Pós-Graduação em Ciência da Computação

“ Módulo de Integração de Bancos de Dados em um Ambiente de Ensino/Aprendizagem Baseado na *Web* ”

Por

Fábio de Jesus Lima Gomes

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, FEVEREIRO/2003

Universidade Federal de Pernambuco
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FÁBIO DE JESUS LIMA GOMES

" Módulo de Integração de Bancos de Dados em um Ambiente de Ensino/Aprendizagem Baseado na Web "

Este trabalho foi apresentado à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: FERNANDO DA FONSECA DE SOUZA

RECIFE, FEVEREIRO/2003

Gomes, Fábio de Jesus Lima

Módulo de integração de bancos de dados em um ambiente de ensino/aprendizagem baseado na Web / Fábio de Jesus Lima Gomes. – Recife : O Autor, 2003.

116 p.: il., fig.

Dissertação (mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, 2003.

Inclui bibliografia.

1. Banco de dados – Integração. 2. Internet – Ensino a distância. 3. Linguagem de programação – Uso. 4. Objetos distribuídos (Linguagem de programação). I. Título.

004.657	**CDU (2.ed.)**	**UFPE**
005.74	**CDD (21.ed.)**	**BC2003-072**

Dedico este trabalho a
minha família, em especial a meus pais.

Agradecimentos

A Deus em primeiro lugar, que me encheu de paz e saúde no andamento deste trabalho.

Aos meus queridos pais, Gomes Neto e Lídia que me deram uma formação pessoal sólida e responsável para que pudesse atingir meus objetivos.

A toda minha família.

A Diretora Geral do Centro Federal de Educação Tecnológica do Piauí (CEFET-PI), Rita Martins de Cássia, que viabilizou os meios necessários para a realização deste estudo.

A Professora Teresa Bernarda Ludermir, coordenadora do Mestrado Interinstitucional do Centro de Informática da UFPE, que acreditou nesse trabalho.

Ao professor orientador Fernando Fonseca, meu agradecimento especial, pois, apesar da distância, soube me conduzir com maestria, firmeza e dedicação.

Aos professores do Centro de Informática da UFPE, Ana Carolina Salgado, Décio Fonseca, Djamel Sadok e Valéria Cesário Times.

Aos componentes da banca examinadora, por aceitarem o convite a participarem da banca de defesa com disposição e interesse.

Aos colegas mestrandos pela troca de conhecimentos e pela solidariedade nos momentos em que estivemos juntos. Em particular, a Ricardo Ramos e Flávio Ferry.

Aos colegas professores de Informática do CEFET-PI, que contribuíram de forma indireta para a conclusão deste trabalho.

Ao colega Adalberto Cajueiro Farias, mestrando do CIn/UFPE, que nos momentos de dúvida, contribuiu com seu conhecimento e experiência para o sucesso deste trabalho.

Resumo

A utilização do computador na educação vem demonstrando ser um grande auxílio no processo ensino-aprendizagem. Uma das formas desta utilização é através de software educacional, um software para auxiliar estudantes no aprendizado de um determinado conteúdo.

Com a grande disseminação do uso da *World Wide Web* (WWW), e com o seu poder de alcançar pessoas nos mais diversos lugares do mundo, a WWW tornou-se um grande recurso para distribuição da informação, e para vários outros fins. Dentro destas perspectivas, pode-se utilizar a WWW com fins educacionais, de diversas formas, sendo uma delas a utilização de software educacional.

A disponibilização e integração de dados surgem como tendência de serviços oferecidos pela WWW. O objetivo de um sistema de integração de dados é oferecer aos usuários uma interface uniforme de acesso a diferentes fontes de dados, de forma que os usuários definam consultas especificando o que desejam saber e o sistema determine onde a informação pode ser encontrada e, em seguida, apresente as respostas para as consultas do usuário.

O módulo de integração implementado possui características de distribuição, autonomia e integração através dos conceitos de bancos de dados federados, que consiste na integração de vários bancos de dados cooperantes e autônomos, mas que participam de uma federação possibilitando o compartilhamento parcial e controlado dos seus dados.

O citado módulo tem por objetivo atender aos requisitos de cooperação e colaboração entre os estudantes e o professor em um ambiente de ensino-aprendizagem baseado na Web, fazendo que vários bancos de dados integrados sejam vistos por usuários globais como um único banco de dados, provendo as transparências desejadas de bancos de dados distribuídos.

Abstract

The usage of the computer in the education has been showing to be a great aid in the teaching-learning process. One of the forms of this usage is the educational software, a software to help students in the learning of a determinate contents.

Because of the great use dissemination of the World Wide Web (WWW), and its power to reach people in the most different places of the world, the WWW became a great resource for distribution of information and many other purposes. Inside those perspectives, it is able to use the WWW with educational purposes, in several forms, being one of those the utilization of a educational software.

The disponibilization and integration of data appear as the tendency of services offered by WWW. The objective of a integration data system is to offer users with a uniform interface for accessing to differents data source, so the users define queries specifying what they want to know and the system determine where the information can be found and, afterwards, shows the reply to the user's queries.

The implemented integration module has characteristics of distribution, autonomy and integration throught the concepts of federated databases, which consist of the integration of several cooperative and autonomous database, but participating in a federation, permiting the partial sharing and control of its data.

The quoted module has the objective of attend the requires of cooperation and collaboration between the students and the professor in a teaching-learning environment based on the web, making the various integrated databases to being viewed by global users as a unique database, suplying the wanted transparencies of distributed databases.

Sumário

Sumário (continuação)

Lista de Figuras

Lista de Figuras (continuação)

Capítulo 1

Introdução

Este capítulo apresenta uma introdução sobre os temas apresentados, a motivação para este trabalho, seus objetivos e a organização desta dissertação.

Capítulo 1

Introdução

A utilização do computador na educação vem demonstrando ser um grande auxílio no processo ensino-aprendizagem. Uma das formas desta utilização é através de software educacional, um software para auxiliar estudantes no aprendizado de um determinado conteúdo.

Um software educacional também possui o objetivo de auxiliar o professor no processo de ensino-aprendizagem [93], fazendo com que o mesmo tenha a seu dispor valiosos recursos para ajudá-lo nas tarefas didáticas junto a seus alunos.

Existem diversos tipos de software educacional, sendo que um dos mais importantes é o Sistema Tutor Inteligente (STI), que segundo Woolf [120] é um sistema capaz de modelar ensino, aprendizagem, comunicação e domínio de conhecimento. Um STI é um software capaz de tutorar uma pessoa em um determinado domínio, sabendo o que ensinar, como ensinar, além de capturar informações relevantes sobre o aprendiz que está sendo tutorado, proporcionando um aprendizado individualizado.

Os sistemas tutores inteligentes utilizam técnicas de inteligência artificial e teorias pedagógicas para conduzir o estudante, proporcionando um ótimo ambiente de aprendizagem.

Com a grande disseminação do uso da *World Wide Web* (WWW) [122], e com o seu poder de alcançar pessoas nos mais diversos lugares do mundo, a WWW tornou-se um grande recurso para distribuição de informação, e para vários outros fins. Dentro destas perspectivas, pode-se utilizar a WWW com fins educacionais [25, 94, 95], de diversas formas, sendo uma delas a utilização de software educacional.

Atualmente, o paradigma do aprendizado moderno está sendo liderado pelas abordagens “aprender fazendo” e “aprender cooperando”, como proposto por Piaget [82] e Vigotsky [100], respectivamente. Ambientes computacionais que dão suporte a estas abordagens estão difíceis de se encontrar atualmente. Não é comum, portanto, encontrar um ambiente computadorizado que suporte estas novas abordagens com uma metodologia flexível e uma interface amigável.

Quando um trabalho é feito de maneira cooperativa, o resultado aparenta ser bem melhor que quando é obtido de uma abordagem individual. Desenvolvendo atividades cooperativas, os estudantes trocam experiências e têm contato com outras possibilidades para resolver o mesmo problema.

Um outro tipo de software educacional é o CSCL (*Computer Supported Collaborative Learning* - Aprendizado Colaborativo Assistido por Computador [22] que se constitui da utilização de computadores em rede e de um aplicativo apropriadamente escolhido ou projetado dentro de um contexto de instrução que apóie os processos de aprendizado em grupo.

O aprendizado colaborativo é uma atividade social [13] que envolve alunos compartilhando conhecimentos e adquirindo outros novos, processo que tem se nomeado como construção social do conhecimento [58].

O aprendizado colaborativo está vinculado à teoria do construtivismo social que assume que o aprendizado se dá entre pessoas e não entre pessoas e coisas [92] e se fundamenta na idéia de que todos juntos somos mais inteligentes que cada um por si só.

No esquema do aprendizado colaborativo o estudante deixa de ser um agente passivo que recebe a informação do professor e passa a ser um agente ativo na construção do conhecimento.

Tinto, Goodsell & Russo [107] apontam que o aprendizado colaborativo promove a discussão interativa entre os estudantes e o professor, favorecendo o desenvolvimento do pensamento crítico. Os estudantes se sentem dentro dos grupos de discussão mais à vontade para expressar seus pensamentos. O trabalho em equipe estimula uma interação verbal maior. Neste tipo de aprendizado, os estudantes se envolvem mais tanto social quanto academicamente e reconhecem que a qualidade do aprendizado é melhor que com os métodos tradicionais porque eles têm realmente que aprender e não apenas memorizar.

A avaliação não pode ser esquecida no processo de ensino colaborativo apoiado por computador porque ela permite obter um retorno sobre o aprendizado alcançado pelo aluno e, conseqüentemente, deverá orientar os educadores nas novas estratégias a serem seguidas para garantir um aprendizado significativo.

1.1. Motivação

A WWW se apresenta como uma tecnologia capaz de atender às expectativas dos pesquisadores da área de ensino/aprendizagem a distância, proporcionando soluções para o problema do oferecimento de educação e treinamento em larga escala, a custos mais acessíveis que os atuais, permitindo a publicação de material didático, aplicação de tutoriais, aplicação de provas e testes, comunicação com os estudantes e apresentação de aulas a distância (conferência multimídia).

As pesquisas apontam para o uso de recursos propiciados pela Inteligência Artificial (IA) a fim de prover aos sistemas computacionais de ensino, capacidade de adaptação ao contexto e de personalização do ambiente de acordo com as características do aluno, além de permitir um alto grau de interatividade entre o ambiente e os usuários e um controle de sessões de ensino em ambientes multiusuários. A introdução das técnicas de IA nestes ambientes tem a finalidade de propiciar mecanismos de modelagem do processo de ensino bem como do estado cognitivo do estudante [68].

Os avanços mais recentes no campo dos ambientes de aprendizagem inteligentes têm proposto o uso de arquiteturas baseadas em sociedades de multiagentes [12].

Os princípios dos sistemas multiagentes têm mostrado um potencial bastante adequado ao desenvolvimento de sistemas de ensino, devido ao fato da própria natureza do problema de ensino-aprendizagem ser mais facilmente resolvido de forma cooperativa [43, 73, 101]. Além disso, ambientes de ensino baseados em arquiteturas multiagentes possibilitam suportar o desenvolvimento de sistemas de forma mais robusta, mais rápida e com menores custos, tornando-os mais atrativos, do ponto de vista de seu aproveitamento real, não ficando restrito a protótipos.

Um sistema tutor inteligente usado na WWW oferece como grande vantagem a possibilidade dele poder ser utilizado por diversos usuários, em diversos lugares.

O custo financeiro e o tempo para desenvolvimento de sistemas tutores inteligentes são grandes, o que torna esses sistemas pouco utilizados. Uma das

maneiras possíveis de diminuir o custo deles é através da utilização de agentes inteligentes, o que os torna modulares e extensíveis, favorecendo o reuso do software, diminuindo o seu custo e o tempo de desenvolvimento.

A arquitetura para um sistema tutor inteligente baseado na *Web*, proposta por Souza e Campos [103] pode ser utilizada para construir outros tutores nas mais diversas áreas. Esta arquitetura, em conjunto com a teoria construtivista de aprendizado, utilizando a metáfora da sala de aula (ou outra mais adequada de acordo com o domínio abordado), oferece um ambiente valioso para auxílio em processos de aprendizagem.

A disponibilização e integração de bancos de dados surgem como tendência de serviços oferecidos pela Web. O objetivo de um sistema de integração de dados é oferecer aos usuários uma interface uniforme de acesso a diferentes fontes de dados, de forma que os usuários definam consultas especificando o que desejam saber e o sistema determine onde a informação pode ser encontrada e, em seguida, apresente as respostas para as consultas do usuário.

O módulo de integração de banco de dados em um STI, inicialmente proposto por Souza e Campos [103], deve possuir características de distribuição, autonomia e integração através dos conceitos de bancos de dados federados [15, 98], que consiste na integração de vários bancos de dados cooperantes e autônomos, mas que participam de uma federação possibilitando o compartilhamento parcial e controlado dos seus dados.

O supracitado módulo tem por objetivo atender aos requisitos de cooperação e colaboração entre os estudantes e o professor em um ambiente de ensino-aprendizagem baseado na Web, fazendo com que vários bancos de dados integrados sejam vistos por usuários globais como um único banco de dados provendo as transparências desejadas de bancos de dados distribuídos [81], tais como, transparência de localização, que consiste no acesso a um recurso remoto, neste caso, um banco de dados, sem precisar informar o seu endereço e sem conhecer a sua localização física.

O módulo de integração implementa um conjunto de consultas pré-definidas, que permite o acesso aos bancos de dados de outros estudantes pertencentes ao curso. Sua função é recuperar dados das fontes de dados escolhidas submetendo os pedidos e realizar o processamento necessário para integrar dados de diversas fontes.

Geralmente, integração requer junção, quando mais de uma fonte de dados está envolvida no processo.

1.2. Objetivos

Este trabalho possui, portanto, três objetivos principais:

1. estudar e documentar o estado da arte em integração de bancos de dados pela *Web*, com a utilização de objetos distribuídos;
2. desenvolver uma arquitetura detalhada para integração de bancos de dados em um sistema tutor inteligente de ensino na *Web* baseada na proposta de Souza e Campos [103];
3. apresentar a definição formal, a especificação e a implementação do módulo de integração de bancos de dados em um ambiente de ensino-aprendizagem pela *Web*, utilizando a linguagem de programação Java [51, 56].

1.3. Estrutura da Dissertação

Os demais capítulos deste trabalho estão organizados como segue.

Capítulo 2 – Sistemas Tutores Inteligentes

Este capítulo aborda STIs, focalizando um breve histórico, sua arquitetura básica e aspectos concernentes ao desenvolvimento destes sistemas. Finalmente, será apresentado um estudo de caso, o WILE (*Web based Intelligent teaching-Learning Environment*).

Capítulo 3 – Java como Linguagem de Programação para a *Web*

Será feito um estudo sobre a linguagem de programação Java. Serão detalhados *servlets* Java e a API JBDC.

Capítulo 4 – Objetos Distribuídos

Serão estudados alguns padrões para sistemas distribuídos com suporte a criação de objetos distribuídos: DCOM da Microsoft, CORBA do OMG (*Object*

Management Group) e Java/RMI. Também, será feita uma análise comparativa entre estes padrões estudados.

Capítulo 5 – Módulo de Integração de Bancos de Dados em Sistemas Tutores Inteligentes

Serão descritos o protótipo desenvolvido e suas características.

Capítulo 6 – Conclusões e Trabalhos Futuros

Serão apresentados os comentários conclusivos e destacadas as contribuições e sugestões de trabalhos futuros.

Capítulo 2

Sistemas Tutores Inteligentes

Este capítulo aborda Sistemas Tutores Inteligentes, focalizando um breve histórico, sua arquitetura básica e aspectos concernentes ao desenvolvimento destes sistemas. Finalmente, será apresentado um estudo de caso, o WILE (*Web based Intelligent teaching-Learning Environment*).

Capítulo 2

Sistemas Tutores Inteligentes

A informática destaca-se entre as ferramentas utilizadas na atualidade para a educação, representando novas maneiras de se atingir um dos objetivos principais do processo ensino/aprendizagem que é tornar a compreensão e absorção do conhecimento mais fácil e consistente. Com o uso das tecnologias de informação, pode-se conseguir uma maior atenção dos aprendizes, formas diferentes de se transmitir a teoria e simular a prática, focalizando todos os sentidos conhecidos para o assunto abordado.

A Multimídia [83], com todos os seus recursos atraentes, e a Inteligência Artificial [41], com a qual pode-se simular raciocínio e poder de abstração humanos, nos proporcionam hoje a construção de sistemas educacionais cada vez mais envolventes e úteis. Com o propósito não de substituir os professores, mas direcionados para o auxílio à aprendizagem, os sistemas educacionais são desenvolvidos na tentativa de conseguir resultados não atingidos utilizando-se apenas as técnicas pedagógicas, como também para permitir a realização de treinamentos e estudos sem a presença ou proximidade obrigatória do mestre humano.

Neste capítulo, serão abordados os Sistemas Tutores Inteligentes, sistemas voltados à educação com o auxílio do computador e que utilizam técnicas de Inteligência Artificial para flexibilizar o ensino, permitindo uma melhor interação entre o sistema e os usuários [3, 38, 69, 75, 99, 110, 119, 121].

2.1. Breve Histórico

A área de Instrução Assistida por Computador (*Computer-Aided Instruction*, CAI) [111] existe desde a década de 50. A abordagem então utilizada baseava-se na psicologia comportamentista e caracterizava-se pelo conceito de “programa linear”, que funciona da seguinte maneira: o sistema apresenta uma determinada unidade de texto, a qual deve levar o aluno a um pequeno passo adiante na direção de um

comportamento esperado; o aluno então responde de alguma forma, por tentativa e erro ou baseado em conhecimento prévio e em seguida, o sistema informa se a resposta está correta ou não. Com isso, o aluno pode aprender em seu próprio ritmo, recebendo "*feedback*" imediato.

No final dos anos 60 e no início dos anos 70, os pesquisadores interessados no uso do computador na educação sustentavam que um programa tutorial deveria ter uma representação daquilo que se propunha a ensinar, além de um modelo de quem receberia o ensinamento e as estratégias que orientassem o aluno.

Os programas tradicionais de CAI são desenvolvidos através do encapsulamento das decisões pedagógicas em seu código, de forma que a base de conhecimento utilizada é fixa. Este tipo de programa apresenta ainda a limitação de não considerar as características individuais dos alunos.

Reconhecendo as limitações dos CAIs, os pesquisadores passaram a recorrer a técnicas de representação do conhecimento, métodos de inferência e estratégias de controle, visando buscar soluções para a questão da adaptação dos sistemas tutores aos estudantes, individualmente. Surgiu então a área denominada ICAI (*Intelligent CAI*).

Em 1982, Sleeman e Brown [102] publicaram uma edição especial do *International Journal of Man-Machine Studies* na forma de um livro que batizou a área com o seu nome mais conhecido: *Intelligent Tutoring Systems*. Em 1988, outro livro foi publicado, sob a supervisão de Self*: Artificial Intelligence and Human Learning* [96]. A partir deste último, é possível perceber o estado da arte em STI e algumas de suas perspectivas e direções futuras.

2.2. Arquitetura Básica

Uma arquitetura deve separar a funcionalidade dos STIs numa coleção de componentes de software com interface e projetos de execução muito bem definidos, com o intuito de alcançar os seguintes objetivos [114]:

- Isolamento e inserção de tecnologia – permitir a inclusão de novas tecnologias, assim como, diminuir as dependências nas tecnologias existentes;

- Facilidade de aquisição – reduzir custos de desenvolvimento e manutenção associados à construção de múltiplos STIs;
- Diminuição da dependência de domínio – permitir amenizar as dependências de domínio dos STIs;
- Diminuição das estratégias instrucionais – permitir estratégias instrucionais com impacto mínimo em outros STIs.

Várias propostas de arquitetura são encontradas na literatura [7, 10, 14, 18, 27, 40, 44, 65, 67, 84, 87, 108, 109, 111, 114, 120].

Uma arquitetura básica, proposta por Fischetti e Gisolf [40] e representada na figura 2.1, é constituída por:

- Modelo do Domínio – define o conhecimento contido no domínio a ser ensinado;
- Modelo do Aluno[1] – é capaz de definir o conhecimento do aluno em cada ponto durante a instrução;
- Modelo do Tutor – é responsável pelas estratégias de ensino-aprendizagem adequadas;
- Interface – permite a interação do estudante com o STI de maneira eficiente.

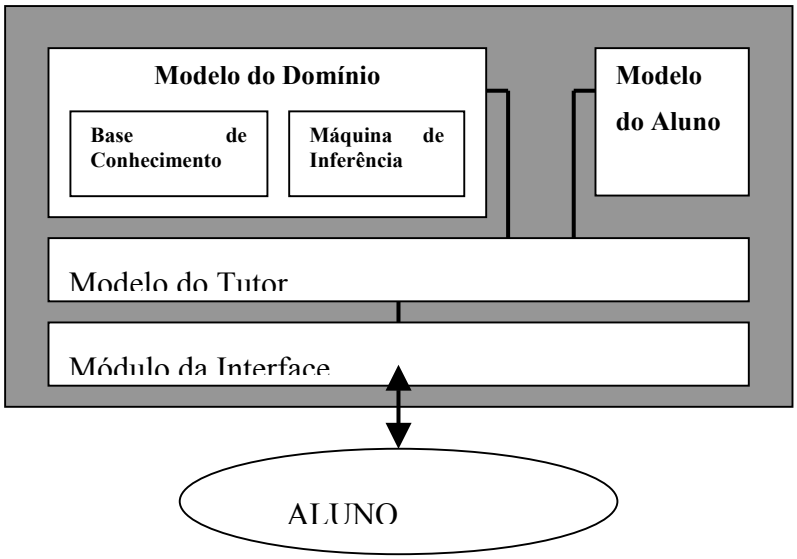

Figura 2.1: Arquitetura Geral dos STIs, baseada em [40]

[1] O modelo do aluno também pode conter características sócio-econômicas, não apenas a evolução acadêmica.

2.3. Dificuldades no Desenvolvimento de STIs

Nos últimos anos, os STIs têm sido objeto de intensa pesquisa, o que pode ser medido através do crescente número de artigos publicados em congressos e revistas, tais como: AI-ED, ED-MEDIA, *Journal of Artificial Intelligence in Education, International Conference of Intelligent Tutoring Systems*. Mesmo assim, a utilização dos STIs em situações educacionais reais permanece restrita.

Segundo Costa [26], os principais fatores envolvidos na baixa disseminação de produtos de software inteligentes são o seu alto custo e o longo tempo requerido para o seu desenvolvimento. Além disso, a necessidade de envolvimento de equipes multidisciplinares e a falta de métodos de desenvolvimento e de métodos de avaliação de qualidade são considerados como pontos críticos.

Alguns dos fatores que dificultam o desenvolvimento e o uso dos STIs atualmente, são:

- A Falta de Métodos Específicos de Engenharia de Software para o Desenvolvimento de STIs [2, 26, 34, 37, 38];
- A Falta de Métodos de Avaliação da Qualidade para STIs [42, 67, 86];
- A Falta de Modelos Pedagógicos Próprios para STIs [24, 26, 86].

A comunidade de pesquisadores na área vem trabalhando ativamente para a eliminação dos problemas anteriores, bem como para obter uma maior usabilidade dos STIs. Um dos trabalhos na direção do aumento de usabilidade desses sistemas é detalhado na seção seguinte.

2.4. WILE – Ambiente Inteligente de Ensino/Aprendizagem Via Web

Atualmente, o paradigma do aprendizado moderno está sendo liderado pelas abordagens “aprender fazendo” e “aprender cooperando”, como proposto por Piaget [82] e Vigotsky [100], respectivamente.

Ambientes computacionais que dão suporte adequadamente a estas abordagens não são fáceis de se encontrar atualmente. Não é comum, por exemplo, encontrar um ambiente computadorizado que dê suporte a estas novas abordagens com uma metodologia flexível e uma interface amigável.

Quando um trabalho de qualquer natureza é feito de maneira cooperativa, o resultado aparenta ser bem melhor que quando é obtido de uma abordagem individual. Desenvolvendo atividades cooperativas, os estudantes trocam experiências e têm contato com outras possibilidades para resolver o mesmo problema.

A disponibilização e integração de bancos de dados surgem como tendência de serviços oferecidos pela *Web*. O objetivo de um sistema de integração de dados é oferecer aos usuários uma interface uniforme de acesso a diferentes fontes de dados, de forma que os usuários definam consultas especificando o que desejam saber e o sistema determine onde a informação pode ser encontrada e, em seguida, apresente as respostas para as consultas do usuário.

No intuito de atender os requisitos descritos acima, foi proposto um ambiente inteligente de ensino-aprendizagem via *Web* chamado WILE [103] (*Web based Intelligent teaching Learning Environment*), que é uma extensão de um projeto em andamento na Universidade Federal de Pernambuco chamado Sistema de Ensino Inteligente (SEI) [104].

Todas as características do SEI são incorporadas pelo WILE, através do módulo de ensino-aprendizagem. O ambiente é baseado em um sistema tutor inteligente (STI), que é um recurso educacional importante no processo de ensino-aprendizagem [28] permitindo tornar a arquitetura mais adaptável e portável. Esse ambiente é fundamentalmente cooperativo e colaborativo, pois sua arquitetura permite que os participantes tenham acesso ao curso e às pesquisas individuais de cada participante, criando uma maior integração e permitindo a troca de informações. WILE permite a construção e o acúmulo de informação através das bases de conhecimento do próprio ambiente e das bases de dados individuais de cada aluno.

WILE é baseado em uma arquitetura cliente-servidor, podendo ser utilizado sob duas instâncias:

- em sala de aula presencial, o aluno se conecta à rede e tem acesso a vários tipos de informações, como por exemplo respostas às suas dúvidas, aos exercícios e pesquisas propostos tanto pelos professores quanto pelos próprios alunos;

- após a aula, os conteúdos trabalhados, as pesquisas e todo o ambiente cooperativo estarão disponíveis através da *Web*, em qualquer lugar e momento de acordo com o interesse, necessidade e/ou viabilidade do aluno.

Seus módulos constituintes são: módulo de ensino-aprendizagem, módulo de autoria, módulo de integração e módulo de comunicação, e são mostrados na figura 2.2.

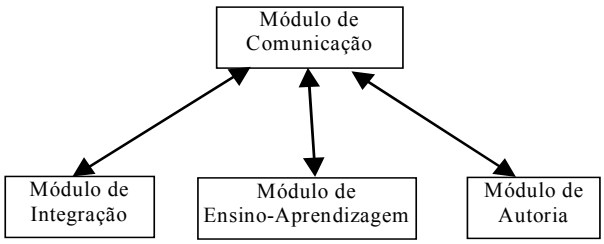

Figura 2.2. Organização dos módulos do WILE

A seguir, serão detalhados os módulos constituintes do WILE.

2.4.1. Módulo de Ensino-Aprendizagem

Esse módulo apresentado em Tedesco [104] possui duas bases de conhecimento: o modelo de estudante – armazena informações relacionadas ao estudante, tais como características cognitivas, histórico da interação, caminhos percorridos no currículo e o modelo de domínio – ele é organizado como uma rede de *frames*, possuindo quatro níveis de abstração: Cursos, Lições, Tópicos, Apresentações. O módulo também possui uma sociedade de agentes inteligentes que é constituída de: Estudante, Domínio, Tutor, Controlador e Comunicador. A figura 2.3 apresenta a arquitetura do módulo ensino-aprendizagem.

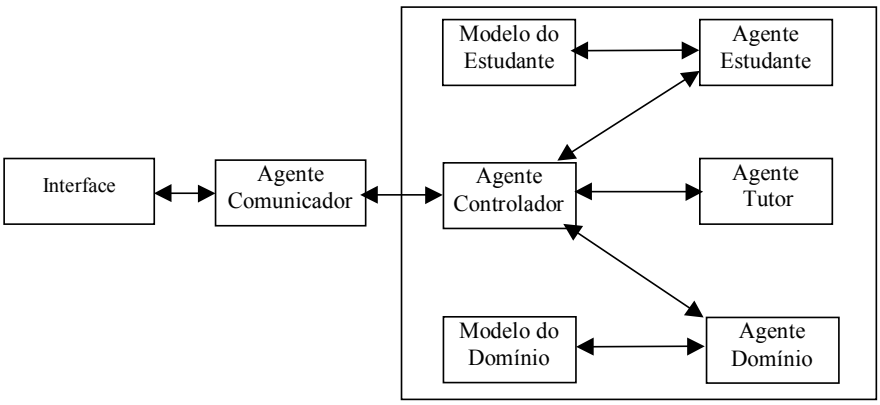

Figura 2.3 Arquitetura do módulo de ensino-aprendizagem

O Modelo do Estudante armazena informações relativas ao estudante corrente e dispõe de três estereótipos para acomodar as diferenças relativas ao nível de conhecimento prévio do aprendiz: Usuário Especialista, Usuário Casual e Usuário Leigo. Esta atribuição é feita na primeira interação com o sistema. A atualização deste modelo é feita a cada passo da interação do aluno com o sistema.

O Modelo do Domínio armazena o conteúdo do curso. Está organizado como uma rede de *frames* [41] que possui quatro níveis de abstração: Curso, Lições, Tópicos e Apresentações. O sistema apresenta as informações em três níveis de aprofundamento, de acordo com as categorias de usuários citadas anteriormente.

O Agente Controlador controla o funcionamento do sistema como um todo gerenciando a troca de informações entre os agentes. É modelado através de regras de produção. Este agente mantém um registro de todos os agentes do sistema, serviços disponíveis e localizações, além de controlar a comunicação entre eles.

O Agente Estudante manipula informações relacionadas aos estudantes e determina o perfil do estudante corrente. Para cada sessão tutorial, um modelo de estudante do aluno é criado, caso seja a primeira interação, ou recuperado para que o sistema execute as suas tarefas de acordo com as interações anteriores.

O Agente Domínio recupera informações do Modelo do Domínio e avalia as respostas do estudante aos exercícios propostos, auxiliando a determinação do desempenho e do perfil do estudante corrente.

O Agente Tutor toma as decisões pedagógicas, ou seja, determina *o* que o sistema vai ensinar, como e quando. Também determina as estratégias de resposta ao aluno de acordo com o seu desempenho e perfil, raciocinando com base nos Modelos

do Estudante e do Domínio. O conhecimento deste agente é modelado através de regras de produção, agrupadas de acordo com vários aspectos da interação como motivação do estudante, desempenho, estratégias pedagógicas, dentre outros.

O Agente Comunicador trata os eventos da interface do sistema apresentando ao aluno o conteúdo, informações complementares, exercícios a serem respondidos e comentários do Tutor. Além disso captura as respostas do aluno ao sistema. Está implementado utilizando *servlet* JAVA, para aumentar o alcance do STI com a sua disponibilização através da WWW.

A interface do sistema é toda feita em HTML. Cabe a esse agente tanto capturar os eventos da interface quanto produzir novas telas para responder às ações dos usuários. É através da Interface que acontece toda a comunicação com o sistema. Possui as seguintes funcionalidades:

- Janela de Apresentação, para o conteúdo do curso;
- Janela de Navegação, para acesso à estrutura curricular do curso;
- Janela de Informações Complementares, para informações adicionais;
- Janela de Comentários, para respostas do Tutor às ações do aluno;
- Ajuda, para auxílio sobre a utilização do sistema;
- Menu "Perguntar", para o aluno solicitar informações mais detalhadas durante o curso.

O módulo de ensino/aprendizagem comunica-se com os professores, através de relatórios sobre o progresso dos alunos, seus erros e desempenho.

Tem-se dois tipos de usuários: estudantes, que podem acessar o sistema individualmente ou associados a uma turma e, professores, que controlam o processo de ensino através da supervisão do comportamento da turma frente ao próprio sistema.

2.4.2. Módulo de Autoria

O módulo de Autoria apresentado em Campos & Souza [16] permite a construção automática de conhecimento em um STI adotando uma abordagem cooperativa. Este é constituído de ferramentas que incluem um editor de página *Web*

(*Web Page Editor* – WPE), um editor de domínio de conhecimento (*Knowledge Domain Editor* – KDE) que permite gerar páginas com texto, imagem, som e animação e um gerenciador de *site* (*Site Manager* – SM).

O KDE permite a inserção e o relacionamento de dados de acordo com quatro níveis de abstração na base de conhecimento. O dados podem ser textuais, imagens, sons e vídeo, exercícios, explicações, erros, que são organizados seguindo a estrutura de: apresentações, tópicos, lições e curso, o qual permite a elaboração do conteúdo didático a ser apresentado ao estudante, tendo o autor acesso tanto a recursos computacionais locais quanto da internet para a atividade de criar a base de conhecimento.

O WPE auxilia a criação de uma rede hipermídia organizada como um conjunto de páginas interligadas, onde as páginas são visitadas em uma seqüência específica, e cada uma delas pode conter conteúdo estático ou conteúdo dinâmico que é gerado em tempo de execução. O autor é responsável pela descrição da rede hipermídia determinando os relacionamentos entre as páginas. A montagem da aplicação envolve também a especificação de métodos de avaliação do estudante e sua monitoração durante atividade no STI. Este sistema pode ser executado a partir de qualquer *browser* permitindo a autores separados geograficamente, elaborar um curso no WILE.

O módulo SM permite ao(s) autor(es), mesmo sem nenhum conhecimento de como funciona o sistema de arquivos da máquina servidora, compartilhar a mesma área de trabalho. Os autores são responsáveis pela produção e acompanhamento do projeto. O SM permite criação, remoção e atualização de tópicos relativos a um curso. Um autor atribuído a um tópico pode utilizar a base de conhecimento para criar, remover ou atualizar páginas *Web*, bem como inserir dados na base de conhecimento. A ferramenta tem controle sobre definição dos tópicos do curso por autor. O autor em seu tópico específico pode remover, inserir e atualizar uma apresentação, ou seja tem controle total sobre seu tópico.

O autor, utilizando este sistema, pode inserir dados em uma base de conhecimento utilizando o KDE, criar páginas *Web* a partir de *templates* fornecidos pelo sistema WPE e gerenciar o curso utilizando o SM. Assim, WILE permite a autoria, disponibilização e gerenciamento de materiais didáticos voltados para o ambiente *Web*. A conexão ao servidor de BD é automática para utilização ou criação de um domínio de

conhecimento específico. O processo de elaboração do material didático consiste na construção de forma cooperativa através da *Web* de uma rede Hipermídia/Hipertexto e de conhecimento, determinando o relacionamento e dependência entre os tópicos a serem manipulados pelo STI. O sistema provê uma interface de desenvolvimento uniforme para seus usuários. Este STI pode ser utilizado em diferentes áreas do conhecimento como Ciência da Computação ou Engenharia Civil, por exemplo.

2.4.3. Módulo de Comunicação

A comunicação entre o agente comunicador (AC) e a interface é feita através de *servlets* [52, 54, 97]. Cada ocorrência de eventos na interface é passada para o AC por um deles. Para cada tipo de evento existe um *handler*[1] no AC para processá-lo e produzir uma resposta adequada. O *servlet* então recebe a resposta produzida pelo AC e envia de volta ao *browser*. Por exemplo, se um estudante pede informações a respeito de um dado tópico, então existe um *handler* no AC para tratar a requisição. O *servlet* apenas informa a requisição ao agente e este chama o *handler* apropriado para manipulá-la. Se o estudante está resolvendo um exercício, o *servlet* informa as respostas do aluno ao AC e o agente fica responsável por endereçá-las ao *handler* específico.

Seguindo essa estrutura, é correto afirmar que s*ervlets* no sistema WILE são usados como uma única ponte entre a interface e núcleo do sistema. Além disso, não fazem nenhuma outra computação. Eles somente delegam este trabalho para os *handlers* do AC. Isto acontece para simplificar a estrutura dos *servlets*: eles somente enviam requisições do cliente e retornam as respostas do sistema.

Os *servlets* são programas residentes no lado do servidor, por esse motivo, o único *servlet* do módulo de ensino-aprendizagem pode ser visto como parte da arquitetura do AC. Ou melhor, pode ser visto como a fachada do agente comunicador, já que tudo que ocorre na interface chega ao agente através deles. As respostas do sistema (respostas geradas a partir da interação dos agentes) são enviadas ao AC. Como o cliente está utilizando um *browser*, essas respostas precisam ser enviadas em

[1] *Handler* é uma representação interna responsável para executar uma ação previamente escolhida.

um formato aceitável por *browsers*. Assim, o agente comunicador fica responsável por receber as respostas dos agentes e montar e enviar uma página HTML de volta para o *servlet*. Deste, a página chegará ao *web browser*. A figura 2.4 ilustra o módulo de comunicação.

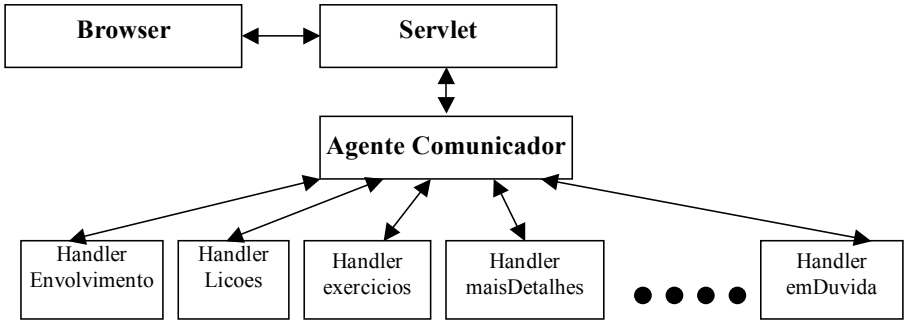

Figura 2.4. Arquitetura do módulo de comunicação

Em muitas arquiteturas, os *servlets* também são responsáveis por montar as páginas HTML. Outras têm adotado a filosofia do seu uso orientado à ação. Isto significa que para cada possível ação existirá um *servlet* para tratá-la. Assim, gera-se um grande número deles no sistema, o que pode causar problemas de performance. O módulo de comunicação usa apenas um deles. Além disso, as páginas HTML são montadas por *handlers* específicos disponibilizados fora do *servlet*, o que contribui para um código mais simples e leve, resultando melhor desempenho do sistema.

2.4.4. Módulo de Integração

Segundo Viccari e Giraffa [108], uma arquitetura para um sistema tutor inteligente deve possuir duas bases de conhecimento: o modelo do aluno e a base do domínio.

Para poder tratar os estudantes de maneira individualizada, um STI deve contar com uma coleção de informações, onde ficam armazenadas as características do estudante que são relevantes para o processo de ensino-aprendizado. Assim, cada estudante possui seu modelo de aluno, contendo informações como:

- Características cognitivas relevantes aferidas a partir de um diálogo direto do sistema com o estudante;

- Caminho de aprendizado percorrido pelo aluno até o presente momento;
- Curso básico a ser sugerido ao aluno;
- Histórico do progresso que registra conhecimentos e habilidades que o aluno presumivelmente possui no momento;
- Calendário da interação que consiste nas datas de visita do aluno aos diversos tópicos do currículo;
- Tópico corrente que armazena o último tópico visitado pelo aluno, para que o estado corrente da interação possa ser restaurado, a cada sessão;
- Resultado da interação inicial, necessário para obter informações para iniciar o Modelo do Estudante.

A base do domínio consiste no conteúdo instrucional que vai ser apresentado ao aluno: lições, exercícios, informações complementares, ajuda, dentre outros.

Na arquitetura proposta por Souza e Campos [103], além das bases de conhecimento, (modelo do aluno e base do domínio), cada estudante tem sua própria base de dados, chamada de cooperante, que é constituída por suas pesquisas. A utilização de bases de dados cooperantes traz algumas vantagens:

- Compartilhamento de dados – Se as bases de dados cooperantes estão conectadas, qualquer usuário habilitado pode ser capaz de fazer acesso a dados disponíveis nestas bases;
- Confiabilidade e disponibilidade – Se uma base de dados não estiver conectada ou falhar no sistema de integração, as bases remanescentes podem ser capazes de continuar operando. Assim, a falha de uma base de dados cooperante não implica necessariamente no desligamento do sistema;
- Aceleração de processamento de consultas – Se uma consulta envolve dados em diversas bases, é possível dividi-la em sub-consultas que podem ser executadas em paralelo;
- Independência de localização - se os dados estiverem localizados em várias bases de dados, isso deve ser imperceptível a qualquer cliente;
- Escalabilidade – como em cada base de dados cooperante ocorre inclusão de

dados (pesquisas do estudante), as bases de conhecimentos sofrem considerável aumento de conteúdo (informação);

- Autonomia e Descentralização – as bases de dados que fornecem os dados para o sistema de integração são autônomas, ou seja, suportam aplicações locais, podendo sofrer alterações nos dados, como também, ocorrendo falha no servidor, não há prejuízo no processo de ensino-aprendizagem;

Essa base de dados local e cooperante pode ser compartilhada com o grupo, permitindo que os participantes tenham acesso às pesquisas individuais de cada participante, permitindo uma maior integração e troca de informações, permitindo a construção e o acúmulo de informação através das bases de conhecimento do próprio ambiente, como também, através das bases de dados individuais de cada aluno, contribuindo assim para um ambiente efetivamente cooperativo.

Sua arquitetura, mostrada na figura 2.5, permite que qualquer usuário do sistema visualize, através do *browser*, as informações disponíveis de seu grupo de estudos. O tipo de informação que o usuário pode recuperar é limitado ao conjunto de funcionalidades disponibilizadas pelas páginas HTML, que são acessos à base de dados local de um estudante pelo tópico ou pela lição para recuperação das pesquisas realizadas pelo estudante.

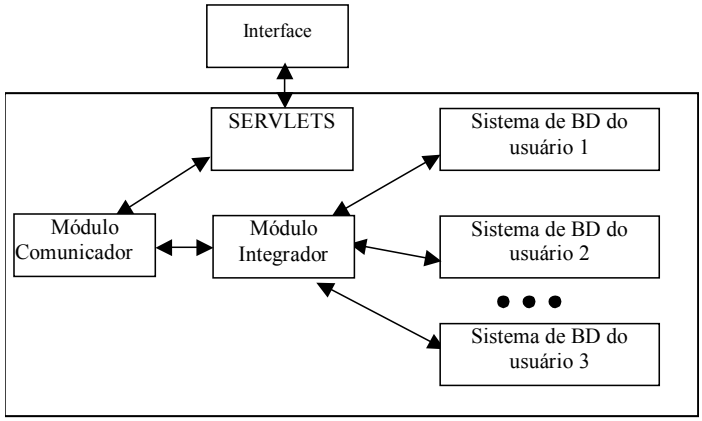

Figura 2.5. Arquitetura do módulo de integração

Desta forma, cada usuário tem armazenado localmente sua própria base de dados, que é similar à base de conhecimento disponibilizada pelo sistema, somente que esta é constituída pelas pesquisas do estudante. Quando o usuário está alimentando sua base de dados com informações de pesquisa na *Web*, informações adicionais são armazenadas para fins de recuperação, são elas: a lição e o tópico da lição.

O módulo de integração implementa um conjunto de consultas pré-definidas, que permite o acesso ao banco de dados de outros estudantes pertencentes ao curso. As solicitações são: consulta sobre uma única base de dados específica para um tópico ou para toda a lição ou mais de uma base de dados para um determinado tópico ou lição. Sua função é recuperar dados das fontes escolhidas submetendo os pedidos e realizar o processamento necessário para integrar dados de diversas fontes. O caso mais simples é quando é feita a consulta sobre uma única fonte de dados. O resultado é devolvido para o usuário através do módulo comunicador. Todo o conhecimento do módulo de integração está no servidor.

A comunicação entre o módulo Integrador (MI), e a interface é também feita através de *servlets*. Analogamente a comunicação de cada ocorrência de evento na interface é passada para o MI por um *servlet*. Para cada tipo de evento existe um *handler* no MI para processá-lo e produzir uma resposta adequada. O *servlet* então recebe a resposta produzida pelo MI e a envia de volta ao *browser*.

Por exemplo, se um estudante pede informações armazenadas nas bases de dados locais de cada estudante a respeito de um dado tópico, então existe um *handler* no MI para tratar a requisição. O *servlet* apenas informa a requisição ao módulo comunicador e este chama o *handler* apropriado para manipulá-la. Se o estudante está solicitando as pesquisas realizadas por outros estudantes, o *servlet* informa esta solicitação do aluno ao MI e este fica responsável por endereçá-las ao *handler* específico.

2.5. Conclusão

Os Sistemas Tutores Inteligentes possibilitam um ensino auxiliado por computador flexível e inteligente na forma de apresentação do conteúdo, na avaliação de qual material será exposto ao aluno dependendo do seu nível de aprendizado, na sua forma de comunicação onde predominam o entendimento do aluno e a satisfação deste no uso de um sistema agradável e atraente.

É percebido, o progresso da pesquisa nesta área, como também a expansão no uso dos sistemas tutores inteligentes como sistemas auxiliares na educação, utilizados sozinhos ou paralelamente à aplicação de aulas normais por professores humanos.

Muitos problemas ainda são encontrados em relação ao desenvolvimento, validação da qualidade e utilização de modelos pedagógicos tradicionais nos STIs, mesmo estando em andamento pesquisas relacionadas a estas áreas.

A construção do WILE como uma sociedade de agentes inteligentes cooperantes dá margem a outras extensões para os trabalhos na área. Um STI desenvolvido sobre a *Web* e utilizando uma plataforma de software amplamente disponível, de baixo custo e arquitetura neutra aumenta enormemente o potencial destes sistemas para o ensino e treinamento a distância.

A possibilidade do estudante em possuir sua própria base de dados, que é constituída por suas pesquisas, traz diversas vantagens, mostradas na seção 2.4.4. O compartilhamento dessa base de dados é de fundamental importância para a melhoria no processo de ensino/aprendizagem em um STI. Dessa forma, a integração das bases de dados dos estudantes se faz necessária e se torna uma opção para o aprendizado cooperativo. Tal integração é o objetivo principal deste trabalho.

O WILE ainda está em desenvolvimento e após sua conclusão necessitará de testes exaustivos, avaliações e ajustes, com a intenção de melhorar a sua eficiência e seu potencial de reutilização, através do refinamento da estrutura da sociedade de agentes e seus mecanismos de raciocínio.

Capítulo 3

Java como Linguagem de Programação para a *Web*

Neste capítulo, será feito um estudo sobre a linguagem de programação Java. Serão detalhados *servlets* Java e a API JBDC.

Capítulo 3

Java como Linguagem de Programação para a *Web*

A linguagem de programação Java [56] foi desenvolvida pela Sun Microsystems no início da década de 90, quando o objetivo principal era o uso de uma linguagem de programação que permitisse a integração total de sistemas de computação com equipamentos de consumo como caixas de comutação de TV a cabo [51].

A linguagem é derivada da sintaxe da linguagem C++, permitindo aos programadores familiarizados com esta, se adaptarem facilmente a Java, que estende a linguagem C++ fornecendo capacidades de sistemas distribuídos e construção de aplicações para a Internet [106].

Uma das suas maiores vantagens é a portabilidade, ou seja, um programa escrito em Java pode ser executado em várias plataformas, como Windows NT, Unix ou Solaris. Essa característica a torna uma linguagem de programação ideal para a WWW, pois um programa Java pode ser executado em qualquer plataforma sem necessidade de alterações no código-fonte. Tal funcionalidade é possível através do processo de compilação que gera, a partir de um código-fonte, um código intermediário chamado *bytecode* que é interpretado em qualquer plataforma com suporte à Java, pela sua máquina virtual. Além desta importante funcionalidade, Java também se caracteriza por ser [51]:

- Orientada a objetos: segue os conceitos da orientação a objetos baseados em classes que incluem os dados e as operações sobre esses dados. Assim, encapsulamento, herança, polimorfismo e identificadores de objetos são suportados pela linguagem;
- Robusta: foi projetada para se escrever software mais confiável e robusto, mas isto não elimina a necessidade de se escrever programas com qualidade. Java é fortemente tipada, e permite verificação de erros de tipo em tempo de compilação, pois requer declarações explícitas para seus métodos. O tratamento de erros (exceções) torna o desenvolvimento de aplicações mais confiável;

- Segura: Um dos seus aspectos mais importantes é a segurança, por causa da natureza distribuída da linguagem. Java implementa várias camadas de controle de segurança contra códigos maliciosos. Na camada mais baixa, não permite o acesso a ponteiros o que dificulta a execução de código malicioso. Em outra camada, o interpretador Java não permite que o código Java (*bytecode*) seja alterado durante a sua carga;
- Simples: possui uma série de funcionalidades que facilitam a escrita de um código limpo e correto, tais como: coletor de lixo (*garbage collection*), impossibilidade de usar ponteiros, controle de exceções e eventos além da semelhança com outras linguagens de programação.

Além das propriedades acima, Java permite múltiplas linhas de execução (*multithreading*). A linguagem também é dinâmica, ou seja, pequenas partes do código são montadas em tempo de execução dentro do programa.

O processo de compilação e execução de programas Java é representado na figura 3.1.

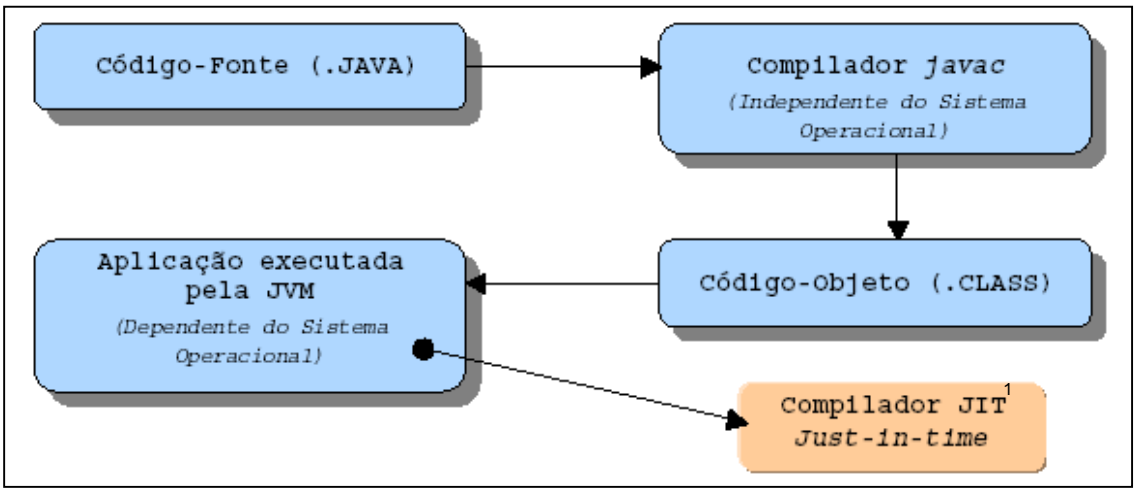

Figura 3.1. Esquema de compilação e execução de programas Java, baseado em [51]

[1] Compilador JIT é um programa que compila o *bytecode* Java (arquivo .class) para o código nativo da máquina que o está executando.

Com relação a ser uma linguagem particularmente adequada a WWW, Java se popularizou com pequenos aplicativos chamados *applets* [51, 56]. Estes pequenos aplicativos são carregados dentro do código de uma página HTML [17] e rodam geralmente dentro dos *browsers* que acessam a Internet. Um outro conjunto de aplicação, os *servlets* [52, 54, 97], estão cada vez mais popularizando a linguagem. *Servlets* são aplicativos que podem ser acoplados em diversos tipos de servidores para expandir suas funcionalidades. Podem ser escritos para trabalhar com diversos tipos de protocolo de comunicação, como SMTP [89], HTTP [50] e IRC [47].

Algumas capacidades dos *servlets* tornam esta tecnologia particularmente interessante:

- Geração dinâmica de páginas HTML — podem ser instalados em servidores *web* para processar informações transmitidas via HTTP a partir, por exemplo de formulários HTML. As aplicações podem incluir acesso a banco de dados ou comunicação com outros *servlets*;
- Balanceamento de carga entre servidores — para entender como utilizar *servlets* para fazer balanceamento de carga, pode-se exemplificar com a seguinte situação: um provedor de serviços via Internet possui uma infra-estrutura composta de cinco servidores, dos quais, quatro são capazes de executar as mesmas aplicações. O servidor restante seria responsável por monitorar a carga dos demais e receber o acesso inicial de cada cliente às aplicações e em seguida, redirecionar os pedidos de acesso para um dos quatro servidores de aplicação, conforme a ocupação de cada um no momento em que o cliente tenta estabelecer uma conexão. Assim, o cliente passa a trocar informações somente com o servidor que foi alvo do redirecionamento;
- Modularização do código — um *servlet* pode executar outro *servlet*, mesmo que remotamente, desta forma é possível executá-los em cadeia. Esta característica possibilita a modularização dos aplicativos, criando *servlets* com funções específicas. Supondo que para acessar um conjunto de aplicativos, o cliente deva ser autenticado. Neste caso, uma configuração possível seria criar um *servlet* responsável apenas pela tarefa de autenticação. Uma vez autenticado, este *servlet* redirecionaria o cliente para outro *servlet*, não necessariamente instalado no mesmo servidor, que executaria o aplicativo. A vantagem deste tipo de

arquitetura, é que se por alguma razão for necessário modificar o procedimento de autenticação, por exemplo pela mudança do banco de dados de usuários, não será necessário reescrever toda a aplicação, e sim apenas o *servlet* responsável pela autenticação.

As aplicações atuais da *Web* tendem a requerer cada vez mais conteúdo dinâmico. Uma das formas mais seguras de armazenamento deste conteúdo é através de Sistemas de Gerenciamento de Banco de Dados (SGBD). O principal problema encontrado nas aplicações tradicionais de banco de dados, entretanto, é que os códigos dos programas ficam dependentes das características dos bancos de dados que são utilizados. Se a aplicação necessita se comunicar com outros bancos de dados, será necessária a alteração dos programas que acessam esses bancos de dados.

A linguagem Java apresentou como solução, uma API chamada JDBC (*Java Database Conectivity*) [55], que é um conjunto de classes e interfaces escritas na própria linguagem para facilitar o envio de comandos SQL (*Structured Query Language*) [36] para sistemas de bancos de dados relacionais.

A vantagem de usar a tecnologia JDBC está em construir aplicações que podem acessar várias fontes de dados heterogêneos, podendo executar essa aplicação em qualquer plataforma que possua uma máquina virtual Java. A seção 3.3 detalha essa tecnologia.

Ainda, com referência à conectividade com banco de dados, devido ao seu ciclo de vida, o *servlet* permite manter abertas as conexões ao banco de dados. Uma conexão existente pode cortar vários segundos de um tempo de resposta, comparado com um *script* CGI [116] que tem que restabelecer sua conexão para cada invocação.

Outra vantagem de *servlets* sobre CGI e muitas outras tecnologias é que, juntamente com a API JDBC, torna-se independente de banco de dados. Um *servlet* escrito para acessar um banco de dados Sybase pode, com a modificação de duas linhas ou uma alteração em um arquivo de propriedades, começar a acessar um banco de dados Oracle.

A seção seguinte detalha *servlets* Java.

3.1. *Servlets* Java

Como ressaltado anteriormente, *servlet* é uma extensão genérica do servidor – uma classe Java que pode ser carregada dinamicamente para expandir a funcionalidade de um servidor. *Servlets* são comumente usados com servidores *web*, onde podem tomar o lugar dos *scripts* CGI. Um *servlet* é similar a uma extensão proprietária de servidor, exceto que ele roda dentro de uma JVM no servidor, assim é seguro e portável. *Servlets* operam unicamente dentro do domínio do servidor, assim, eles não requerem suporte para Java no *browser*.

Diferentemente de CGI e FastCGI[1], os quais usam múltiplos processos para manipular programas em separado e/ou requisições em separado, *servlets* são todos manipulados por *threads* em separado dentro de um processo no servidor *web*. Isto significa que eles são também eficientes e escaláveis. Devido a rodarem dentro de um servidor *web*, eles podem interagir mais proximamente com o servidor para fazer coisas que não são possíveis com *scripts* CGI.

Embora *servlets* sejam mais comumente usados como substituição a *scripts* CGI em um servidor *web*, também podem estender qualquer tipo de servidor, por exemplo, um servidor FTP baseado em *Java* que manipula cada comando com um *servlet* em separado. Novos comandos podem ser adicionados em novos *servlets*. Ou, um servidor de correio que permite aos *servlets* estender sua funcionalidade, por exemplo, executando uma busca de vírus em todos os documentos anexados ou manipulando tarefas de filtragem de mensagens.

Como a linguagem Java, *servlets* foram projetados para portabilidade. Eles têm suporte em todas as plataformas que rodam *Java*, e trabalham com todos os principais servidores *web*. *Servlets* Java, como definido pela Java Software *Division* da Sun Microsystems (formalmente conhecida como JavaSoft), são a primeira extensão

[1] A diferença entre CGI e FastCGI é que FastCGI cria um simples processo persistente para cada *script* CGI. Isso elimina a necessidade de criar um novo processo para cada requisição.

padrão para Java. Isto significa que *servlets* são oficialmente aprovados pela Sun e são parte da linguagem Java, mas não são parte do núcleo da *Application Programming Interface (*API) de Java. Portanto, embora eles possam trabalhar com qualquer Máquina Virtual Java, as classes *servlet* não precisam ser empacotadas em todas elas.

Para facilitar o desenvolvimento de *servlets*, a Sun tem tornado publicamente disponível um conjunto de classes que provêm suporte básico a *servlet*. Os pacotes *javax.servlet* e *javax.servlet.http* constituem esta API. A versão 2.0 destas classes vem embutida com o *Java Servlet Development Kit (*JSDK*)* [56] para uso com o *Java Development Kit* (JDK) [56] versão 1.1 em diante.

Muitos vendedores de servidores *web* têm incorporado estas classes em seus servidores para prover suporte a *servlet*, e vários têm também provido funcionalidade adicional. O servidor *web* Java da Sun [56], por exemplo, inclui uma interface proprietária para aspectos de segurança do servidor.

3.1.1. "Máquinas" *Servlet*

Somando-se às classes *servlet*, é necessária uma "máquina" *servlet*, para disponibilizá-los. A escolha dessa "máquina" *servlet* depende em parte do servidor ou dos servidores que estão disponíveis. Existem três tipos de "máquinas" *servlet* [54]:

- *Standalone* – *s*ervidor Web que inclui suporte embutido para *servlets*. Uma desvantagem, é que se tem que esperar um novo lançamento do servidor *Web* para obter o suporte mais recente ao *servlet*.
- *Add-on* – *u*ma "máquina" *servlet add-on* funciona como um *plug-in* para um servidor existente – ela adiciona o suporte a *servlet* para um servidor que não foi originalmente projetado para *servlets*. Elas têm sido escritas para muitos servidores, incluindo *Apache*, *FastTrack Server* e *Enterprise Server* da Netscape e *Internet Information Server* da Microsoft, dentre outros. Este tipo de "máquina" age como uma solução paliativa até que um futuro lançamento do servidor incorpore o suporte a *servlet*. Um *plug-in* também pode ser usado com um servidor que provê uma implementação pobre ou expirada de *servlet*.

- Embutíveis – uma "máquina" embutível é geralmente uma plataforma leve de disponibilização de *servlets* que pode ser embutida em outra aplicação. Aquela aplicação torna-se o verdadeiro servidor.

3.1.2. Propriedades dos *Servlets*

Até agora, foi retratado que os *servlets* são uma alternativa para outras tecnologias de conteúdo dinâmico para *web*, mas não foi mostrado por que usá-los. Acredita-se que *servlets* oferecem um número superior de vantagens que outras abordagens, incluindo: portabilidade, poder computacional, eficiência, tolerância, segurança, elegância, integração, extensibilidade e flexibilidade [54].

I. Portabilidade

Como *servlets* são escritos em *Java* e obedecem a uma *API* bem definida e amplamente aceita, são altamente portáveis para diversos sistemas operacionais e implementações diversas de servidor.

Servlets têm que trabalhar somente nas máquinas servidoras que se está usando para desenvolvimento e disponibilização. Assim, evitam a maioria dos erros propensos de inconsistência implementados em uma porção da linguagem Java: o *Abstract Windowing Toolkit* (AWT) [23] que forma a base das interfaces gráficas Java do usuário.

O AWT é escrito em C, assim, um programa Java com recursos AWT, embora seja independente de plataforma, acaba usando recursos que não são independentes de plataforma, isto é, os objetos da interface gráfica pertencem ao sistema operacional que se está usando. Além disso, outros problemas foram detectados, tais como, baixa velocidade em algumas plataformas e problemas de portabilidade.

Para resolver os problemas do AWT, foi desenvolvida a API *Swing*. Além disso, foram adicionados outros componentes, tais como, tabelas e árvores. Ela é escrita em Java, portanto um programa Java com recursos *Swing* é realmente

independente de plataforma (apesar de ainda usar o AWT para o tratamento dos eventos).

II. Poder Computacional

Servlets podem explorar todo o poder das APIs *core Java*: rede e acesso à URL, *multithreading*, manipulação de imagens, compressão de dados, conectividade com banco de dados, internacionalização, invocação de método remoto (RMI) [88], conectividade CORBA [20] e serialização de objetos, dentre outros.

III. Eficiência e Tolerância

A invocação de *servlets* é altamente eficiente. Uma vez que um *servlet* é carregado, ele geralmente permanece na memória do servidor como uma simples instância de um objeto. Mais tarde, o servidor invoca o *servlet* para manipular a requisição usando uma simples e leve invocação de método. Diferente do CGI, não há processo para gerar ou interpretar esta invocação, assim o *servlet* pode iniciar a manipulação da requisição quase que imediatamente. Requisições múltiplas e concorrentes são manipuladas por *threads* em separado, assim *servlets* são altamente escaláveis.

Servlets, em geral, são naturalmente objetos tolerantes. Por isso, um *servlet* permanece na memória do servidor como uma simples instância de um objeto, e automaticamente mantém seu estado e pode se manter conectado a recursos externos, tais como, conexões a banco de dados, que podem levar vários segundos para serem estabelecidas.

IV. Segurança

Como são escritos em *Java*, *servlets* herdam a segurança da tipagem forte da linguagem. Somando-se a isso, a *API Servlet* é implementada para ser segura. Enquanto a maioria dos valores em um programa *CGI* incluem um item numérico como

o número da porta do servidor e são tratados como *strings*, valores são manipulados pela *API Servlet* usando seus tipos nativos, assim o número da porta do servidor é representado por um inteiro. A coleta de lixo automática e a falta de ponteiros em *Java* significa que *servlets* geralmente são seguros de problemas de gerenciamento de memória como balanceamento de ponteiros, referências inválidas a ponteiros e vazamento de memória.

Servlets podem manipular erros seguramente, devido ao mecanismo de manipulação de exceções de *Java*. Se um *servlet* divide por zero ou executa alguma outra operação ilegal, ele dispara uma exceção que pode ser seguramente capturada e manipulada pelo servidor, que pode polidamente registrar o erro e desculpar-se ao usuário. Se uma extensão de servidor baseado em outra linguagem apresentasse o mesmo erro, ela poderia potencialmente arruinar o servidor.

Um servidor pode adicionar proteção a si próprio com *servlets* através do uso de um gerente de segurança *Java*. Um servidor pode executar seus *servlets* segundo os cuidados de um rigoroso gerente de segurança que, por exemplo, faz cumprir uma política de segurança projetada para prevenir que um *servlet* malicioso ou pobremente escrito possa danificar o sistema de arquivos do servidor, por exemplo.

V. Elegância

A elegância do código de um *servlet* é notável: claro, orientado a objetos, modular e surpreendentemente simples. Uma razão para esta simplicidade é a própria *API Servlet*, que inclui métodos e classes para manipular muitas das tarefas de rotina de desenvolvimento de *servlets*. Igualmente, operações avançadas, como manipulação de *cookies*[1] e localização de sessões são abstraídas de classes adequadas.

VI. Integração

Servlets são altamente integrados com o servidor. Esta integração permite a colaboração com o servidor em maneiras que um programa *CGI* não pode. Por exemplo, um *servlet* pode usar o servidor para traduzir caminhos de arquivos, realizar

logging, verificar autorização, executar mapeamento tipo *MIME* e, em alguns casos, adicionar usuários para o banco de dados de usuários do servidor.

VII. Extensibilidade e Flexibilidade

A *API Servlet* foi projetada para ser facilmente extensível. Assim, atualmente, ela inclui classes que são otimizadas para *servlets HTTP*. Mas, futuramente, poderá ser estendida e otimizada para outro de tipo *servlet*, tanto pela *Sun* quanto por um terceiro. É também possível que seu suporte para *servlets HTTP* possa ser melhorado.

Servlets são também bastante flexíveis. Um *servlet HTTP* pode ser usado para gerar uma página *Web* completa; pode ser adicionado a uma página estática usando uma *tag* <*SERVLET*> que é conhecida com S*erver-Side Include* [54]; e pode ser usado em colaboração com qualquer número de outros *servlets* para filtrar conteúdo em algo chamado de cadeia de *servlets*. Somando-se a isso, a *Sun* introduziu a tecnologia *JavaServer Pages* [54], que oferece uma maneira de escrever *trechos* de código *servlet* diretamente dentro de uma página estática *HTML*, usando uma sintaxe que é curiosamente similar a *Active Server Pages* (*ASP*) [30] da *Microsoft*.

3.1.3. A *API Servlet*

A *API Servlet* pode ser usada para criar *servlets HTTP*, e outro tipo de *servlet*, para qualquer finalidade. O pacote *javax.servlet* contém classes para suporte genérico e *servlets* independentes de protocolo. Estas classes são estendidas pelas classes do pacote *javax.servlet.http* para adicionar funcionalidade específica de *HTTP*. O nome do pacote de alto nível é *javax* ao invés do familiar *java*, para indicar que a *API Servlet* é uma extensão padrão.

[1] Cookies são pequenos arquivos que são enviados por um *servlet* (ou outra tecnologia semelhante) como parte de uma resposta ao cliente.

Todo *servlet* deve implementar a interface *javax.servlet.Servlet*. A maioria dos *servlets* são implementados estendendo uma das duas classes especiais: *javax.servlet.GenericServlet* ou *javax.servlet.http.HttpServlet*. Um *servlet* independente de protocolo deve ser uma subclasse de *GenericServlet*, enquanto um *servlet HTTP* deve ser uma subclasse de *HttpServlet*, que é uma subclasse de *GenericServlet* com funcionalidade específica de *HTTP*, como mostrado na figura 3.2.

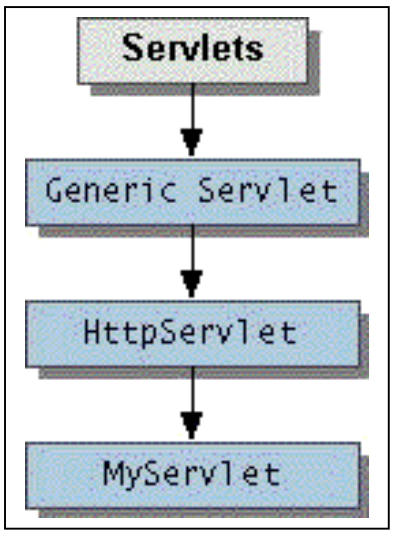

Figura 3.2 : Arquitetura de um *servlet*, baseada em [54]

Diferentemente de um programa regular em *Java*, um *servlet* não tem o método *main()*. Ao invés disso, certos métodos de um *servlet* são invocados pelo servidor no processo de manipulação de requisições. A cada momento o servidor dispara uma requisição para um *servlet*, e este invoca o método *service()* do *servlet*.

Um *servlet* genérico pode sobrescrever (*override*) seu método *service()* para manipular requisições adequadas para si. O método s*ervice()* aceita dois parâmetros: um objeto *request* e um objeto *response*. O objeto *request* diz ao *servlet* sobre a requisição, enquanto o objeto *response* é usado para retornar uma resposta. A figura 3.3 mostra como um *servlet* genérico manipula requisições.

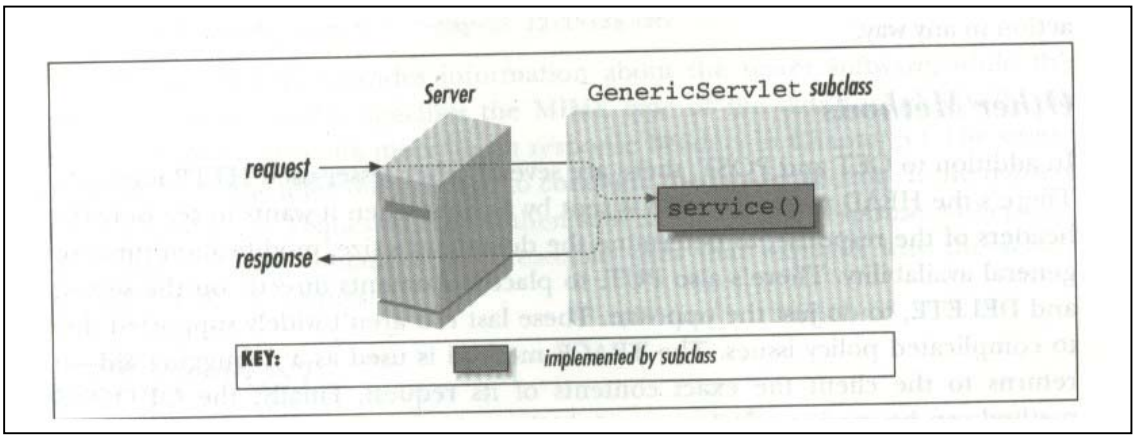

Figura 3.3 : Um *servlet* genérico manipulando uma requisição [54]

Ao contrário, um *servlet HTTP* geralmente não sobrescreve o método *service()*. Ao invés disso, sobrescreve os métodos *doGet()* para manipular requisições *GET* e *doPost()* para manipular requisições *POST*. Um *servlet HTTP* pode sobrescrever um ou ambos destes métodos, dependendo dos tipos de requisições que ele precisa manipular. O método *service()* de *HttpServlet* manipula a configuração e envio para todas os métodos *doXXX()*, por isso, ele geralmente não deveria ser sobrescrito. A figura 3.4 mostra como um *servlet HTTP* manipula requisições *GET* e *POST*[1].

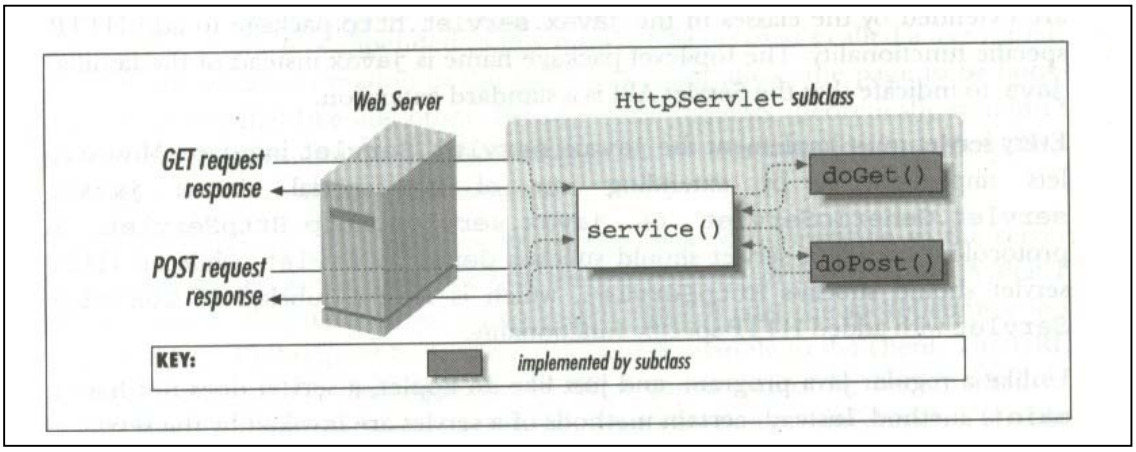

Figura 3.4 : Um *servlet HTTP* manipulando requisições *GET* e *POST* [54]

[1] Os métodos mais freqüentemente usados em uma requisição HTTP são o *GET* e *POST*. O método *GET* é projetado para obter informações, enquanto o método *POST* para postar informações.

Um *servlet HTTP* pode sobrescrever os métodos *doPut()* e *doDelete()* para manipular requisições de *PUT* e *DELETE*, respectivamente. Entretanto, *servlets HTTP* geralmente não alteram os métodos *doHead()*, *doTrace()* e *doOptions()*. Para estes, a implementação padrão é quase sempre suficiente.

O restante nos pacotes *javax.servlet* e *javax.servlet.http* são classes de suporte amplo. Por exemplo, as classes *ServletRequest* e *ServletResponse* no pacote *javax.servlet* provêm acesso para requisições e respostas de um servidor genérico, enquanto *HttpServletRequest* e *HttpServletResponse* no pacote *javax.servlet.http* provêm acesso para requisições e respostas *HTTP*. O pacote *javax.servlet.http* também contém uma classe *HttpSession* que provê a funcionalidade de rastreamento de sessão e uma classe *Cookie* que permite trabalhar e processar *cookies HTTP* rapidamente.

I. Geração de Páginas

O tipo mais básico de um *servlet HTTP* gera uma página completa *HTML*. Pode ser usado para todas as tarefas onde *CGI* é usado correntemente, tais como, processamento de formulários *HTML*, produção de relatórios a partir de um banco de dados, verificar identidades e assim por diante.

II. Escrevendo *Servlets*

A figura 3.5 mostra um *servlet HTTP* que gera uma página completa *HTML*. Este *servlet* apenas mostra o texto "*Hello World*" a cada vez que é acessado por um *browser* via *Web*.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet
{
  public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOExcepetion
  {
    res.setContextType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML>");
    out.println("<HEAD><TITLE>Hello World</TITLE></HEAD>");
    out.println("<BODY>");
    out.println("<BIG>Hello World</BIG>");
    out.println("</BODY></HTML>");
  }
}
```

Figura 3.5 : HelloWorld.java

Este *servlet* estende a classe *HttpServlet* e sobrescreve o método herdado *doGet()*. Cada vez que o servidor *web* recebe uma requisição *GET* para este *servlet*, o servidor invoca este método *doGet()*, passando para ele um objeto *HttpServletRequest* e um objeto *HttpServletResponse*.

O objeto *HttpServletRequest* representa a requisição do cliente. Este objeto fornece ao *servlet* acesso a informações sobre o cliente, os parâmetros para esta requisição, os cabeçalhos *HTTP* passados adiante com a requisição, e assim por diante.

O objeto *HttpServletResponse* representa a resposta do *servlet*. Um *servlet* pode usar este objeto para retornar dados ao cliente. Estes dados podem ser de qualquer tipo de conteúdo, contanto que o tipo deva ser especificado como parte da resposta. Um *servlet* pode também usar este objeto para ajustar os cabeçalhos da resposta *HTTP*.

O *servlet* mostrado, primeiro usa o método *setContentType()* do objeto *response* para ajustar o tipo de conteúdo de sua respostas para "*text/html*", o tipo de conteúdo padrão *MIME* para páginas *HTML*. Então, usa o método *getWriter()* para recuperar uma *PrintWriter*, a equivalente internacionalmente compatível para uma *PrintStream*. *PrintWriter* converte os caracteres *Unicode* de *Java* para um código local

específico. Para um local em inglês, ele se comporta como um *PrintStream*. Finalmente, o *servlet* usa este *PrintWriter* para enviar o *HTML* "*Hello World*" para o cliente.

III. Executando *Servlets*

Quando se desenvolve *servlets*, precisa-se de duas coisas: os arquivos das classes *API Servlet*, que são usadas para compilação, e uma "máquina" *servlet*, tal como, um servidor *web*, que é usado para disponibilização. Para obter os arquivos das classes *API Servlet*, tem-se as opções:

- Instalar o *Java Servlet Development Kit* (*JSDK*), disponível grátis em *http:/java.sun.com/products/servlet/*. *JSDK* versão 2.0 contém os arquivos das classes para a *API Servlet* 2.0, junto com seu código fonte e um simples servidor *web* que atua como uma "máquina" *servlet* para *servlets HTTP*. Ele trabalha com *JDK* 1.1 em diante. O *JSDK* é a referência de implementação da *API Servlet*, e como tal seu número de versão determina o número da versão da *API Servlet*;
- Instalar uma das muitas "máquinas" *servlet*, cada uma tipicamente empacota os arquivos das classes *API Servlet*. Existem várias "máquinas" *servlet* disponíveis.

IV. Server-Side Includes (SSI)

Até agora foi mostrado que *servlets* geram páginas *HTML* completas. Entretanto, podem também ser embutidos dentro de páginas *HTML*, em uma técnica chamada *Server-Side Include* (*SSI*) [54].

Em muitos servidores que suportam *servlets*, uma página pode ser pré-processada pelo servidor para incluir saída para *servlets* em certos pontos dentro da página.

V. Cadeia de *Servlets* e Filtros

Foi mostrado como um *servlet* individualmente pode criar conteúdo gerando uma página completa ou pode ser usado em um *Server-Side include*. *Servlets* podem também cooperar para criar conteúdo em um processo chamado cadeia de *servlets*.

Em muitos servidores que os suportam, uma requisição pode ser manipulada por uma seqüência de *servlets*. A requisição do *browser* cliente é enviada para o primeiro *servlet* na cadeia. A resposta do último *servlet* na cadeia é retornado ao *browser*. Dentro da cadeia, a saída de cada *servlet* é passada como entrada para o próximo, assim cada um deles na cadeia tem a opção de mudar ou estender o conteúdo, como mostrado na figura 3.6.

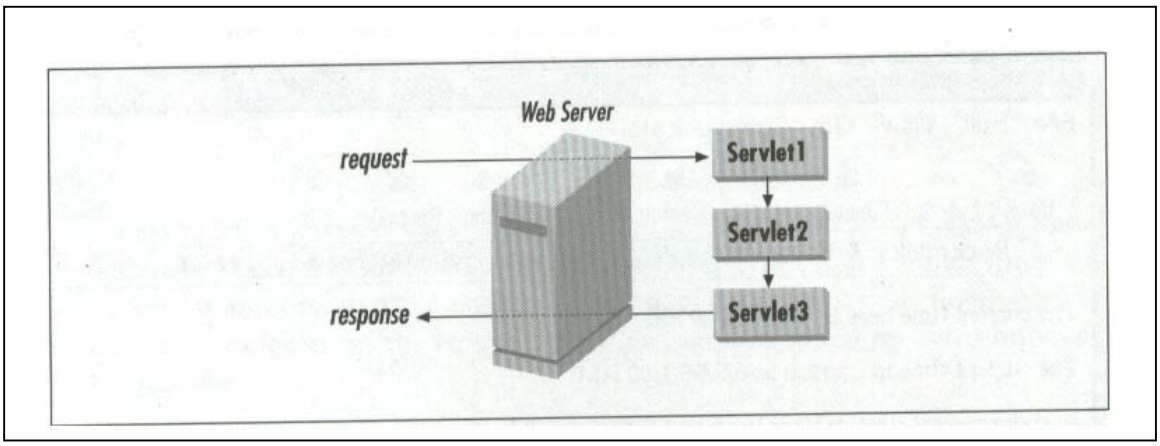

Figura 3.6 : Cadeia de *servlets* [54]

Há duas maneiras de engatilhar uma cadeia de *servlets* para uma requisição recebida. Primeiro, pode-se dizer ao servidor que certas *URLs* devem ser manipuladas por uma cadeia especificada explicitamente. Alternativamente, pode-se dizer ao servidor para enviar todas as saídas de um tipo de conteúdo em particular através de um *servlet* específico antes de ser retornado para o cliente, efetivamente criando uma cadeia em tempo de execução. Quando um *servlet* converte um tipo de conteúdo em outro, a técnica é chamada filtragem.

Cadeia de *servlets* podem mudar a forma de pensar sobre a criação de conteúdo para a *web*. Abaixo, alguns exemplos de utilização desta técnica:

- Mudar rapidamente a aparência de uma página, de um grupo de páginas ou um tipo de conteúdo. Por exemplo, pode-se melhorar um *site* suprimindo as *tags* <*BLINK*> das páginas de um servidor. Pode-se informar para aqueles que não entendem Inglês para traduzir dinamicamente o texto de suas páginas para a linguagem de leitura do cliente. Pode-se suprimir certas palavras para que não sejam lidas, como palavras indecentes ou desconhecidas, como o nome não-lançado de um projeto secreto. Pode-se também suprimir páginas inteiras em que estas palavras aparecem e ressaltar certas palavras em um *site*;
- Carregar um *kernel* de conteúdo e mostrá-lo em formatos especiais. Por exemplo, pode-se embutir *tags* comuns em uma página e ter um *servlet* para substituí-las com conteúdo *HTML*. Seja uma *tag* <*SQL*> que consulta conteúdos que são executados em um banco de dados e cujos resultados são colocados em uma tabela *HTML*. Isto é, de fato, similar a como o *Java Web Server* suporta a *tag* <*SERVLET*>;
- Suporte de tipos de dados estranhos. Por exemplo, pode-se tratar tipos não suportados de imagens com um filtro que converte tipos não-padrão de imagens para os formatos *GIF* ou *JPEG*.

VI. *JavaServer Pages* (*JSP*)

A *Sun* anunciou uma nova maneira de usar *servlets*, chamada *Java Server Pages* (comumente, mas não oficialmente, referenciada como *JSP*) [54]. A funcionalidade e a sintaxe têm extraordinária semelhança com *Active Server Pages* (*ASP*) [30].

JSP opera em muitas maneiras como S*erver-Side Includes*. A diferença principal é que ao invés de embutir uma *tag* <*SERVLET*> na página *HTML*, *JSP* embute trechos de código *servlet*. É uma tentativa da *Sun* para separar conteúdo de apresentação, mais conveniente que *server-side includes* para páginas que têm pedaços de conteúdo dinâmico misturado com conteúdo estático em vários diferentes

lugares. Assim como *server-side includes* e cadeia de *servlets*, *JSP* não requer nenhuma mudança na *API Servlet*. Mas requer suporte especial no servidor *Web*.

JSP permite a inserção direta de código *servlet* em um arquivo estático *HTML* de uma outra maneira. Cada bloco de código *servlet* (chamado de *scriptlet*) é circundado no início pela *tag* "<%" e no fim pela *tag* "%>" (uma tecnologia mais recentemente chamada *Page Compilation*, usa as *tags* <*JAVA*> e </*JAVA*> e uma sintaxe interna diferente). Por conveniência, um *scriptlet* pode usar quatro variáveis pré-definidas:

- *request* : o *servlet* requisita um objeto *HttpServletRequest*;
- *response* : o *servlet* responde um objeto *HttpServletResponse*;
- *out* : o escritor para a saída, um objeto *PrintWriter;*
- *in* : o leitor para a entrada, um objeto *BufferedReader*.

A figura 3.7 mostra uma página *JSP* que diz "*Hello*" de uma maneira similar ao exemplo da figura 3.5, embora com menos código. Ela faz uso das variáveis pré-definidas *request* e *out*.

```
<HTML><HEAD><TITLE>Hello</TITLE></HEAD>
<BODY><H1>
<%
   if (request.getParameter("name") == null) {
      out.println("Hello World");
   }
   else {
      out.println("Hello, " + request.getParameter("name"));
   }
%>
</H1></BODY></HTML>
```

Figura 3.7 : hello1.jsp

O serviço prestado pelo JSP é mostrado na figura 3.8 e descrito a seguir. O servidor automaticamente cria, compila, carrega e executa um *servlet* especial para gerar o conteúdo da página. Pode-se pensar que este *servlet* especial funciona como um *servlet background*. As porções estáticas da página *HTML* são geradas pelo *servlet background* usando o equivalente às chamadas *out.println()*, enquanto as porções dinâmicas são incluídas diretamente.

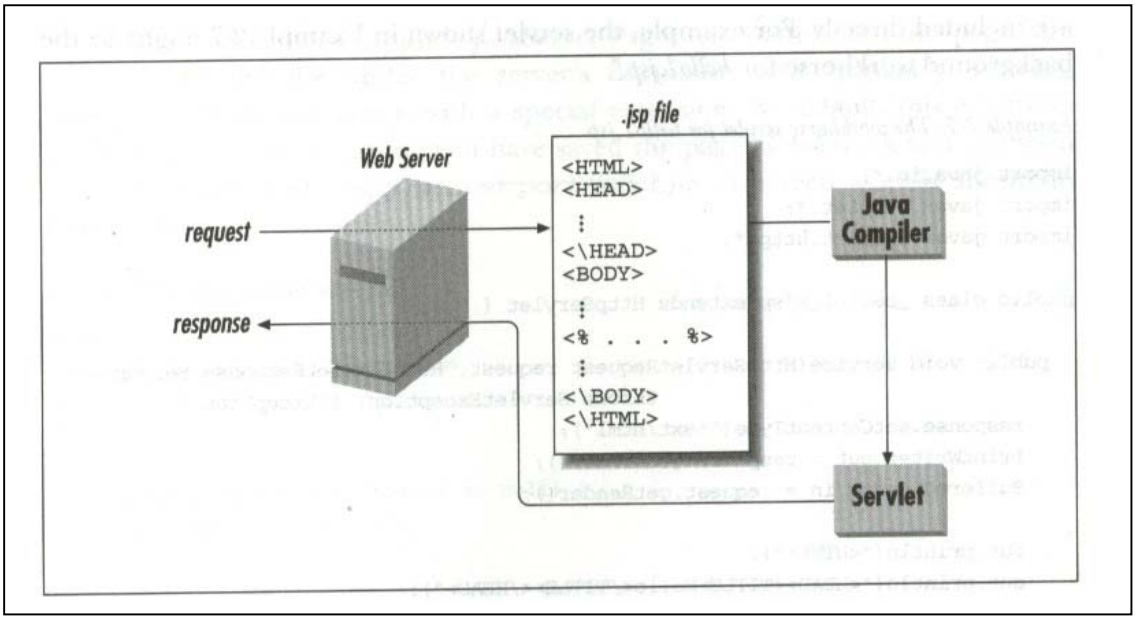

Figura 3.8 : Gerando *JavaServer Pages* [54]

Por exemplo, o *servlet* mostrado na figura 3.9 poderia ser o *background* para o *hello1.jsp*. O verdadeiro código fonte de um *servlet* para uma página *JSP* pode estar localizado nos diretórios sob a raiz do servidor. Depois de tudo, o servidor precisa salvar o código fonte *Java* em algum lugar antes de compilá-lo. Se encontrado o verdadeiro código fonte do *servlet*, é provável que seja muito mais complicado e confuso que o que é mostrado a seguir.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class _hello1_xjsp extends HttpServlet {
   public void service (HttpServletRequest request,
   HttpServletResponse response) throws ServletException, IOExcepetion {
      response.setContextType("text/html");
      PrintWriter out = response.getWriter();
      BufferedReader in = request.getReader();
      out.println("<HTML><HEAD><TITLE>Hello</TITLE></HEAD>");
      out.println("<BODY><H1>");
      if (request.getParameter("name") == null) {
         out.println("Hello World");
      }
      else {
         out.println("Hello, " + request.getParameter("name"));
      }
      out.println("</H1></BODY></HTML>");
   }
}
```

Figura 3.9 : Provável Código fonte do *servlet background* de *hello1.jsp*

A primeira vez que se acessa uma página *JSP*, pode-se notar que ela leva um pequeno tempo para responder. Este é o tempo necessário para o servidor criar e compilar o *servlet background*. Posteriormente, requisições deveriam ser tão rápidas em qualquer momento, porque o servidor pode reusar o *servlet*. A única exceção é quando o arquivo *.jsp* muda, neste caso o servidor anuncia e recompila um novo *servlet background*. Se a qualquer momento houver um erro na compilação, pode-se esperar o servidor de alguma maneira informar sobre o problema, geralmente na página retornada ao cliente.

VII. *JavaBeans*

Uma das mais interessantes e poderosas maneiras de usar *JavaServer Pages* está na cooperação com componentes *JavaBeans*, que são classes *Java* reusáveis cujos métodos e variáveis seguem convenções específicas para lhes dar habilidades adicionais. Elas podem ser embutidas diretamente em uma página *JSP* usando *tags* <*BEAN*>. Um componente *JavaBean* pode realizar uma tarefa bem definida (executar consultas a banco de dados, conectar-se a um servidor de correio eletrônico ou manter informações sobre o cliente) e torna seu resultado uma informação disponível para a página *JSP* através do simples acesso aos métodos.

A diferença entre um componente *JavaBean* embutido em uma página *JSP* e uma terceira classe usada pelo *servlet* gerado é que o servidor *web* pode dar a *JavaBeans* tratamento especial. Por exemplo, um servidor pode automaticamente configurar as propriedades do *bean* (variáveis de instância) usando os valores dos parâmetros na requisição do cliente. Em outras palavras, se a requisição inclui o parâmetro NOME e o servidor detecta através da introspecção (uma técnica onde os métodos e variáveis de uma classe *Java* podem ser determinados pelo programa em tempo de execução) que o *bean* tem uma propriedade NOME e um método *setName*(*String* nome), o servidor pode automaticamente chamar *setName*() com o valor do parâmetro NOME. Não há necessidade para *getParameter*().

Um *bean* pode também ter seu escopo gerenciado automaticamente pelo servidor. Um *bean* pode ser designado para uma requisição específica (onde é usado

uma vez e destruído ou reciclado) ou para uma sessão do cliente (onde é automaticamente tornado disponível cada vez que o mesmo cliente se reconecta).

Um *bean* pode igualmente ser implementado como um *servlet*. Se o servidor detecta que o referido *bean* implementa a interface *javax.servlet.Servlet* (ou diretamente ou estendendo *GenericServlet* ou *HttpServlet*), ele chamará o método *service()* do *bean* uma vez para cada requisição e o método *init()* quando o *bean* é inicialmente criado.

VIII. Ciclo de vida do *servlet*

O ciclo de vida é um dos aspectos mais empolgantes dos *servlets*. Este ciclo de vida é uma poderosa combinação dos ciclos de vida usados na programação *CGI* e da programação de baixo nível de *NSAPI* [115] e *ISAPI* [115], *APIs* da *Netscape* e *Microsoft*, respectivamente.

O ciclo de vida do *servlet* permite que "máquinas" *servlet* enfoquem a performance e problemas de recurso de *CGI* e os aspectos de segurança da programação de baixo nível da *API* do servidor. Uma "máquina" *servlet* pode executar todos seus *servlets* em uma simples *JVM*. Como estão na mesma *JVM*, *servlets* podem eficientemente compartilhar dados entre si, no entanto, eles são impedidos pela linguagem *Java* de acessar dados privados de um outro. *Servlets* podem ter permissão para persistir entre requisições como instâncias de objetos.

O ciclo de vida do *servlet* é altamente flexível. Servidores têm significante liberdade de ação em como escolher suporte aos *servlets*. A única regra permanente e rígida é que uma "máquina" *servlet* deve se ajustar ao seguinte ciclo de vida:

- Criar e inicializar o *servlet*;
- Manipular zero ou mais chamadas de serviço dos clientes;
- Destruir o *servlet* e depois coletá-lo.

É perfeitamente normal para um *servlet* ser carregado, criado e instanciado em sua própria *JVM*, somente para ser destruído e coletado sem manipular qualquer requisição de clientes ou depois de manipular apenas uma requisição. Qualquer

"máquina" *servlet* que faz disto um hábito, entretanto, provavelmente não permanecerá muito tempo no mercado. Tal ciclo de vida é mostrado na figura 3.10.

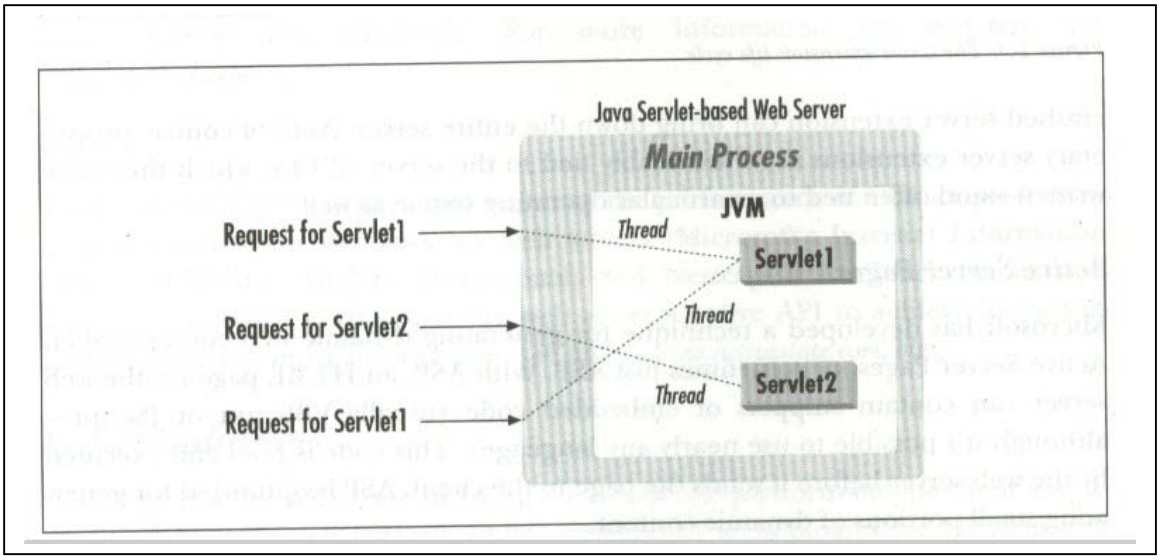

Figura 3.10 : Ciclo de vida do Servlet [54]

IX. Uma Simples Máquina Virtual *Java*

A *JVM* pode freqüentemente ser embutida dentro do processo servidor. Isto melhora a performance, pois um *servlet* torna-se, em um sentido, apenas outra extensão da *API* do servidor de baixo nível. Assim, um servidor pode invocar um *servlet* com uma leve mudança de contexto e pode prover informação sobre requisições através de invocação direta de métodos.

Um servidor *web* multiprocessado (que roda vários processos para manipular requisições) não tem realmente a escolha de embutir uma *JVM* diretamente em seu processo pois não há um processo. Este tipo de servidor geralmente roda uma *JVM* externa que pode compartilhar seus processos. Com esta abordagem, cada acesso ao *servlet* invoca uma pesada mudança de contexto remanescente de *FastCGI*. Todos os *servlets*, entretanto, ainda compartilham o mesmo processo externo.

Felizmente, da perspectiva do *servlet*, a implementação do servidor realmente não é a questão, pois o servidor sempre se comporta da mesma maneira.

X. Persistência da Instância

Foi dito que *servlets* persistem entre requisições como instâncias de objetos. Em outras palavras, no tempo em que o código para o *servlet* é carregado, o servidor cria uma única instância da classe. Esta única instância manipula toda requisição feita ao *servlet*. Isto provê melhor performance de três maneiras:

- Utiliza pequeno espaço na memória;
- Elimina o *overhead* da criação do objeto que seria necessário para criar um novo objeto *servlet*. Um *servlet* já pode ser carregado em uma máquina virtual quando uma requisição chega, levando-o a iniciar a execução rapidamente;
- Ativa persistência. Um *servlet* pode já ter sido carregado e é provável que ele seja necessário durante a manipulação de uma requisição. Por exemplo, uma conexão a um banco de dados pode ser aberta uma vez e usada repetidamente depois disso. Ainda, outro *servlet* pode escolher fazer *cache* de páginas inteiras de saída para economizar tempo se ele recebe a mesma requisição novamente.

3.2. JDBC (*Java DataBase Connectivity*)

Em meados de 1995, a Sun Microsystems, tão logo foi feito o lançamento das primeiras versões da linguagem Java, passou a investigar o potencial da linguagem para o acesso a bases de dados.

Neste ponto, ao invés do desenvolvimento de um gerenciador de bases de dados baseado em Java, optou-se pelo desenvolvimento de uma biblioteca de programação que permitisse o acesso a bases de dados através da linguagem SQL (*Structured Query Language*), já bastante difundida na época. Tal biblioteca, denominada JDBC [55], se tornou um dos pontos de maior investimento da Sun Microsystems.

A biblioteca JDBC é uma API que permite aos programadores o desenvolvimento de programas pouco complexos, em termos da codificação Java, para executar instruções SQL em ambientes distribuídos [57]. A combinação Java - JDBC

permite o desenvolvimento de aplicações multiplataforma, visto que as bases de dados podem estar distribuídas em servidores remotos que utilizam diferentes ambientes operacionais, onde as plataformas necessitam possuir uma JVM. As versões mais novas da linguagem Java trazem esta biblioteca embutida no conjunto de bibliotecas da linguagem, não sendo mais necessária a sua instalação independente.

É difícil encontrar hoje um *web site* profissional que não tenha algum tipo de conectividade com banco de dados. Mas a interação *web*-banco de dados tem um preço: *web sites* com uso de banco de dados podem ser difíceis de desenvolver e podem freqüentemente resultar em graves perdas de performance.

3.2.1. *Drivers* JDBC

A *API JDBC*, encontrada no pacote *java.sql*, contém somente algumas classes concretas. Muitas das *APIs* são distribuídas como classes neutras de interface do banco de dados que especificam comportamento sem prover qualquer implementação. As implementações de fato são providas por terceiros.

Um sistema individual de banco de dados é acessado via um *driver JDBC* específico que implementa a interface *java.sql.Driver*. Existem *drivers* para quase todos os sistemas (SGDB Relacionais) conhecidos, embora poucos são disponíveis gratuitamente. A *Sun* empacota um *driver* gratuito da ponte *JDBC-ODBC* com o *JDK* para permitir acesso ao padrão *ODBC* de fontes de dados, tais como, bancos de dados *Microsoft Access*.

Os tipos de *driver* JDBC são:

1. Pontes JDBC-ODBC que provêem acessos JDBC através de *drivers* ODBC. Esses *drivers* somente são usados no caso de necessidade de acesso à fonte de dados que não possuem *drivers* JDBC do tipo 2, 3 e 4;
2. "*Native API*", esse tipo de *driver* converte chamadas JDBC em chamadas para uma API cliente dos fornecedores de bancos de dados e essa API faz a chamada para o banco de dados através do protocolo proprietário do SGBD;

3. "*Net pure Java driver*", esse tipo de *driver* converte chamadas JDBC em chamadas para um SGBD padrão que são traduzidas por um servidor para as chamadas nativas aos diversos fornecedores de bancos de dados;

4. "*Native-protocol pure Java driver*", converte chamadas JDBC diretamente nos protocolos nativos de cada fornecedor de banco de dados.

De modo geral, a arquitetura de software para uso do JDBC é simples. Inicialmente, uma aplicação faz um pedido a JDBC para que seja criada uma conexão, via um *driver JDBC*, a uma base de dados. A partir do estabelecimento da conexão, um serviço de passagem de mensagens permite à aplicação o envio de *queries* (instruções SQL) ao banco de dados, que por sua vez executa os pedidos da aplicação e promove o envio dos resultados. A figura 3.11 ilustra a arquitetura de software utilizada para comunicação via JDBC.

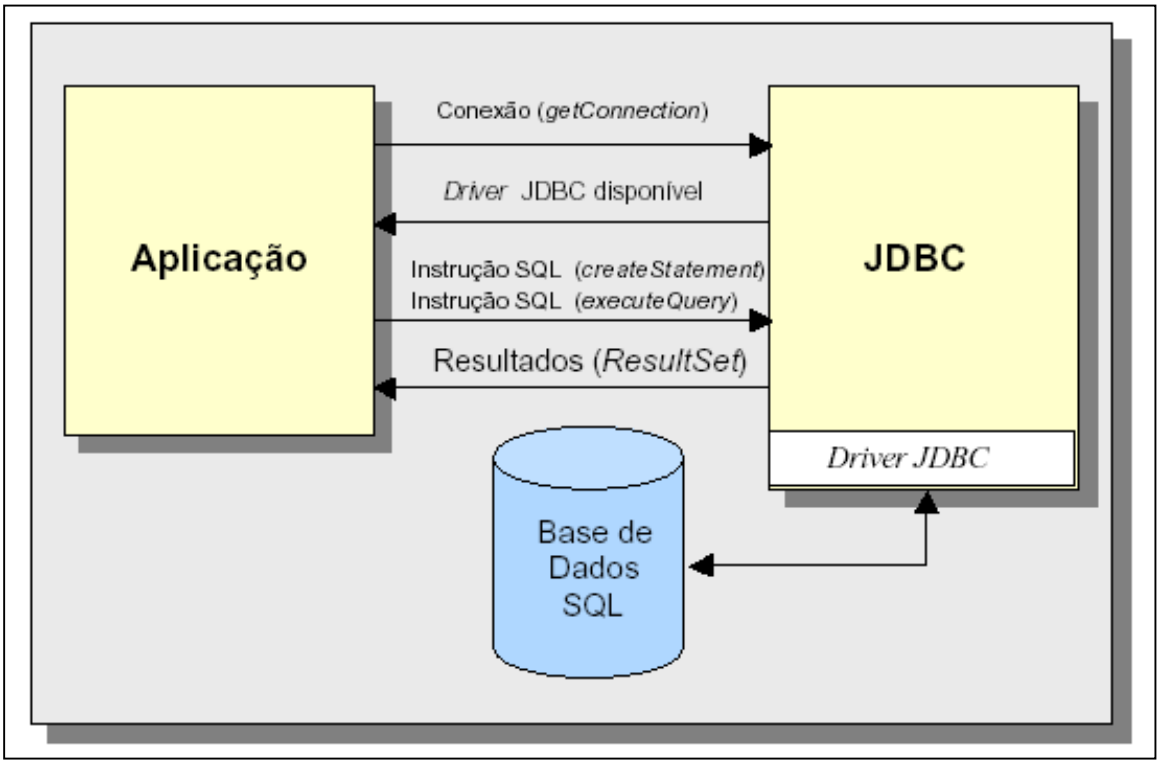

Figura 3.11 - Arquitetura do ambiente de passagem de mensagem do JDBC, baseada em [51]

3.2.2. Conectividade com Banco de Dados

Nesta seção, serão descritos os procedimentos para conexão com o banco de dados, processamento de consultas e alterações, assim como o tratamento de exceções.

I. Obtendo uma conexão

O primeiro passo para usar um *driver JDBC* para obter uma conexão ao banco de dados envolve a carga da classe do *driver* específico dentro da JVM da aplicação. Isto torna o *driver* disponível mais tarde, quando precisa-se dele para abrir a conexão. Uma maneira fácil para carregar a classe do *driver* é usar o método *Class.forName()*:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Quando o *driver* é carregado para a memória, ele próprio se registra com a classe *java.sql.DriverManager* como um *driver* disponível para banco de dados.

O próximo passo é pedir à classe *DriverManager* para abrir uma conexão para um dado banco de dados, onde o banco de dados é especificado por uma *URL* formatada especialmente. O método usado para abrir a conexão é *DriverManager.getConnection()*. Ele retorna uma classe que implementa a interface *java.sql.Connection*.

```
Connection con = DriverManager.getConnection("jdbc:odbc:algumbd", "usuário",
"senha");
```

Uma *URL JDBC* identifica um banco de dados individual em um modo específico do *driver*. Diferentes *drivers* podem precisar de informações diferentes na

URL para especificar o banco de dados do *host*. *URLs JDBC* geralmente iniciam com *jdbc:sub-protocolo:sub-nome*. Por exemplo, o *driver Oracle JDBC-Thin* usa uma *URL* na forma: *jdbc:oracle:thin:@dbhost:porta:sid.*

Durante a chamada para *getConnection()*, o objeto *DriverManager* solicita a cada *driver* registrado, o reconhecimento da *URL*. Se um *driver* a reconhece, o gerenciador de *driver* usa aquele *driver* para criar o objeto *Connection*.

II. Executando Consultas *SQL*

Para realmente usar um banco de dados, precisa-se ter alguma forma de executar consultas. A forma mais simples para executar uma consulta é usar a classe *java.sql.Statement*. Objetos *Statement* nunca são instanciados diretamente; ao invés disso, um programa chama o método *createStatement()* de *Connection* para obter um novo objeto *Statement*:

```
Statement stmt = con.createStatement();
```

Uma consulta que retorna dados pode ser executada usando o método *executeQuery()* de *Statement*. Este método executa o comando e retorna um *java.sql.ResultSet* que encapsula os dados recuperados:

```
ResultSet rs = stmt.executeQuery("SELECT * FROM EMPREGADOS");
```

Pode-se imaginar um objeto *ResultSet* como uma representação do resultado retornado da consulta, dado uma linha por vez. Usa-se o método *next()* de *ResultSet* para mover de uma linha para outra. A interface *ResultSet* também dispõe de um grande número de métodos projetados para recuperação de dados de linha corrente. Os métodos *getString()* e *getObject()* estão entre os mais freqüentemente usados para recuperação dos valores das colunas:

```
while (rs.next())
{
   String event = rs.getString("event");
   Object count = rs.getObject("count");
}
```

É importante ressaltar que o *ResultSet* é ligado ao seu "pai" *Statement*. Portanto, se um *Statement* é fechado ou usado para executar outra consulta, quaisquer dados relacionados aos objetos *ResultSet* são fechados automaticamente.

A figura 3.12 mostra um simples *servlet* que usa o *driver Oracle JDBC* para realizar uma simples consulta, imprimindo nomes e números de telefones de todos os empregados listados em uma tabela no banco de dados. É assumido que o banco de dados contém uma tabela chamada *EMPREGADOS*, com ao menos dois campos, *NOME* e *TELEFONE*.

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Agenda extends HttpServlet
{
  public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
  {
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    try
    {
// Carrega e portanto registra o Driver Oracle
      Class.forName("oracle.jdbc.driver.OracleDriver");
// Obtém a conexão ao banco de dados
      con = DriverManager.getConnection("jdbc:oracle:thin:dbhost:1528:ORCL",
"user", "passwd");
// Cria um objeto Statement
      stmt = con.createStatement();
// Executa uma consulta SQL, obtém um ResultSet
      rs = stmt.executeQuery("SELECT NOME, TELEFONE FROM EMPREGADOS");
// Mostra o resultado como uma lista
      out.println("<HTML><HEAD><TITLE>Agenda</TITLE></HEAD>");
      out.println("<BODY>");
      out.println("<UL>");
      while(rs.next())
      {
        out.println("<LI>"    +    rs.getString("nome")    +    "    "    +
rs.getString("telefone"));
      }
      out.println("</UL>");
      out.println("</BODY></HTML>");
    }
    catch(ClassNotFoundException e)
    {
      out.println("Não pôde carregar driver do BD: " + e.getMessage());
    }
    catch(SQLException e)
    {
      out.println("SQLException: " + e.getMessage());
    }
    finally
    {
// Sempre fecha a conexão do banco de dados.
      try
      {
        if (con != null) con.close();
      }
      catch (SQLException ignored) { }
    }
  }
}
```

Figura 3.12 : Agenda.java

Tudo o que a classe Agenda faz é conectar-se ao banco de dados, executar uma consulta que recupera todos os nomes e números de telefone da tabela de empregados e mostra a lista para o usuário.

III. Manipulando exceções *SQL*

A classe Agenda mostra em seu código um bloco *try/catch*. Este bloco abrange duas exceções: *ClassNotFoundException* e *SQLException*. A primeira é disparada pelo método *Class.forName()* quando a classe do *driver JDBC* não puder ser carregada. A segunda é disparada por qualquer método *JDBC* que tem algum problema. Objetos *SQLException* são como qualquer outro tipo de exceção, com a característica adicional que eles podem se encadeiar. A classe *SQLException* define um método extra, *getNextException()*, permitindo que a exceção encapsule objetos *Exception* adicionais. Esta característica não foi contemplada no exemplo da figura 3.12, mas na figura 3.13, é mostrado como usá-la:

```
catch (SQLException e)
{
   out.println(e.getMessage());
   while ((e  = e.getNextException()) != null)
   {
      out.println(e.getMessage());
   }
}
```

Figura 3.13: Encapsulando objetos *Exception* adicionais

Este código mostra a mensagem da primeira exceção e então executa um laço até que todas as exceções restantes, mostrem suas mensagens de erro associadas a cada uma delas. Na prática, a primeira exceção geralmente incluirá a informação mais relevante.

IV. Alterando um banco de dados

A maioria dos *sites* na *web* com acesso a banco de dados precisa fazer mais que apenas executar consultas. Quando um cliente submete uma ordem ou provê algum tipo de informação, os dados precisam fazer parte do banco de dados. Quando se está executando um comando *SQL UPDATE*, *INSERT* ou *DELETE* e não se espera um *ResultSet*, pode-se usar o método *executeUpdate()* de *Statement*. Ele retorna uma quantidade que indica o número de linhas modificadas pelo comando, conforme o exemplo abaixo.

```
int count = stmt.executeUpdate("DELETE FROM EMPREGADOS WHERE NOME = 'JOSÉ'");
```

Pode-se executar um comando *SQL*, utilizando-se o método genérico *execute()* de *Statement*. Ele retorna um *boolean*, cujo valor é verdadeiro se o comando *SQL* produziu um ou mais objetos *ResultSet* ou falso se ele resultou em uma quantidade de alterações. Os métodos *getResultSet* e *getUpdateCount* de *Statement* provêm acesso aos resultados do método *execute()*, como mostra o exemplo da figura 3.14.

```
if (stmt.execute(sql))
{
   // Há um ResultSet como retorno
   ResultSet rs = stmt.getResultSet();
.
.
}
else
{
   // Há um número correspondente a quantidade de alterações como retorno
   int count = stmt.getUpdateCount();
.
.
}
```

Figura 3.14: Métodos *getResultSet* e *getUpdateCount*

3.3. Conclusão

Constata-se que a linguagem de programação Java possui: orientação a objetos, *multithreading*, portabilidade, segurança, suporte à rede e facilidades para construção de interfaces e tratamento de dados multimídia [51], tornando-se, portanto, a escolhida para o protótipo desenvolvido nesta dissertação.

Foi visto que os *servlets* se constituem em aplicativos escritos em *Java* que podem ser disponibilizados em diversos tipos de servidores para expandir suas funcionalidades. Assim, o uso desta técnica no desenvolvimento de aplicações para trabalhar com diversos tipos de protocolos de comunicação, tem demonstrado na prática que *servlets* podem rodar em qualquer plataforma sem a necessidade de serem reescritos.

Servlets permitem:

- uma rápida, poderosa e portável substituição dos *scripts CGI*;
- a execução dentro de um espaço no processo do servidor *web* e a persistência entre invocações, que lhes fornecem benefícios de alta performance em relação aos programas *CGI;*
- o acesso completo às várias *APIs Java* e classes de terceiros, tornando-os ideais para uso em comunicação com *applets*, banco de dados e servidores *RMI*;
- a portabilidade para diversos sistemas operacionais e servidores. Com *servlets* pode-se “escrever uma vez, trabalhar em todo lugar”.

Também constata-se que com a API JDBC, pode-se construir aplicações que podem acessar várias fontes de dados heterogêneas, podendo executar essa aplicação em qualquer plataforma que possua uma máquina virtual Java.

No módulo de integração de bancos de dados implementado, tem-se um *servlet* responsável por receber as solicitações das consultas dos estudantes, acessar, em paralelo (devido a sua característica de *multi-threading*), os bancos de dados locais dos demais estudantes, construir e enviar a resposta ao estudante solicitante.

Como um s*ervlet* roda no servidor *Web*, pode-se evitar problemas relativos à compatibilidade de interfaces gráficas entre os diversos clientes. Tal característica, também foi razão importante para escolha da citada tecnologia.

Capítulo 4

Objetos distribuídos

Neste capítulo, serão estudados alguns padrões para sistemas distribuídos com suporte a criação de objetos distribuídos: DCOM da Microsoft, CORBA do OMG (*Object Management Group*) e Java/RMI. Também, será feita uma análise comparativa entre estes padrões estudados.

Capítulo 4

Objetos distribuídos

Devido a fatores como o barateamento do processamento, a maturidade de tecnologias utilizadas em redes de computadores, necessidade de maior flexibilidade e escalabilidade na infra-estrutura computacional, bem como a crescente necessidade de comunicação entre organizações, a atual infra-estrutura de informação predominante é composta por nodos processadores heterogêneos interconectados para a troca de informação.

Neste contexto, as aplicações conseqüentemente passam a ser compostas por módulos comunicantes dispostos em máquinas remotas. A interação entre módulos distribuídos pode acontecer mediante a utilização de vários tipos de mecanismos. Alguns deles exigem que o desenvolvedor de aplicações trate problemas de localização e comunicação entre os módulos. As duas áreas de pesquisa envolvidas são: orientação a objetos e sistemas distribuídos.

O paradigma de orientação a objetos tem causado mudanças significativas na análise, projeto e desenvolvimento de sistemas. O desenvolvimento de software orientado a objetos tem se acentuado com a promessa de aumento de modularidade, reusabilidade e conseqüente facilidade de construção. Apesar de suas vantagens, existem áreas ainda não totalmente exploradas, especialmente com sistemas legados [113].

O aumento da capacidade de processamento das estações de trabalho acelerou o interesse em adaptar as aplicações de grande porte para essas estações, distribuindo funcionalidades entre esses nodos. Através dos conceitos definidos pela pesquisa em sistemas distribuídos migra-se de uma infra-estrutura de plataformas homogêneas para outra geograficamente dispersa e composta por máquinas heterogêneas [113]. Este fato contribuiu para uma crescente necessidade de desenvolvimento de software para ambientes distribuídos.

A partir destas justificativas, testemunhou-se nos últimos anos a fusão da orientação a objetos com sistemas distribuídos. A fusão permitiu que objetos fossem

utilizados em sistemas distribuídos para representarem unidades de distribuição, mobilidade e comunicação.

Objetos Distribuídos são componentes de software criados com os conceitos da orientação a objetos [33], podendo estar distribuídos através de uma rede heterogênea, mas visto pelas aplicações como se fossem componentes locais, fornecendo com isso vários tipos de transparências desejadas em sistemas distribuídos. Alguns exemplos de transparências encontradas são:

- Transparência de acesso: permite comunicação de objetos escritos em diferentes linguagens de programação, executados em plataformas heterogêneas;
- Transparência de localização: torna transparente ao desenvolvedor os detalhes de localização de objetos (endereços de rede, portas);
- Transparência de relocação: torna transparente ao desenvolvedor o problema da atualização de referências quando objetos utilizados são relocados (mudam de local onde estão executando);
- Transparência de migração: torna transparente ao desenvolvedor o fato de que o local de execução do objeto pode ser mudado;
- Transparência de persistência: esconde aspectos de ativação e desativação de objetos, ou seja, os objetos são inicializados sob demanda e desativados quando não são mais necessários, porém o seu estado é mantido de forma persistente;
- Transparência de replicação: torna transparente a existência de um grupo de objetos comportamentalmente compatíveis. Objetos podem ser replicados para manter níveis de disponibilidade e desempenho de seu serviço.

A utilização da orientação a objetos agrega inúmeras características ao processo de desenvolvimento dos objetos distribuídos, das quais merecem destaque o encapsulamento, comportamento, polimorfismo, herança e identidade de objetos. Já os sistemas distribuídos agregam novas funcionalidades como interoperabilidade, escalabilidade, disponibilidade e desempenho.

Objetos distribuídos são componentes de software acessados por clientes remotos através da invocação de operações desses objetos. Entretanto, seus clientes não precisam saber onde os objetos distribuídos foram criados, onde estão sendo executados, ou em que plataforma podem ser executados. Outra característica marcante dos objetos distribuídos é o encapsulamento. Os clientes solicitam serviços

através das operações descritas nas interfaces mas não sabem como os objetos distribuídos implementam ou executam os serviços solicitados.

Segundo Ferraz [39], a tecnologia de objetos distribuídos pode trazer uma série de benefícios às aplicações através da Internet. Dentre eles, destacam-se:

- Transparência: esconde a natureza distribuída dos recursos da aplicação, permitindo que o desenvolvimento e manipulação dos objetos seja planejado e utilizado sem a preocupação com localização, protocolos ou plataformas;
- Balanceamento de carga: um serviço pode estar replicado em varias máquinas, de tal modo que aumenta a disponibilidade do sistema e permite um balanceamento de uso de recursos, permitindo um melhor desempenho;
- Processamento Distribuído: um serviço da aplicação também pode ser realizado por um conjunto de servidores que trabalham em cooperação. Essa distribuição das tarefas é mais uma maneira de otimizar o desempenho do sistema;
- Independência de plataforma e Interoperabilidade: os clientes e servidores de uma aplicação podem estar distribuídos em plataformas de software e hardware diferentes. Dessa forma, os sistemas de objetos distribuídos buscam ser independentes de plataforma sendo capazes de trocar informações mesmo que sejam escritos em linguagens diferentes ou estejam sendo executados em sistemas operacionais e arquiteturas diversas;
- Escalabilidade: a replicação de dados e serviços dos sistemas, possível através da utilização de objetos distribuídos, permite uma maior escalabilidade, ou seja, as aplicações ganham maior poder de atender a mudanças de demanda;
- Tolerância a falhas: a utilização de objetos distribuídos permite a replicação e a redundância de determinados serviços em servidores diferentes. Esta característica pode ser explorada para assegurar que tarefas essenciais sejam alocadas para outro servidor quando ocorrer falha no sistema;
- Extensibilidade e Flexibilidade: extensibilidade é a capacidade de acoplar novas funcionalidades a uma aplicação, de forma que não seja necessário realizar grandes mudanças em outras partes dela. Com isso amplia-se a flexibilidade das aplicações, permitindo que estas sejam alteradas sem maiores problemas.

A seguir, serão estudados alguns padrões para sistemas distribuídos com suporte a criação de objetos distribuídos: DCOM [71] da Microsoft, CORBA [20] do OMG (*Object Management Group*) e Java/RMI [88].

4.1. DCOM (*Distributed Component Object Model*)

O DCOM [71] foi criado pela Microsoft como uma extensão do COM (*Component Object Model*) [21], com o objetivo de suportar a comunicação entre objetos em diferentes computadores.

O DCOM faz uso dos investimentos realizados no COM, tais como aplicações, componentes, ferramentas e conhecimento para ingressar no mundo da computação distribuída baseado nas padronizações existentes. Desta maneira, o DCOM é responsável em implementar os detalhes de baixo nível dos protocolos de rede, fazendo com que os desenvolvedores possam se preocupar com tarefas mais importantes, tal como fornecer melhores soluções para seus clientes.

O COM define como deve ser a interação entre os objetos e seus clientes, sem a intermediação de qualquer componente do sistema.

Em situações de componentes que estejam sendo executados em diferentes processos, existe a necessidade de ser estabelecida uma comunicação entre processos, a qual é feita pelo sistema operacional. O COM fornece esta comunicação através da biblioteca de execução COM/DCOM que estabelece o *link* entre o cliente e o componente.

Quando o processo cliente e o componente estão localizados remotamente, o DCOM utiliza um protocolo de rede para estabelecer a comunicação entre eles de forma transparente. O *proxy* e o *stub* fornecem serviços orientados a objetos ao cliente e ao componente, respectivamente, utilizando uma chamada de procedimento remoto (*Remote Procedure Call* [74]) e o provedor de segurança para gerar pacotes de redes padrões que sejam compatíveis com o protocolo padrão do DCOM.

A forma de comunicação estabelecida no DCOM permite a localização dos componentes transparentemente. Desta maneira, não é necessário saber o local onde eles estão localizados. A forma como são feitas as chamadas é a mesma tanto para os

objetos localizados no mesmo processo, quanto para os que estão remotamente distribuídos.

4.1.1. Definição dos Objetos e Interfaces DCOM

DCOM utiliza a linguagem DCOM/IDL para construção das interfaces, possuindo o compilador Microsoft MIDL para gerar os clientes (*stubs*) e servidores (esqueletos). Apesar do DCOM estar disponível para a plataforma Windows, Linux e Solaris, é normalmente usado em ambiente Windows.

A IDL (*Interface Definition* Language) [79] é utilizada pelo DCOM para declarar as interfaces dos objetos, as quais não são declaradas juntas com a sua implementação.

O conceito de herança de interfaces não é suportado pelo DCOM, entretanto, seus componentes podem suportar múltiplas interfaces e ativar a reusabilidade através do encapsulamento dos componentes internos das interfaces e representação dos mesmos no cliente.

Um objeto pode suportar várias interfaces de acordo com a sua classe. Os objetos são instâncias em tempo de execução da classe. Por isso, a palavra "objeto" pode ser usada para se referir tanto a classe do objeto, quanto a uma instância individual da classe.

Os objetos DCOM não possuem um identificador único e para que eles sejam acessados, os clientes necessitam utilizar os ponteiros de interface. Um cliente não poderá se reconectar ao mesmo objeto mais tarde com o mesmo estado, ele poderá apenas reconectar-se com o ponteiro de uma interface da mesma classe. Isto ocorre dado que os objetos DCOM não mantêm um estado entre conexões, o que pode ser um problema em ambientes com falhas de conexões, como a Internet.

Para que um cliente acesse uma interface, ele deve utilizar ponteiros para um vetor de funções de ponteiros, chamados de tabela virtual (*virtual table*). Estas funções correspondem às operações de implementações dos objetos do servidor. Cada objeto pode possuir uma ou mais tabelas virtuais que definem o contrato entre a implementação do objeto e seus clientes.

4.1.2. Servidor DCOM

O servidor DCOM é um código, que pode ser uma DLL, um EXE ou uma classe java, que pode dar suporte a várias classes de objetos, cada uma com seus CLSID (*class identifier* – identificador da classe) e que irá criar um objeto de uma determinada classe toda vez que for solicitado. Portanto, toda vez que um cliente solicitar um objeto de uma classe específica, O DCOM deverá carregar o servidor e requisitar a este que crie um objeto desta classe. Uma classe denominada *factory* deve ser fornecida pelo servidor para a criação do referido objeto. Um ponteiro para a interface primária do objeto será retornada ao cliente após a criação deste objeto.

Um servidor DCOM pode ser implementado de várias formas:

- Servidor *In-process*: ocorre quando o servidor estiver sendo executado no mesmo processo que o cliente. No ambiente Windows e Windows NT, estes servidores são implementados como DLLs (*Dynamic Link Libraries*) e são carregados diretamente no processo cliente;
- Servidores Locais: neste caso, os servidores são executados em processos diferentes, porém na mesma máquina. O LRPC (*Lightweight* RPC) do DCOM é utilizado pelos clientes para comunicação com estes servidores;
- Servidores Remotos: são executados em processos e máquinas diferentes, podendo ser também em outro sistema operacional. Para a comunicação com estes servidores, os clientes utilizam o mecanismo de RPC [74].

Tanto para os clientes quanto para os servidores, toda esta comunicação é feita de forma transparente. Para os clientes, todos os acessos são feitos através de ponteiros de interfaces, que de uma forma ou de outra, estarão em algum momento no mesmo processo que o servidor (ou *in-process*). Quando o objeto já está *in-process*, a chamada automaticamente também já está neste processo. Porém, se o objeto estiver *out-of-process*, ou seja, em processo ou máquina diferente do servidor, sua solicitação deverá ser passada primeiramente para o *proxy* (corresponde ao *stub* do cliente no CORBA, seção 4.2), que irá gerar um mecanismo de RPC apropriado e transmiti-la ao processo ou máquina destino. O Gerente de Controle de Serviço (*Service Control*

Manager - SCM) é o elemento responsável por localizar o servidor, além de ser quem se envolve com as invocações de RPC entre clientes e servidores.

No caso dos servidores, é o processo requisitante que em algum momento estará *in-process*. Se o objeto já estiver no mesmo processo, então é porque este é o próprio cliente. Caso contrário, este objeto é o *stub* (corresponde ao esqueleto no CORBA, seção 4.2), que recebeu a mensagem remota do *proxy* e transformou-a em uma chamada de interface para o objeto do servidor.

DCOM foi idealizado visando a integração de ambientes heterogêneos, de forma a permitir a reutilização de ferramentas e objetos, diminuindo assim, os custos de novas aquisições e produções e também para um maior aproveitamento da produção da equipe de desenvolvimento.

O objetivo foi alcançado com sucesso sendo que através das plataformas Windows. Objetos podem migrar de um Windows 95 para um Windows NT sem maiores dificuldades. Já em relação a integração de plataformas heterogêneas não aconteceu o mesmo. Neste caso, é necessário o auxílio de outros pacotes de software para permitir que, por exemplo, um sistema UNIX se torne um cliente do NT, contudo o inverso não é verdadeiro.

Desta maneira, diz-se que DCOM é um tecnologia apropriada para a interação de sistemas baseados na plataforma Windows que não necessitam interagir com objetos localizados em sistemas com outros tipos de plataforma.

4.2. CORBA (*Commom Object Request Broker Architecture*)

Definida pelo OMG [77], a *Object Management Architecture* (OMA) [76] tem por objetivo definir padrões mínimos para alcançar a interoperabilidade em ambientes heterogêneos. O OMG é um consórcio de empresas, universidades e centros de pesquisa que buscam a identificação e especificação destes padrões.

A OMA é composta por um modelo de objetos e por um modelo de referências. O modelo de objetos define padrões para a descrição de objetos distribuídos em ambientes heterogêneos, enquanto o modelo de referências, a interação entre estes objetos.

No modelo de objetos definido pela OMA, um objeto é uma entidade encapsulada composta por uma identificação imutável e por um conjunto de serviços. O conjunto de serviços de cada objeto só pode ser acessado por meio de interfaces bem definidas. A implementação e a localização dos serviços são transparentes (de acesso e localização) ao cliente.

O *Object Request Broker* (ORB), definido como o principal componente da OMA, é o responsável por facilitar a comunicação entre clientes e objetos. A especificação CORBA detalha as interfaces e características do componente ORB padronizado pela OMA .

Os outros componentes da OMA são:

- Objetos de serviços: interfaces de serviços que adicionam mais funcionalidades às funções básicas do ORB. Por exemplo, um serviço que recupere referências a objetos remotos. Dois exemplos de serviços definidos pela OMA resolvem este problema: o serviço de nomes (o qual permite que o cliente recupere a referência a um objeto remoto através do seu nome) e o serviço de onde o cliente recupera a referência através de uma consulta às propriedades do objeto remoto). O conjunto completo de serviços está definido em [20];
- Facilidades comuns: interfaces de serviços orientados a aplicações de usuários finais (*end-user applications*). Por exemplo, a facilidade de criar uma ligação entre uma planilha eletrônica e um relatório, definida pelo padrão *Distributed Document Component Facility* (DDCF) [5];
- Facilidades de domínio: interfaces de serviços orientadas a domínios de aplicação específicos. Por exemplo, alguns casos de interfaces de domínios já disponíveis estão ligadas às áreas de telecomunicações, medicina e economia. O conjunto completo de domínios padronizados pelo OMG está definido em [20];
- Objetos de aplicação: interfaces desenvolvidas para uma aplicação específica. Como o OMG não desenvolve aplicações (mas padrões), estas interfaces não possuem padronização. Porém, se com o passar do tempo um grupo de serviços sobressair-se em determinado domínio de aplicação, ele pode tornar-se candidato a uma futura padronização do OMG.

CORBA é um conjunto de especificações definidas pelo OMG e possui os seguintes componentes:

- ORB *Core*: conforme citado anteriormente, o ORB realiza a passagem de requisições aos objetos e retorno das respostas aos clientes de origem. A principal característica apresentada pelo ORB é a transparência na comunicação entre o cliente e o objeto requerido;
- OMG *Interface Definition Language* (OMG IDL): para que o cliente possa realizar requisições, ele deve conhecer os tipos das operações suportadas pelo objeto. A interface do objeto especifica as operações, os tipos suportados e as requisições que podem ser realizadas pelo cliente. A linguagem de interfaces criada pelo OMG chama-se OMG IDL. A OMG IDL é uma linguagem apenas declarativa, pois não suporta estruturas de controle;
- Repositório de interfaces: toda aplicação baseada na arquitetura CORBA necessita de interfaces OMG IDL durante a execução. O repositório de interfaces permite que o sistema de tipos da interface possa ser conhecido em tempo de execução. Assim, evita-se que a aplicação tenha de ser recompilada a qualquer modificação nos tipos de dados utilizados pela interface;
- *Stubs* e esqueletos: o *stub* é um mecanismo que permite ao cliente realizar a requisição, enquanto o esqueleto é o mecanismo que entrega a requisição ao objeto. Outra funcionalidade do *stub* é armazenar a referência remota ao objeto. Enviar requisições através dos *stubs* e esqueletos denomina-se invocação estática, ou seja, cada aplicação cliente possuirá o seu *stub* respectivo, bem como a implementação do objeto, o seu esqueleto (os *stubs* e esqueletos são compilados a cada modificação realizada na interface da implementação do objeto);
- Interface de invocação dinâmica (*Dynamic Invocation Interface* – DII) e interface de entrega dinâmica (*Dynamic Skeleton Interface* – DSI): a DII suporta uma requisição dinâmica do cliente, enquanto a DSI possibilita a entrega dinâmica da requisição ao objeto. A DII e a DSI podem ser vistas como um *stub* e um esqueleto dinâmicos, respectivamente. Ou seja, a DII e a DSI substituem os *stubs* e os esqueletos pré-compilados. Estes mecanismos são muito úteis em aplicações que necessitam de alterações com muita freqüência, pois evita que *stubs* e esqueletos tenham de ser alterados a cada modificação realizada na aplicação cliente ou na implementação do objeto;

- Adaptador de objetos: é um componente que converte as requisições da aplicação cliente na interface que verdadeiramente irá requisitar o serviço à implementação do objeto. É através do adaptador de objetos que a arquitetura CORBA suporta a diversidade de implementação de objetos. O adaptador de objetos comprova a tentativa do OMG de manter o ORB o mais simplificado possível.

4.2.1. Interoperabilidade entre ORBs

Em uma rede de comunicação, podem existir diferentes ORBs provenientes de diferentes fabricantes. Programas desenvolvidos com um ORB devem poder se comunicar com outros programas desenvolvidos para outros ORBs . Para assegurar que isto aconteça, o padrão CORBA especifica que os componentes do ORB devem se comunicar utilizando protocolos de rede padrão, denominados *General Inter-ORB Protocol* (GIOP) [77] e *Internet Inter-ORB Protocol* (IIOP) [77].

O GIOP especifica um conjunto de formatos de mensagens e uma representação de dados única entre ORBs. Foi projetado para operar sobre um protocolo da camada de transporte orientado a conexão, como o TCP (*Transmission Control Protocol*) ou IPX (*Internet Packet Exchange*). O GIOP apresenta apenas sete formatos de mensagens e não realiza nenhum tipo de negociação de formatos. Os tipos de dados mapeados a partir da linguagem IDL são transmitidos utilizando o mapeamento CDR (*Common Data Representation*), o que permite que estes tipos de dados possam ser interoperáveis e tenham uma representação independente do hardware, ou do sistema das máquinas envolvidas.

O IIOP é o protocolo que define como as mensagens GIOP são transmitidas sobre o protocolo TCP/IP. Ele permite que a Internet seja utilizada como um poderoso meio de comunicação entre ORBs, possibilitando a comunicação entre ORBs que podem estar localizados em diferentes partes do mundo.

4.3. Java/RMI (*Remote Method Invocation*)

Java/RMI é um sistema de invocação remota de métodos que estende os conceitos de RPC (*Remote Procedure Call*) [74] para dar suporte a objetos distribuídos Java. Mecanismos de RPC tradicionais foram construídos objetivando a heterogeneidade dos componentes de rede e de *hardware de* diferentes processos. Um sistema distribuído em uso pode conter máquinas diferentes, com sistemas operacionais e representações de dados distintos. A solução encontrada para que esses componentes pudessem se comunicar foi o desenvolvimento de procuradores (*stubs* e esqueletos) que convertem dados enviados por outras máquinas. Linguagens para a definição de interfaces de procedimentos remotos, tal como a IDL [79], foram desenvolvidas para lidar com representações de dados diferentes. Há duas desvantagens principais relacionadas ao uso de IDLs:

- A natureza neutra de uma IDL restringe a classe de parâmetros que podem ser passados em um procedimento remoto a tipos básicos (como *int*) e a objetos remotos. Essa restrição vem do fato de que os parâmetros definidos em IDL devem possuir uma representação equivalente na linguagem de programação usada pelo cliente/servidor;
- A natureza dos dados passados em procedimentos remotos é estática. O receptor de uma chamada remota precisa saber exatamente o tipo do objeto que está sendo transmitido pela rede, para que ele possa associar o *stub*/esqueleto correspondente ao objeto transmitido. Portanto, o uso de polimorfismo [72] não é permitido - a classe de um objeto transmitido não pode ser uma subclasse (mais específica) da classe esperada. Caso isso aconteça, o receptor converte o objeto transmitido (*implicit casting*) ao tipo esperado por ele (menos específico), introduzindo a possibilidade de erros no programa.

Java/RMI, por sua vez, não precisa lidar com questões relacionadas com heterogeneidade, já que tanto as classes do cliente como as do servidor são escritas em Java e executam em JVMs, o que torna a rede uma coleção de máquinas (virtuais) homogêneas e elimina a necessidade de linguagens adicionais como a IDL.

Em um sistema que utilize Java/RMI, a aplicação servidora não necessariamente precisa estar executando em um computador servidor. Ela pode vir a ser uma aplicação sendo executada em um cliente *Web* e o servidor *Web* terá a função de cliente, isso dependerá de quem está disponilizando o objeto remoto.

Java/RMI utiliza um mecanismo de comunicação padrão, adotado pelo RPC, para comunicação entre objetos remotos: *stubs* e esqueletos. O *stub* atua no lado cliente da aplicação como um identificador local do objeto remoto. Já o esqueleto é responsável pela entrega da requisição ao objeto.

As informações trocadas entre o *stub* e o esqueleto sempre são entregues ou recebidas através do *marshal stream*, usado para transportar parâmetros, exceções e erros.

Na arquitetura Java/RMI, o *stub* realiza as seguintes operações para que uma aplicação cliente possa requisitar serviços a um objeto:

- Inicialização da conexão com a JVM remota que contém o objeto requerido;
- *Marshaling* (escrita e transmissão) dos parâmetros da requisição a JVM remota;
- Aguardar pelo resultado da requisição disparada, posto que o RMI possui suporte somente para comunicação síncrona;
- *Unmarshaling* (leitura) dos valores ou exceções recebidos;
- Retorno do resultado a aplicação cliente.

O esqueleto, ao receber a requisição de um *stub* é responsável por:

- *Unmarshaling* (leitura) dos parâmetros recebidos do stub;
- Invocar a implementação do objeto, realizando a execução do método requisitado com os parâmetros recebidos;
- *Marshaling* (escrita e transmissão) do resultado ao stub de origem.

Outros componentes da estrutura RMI são a camada de referência remota e a camada de transporte, conforme mostra a figura 4.1.

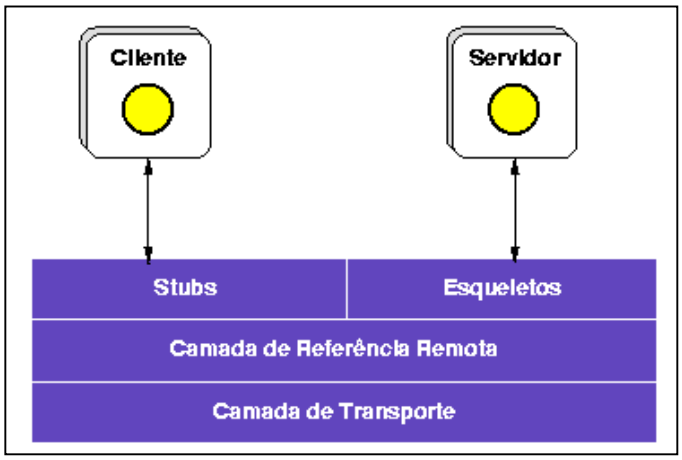

Figura 4.1 - Arquitetura de alto-nível do RMI, baseada em [90]

A camada de referência remota é a abstração entre as classes *stub*/esqueleto e os protocolos suportados pela camada de suporte. A comunicação entre a camada de referência remota e a camada de transporte é realizada através de um *stream* orientado à conexão. Serão adicionados, suporte à replicação de objetos, estratégias para a recuperação de falhas em conexões e persistência de objetos [88].

A camada de transporte é responsável por gerenciar a comunicação entre máquinas remotas. Ela disponibiliza *streams* que são acessados pela camada de referência remota para enviar e receber dados de máquinas remotas. O protocolo padrão utilizado é o TCP/IP. A camada de transporte ativa as conexões às máquinas remotas, gerencia e monitora a conexão, verificando se ela está ativa e aguarda por conexões de outras máquinas. A camada de transporte pode ser modificada para suportar *streams* com suporte a criptografia e algoritmos para compressão de dados, entre outros aprimoramentos. Pela independência que existe entre esta camada e a camada de referência remota, os *stubs*/esqueletos não necessitam conhecer as modificações ocorridas na camada de transporte.

A principal característica que distingue Java RMI de outros sistemas RPC, no entanto, é a sua capacidade de carregar o código fonte de classes remotas dinamicamente, permitindo o uso de polimorfismo. O mecanismo que realiza essa operação é composto por três componentes principais: um carregador de classes especializado, um gerenciador de segurança e a técnica de conserva (*pickling* [91]). Esta técnica permite que um objeto seja representado como uma seqüência de bytes e inclui informação sobre o tipo de objeto e sobre a localização do seu código fonte. Quando uma JVM recebe a referência de um objeto remoto, ela verifica o tipo

específico do objeto. Se a JVM não possuir o *stub* específico para esse objeto, o carregador de classes especializado pode localizar e carregar o seu código fonte utilizando o endereço que o objeto carrega consigo. O código é verificado pelo gerenciador de segurança antes de ser executado. Uma descrição mais detalhada sobre o mecanismo usado por Java para carregar código dinamicamente encontra-se em [90].

A figura 4.2 exemplifica uma interação típica entre um cliente e um servidor Java RMI. O servidor (1) registra uma referência para um objeto remoto no serviço de nomes e (2) após obter essa referência, (3) o cliente pode executar chamadas diretamente nesse objeto, possivelmente obtendo referências para outros objetos remotos. A figura também mostra um servidor *Web* sendo usado para transferir o código e o estado dos objetos que são passados em invocações remotas.

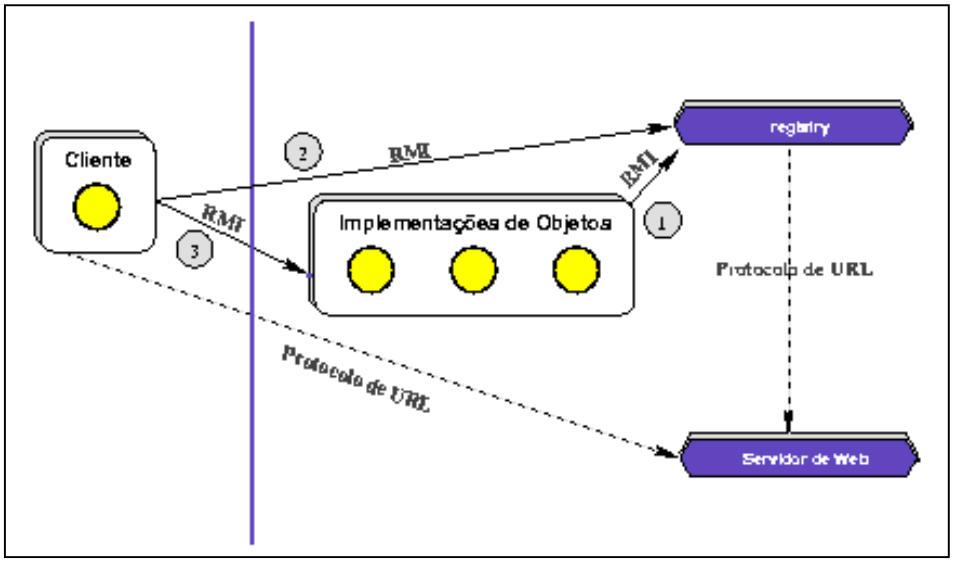

Figura 4.2 – Exemplo de uma interação típica entre um cliente e um servidor RMI, baseado em [90]

4.3.1. Modelo de Sistema Básico

A funcionalidade básica do sistema Java RMI se encontra nos pacotes *java.rmi*, *java.rmi.server*, *sun.rmi.server* e *sun.rmi.transport*. As classes contidas no pacote *sun* são específicas à implementação do JDK (*Java Development Kit*) [56] da Sun.

No modelo do Java RMI, um objeto é denominado remoto se as suas operações podem ser invocadas por objetos residentes em um espaço de memória distinto. Uma interface remota deve ser derivada, direta ou indiretamente, da interface *java.rmi.Remote*. Operações definidas em interfaces remotas são denominadas remotas e devem incluir a exceção *java.rmi.RemoteException* na sua cláusula *throws***.** Uma exceção *RemoteException* é criada quando, durante uma invocação remota, alguma falha de comunicação ocorre. Somente operações remotas estão disponíveis remotamente. Além deles, um objeto remoto pode definir outras operações não-remotas, que só serão acessíveis localmente.

A arquitetura de Java RMI consiste de três camadas, como mostra a figura 4.1: a camada de *stub*/esqueleto, a camada de referência remota e a camada de transporte. O servidor que estiver exportando interfaces remotas pode estender *java.rmi.server.UnicastRemoteObject* (caso esse objeto não estenda *UnicastRemoteobject*, ele deve definir funcionalidade equivalente à que esse objeto implementa), que define a funcionalidade básica para um objeto remoto (como explicado a seguir).

Stubs e esqueletos são gerados pelo *script rmic*, que recebe como entrada o arquivo *.class* correspondente à implementação do servidor.

Quando um servidor é criado, o construtor de *UnicastRemoteObject* executa o método *exportObject*, que instancia e exporta um objeto *sun.rmi.server.UnicastServerRef*. Ele também cria uma referência viva (*live reference*), que representa a camada de transporte do servidor e contém um endereço IP, uma porta TCP e um identificador do objeto. Além disso, *exportObject* instancia o *stub* e o esqueleto do objeto. Depois disso, uma associação entre o identificador do objeto e o *stub*/esqueleto é registrado em uma tabela de objetos, residente na camada de transporte.

O cliente, por sua vez, obtém uma referência a um objeto remoto por meio do serviço de nomes *rmiregistry* ou através de outros objetos remotos. O *stub* funciona como uma camada entre o cliente e as outras camadas do Java RMI e a sua principal função é o empacotamento *(marshaling)* e desempacotamento *(unmarshaling)* de parâmetros enviados e recebidos por/de invocações remotas. O *stub* contém um objeto *java.rmi.server.RemoteRef*, que encapsula a camada de transporte do cliente e que obtém a referência viva do servidor quando uma invocação remota é realizada,

estabelecendo uma conexão com ele. O *stub* chama o método *invoke* de *RemoteRef* para realizar a invocação remotamente. Quando a chamada chega ao servidor, a camada de transporte usa a sua tabela de objetos junto com o identificador da invocação para obter o esqueleto do objeto remoto (a partir da versão 1.2.2 do JDK, Java RMI não utiliza esqueletos. A camada de referência do lado do servidor possui recursos suficientes para desempacotar argumentos e fazer uma chamada diretamente à implementação do objeto). O esqueleto desempacota os argumentos, realiza uma chamada à implementação do objeto e retorna o resultado ao cliente. Java RMI possui recursos suficientes para diferenciar o resultado normal de uma invocação de uma exceção.

Qualquer objeto Java pode ser passado como argumento ou retornado como resultado de uma chamada remota. Java RMI implementa a semântica de transmissão *call-by-value* (chamada por valor) para objetos não-remotos.

4.3.2. Coleta de Lixo Distribuída

Uma outra característica que distingue Java RMI de outros mecanismos de RPC tradicionais é o seu mecanismo distribuído de coleta de lixo. Técnicas de gerenciamento de dados distribuídos, como a contagem explícita de referências, geralmente são complicadas e por isso têm um grande potencial para introduzir falhas no sistema [112]. Java RMI fornece suporte transparente para a coleta de lixo em aplicações distribuídas através de um protocolo que utiliza a contagem de referências. O ambiente de execução de Java RMI mantém uma tabela que contém, para cada objeto remoto, o número de referências locais para esse objeto. Quando a referência de um objeto remoto é instanciada pela primeira vez em uma JVM, Java RMI invoca o método *referenced* desse objeto e cria uma entrada para ele em sua tabela. Essa entrada contém um contador do número de processos locais que contém referências a esse objeto. À medida que clientes deixam de usar uma referência, o contador do objeto remoto decresce e, ao chegar em zero, Java RMI invoca o método *unreferenced* desse objeto. Quando nenhuma máquina virtual possui referências para um servidor e o mecanismo de coleta de lixo local dessa máquina indica que não há processos locais que possuem uma referência para ele, o servidor é coletado. Esse protocolo não funciona corretamente na presença de falhas. Especificamente, se uma partição na

rede separar o cliente de um servidor, é possível que o servidor seja coletado prematuramente.

4.3.3. A Hierarquia de Classes

A figura 4.3 apresenta a hierarquia das classes e interfaces do RMI relevantes ao projeto desenvolvido. A seguir será fornecida uma breve explicação de cada uma delas. As classes *UnicastRef* e *UnicastServerRef* e as interfaces *RemoteRef* e *ServerRef* representam a camada de referência remota da arquitetura de Java RMI, conforme ilustrado na figura 4.1.

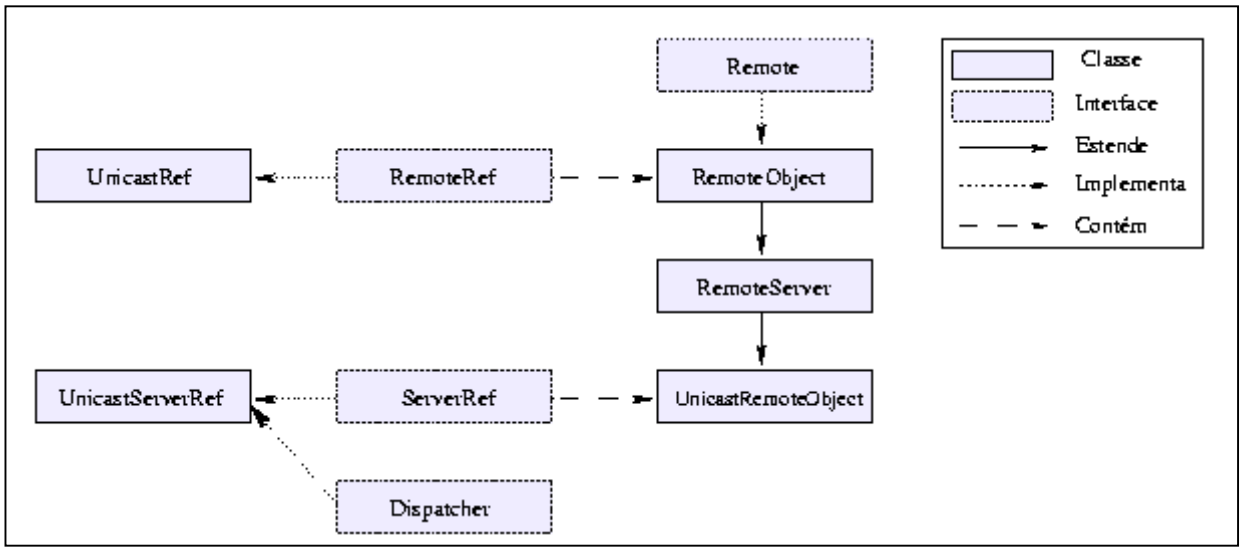

Figura 4.3: Hierarquia parcial de classes do RMI, baseada em [52]

I. Interface *java.rmi.Remote*

Remote serve para identificar interfaces cujas operações podem ser chamados de uma JVM remota. Toda interface remota deve estender, direta ou indiretamente, essa interface. Somente as operações de um objeto especificados por uma interface remota estão disponíveis remotamente. Cada operação de uma interface remota deve incluir a exceção *RemoteException* em sua cláusula *throws*.

II. Interface *java.rmi.server.RemoteRef*

Essa interface representa a ligação para um objeto remoto. O seu método mais importante é *invoke*, que permite a um *stub* invocar um método de um objeto remoto. *RemoteRef* contém uma referência viva (*sun.rmi.transport.LiveRef*), que representa a camada de transporte do sistema Java RMI.

III. Interface *java.rmi.server.ServerRef*

Essa interface estende *RemoteRef* e adiciona métodos para permitir que um objeto remoto possa ser exportado. Ao ser exportado, um objeto disponibiliza o endereço da máquina onde ele está executando, a porta onde ele receberá invocações remotas, o seu identificador e um objeto *Dispatcher*, como explicado na próxima seção. A camada de transporte armazena essas informações em uma tabela de objetos para que quando uma invocação chegue, ela possa ser direcionada ao esqueleto correto.

IV. Interface *sun.rmi.server.Dispatcher*

Essa interface define um método, *dispatch*, que é responsável por delegar uma invocação remota interceptada pela camada de transporte de um servidor para a implementação de um objeto remoto. Quando um objeto é exportado, ele passa à camada de transporte um objeto *Dispatcher*, que é armazenado em uma tabela de objetos. Quando uma invocação remota chega ao servidor, a camada de transporte utiliza essa tabela para identificar o objeto *Dispatcher* ao qual a invocação deve ser encaminhada e invoca o método *dispatch* desse objeto, que identifica o método do objeto remoto a ser invocado.

V. Classe java.rmi.server.RemoteObject

Essa classe implementa a funcionalidade de *java.lang.Object* no contexto de objetos remotos. Especificamente, *RemoteObject* sobrescreve os métodos *equals*, *hashCode*, *toString* e *clone*. A seguir, serão analisadas as semânticas desses métodos em objetos remotos:

- *equals*: enquanto que o método *Object.equals* implementa a semântica de igualdade de conteúdo (ou seja, dois objetos são iguais se possuem o mesmo estado interno), o método *RemoteObject.equals* implementa a semântica de igualdade referencial: dois objetos remotos são iguais se as suas referências remotas são iguais, ou seja, se eles se referem à mesma implementação de um mesmo objeto. Implementar a semântica de igualdade de conteúdo em objetos remotos iria requerer chamadas remotas para comparar os estados de dois objetos, aumentando consideravelmente a latência de uma invocação típica a *equals*. Além disso, chamadas remotas podem ocasionar a exceção *RemoteException*, que o método *Object.equals* não inclui na sua cláusula *throws*. Portanto, somente a semântica de igualdade referencial pode ser implementada em Java RMI sem custos adicionais;
- *hashCode*: retorna o mesmo valor para dois objetos que possuem a mesma referência remota;
- *toString*: retorna um objeto String que representa a referência desse objeto. Na implementação atual do JDK, que suporta somente objetos não-replicados, o resultado desse método contém informação sobre o endereço, a porta e um identificador do objeto remoto;
- *clone*: fazer o clone de um objeto remoto é uma operação local e não pode ser usada por clientes para criar novos objetos remotos. Essa operação, quando realizada em clientes, cria uma nova referência para o mesmo objeto remoto, mas não efetua a cópia de nenhuma das suas estruturas internas, já que isso envolveria chamadas remotas e poderia gerar uma possível exceção *RemoteException*, que *Object.clone* não inclui na sua cláusula *throws*.

VI. Classe *java.rmi.server.RemoteServer*

RemoteServer é a superclasse comum a todas as implementações de servidores e oferece um modelo para uma grande variedade de referências remotas. Especificamente, *RemoteServer* define métodos abstratos para que um objeto remoto possa ser exportado. Atualmente, o Java RMI só implementa referências remotas não-replicadas *(unicast),* que podem ser transientes ou persistentes. Referências transientes são representadas pela classe *UnicastRemoteObject* e são válidas somente enquanto o servidor estiver executando. Referências persistentes são representadas pela classe *Activatable* e podem ser ativadas automaticamente.

VII. Classe *java.rmi.server.UnicastRemoteObject*

Essa classe define um objeto não-replicado *(unicast),* cujas referências são válidas somente enquanto o processo servidor estiver executando. A principal funcionalidade que *UnicastRemoteObject* oferece é a exportação de objetos remotos.

Um objeto remoto não derivado de *UnicastRemoteObject* deve ser capaz de exportar a si mesmo e deve implementar toda a funcionalidade definida por *RemoteServer*.

VIII. Classe *sun.rmi.server.UnicastRef*

Essa classe é a referência remota de um cliente e implementa a interface *RemoteRef*. Ela disponibiliza o método *invoke*, que realiza uma invocação remota a um objeto. *UnicastRef* representa a camada de referência de um objeto remoto não-replicado, cujo estado não pode ser replicado em nenhuma JVM remota.

IX. Classe *sun.rmi.server.UnicastServerRef*

Essa classe é a referência remota de um servidor e implementa as interfaces *ServerRef* e *Dispatcher*. Ela disponibiliza métodos para exportar um objeto remoto e para despachar uma invocação remota enviada por um cliente de uma JVM remota à implementação do objeto.

4.4. Análise das Arquiteturas

DCOM pode ser uma boa solução para aplicações que rodam exclusivamente no ambiente Windows. Entretanto, os servidores projetados no protótipo desenvolvido, têm como requisito principal, a possibilidade de rodar em qualquer plataforma de software e hardware que possua uma máquina virtual Java. Além disso, conforme apresentado anteriormente na seção 4.1, os objetos DCOM não mantêm um estado entre conexões, o que pode ser um problema em ambientes com falhas de conexões, como a Internet. Dessa forma, DCOM não atende aos requisitos para o desenvolvimento do protótipo. Portanto, serão agora analisadas as arquiteturas CORBA e Java/RMI.

Segundo Hericko [49], os critérios que devem ser levados em consideração na comparação entre duas arquiteturas de objetos distribuídos são: propriedades apresentadas, maturidade, suporte a aplicações legadas, facilidade de desenvolvimento, desempenho e escalabilidade.

Neste sentido serão apresentados os seguintes resultados:

- Propriedades das arquiteturas: por ser uma plataforma de objetos distribuídos completa, com suporte a diferentes linguagens de programação, CORBA apresenta vantagens em comparação a RMI. Porém, o objetivo de RMI dar suporte apenas à linguagem Java faz com que esta arquitetura apresente algumas vantagens não encontradas na combinação Java/CORBA;
- Maturidade da arquitetura: Com a primeira versão datada de 1992, CORBA é amplamente utilizada no mercado tecnológico. A versão atual 3.0 é estável e

segura. RMI, no entanto, foi apresentada em 1998 e o mercado não apresenta aplicações de missão crítica que utilizem a arquitetura. Conclui-se que a arquitetura CORBA é mais madura que a arquitetura RMI;

- Suporte a aplicações legadas: difícil em RMI pois não apresenta mapeamentos para outras linguagens de programação. A arquitetura CORBA, por outro lado, suporta mapeamentos para diversas linguagens de programação. Conclui-se que CORBA possui maior suporte a aplicações legadas em comparação a RMI;
- Facilidade de desenvolvimento: por apresentar uma arquitetura mais completa, CORBA exige maior dedicação do programador. A adição de uma linguagem para definição de interfaces (IDL) e o mapeamento dos tipos de dados entre as linguagens de programação tornam o desenvolvimento de objetos CORBA mais complexo. Em RMI, as interfaces e as aplicações são escritas em Java, logo não existem mapeamentos entre elas. Conclui-se que o desenvolvimento em RMI é mais simples que em CORBA;
- Desempenho e escalabilidade: segundo teste realizados em [29], concluiu-se que a arquitetura RMI teve desempenho superior à CORBA quando o número de clientes é inferior a quatro. A partir de quatro clientes simultâneos conectados ao objeto, a arquitetura CORBA teve um desempenho superior a RMI.

Segundo Hericko [49], o custo para o desenvolvimento de aplicações baseadas na arquitetura CORBA justifica-se para aplicações de larga escala onde o suporte a aplicações legadas é necessário e uma alta carga de clientes é esperada. A arquitetura RMI, por outro lado, é indicada para aplicações de baixa escala onde o suporte a aplicações legadas pode ser gerenciado por módulos individuais criados pelo próprio programador e a facilidade de programação é mais crítica que o desempenho do sistema.

Segundo Orfali [80], a facilidade de codificação RMI está ligada a dois pontos: a utilização de apenas uma linguagem de programação e a conversão de tipos. Enquanto em RMI utiliza-se somente a linguagem Java, na arquitetura CORBA o programador necessita utilizar duas linguagens de programação, uma para codificação das interfaces (OMG IDL) e outra para a codificação dos objetos. Como conseqüência disso, o programador CORBA lida com o problema da conversão de tipos na passagem de parâmetros entre interfaces e objetos. Em RMI esta dificuldade não se verifica.

Segundo Oliveira [78], Java/RMI é muito mais eficiente que CORBA quando temos objetos homogêneos, ou seja, clientes Java e servidores Java. Além disso, pode-se citar outras vantagens em relação à arquitetura CORBA:

- A necessidade de domínio de apenas uma linguagem, no caso, Java, como destacado acima;
- O suporte à passagem de objetos por valor nos métodos das interfaces RMI;
- A utilização da linguagem IDL na arquitetura CORBA exige que o programador preocupe-se com mapeamentos entre tipos de dados, no caso de utilização entre estruturas de dados complexas (*structs*). No caso de RMI, por utilizar apenas uma linguagem de programação, esta heterogeneidade entre tipos não se verifica;
- A simplicidade de instalação da arquitetura RMI, pois trata-se de um pacote integrado às JVMs e de domínio público;
- Alta portabilidade;
- Suporte à programação concorrente através do mecanismo *multithread* da linguagem Java, possibilitando que os servidores desenvolvidos possam alocar uma *thread* para atender a cada requisição de cliente.

4.5. Conclusão

À primeira vista, CORBA parece ser a escolha mais indicada para compor a arquitetura a ser apresentada no capítulo 5, pois possui uma especificação bem definida e abrangente, é independente de plataforma e de linguagem de programação. Entretanto, o suporte à heterogeneidade de linguagens de programação fornecida por CORBA se torna irrelevante para construção do protótipo, visto que, todos os objetos serão escritos na linguagem Java. Apesar de CORBA apresentar as melhores especificações, os produtos disponíveis são caros e pesados, além disso a utilização da linguagem de programação Java na implementação do protótipo é indispensável e será abordada em detalhes no capítulo 5.

A escolha de Java/RMI para a implementação do protótipo se baseia nos requisitos para construção do referido protótipo, na arquitetura proposta, na disponibilidade e amadurecimento da tecnologia e na análise de custos financeiro e

computacional envolvidos, ou seja Java/RMI é uma API que vem em conjunto com o JDK 1.1 ou superior, é gratuita, é mais eficiente que CORBA para aplicações Java-Java e precisa somente de uma JVM do lado do cliente, comuns aos *browsers* mais conhecidos.

Capítulo 5

Módulo de Integração de Bancos de Dados em Sistemas Tutores Inteligentes

---

Neste capítulo, serão descritos o protótipo desenvolvido e suas características.

Capítulo 5

Módulo de Integração de Bancos de Dados em Sistemas Tutores Inteligentes

A arquitetura implementada é uma instância da arquitetura cliente / servidor em três camadas, onde na primeira camada é descrita a interface com o usuário, a segunda camada é representada pelos objetos distribuídos, onde está contida a lógica do negócio e na última camada são apresentados os diversos bancos de dados a serem integrados, podendo ser visualizada na figura 5.1 e é composta dos seguintes componentes:

- interface do usuário escrita nas linguagens HTML e Java;
- servidor escrito em Java / *Servlets* e Java / JDBC para acessar o banco de dados servidor;
- servidores escritos em Java / RMI como suporte a objetos distribuídos e usando Java / JDBC para acessar os bancos de dados cooperantes.

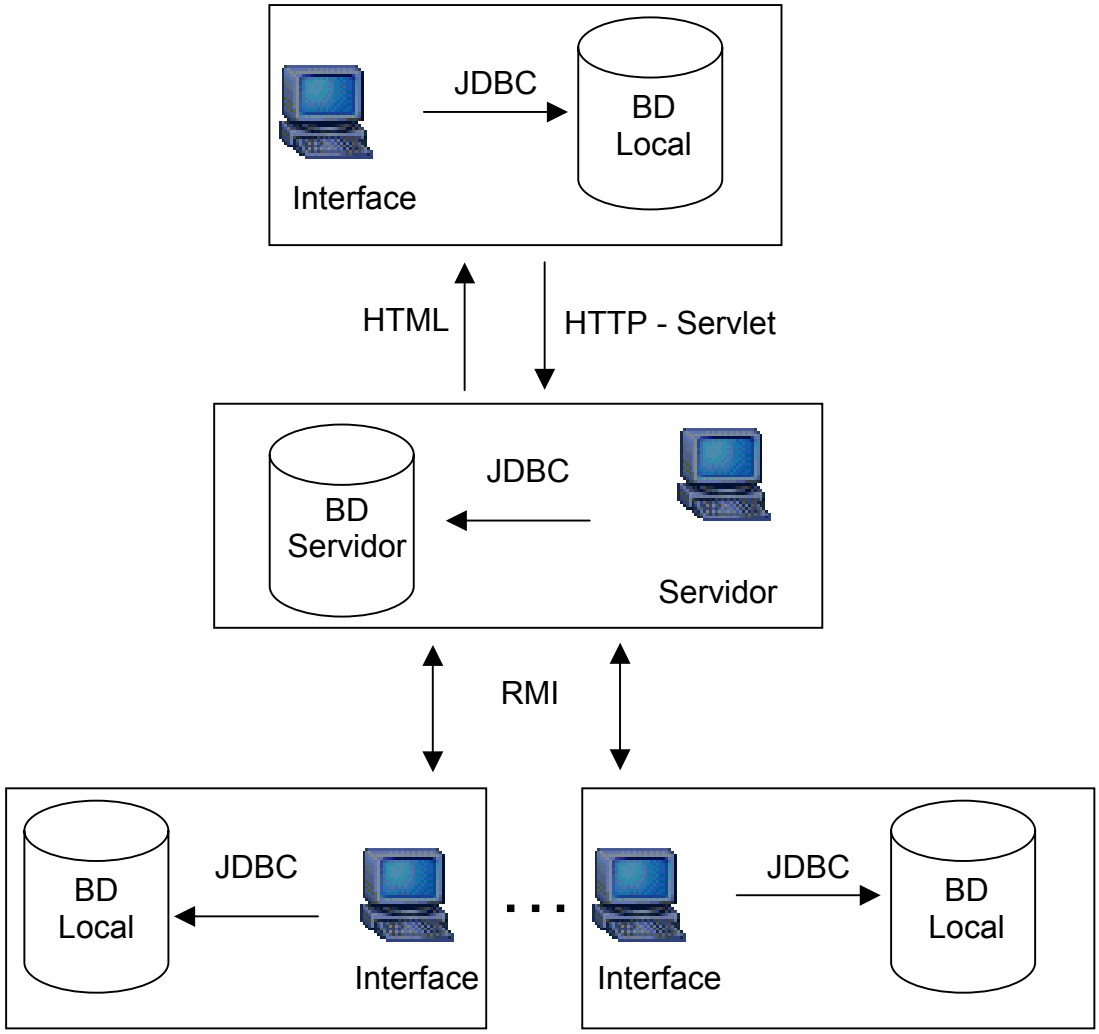

Figura 5.1. Arquitetura proposta para o módulo de integração de banco de dados

Nota-se a importância da linguagem Java para esta arquitetura através das camadas de interface do usuário, objetos distribuídos e interface de acesso aos bancos de dados.

A interface do usuário pode ser executada em qualquer *browser* que possua uma Máquina Virtual Java (JVM) [63]. O acesso ao conteúdo instrucional do STI é feito através do protocolo HTTP com chamadas aos *servlets*, e recebendo como respostas páginas HTML. Os *servlets* são responsáveis pela conexão com o banco de dados do servidor através de um *driver* JDBC.

Quando um usuário solicita uma consulta aos bancos de dados locais dos demais estudantes, os *servlets* chamam os servidores RMI através do protocolo JRMP

(Java *Remote Method Protocol*) [88]. Os servidores RMI são responsáveis pelos acessos aos bancos de dados cooperantes através dos *drivers* JDBC.

Essa arquitetura é flexível, pois as camadas e suas funcionalidades estão bem descritas e a comunicação entre as camadas é feita através de protocolos padrão. É independente de plataforma, pois necessita somente de um *browser* do lado do cliente. Do lado do servidor é mantida essa característica, visto que os demais componentes estão escritos na linguagem Java, portanto também são portáveis e independentes de plataforma.

A API JDBC está sendo utilizada para acessar todas as bases de dados, quer a base de dados esteja localizada no servidor, quer as bases de dados sejam cooperantes. A funcionalidade básica dessa API é prover basicamente: conexão com bancos de dados, enviar comandos SQL e processar resultados.

Nesta arquitetura, A API RMI está sendo utilizada para comunicação entre objetos distribuídos (remotos) no que diz respeito aos acessos aos bancos de dados cooperantes. A estratégia principal é manter vários servidores RMI rodando, permitindo com isso, a possibilidade de rodar solicitações de conexões com os bancos de dados cooperantes em paralelo. Isso somente é possível pela característica *multi-threading* da linguagem Java, em especial a característica *multi-threading* dos *servlets*.

Esse módulo, inicialmente proposto por Souza e Campos [103], deve possuir características de distribuição, autonomia e integração através dos conceitos de banco de dados federados [15, 98], que consiste na integração de vários bancos de dados cooperantes e autônomos, mas que participam de uma federação possibilitando o compartilhamento parcial e controlado dos seus dados.

O esquema da base de dados cooperante é similar à base de conhecimento disponibilizada pelo sistema, somente que esta é constituída pelas pesquisas do estudante. Dessa forma, não foi necessária a utilização de XML [1]. O citado esquema é mostrado na figura 5.2, onde (PK) significa que o atributo é chave primária (ou parte dela) e (FK) significa que o atributo é chave estrangeira.

Tabela: Curso

Atributo	Tipo	Descrição
cod_curso (PK)	Inteiro	Código do Curso
titulo	Texto	Título do Curso

Tabela: Lição

Atributo	Tipo	Descrição
cod_curso (PK, FK)	Inteiro	Código do Curso
cod_lição (PK)	Inteiro	Código da Lição
titulo	Texto	Título da Lição

Tabela: Tópico

Atributo	Tipo	Descrição
cod_curso (PK,FK)	Inteiro	Código do Curso
cod_lição (PK,FK)	Inteiro	Código da Lição
cod_tópico (PK)	Inteiro	Código do Tópico
titulo	Texto	Título do Tópico
descrição	Memo	Conteúdo do Tópico

Figura 5.2. Esquema do Banco de Dados Cooperante

O tratamento de heterogeneidades pode ser totalmente implementado nessa arquitetura. As diferenças de hardware, software e sistemas operacionais foram totalmente suportadas pelas próprias características da linguagem Java discutidas no capítulo 3. A API JDBC também facilita muito o tratamento das heterogeneidades entre os Sistemas de Gerenciamento de Banco de Dados (SGBDs), dado que fornece uma interface única dos bancos de dados cooperantes e esconde as características individuais e as diferenças dos dialetos SQL.

5.1. Implementação

A necessidade de se construir um protótipo voltado para Web enfatizou a necessidade do uso da linguagem Java como parte integrante da arquitetura e da construção do módulo de integração.

A escolha da tecnologia dos *Servlets* se deu ao fato que eles rodam nos servidores evitando-se assim problemas relativos à compatibilidade de interfaces gráficas entre os diversos clientes, assim como, ao fato de que interfaces HTML são mais simples e fáceis de serem escritas e entendidas.

A API JDBC proporciona um nível de abstração desejável para acesso aos bancos de dados cooperantes. A evolução dessa API, através da especificação JDBC 2.0, vem enfatizar a sua importância e abrangência. Essa API tende a se tornar um padrão de acesso a bancos de dados de qualquer natureza, seja relacional, orientado a objetos ou objeto relacional.

A análise comparativa feita na seção 4.4 enfatizou a necessidade da utilização da API Java / RMI para implantação dos objetos distribuídos.

5.2. Ambiente de desenvolvimento

Durante o desenvolvimento desse protótipo foram destacadas algumas características importantes, sendo algumas inerentes à linguagem Java e outras relativas à arquitetura proposta:

- O módulo de integração foi escrito em Java usando alguns recursos de HTML e JavaScript, e principalmente as APIs de Java: *Servlets*, Java/RMI e JDBC;
- Pode ser acessado via Web através de um browser que suporte Java;
- Tanto os servidores quanto os clientes são independentes de plataformas;
- Suporta acessos concorrentes e simultâneos;
- A transparência fornecida pelas APIs permite uma maior abstração por parte do desenvolvedor. A transparência fornecida por RMI esconde os problemas

relativos à comunicação entre objetos distribuídos. JDBC permite uma abstração no tratamento de heterogeneidades de cada SGBD participante;

O módulo de integração foi projetado para ser executado em qualquer plataforma ou ambiente que possua uma máquina virtual Java.

Para o desenvolvimento do protótipo, foi necessário um microcomputador com acesso à Web, e com os seguintes programas instalados:

- Sistema operacional Windows NT 4.0;
- JDK (*Java Development Kit*) 1.3;
- Servidor Web JSDK (*Java Servlet Development Kit*) 2.1;
- Microsoft Access, Microsoft SQL Server e MySQL.
- TextPad, para edição de arquivos .Java e .HTML;

Mesmo tendo-se optado pelo servidor *Web* JSDK, é interessante destacar o servidor Web Tomcat, definido pela SUN, através do Projeto Apache Jakarta [4]. Este servidor é usado como referência oficial da SUN para a implementação das tecnologias Java *Servlets* e Java *Server Pages*.

5.3. Requisitos

Inicialmente foram analisados os seguintes requisitos para o desenvolvimento do módulo de integração: ser independente de plataforma, robusto, seguro, voltado para Web, com interfaces fáceis de usar e de dar manutenção e que permitisse o acesso a várias fontes de dados simultaneamente, prevendo uma integração entre elas.

A escolha da linguagem Java oferece suporte à construção de um protótipo independente de plataforma, robusto e seguro como abordado na seção 3.6.

Os *Servlets* e a linguagem HTML possibilitam o desenvolvimento de um software voltado para Web, com interfaces simples e fáceis de serem usadas e alteradas.

A escolha da API JDBC para acesso a bases de dados fornece uma interface segura e poderosa para acesso a várias fontes de dados simultaneamente, mantendo-

as autônomas e independentes. No módulo de integração foi testado a integração de bases de dados dos seguintes SGBDs: Microsoft Access, Microsoft SQL-Server e MySQL, entretanto existe a possibilidade de integração de qualquer SGBD que possua um *driver* JDBC.

5.4. Especificação do Software

O diagrama de classes UML [11] do módulo de integração é mostrado na figura 5.3.

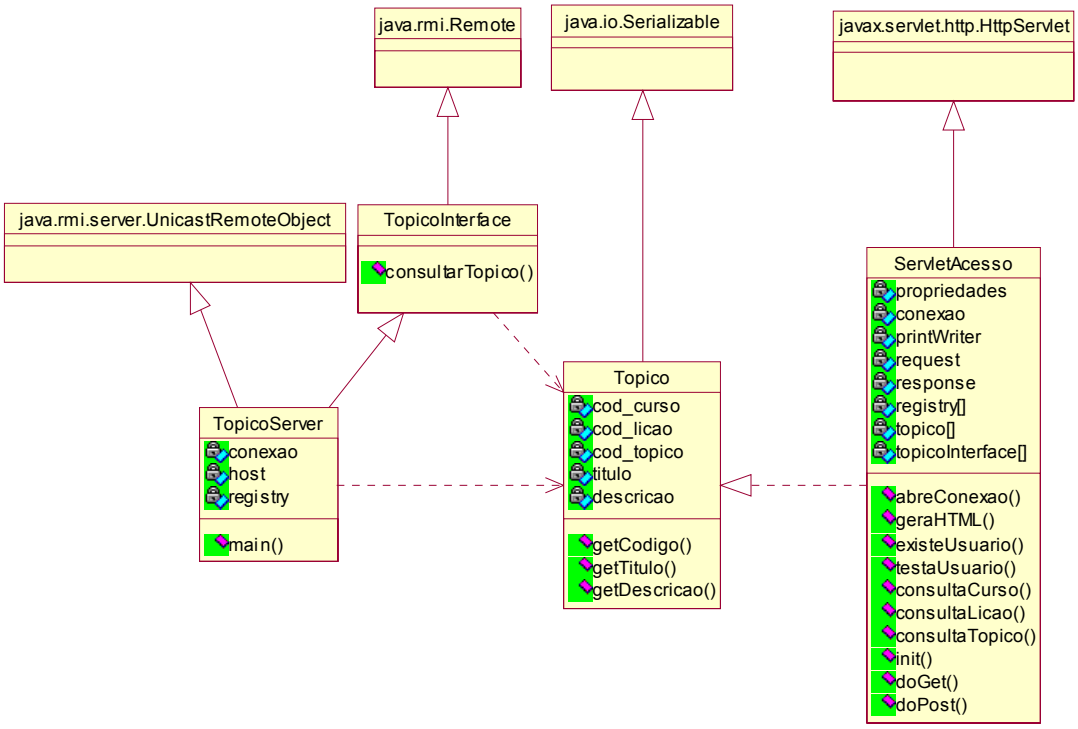

Figura 5.3. Diagrama de Classes do Módulo de Integração de Bancos de Dados

As classes e seus respectivos métodos são descritos a seguir:

- Topico: classe que possui os atributos e métodos referentes ao tópico que será consultado.
  - Métodos:
    - getCodigo: que retorna o código do tópico;
    - getTitulo: que retorna o título do tópico;
    - getDescricao: que retorna a descrição (conteúdo) do tópico.

- TopicoInterface: interface que possui a assinatura do método utilizado para consulta ao tópico escolhido. É um dos componentes da arquitetura Java/RMI.
    - Método:
        - consultarTopico: que retorna um objeto da classe Topico.
- TopicoServer: classe que implementa o método definido em TopicoInterface. Também é um dos componentes da arquitetura Java/RMI. Esta classe deve estar localizada na máquina do usuário e, caso o usuário queira disponibilizar seu banco de dados local para o módulo de integração, ela deve ser instanciada.
    - Métodos:
        - consultarTopico: que é responsável em executar a consulta na base de dados local e retornar a resposta ao *servlet*;
        - main: que é responsável em fazer a conexão à base de dados local e disponibilizá-la para o módulo de integração.
- ServletAcesso: classe que implementa o servlet que é o responsável pelo processamento de consultas solicitadas nas bases de dados cooperantes.
    - Métodos
        - *abreConexao: conexão com o banco de dados servidor, onde estão as informações sobre os usuários;*
        - geraHTML: geração de páginas HTML em resposta a requisições dos usuários;
        - existeUsuario: controle de acesso ao módulo de integração;
        - testaUsuario: testa e estabelece a comunicação com as bases de dados cooperantes. Neste caso, funciona como um dos componentes da arquitetura Java/RMI;
        - consultaCurso: consulta e visualiza os cursos disponíveis no banco de dados servidor;
        - consultaLicao: consulta e visualiza as lições do curso escolhido;
        - consultaTopico: consulta e visualiza os tópicos da lição escolhida;
        - doGet e doPost, que foram sobrescritos para tratar requisições GET e POST, respectivamente.

A figura 5.4 apresenta o diagrama de casos de uso UML [11] do módulo de integração de bancos de dados implementado.

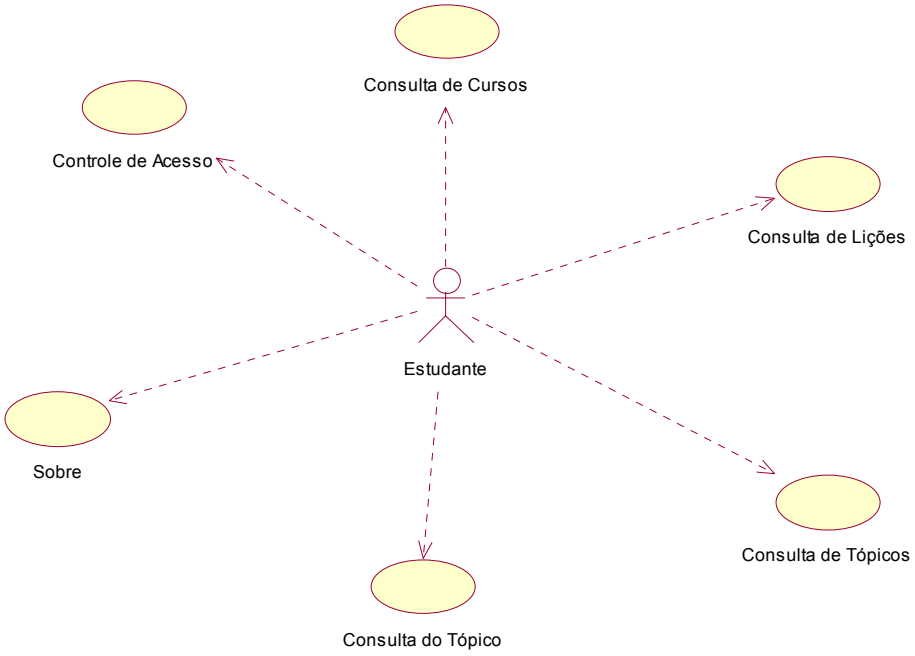

Figura 5.4. Diagrama de Casos de Uso do Módulo de Integração de Bancos de Dados

A figura 5.5 apresenta a estrutura do software, bem como o relacionamento entre os seus módulos. A abordagem utilizada para representar essa estrutura é semelhante às utilizadas para documentar os mapas de sites (*Site Maps*).

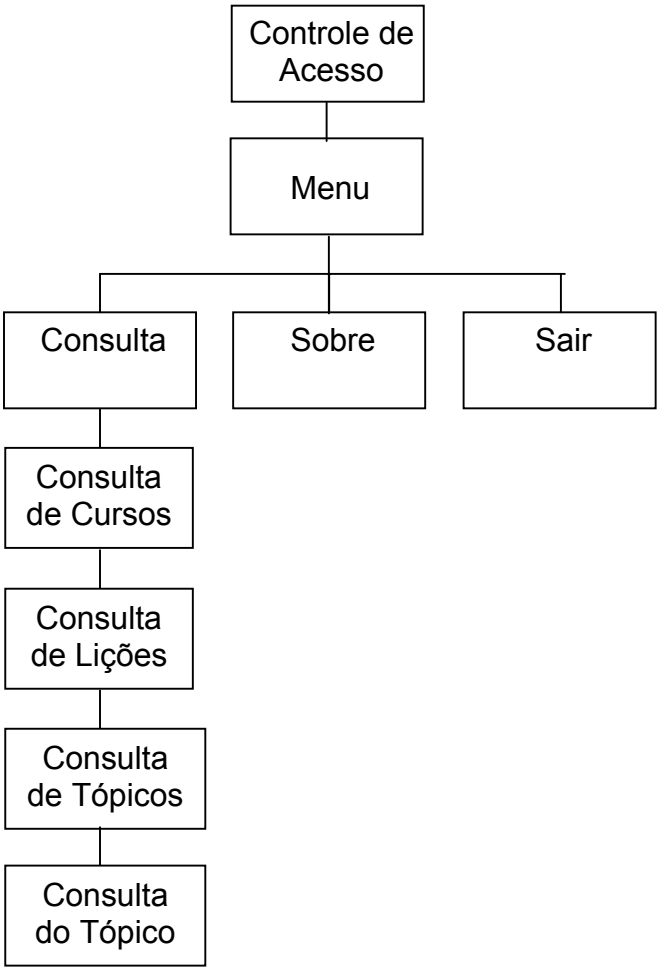

Figura 5.5. Visão Modular do Módulo de Integração de Bancos de Dados

O módulo **Controle de Acesso** foi implementado através do método **existeUsuario** da classe **ServletAcesso**, que envolve a validação dos dados digitados pelo usuário para ter acesso ao sistema.

O módulo **Menu** foi implementado através do método **geraHTML**[1] da classe **ServletAcesso**. Ele gera uma página HTML com as opções de **Consulta**, **Sobre** e **Sair**.

O módulo **Sobre** foi implementado através do método **geraHTML** da classe **ServletAcesso**. Ele gera uma página HTML com informações sobre o módulo de integração.

O módulo **Sair** foi implementado através do método **geraHTML** da classe **ServletAcesso**. Ele gera uma página HTML e desconecta o usuário do sistema.

[1]O método geraHTML, através da passagem de parâmetros, é usado para gerar todas as páginas HTML.

O módulo **Consulta** foi implementado através do método **geraHTML** da classe **ServletAcesso**. Ele chama o módulo **Consulta de Cursos** que foi implementado através do método **consultaCurso** da classe **ServletAcesso**, que gera uma página HTML contendo um formulário para escolha de um dos cursos cadastrados no sistema para consulta.

O módulo **Consulta de Lições** foi implementado através do método **consultaLicoes** da classe **ServletAcesso**. Este módulo é chamado após a escolha do curso e gera uma página HTML contendo um formulário para escolha de uma das lições cadastradas do curso escolhido.

O módulo **Consulta de Tópicos** foi implementado através do método **consultaTopicos** da classe **ServletAcesso**. Este módulo é chamado após a escolha da lição e gera uma página HTML contendo um formulário para escolha de um dos tópicos cadastrados da lição escolhida.

O módulo **Consulta do Tópico** foi implementado através do método **geraHTML** da classe **ServletAcesso**. Este módulo é chamado após a escolha do tópico e gera uma página HTML sobre a consulta solicitada a respeito do tópico escolhido.

5.5. Exemplo de Utilização do Módulo de Integração

Esta seção apresenta as principais páginas do protótipo e explica o funcionamento delas. A Figura 5.6, apresenta a janela inicial de entrada do sistema.

O usuário cadastrado deve informar seu *login* e sua senha. Após o envio destas informações (ao clicar com o mouse no botão Entrar) o sistema faz uma conexão com o banco de dados servidor para validação do usuário. Caso o usuário não esteja cadastrado no sistema ou a senha esteja incorreta, a janela mostrada na Figura 5.7 é exibida.

Figura 5.6: Janela Inicial do Módulo de Integração

Figura 5.7: Janela com Tentativa de Entrada no Sistema Mal Sucedida

Caso o *login* e a senha tenham sido informados corretamente, o usuário está habilitado a consultar as bases de dados cooperantes. O sistema irá apresentar a página mostrada na Figura 5.8, com as opções **Consulta**, **Sobre** e **Sair**.

Figura 5.8: Menu Principal para Usuário

Ao pressionar o botão **Consulta**, o sistema irá apresentar a página mostrada na figura 5.9. Esta página permite escolher o curso que se deseja consultar.

Figura 5.9: Consulta de Curso

Após escolher o curso, será exibida uma página com as lições referentes ao curso escolhido, conforme a figura 5.10.

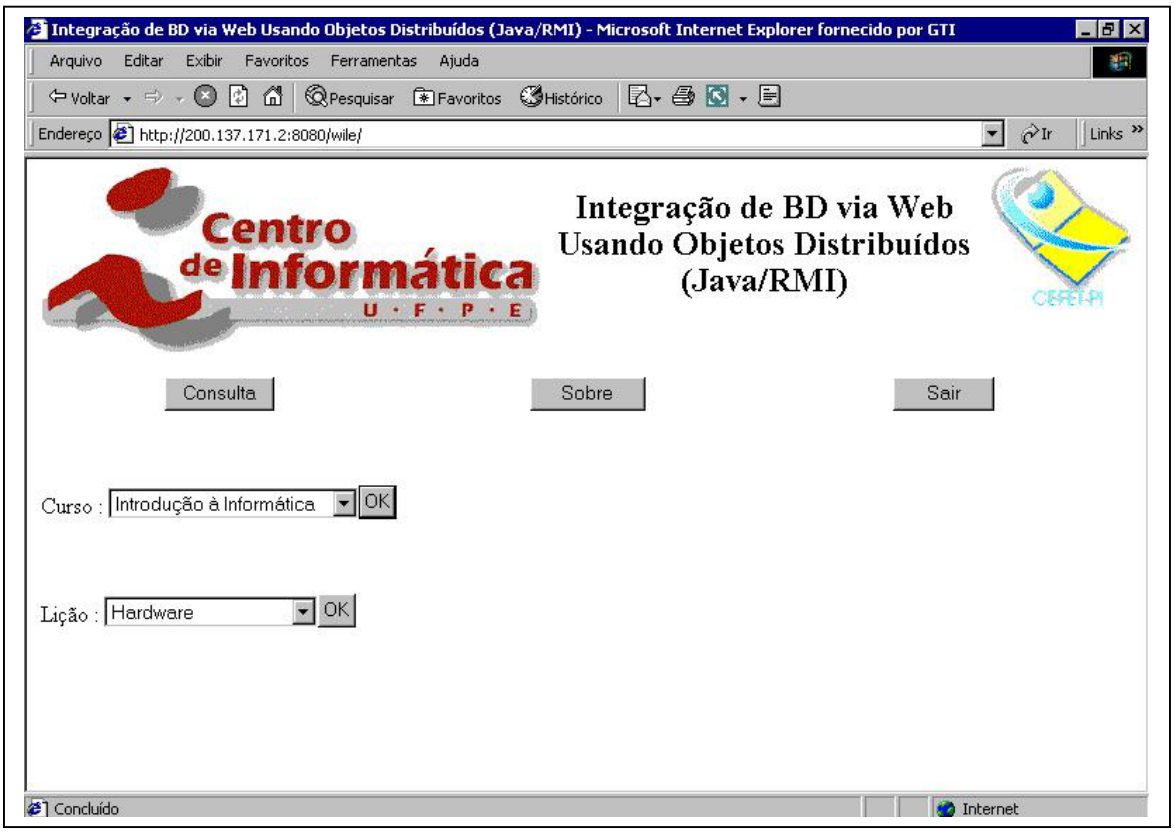

Figura 5.10: Consulta de Lição

Após escolher a lição, será exibida uma página com os tópicos referentes à lição escolhida, conforme a figura 5.11.

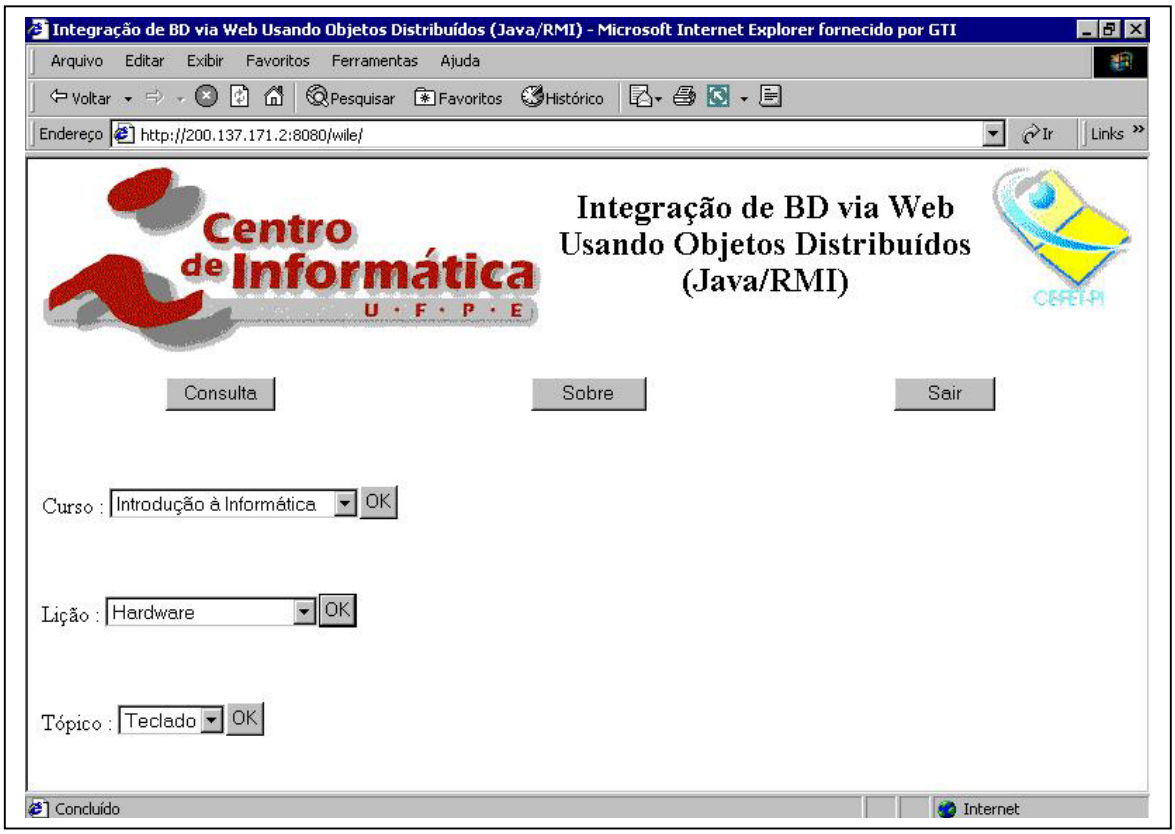

Figura 5.11: Consulta de tópico

Após escolher o tópico, será exibida uma nova página com o resultado da consulta feita nas bases de dados cooperantes. Este resultado pode ser uma página informando que o sistema não encontrou as bases de dados cooperantes conectadas no momento ou conteúdo relacionado ao tópico escolhido, conforme mostra a figura 5.12 ou uma página com o conteúdo armazenado nas bases de dados cooperantes conectadas no momento, referentes ao tópico selecionado para consulta, conforme mostra a figura 5.13.

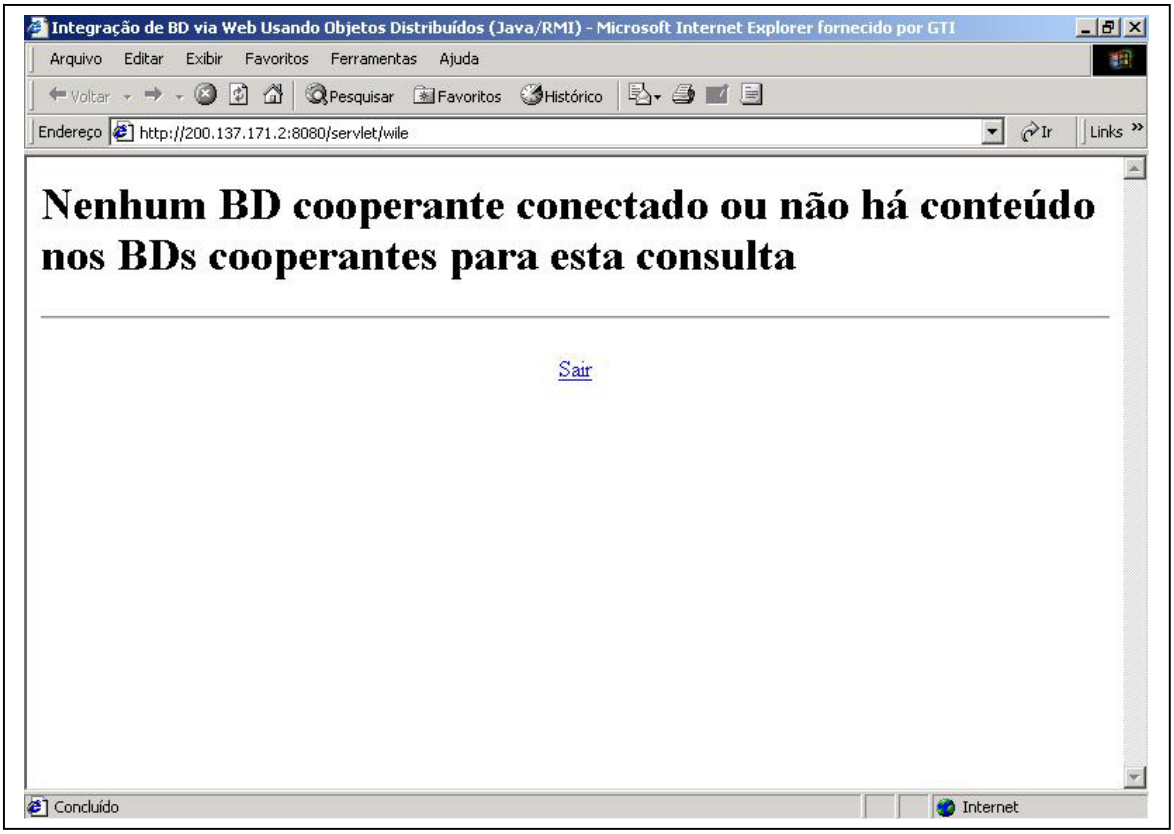

Figura 5.12: Resultado da Consulta Mal-Sucedida

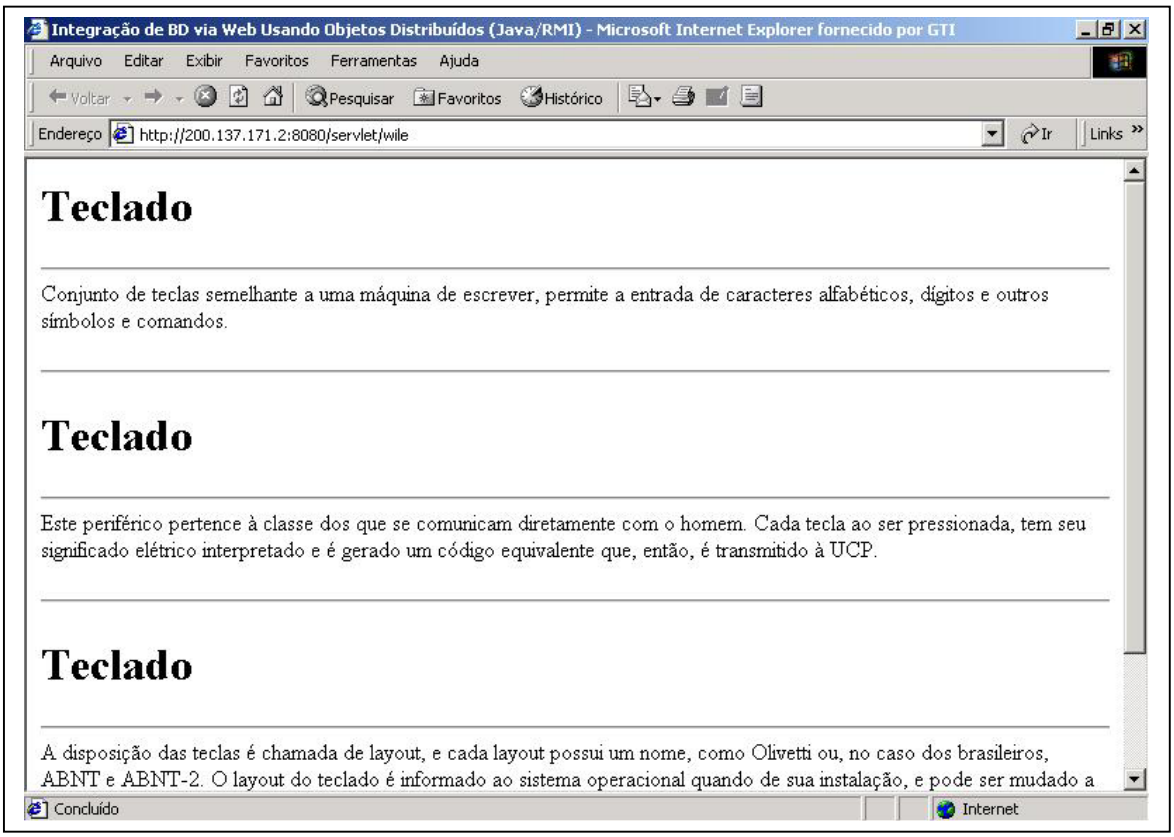

Figura 5.13: Resultado da Consulta Bem-Sucedida

Ao pressionar o botão **Sobre** será exibida uma página explicando como funciona o módulo de integração, conforme ilustra a figura 5.14. Tal funcionamento é explicado na seção 5.6.

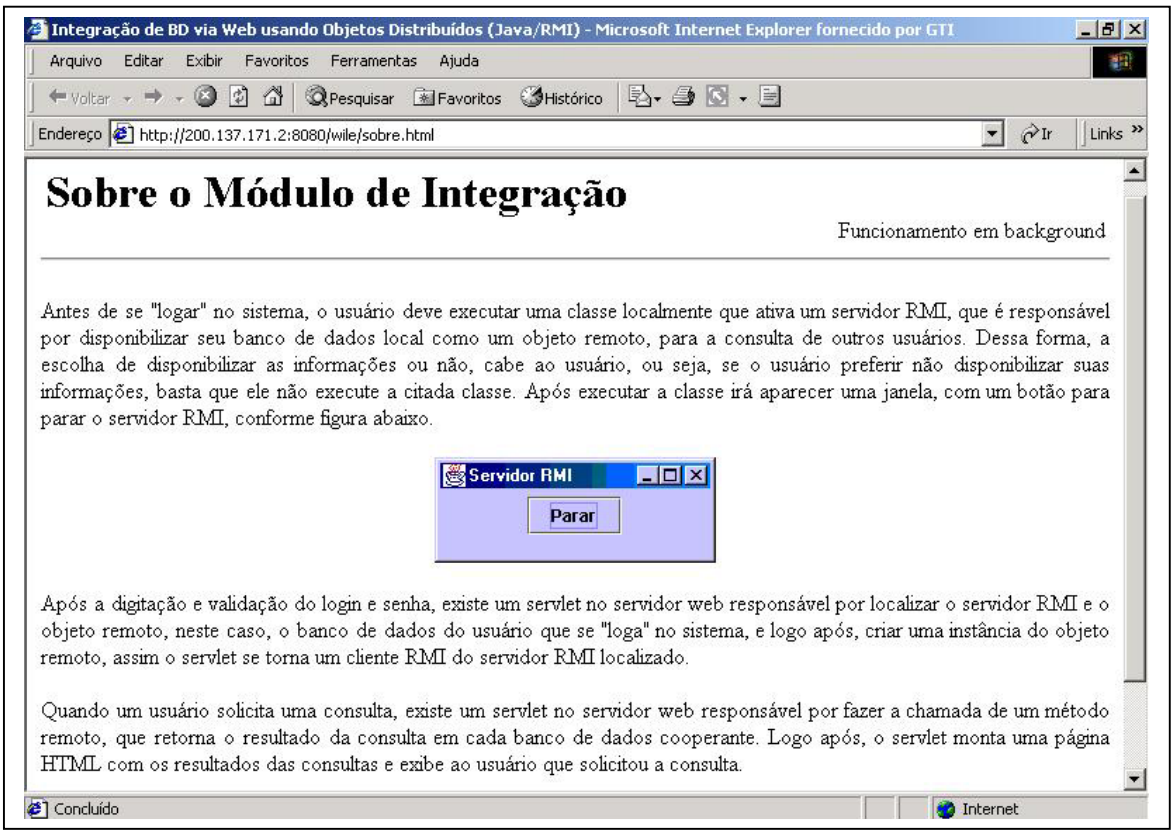

Figura 5.14: Informações sobre o Módulo de Integração

O botão **Sair** desconecta o usuário do sistema. Ao pressioná-lo, será apresentada a janela mostrada na figura 5.15. Esta página alerta o usuário para que desconecte o seu banco de dados, que foi disponibilizado antes de se "*logar*" no sistema.

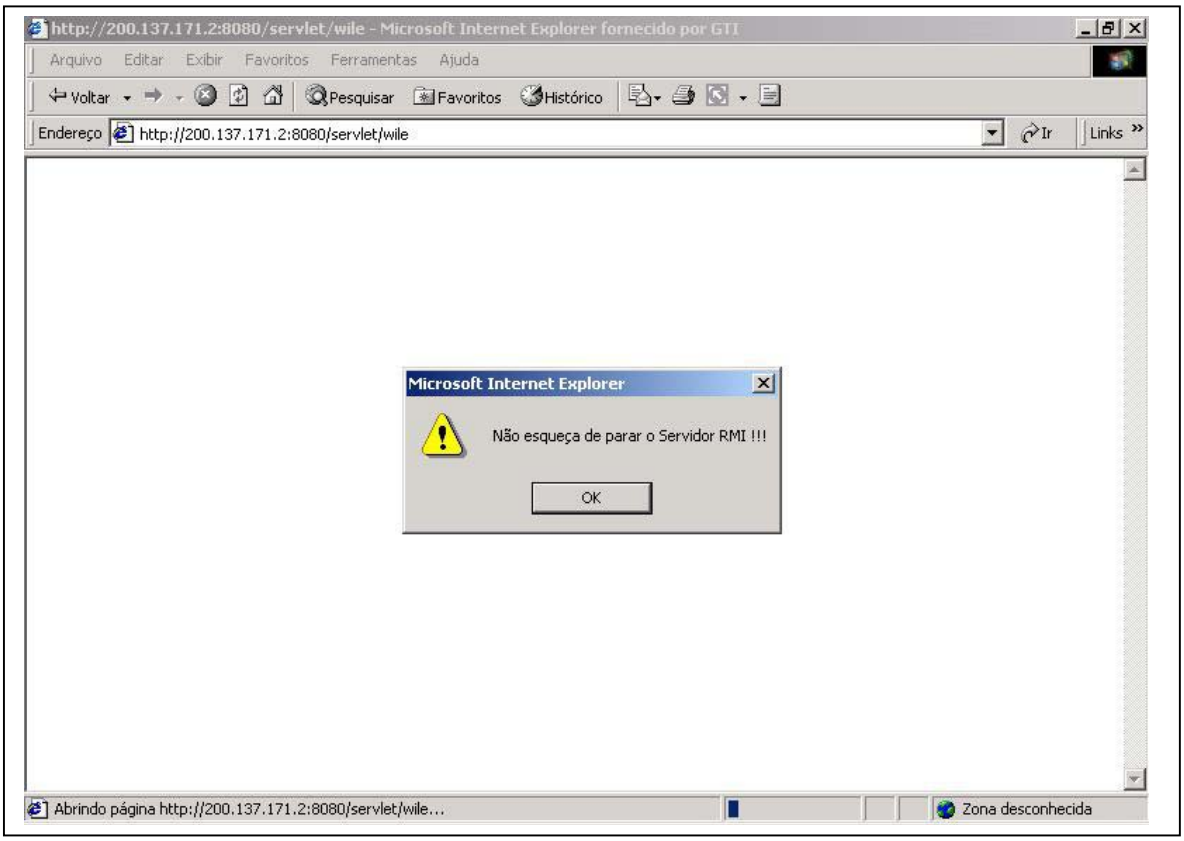

Figura 5.15: Usuário Desconectado do Sistema

5.6. Funcionamento em *Background*

É necessário, explicar como o módulo de integração funciona em *background*, já que para o usuário, o processamento da consulta é totalmente transparente.

Antes de se "*logar*" no sistema, o usuário pode executar uma classe localmente que ativa um servidor RMI, responsável por disponibilizar o banco de dados local como um objeto remoto, para a consulta de outros usuários. Dessa forma, a escolha de disponibilizar as informações ou não, cabe ao usuário, ou seja, se o usuário preferir não disponibilizar suas informações, basta que ele não execute a citada classe. Após executar a classe, irá aparecer uma janela com um botão para parar o servidor RMI, conforme mostra a figura 5.16.

Figura 5.16: Janela Ativa após Execução da Classe que Disponibiliza o BD Local

Após a digitação e validação do login e senha, existe um *servlet* no servidor *web* responsável por localizar o servidor RMI e o objeto remoto, neste caso, o banco de dados do usuário que se "*loga*" no sistema, e logo após, criar uma instância do objeto remoto, o *servlet* se torna um cliente RMI do servidor RMI localizado.

Quando um usuário solicita uma consulta, o supracitado *servlet* no servidor web faz a chamada de um método remoto, que retorna o resultado da consulta em cada banco de dados cooperante. Logo após, o *servlet* gera uma página HTML com os resultados das consultas e exibe ao usuário que solicitou a consulta.

5.7. Conclusão

O módulo de integração tem por objetivo atender os requisitos de cooperação e colaboração entre os estudantes e o professor em um ambiente de ensino-aprendizagem baseado na Web, fazendo com que vários bancos de dados integrados sejam vistos por usuários globais como um único banco de dados.

Quando um trabalho é feito de maneira cooperativa, o resultado aparenta ser bem melhor que quando é obtido de uma abordagem individual. Desenvolvendo atividades cooperativas, os estudantes trocam experiências e têm contato com outras possibilidades para resolver o mesmo problema.

O módulo de integração permite que qualquer usuário do sistema visualize, através do *browser*, as informações disponíveis de seu grupo de estudos. O tipo de informação que o usuário pode recuperar é limitado ao conjunto de funcionalidades disponibilizadas pelas páginas HTML, que são acessos à base de dados local de um estudante pelo tópico para recuperação das pesquisas realizadas pelo estudante.

No protótipo implementado foram, utilizadas as seguintes tecnologias:

- Java/JDBC – acesso aos bancos de dados;
- Java/RMI – disponibilização dos bancos de dados cooperantes;
- Java/Servlets – recebimento das solicitações das consultas, acesso aos bancos de dados, construção e envio das respostas aos estudantes solicitantes;
- HTML – confecção das páginas *Web*.

Capítulo 6

Conclusões e Trabalhos Futuros

Neste capítulo, serão apresentados os comentários conclusivos e destacadas as contribuições e sugestões de trabalhos futuros.

Capítulo 6
Conclusões e Trabalhos Futuros

A área de banco de dados está em crescente evolução. Modelos e sistemas de bancos de dados vêm se aperfeiçoando a cada dia, e a prova disso é a nova safra de Bancos de Dados Objeto Relacional [33]. A orientação a objetos [33] vem ganhando mercado rapidamente em todos os campos da ciência da computação, seja nas linguagens de programação com os avanços da linguagem Java [56], seja entre os projetos de sistemas com a evolução da UML (*Unified Modeling Language*) [1]. Aliar os recursos da orientação a objetos, sistemas distribuídos e arquitetura de software para *Web* apresenta soluções interessantes para as pesquisas na área de integração de dados.

As aplicações, a cada dia, incorporam mais características e requisitos dos sistemas voltados para Web. O grande desafio deste trabalho foi juntar as tecnologias emergentes da orientação a objetos e das aplicações distribuídas (os objetos distribuídos) e aplicá-las como solução de um problema real de tópicos de pesquisa em integração de dados.

A construção do módulo de integração usando tecnologias de várias áreas da ciência da computação tais como banco de dados, sistemas distribuídos, linguagens de programação e arquitetura de software para Web proporcionaram o desenvolvimento de um trabalho multidisciplinar.

6.1. Contribuições

Este trabalho apresenta as seguintes contribuições:

- **Desenvolvimento do Módulo de Integração de Banco de Dados**

O desenvolvimento do módulo de integração utilizou tecnologias, tais como Objetos Distribuídos, Banco de dados com a linguagem Java e *Servlets*. Os exemplos da utilização da interface JDBC podem ser utilizados para qualquer banco de dados e qualquer *driver* JDBC.

A aplicação prática das tecnologias selecionadas, principalmente a escolha da linguagem Java, e sua interação com banco de dados e sistemas distribuídos, serve de referência para outras aplicações de negócios e como soluções para outros problemas na mesma linha de pesquisa.

- **Ferramenta Especializada de Busca *On-Line* na *Web***

A metodologia empregada no desenvolvimento do módulo de integração de bancos de dados pode ser utilizada como uma ferramenta de busca de dados *on-line*, visto que a consulta é realizada apenas nos sites que estão conectados, diferentemente da maioria das atuais ferramentas de busca, que muitas vezes, retornam respostas de páginas inexistentes.

- **Integração de Bancos de Dados via *Web* para Fins Não-Educacionais**

A integração promovida pelo módulo de integração de bancos de dados pode ser utilizada para fins não educacionais.

6.2. Trabalhos Futuros

Como trabalhos futuros algumas extensões ao protótipo podem ser sugeridas, de modo a permitir novas contribuições ao estado da arte e validações de novos conhecimentos nas áreas e tópicos de pesquisa relacionados:

- Tratamento de heterogeneidades;
- Incorporação de consultas por palavras chaves;
- Testes com produtos CORBA e
- Testes do protótipo com Sistemas de Gerenciamento de Banco de Dados Orientada a Objetos (SGBDOO).

A avaliação do suporte a outras linguagens de consulta também é importante, apesar de SQL ser um padrão e da previsão da API JDBC dar suporte à nova especificação SQL 3 [36]. É necessário um estudo mais detalhado das vantagens e desvantagens de se utilizar essa arquitetura para SGBDOO em vez de outras linguagens de consulta como as que implementam o padrão ODMG.

A integração dos SGBDOO e SGBDR pode trazer uma complexidade adicional ao problema de tratamento de heterogeneidades. Um estudo mais detalhado para fazer essa integração é necessário.

A avaliação das linguagens de programação orientadas a objetos que implementem recursos de persistência de objetos, também deve ser considerada como trabalho futuro, para permitir uma análise das vantagens e desvantagens dessa abordagem.

Referências Bibliográficas

Serão enumeradas as referências bibliográficas.

Referências Bibliográficas

[1] ABITEBOUL, S.; BUNEMAN, P.; SUCIU, D. Data on the Web. 1st Edition. Editora Morgan Kaufmann Publishers, 2000.

[2] ALEXE, C.; GECSEI, J.; *A Learning Environment for the Surgical Intensive Care Unit*; Procc. of ITS96 - Frasson, Gauthier, Lesgold (eds), Montreal, pp439-446, jun 1996.

[3] ANDERSON, J.; BOYLE, C.; CORBETT, A.; LEWIS, M.; *Cognitive Modeling and Intelligent Tutoring*; Artificial Intelligence, Vol 42, nº 1, pp 7-49, 1990.

[4] APACHE JAKARTA PROJECT: http://jakarta.apache.org/. *Home Page* visitada em fevereiro/2003.

[5] APPLE COMPUTER INC. COMPONENT INTEGRATION LABORATORIES, INC. INTERNATIONAL BUSINESS CORPORATION, NOVELL INCORPORATED, Compound Presentation and Compound Interchange Facilities, Part I, OMD, December 1995.

[6] ARPANET, http://www.dei.isep.ipp.pt/docs/arpa.html. Home Page visitada em outubro/2002.

[7] ASANOME, C. ; *Sistemas Tutoriais Inteligentes: um Estudo*; Monografia de final de curso de Engenharia de Software, COPPE/UFRJ, 1991.

[8] AZEVEDO, B. F. T., "Tópicos em Construção de Software Educacional", Estudo Dirigido, Mestrado em Informática, Centro Tecnológico, Universidade Federal do Espírito Santo, 1997.

[9] BATINI, C. et. All, "*A comparative Analysis of Methodologies for Database Schema Integration",* In, ACM Computer Survey, Dez. 1986.

[10] BERTLS, K.; *A Dynamic View on Cognitive Student Modelling in Computer Programming*; Journal of Artificial Intelligence in Education, Vol 5, nº 1, pp 85-105, 1994.

[11] BOOCH, Grady; RUMBAUGH, James & JACOBSON, Ivar. UML: Guia do Usuário. Editora Campus. 2000.

[12] BRADSHAW, J. M. "An introduction to software agents" in BRADSHAW, J. M. Ed. Software Agents. MIT Press. Massachusetts, 1997.

[13] BRUFFEE, KENNETH A. The Art of Collaborative Learning: Making the Most of Knowledgeable Peers. Change. 26,3 : 39-44. 1994

[14] BRUSILOVSKY, P.*; Student as User: Toward an Adaptative Interface for an Intelligent Learning Environment*; Proccedings of AI-ED93 World Conference on Artificial Intelligence in Education, pp 386-393, 1993.

[15] BUSSE, S.; KUTSCHE, R. D.; LESER U.; WEBER H., "Federated Information Systems: concepts, terminology and architectures, Technical Report Nr. 99-9", TU Berlin, 1999.

[16] CAMPOS, V. V. de S., SOUZA, F. da F. de. Automatic Knowledge Construction in Intelligent Tutoring Systems Trough a Cooperative Approach. In: INTERNATIONAL CONFERENCE ON ENGINEERING AND COMPUTER

EDUCATION, 2000, São Paulo. Proceedings of International Conference on Engineering and Computer Education. 2000.

[17] CASTRO, Elizabeth. “HTML: programação para World Wide Web”. Editora Peachipit Press. 1997.

[18] CAVALLO, D.; *New Meaning for Learning*; Anais do VII Simpósio Brasileiro de Inteligência Artificial, SBC, 1991.

[19] CERI, S.; DE ANTONELLIS, V. “Deriving production rules for incremental view maintenance”. In Process of International Conference on Very Large Data Bases, pp. 577-589, 1991.

[20] *COMMOM OBJECT REQUEST BROKER ARCHITECTURE* (CORBA), http://www.omg.org/corba/. Home Page visitada em Junho de 2002.

[21] *COMPONENT OBJECT MODEL (COM),* http://www.microsoft.com/com. *Home Page* visitada em janeiro/2003.

[22] *COMPUTER SUPPORTED COLLABORATIVE LEARNING (*CSCL), **http://www.edb.utexas.edu/csclstudent/Dhsiao/theories.html**. *Home Page* visitada em fevereiro/2003.

[23] CORNELL, G. e HORSTMANN, C.S.: *CORE JAVA – Guia Autorizado da SUN Microsystems*, São Paulo, Makron Books, 1997.

[24] CORREDOR, M. V.; *La Inteligencia Artificial y la Educacion: lo aprendido y las futuras acciones*; Informática Educativa, Colômbia, vol 6, n. 3, pp 235-242, 1993.

[25] COSTA, R. M. E. M. da, XEXÉO, G. B., "A Internet nas escolas: uma proposta de ação", Anais do VII Simpósio Brasileiro de Informática na Educação, Belo Horizonte, p.105-118, 1996.

[26] COSTA, R. M. E. M. da; ROCHA, A.R.C. da; SANTOS, N. dos; WERNECK, V.M.B.; *Desenvolvimento de Sistemas Tutores Inteligentes: Questões e Perpectivas*, COPPE Sistemas e Computação, UFRJ, 1998.

[27] COSTA, R. M. E. M. da; WERNECK, V.M.B.; *Sistemas Tutoriais: Aplicações das Tecnologias de Hipermídia e de Inteligência Artificial em Educação*, COPPE Sistemas e Computação, UFRJ, 1998.

[28] CRESPO, A; BIER, E. A. "WebWriter: a browser-based editor for constructing Web applications." Fifth International World Wide Web Conference. Held: Paris, France, 6-10 May 1996. COMPUTER NETWORKS AND ISDN SYSTEMS (May 1996) vol.28, no.7-11, p. 1291-306.

[29] CRISTAL, Maurício de Oliveira. Uma Arquitetura Distribuída de Suporte a Centros de Otimização Cooperantes na Internet. Dissertação de Mestrado, Pontifícia Universidade Católica do Rio Grande do Sul, 2001.

[30] DAMASCENO JR., Américo. Java – Programação na Internet. Editora Érica. 1ª Edição. 1999.

[31] DATE, C. J. *An Introduction to Database System,* Addison-Wesley, 1995.

[32] DEITEL, H. M.; DEITEL, P.J. JAVA: How to program. 3a. ed., New Jersey, Prentice-Hall, 1999.

[33] DISTRIBUTED OBJECTS, http://www.cetus-links.org/. Home Page visitada em fevereiro de 2003.

[34] ELIOT, C.; WOOLF, B.; *Iterative Development and Validation of a Simulation-Based Medical Tutor*; Procc. of ITS96 - Frasson, Gauthier, Lesgold (eds), Montreal, pp 540-549, jun 1996.

[35] ELMAGARMID, A., et all, *Management of Heterogeneous and Autonomous Database System*, Morgan Kaufmann Publishers, 1998

[36] ELMASRI, Ramez, NAVATHE Shamkant B., "Fundamentals of Database Systems". 3rd ed, Addison-Wesley, 2000.

[37] ELORRIAGA, J.; FERNÁNDEZ-CASTRO, I.; *The HSIIP Approach. An Extension for a Teacher's Apprentice*; Procc. of ITS96 - Frasson, Gauthier, Lesgold (eds),Montreal, pp 401-410, jun 1996.

[38] FARJADO, F.R.; *Que puede aportar la Inteligência Artificial al desarrollo de la Informática Educativa ?*; Tese de Mestrado - COPPE Sistemas e Computação, UFRJ, 1995.

[39] FERRAZ, Carlos. Oliveira, Fabíola. Trinta, Fernando, et al. Uma aplicação distribuída para Educação à Distância na Web. X Simpósio Brasileiro de Informática na Educação-SBIE. Anais p. 248-255. Curitiba, 1999.

[40] FISCHETTI, E.; GISOLF, A.; *From Computer-Aided Instruction to Intelligent Tutoring Systems*; Educational Tecnology, pp 7-17, agosto de 1990.

[41] GANASCIA, Jean-Gabriel. Inteligência Artificial. Editora Ática. 1998.

[42] GEISSMAN, J.; SCHULTZ, R.; *Verification & Validation of Expert Systems, Knowledg-based Systems: Fundamentals and Tool*s, eds Garcia e Chien, IEEE Computer Society Press, 1992.

[43] GIRAFFA, L.M.M. "Uma arquitetura de tutor utilizando estados mentais". Porto Alegre: PGCC/UFRGS, 1999. (Tese de Doutorado).

[44] GLANZMANN, J.H.; *Expert Piano: Um Ambiente de Auxílio a Aprendizagem Musical*; Tese de Mestrado - COPPE Sistemas e Computação, UFRJ, 1995.

[45] GUPTA, A.; MUMICK, I.S. "What is the data warehousing problem ? (Are materialized views the answer ?)". In Process of the 22rd VLDB Conference, 1996.

[46] HADJEFTHYMIADES, S. MARTAKOS, D. I. A generic framework for the deployment of structured databases on the World Wide Web. 5th International World Wide Web Conference. Paris, France. 1996. http://www5conf.inria.fr/fich_html/paper-sessions.html

[47] HARRIS, E. L. V.; "Irc Survival Guide, The". Addison-Wesley, 1995.

[48] HELAL, A., *Research Issues in Heterogeneous Distributed Database System*, University of Texas, 1994.

[49] HERICKO, M; JURIC, M. ROZMAN, I.; DOMANJNKO, T; KRISPER, M. Java and Distributed Object Models: An Analysis. Eslovenia, 1998.

[50] HETHMON, Paul S; "Illustrated Guide To Http". Prentice Hall, 1997.

[51] HOSTMANN Cay S., CORNELL Gary, "Core Java 2 – Volume 1 – Fundamentos". São Paulo, Makron Books, 2001.

[52] HOSTMANN Cay S., CORNELL Gary; “Core Java 2 – Volume 2 – Recursos Avançados”. São Paulo, Makron Books, 2001.

[53] HULL, R. & ZHOU G. A framework for supporting data integration using the materialized and virtual approaches, Proc. of ACM SIGMOD Conference, 1996, pp. 481-492.

[54] HUNTER, Jason. “Java Servlet Programming”. O’Reilly & Associates, United States of America, CA, 1998.

[55] JAVA DATABASE CONECTIVITY: http://java.sun.com//jdbc/. Home Page visitada em fevereiro de 2003.

[56] JAVA: http://www.javasoft.com/. Home Page visitada em fevereiro de 2003.

[57] JEPSON, B.: *Dominando JAVA*. São Paulo, Makron Books, 1997.

[58] JONASSEN, D., MAYES, T., MCALESEE, R. Designing Environments for Constructive Learning. A Manifesto for a Constructivist Approach to Uses of Technology in Higher Education. 231-247. Springer-Verlag, Berlin. 1992

[59] KIM, W., *Modern Database System*, ACM Press, 1995.

[60] LANE D.; WILLIAMS H. E. Web Database Applications with PHP & MySQL. March 2002. Editora O'Reilly.

[61] LIFSCHITZ, Sérgio; LIMA Iremar Nunes de. Arquiteturas de Integração Web SGBD: um Estudo do Ponto de Vista de Sistemas de Banco de Dados. www.cecom.ufmg.br/~iremar/publicacoes/semish98html/semish98.htm. Home Page visitada em Junho de 2002.

[62] LIMA, Iremar Nunes de. Dissertação de Mestrado: O Ambiente Web Banco de Dados: Funcionalidade e Arquitetura de Integração. Departamento de Informática da PUC-RJ. 1997.

[63] LINDHOLM, Tim and YELLIN, Frank. The Java Virtual Machine Specification. Addison-Wesley, 1999.

[64] M.R. GENESERETH, A. KELLER, and O.M. DUSCHKA. *Infomaster: An Information Integration System*. In SIGMOD RECORD, Proceedings of the 97 ACM SIGMOD International Conference on Management of Data, pages 539-542, Tucson-Arizona, 1997.

[65] MAGALHÃES NETTO, J.; *Um Tutor Inteligente para o Ensino de Xadrez*; Tese de Mestrado do Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, 1995.

[66] MARCON Antonio M., NEVES Denise, *Aplicações e Banco de Dados para Internet.* São Paulo, Ed. Érica, 1999.

[67] MARK, M.; GREER, J.; *Evaluation Methodologies for Intelligent Tutoring Systems*; Journal of Artificial Intelligence in Education, 4(2/3), pp 129-153, 1993.

[68] MATHOFF, J. & Van Hoe, R.. APEALL: “A Multi-agent approach to interactive learning environments”. In: Perran, J & Müller J.(eds.) Distributed software agents and applications. European Workshop on Modeling Autonomous Agents MAAMAW'94, 6., 1994. Proceedings… Berlin: Springer-Verlag, 1996.

[69] McCALLA, G.*; Artificial Intelligence and Educational Technology: a Natural*

Synergy; Educational Multimedia an Hypermedia Proccedings of ED-MEDIA 94, Vancouver, pp 47-49, junho 1994.

[70] MELO, R. et al., *Banco de Dados em aplicações cliente servidor*, Infobook, 1998.

[71] MICROSOFT DCOM TECHNOLOGIES, http://www.microsoft.com/dcom/. Home Page visitada em Junho de 2002.

[72] MILNER, Robin. A Theory of Type Polymorphism in Programming. Journal of Computer and System Sciences. 17 (3): 348-375, December 1978.

[73] MOUSSALLE, N.M. & VICCARI, R.M. & CORRÊA, M. "Intelligent Tutoring Systems Modelled Through the Mental States". Advances in Artificial Intelligence. SBIA'95. Proceedings. 1995. Pg. 221-230.

[74] NELSON, B. J. Remote Procedure Call. PhD Thesis, Department of Computer Science. Carnegie-Mellon University, Pittsburgh, 1981.

[75] NUNES, M. das G.; TURINE, M.; MALTEPI, M.; HASEGAWA, R.; *Uso de Hipertexto/Hipermídia em Sistemas Tutores Inteligentes*; Notas Didáticas do Instituto de Ciências Matemáticas de São Carlos, SP, nº 9, março de 1993.

[76] OBJECT MANAGEMENT ARCHITECTURE, http://www.omg.org/library/omaindx.htm/, Home Page visitada em Junho de 2002.

[77] OBJECT MANAGEMENT GROUP, http://www.omg.org/ , Home Page visitada em Junho de 2002.

[78] OLIVEIRA, G. S. Uma Interface Independente para Acesso a Banco de Dados Heterogêneos. Dissertação de Mestrado, Universidade Federal de Pernambuco, 1999.

[79] OMG. OMG IDL syntax and Semantics, The Common Object Request Broker: Architecture and Specification, February 2001. Version 2.4.2.

[80] ORFALI, R.; HARKEY, D. CORBA. New York, NY: John Wiley and Sons Inc, 1998. 1022p.

[81] OZSU, M. TAMER; VALDURIEZ, PATRICK. Princípios de Sistemas de Bancos de Dados Distribuídos. Editora Campus. 2001.

[82] PAPERT, Seymour. *"A máquina das crianças*: Repensando a Escola na era da informática." Tradução de Sandra Costa. Porto Alegre. Artes Médicas, pp.210. 1994.

[83] PAULA FILHO, Wilson de Pádua. Multimídia: Conceitos e Aplicações. Editora LTC. 2000.

[84] PINTO, S.C.; *M-Assiste: Um Meta-Assistente Adaptativo para Suporte à Navegação em Documentos Hipermídia*; Tese de Mestrado do Programa de Engenharia de Sistemas e Computação, COPPE-UFRJ, dezembro de 1995.

[85] PRESSMAN, Roger S. Engenharia de Software. Makron Books. 1995.

[86] REGIAN, J.W.; SHUTE, V.; *Evaluating Intelligent Tutoring Systems*; Technology Assessment in Education and Training, Baker e O'Neil (Eds), Lawrence Erlbaum Associates, New Jersey, pp. 79-96, 1994.

[87] REINHARDT, B.; SCHEWE, S.A.; *Intelligent Tutoring Systems*; Proccedings of AI-ED95 World Conference on Artificial Intelligence in Education, Washington, 1995.

[88] REMOTE METHOD INVOCATION (RMI), http://java.sun.com/products/jdk/rmi/index.html Home Page visitada em Junho de 2002.

[89] RHOTON, John; "SMTP, X-400 and X-500 an Introduction". Digital Press, 1997

[90] RIGGS R., WALDO J., WOLRATH A. and BHARAT K.. A Distributed Object Model for the Java System. Computing System, 9 (4): 265-290, 1996.

[91] RIGGS R., WALDO J., WOLRATH A. and BHARAT K.. Pickling State in the Java System. In Proceedings of the USENIX 1996 Conference on Object-Oriented Technologies (COOTS), pages 241-250, Toronto, Ontario, Canada, 1996.

[92] SCARDAMALIA, M., BEREITER, C. Higher Levels of Agency for Childern in Knowledge Building: A Challenge for the Design of New Knowledge Media. The Journal of the Learning Sciences , Vol. 1, N. 1, 37-68. 1991.

[93] SOWE, Jeff. Construindo Servidores de Banco de Dados Internet com CGI. Traduzido por João Eduardo Nóbrega Tortello. Revisado por Roberto Gabriel Labrada. São Paulo. Editora Makron Books. 1998.

[94] SCHNEIDER, D., "Teaching & Learning with Internet Tools", 1994. http://tecfa.unige.ch/edu-ws94/contrib/schneider/schneide.fm.html

[95] SCHNEIDER, D., Block K., "The World Wide Web in Education", 1995. http://tecfa.unige.ch/tecfa/tecfa-research/CMC/andrea95/andrea.text.

[96] SELF, J. *Artificial Intelligence and Human Learning*. New York, NY. 1988.

[97] SERVLET: http://java.sun.com/products/servlet/. Home Page visitada em fevereiro de 2003.

[98] SHETH, A.P.; LARSON, J. A. *Federated Database System for Managing Distributed, Heterogeneous and Autonomous Databases.* ACM Computing Surveys, Set. 1990.

[99] SHUT, V.; *Regarding the I in ITS: Student Modeling*; Educational Multimedia and Hypermedia Proccedings of ED-MEDIA 94, Canadá, junho de 1994.

[100] SHÜTZ, Ricardo. *"A Review of Language Teaching Methodology."*, São Paulo, [on line]. 1997.

[101] SILVEIRA, R. "Modelagem orientada a agentes aplicada a ambientes distribuídos de ensino". Porto Alegre: PPGC da UFRGS, 1999.

[102] SLEEMAN D. and BROWN, J. S. *Intelligent Tutoring Systems*. Orlando, Florida: Academic Press, Inc. 1982.

[103] SOUZA, F.F. & CAMPOS, V.V.S. "WILE – Web based Intelligent teaching-Learning Environment". Proceedings of the ACIS International Conference on Computer Science, Software Engineering, Information Technology, e-Business and Applications – CSITCA '02. Foz do Iguaçu, 2002, páginas 385-390.

[104] TEDESCO, P. Dissertação de Mestrado: SEI - Sistema de Ensino Inteligente. Departamento de Informática da UFPE. 1997.

[105] THOMAS, G. et all, *Heterogeneous Distributed Database System for Producion Use*, ACM Computing Surveys, v.22(3), Set, 1990.

[106] THOMAS, M.D. et al.: *Programando em JAVA para a Internet.* São Paulo, Makron Books, 1997.

[107] TINTO, V., GOODSELL, A., RUSSO, P. Building learning communities for new students: A summary of research findings of the Collaborative Learning Project. National Center on Postsecondary Teaching, Learning, and Assessment. 1995.

[108] VICCARI, R., Giraffa, M. M. L.*; Sistemas Tutores Inteligentes: Abordagem Tradicional x Abordagem de Agentes*; XIII Simpósio Brasileiro de Inteligência Artificial, Sociedade Brasileira de Computação, Tutorial T6, 1996.

[109] VICCARI, R*.,* MOUSSALE, N.*; Tutores Inteligentes para o Ensino de Linguagem PROLOG*; Anais do I Simpósio Brasileiro de Informática na Educação, 1990.

[110] VICCARI, R.*; Inteligência Artificial e Educação: Indagações Básicas*; Anais do IV Simpósio Brasileiro de Informática na Educação, pp 207-216, 1993.

[111] VICCARI, R.; OLIVEIRA, F.; *Sistemas Tutores Inteligentes*; Monografia do Instituto de Informática – UFRGS, setembro de 1992

[112] WALDO, J. *Remote Procedure Calls and Java Remote Method Invocation* IEEE *Concurrency*, pages 5-7, July-September 1998.

[113] WALLNAU, K.; WEIDREMAN, N.; NORTHOP, L. *Distributed Object Technology With CORBA and Java: Key Concepts and Implications*. Technical Report CMU SEI-97-tr-004. Carniege Mellon University, Pennsylvania, 1997.

[114] WARREN, K.C.; GOODMAN, B.A.; MACIOROWSKI, S.M.; *A Software Architecture for Intelligent Tutoring Systems*; Proccedings of AI-ED93 World Conference on Artificial Intelligence in Educational, Edinburgh, pp 50-57, 1993.

[115] WEBOPEDIA, http://www.webopedia.com. *Home Page* visitada em janeiro/2003.

[116] WEINMAN, William E. "Manual de CGI". São Paulo, Makron Books, 1997.

[117] WIDERHOLD G. "Mediators in the architecture of future information systems". IEEE Computer, pp-38-49, 1992.

[118] WIDOM, J. "Research problems in data warehouse". In Process of the 4th International Conference on Information and Knowledge Management (CIKM), 1995.

[119] WINKELS, R.; BREUKER, J.; *Rational Reconstruction of Diagnostic Expertise*; *KADS: A Principled Approach to KBS Development*, Scheriber, Wielling and Breuker (ed), Academic Press, pp 314-336, 1993.

[120] WOOLF, B. P. "Intelligent Tutoring Systems: a Survey", in Exploring Artificial Intelligence, H. Schrobe and AAAI (eds), Morgan Kaufmann, 1988, pp 1-43.

[121] WOOLF, B.; *Intelligence Tutoring Systems*; In: Exploring Artificial Intelligence: Survey Talks from Natural Conferences on Artificial Intelligence, S. Howard (Ed). Morgan Kaufmann, 1988.

[122] WORLD WIDE WEB CONSORTIUM : http://www.w3c.org/. Home Page visitada em fevereiro de 2003.

[123] ZHOU, G.; HULL, R.; KING, R. “Generating data integration mediators that use materialization”. Journal of Intelligent Information Systems, vol. 6, nº 2/3, pp. 199-221, 1996.