidas no desenvolvimento do mesmo, bem como suas características e requisitos de funcionamento.

5.2 Análise e Modelagem do Sistema

Para fazer a engenharia reversa do CELLP utilizou-se a UML por ser uma linguagem de modelagem que possui características fundamentais como:

- Possibilitar a visualização clara dos relacionamentos existentes entre os diversos componentes do sistema;
- Suportar múltiplas linguagens de programação, essencial já que o WebCELLP possui módulos implementados na linguagem de programação Delphi e em Kylix;
- Facilitar a reutilização e integração de sistemas, pois os módulos do WebCELLP estão prontos para serem reutilizados e integrados com outros módulos que por ventura surgirem.

5.2.1 Caso de Uso

Uma breve descrição dos cenários do ambiente é a seguinte: O usuário acessará uma página *Web* que contém um material teórico sobre planejamento de sistema móvel celular e também possui um link para a execução do WebCELLP, como mostra a figura 5.1. Porém, para ter acesso ao sistema, o usuário deverá solicitar e receber através de e-mail o seu login de acesso, como mostra a figura 5.2. O gerenciamento de usuário é feito pelo administrador, conforme é mostrado na figura 5.3.

De posse do login de acesso, o usuário será capaz de acessar o WebCELLP, e então executar um dos seus módulos, como mostra a figura 5.4. Nas figuras 5.5 e 5.6 são mostradas as interações entre os módulos, e na figura 5.7 são mostradas outras operações que o usuário poderá fazer, como por exemplo: criar, abrir e salvar um projeto, realizar simulações e emitir relatórios.

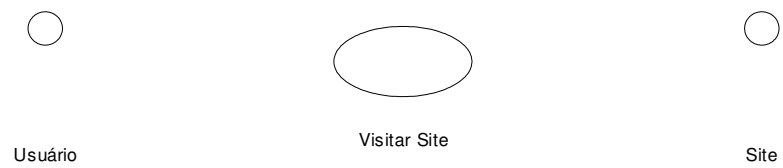


Figura 5.1 – Caso de uso visitação do site

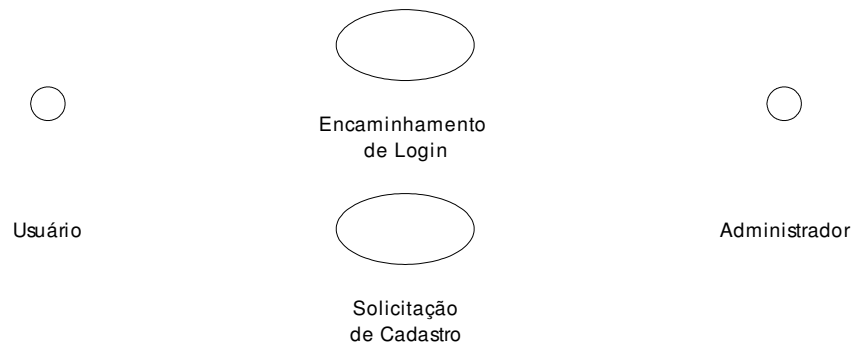


Figura 5.2 – Caso de uso solicitação de cadastro

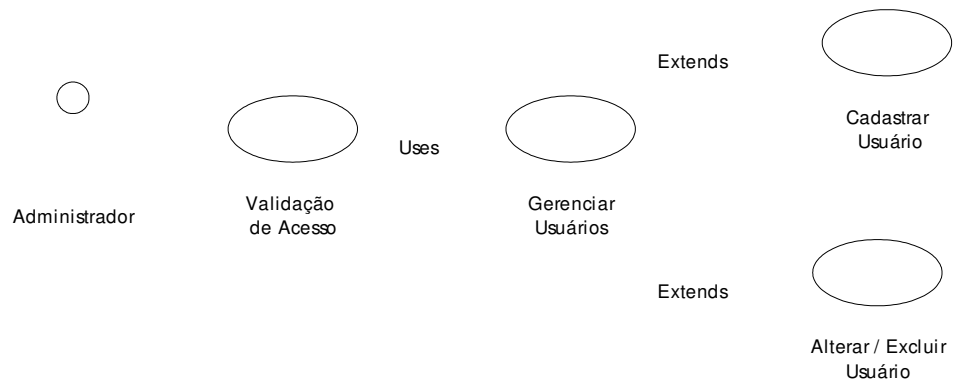


Figura 5.3 – Caso de uso gerenciamento de usuário

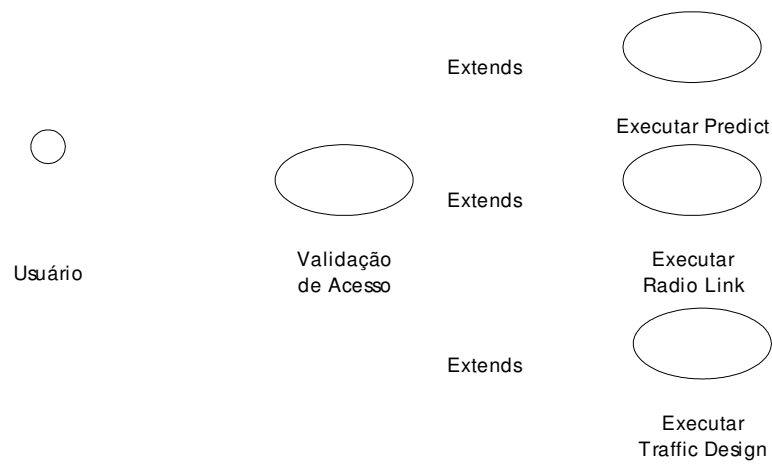


Figura 5.4 – Caso de uso acessar o WebCELLP

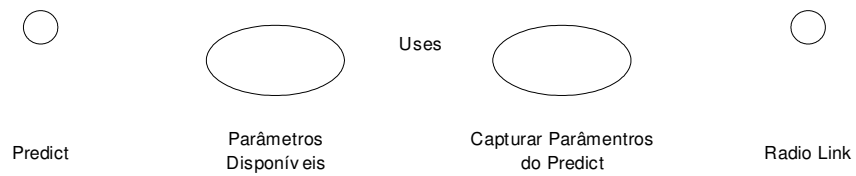


Figura 5.5 – Caso de uso execução do módulo Radio Link

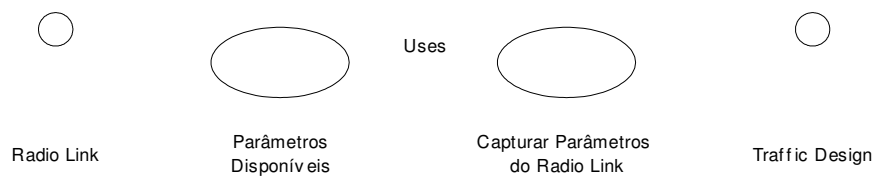


Figura 5.6 – Caso de uso execução do módulo Traffic Design

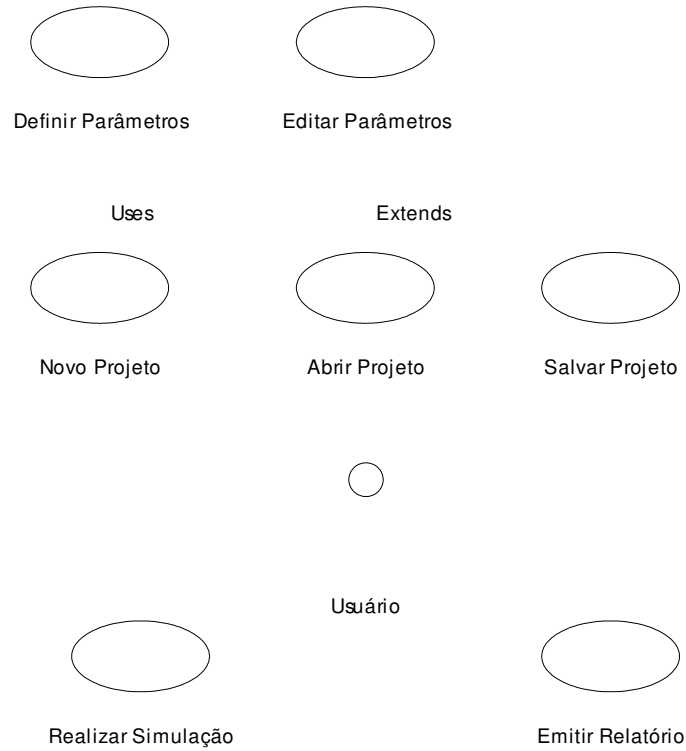


Figura 5.7 – Caso de uso manipulação de projeto

5.2.2 Diagrama de Classe

A seguir, é apresentado através da figura 5.8, o Diagrama de Classe do WebCELLP, visando uma melhor compreensão da comunicação entre essas classes.

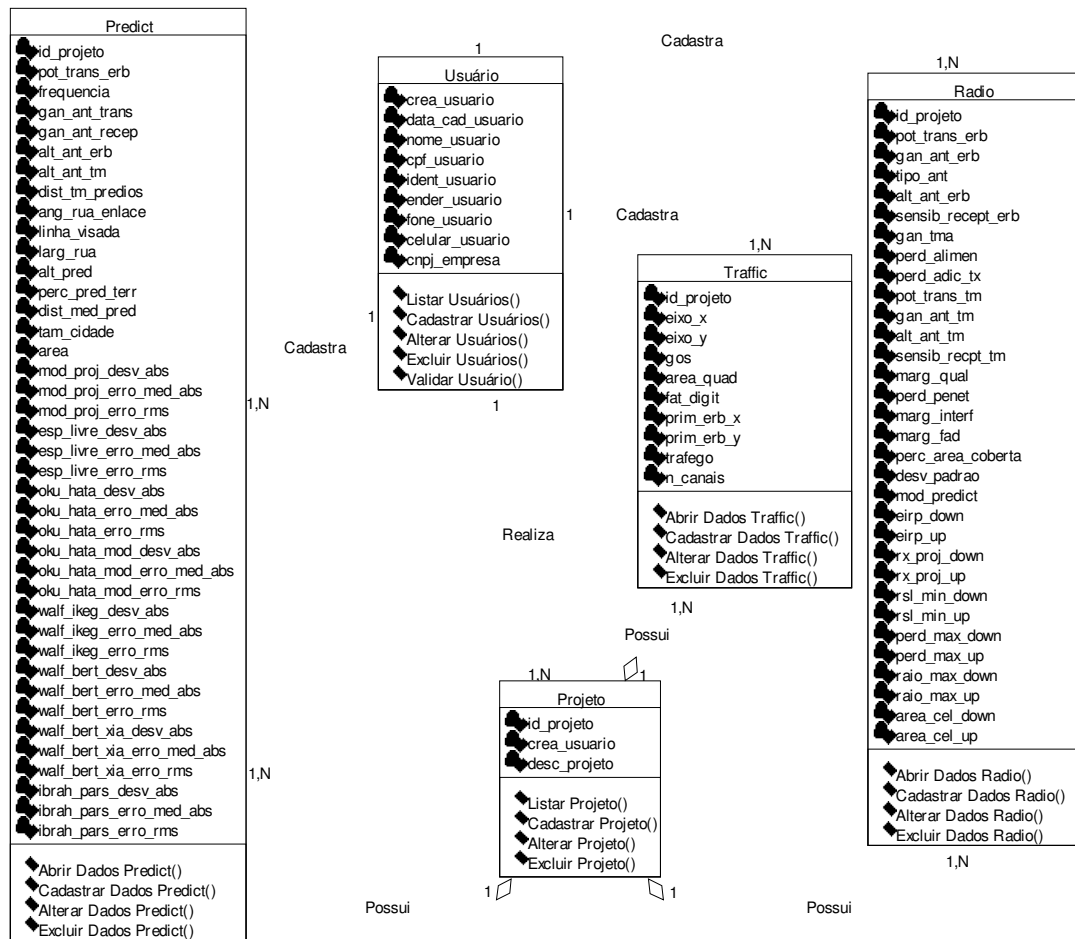


Figura 5.8 – Diagrama de classe do WebCELLP

5.2.3 Diagrama de Sequência

A seguir serão apresentados os diagramas de seqüência de interação para as principais tarefas do WebCELLP, tais como: logar no sistema (figura 5.9), abrir um projeto (figura 5.10), executar o módulo Predict (figura 5.11), criar e salvar um novo projeto (figura 5.12), alterar os dados de um projeto (figura 5.13), executar o módulo Traffic Design (figura 5.14) e executar o módulo Radio Link (figura 5.15).

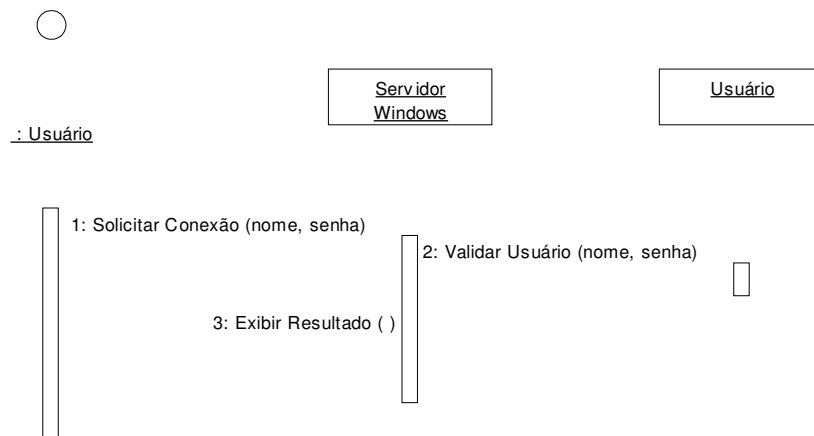


Figura 5.9 – Diagrama de seqüência de interações para o login de um usuário no WebCELLP

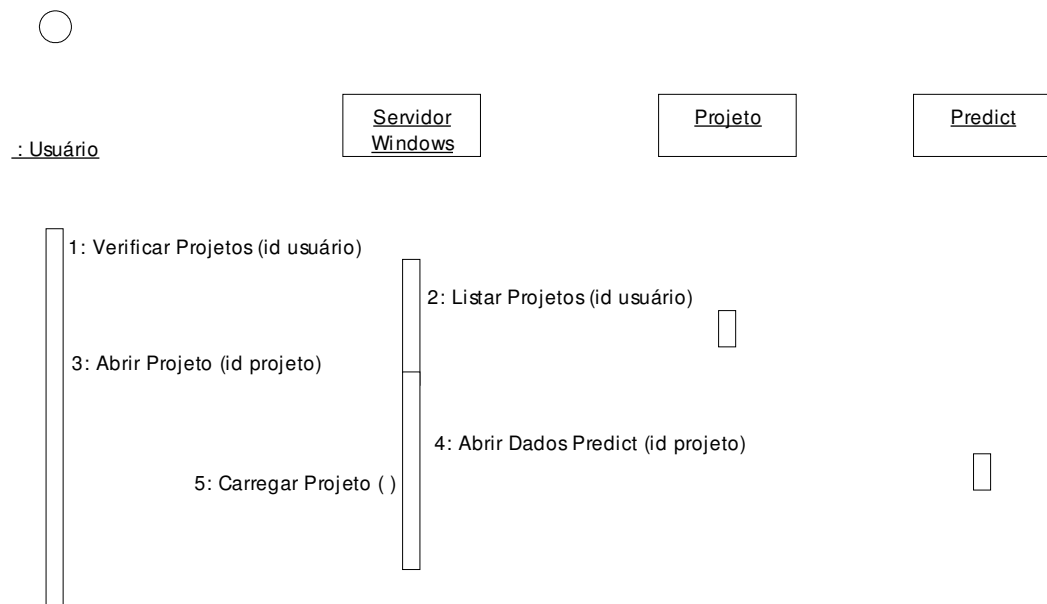


Figura 5.10 – Diagrama de seqüência de interações para abrir um projeto

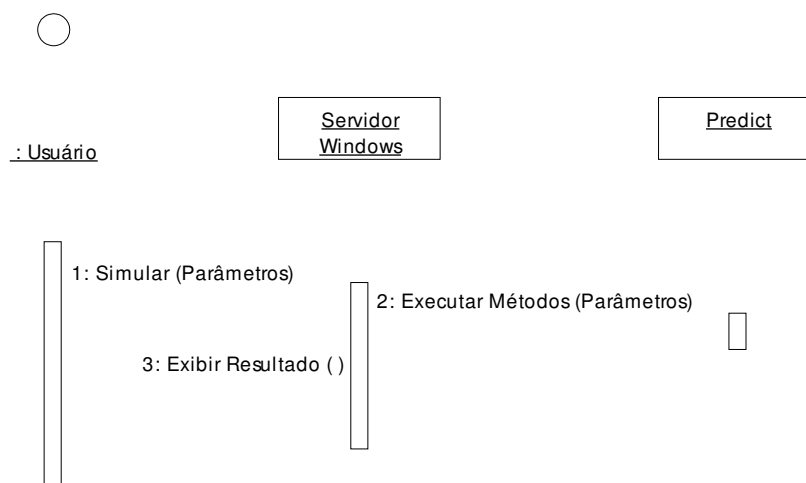


Figura 5.11 – Diagrama de seqüência de interações para a execução do módulo Predict

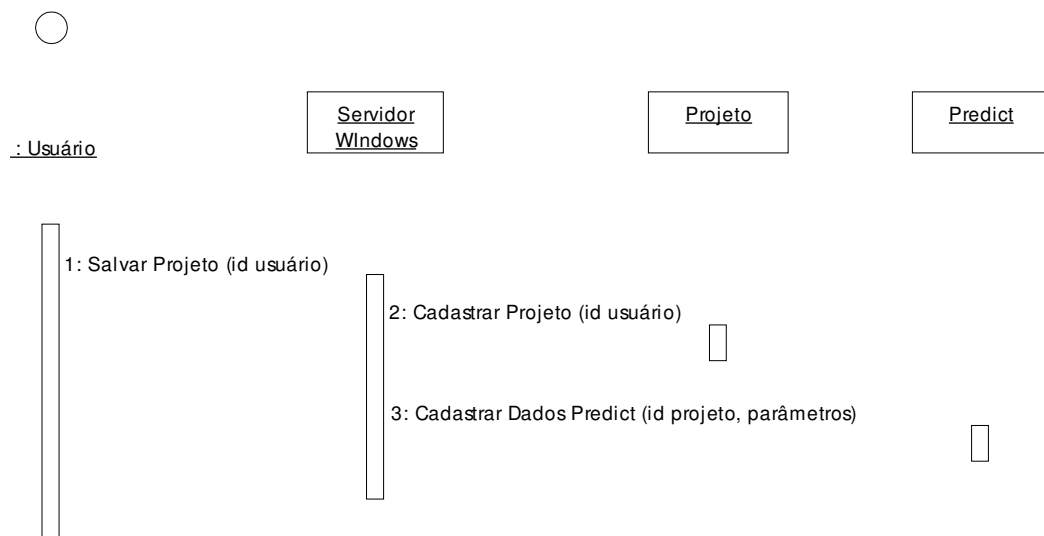


Figura 5.12 – Diagrama de seqüência de interações para criar e salvar um projeto novo

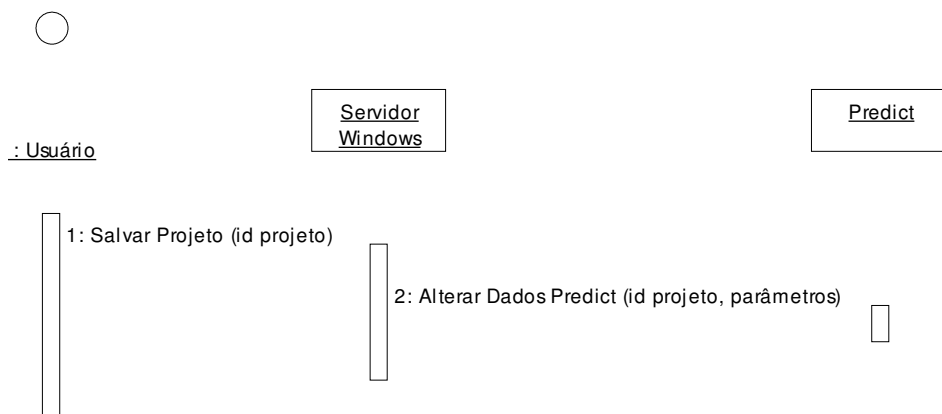


Figura 5.13 – Diagrama de seqüência de interações para alterar os dados de um projeto

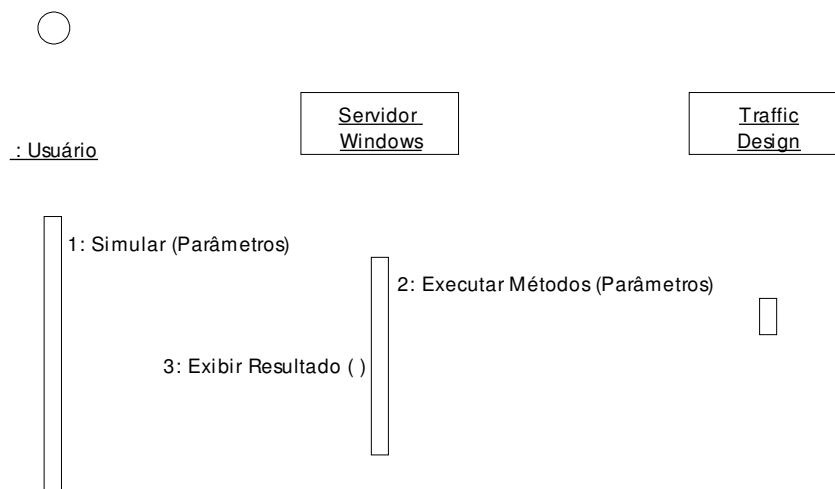


Figura 5.14 – Diagrama de sequência de interações para a execução do módulo Traffic Design

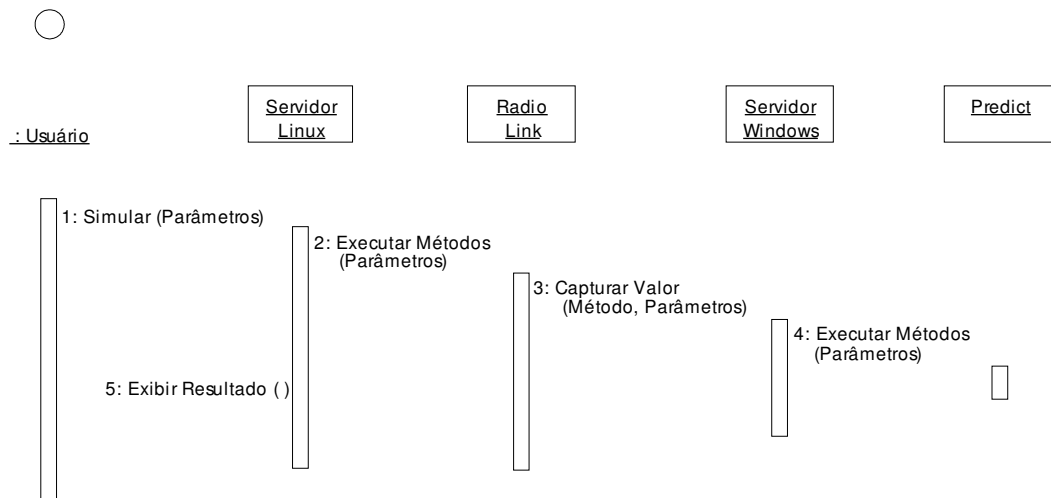


Figura 5.15 – Diagrama de sequência de interações para a execução do módulo Radio Link

5.3 Aspectos de Implementação

Como comentado anteriormente, o WebCELLP foi projetado para ser acessado pela Internet, tornando-se multiusuário e multiplataforma. Sendo assim, serão apresentadas as tecnologias envolvidas na fase de desenvolvimento do sistema, bem como suas características e requisitos de funcionamento.

5.3.1 MVC

Como o sistema será executado na *Web*, houve então a necessidade de fazer algumas adaptações, para que os três módulos do CELLP pudessem atender este requisito. Para que essas adaptações fossem realizadas, utilizou-se a UML e o padrão de projeto MVC, fazendo-se então o processo de engenharia reversa do CELLP.

O padrão de projeto MVC foi escolhido pelo fato de ele ser ideal para projetar aplicações *Web* e pelas outras inúmeras vantagens apresentadas anteriormente. Na figura 5.16 pode-se visualizar o modelo MVC do WebCELLP. Durante a fase de implementação, é importante deixar registrados os seguintes aspectos referentes ao MVC:

- Na Vista (interface gráfica do sistema) utilizou-se a interface gráfica já existente no software legado, fazendo-se algumas adaptações, como a retirada das regras de negócios da própria interface.
- No Modelo, onde estão armazenadas as regras de negócios, construíram-se dois servidores de aplicação, um implementado em Delphi 7 e outro em Kylix 3, sendo

que em ambos foram criados objetos utilizando os recursos de programação distribuída do padrão CORBA. Além dos servidores de aplicação, criou-se também um banco de dados no SGBDR InterBase 6.5, o qual armazena além das tabelas algumas regras de negócios.

- O elemento que faz o papel de Controlador é o próprio CORBA, que serve de intermediador entre o cliente e os objetos do Modelo.

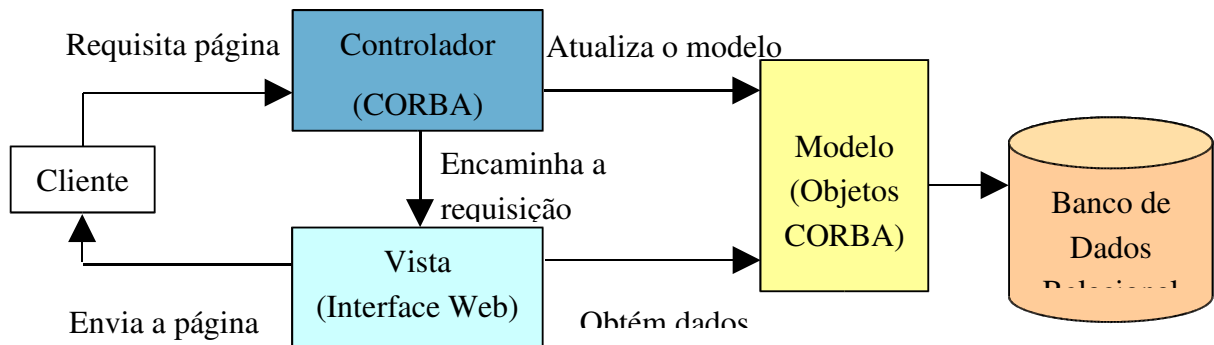


Figura 5.16 – Modelo MVC do WebCELLP

5.3.2 Linguagem de Programação

No desenvolvimento do WebCELLP foram utilizados dois ambientes de programação visual que usam a linguagem de programação *Object Pascal*, como mostra a figura 5.17: o Delphi, que foi construído para ser executado no sistema operacional Windows, e o Kylix, que foi construído para o sistema operacional Linux. Através do Delphi/Kylix pode-se criar desde aplicações genéricas, que não acessem banco de dados, até aplicações Cliente/Servidor com n-camadas que integrem várias linguagens, como é o caso desse sistema.

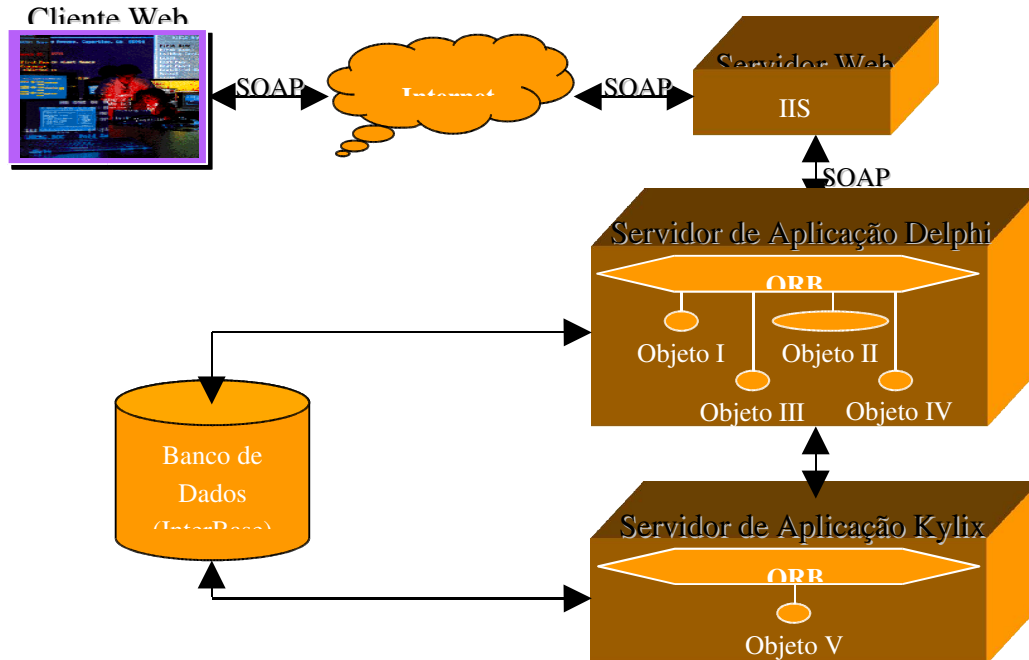


Figura 5.17 – Arquitetura do WebCELLP

Como o WebCELLP foi projetado para ser acessado pela Internet e executado em várias plataformas, sistemas operacionais e linguagens de programação. Precisou-se então utilizar os recursos de computação distribuída do Delphi/Kylix, os quais permitem a criação de aplicações com n-camadas, que é composta por um conjunto de componentes e outras tecnologias como CORBA, DCOM, SOAP e TCP/IP, permitindo assim a construção de uma variedade de aplicações com acesso à banco de dados.

Especificamente no sistema em questão, utilizou-se do CORBA para prover a comunicação entre a aplicação cliente e a aplicação servidora. Porém, para ser flexível o bastante a ponto de operar sobre o protocolo HTTP, houve a necessidade de utilizar a tecnologia de *WebServices*, baseado no protocolo SOAP (*Simple Object Access Protocol*).

5.3.3 SOAP (*Simple Object Access Protocol*)

O SOAP é um protocolo que foi elaborado para facilitar a chamada remota de funções via Internet, permitindo que dois programas se comuniquem de uma maneira tecnicamente muito semelhante ao RPC (*Remote Procedure Call*). O SOAP habilita a comunicação de objetos (ou código) de qualquer tipo, plataforma, linguagem e localização [BOX et. al., 2000].

Analisando a figura 5.18, percebe-se que quando a aplicação cliente faz uma chamada de método, o mecanismo *Parse* automaticamente cria um arquivo XML, que é transportado até a aplicação servidora. Aplicação servidora também possui um mecanismo de *Parse* que interpreta o arquivo XML, executa a função solicitada e gera um outro arquivo XML que será devolvido à aplicação cliente.

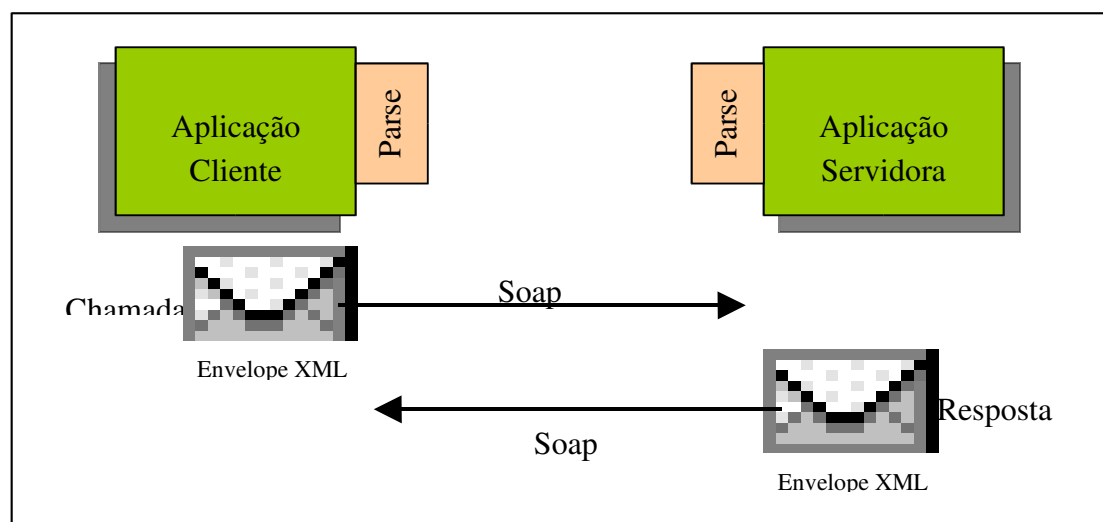


Figura 5.18 – Ação de requisição e recebimento através do protocolo SOAP

A aplicação servidora além das funcionalidades de *Parse*, possui também a capacidade de geração do WSDL (*Web Services Description Language*), que é um arquivo XML que contém todas as informações necessárias para a interação com a aplicação servidora.

No WebCELLP o SOAP é o protocolo responsável pela comunicação entre a aplicação cliente e os objetos CORBA da aplicação servidora, isto foi necessário para que o sistema pudesse operar sobre o protocolo HTTP. Sendo assim, o cliente faz a requisição através do protocolo SOAP, que delega a responsabilidade ao respectivo objeto CORBA, o qual executa os métodos que foram solicitados e repassa o resultado para o protocolo SOAP, que por sua vez o devolve para o cliente no formato XML.

5.3.4 Banco de Dados

O *InterBase* foi o SGBDR (Sistema Gerenciador de Banco de Dados Relacional) escolhido para a criação do banco de dados, pelo fato de ser um SGBDR baseado no padrão SQL ANSI-92, de alto desempenho, seguro, simples de administrar, multi-usuário, *multi-thread* e multi-plataforma [NETO, 2002].

Um outro aspecto importante é que o *InterBase Server* possui a capacidade de processamento diretamente no banco de dados, permitindo assim a implementação de algumas regras de negócios. No WebCELLP utilizou-se desse recurso através da implementação de algumas *stored procedures*, as quais tem como finalidade: listar os projetos de um determinado usuário, extrair os dados de um projeto específico e tratar das questões de integridade referencial. Com a implementação dessas *stored procedures* foi possível ganhar desempenho e confiabilidade, evitando-se I/O em disco local e de rede, pontos de alta vulnerabilidade na perda de informação (devido às falhas em disco, perdas em colisões e erros de comunicação).

A figura 5.19 mostra a compatibilidade de conexão e desenvolvimento de aplicativos no InterBase.

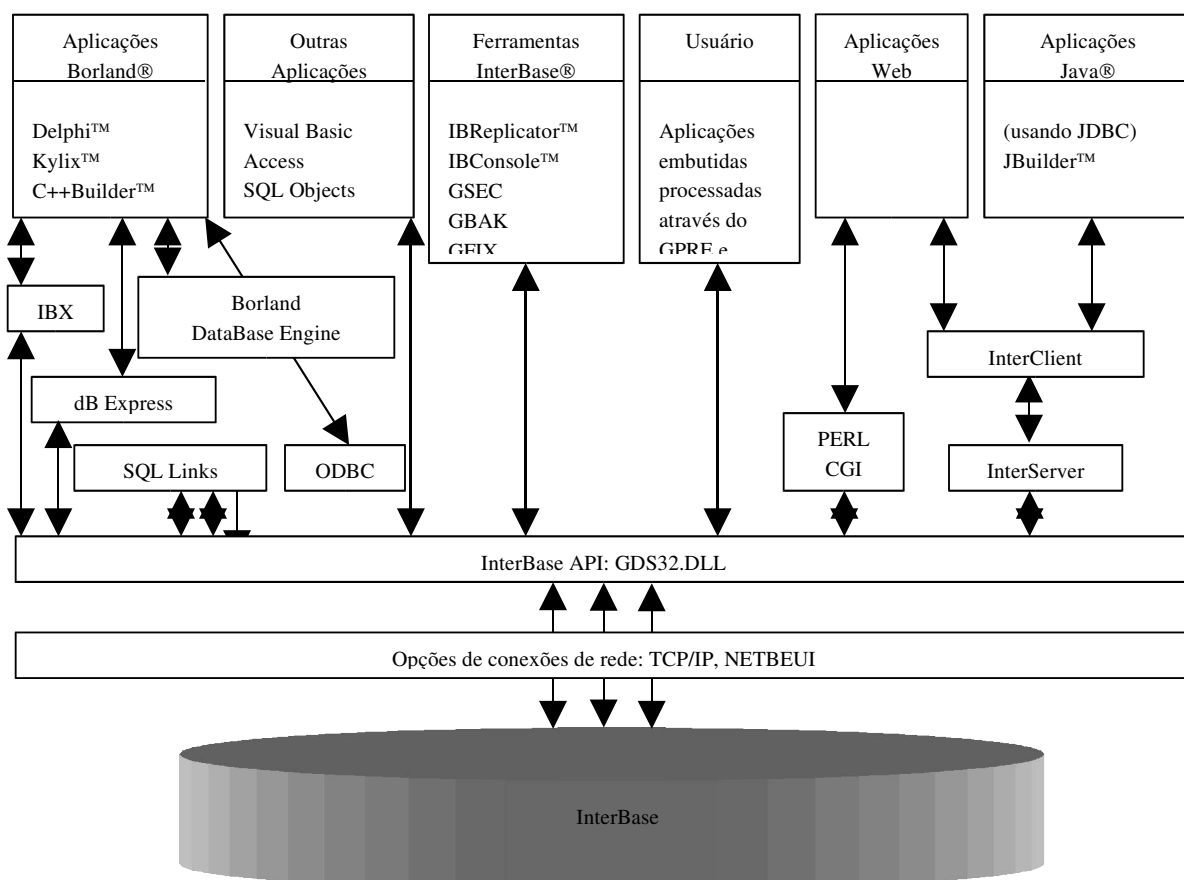


Figura 5.19 – Diagrama de conexão e compatibilidade de ferramentas de desenvolvimento do Borland InterBase Server

6 Considerações Finais e Trabalhos Futuro

Neste trabalho procurou-se demonstrar alguns pontos relevantes a aplicações distribuídas e à integração de sistemas legados, procurando viabilizar a interoperabilidade entre ambientes heterogêneos, pois a falta de integração entre sistemas legados é um problema facilmente encontrado nas diversas áreas da engenharia. Através do estudo de caso do WebCELLP, que é um sistema relacionado à área de engenharia elétrica, pôde-se por em prática esses conceitos, onde foi percebido que grandes modificações foram necessárias para que os sistemas legados (módulos do CELLP) pudessem atender aos requisitos exigidos por um ambiente distribuído, fato que foi alcançado através da fatoração dos módulos do CELLP em objetos CORBA.

Percebeu-se também que através do padrão de projeto MVC, facilmente identificaram-se as classes e instâncias participantes, seus papéis, colaborações e a distribuição de responsabilidade. A utilização desse padrão tornou a aplicação menos complexa e agilizou a fase de implementação, pois o padrão MVC é ideal para ser utilizado em aplicações com três camadas, como o WebCELLP. Outro ponto importante é que o sistema está pronto para uma possível reutilização.

A construção do WebCELLP possibilitou uma grande experiência com a plataforma de distribuição CORBA, o que permitiu o conhecimento das possibilidades, facilidades e dificuldades para se integrar sistemas legados que operam em sistemas operacionais diferentes, e foram desenvolvidos em várias linguagens de programação. Sendo que tal experiência foi centrada principalmente nos aspectos referentes à interação entre CORBA, Delphi e Kylix.

Em relação à base de dados remota, pode-se afirmar que os acessos a essa base poderiam ser mais rápidos, caso os acessos fossem feitos diretamente ao SGBD. Porém, por questão de segurança, os acessos foram feitos através do ORB a um servidor remoto, que têm implementado todos os métodos para acessar a base de dados do sistema. Outra vantagem, além da segurança, é a reutilização e a padronização dos códigos de acesso à base de dados, facilitando as manutenções futuras.

Outro aspecto relevante neste trabalho é o fato de que a maioria das aplicações *Web* baseadas em dados, sofrem com a baixa velocidade de comunicação. Como a natureza da *Web* é *stateless*, ou seja, cada requisição do cliente ao servidor precisa conter todas as informações de estado, que são necessárias para satisfazer tal requisição. Portanto, o servidor precisa executar uma nova consulta à base de dados, ao invés de reutilizar os conjuntos de resultados previamente estabelecidos.

No WebCELLP tal fato não ocorre porque, utilizando CORBA, pode-se fazer com que as aplicações *Web* melhorem o desempenho e o tempo de resposta ao cliente, reduzindo-se o número de requisições que o cliente faz ao servidor e a quantidade de trabalho que o servidor precisa executar. Para melhorar o desempenho, o cliente faz a requisição através do protocolo SOAP, que delega a responsabilidade ao respectivo objeto CORBA, o qual obtém os dados do SGBD e os repassa para o protocolo SOAP, que em fim, devolve os dados para o cliente no formato XML. Essa divisão de responsabilidade permite que o cliente visualize, pague, busque e ordene os dados, localmente, atualizando a tela conforme necessário sem fazer novas conexões ao servidor.

Para realizar todas as modificações feitas no CELLP, muitas barreiras precisarão ser vencidas, e entre elas, merecem maiores destaques as seguintes:

- Fatoração dos módulos do CELLP em objetos distribuídos: os módulos do CELLP operavam isoladamente, precisando-se então fatorá-los. Isto dificultou fazer a integração desses módulos e deixá-los prontos para serem reutilizados;
- Configuração do sistema operacional: para possibilitar a comunicação do CORBA com o Linux foi preciso configurar algumas variáveis de ambiente do Linux. Portanto, num projeto como este é necessária a participação de profissionais que dominem tanto a área de sistemas operacionais quanto a de redes de computadores;
- Configuração do Kylix: foi preciso também incluir alguns arquivos do Kylix no Linux, para que fosse possível a comunicação do CORBA com o servidor de aplicação Kylix.

Outras questões que fogem do escopo desse trabalho, e ficam como proposta para trabalho futuros, são:

- Integrar ao WebCELLP outros módulos já desenvolvidos, e testar a integração com outras linguagens de programação e sistemas operacionais;
- Desenvolver novos softwares que complementarão o WebCELLP, sempre visando à integração, reutilização e os aspectos de heterogeneidade referentes a sistemas distribuídos;
- Difundir em outras IES a solução de integração proposta neste trabalho, a fim de compor parcerias que visam à integração de novos módulos, fazendo com que o WebCELLP seja cada vez mais completo;
- Acompanhar o processo de utilização do sistema, a fim de verificar a opinião dos usuários, no que diz respeito a critérios como: disponibilidade, facilidade de uso, interface adequada e intuitiva, etc;
- Criar um arquivo de *log* da sessão para permitir o acompanhamento das atividades desenvolvidas;

- Adequar o site de forma que o mesmo atenda os requisitos fundamentais do ensino a distância. Sendo assim, a disciplina de Sistemas Móveis Celulares que hoje é totalmente presencial, poderá ser ministrada de forma semipresencial, inserindo o curso de Engenharia Elétrica da UFPA no âmbito da EAD.

Anexo A – Janelas do WebCELLP

Para que o usuário possa acessar o WebCELLP, ele deverá primeiramente visitar o site que é mostrado na figura A.1, para então executar os seus módulos.

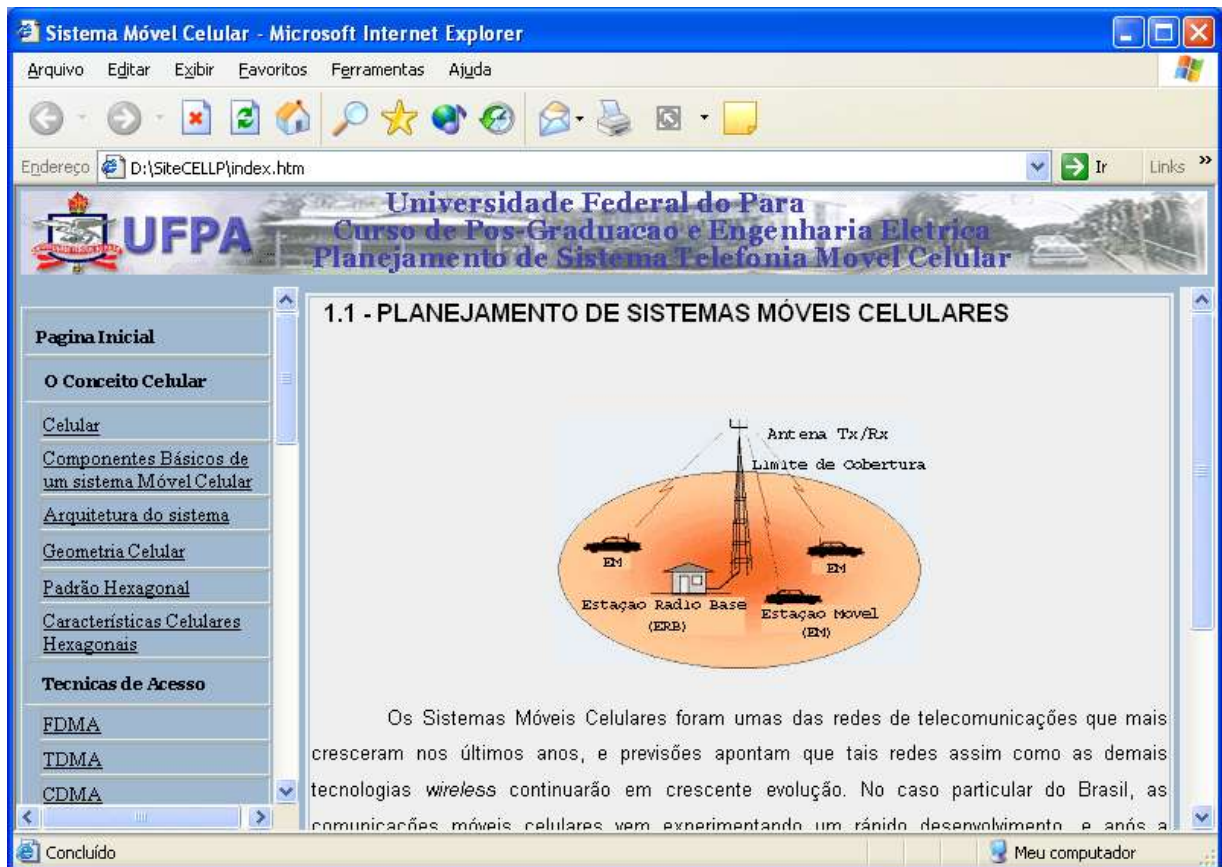


Figura A.1 – Site que dá acesso ao WebCELLP

Após ter acessado o site, o usuário poderá selecionar a opção “Executar o WebCELLP”, a qual lhe exibirá a janela de login, como mostra a figura A.2. Ao efetuar o login, ele terá a possibilidade de executar um dos módulos do sistema ou acessar a janela de gerenciamento de projetos, como mostra a figura A.3.



Figura A.2 – Janela de login



Figura A.3 – Janela principal do WebCELLP

Caso o usuário acesse a janela de Gerenciamento de Projetos, lhe será exibida a figura A.4, na qual ele terá a opção de adicionar ou remover um projeto.

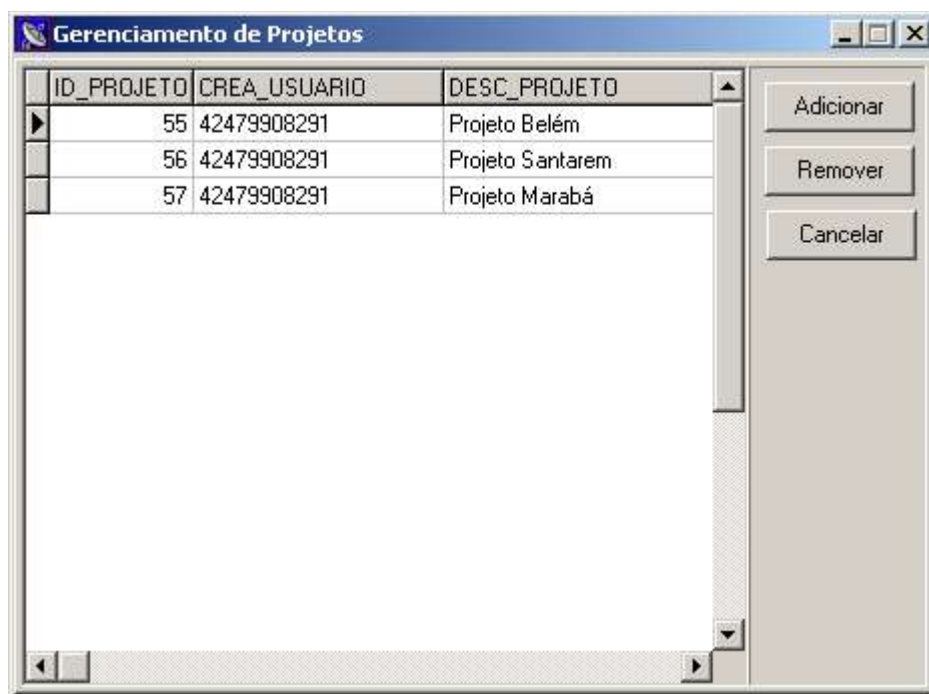


Figura A.4 – Janela de Gerenciamento de Projetos

A seguir é apresentada uma visão geral do funcionamento dos três módulos que compõe o sistema, porém poderão ser encontradas maiores informações em [CAVALCANTE, 2002].

Caso o usuário opte por executar o módulo Predict, lhe será exibida a figura A.5. O *software* Predict foi desenvolvido principalmente para realizar estudos estatísticos comparativos dos modelos de predição. Tal comparação visa avaliar o desempenho desses

modelos de predição frente aos dados de uma campanha de medições realizada em uma determinada região.

The screenshot shows the 'Predict' software window with four tabs: 'Parâmetros', 'Base de Dados', 'Análise Gráfica', and 'Análise Estatística'. The 'Parâmetros' tab is active, displaying the following sections:

- Parâmetros do Sistema:** A list of input fields for system parameters: Potência Transmitida (dBm) [30], Frequência de Operação (MHz) [900], Ganho da Antena Transmissora (dBi) [2,14], Ganho da Antena Receptora (dBi) [2,14], Altura Efetiva da Antena da ERB (m) [50], Altura Efetiva da Antena do TM (m) [3], Distância entre o TM e os prédios na rua (m) [10], and Ângulo da rua em relação ao enlace (graus) [45].
- Logotipo:** A circular logo for 'Laboratório de Eletromagnetismo Aplicado LEA UFPA'.
- Linha de Visada:** Radio buttons for 'Sim' and 'Não' (selected).
- Modelos de Predição:** A list of checkboxes for prediction models: 'Modelo do Projetista' (checked), 'Espaço Livre' (checked), 'Okumura-Hata' (checked), 'Okumura-Hata Modificado' (checked), 'Walfisch-Ikegami' (checked), 'Walfisch-Bertoni' (checked), 'Maciel-Bertoni-Xia' (checked), and 'Ibrahim-Parsons' (checked). A button 'Abrir dados do Predict' is at the bottom.
- Parâmetros Urbanos:** Input fields for urban parameters: 'Largura Média das Ruas (m)' [20], 'Altura Média dos Prédios (m)' [30], '% de Prédios sobre o Terreno (%)' [30], and 'Distância média entre Prédios (m)' [30]. Below these are two groups of radio buttons: 'Tamanho da Cidade' (Grande selected, Média, Pequena) and 'Área' (Urbano Denso selected, Urbano, Sub-Urbano, Rural).

Figura A.5 – Janela inicial do Predict

A janela desse programa é dividida em diversas abas que são descritas a seguir [CAVALCANTE, 2002]:

- **Parâmetros:** nela o usuário deverá fornecer os dados referentes aos parâmetros do sistema (potência transmitida, frequência, etc) que foram utilizados na campanha de medições de referência. O usuário também deve selecionar nesta aba, os modelos que serão colocados sob avaliação assim como os parâmetros urbanos necessários para a aplicação dos mesmos;
- **Base de Dados:** nela o usuário deverá fornecer a base de dados das campanhas de medições realizadas na região de interesse, ele terá também a opção de editar os dados e depois salvá-los em um arquivo;
- **Análise Gráfica:** nela são gerados gráficos da potência recebida (dBm) *versus* distância (m), baseando-se nos modelos colocados sob avaliação e nos dados da campanha de medições;
- **Análise Estatística:** nela são apresentados os desempenhos dos modelos testados frente às medidas. O desempenho é especificado sobre três aspectos: Desvio Absoluto (dB), Erro Médio Absoluto (dB) e Erro RMS (dB). O usuário ainda tem a opção de gerar um relatório apresentando os resultados obtidos e de salvar esses resultados em um determinado projeto.

Caso o usuário deseje abrir um projeto existente, ele deverá selecionar o botão “Abrir dados do Predict”, para então ter acesso à janela “Lista de Projetos”, a qual é mostrada na figura A.6. Esta janela é a mesma para a abertura de projetos nos módulos Radio Link e Traffic Design.



Figura A.6 – Janela Lista de Projetos

Caso o usuário opte por executar o módulo Radio Link, lhe será exibida a figura A.7. O *software* Radio Link foi desenvolvido para realizar o balanço de potência do sistema, determinando o raio máximo de cobertura de um determinado modelo para os dois enlaces básicos (descida e subida).

Esse módulo possui duas abas, uma denominada “Dados de Entrada”, onde o usuário fornecerá os parâmetros do sistema e escolherá o modelo que deseja realizar o estudo, e outra denominada “Resultados”, onde o usuário poderá verificar os resultados dos cálculos realizados, gerar relatório contendo os resultados obtidos e salvar esses resultados em um determinado projeto.

The screenshot shows the 'Radio Link' software window with two tabs: 'Dados de Entrada' (selected) and 'Resultados'. The interface is divided into several sections for parameter input:

- Parâmetros da ERB:**
 - Potência Transmiteda: 36 [dBm]
 - Ganho da Antena: 15 [dBi]
 - Tipo da Antena: Omni (dropdown)
 - Altura Efetiva da Antena: 50 [m]
 - Sensibilidade do Receptor: -113 [dBm]
 - Ganho TMA: 0 [dB]
 - Perdas de Alimentação: 0 [dB]
 - Perdas Adicionais Tx: 0 [dB]
- Parâmetros do TM:**
 - Potência Transmiteda: 28 [dBm]
 - Ganho da Antena: 0 [dBi]
 - Altura Efetiva da Antena: 3 [m]
 - Sensibilidade do Receptor: -103 [dBm]
- Parâmetros dos Modelos de Predição:**
 - Modelo de Predição: Modelo do Projetista (dropdown)
 - Largura Média das Ruas: 20 [MHz]
 - Altura Média dos Prédios: 30 [m]
 - % de Prédios sobre o Terreno: 30 [Graus]
 - Distância média entre Prédios: 30 [dB]
 - Distância entre o TM e os prédios: 10 [m]
 - Ângulo da rua x enlace: 45 [Graus]
- Parâmetros do Sistema:**
 - Frequência de Operação: 900 [MHz]
 - Margem de Alta Qualidade: 3,5 [dB]
 - Perdas de Penetração: 4 [dB]
 - Margem para Interferências: 3,5 [dB]
 - Margem para Fading: 5,5 [dB]
 - % de Área Coberta: 90 [%]
 - Desvio Padrão: 8 [dB]
 - Constante de Propagação: 3,5
- Other Options:**
 - Linha de Visada:** ☐ Sim, ☒ Não
 - Tamanho da Cidade:** ☒ Grande, ☐ Média, ☐ Pequena
 - Área:** ☒ Urbano Denso, ☐ Urbano, ☐ Sub-Urbano, ☐ Rural

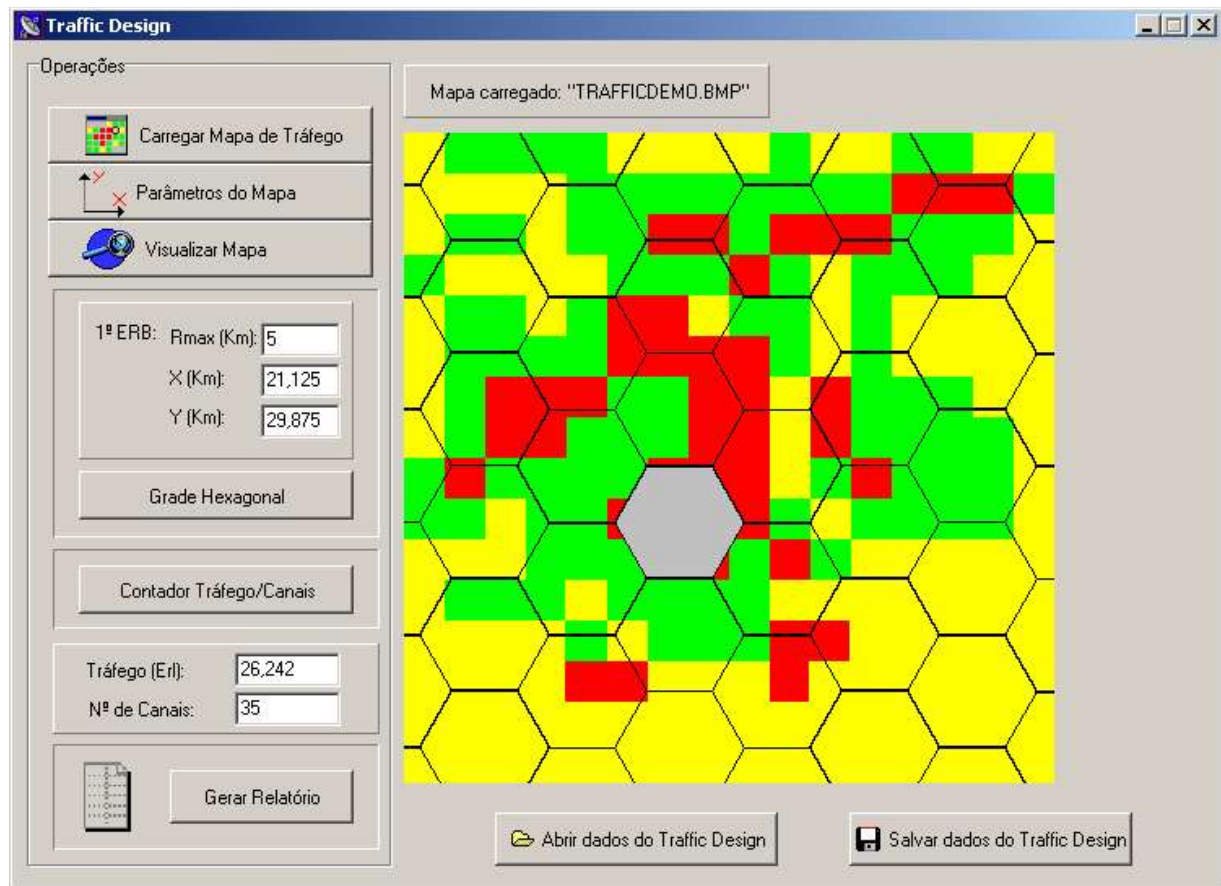
Buttons include 'Abrir dados do Radio Link' and 'Balanço de Potência (Link Budget)'.

Figura A.7 – Janela do módulo Radio Link

Caso o usuário opte por executar o módulo Traffic Design, lhe será exibida a figura A.8. Esse módulo realiza o projeto de tráfego baseado em um mapa que demonstra o tráfego previsto para uma determinada região. O mapa é dividido em quadriculas que representam o tráfego estimado na área delimitada pela mesma.

A utilização desse programa consiste em o usuário definir alguns parâmetros do mapa, para poder então especificar o raio das células que formarão a grade hexagonal e especificar também a localização da 1ª ERB através das coordenadas “X” e “Y”. Após definir todos os parâmetros necessários, o usuário poderá solicitar a realização da contagem de tráfego na área

da célula correspondente à 1ª ERB, afim de que seja determinado o número de canais necessários para atender tal tráfego. O usuário ainda tem a possibilidade de gerar um relatório do resultado dessa contagem e de salvar esse resultado em um determinado projeto.



A.8 – Janela do módulo Traffic Design

7 Referências Bibliográficas

ARAÚJO, Kleitor Franklint Correa de. **Delphi 5 com Banco de Dados para Internet**. São Paulo: Érica, 2000.

BELHOT, R. V. **Reflexões e propostas sobre o “ensinar engenharia” para o século XXI**. São Carlos. 1997. Tese (Livre Docência) - Escola de Engenharia de São Carlos, Universidade de São Paulo. 1997. 113p.

BORLAND SOFTWARE CORPORATION. **Delphi 3: Desenvolvimento Client/Server**. [s.l.; s.n.], 1997.

BOX, Don et. al. **Simple Object Access Protocol (SOAP) 1.1**. W3C, 2000. Disponível em: <<http://www.w3.org/TR/SOAP>>. Acessado em: 30/08/2003.

BROWN S. et. al.. **Professional JSP 2nd Edition**. [s.l.]: Wrox, 2001.

CAVALCANTE, André M. et. al. Software Educacional para Dimensionamento de Sistemas Móveis Celulares. Trabalho apresentado no X Simpósio Brasileiro de Microondas e Optoeletrônica - SBMO. Recife, 2002.

CHAN, Mark C.; GRIFFITH, Steven W.; IASI, Antony F. **Java 1001 dicas de programação**. São Paulo: MAKRON Books, 1998.

CORNELL, Gary; HORSTMAN, Cay S. **Core Java 2 Volume II – Recursos Avançados**.

São Paulo: Makon Books, 2001.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Distributed Systems: Concepts and Design**. England: Pearson Education, 2001.

CUMMINS, Fred A. **Integração de Sistemas: arquiteturas para integração de sistemas e aplicações corporativas**. Rio de Janeiro: Campus, 2002.

DAMSKI, José Carlos B. **Internet – guia do usuário brasileiro**. São Paulo: MAKRON Books, 1996.

GAMMA, Erich et. al. **Padrões de Projetos: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2000.

GOMES, Ana. Claudia da Silva. **Ferramenta Baseada em Web Aplicada à Gerência de Sessão de Experimentação Remota e Virtual**. Dissertação de Mestrado, 2002. Programa de Pós-Graduação em Engenharia Elétrica. Universidade Federal do Pará, 2002.

KURI, N. P. **Ciclo de Aprendizagem: uma estratégia para o planejamento do ensino aprendizagem**. Dissertação de Mestrado, 1998. Pós-graduação em Engenharia de Produção da Escola de Engenharia de São Carlos. Universidade de São Paulo, 1998.

MAIA, Carmem. Educação a distância no Brasil na Era da Internet. in: — (Coord.). **ead.br**. São Paulo: Anhembi Morumbi, 2000.

NETO, Álvaro Pereira. **InterBase: técnicas avançadas: versões open source 6.X: soluções para desenvolvedores e administradores de banco de dados**. São Paulo: Érica, 2002.

NUNES, I.V. Noções de Educação a Distância. Revista **Educação a Distância** no. 4/5, pp. 7-25. 1994.

OMG, Object Management Group. **Common Object Request Broker Architecture: Core Specification Version 3.0**, OMG Document, December 2002. Disponível em: <http://www.omg.org/technology/documents/corb_spec_catalog.htm>. Acessado em: 09/02/2003.

PRETI, Oreste (Org.). **Educação a Distância: inícios e indícios de um percurso**. Cuiabá: UFMT, 1996.

RICCIONI, Paulo Roberto. **Introdução a Objetos Distribuídos com CORBA**. Florianópolis: Visual Books, 2000.

SUN MICROSYSTEMS. **RMI Architecture and Functional Specification**. [s.d.]. Disponível em: <<http://java.sun.com/j2se/1.4.1/docs/guide/rmi/spec/rmiTOC.html>>. Acessado em: 10/04/2003.

TAJRA, S. F. **Informática na Educação: Novas Ferramentas Pedagógicas para o Professor da Atualidade**. São Paulo, Érica. 2000.