UNIVERSIDADE FEDERAL DO MARANHÃO

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

CURSO DE PÓS-GRADUAÇÃO EM ENG. DE ELETRICIDADE

UMA METODOLOGIA PARA INTEGRAÇÃO DE APLICAÇÕES EMPRESARIAIS COM WEB SERVICES USANDO MDA

SAMYR BÉLICHE VALE

SÃO LUÍS

2004

UMA METODOLOGIA PARA INTEGRAÇÃO DE APLICAÇÕES EMPRESARIAIS COM WEB SERVICES USANDO MDA

Por

SAMYR BÉLICHE VALE

Dissertação de Mestrado submetida à Coordenação do curso de Pós-Graduação em Engenharia de Eletricidade da UFMA, como parte dos requisitos para obtenção do título de mestre em Engenharia de Eletricidade, com concentração em Ciência da Computação.

2004

UMA METODOLOGIA PARA INTEGRAÇÃO DE APLICAÇÕES EMPRESARIAIS COM WEB SERVICES USANDO MDA

MESTRADO

Área de Concentração: CIÊNCIA DA COMPUTAÇÃO

SAMYR BÉLICHE VALE

Orientador: PhD. ZAIR ABDELOUAHAB

Curso de Pós-Graduação

em Engenharia de Eletricidade

Universidade Federal do Maranhão

UMA METODOLOGIA PARA INTEGRAÇÃO DE APLICAÇÕES EMPRESARIAIS COM WEB SERVICES USANDO MDA

SAMYR BÉLICHE VALE

DISSERTAÇÃO APROVADA EM ___/ ___/ 2004

_____

Prof. PhD. Zair Abdelouahab
(Orientador)

_____

Prof. PhD. Carlos André Guimarães Ferraz
(Membro da Banca Examinadora)

_____

Prof. Dr. Francisco José da Silva e Silva
(Membro da Banca Examinadora)

Este trabalho é dedicado a Deus, minha esposa, irmão e pais.

“O coração do homem pode fazer planos, mas a resposta certa dos lábios do Senhor vem”. (Provérbios 16:1)

AGRADECIMENTOS

A Deus, por ter me conduzido até aqui debaixo de sua destra fiel.

A minha esposa, Magda, pelo amor e incentivo.

Ao meu irmão, Gustavo, companheiro inseparável, sempre disposto quando preciso de sua ajuda.

A meus pais, Jeronimo e Dinha, pelo amor, sustento e educação que me proveram ao longo da minha vida.

Ao Professor e Orientador, PhD. Zair Abdelouahab, pela orientação, apoio e incentivo a mim despendidos.

A Professora e Coordenadora da Pós-Graduação, PhD. Maria da Guia da Silva, pelo apoio e estímulo.

A todos os meus familiares e amigos pela compreensão nos momentos da minha ausência.

A todos os professores do Programa de Pós-Graduação em Engenharia de Eletricidade.

RESUMO

Este trabalho apresenta uma metodologia para Integração de Aplicações Empresariais com Web Services através da criação de meta-modelos em MDA. Esta metodologia abrange fases para o processo de integração e apresenta uma arquitetura baseada em *middleware* semântico para integrar aplicações legadas. O modelo MDA provê um padrão de integração reutilizável e interoperável a fim de permitir maior facilidade em futuras integrações. Os Web Services são sistemas identificados pela URL, com interfaces escritas em XML, que possuem uma estrutura de tecnologias que visam transformar a Internet em um grande repositório de serviços.

Por fim, é apresentado um estudo de caso real, utilizando a metodologia e arquitetura propostas para solucionar um problema de integração.

ABSTRACT

This thesis presents a methodology for Enterprise Applications Integration with Web Services through the creation of meta-models in MDA. This methodology contains phases for the integration process and presents an architecture based on semantic middleware to integrate legacy applications. The MDA model provides a pattern reusable and interoperable of integration in order to allow larger easiness in future integrations. The Web Services are systems identified by URL, with interfaces written in XML, that possess a structure of technologies that aim to transform Internet in a great repository of services.

Finally, a study of real case is presented, using the methodology and architecture proposed to solve an integration problem.

LISTA DE FIGURAS

LISTA DE SIGLAS

API	*Application Programming Interface*
ASP	*Active Server Pages*
B2B	*Business to Business*
B2C	*Business to Consumer*
CCF	*COMS Custom Connect Facility*
CGI	*Common Gateway Interface*
CLI	*Call Level Interface*
COM	*Component Object Model*
CORBA	*Common Object Request Broker Architecture*
DCE	*Distributed Computing Environment*
DCOM	*Distributed Component Object Model*
DMZ	*Demilitarized Zone*
DNS	*Domain Name Service*
DTD	*Document Type Definition*
EAI	*Enterprise Application Integration*
ERP	*Enterprise Resource Planning*
FTP	*File Transfer Protocol*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
GIOP	*General Inter-ORB Protocol*
IDL	*Interface Definition Language*
IIOP	*Internet Inter-ORB Protocol*
IIS	*Internet Information Server*

IP	*Internet Protocol*
J2EE	*Java 2 Enterprise Edition*
JDBC	*Java DataBase Connectivity*
JVM	*Java Virtual Machine*
MDA	*Model Driven Architecture*
MIDL	*Microsoft IDL*
MOM	*Middleware Orientado a Mensagem*
MQ	*Message Queuing*
ODBC	*Open DataBase Connectivity*
OMA	*Object Management Architecture*
OMG	*Object Management Group*
OMT	*Object Modeling Technique*
OOP	*Object Oriented Programming*
ORB	*Object Request Broker*
OSF	*Open Software Foundation*
PC	*Personal Computer*
PDF	*Portable Document Format*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
RMI	*Remote Method Interface*
RPC	*Remote Procedure Call*
RUP	*Rational Unified Process*
SGBD	*Sistema de Gerenciamento de Banco de Dados*
SGML	*Standard Generalized Markup Language*
SMTP	*Simple Mail Transfer Protocol*

SNA	*System Network Architecture*
SOAP	*Simple Object Access Protocol*
SSL	*Security Socket Layer*
TCP	*Transmission Control Protocol*
UDDI	*Universal Description, Discover and Integration*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
VB	*Visual Basic*
W3C	*World Wide Web Consortium*
WML	*Wireless Markup Language*
WSDL	*Web Service Description Language*
XML	*Extended Markup Language*
XSD	*XML Schema Definition*
XSL	*Extensible Style Language*
XSLT	*Extensible Style Language Transformation*

SUMÁRIO

1. INTRODUÇÃO

Aplicações corporativas são lógicas de negócios implementadas sob a forma de sistemas computacionais que rodam em plataformas empresarias. Uma grande parte destes sistemas implementam lógicas complexas que despenderam anos de desenvolvimento e testes para serem disponibilizadas em ambientes de produção, que necessitam de performance, segurança e confiabilidade.

Também, em sua maioria, foram implementadas em plataformas de grande porte (*mainframes*) e foram concebidas para ter uma vida útil longa, a fim de cobrir os custos do seu desenvolvimento.

No entanto, a disponibilização destes tipos de aplicações na Internet tem sido cada vez mais exigida, em decorrência do enorme crescimento do Comércio Eletrônico (*e-Business*) e das novas relações B2B (*Business to Business*) e B2C (*Business to Consumer*) estabelecidas através da Web.

Consumidores entusiasmados com a facilidade de efetivar transações e empresas ávidas por mais faturamento e à procura de novas parcerias, que possam lhes trazer maior rentabilidade, impulsionam o mercado da oferta e da procura na Internet. Portanto, incorporar-se a Web tem importância precípua para a sobrevivência das empresas, no âmbito mercadológico atual.

A necessidade de oferecer serviços na Web requer, na maioria das vezes, o desenvolvimento de novos sistemas ou até uma reengenharia completa dos sistemas legados existentes, o que indica um grande risco e um alto custo para empresas que têm, em seu ambiente computacional, sistemas confiáveis e seguros.

A solução empregada, na maioria dos casos, é a integração de aplicações, onde a aplicação existente fornecerá dados ou funções para a aplicação que será desenvolvida com suporte para Web.

Uma solução para integrar aplicações é a utilização do CORBA[1] [1], que tem sido o padrão mais difundido de arquitetura para integração de aplicações. Entretanto, apesar das facilidades, CORBA não foi projetado para dar suporte a aplicações HTTP[2] da Web, que sobretudo rodam sob a forma de *scripts* e que não permitem acesso direto a objetos da aplicação.

Os Web Services representam a maior evolução em tecnologia para a Internet. São aplicações que descrevem um conjunto de operações que são acessíveis através de mensagens XML[3] [2] padronizadas e que visam transformar a Web em uma rede de aplicações distribuídas.

Não existe uma metodologia padrão para estabelecer normas ou métodos ao processo de integração de aplicações, algo que defina quais são os procedimentos que devem ser tomados ao iniciar a integração entre duas ou mais aplicações, quais são os requisitos a serem analisados, qual o modelo de integração a ser utilizado, a arquitetura e quais ferramentas serão escolhidas para implementação do processo.

A maioria dos modelos existentes focalizam apenas detalhes técnicos da adaptação, sem levar em consideração os impactos no ambiente do negócio, além de serem modelos particulares a problemas específicos. O objetivo da presente dissertação, portanto, é a elaboração de uma metodologia para integração de aplicações empresariais legadas com a web através da tecnologia de Web Services.

---

[1] CORBA (Common Object Request Broker Architecture).
[2] HTTP (HyperText Transfer Protocol)
[3] Extended Markup Language.

A metodologia proposta fornece um fluxo lógico de fases que envolvem o processo, baseados no novo padrão de desenvolvimento de aplicações proposto pela OMG[4], o MDA (Model Driven Architecture) [3].

Este trabalho também propõe um modelo de arquitetura de integração de aplicações empresariais com Web Services, que proporcione as Empresas, que necessitam migrar lógicas de negócios para Internet, uma solução que apresente interoperabilidade e capacidade de reutilização.

A arquitetura também visa fornecer maior flexibilidade de adaptação de sistemas legados com a web através do uso de *middlewares* semânticos, o que tornará a integração extensível a futuros parceiros de negócios.

A solução permitirá a integração sem que haja manipulação no código dos sistemas empresariais, proporcionando segurança na integração, uma vez que se a aplicação Web desenvolvida não satisfizer as necessidades da Empresa ou se a tecnologia Web escolhida não proporcionar segurança, performance e confiabilidade, esta poderá ser desvinculada, sem causar problemas ao sistema legado.

Outra vantagem do modelo é que o ambiente de produção não sofre interrupções, uma vez que o software empresarial não necessita ser recompilado, durante o processo da integração.

Propõe-se a utilização dos Web Services nesta arquitetura, pois, além de ser uma tecnologia bastante recente, ela vem permitir interoperabilidade, suporte a integração, desenvolvimento de componentes distribuídos e disponibilidade de serviços semelhantes ao do padrão CORBA.Para garantir que os Web Services, usados na integração, sejam extensíveis, ou seja, que possam ser utilizados em futuras integrações, destaca-se o MDA (Model Driven Architecture). O MDA vem estabelecer modelos que

[4] OMG - Object Management Group

reconhecem que os investimentos das empresas não podem ser desperdiçados com mudanças de tecnologias.

Através de modelos abstratos independentes de plataforma, propostos pelo MDA, permitir-se-á que a lógica implementada pelos Web Services possa ser reutilizada por outras empresas que possuam necessidade de integração de sistemas, na mesma área de domínio, ou possam ser extensíveis para integração com novas tecnologias que estão por surgir.

Os capítulos iniciais deste trabalho apresentam algumas tecnologias utilizadas para a proposição da metodologia e da arquitetura, contudo somente são abordados os detalhes que permitem a contribuição com a integração de aplicações empresariais com Web Services.

O capítulo 2 apresenta um estudo sobre a Integração de Aplicações Empresariais (EAI – *Enterprise Application Integration*), uma área com bastante ênfase nos últimos anos devido à necessidade de crescimento e avanço tecnológico que os softwares das empresas têm sofrido. São apresentados neste capítulo os tipos e modelos de integração existentes e as tendências para o futuro.

O capítulo 3 apresenta o elemento pivô na Integração de Aplicações, que é o *middleware*. São relatados os tipos de *middleware* e a forma como provêem comunicação entre dois ou mais sistemas. Também são abordados conceitos de sistemas distribuídos e da tecnologia de objetos distribuídos, bem como, algumas tecnologias de integração de sistemas empresariais com a Web.

O capítulo 4 apresenta a linguagem XML como solução de *middleware* semântico para integração de aplicações. O mesmo capítulo aborda a tecnologia de Web Services que vem propor uma revolução no conceito de relacionamento entre aplicações Web.

O capítulo 5 discorre sobre a Arquitetura Direcionada a Modelos (MDA – *Model Driven Architecture*). O MDA propõe a criação de modelos interoperáveis e adaptáveis para construção de softwares.

O capítulo 6 apresenta a metodologia proposta para integração de aplicações empresariais com Web Services usando MDA. Dentro da metodologia é proposta uma arquitetura que provê comunicação entre sistemas legados com Web Services através do uso do XML/SOAP como *middleware* semântico suportado por *firewalls*.

O capítulo 7 apresenta um estudo de caso real, realizado em uma instituição governamental. Foram aplicadas, a metodologia e arquitetura propostas para possibilitar o acesso a funções de aplicativos escritos na linguagem COBOL, implementados em *mainframe*, através de Web Services.

No oitavo e último capítulo é relatado a conclusão sobre o trabalho proposto, bem como são apresentadas sugestões para trabalhos futuros.

2. INTEGRAÇÃO DE APLICAÇÕES EMPRESARIAIS – EAI (*ENTERPRISE APPLICATION INTEGRATION*)

A Integração de Aplicações Empresarias(EAI) oferece uma solução para a urgente necessidade dos negócios atuais: integrar sistemas legados dispostos em diferentes plataformas, sistemas operacionais, tecnologias de rede, e desenvolvidos em diferentes técnicas e linguagens de programação. EAI é responsável pelo compartilhamento de dados e processos de negócios entre quaisquer aplicações dentro da empresa e entre empresas, contudo, sem realizar grandes alterações nas aplicações ou estruturas de dados.

Por décadas sistemas foram desenvolvidos com necessidades específicas, o simples propósito de solucionar problemas e automatizar processos, utilizando tecnologias que estavam disponíveis na época do desenvolvimento[42]. Não existia a preocupação com os avanços tecnológicos ou sequer era levantada a possibilidade de uma necessidade futura de integração com outras aplicações.

A maioria destas aplicações se tornaram obsoletas, mas são responsáveis por trabalhos críticos dentro das grandes empresas e instituições. Apesar disto, são aplicações robustas e seguras, que passaram por refinamentos sucessivos, exaustivos testes, além de estarem em produção há bastante tempo.

Reescrever aplicações desta natureza, apenas para aplicar tecnologias recentes, é uma tarefa perigosa e bastante onerosa. A contratação de uma equipe de experientes analistas e programadores seria necessária, investimento com a aquisição de ferramentas, tempo para desenvolvimento, testes e implementação. Todo este aparato, por si só, ainda não garante a mesma segurança e o mesmo desempenho. Estes sistemas legados geralmente implementam lógicas complexas, em sua maioria foram escritos em

linguagens estruturadas e rodam em máquinas de bom desempenho, alguns em *mainframe*. São softwares de CRM, aplicações bancárias, folhas de pagamento, sistemas corporativos de domínio específico, etc. Construir sistemas semelhantes, a partir do zero, para fazer *downsizing*, implementar técnicas de orientação a objetos, disponibilizar clientes na Web, prover suporte à plataforma TCP/IP, rodar em Windows ou Linux, e quaisquer outra exigência do mercado atual, abrirá brechas para possíveis problemas de desempenho, confiabilidade, segurança, adaptação de usuários, dentre outros.

O EAI visa a criação de arquiteturas estratégicas para atender algumas das necessidades naturais da evolução do desenvolvimento de softwares, a extensibilidade e a interoperabilidade, sem a necessidade de reescrever aplicações.

Integrar aplicações, compartilhar dados, integrar novos processos, permitir crescimento de funcionalidades em aplicações robustas já existentes, tem sido a melhor solução para aumentar a sobrevida dos sistemas existentes nas empresas.

A integração não é uma tarefa fácil, existem diversas tecnologias que se propõem a solucionar problemas desta natureza e existem, também, diversos tipos diferentes de integração.

Algumas tecnologias tradicionais usadas são: o uso de *middlewares*, *message-queuing*, soluções ponto a ponto (*links*), chamada a procedimentos remotos (RPC[5]).

Antes de pensar em integração, a empresa necessita entender os processos e os dados que existem, analisar os objetivos da integração, quais sistemas integrar-se-ão, para que sejam escolhidos: a melhor arquitetura, metodologia e procedimentos.

[5] RPC – Remote Procedure Call

Pequenas aplicações concentradas no mesmo ambiente são mais fáceis de serem integradas, mas uma organização que possua diferentes aquisições de software, grandes sistemas distribuídos e centenas de usuários requerem estudo mais aprofundado antes de começar a integração dos mesmos. Por isso, o sucesso da EAI é analisado caso a caso, empresa a empresa. Se a integração não propõe um ganho significante de produtividade, deve-se considerar o caso da substituição da aplicação.

2.1 Tipos de EAI

Os proprietários dos softwares têm que selecionar quais processos e dados requerem integração. Este processo pode abranger diversas abordagens. As aplicações em EAI variam consideravelmente, e é possível considerar algumas categorias gerais [4]:

- Orientado a Informação;
- Orientado a Integração de Processos de Negócios;
- Orientado a Serviços;
- Orientado a Portais.

2.1.1 Orientado a Informação

É uma abordagem de integração que envolve a troca de informações entre diferentes SGBDs (Sistemas de Gerenciamento de Banco de Dados). Isto descreve a extração da informação de um banco de dados, talvez processando a informação (quando necessário) e a inserção da informação em outro banco de dados. Geralmente, isto significa que algumas centenas de bases de dados e milhares de tabelas irão passar

pelos processos de extração e carga. A vantagem deste tipo de abordagem é o baixo custo, pois não necessita manipulação de código e conseqüentemente evita gastos com desenvolvimento, testes e implementações. Esta abordagem é utilizada em *Data Warehouses*.

A solução orientada a informação é agrupada em três categorias: replicação e dados, federação de dados e processamento de interfaces.

- Replicação de Dados

É simplesmente o movimento dos dados entre diferentes bases de dados. Estas bases podem ser do mesmo fabricante ou não, podem inclusive ser bases de diferentes modelos. O requisito fundamental, na replicação de base de dados, é que haja uma infraestrutura que possibilite a troca de dados entre diferentes modelos e esquemas.

- Federação de Dados

É a integração de múltiplas bases de dados e modelos de bases de dados em apenas uma só base de dados.A federação manipula a coleção dos dados, criando um banco de dados físico ou apenas virtual para a extração da informação requerida. Esta é uma solução mais elegante para o problema da integração de aplicações orientadas a troca de dados. Como na replicação, também não há a necessidade da mudança ou adaptação de códigos fontes.

- Processamento de Interfaces

É o processo de desenvolver interfaces bem definidas que focalizem a integração de pacotes de aplicativos. Usam-se adaptadores para estabelecer a conexão com a aplicação e para exportar as interfaces proprietárias destas aplicações.

2.1.2 Orientado a Integração de Processos de Negócios

É o tipo de integração de aplicações que define um conjunto de processos de gerenciamento central no topo de um conjunto de processos existentes dentro das aplicações empresariais.

A integração de processos de negócios é o mecanismo de gerenciamento da troca de dados e da invocação de processos, em uma ordem relacionada, para permitir o gerenciamento e a execução dos processos que existem em e entre aplicações. O objetivo desta técnica é unir processos de uma empresa ou negócio para obter um maior fluxo de informações e um maior controle entre eles. Na realidade é a inserção de mais uma camada na lógica da aplicação, que geralmente envolve servidores de aplicação, integração, *middlewares*, etc.

O agrupamento de *middlewares* e ferramentas de automação de processos de negócios representa o futuro da integração de aplicações [4].

Este tipo de integração deve ser flexível, além de prover uma camada de tradução entre os sistemas e uma máquina de integração.

As principais diferenças entre aplicações mais tradicionais e a integração de processos de negócios são:

- Uma simples instância de integração de processos de negócios abrange muitas instâncias da integração de aplicações tradicionais.
- Integração de Processos de Negócios cria um modelo de processos e transporta informações entre aplicações que suportam tal modelo.
- Integração de Processos de Negócios é estratégico, utiliza-se de regras de negócios que determinam como sistemas podem interagir e agregam a cada sistema um modelo abstrato de negócios.

Muitos fabricantes têm investido na construção de ferramentas (servidores de integração e aplicação) que permitam integração de aplicações, utilizando *middlewares* já existentes, aplicando regras próprias, seqüências lógicas de migração de informações, compartilhamento de processos, visualização e a criação de um processo abstrato comum que abranja sistemas internos e externos.

Destaca-se em todo este processo de integração, três serviços principais: a visualização dos processos contidos nos sistemas legados, a abstração das interfaces e medidas de performance dos processos integrados. A utilização de cenários, gráficos e diagramas é muito importante para capacitar os participantes do processo na visualização dos pontos de integração e na escolha do *middleware* a ser utilizado. Isto, também facilita o processo de mudanças no modelo do processo.

A abstração da interface refere-se ao mapeamento do modelo de integração do processo do negócio em uma interface física abstraindo a conectividade e a solução de integração. Como mencionado anteriormente, a integração do processo de negócio situa-se no topo da pilha do *middleware* utilizado na integração.

A medida da performance é feita através de uma análise do negócio após a integração, em tempo real e no âmbito da produtividade, através de estatísticas realizadas entre toda a comunidade atingida pelo sistema.

2.1.3 Orientado a Serviços

Integração de aplicação orientada a serviços permite que aplicações compartilhem lógicas ou métodos de negócios comuns. Métodos são implementações de operações de objetos. O termo operação aplica-se à visão externa de um pedaço de código, enquanto o termo método aplica-se à visão interna [5].

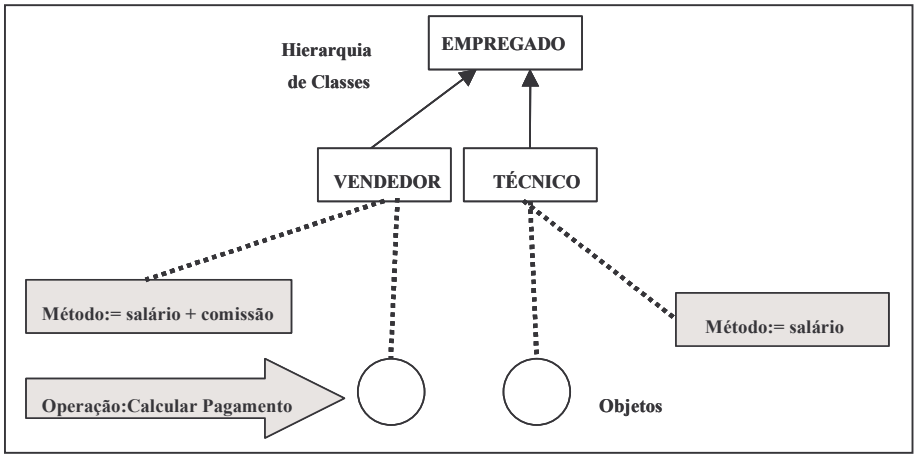


Figura 2.1 – Paradigma Orientado a Objetos

Por exemplo, na Figura 2.1, a operação calcular pagamento é realizada para todos os empregados, contudo o cálculo (método) é diferente para técnicos e vendedores.

Pela definição métodos podem ser compartilhados para acesso entre aplicações ou através de mecanismos alternativos, como Web Services, por exemplo.

O compartilhamento de processos iniciou-se quando do surgimento das aplicações cliente/servidor e multicamadas, onde um conjunto de serviços disposto em um servidor pode ser acessado por diversos clientes, portanto integrados.

Em EAI objetiva-se expandir este compartilhamento entre empresas, por exemplo, compartilhar uma lógica comum para processar requisições de crédito de clientes ou para calcular custos de entrega, usando um conjunto de Web Services. Isto constitui uma grande vantagem, pois o processo de integração além de promover o compartilhamento de dados, oportunamente poderá compartilhar métodos comuns em lógicas de negócios semelhantes.

Apesar das vantagens deste tipo de integração, manipulação de códigos, como dantes mencionado, é cara e perigosa. A melhor solução para a abordagem orientada a serviço é a utilização de tecnologias, como o CORBA (*Common Object*

Request Broker Architecture) ou Microsoft .Net, esta é a última arquitetura baseada em serviço lançada no mercado.

2.1.4 Orientado a Portal

Integração de Aplicações Orientada a Portal permite a visualização de uma multidão de sistemas – tanto interno à empresa, quanto dos parceiros de negócios, através do desenvolvimento de uma simples interface web.

O principal benefício desta abordagem é que ela evita o problema da integração dos sistemas *back-ends* ou de retaguarda, e adapta a interface do usuário de cada sistema em uma interface comum (agregada), disponível para acesso através da Internet. Como resultado, todos os sistemas participantes integram-se através do browser.

Enquanto os outros tipos de integração focam na troca de informações entre sistemas e empresas, a abordagem orientada a portal refere-se à exteriorização da informação para uma interface na web. A informação disponibilizada na web por uma determinada aplicação é capturada por outra que necessite desde dado. Contudo, a integração de aplicações que se refere ao movimento automático de informação ou ao agrupamento de processos entre duas ou mais aplicações, sem a intervenção do usuário final, pode também ser realizada na orientada a portal.

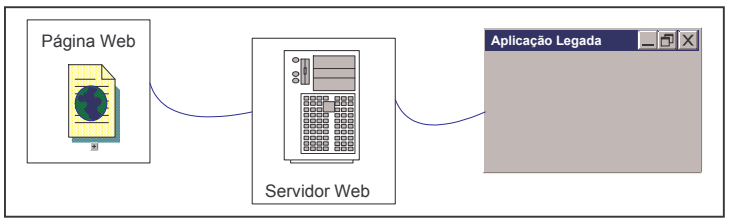


Figura 2.2 – Integração de uma aplicação legada com a Web

A aplicação do portal também deve passar pelos requisitos tradicionais de análise e projeto, pois será responsável pelo controle da interface do usuário, captura e processamento de erros, controle de transação e acesso à base de dados.

Existem muitas tecnologias disponíveis para a criação de portais e diversos tipos de servidores de aplicação que provêem o ambiente para o desenvolvimento das interfaces e conectores de *back-end*, inclusive de suporte a *mainframe*. Entretanto não integram as aplicações diretamente, a interface web apenas exterioriza a informação para o parceiro de negócio através de uma página HTML.

Uma vantagem bastante significativa dos portais é que suas aplicações podem trafegar sem problemas através de *firewalls* em virtude da utilização do protocolo HTTP. As interfaces web apenas se conectam ao *back-end*, em um ponto de integração e obtém a informação requisitada. Todavia, a lógica de conexão com banco de dados fica armazenada no servidor de aplicação.

Como desvantagem, sita-se: a informação não trafega sem a intervenção do usuário e existe a necessidade do desenvolvimento de uma camada adicional de aplicação a ser inserida no servidor de aplicação.

A arquitetura orientada a Portal apresenta alguns componentes básicos, a saber:

- Cliente Web:

Cliente Web é um PC ou qualquer dispositivo que rode um Web *browser* e que seja capaz de apresentar páginas HTML. O web browser faz requisições ao servidor web, que processa os dados e retorna os resultados requeridos ao cliente.

- Servidor Web:

É o núcleo deste tipo de integração. Eles recebem as requisições dos clientes, fazem consultas a bancos de dados, executam diretivas de segurança, convertem resultados em páginas HTML e as encaminham ao web browser.

- Aplicações *Back-end*:

São aplicações existentes nas empresas, geralmente robustas e estáveis. Em determinadas empresas constituem-se em uma mistura de sistemas e plataformas diferentes. Também se enquadram aqui, as aplicações proprietárias ou sistemas legados, desenvolvidos sob encomenda, que geralmente atendem necessidades específicas da empresa. Implementam lógicas complexas, apresentam em seus fontes milhares de linhas de código, rodam em *mainframes* ou grandes servidores e são os mais difíceis de se integrarem a outros sistemas.

Os portais devem adquirir a informação apropriada destas aplicações e fornecer o resultado na interface do usuário.

Se estes tipos de aplicações existirem no processo de integração, a empresa deve investir em adaptadores ou *middlewares*, para que seja estabelecida a comunicação entre aplicações, além de um conjunto de APIs (*Application Programming Interface*) para permitir a agregação dos sistemas legados com a tecnologia web.

- Servidores de Aplicação

Servidores de Aplicação constituem uma camada adicional e intermediária entre aplicações *back-end*, bancos de dados e servidores web.

Na integração orientada a portal, os servidores de aplicação comunicam-se com o servidor web através de transações. Geralmente são utilizados para a aplicação

da lógica de negócios mais complexa e implementação de requisitos adicionais não providos pelo sistema legado, como: tolerância à falhas, segurança, balanceamento de carga, dentre outros.

Por exemplo, se uma determinada aplicação legada necessita integrar-se com a Web, é necessário que haja uma estimativa do número de acessos de clientes web, para que seja feito um perfeito balanceamento do acesso ao banco de dados. Além disso, é necessária a criação de mecanismos de segurança de *logon/logoff* para Web e mecanismos de *rollback* (para que a informação seja retornada ao estado original em casos de interrupção no momento da transação). Estas implementações geralmente são efetivadas nos servidores de aplicação.

A metodologia proposta por este trabalho, utiliza características de alguns dos tipos de integração mencionados. Contudo, extrai de cada tipo de integração noções relevantes, propondo melhoramentos, para suprir as necessidades da tendência dos cenários de integração.

3. MIDDLEWARE E TECNOLOGIA DE OBJETOS DISTRIBUÍDOS

Para integrar duas aplicações, mesmo que seja um cliente e um servidor, é necessário estabelecer uma comunicação entre elas, para acessar um serviço, uma informação em um banco de dados remoto ou uma operação de um objeto.

O mecanismo que permite uma entidade (aplicação ou banco de dados) comunicar-se com outra é o *middleware*. Em outras palavras, *middleware* é o software que estabelece a comunicação entre dois ou mais softwares.

Este mecanismo possibilita tanto a integração de aplicações dentro de uma empresa quanto entre empresas.

O objetivo do *middleware* é esconder a complexidade do processo de integração. Os programadores ficam isentos de implementar a programação de baixo nível, como APIs e protocolos de rede, permitindo a concentração no compartilhamento da informação.

Outra vantagem desta tecnologia é que o mesmo *middleware* pode ser reutilizado para diferentes tipos de aplicações que podem estar rodando em diferentes arquiteturas e plataformas. Hoje, *middleware* é o que existe de mais simples e eficaz para solucionar o problema de integração entre aplicações empresariais.

Existem dois modelos principais de *middleware*: ponto-a-ponto e multi-ponto:

- *Middleware* ponto-a-ponto:

Usa um canal para permitir a ligação entre duas aplicações. Quando uma aplicação A deseja se comunicar com uma aplicação B, esta apenas deposita a mensagem no canal de comunicação, conforme apresentado na figura 3.1.

Comparado com outros tipos, a incapacidade de integrar mais de duas aplicações é uma grande limitação do *middleware* ponto-a-ponto.

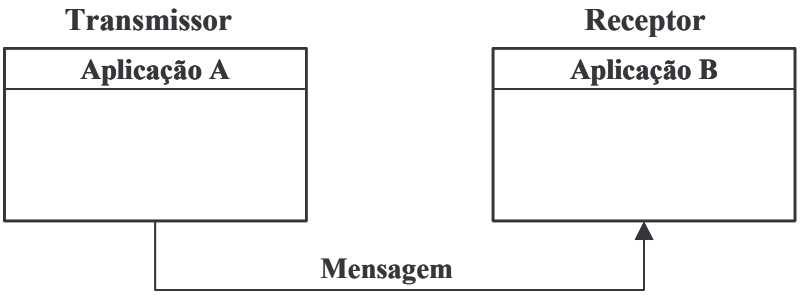


Figura 3.1 – Esquema do *Middleware* ponto-a-ponto

- *Middleware* Multi-ponto:

São *middlewares* que integram diversas aplicações. Esta capacidade os torna a melhor opção de integração de aplicação, desde que venham permitir flexibilidade e aplicabilidade em diversos domínios de aplicações.

Existem muitos exemplos de *middleware* multi-ponto que incluem: servidores de integração, *middlewares* de transação e objetos distribuídos.

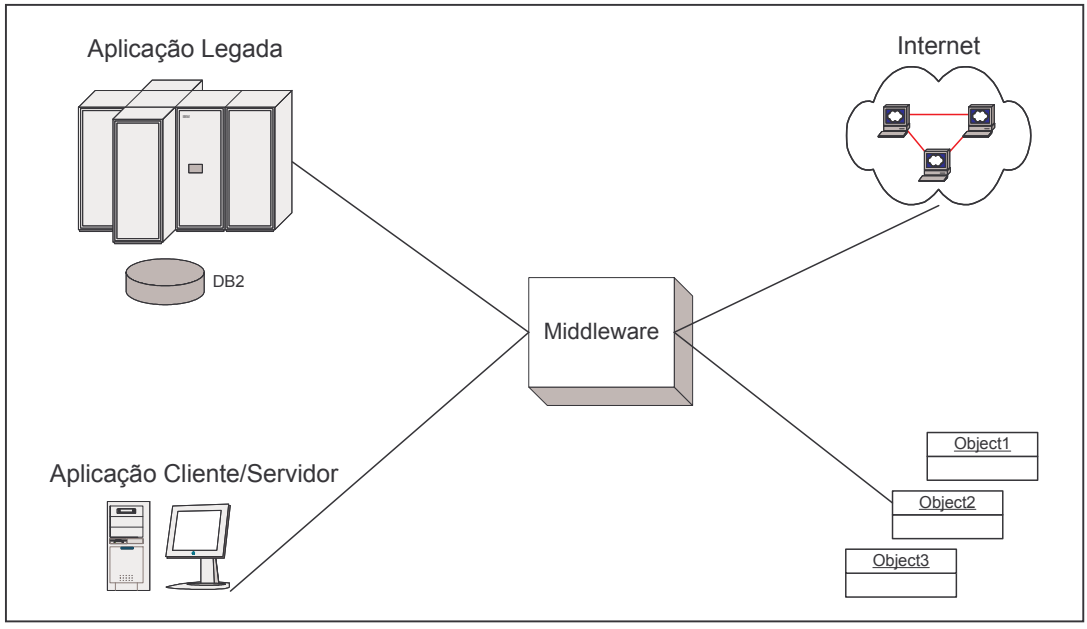


Figura 3.2 – Esquema de um *Middleware* Multi-ponto

Estes *middlewares* utilizam um padrão de comunicação que permitem aplicações implementadas em diferentes linguagens e arquiteturas integrarem-se, conforme mostra a figura 3.2.

Enquanto a vantagem do *middleware* ponto-a-ponto é a simplicidade, a desvantagem deste modelo é a complexidade. No entanto o mercado tem investido na construção de ferramentas que venham, utilizando este modelo, facilitar o processo de integração.

3.1 Tipos de Middleware

A evolução da tecnologia dos middlewares tem mudado diversas características nas categorias existentes, por isso é difícil estabelecer um padrão de classificação. No entanto para o propósito de integração de aplicações descrever-se-á alguns tipos importantes, como: RPC, MOM, objetos distribuídos e middlewares transacionais.

3.1.1 Chamadas a Procedimentos Remotos – RPC (Remote Procedure Call)

Chamadas a Procedimentos Remotos são a maior característica de muitas linguagens de programação. Se houver a necessidade de acessar um serviço em uma máquina como um banco de dados ou uma função do sistema operacional chama-se um procedimento. Caso um programa escrito em linguagem C queira chamar um procedimento em outro módulo, este deve incluir um arquivo *header* que contenha a declaração dos procedimentos do módulo chamado – que é o nome do procedimento e os parâmetros, mas não a lógica.

RPC é um mecanismo de comunicação que permite a clientes fazerem chamadas de procedimento local. Um cliente pode usar o serviço de diretório para conectar-se a um servidor particular em tempo de execução. Além disso o cliente e o servidor podem usar serviços de segurança para garantir os níveis de autenticação, autorização, integridade e de privacidade. O mecanismo de RPC isola o cliente: os detalhes de localização dos servidores na rede, o tipo de plataforma da máquina ou do sistema operacional, as diferenças entre a representação dos dados entre plataformas, o transporte de rede em uso e permite que programas distribuídos trabalhem de forma transparente ao usuário.

No caso do RPC, ao invés de escrever-se um arquivo *header* para uma chamada a um procedimento, escreve-se uma IDL [19] (Linguagem de Definição de Interface). A IDL é uma linguagem que define interfaces independentes da linguagem de programação. Sintaticamente o arquivo IDL é muito parecido com o *header*, no entanto ele gera clientes STUBS e servidores SKELETONS, que são grandes pedaços do código em C compilados e linkeditados dentro dos programas cliente e servidor.

O propósito do STUB é converter parâmetros em uma string de bits e enviar a mensagem através da rede. O SKELETON recebe a mensagem converte-a para os parâmetros do servidor e chama o servidor. O processo de conversão dos parâmetros da mensagem é chamado *marshalling*.

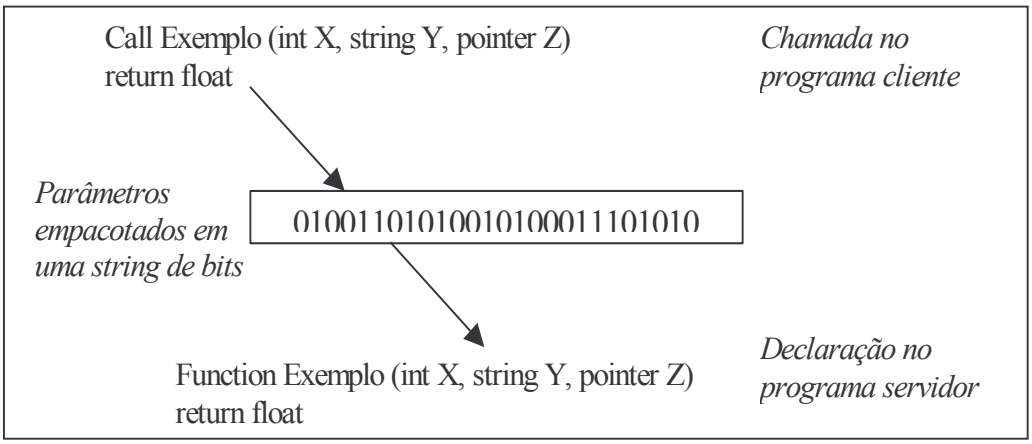


Figura 3.3 – *Marshalling*

A vantagem do *marshalling* é que ele manipula diferentes formatos de dados. Por exemplo se o programa cliente usar um inteiro de 32 bits e o servidor usar um inteiro de 64 bits, o *marshalling* faz a tradução, ver figura 3.3.

Middlewares RPC são bloqueantes, ou seja, comunicam-se através de passagem de mensagem síncrona. O mais conhecido tipo de RPC é o DCE (Ambiente de Computação Distribuída) da OSF (Fundação de Software Aberto), hoje chamado de Open Group.

O DCE implementou um mecanismo sofisticado de RPC com muitas camadas de serviços (como segurança, diretório, integridade). Um grande problema que o DCE apresenta é o baixo desempenho. Um RPC requer de 10.000 a 15.000 instruções para processar uma requisição remota e são necessários 24 passos distintos para completar esta requisição [4].

RPCs são utilizados por muitos pacotes e tecnologias. Por exemplo, objetos distribuídos com CORBA são uma camada adicional ao topo de um RPC.

3.1.2 Middleware Orientado à mensagem – MOM

As deficiências do RPC resultaram na criação do MOM, que utilizam mensagens – unidades (bytes) de informação que se movem entre aplicações – como um mecanismo de transporte de informação ponto-a-ponto.

As mensagens são diretamente acopladas com o *middleware* de uma forma independente da aplicação e assíncrona, ou seja, há uma continuação no processamento depois de feita uma requisição ao serviço do *middleware*.

A mensagem é entregue a um gerenciador de fila, que é responsável por recebê-la e entregá-la ao destino final.

Este modelo é mais complexo que o síncrono, porque é necessário obter a garantia da entrega da mensagem de uma forma não bloqueante. O MOM garante a entrega através da utilização de mensagens persistentes.

As mensagens possuem uma estrutura (esquema) e um conteúdo (dados). No modelo de fila de mensagens (*Message Queuing* - MQ) o programa faz um broadcast da mesma mensagem para diversos programas remotos sem ter que esperar que estes estejam disponíveis para recebê-la, conforme figura 3.4.

Se o servidor estiver inoperante, a requisição permanece na fila. Tão logo o servidor retorne a operar, a requisição é entregue. O MOM também provê execuções concorrentes que permitem o processamento de mais de uma requisição ao mesmo tempo.

Sistemas MQ requerem o uso de APIs a serem acrescentados nos sistemas que são integrados, adaptados ao *middleware*.

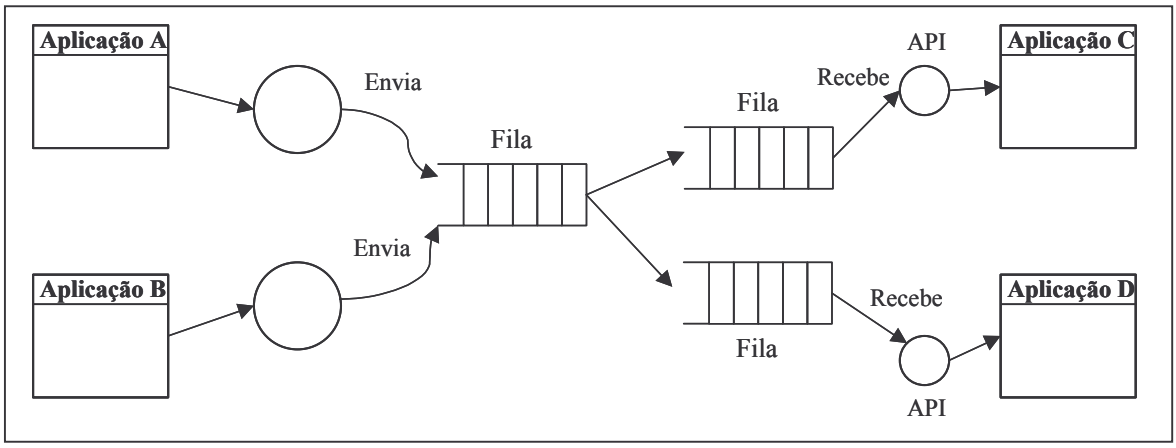


Figura 3.4 – *Middleware* Orientado à Filas

Alguns sistemas MQ podem trafegar diferentes tipos de redes, como SNA, TCP/IP, Novell IPX/SPX. Os mais conhecidos são MQSeries da IBM, MSMQ da Microsoft e Tuxedo da BEA Systems.

A desvantagem do *middleware* orientado a filas de mensagem é que não possuem IDL, nem *marshalling*; a mensagem é uma string de bytes que é traduzida caractere por caractere.

3.1.3 *Middleware* Orientado a Banco de Dados

É qualquer *middleware* que facilita a comunicação com o banco de dados, de uma aplicação de banco de dados ou entre bancos de dados. É usado unicamente para ler e escrever informações em uma base local ou em bases remotas.

Por exemplo, para extrair um dado residente em um banco de dados Oracle, o desenvolvedor tem que invocar o *middleware* para efetuar o *log-on* no banco de dados, requisitar a informação e processar a informação retirada.

Este tipo de *middleware* trabalha com dois tipos básicos: Interfaces de Nível de Chamada (CLI) e Nativos ao banco de dados.

Os CLIs acoplam um conjunto de APIs de diversos tipos de banco de dados e possibilitam acesso para quaisquer quantidades de bancos. Como é o caso do Microsoft ODBC (Conectividade de Bancos de Dados Abertos).

O ODBC cria uma interface comum para estabelecer o acesso a qualquer banco de dados, usando um gerenciador de *drivers* para carregar o *driver* dos bancos envolvidos no processo de integração.

A figura 3.5 mostra um esquema de *middleware* orientado a banco de dados utilizando as APIs OLE DB e ODBC.

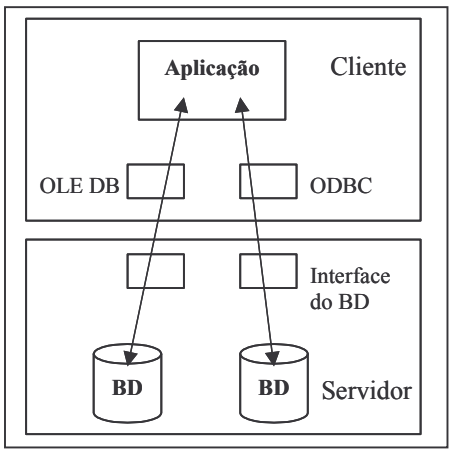


Figura 3.5 – APIs OLE DB e ODBC.

Outros exemplos de CLIs são: JDBC [31] (interface padrão que usa um conjunto de métodos Java para permitir o acesso a diversos bancos de dados) e o OLE DB da Microsoft (*middleware* que provê um mecanismo padrão para acessar bancos de dados e objetos distribuídos COM).

Middlewares nativos não fazem uso de uma API padrão, ao invés disto, eles utilizam funções específicas de um banco de dados particular. A comunicação com apenas um tipo de banco de dados é uma grande limitação do *middleware* nativo.

3.1.4 Objetos Distribuídos e *Middleware* de Objetos

No modelo de programação orientado a objetos (OOP – *Object Oriented Programming*), os sistemas são um conjunto de objetos com estrutura, estado, comportamento e que pertencem a uma classe.

Objetos distribuídos são classificados como *middleware* porque eles facilitam a comunicação entre aplicações. No entanto, eles são também, um mecanismo para o desenvolvimento de aplicações orientada a objetos.

Um programa orientado a objetos, por exemplo em Java ou C++, consiste de uma coleção de objetos que interagem entre si. E cada objeto, por sua vez, é um conjunto de dados e métodos.

Um objeto se comunica com outros pela invocação de seus métodos, passando argumentos e recebendo resultados. Os objetos encapsulam seus dados e o código de seus métodos.

- Referências de Objetos: é o meio de acesso a um objeto. Para invocar um método em um objeto, a referência do objeto e o nome do método são os únicos argumentos necessários. O objeto cujo método é invocado é chamado de Receptor ou Servidor.
- Interfaces: uma interface é uma definição das assinaturas de um conjunto de métodos (que são, os tipos de seus argumentos, valores de retorno e exceções) sem especificar sua implementação.

Para acessar um objeto em uma máquina remota, o programa tem que ter uma referência que aponte para o objeto. O uso de referências abstrai do programador protocolos de rede, localização física do objeto.

A figura 3.6 demonstra a diferença entre uma chamada RPC e uma orientada a objeto.

RPC	call debito_em_conta (3429-2,100) // 3429-2 é o número da conta // 100 é o valor do débito
Orientação a Objetos	call conta_corrente.débito (3429-2) return RefConta; call RefConta.debito_em_conta (100);

Figura 3.6 – Exemplo de chamada RPC x Orientada a objetos

Existem quatro formas de obter uma referência de objeto:

- Uma referência do objeto é retornada ao cliente quando este se conecta ao *middleware*.
- O cliente chama o serviço de nomes que de posse do nome fornecido pelo cliente faz uma pesquisa no diretório. O diretório retorna a localização do objeto e o serviço de nomes converte um uma referência para o objeto.
- O cliente requisita o nome do objeto de uma certa classe e recebe de volta a referência.
- Uma operação em um objeto retorna uma referência para outro objeto.

O caminho escolhido depende da escolha do *middleware* e do código do Servidor.

Existem algumas similaridades entre *Middleware* orientado a objetos distribuídos e RPCs. Em particular, as operações são declaradas em uma IDL (Linguagem de Definição de Interfaces), da mesma maneira do RPC.

A diferença é que no RPC é necessária a conversão da referência do objeto em uma string binária (diferente para cada *middleware*) e no *middleware* de objetos a mensagem contém uma referência do objeto interpretada pela interface.

No *middleware* de objetos, ver figura 3.7, o compilador IDL gera um *stub* que converte as chamadas às operações em mensagens (processo de *marshalling*). Cada mensagem contém uma referência do objeto e retorna esta referência.

Ao contrário do RPC, as operações são chamadas através de uma interface interpretada, como uma macro-linguagem. Estas interfaces necessitam encontrar as operações em tempo de execução.

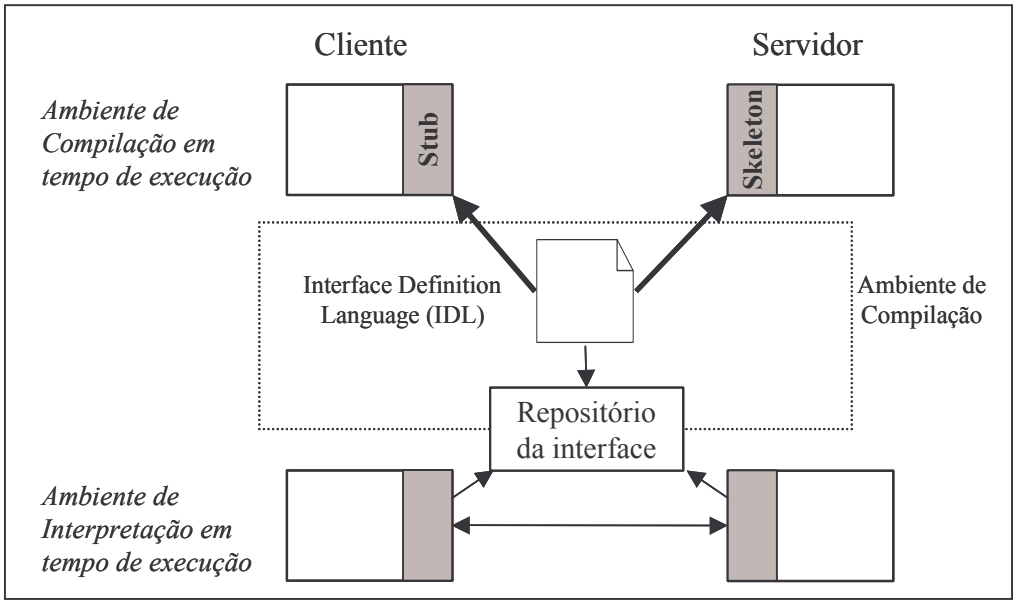


Figura 3.7 – Interpretação e Compilação do Middleware de Objetos

Os três principais padrões de *middlewares* de objetos distribuídos são CORBA, COM/DCOM [18] e Java RMI (Invocação de Métodos Remotos). COM/DCOM é uma estratégia de objetos distribuídos desenvolvida pela Microsoft. Java RMI é utilizado somente para comunicação entre aplicações Java e CORBA é um padrão desenvolvido pela OMG (*Object Management Group*).

3.2 CORBA (Common Object Request Broker Architecture)

Foi inicialmente implementado em 1991, com o interesse de estabelecer uma estrutura comum para o desenvolvimento independente de aplicações, usando técnicas de orientação a objetos para sistemas distribuídos em redes de computadores heterogêneas.

Ao invés de aplicações, a OMG produz especificações que tornam a programação orientada a objeto possível. Este modelo baseado em objetos permite que os métodos de objetos sejam ativados remotamente, através de um elemento

intermediário chamado ORB (*Object Request Broker*) situado entre o objeto da aplicação e o sistema operacional, acrescido de uma série de funcionalidades, que dentre outras tarefas, permite comunicação através de redes heterogêneas.

A segunda versão do CORBA, surgiu em 1994 e apresentou-se como uma solução definitiva para a interoperabilidade entre objetos de diferentes plataformas, linguagens de programação e sistema operacional. Isto se deve ao protocolo IIOP (*Internet Inter-ORB Protocol*), inserido nesta versão do CORBA. No CORBA 1.0 a interação entre objetos distribuídos era permitida apenas para produtos de mesmo fabricante, pois os serviços de interface deixavam abertos aos desenvolvedores, a parte da implementação. Isto dificultava bastante a interoperabilidade entre objetos de diferentes implementações de ORBs.

O IIOP, avanço da GIOP (Protocolo Inter-ORB Geral), especifica um conjunto de formatos de mensagens e dados para comunicação entre ORBs através de uma rede TCP/IP.

O avanço do CORBA e o fato de ter se tornado o padrão mais utilizado para integração de aplicações deve-se ao fato de que CORBA provê: acesso a serviços de forma uniforme, localização de recursos, nomeação de objetos de forma uniforme, detecção automática de erros e política de segurança.

A OMG completou seu objetivo com o estabelecimento de uma arquitetura chamada OMA (*Object Management Architecture*), da qual CORBA faz parte.

O papel do CORBA nesta arquitetura é a localização de objetos solicitados. O OMA abrange as seguintes componentes:

- CORBA e ORB: manipulam a requisição entre objetos.

- Serviços CORBA [32]: definem serviços a nível de sistema que ajudam a gerenciar e manter objetos. Tais como: nomeação, transação, consulta, "trader", concorrência, gerenciamento, persistência, segurança, replicação, externalização, dentre outros.
- Facilidades CORBA [33]: definem uma série de facilidades e interfaces a nível de aplicação. São: interfaces do usuário e gerenciamento de informação do sistema e de tarefas.

Além disso, existem outras facilidades, chamadas facilidades de domínio que visam estender o CORBA a diversas áreas de atuação do mercado, a saber: bancos, ERP[6], Médico, Financeiro, Telecomunicações, Comércio Eletrônico, Tempo-real, etc.

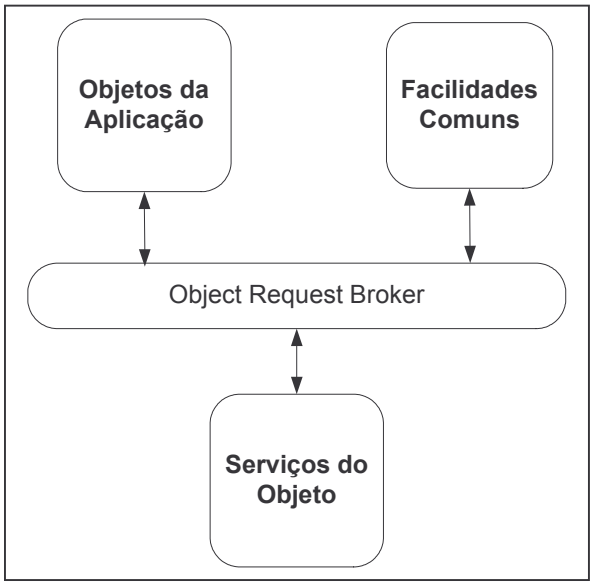


Figura 3.8 – Quatro partes do ORB CORBA.

3.2.1 A Estrutura dos Componentes do CORBA

A arquitetura em questão é construída a partir de um conjunto de componentes (alguns complexos), que interagem entre si, para proporcionar a flexibilidade e a interoperabilidade entre aplicações distribuídas.

[6] ERP - Enterprise Resource Planning

A estrutura da arquitetura CORBA é composta por:

- ORB: é responsável pela localização do objeto ao qual se destina a requisição, assim como o envio dos parâmetros da requisição no formato aceito por este objeto. Também e função do ORB, o retorno de parâmetros de saída da requisição para o cliente, se assim houver.
- Interface do ORB: a interface com a qual o cliente tem contato é totalmente independente de qualquer fator relacionado à heterogeneidade do ambiente distribuído no qual se encontra, como por exemplo: a localização do objeto, a linguagem de programação utilizada para sua implementação e a ordenação de bytes da máquina na qual se executa o objeto. A interface é escrita em IDL (Linguagem de Definição de Interface).
- *Stubs*: escondem os detalhes específicos de implementação, são gerados pelo compilador IDL para o lado do cliente. O *stub* empacota (*marshal*) os parâmetros que serão trafegados pelo núcleo do ORB.
- *Skeletons*: semelhantemente aos *stubs*, os *skeletons* operam no lado do servidor recebendo a mensagem, interpretando-a (*un-marshal*) e chamando a implementação do objeto.
- Invocação Dinâmica: permite que requisições de objetos sejam feitas em tempo de execução, pois os *stubs* são estáticos criados em tempo de compilação.
- *Skeletons* Dinâmicos: interpretam a implementação do objeto em tempo de execução, ao contrário dos *skeletons* que são implementados estaticamente.

- Adaptador de Objeto: permite que uma implementação possa acessar serviços fornecidos pelo ORB. Entre estes serviços destaca-se a geração de referência de objetos e o mapeamento destas referências para a implementação correspondente. Cabe ao adaptador de objetos fazer a ativação e desativação de implementações de objetos, promover a requisição dos métodos e garantir a segurança da interação.

A estrutura do CORBA é ilustrada na figura 3.9.

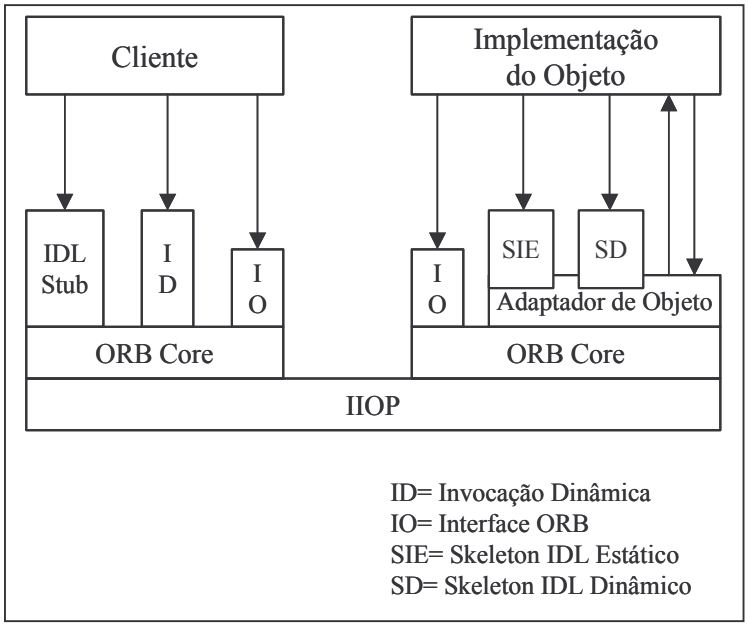


Figura 3.9 – O Modelo CORBA

3.3 Middleware e a Tecnologia Web

O surgimento da Internet, no início dos anos 90, agrupada a um conjunto de tecnologias, como o serviço de publicação de páginas Web, estabeleceu um marco na história da informática.

As primeiras páginas Web eram escritas em HTML (*HyperText Markup Language*), utilizavam o protocolo HTTP (*HyperText Transfer Protocol*) e eram localizados através de uma URL (*Uniform Resource Locator*).

A Internet trouxe uma série de novidades tecnológicas e ajudou a solidificar algumas existentes utilizadas em pequena escala:

- permitiu a universalização do protocolo TCP/IP, como um padrão de comunicação em redes de computadores,
- permitiu o compartilhamento de recursos e se tornou uma base para a construção da maioria dos sistemas distribuídos da atualidade,
- inspirou o desenvolvimento de uma grande quantidade de sistemas disponíveis para Web,
- mudou a forma de pensar a respeito da segurança,
- permitiu o avanço nos serviços de diretórios e resolução de nomes, principalmente pelo uso do DNS (*Domain Name Service*).

Como foi projetada para operar em qualquer plataforma ou sistema operacional, a Web proporciona a capacidade de viabilizar a execução de sistemas em diferentes linguagens de programação e que trafegue sobre qualquer rede que utilize o TCP/IP.

Hoje, a Web tem sido bastante usada como *front-end* para a quase totalidade das aplicações em fase de desenvolvimento, por diversas razões:

- o browser pode rodar em diferentes plataformas (PC, *handheld*, dispositivos sem fio, etc.).
- a tecnologia usada pela Web pode ser usada em *intranets* ou *extranets*.
- o surgimento das relações B2C e B2B e o sucesso do comércio eletrônico estão levando as empresas a disponibilizarem suas aplicações para a Web.

- a facilidade de efetuar transações de compra e venda, de acessar serviços, fazer consultas e pesquisas e de se divertir tem atraído uma infinidade de novos usuários a cada dia, clientes em potencial para as empresas.
- a possibilidade de estender aplicações legadas, através do encapsulamento da lógica de negócios.

Os primeiros documentos publicados na Web eram exclusivamente páginas HTML estáticas munidas de informações textuais que, distribuídas pela Web, faziam da Internet um grande servidor de arquivos.

Com o surgimento da arquitetura multi-camadas (avanço da arquitetura 2 camadas – cliente/servidor), que integrou a camada Web na arquitetura de sistemas, e com a criação do protocolo CGI (*Common Gateway Interface*), as páginas Web passaram a se tornar dinâmicas. O protocolo CGI possibilitou o roteamento de páginas HTML entre o servidor Web e o servidor de aplicação.

Um software CGI é bastante lento, e o compartilhamento de estado entre as invocações é difícil, além de apresentar complexidade para o balanceamento de carga.

Um processo CGI necessita de uma *template* HTML para interpretar as *tags* não formatadas, construir as chamadas e formatar as saídas em HTML, disponibilizando os resultados para a Web.

Este processo é ilustrado na figura 3.10.

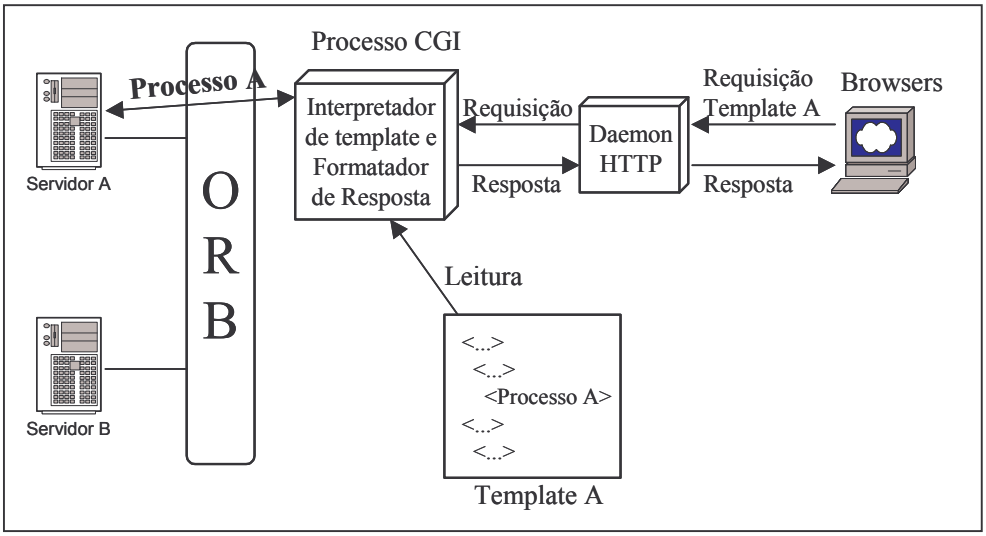


Figura 3.10 – Arquitetura de uma aplicação CGI

O uso de *templates* serviu como base para as implementações iniciais do ASP (*Active Server Pages*) que contém *scripts* no lado do servidor. A página do servidor Web executa um script que permite a invocação de objetos.

Os *scripts* permitem ao usuário interagir com o software cliente sem necessidade de manter-se conectado todo o tempo com o servidor.

Contudo, esta tecnologia necessita do servidor Web e do HTTP para intermediar a comunicação entre os objetos cliente e servidor.

O protocolo HTTP não provê um padrão para estabelecer uma associação com um cliente específico. Uma conexão é feita de um browser a um servidor, o cliente requisita o dado, o servidor responde e a conexão é encerrada. Na próxima conexão do browser ao servidor, este não tem nenhuma maneira de identificar o relacionamento anterior mantido entre eles. Geralmente uma página Web não necessita que o servidor mantenha seu relacionamento, no entanto, aplicações geralmente necessitam manter o estado das primeiras ações para determinar as próximas.

A linguagem de programação Java e os *applets* Java, lançados em 1995, pela Sun Microsystems, foram os primeiros passos para a criação de objetos distribuídos

na Web. Java é caracterizada por sua portabilidade, robustez, segurança e suporte a programação distribuída.

A habilidade de se carregar *applets* e programas Java em um browser tornam-na a ferramenta ideal para a Internet. Esta portabilidade é alcançada porque Java é interpretada, ou seja, na compilação de um programa é gerado um código intermediário e não um código de máquina, como acontece na maioria das linguagens. Isso garante a independência de plataforma de hardware e software.

A execução do código Java é realizada sobre qualquer plataforma através da Máquina Virtual Java (JVM), que é o interpretador do código intermediário (*byte-code*) para a plataforma em que se deseja executar a aplicação. Ver figura 3.11.

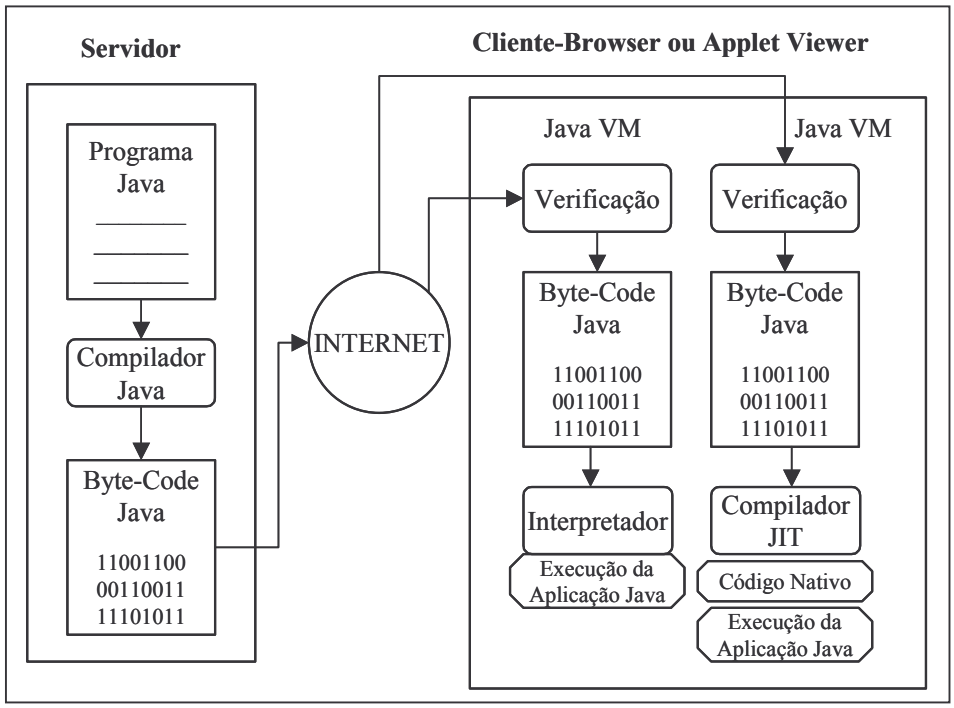


Figura 3.11 – Execução de uma Aplicação Java

Java não é considerada apenas uma linguagem, mas um *framework* que abrange vários componentes, incluindo: um conjunto de APIs, JVM, Java OS (uma implementação de sistema operacional) e muitas interfaces que facilitam o desenvolvimento e o suporte ao desenvolvimento de sistemas distribuídos e de rede.

O suporte aos sistemas distribuídos é dado pela arquitetura RMI (*Remote Method Invocation*). Este suporte pode ser tratado como um ORB nativo Java, isto é, RMI é um ORB (não compatível com o CORBA) que permite invocações a métodos de objetos remotos.

O grande problema do RMI é a impossibilidade de sua utilização em ambientes heterogêneos, com objetos desenvolvidos em outras linguagens.

A arquitetura RMI é constituída de três camadas:

- camada *stub*/*skeleton* para cliente e servidor
- camada de referência remota: responsável pela semântica da invocação dos objetos.
- camada de transporte: que trata da conexão com os objetos remotos.

Utilizando o RMI, o cliente ao requisitar o serviço de um servidor remoto faz um *download* do *stub* cliente, em *bytecode*, proveniente do servidor via HTTP. Este *bytecode* é executado pela JVM. De posse do *stub*, o cliente faz uma chamada RMI ao servidor para acessar os serviços desejados.

3.3.1 Implementação Java e CORBA para a Web

As pesquisas sobre tecnologia de objetos Web avançaram, e uma solução para integração de Java com CORBA foi implementada. A Empresa Netscape implementou a solução em um ORB CORBA/Java e proporcionou, pela primeira vez, a possibilidade da criação de um padrão para integração de aplicações legadas com a Web. Usando apenas o Java, os *applets* podem estabelecer conexões com o servidor HTTP.

Juntamente com CORBA, distribuições IIOP podem rodar em servidores HTTP fornecendo suporte para conexões com outros sistemas.

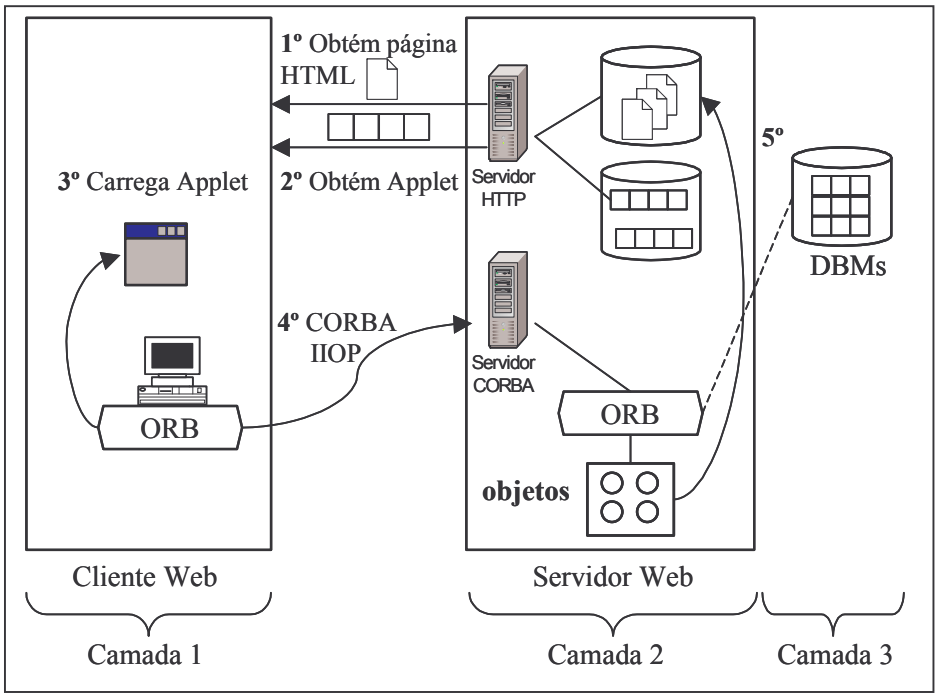


Figura 3.12 – Integração CORBA/Web usando Applet Java

A interação Web, Java e CORBA dá-se da seguinte forma (ver figura 3.12):

1. o *browser* exibe a página HTML com os *applets*;
2. o *browser* solicita o *applet* do servidor;
3. o servidor HTTP recupera o *applet* e envia para o browser na forma de *bytecodes*;
4. o *applet* é carregado pelo browser e é executado;
5. o *applet* invoca objetos do servidor CORBA, através dos *stubs* clientes gerados a partir da IDL. Então a sessão *applet*/objeto CORBA é estabelecida. É necessária a utilização de um *firewall* IIOP para permitir esta transação.

Java com CORBA elimina o problema de performance do CGI e permite a invocação de qualquer método definido em IDL no Servidor. CORBA também mantém o estado da conexão com o cliente, o que não é provido pelo CGI.

Os objetos CORBA podem interagir entre diversos servidores permitindo o balanceamento de carga da aplicação servidora. Isto é uma grande vantagem na criação de *intranets*, onde diversos servidores e aplicações de uma empresa podem se interagir através dos protocolos da Internet, além disso, as máquinas clientes apenas necessitam de um browser para rodar a aplicação.

A tendência da criação de *intranets* tem aumentado espantosamente, pois através desta integração Java/Corba/Web, clientes que rodam em diferentes plataformas e sistemas operacionais podem acessar aplicações servidoras sem a necessidade de criação de um software cliente específico.

Algumas empresas como IBM e Unisys, têm desenvolvido ferramentas que integrem Java e CORBA para permitir que aplicações empresariais em mainframe possam ser acessadas por clientes Java, através da criação de uma Máquina Virtual específica da plataforma.

A grande limitação desta solução usando o *applet* é que o único browser com suporte Java/CORBA [36] é o *Netscape Navigator*, ou seja, esta solução não se tornou um padrão.

Além disso, a dependência de um *firewall* IIOP torna a solução mais cara e complexa e ainda torna o ambiente interno passível de invasão, uma vez que as portas 683 e 684 (IIOP/CORBA) do *firewall* deverão estar abertas.

A arquitetura da metodologia proposta apresenta uma solução para este problema.

4. XML (EXTENSIBLE MARKUP LANGUAGE) E WEB SERVICES

Milhões de usuários acessam a Web todos os dias, milhares de terabytes de dados trafegam, por mês, na Internet [8].

Uma enorme quantidade de aplicações está sendo construída para a Internet e para as crescentes *intranets* nas corporações.

Existem uma variedade de extensões e de ferramentas que permitem à Web disponibilizar uma grande variedade de tipos de dados: textos, som, imagens, vídeos.

Surgem algumas novas alternativas para integrar dados e aplicações legadas com a Web. Uma outra geração de tecnologia, além dos *applets* Java, são puramente extensões básicas da Web.

Clientes fazem requisições estáticas que obedeçam a capacidade do HTML e servidores interpretam requisições e geram respostas.

Esta nova geração objetiva prover um ambiente mais rico para o desenvolvimento de aplicações Web. Uma nova linguagem chamada XML (*Extensible Markup Language*) garante o acesso ao estado do objeto Web através de métodos específicos.

O XML é uma versão amigável à Internet de uma linguagem, criada em meados da década de 80, denominada SGML[7]. O SGML separa os dados propriamente ditos do processamento dos dados. No entanto, o poder de separação entre os dados e o processamento é possível mediante a reconstrução dos dados através dos processos.

[7] SGML - Standard Generalized Markup Language

O SGML define um padrão chamado Definição de Tipo de Documento (DTD), para definir um conjunto de *tags* e um conjunto de documentos legais para propósitos específicos.

XML é uma meta linguagem, uma linguagem para definição de outras linguagens de marcação projetada para Web. XML foi projetada para ser um formato padrão para descrever e trocar estruturas de dados na Internet.

Com XML, um documento ou um pedaço de dado é separado em três partes: estrutura, conteúdo e estilo. XML especifica a estrutura, XSL (*Extensible Style Language*) especifica o estilo e as definições dos usuários para o conteúdo.

- Extensibilidade: XML permite a definição de suas próprias *tags* ou atributos. E mais, XML *Namespaces* permite o uso e a troca entre conjuntos de *tags* ou atributos, incluindo reutilização do código existente.
- Estrutura: XML foi projetado para prover dados estruturados. A estrutura pode ser aninhada e em qualquer nível de profundidade.
- Descrição: XML permite incluir metadados para descrever seus dados.
- Validação: Se um metadado é incluído, XML suporta a checagem dos dados para uma validação estrutural.

XML descreve uma classe de dados chamada documento XML e descreve o comportamento de programas que o processam. Documentos XML consistem de entidades que contém dados *parsed* (analisados gramaticalmente) e *unparsed* (não analisados gramaticalmente).

Dados *parsed* consistem de dados que formam a linguagem de marcação, o resto dos caracteres forma os dados propriamente ditos. A linguagem de marcação

codifica a descrição do documento em uma estrutura lógica. Dados não analisados gramaticalmente é um recurso que contém texto XML ou não.

Um processador XML é usado é usado para processar documentos XML e prover acesso à sua estrutura e conteúdo.

Um documento XML bem formado pode ser válido se obedecer a algumas restrições. Fisicamente o documento é composto de entidades e logicamente é composto de declarações, elementos, comentários, referências de caracteres e instruções de processamento.

4.1 Declarações de Tipos de Documentos

A função da marcação em um documento XML é descrever a estrutura lógica e o layout de armazenamento e associá-los ao par valor-atributo com sua estrutura lógica.

XML provê um mecanismo, a declaração de tipo de documento que define restrições na estrutura lógica e suportam o uso de dados pré-definidos.

A declaração de tipos de dados XML contém ou aponta para declarações de marcação que provêem uma gramática para uma classe de documentos. Esta gramática é conhecida como Definição de Tipo de Documento (DTD).

Uma declaração de marcação é uma declaração de tipos de elementos, uma declaração de uma lista de atributos, uma declaração de entidade ou uma declaração de uma notação.Uma declaração de atributos especifica o nome, o tipo de dado e o valor padrão de cada atributo associado com um elemento dado. Declaração de entidade define unidades de armazenamento. Declarações de Notação provêem nomes para as notações que identificam pelo nome o formato das entidades não analisadas gramaticalmente.

A figura 4.1 apresenta um exemplo de um documento XML com uma declaração de um tipo de documento interno.

```
<?xml version=”1.0:?>
<!DOCTYPE greeting >
<!ELEMENT greeting (#PCDATA)>
<greeting>Hello World! </greeting>
```

Figura 4.1 – Documento XML com declaração interna

A declaração, também pode ser dada externamente, conforme apresentado na Figura 4.2.

```
<?xml version=”1.0:?>
<!DOCTYPE greeting SYSTEM “hello.dtd”>
<greeting>Hello World!</greeting>
```

Figura 4.2 – Documento XML com declaração externa

4.2 Estruturas Lógicas

Cada documento XML contém um ou mais elementos, que são delimitados por *tags* de início e *tags* de fim, ou por elementos vazios, por uma *tag* de elemento vazio. O texto entre as *tags* de início e de fim é chamado de conteúdo. Cada elemento tem um tipo, identificado pelo nome e pode ter um conjunto de atributos. Cada atributo tem um nome e um valor.

A estrutura elementar de um documento XML pode, para validar propósitos, ser restringida pelo uso de declarações de tipos de elementos e declarações de listas de atributos. Como mencionado anteriormente, uma declaração de tipos de elementos contém:

- EMPTY: o elemento não tem conteúdo;

- Conteúdo do Elemento: o elemento tem elementos filhos. Existem restrições para os elementos filhos, na ordem em que eles vão aparecer e seus multiplicadores (zero para um “?”, um “ ”, zero ou mais “?” e um ou mais “+”).
- ANY: o elemento tem elementos filhos, seus tipos, ordens e multiplicidades não são restringidos.
- MIXED:o elemento contém dados caracteres, opcionalmente com elementos filhos. Os tipos de elementos filhos podem ser restringidos, mas não na ordem e no número de ocorrências.

Uma declaração de lista de atributos especifica o nome, tipos de dados, e um valor padrão de cada atributo associado com o tipo de elemento dado. Ele pode aparecer somente dentro de *tags* de início e *tags* de elementos vazios.

4.3 Definição de Esquemas XML

Apesar de sua grande utilidade o DTD tem limitações que rapidamente se tornam aparentes no desenvolvimento corporativo. Uma grande limitação é que o DTD não é um documento XML válido, além de não poder manipular espaços de nomes dentro de estruturas XML ou descrever relacionamentos complexos entre documentos e elementos. DTDs não são modulares e restringem definições para um elemento de dados não poder ser reutilizado (herdado). Por estas e outras razões o World Wide Web Consortium-W3C está trabalhando na definição de um esquema XML [34] com declarações gramaticais baseadas em XML.

XML Schema Definition – XSD é uma linguagem XML de restrição de dados, relacionamentos hierárquicos e elementos *namespaces* que podem expressar-se

mais completos que com DTD. Esquemas permitem definição muito precisa de tipos de dados simples e complexos e permitem herança de tipos.

Existem tipos de dados comuns construídos com base em XSL como ponto inicial para construção de linguagens.

A Figura 4.3 mostra uma definição de esquema XML para um exemplo de catálogo de produtos.

```
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element type="product-catalog"/>
<xsd:complexType name="productCatalog">
<xsd:element type="productType"
minOccurs="1"/>
</xsd:complexType>
<xsd:complexType name="productType">
<xsd:element name="description"
type="xsd:string" minOccurs="1">
<xsd:attribute name="locale"
type="xsd:string"/>
</xsd:element>
<xsd:element name="price"
type="xsd:decimal" minOccurs="1">
<xsd:attribute name="locale"
type="xsd:string"/>
<xsd:attribute name="unit"
type="xsd:string"/>
</xsd:element>
<xsd:attribute name="sku"
type="xsd:decimal"/>
<xsd:attribute name="name"
type="xsd:string"/>
</xsd:complexType>
</xsd:schema>
```

Figura 4.3 – Exemplo de Definição de Esquema XML

Este XSD define um tipo complexo chamado productType, que foi construído sobre outros tipos de dados primitivos. O tipo complexo contém atributos e outros elementos como parte de sua definição. Somente através deste simples exemplo verificam-se as vantagens do XSD sobre o DTD.

4.4 XSL (Extensible Style Language)

XSL é uma linguagem declarativa usada para estabelecer regras para elementos XML e também para proporcionar comportamento a estes documentos. Extensível através de Java script e escrito em XML, XSL constrói padrões de documentos providos de semântica.

A habilidade de prover uma camada de regras de semântica à documentos XML capacita o XSL como uma grande arma para prover soluções de interoperabilidade na integração de aplicações. O XSL modifica o formato do documento XML, podendo efetuar alterações tanto no esquema quanto no conteúdo.

Um estilo é representado por um elemento *xsl:stylesheet* ou *xsl:transform* em um documento XML. Estes elementos são chamados de elementos de nível do topo e não podem ser iguais a nenhum elemento do espaço de nomes do transformador XSLT [24], para não mudar seu comportamento.

A Figura 4.4 apresenta um exemplo de uma estrutura de um estilo XSL.

```
<xsl:stylesheet version”1.0”
xmlns:xsl=”http://www.w3.org/1999/ XSL/transform”>
<xsl:import href=”...”/>
<xsl:include href=”...”/>
<xsl:key name=”...” match=”…” use=”…”/>
<xsl:template name=”...”/>
</xsl:template>
```

Figura 4.4 – Exemplo de um estilo XSL.

4.5 XSLT (Extensible Style Language Transformation)

XSLT é uma linguagem que define regras de transformação para documentos XML. Para transformar os documentos XML, o XSLT precisa: modificar a

estrutura, onde os dados são transformados da estrutura de saída de uma aplicação para a estrutura de entrada da outra aplicação. Esta tarefa envolve seleção dos dados e o agrupamento, dependendo da necessidade da transformação, além da formatação do texto em uma nova estrutura: XML, HTML, PDF, etc.

4.5.1 O Processamento XSLT

O XSLT não opera diretamente no arquivo XML, ele necessita de um interpretador (*parser*) para converter os valores em uma árvore de objetos que serão então processados. Ele usa esta árvore para manipular a estrutura na memória, conforme apresenta a Figura 4.5. O XSLT aplica um estilo XSL em um documento XML e gera os resultados (o documento de saída). O estilo XSL define qual transformação irá ocorrer.A árvore de objetos é um tipo de dados abstrato e não existe padrão para representá-la, por isso o uso do estilo.

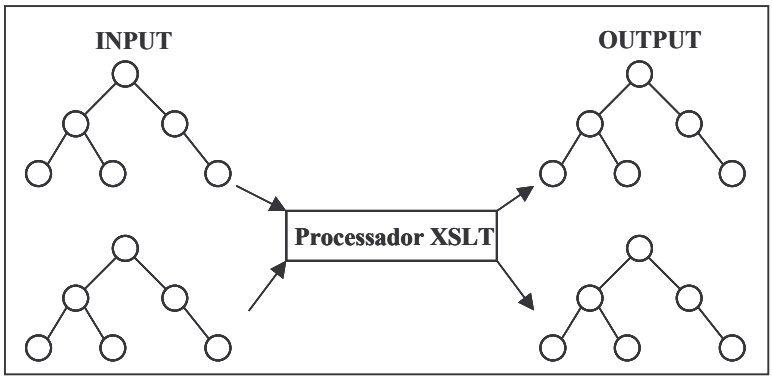


Figura 4.5 – Árvore de elementos processados pelo XSLT.

4.5.2 O Processo de Transformação

As propriedades do XSLT contêm um estilo (*stylesheet*) com um número de *templates* expressas como <XSL:template>, que define os atributos iniciais. O valor dos atributos iniciais é um padrão, e o padrão determina quais os nós na árvore inicial que a *template* aplicará as regras. Por exemplo:

<XSL:template match= “/”>

Diversos eventos ocorrem durante uma transformação típica. Quando uma transformação XSLT é executada, o estilo é aplicado para avaliar e processar o documento inicial, remapeando o documento para uma árvore que está contida na memória. O próximo passo é encontrar a regra, na *template*, que inicializa o nó raiz nesta árvore. Então o processador XSLT inicia o conteúdo da regra que, dentro de um estilo, é a seqüência dos elementos e nós.

Quando a *template* é inicializada, cada instrução é executada e trocada pela árvore resultado.

```
<xsl:stylesheet
id = id
extension-element-prefixes = tokens
exclude-result-prefixes = tokens
version = number>
<!- Content: (xsl:import*, top-level-elements) ->
</xsl:stylesheet>
<xsl:transform
id = id
extension-element-prefixes = tokens
exclude-result-prefixes = tokens
version = number>
<!- Content: (xsl:import*, top-level-elements) ->
</xsl:transform>
```

O XSLT faz do XML um bom candidato para conectar componentes e aplicações. Como os *parsers* e os processadores de transformação XSLT são interpretados: eles lêem o documento XML, analisam a definição da estrutura e valida o documento.

No contexto de componentes os DTDs e esquemas são gerados em tempo de compilação. O XSLT, no entanto, é executado em tempo de execução.

O XSLT não é uma solução para todo tipo de integração de aplicação, no entanto possui um grande potencial pelo fato de fornecer um mecanismo de transformação padrão. Algumas das maiores limitações do XSLT são baixo desempenho e implementações de segurança.

Arquiteturas modernas como Microsoft.Net [25] usam codificações para XML na chamada de componentes e os Web Services (tecnologia utilizada pela arquitetura proposta), através do padrão SOAP definem um envelope para troca de dados entre sistemas.

4.6 Web Services

Os Web Services são a mais recente tecnologia para desenvolvimento de aplicações distribuídas, permite a execução de uma chamada a procedimento remoto de um objeto através da Internet ou de uma intranet de uma empresa. Utiliza padrões, como HTTP e XML, e possibilita o desenvolvimento de aplicações independentes de plataforma, linguagem de programação ou sistema operacional.

O modelo de Web Services destaca dois elementos importantes de qualquer sistema – papéis e operações.

Por papéis, entende-se que são os diferentes tipos de entidades e as operações representam as funções executadas por essas entidades, para que o Web Service possa funcionar.

4.6.1 Arquitetura Baseada em Serviço

Um serviço é definido como uma lógica de software bem definida, invocável, endereçável em rede, transparente de localização e que não depende de um contexto ou estado de outros serviços empregados na arquitetura[9]. Serviços podem ser criados a partir:

- do "zero", usando tecnologias como J2EE [20] (Java 2 Enterprise Edition) ou Microsoft .Net;
- de serviços pré-existentes;
- com o propósito de integração através de um envelopamento.

A arquitetura baseada em serviços é relativamente nova e ainda não foi abraçada universalmente. No entanto, esta arquitetura visa preencher todas as lacunas no processo de integração de aplicações.

Existem alguns destaques na arquitetura, que são:

- aceitação universal pela Internet como um elemento capaz de conduzir negócios pela rede,
- o uso de interfaces de padrão aberto,
- uma abundância de sistemas existentes que podem utilizar esta tecnologia.

Para usufruir destes benefícios, uma organização tem que articular seus negócios e objetivos tecnológicos e definir uma rota para alcançá-los.

Os benefícios da arquitetura baseada em serviços são:

- Maior mobilidade lógica: por causa da transparência da localização e de não estar atrelada à linguagem de programação, uma organização pode mais facilmente exportar estes serviços.
- Especialização nos Papéis do Desenvolvimento: onde desenvolvedores podem aperfeiçoar cada serviço, independentemente do restante da lógica empregada.
- Redução dos custos e qualidade do software: os serviços usam interfaces públicas, que desenvolvedores podem testar mais facilmente e validá-las para aplicação em larga de escala. Isto se traduz em menos erros, maior qualidade no software e conseqüentemente menores gastos com manutenção.
- Disponibilidade mais rápida no mercado: os serviços criados irão compor um repositório de serviços reutilizáveis, que facilmente irão disponibilizar novas lógicas de negócios na Web.
- Reutilização de Código: reutilizar é a palavra chave, hoje, no processo de desenvolvimento de aplicações e o uso do XML possibilita uma maior abrangência na reutilização.
- Melhor Escalabilidade e Disponibilidade: o emprego da arquitetura baseada em serviços permite rodar muitas instâncias de serviços, com separação de configuração de hardware e software.
- Facilidade na Integração de Aplicações: os serviços permitem facilidade na integração com novas e já existentes aplicações pela permissibilidade de empacotamentos.

4.6.2 A Arquitetura dos Web Services

Em um ambiente de Web Services, identifica-se três tipos de papéis:

- Provedor de Serviços: é a entidade que cria o Web Service, descrevendo-o em um formato padrão e publicando o serviço em um registro central que esteja publicamente disponível para todos os interessados.
- Consumidor de Serviços: qualquer empresa que utilize o Web Service publicado. O consumidor deve conhecer a funcionalidade do Web Service, a partir da descrição disponibilizada pelo provedor. O consumidor precisa então estabelecer um vínculo com o serviço para poder utilizá-lo.
- Registro do Serviço: é a localização onde o provedor de serviços pode relacioná-los. O registro informa o tipo e a capacidade do serviço, relatada pelo provedor, para que o consumidor os encontre e estabeleça um vínculo.

A figura 4.6 apresenta um esquema dos papéis dos Web Services.

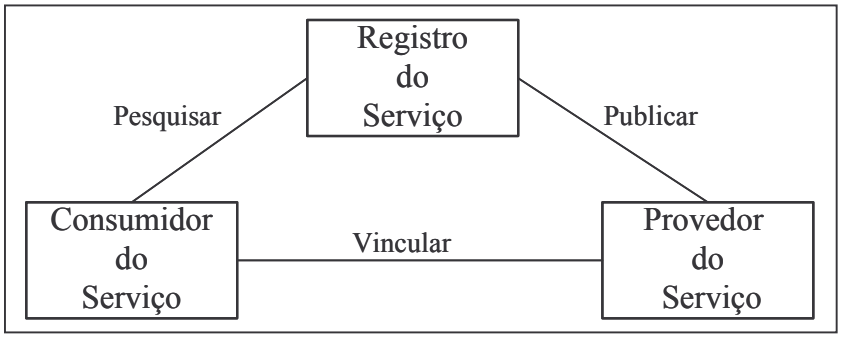


Figura 4.6 – Tipos de papéis dos Web Services.

Existem alguns padrões estabelecidos para criação dos Web Services, a saber:

- WSDL [21] (*Web Service Description Language*) é um padrão que utilize o formato XML para descrever os Web Services. Basicamente, o documento WSDL define os métodos do serviço e os parâmetros de entrada e saída para cada um dos métodos, os tipos de dados, o protocolo de transporte usado e a URL onde o serviço está hospedado[38].
- Um protocolo padrão para publicar ou localizar Web Services, chamado UDDI (*Universal Description, Discover and Integration*) que permite aos provedores a publicação dos detalhes dos serviços em um registro central (descrição) e aos consumidores localizarem provedores e detalhes dos serviços (descoberta).
- Um protocolo para vínculo com os Web Services. O SOAP [22] (*Simple Object Access Protocol*) é um mecanismo superficial do XML, usado para trocar informações entre aplicações, independentemente do sistema operacional e da linguagem de programação.

4.6.3 Pilhas Básicas dos Web Services

A seguir tem-se a pilha básica dos Web Services:

UDDI	Serviço de Publicação
WSDL	Linguagem de Descrição
SOAP	Mensagem XML
HTTP, SMTP, FTP	Transporte

4.6.3.1 Rede de Transporte

Essa camada da pilha de Web Services é responsável pela disponibilização dos Web Services, tornando-os acessíveis por intermédio de alguns dos

protocolos de transporte disponíveis. No cenário atual o HTTP é o protocolo de comunicação mais amplamente utilizado, tanto para a Internet quanto internamente nas empresas (Intranet).

4.6.3.2 SOAP

Esta camada define o formato da mensagem (XML) usado na comunicação entre as aplicações. O SOAP é um padrão para codificação da mensagem XML que invoca funções de outras aplicações. É análogo ao RPC, usado no modelo CORBA e DCOM, mas elimina várias complexidades no uso de interfaces.

Uma mensagem SOAP consiste em um elemento Envelope SOAP, que inclui um elemento HEADER (cabeçalho) SOAP e um elemento BODY (corpo) SOAP. O envelope do SOAP define um *framework* para a descrição da composição da mensagem e como ela deve ser processada. O elemento HEADER, opcional, pode ser usado efetivamente pelos usuários para fornecer os dados adicionais necessários. O elemento BODY contém a carga útil do XML, que codifica a chamada do procedimento, as respostas da chamada e/ou o relatório das falhas.

Figura 4.7 – Corpo de uma Mensagem SOAP.

O elemento HEADER, embora seja opcional, amplia a funcionalidade do SOAP, ao passar um parâmetro adicional, usado para fornecer informações de identidade.

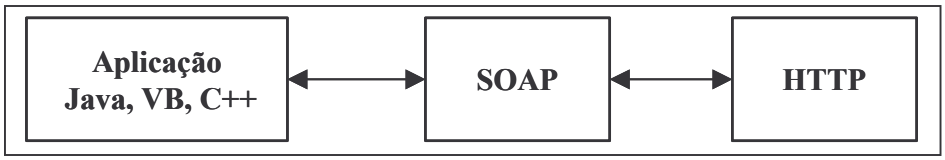


Figura 4.8 – Esquema de uma solicitação SOAP.

Uma aplicação cliente, conforme apresentado na Figura 4.8, faz solicitações em SOAP, que envolve operações em XML, que é então transportada pelo protocolo HTTP. Na parte do Web Service, a camada de transporte passa a chamada para o servidor SOAP, que chama a funcionalidade da aplicação apropriada (exposta como um Web Service). Todas as respostas do Web Service são codificadas de volta para uma resposta do SOAP, que é transportada de volta, por meio do HTTP, para o cliente.

A comunicação inter-aplicações foi equipada por várias tecnologias padrões, como CORBA e DCOM. Estas tecnologias apresentam dificuldades em transpor firewalls. O SOAP efetua a comunicação através do uso do XML, com base em padrões abertos, sem ficar preso a um mecanismo de terceiro.

4.6.3.3 WSDL – Descrição do Serviço

A camada de Descrição do Serviço, fornece um mecanismo ao Provedor de Serviços, a fim de descrever a funcionalidade proporcionada pelo Web Service. A WSDL define a gramática XML do serviço, através de portas, tanto nas mensagens orientadas a documentos quanto nas orientadas a procedimentos.

O WSDL é para um Web Service o que o CORBA IDL é para o CORBA ou o Microsoft MIDL é para os componentes COM.

Um Web Service padrão contém os seguintes elementos:

- Tipos de Porta: os elementos *portType* contêm um conjunto de operações representadas como elementos *operation*, que estão presentes em um Web Service, como, por exemplo GetStockQuote, SellStock, BuyStock, etc.
- Mensagens: um elemento *message* contém uma definição de dados a serem transmitidos. É semelhante ao parâmetro na chamada do método.
- Tipos: o elemento *types* contém os tipos de dados que estão presentes na mensagem.
- Vínculos; o elemento *binding* mapeia os elementos *operation* em um elemento *portType* para um protocolo específico.

Estes elementos representam uma definição abstrata do serviço, contudo para uma operação ser executada em uma rede, deve-se utilizar o protocolo de rede específico para enviar os dados pela rede. É onde entra em cena o elemento *binding* da linguagem de definição do Web Service. Um elemento *binding* captura o protocolo particular (SOAP, HTTP GET, HTTP POST) e os elementos *portType*, *message* e *types*. Portanto, pode-se ligar a definição abstrata a vários protocolos de transporte.

4.6.3.4 Publicação e descoberta do Serviço – UDDI

A partir de um documento WSDL, um consumidor de serviço pode determinar os detalhes do Web Service, como as diferentes operações, tipos de dados, extremidades, protocolos de vínculo e outros.

Ao invés de publicar um documento WSDL para cada possível cliente, um provedor de serviços estará bem servido se publicar as informações sobre seus serviços em um registro central público.

Faz mais sentido para o consumidor dos serviços (clientes) pesquisar em um registro central, que pode agrupar serviços em categorias comerciais ou em domínios semelhantes.

O UDDI [23] fornece uma estrutura de negócios, contendo as seguintes informações:

- Negócio: são as informações comerciais, como o nome do negócio ou o contato comercial. A especificação UDDI permite que uma definição comercial contenha um código de classificação que a identificará como pertencente a uma determinada categoria de negócios.
- Serviço: captura os diferentes serviços oferecidos pelo negócio. Cada serviço contém uma estrutura de uma especificação técnica.
- Especificação Técnica: contém detalhes técnicos sobre um Web Service. Uma das especificações técnicas é a WSDL a qual um consumidor pode fazer a referência.

As tecnologias citadas, funcionam em conjunto conforme descrito a seguir e apresentado na Figura 4.9.

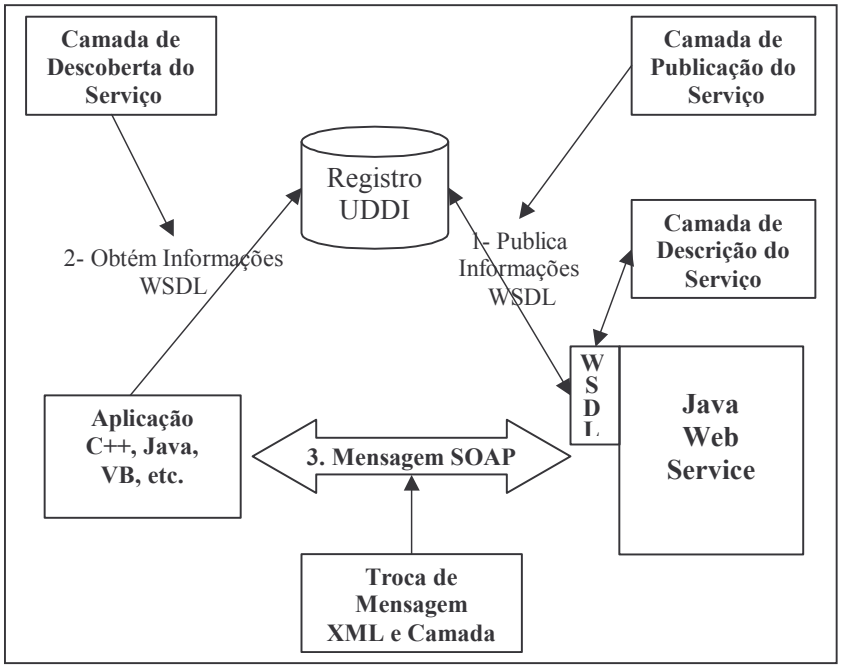


Figura 4.9 – Ciclo de vida de um Web Service.

1. A primeira etapa do provedor de serviços envolve a codificação do arquivo WSDL. Têm surgido algumas ferramentas para auxiliar a descrição do arquivo WSDL. A seguir, é necessária a publicação das informações sobre o serviço do negócio e a especificação técnica, como um arquivo WSDL no registro UDDI central. Assim a descrição do Web Service por meio da codificação do arquivo WSDL é capaz de capturar a camada de Descrição do Serviço, enquanto a publicação das informações comerciais e o arquivo WSDL representam a camada de publicação do serviço.

2. A aplicação do consumidor do serviço pode descobrir os Web Services que ele tem interesse em utilizar. A descoberta envolve não apenas a pesquisa de negócios e seus serviços, como também o carregamento das especificações técnicas mencionadas no arquivo WSDL. Esta etapa corresponde à descoberta do Serviço.

3. A aplicação do consumidor do serviço utiliza o arquivo WSDL para determinar as mensagens que precisam ser passadas na comunicação com o Web Service do provedor de serviços. Determina, também, as informações sobre o vínculo, SOAP através de HTTP. Então, é feito o envio através de solicitações SOAP e ocorre o

recebimento das respostas, também, em SOAP, correspondendo à camada de Mensagens e Transporte de XML na pilha de Web Services.

O principal *gateway* para solicitações SOAP a partir de um cliente de um Web Service é o Servidor Web. O Servidor Web comunica-se através do protocolo HTTP através da porta 80.

A partir desta porta é possível o tráfego de mensagens XML incorporado ao SOAP, pois este último trafega, também, sobre o protocolo HTTP.

É importante também ressaltar que o arquivo WSDL está localizado no Servidor Web. Os servidores Web encontram-se em uma área acessível a clientes externos de uma empresa, área esta chamada de DMZ (Zona Desmilitarizada), que possui IPs válidos de Internet e hospedam páginas identificadas por URL's específicas, conforme apresentado na Figura 4.10. Sendo assim o arquivo WSDL também pode estar acessível globalmente através de sua referência.

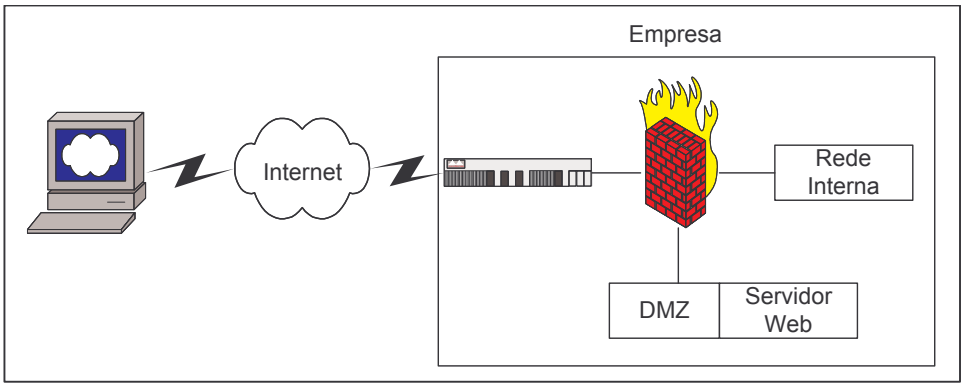


Figura 4.10 – Zona Desmilitarizada (DMZ)

A maioria dos Web Services implementados hoje são simples, ainda não existem domínios de negócios expostos, utilizando esta tecnologia, na Internet. Os serviços de negócios envolvem representações dos cenários de comércio eletrônico do mundo real, onde existem envolvimento e colaboração entre um pool de empresas da mesma área de atuação.

Sugestões, de Web Services em comércio eletrônico, poderiam ser:

- Web Services de validação de transação em cartões de crédito. Poderia agilizar a compra na Internet sem a intervenção de uma consulta a outro sistema, pertencente à operadora de cartão-de-crédito. Comerciantes em potencial seriam clientes deste serviço.
- Consulta de CEP/logradouro seria também em Web Service interessante, que evitaria erros ou demora na entrega de mercadorias. A consulta poderia ser feita automaticamente pela página do comércio eletrônico a um Web Service disponível pelo *site* do Correio.
- Uma agência de empregos deseja publicar as ofertas de emprego como um Web Service. Os interessados por contratação seriam clientes deste serviço.

Os Web Services podem ser uma alternativa em EAI, onde as empresas deverão expor a interface de suas aplicações legadas em XML, dando um grande passo para a comunicação global entre aplicações corporativas.

Para que a empresa suporte esta complexa infraestrutura para construção de aplicações distribuídas e para integração de múltiplas aplicações com Web Services é necessária a utilização de um modelo de projeto e implementação de sistemas, que assim como XML e Web Services, proporcione reutilização e independência de plataforma, sistema operacional e linguagem de programação. Escolheu-se como modelo para este trabalho o uso da mais nova especificação para suporte ao desenvolvimento de softwares distribuídos, o MDA (*Model Driven Architecture*).

5. MDA (*MODEL DRIVEN ARCHITECTURE*)

A tecnologia tem evoluído com uma velocidade cada vez maior a cada dia, o sucesso da Integração de Aplicações Empresariais (EAI) depende do impacto de três elementos: uma abordagem de integração que facilite o desenvolvimento de sistemas e a integração de processos através de um ciclo de vida, uma arquitetura flexível e escalável e uma infra-estrutura que suporte a transição ou integração de lógicas de negócios relevantes e um conjunto de tecnologias interoperáveis que determine como várias aplicações podem ser implementadas e integradas.

O *Model Driven Architecture* (MDA) é uma abordagem inovadora para construção de arquiteturas empresariais. Criado pela *Object Management Group* (OMG), o MDA abstrai e visualiza os requisitos do negócio através da construção de modelos independentes da plataforma e tecnologia empregadas, separando os detalhes de implementação das funções de negócios.

Um processo típico do desenvolvimento de software, definido pelo RUP[8] [26], inclui as seguintes fases:

1. Engenharia de Requisitos;
2. Análise e Descrição Funcional;
3. Projeto;
4. Codificação;
5. Teste;
6. Implantação.

Quando este processo é desempenhado de uma forma interativa e incremental, documentos e diagramas são produzidos até a fase de Projeto. A maioria

[8] RUP – Rational Unified Process

destas descrições tem utilizado diagramas em UML (Linguagem de Modelagem Unificada)[45], apresentada como um padrão, pela OMG, em 1997. A UML tornou-se um padrão mundial ao englobar e unificar os métodos de Rumbaugh [27] (OMT), Booch [28] e Jacobson [29] (OOSEA) para especificação, visualização e documentação de artefatos em um sistema baseado em objetos.

A Figura 5.1 apresenta um esquema do desenvolvimento iterativo e incremental, com a seqüência das fases e o produto final de cada fase, que serve como dados iniciais da fase subseqüente.

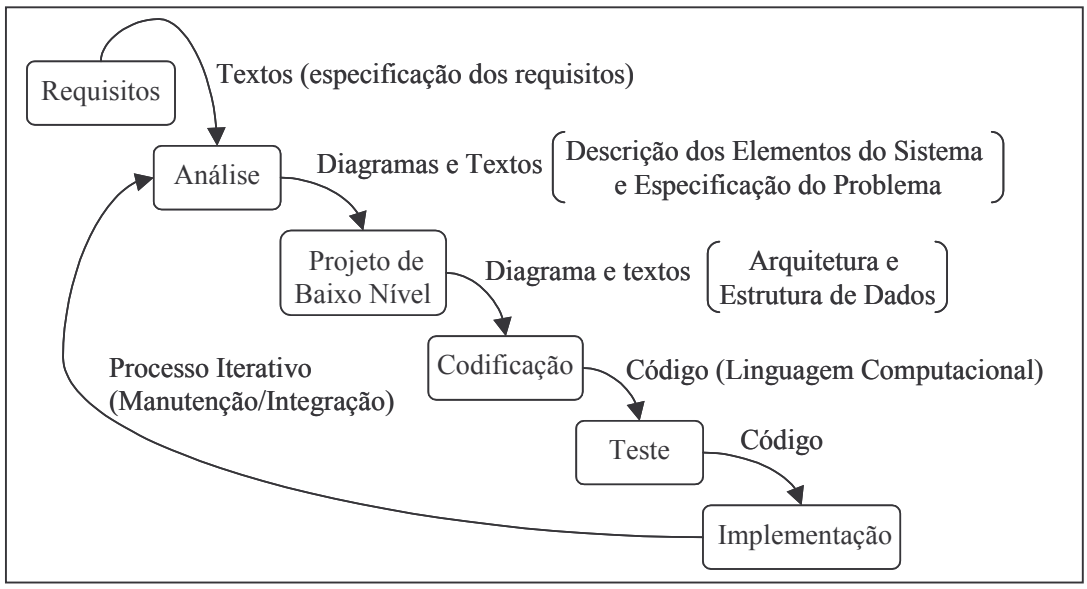


Figura 5.1 – Processo Iterativo e Incremental de Desenvolvimento de Software

O UML produz diversos diagramas como: casos de uso, classes, interação, atividades e outros. A pilha de papéis produzidos é enorme. No entanto, alguns documentos produzidos na fase inicial perdem seu valor na codificação, em virtude do número de alterações naturais que surgem no decorrer do processo.

Quando o número de alterações é grande, torna-se inviável a alteração de todos os documentos desde a fase inicial, levando alguns a crerem que é um tempo perdido a construção destas especificações.

Apesar da aparente perda de tempo, fazer manutenção em sistemas apenas pelos códigos é bastante difícil. Entender a lógica utilizada em centenas de milhares de linhas de código, implementando lógicas complexas, que envolvem os sistemas atuais é um grande desafio. Sem levar em consideração, a gama de linguagens disponíveis como Java, XML, ASP, VB, .Net, C++, etc. e a popularização de sistemas operacionais como Windows e Linux. As empresas ficam confusas com tanta tecnologia associada à real necessidade de modernidade para alcançar mais clientes (através da Internet) ou para integrar suas aplicações à de parceiros corporativos.

Como escolher a linguagem correta? Vale a pena desenvolver outro sistema corporativo do zero? Que tecnologia de sistema operacional e linguagem de programação utilizar? O que fazer para integrar as aplicações existentes? Qual será a sobrevida das tecnologias usadas atualmente no mercado? E se a tecnologia escolhida se tornar obsoleta?

Estas são algumas questões que precisam ser respondidas pelas empresas, em tempo hábil, para que não percam espaço junto ao mercado. As decisões precisam ser acertadas, pois envolve altos investimentos e uma grande demanda de tempo.

A situação é bem complexa, porque as tecnologias mudam, novas versões são lançadas, ferramentas são lançadas no mercado sem fornecerem garantia de compatibilidades com novas que brevemente surgirão.

Raramente os sistemas vivem isolados, em sua maioria, em algum momento do seu ciclo de vida, necessitam integrar-se com outros existentes. Um exemplo típico é a adaptação de sistemas para a Web.

A construção de sistemas monolíticos não existe mais, os sistemas atuais devem prover duas características essenciais para os dias de hoje, interoperabilidade e capacidade de reutilização.

O MDA vem possibilitar um rápido desenvolvimento de novas implementações e facilitar o processo de integração.

O MDA determina que a melhor forma de desenvolver aplicações é separar a lógica de negócios e elementos do domínio de recursos que serão incorporados ao projeto[43].

5.1 Modelo Independente de Plataforma – PIM

PIM é o primeiro modelo definido pelo MDA, com um alto nível de abstração que o torna independente de qualquer tecnologia de implementação.

Um PIM descreve um sistema que suporta uma determinada lógica de negócios. Onde o sistema vai ser implementado, que linguagem e banco de dados utilizará não é um papel do PIM.

Existe uma série de definições para um modelo, mas todas têm algumas características em comum:

- Um modelo é sempre uma abstração de algo que exista na realidade e pode ser usado como exemplo para produzir algo real.

Um sistema é geralmente caracterizado por um contexto que envolve: programas com componentes executáveis (código) e não executáveis (requisitos do projeto), estruturas de dados e documentos. No caso do modelo de negócios, o negócio é o sistema.

Os modelos têm que ser escritos em uma linguagem padrão, no caso o MDA utiliza a UML.

Um sistema pode ter diferentes modelos e dois modelos do mesmo sistema tem um certo relacionamento. Um modelo pode representar uma parte de um

sistema, enquanto outro pode representar uma parte que englobe a primeira. Um modelo pode descrever um sistema com mais detalhes do que outro.

O MDA foca no tipo de relacionamento entre modelos: geração automática de um modelo em outro.

Um sistema descrito por um modelo de negócio é o próprio negócio ou a empresa (ou parte dela). Linguagens que são usadas para modelar negócios contêm um vocabulário que permite especificar os processos dos negócios, departamentos, dependências entre processos, etc.

Um modelo de negócio não necessariamente diz algo a respeito do software usado dentro da empresa, por isso é chamado *Modelo Independente da Plataforma*.

Os requisitos do software são derivados do modelo de negócios que o software precisa suportar. Cada software é usado para suportar um pedaço diferente de um modelo de negócio. (ver Figura 5.2)

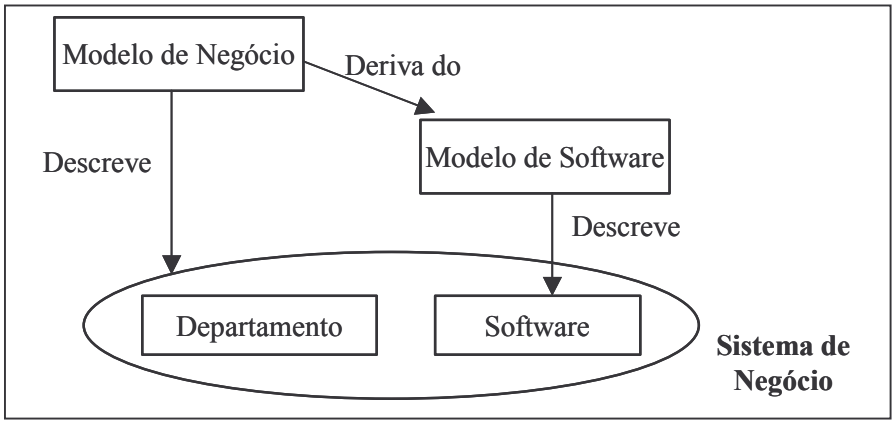


Figura 5.2 – Software como descrição do modelo do Negócio.

Em UML a modelagem do software começa pelo desenho das classes em uma diagrama de classes, o que não significa um desenvolvimento de um modelo de classes.

5.2 Modelo Específico de Plataforma – PSM

Neste passo o PIM é transformado em um ou mais modelos específicos de plataforma. O Modelo Específico de Plataforma - PSM especifica um sistema em termos de construção da implementação, disponível em uma tecnologia específica. Por exemplo, um PSM *Enterprise Java Bean* contém toda a estrutura de um EJB, como: "interface home", "bean de entidade", "bean de sessão", dentre outros.

Para cada tecnologia diferente pode ser gerado um PSM diferente, a partir do mesmo PIM.

As transformações de um modelo para outro são principalmente manuais. Algumas ferramentas podem gerar algum código do modelo, mas a maioria do trabalho ainda é feita pela equipe de desenvolvimento. Contudo, a transformação de um PSM para código é geralmente feita pelo uso de ferramentas por causa da proximidade entre ambos, conforme apresentado na Figura 5.3.

Estas transformações são, basicamente, a essência do processo de desenvolvimento do MDA. Um modelo serve como dados de entrada para uma entidade de transformação que irá gerar dados de saída em um outro modelo ou código.

A ferramenta de transformação apresenta os elementos envolvidos em desempenhar a transformação, e estes elementos possuem a descrição de como um modelo poderá ser transformado (definição da transformação). Segue a estrutura básica de uma ferramenta de transformação.

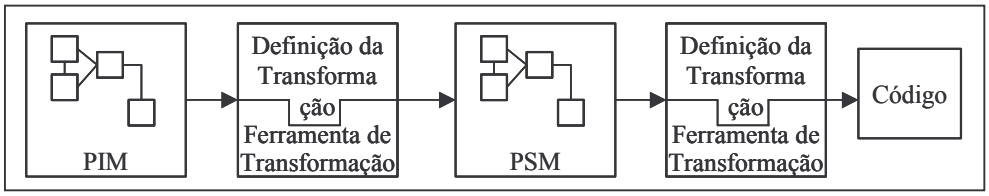


Figura 5.3 – Modelos de Transformações em MDA.

Pode-se, por exemplo, criar a definição da transformação em UML para C++, que descreve o que o C++ vai gerar para qualquer modelo UML.

O UML é poderoso para modelar aspectos estruturais de um sistema. Isto é feito, principalmente, através do uso do modelo de classes, que facilita a geração de um PSM. O ponto fraco do UML está na definição dos PIMs que especificam a parte comportamental ou dinâmica do sistema. O UML apresenta alguns diagramas para modelos dinâmicos, contudo através de um diagrama de interação ou casos de uso não se consegue chegar a um código específico eficiente para uma determinada plataforma.

CAPLAT e SOUROUVILLE, em [10], propuseram um formalismo para representar a definição da transformação entre os modelos do MDA.

Para ambos, um modelo é uma representação de um sistema em um formalismo. “Qualquer formalismo apóia-se em ontologias que determinam um conjunto de primitivas de modelagem e suas semânticas através de um caminho ortodoxo.[10]”

Em outras palavras o formalismo é um ponto de vista em que qualquer sistema deve ser considerado (ver Figura 5.4). O formalismo é composto de primitivas abstratas que em noções de modelagem são classes, heranças e restrições, por exemplo: que são mapeadas para expressões formais em UML, como caixas, setas, etc.

A semântica representa o significado destas noções (por exemplo, o que uma classe é?) descritos em uma linguagem.

Por exemplo, a semântica da herança determina que se uma classe *x* é subclasse de uma classe *y*, *y* não pode ser subclasse de *x*. Isto seria uma violação da semântica de herança.

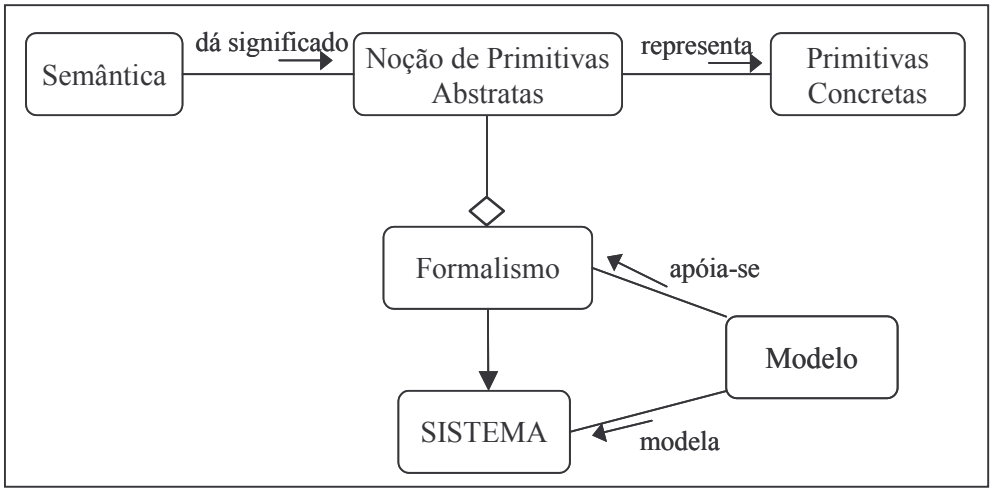


Figura 5.4 – Modelo de Sistemas baseado em Formalismo.

Conclui-se que o formalismo tem as características dos sistemas e que qualquer sistema pode ser modelado. Para criar um formalismo deve-se definir: uma ontologia que o apóie, um conjunto de noções abstratas e sua semântica. O modelo de um formalismo é chamado de metamodelo.

Conforme a Figura 5.5, dado *m(s)/f* como um modelo *m* de um sistema *s* em um formalismo *f*. A transformação é dada através de um mapeamento de $m_1(s)/f_1 \rightarrow m_2(s)/f_2$. O formalismo é composto pela UML mais a lógica do negócio, ou seja, $f_1=\{UML + Perfil\}$, $m_1(s)/f_1$ é um modelo independente da plataforma-PIM e $m_2(s)/f_2$ é um PIM enriquecido com os elementos do PSM. No final, $m_2(s)$ possui todas as informações específicas da plataforma.

Se m_1 for um modelo de um sistema (:Sistema é qualquer instância de uma classe sistema em UML) apoiado sobre o formalismo f_1. Neste caso m_1 necessita da linguagem de f_1.

As primitivas e a semântica de f_1 (considerando a herança) são modeladas por m_2. m_2 apóia-se sobre f_2.

Se for entendido o formalismo de f_2 conseqüentemente o formalismo de f_1 também o será.

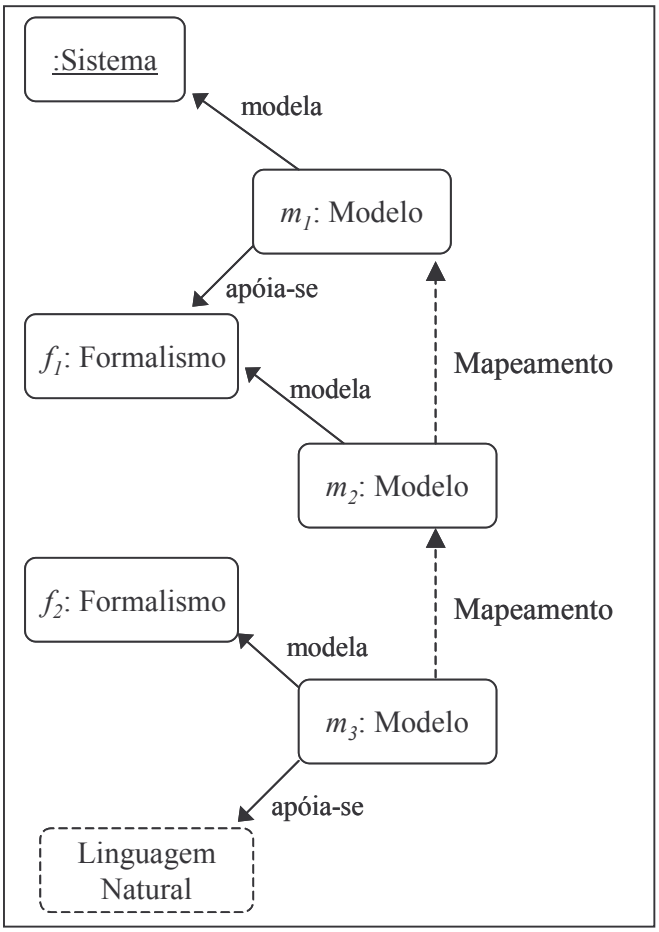


Figura 5.5 – Modelo de CAPLAT/SOROUVILLE

No caso do mapeamento do PIM para o PSM, o PIM pode ser representado pelo UML adicionado aos perfis do PSM. Por exemplo, o perfil UML para CORBA especifica como usar o padrão UML para definir interfaces IDL CORBA, structs, unions, etc.

As Figuras 5.6 e 5.7 apresentam exemplos do mapeamento PIM-PSM de um sistema de livraria.

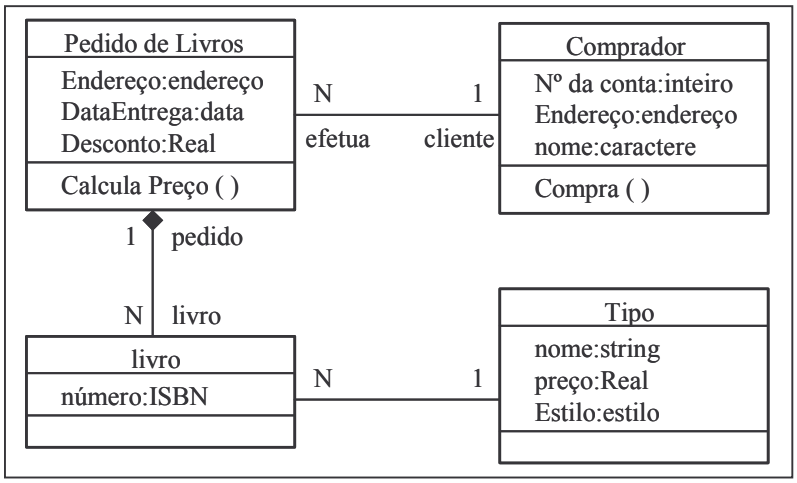


Figura 5.6 – Modelo Independente de Plataforma.

Na transformação PIM-PSM são inseridos modelos de bancos de dados, chaves primárias e estrangeiras, tipos primitivos de dados, etc.

- Uma string UML pode ser mapeada para um SQL VARCHAR (40)
- Um inteiro UML pode ser mapeado para um inteiro (32).

Algumas regras precisam ser aplicadas, pois o tipo de dados *endereço* não existe. Uma solução é fazer uma tabela separada para cada tipo de dado ou inserir o tipo de dados na linha do atributo.

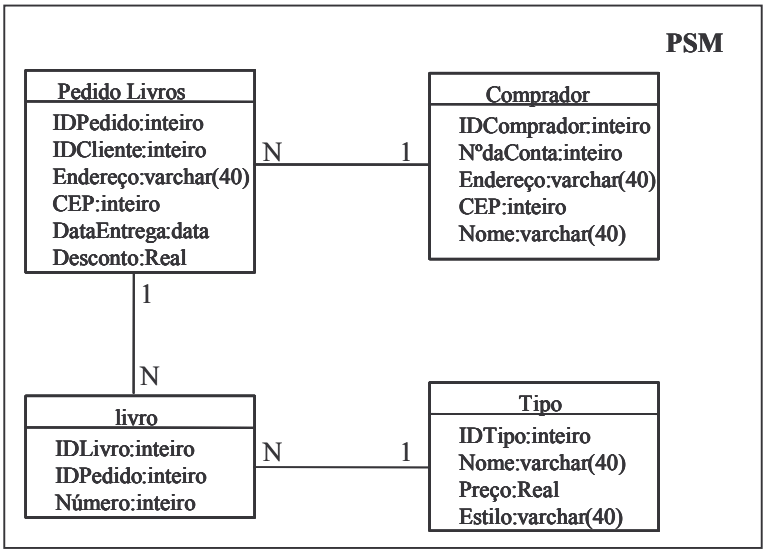


Figura 5.7 – Exemplo de Modelo Específico de Plataforma

Através da utilização dos modelos de MDA, os sistemas adquirem portabilidade (reutilização), interoperabilidade, facilidade de manutenção e extensibilidade.

Os benefícios oferecidos pelas novas tecnologias que surgirão podem ser adquiridos pelo mapeamento dos modelos para estas tecnologias. Programas Java podem facilmente ser convertidos para Visual Basic, ou integrados da plataforma Windows para Linux. O adicionamento de um recurso adicional ao sistema, através dos modelos, também é facilitado.

Um problema de EAI pode construir modelos independentes da plataforma, durante a fase de Modelagem do Negócio. Na perspectiva do negócio, isto permite o ganho de tempo, enquanto os requisitos específicos da plataforma são pesquisados e considerados. No caso da escolha de uma tecnologia que por algum motivo não satisfaça (performance e segurança, por exemplo), a substituição de tecnologia será menos traumática.

Os modelos também facilitam a incorporação de novos pedaços da lógica de negócios que não participaram de uma integração inicial. A integração de novos parceiros também é melhor contextualizada ao ciclo de vida do negócio.

A abordagem MDA também permite a engenharia reversa de um modelo específico de plataforma para um modelo independente da plataforma. Isto tem um grande valor na integração de aplicações legadas, onde através da abstração de um projeto de baixo nível pode-se chegar a construção de um modelo que possa ser bem entendido pela administração do negócio, para que possam ser elaboradas melhores estratégias para o futuro. O fato de não estar vinculado a fabricantes também ameniza os custos na execução do processo de integração.

Existem algumas ferramentas, recentemente lançadas, que tentam auxiliar o processo de transformação de modelos, como: ArcStyler, Kabira e Secant Technologies Model Methods.

A OMG descreveu o MDA como uma arquitetura que irá ganhar o mundo, por ser aberta à indústria e por utilizar padrões como o UML. O MDA também promete a criação de nichos de negócios, onde modelos de um domínio possam ser especializados e compartilhados em processos de EAI. A perspectiva é chegar a processos de integração “plug-and-play”, onde as interfaces de negócios serão destacadas dos detalhes técnicos da implementação.

Pesquisadores da área têm previsto que o uso do MDA associado ao desenvolvimento de Web Services é a tendência para o amanhã.

6. METODOLOGIA PARA INTEGRAÇÃO DE APLICAÇÕES EMPRESARIAIS COM WEB SERVICES USANDO MDA

Este capítulo apresenta a metodologia proposta para integração de aplicações corporativas, existentes nas empresas, com Web Services. Também é proposto um modelo arquitetural que fornece interoperabilidade aos Web Services construídos, através do uso do padrão MDA.

A Metodologia visa estabelecer um conjunto de técnicas para dar suporte ao problema de integração de aplicações empresariais e/ou legadas com Web Services, de forma que serviços de negócios destas empresas possam estar expostos na Web para interação com usuários, clientes e possíveis parceiros de negócios.

Conforme apresentado em capítulos anteriores, existem diversas tecnologias que possibilitam a integração de aplicações, inclusive com a Web. No entanto, as propostas existentes são na maioria, limitadas, dependentes de fabricantes, complexas e arriscadas.

Esta metodologia propõe a definição de uma seqüência de procedimentos e métodos que auxilie um determinado negócio a dar um passo correto nas inúmeras decisões a serem tomadas em integração B2C e B2B.

Pesquisas em desenvolvimento de aplicações[50] revelam que sistemas atuais devem ser desenvolvidos para rodar em redes, devem dar suporte a Web, devem ser distribuídos, reutilizáveis, utilizar o modelo de N camadas e principalmente devem ser interoperáveis (rodar em diversas plataformas e sistemas operacionais).

Para suportar a diversidade de tecnologias, utiliza-se o *middleware* para esconder a complexidade das conexões. Através do uso de abstrações, o *middleware* isola as aplicações das mudanças de plataformas, sistemas operacionais, redes e

protocolos que são implementados nas empresas. Contudo, como apresentado no estado da arte deste trabalho, existem diversos tipos de *middleware*, com abordagens de abstrações diferentes e que dificultam o acesso a diferentes tipos de aplicações.

No decorrer de alguns anos, a maioria das empresas, tem que reestruturar seus sistemas para se integrarem com outras aplicações ou proverem funcionalidades adicionais à versão inicial. A inabilidade da utilização de vários tipos de *middleware* é um grande obstáculo à criação de uma arquitetura de integração. Para permitir a utilização de múltiplos *middlewares* no processo de integração de aplicações, é necessário a abstração do próprio *middleware.*

A arquitetura de um sistema é uma especificação de partes e conectores do sistema, juntamente com regras de interação entre as partes usando estes conectores [51].

A arquitetura que propomos para integrar aplicações utiliza uma tecnologia de envelopamento (*wrapper*) do *middleware* em uma camada adicional à lógica implementada. Os Web Services promovem este tipo de abstração.A Figura 6.1 apresenta a arquitetura proposta para integração de Web Services com outras aplicações.

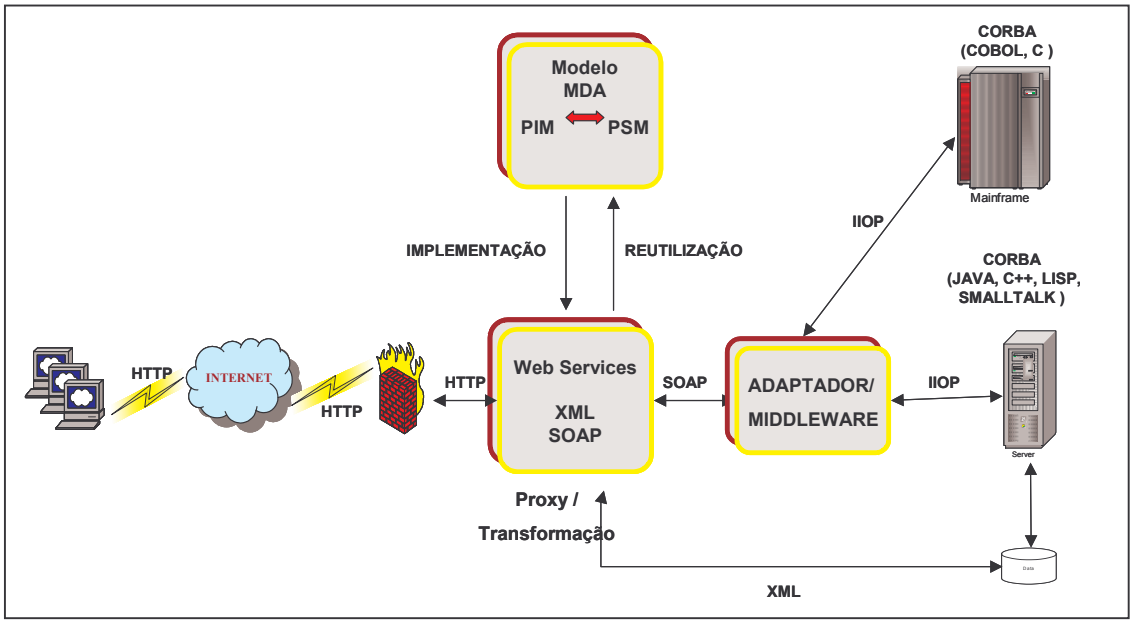


Figura 6.1– Layout da Arquitetura EAI vs. Web Services.

Através do protocolo SOAP, as mensagens entre as aplicações podem ser trocadas (criadas, enviadas e recebidas) em um nível de abstração, utilizando a técnica do envelopamento da mensagem. As APIs SOAP suportam HTTP nativo, escondendo do programador detalhes de transporte sobre a rede TCP/IP.

A linguagem de definição dos Web Services (WSDL) define duas partes principais:

- Uma parte lógica, chamada *port type* ou interface, que define as mensagens de entrada e saída.
- Uma parte física, que define as ligações específicas de protocolos para acesso aos Web Services.

Como o protocolo de comunicação proposto é o SOAP sobre HTTP, a descrição das interfaces poderá ser implementada em XML. Vinoski[52] relata que a vantagem do WSDL é a possibilidade de utilização de outras tecnologias de comunicação como: EJB, JMS e J2EE Connector Architecture. Contudo, nosso objetivo é explorar a interoperabilidade do XML, em mensagem SOAP, encapsuladas sobre o protocolo nativo HTTP.

Esta metodologia não propõe uma técnica para criação de interfaces Web, mas a integração dos sistemas existentes com a Web, usando o XML como um *middleware* semântico, através dos Web Services. Se as transações dos sistemas legados puderem ser acessadas via Web Services, outros Web Services, de outras empresas ou parceiros de negócios poderão trocar informações sem a interferência humana.

O MDA vem propor a capacidade de interoperabilidade, pois a lógica de negócio criada para suportar a Web poderá ser implementada usando diversas tecnologias e reutilizada por empresas que possuam domínios semelhantes. Isto também proporciona menor risco no processo de integração.

Utilizou-se a arquitetura direcionada a modelos, por proporcionar um curso no entendimento, projeto, construção, desenvolvimento, operação, manutenção e modificação de sistemas[53].

Baseado no MDA, foi levado em consideração três pontos de vista da integração de sistemas: o ponto de vista independente da computação, o ponto de vista independente da plataforma e o específico da plataforma.

Na fase de modelagem do ciclo de vida do negócio, pode-se inclusive detectar que a melhor solução para a empresa é a substituição do sistema existente.

A seguir são propostas algumas fases para o processo de integração de sistemas empresariais através da arquitetura direcionada a modelos de Web Services.

6.1 Fases propostas para o processo de integração entre aplicações

6.1.1 Fase 1: Análise do Modelo do Negócio.

Esta é a parte mais complexa e que demanda maior quantidade de tempo. Geralmente o modelo do negócio já é constituído por sistemas complexos com múltiplas operações, dependências entre componentes, heterogeneidade de recursos envolvidos (linguagens, protocolos de comunicação, sistemas operacionais).

A maioria dos sistemas em uso pelas empresas foi projetada para suportar necessidades individuais de algumas funções do domínio do negócio, e não do domínio como um todo. Tem alcance bastante limitado e são incapazes de evoluir com facilidade. Manipulação de grandes quantidades de dados é outra característica destes sistemas.

O domínio do problema deve ser bem estudado. É necessária a manipulação de toda a estrutura de vários sistemas de informação e o conhecimento do fluxo percorrido pelos processos envolvidos para o entendimento dos requisitos do domínio do negócio.

Uma organização é sempre diferente da outra, mesmo que pertençam à mesma área de atuação, e portanto tem que ser dissecada para que seja alcançado um entendimento maduro do negócio.

Em EAI não existem soluções prontas, pois cada ambiente possui sua particularidade, que são implementações realizadas no sistema ao longo do tempo, geralmente com a integração de componentes que criam cercas naturais entre as diversas partes do todo.

Esta fase necessita de requerimentos como investigação de papéis, pessoas e sistemas para definir quais informações participarão da integração, bem como a fonte destas informações, o formato, o caminho percorrido por ela no processo de negócio, as manipulações sofridas e as tecnologias envolvidas.

É necessário nesta fase:

- Utilizar a experiência do usuário. É complicado o trabalho de enumerar as interfaces existentes nos aplicativos legados, juntamente com os serviços oferecidos por elas, para propor uma integração com novas interfaces. Por isso é muito importante usufruir da experiência dos usuários para identificar os processos nos papéis de negócios.
- Identificação do mecanismo de segurança. Geralmente cada aplicação possui sua própria segurança, é necessário entender com este processo funcionará após a integração, para que não haja diversos

níveis de segurança com várias autenticações diferentes para acessar o mesmo serviço.

- Propor modelos reutilizáveis, adaptáveis e interoperáveis. O paradigma Orientado a Objetos de desenvolvimento de sistemas estabelece que um bom produto de software deve ser interoperável, quer dizer implementado sobre qualquer plataforma e sistema operacional. Além disso, deve ser reutilizável por aplicações futuras, o que é nativo da orientação a objetos e aperfeiçoado com o uso de padrões de semântica (independência da linguagem de programação).
- Desenvolver aplicações extensíveis. Os sistemas desenvolvidos devem ser extensíveis, pois as empresas estão alargando cada vez mais suas fronteiras e se integrando com novos parceiros de negócios a cada dia.

A técnica usada nesta fase é a investigação através de cenários, que gera uma especificação de artefatos com atividades definidas.

Sugere-se para esta fase a criação de casos de uso, para identificação do comportamento do ambiente e não para identificação do comportamento do sistema como proposto pela UML. Contudo a representação fornecida pela UML para Atores e Relacionamentos pode ser utilizada nesta fase, uma vez que tem sido a representação mais usada em MDA.

6.1.2 Fase 2: Criando um Meta-modelo em MDA.

O ciclo de vida, do desenvolvimento de sistemas proposto utiliza MDA como requisito para prover a interoperabilidade através da construção dos modelos propostos por este padrão, já mencionado anteriormente.

Para que seja criado um modelo de negócio é necessário primeiramente identificar os dados, o modelo do banco de dados, formato dos dados, parâmetros de segurança, integridade, procedimentos armazenados (*stored procedures*) e aplicações que o acessam. A identificação dos dados e do seu fluxo deve ser o primeiro passo no processo, uma vez que a informação que ele apresenta constitui-se, na maioria das vezes, o produto final no desenvolvimento de qualquer sistema.

De posse das características dos dados, deve-se construir o meta-modelo do negócio. A proposta do meta-modelo não é só descrever sistemas, mas o ambiente como um todo, que pode constituir-se de um conjunto de partes de sistemas sob o controle de um conjunto de pessoas que desempenham atividades diferentes.

O meta-modelo não definirá somente as estruturas de dados existentes na empresa, mas como todas as estruturas de dados existentes interagirão dentro da solução de integração de aplicação.

Esta metodologia sugere a utilização da arquitetura MDA para construção dos meta-modelos, partindo de um modelo independente de plataforma.

O PIM é construído abstraindo-se os modelos, ferramentas, tecnologias e plataformas. O modelo do dado lógico é uma visão do dado do negócio integrado.

O PIM deve conter as abstrações de comunicação (envelopamento), interfaces, processos, acesso à lógica existente e interpretação dos dados.

Representado como um PIM, um conjunto de aplicações usadas por uma companhia ou indústria pode ser importado por uma ferramenta de modelagem baseada em MDA [11].

Como a intenção deste trabalho é integrar sistemas empresariais com Web Services, as interfaces têm que prover suporte a Web e devem exibir dados semânticos formatados dinamicamente. Os Web Services são modelados através dos

conceitos MDA proporcionando capacidade de reutilização, refinamento e futuras integrações.

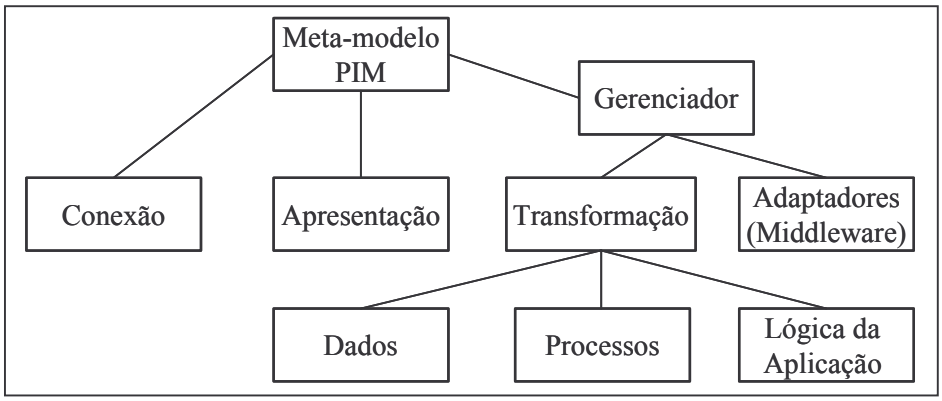


Figura 6.2 – Níveis do PIM proposto.

O PIM proposto por este trabalho deve fornecer um framework que alcance todos os requisitos para integrar uma aplicação legada com Web Services.

Conforme apresenta na figura 6.2, existem alguns níveis que envolvem este tipo de integração.

- Conexão: A conectividade é o âmago da integração de aplicações e constitui-se na habilidade de extrair informação através da invocação de serviços remotos. O valor deste tipo de tecnologia dá-se pela capacidade de integrar sistemas complexos e gerenciar a produção da informação e serviços fora destes sistemas e ao mesmo tempo publicar esta informação no formato necessário.
- Nível de Dados: o modelo deve apresentar *links* de comunicação entre os dados, o que requer um estudo aprofundado sobre estrutura de base de dados. Talvez seja necessária a definição de mapeamentos e transformações sobre os dados ou a criação de um repositório de dados que conterá as informações relevantes ao processo de integração.

- Nível de Transformação: será determinado o estilo da transformação a ser empregado sobre os dados. Como a proposta possui a visão de interoperabilidade e reutilização, o uso do XML será empregado como representação das interfaces e dos dados envolvidos no processo de integração.
- Nível da Lógica de Aplicação: permitirá o acesso à lógica de negócios e aos dados através do estabelecimento de rotinas de segurança, bem como poderá prover recursos adicionais ao sistema legado existente.
- Nível de Apresentação: a apresentação será desenvolvida sob a forma de Web Services, utilizando o XML como meta-linguagem através da criação de estilos XSL que venham representar as características do domínio em questão.
- Nível de Processos: que será apresentado sob a forma de serviços que endereçam os processos de negócios existentes providos para acesso pelos parceiros de negócios e clientes dos Web Services.
- Nível de *Middleware*: a proposta não descarta a utilização de qualquer *middleware* existente para prover a integração entre aplicações, contudo como a especificação CORBA ainda não provê um mapeamento eficiente para a linguagem dos Web Services, a arquitetura proposta vem prover uma solução que forneça interoperabilidade, através do XML/SOAP, ao acesso a objetos distribuídos CORBA/Java.

Com este paradigma de independência tem-se como resultado uma melhor visibilidade e entendimento do modelo do negócio, bem como dos processos, funções e características – o que futuramente pode ser combinado e reutilizado.

Alguns arquitetos de integração de aplicação usam UML para definir este modelo. Criar um modelo de negócio não visa estabelecer procedimentos para modificar códigos fontes de aplicativos, mas documentar o ambiente da integração.

6.1.3 Fase 3: Criando a Arquitetura e o Modelo Específico da Plataforma.

Nesta fase serão apresentadas as tecnologias e o detalhamento do funcionamento dos mecanismos pertencentes ao processo de integração, bem como será fornecido o esboço da arquitetura proposta.

A partir do modelo do processo do negócio o arquiteto identificará os novos processos a serem criados para auxiliar o processo de integração.

O uso de padrões simplifica a criação da arquitetura da aplicação e a aplica à arquitetura da empresa. Os modelos formalizam a idéia, descrevem o contexto do problema, permitem a geração de esquemas de estilos, definem responsabilidades, relacionamentos e permitem a reutilização. Esta metodologia pretende potencializar a capacidade de reutilização e interoperação, por concordar, que esta geração do desenvolvimento de software requer a construção de sistemas reutilizáveis e capazes de serem implementados sobre qualquer tecnologia.

A reutilização é enfocada através do uso dos modelos e da linguagem XML, que permite a criação e a adaptação de novos serviços, além de ser um padrão universal. Os Web Services propõem a inclusão de uma máquina de execução virtual que pode ser portável a novos ambientes, rapidamente.

O encapsulamento dos serviços é que garantem a independência de mecanismos, fazendo-os reutilizáveis em inúmeros contextos.

A arquitetura dos Web Services é, originalmente, em 3 camadas cliente, *proxy* ou interceptor e servidor, conforme apresentado na Figura 6.3.

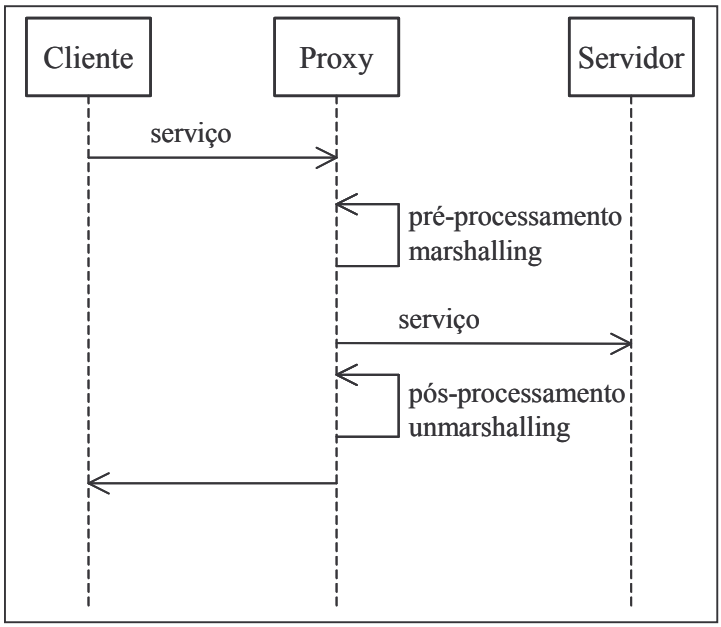


Figura 6.3 – Arquitetura dos Web Services.

O *proxy* é uma camada intermediária entre o cliente e o servidor, que agrega as interfaces através de um padrão. O *proxy* também esconde do usuário detalhes de distribuição, transporte, localização, protocolos, etc.

Em tempo de execução o *proxy* roda como um interceptador das requisições dos métodos clientes, obtém todos os argumentos de uma pilha, envelopa a informação em pacotes de acordo com o formato definido e transmite através do protocolo de comunicação para o servidor. Finalmente o *proxy* retorna os resultados do servidor, copia-os para uma pilha e retorna-os ao cliente.

Esta solução não é suficiente para a integração de aplicações, porque a informação de localização está separada do cliente e do servidor pelo *proxy*. Outro problema do *proxy* é a ativação/desativação, ele é ativado quando o serviço do *proxy* é inicializado e desativado quando o serviço é desligado, isto consome bastante recurso e diminui a performance.

Para solucionar este problema sugere-se a utilização de um *broker* (middleware) para fazer um mapeamento da referência lógica para referência física das localizações. O *broker* também manipulará as ativações e desativações.

O *broker* possui uma porta onde recebe as requisições e também possui uma biblioteca compartilhada e uma pilha. Para permitir a independência de linguagem , uma linguagem de descrição de interface deve ser introduzida, para estabelecer convenções de codificação e restringir as especificações de interfaces de acordo com as limitações impostas pelo modelo de objetos distribuídos [30].

Quando documentos e dados são trocados entre aplicações heterogêneas, é necessário um mapeamento para prover adaptação entre as entidades proprietárias para o formato que os clientes entendam (XMLSchema, XSLT, etc.). A Figura 6.4 apresenta o ciclo de vida de um Web Service integrado a uma aplicação legada através do uso de um servidor *proxy* ou adaptador, conforme proposto.

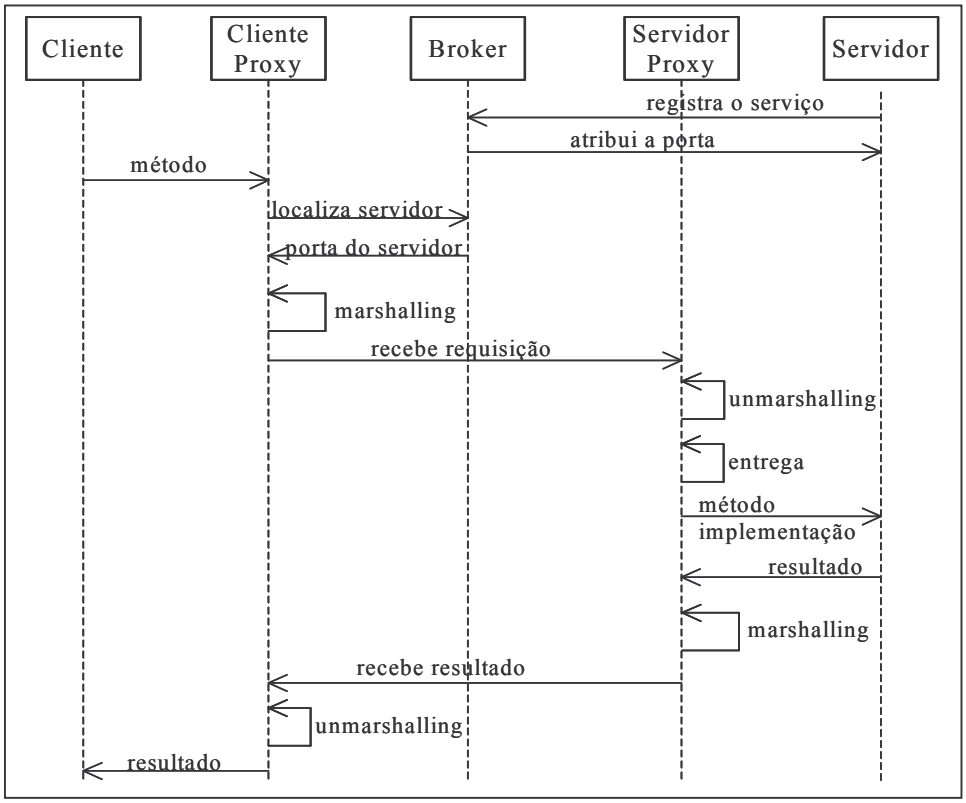


Figura 6.4 – Ciclo de vida de um Web Service integrado a uma aplicação empresarial

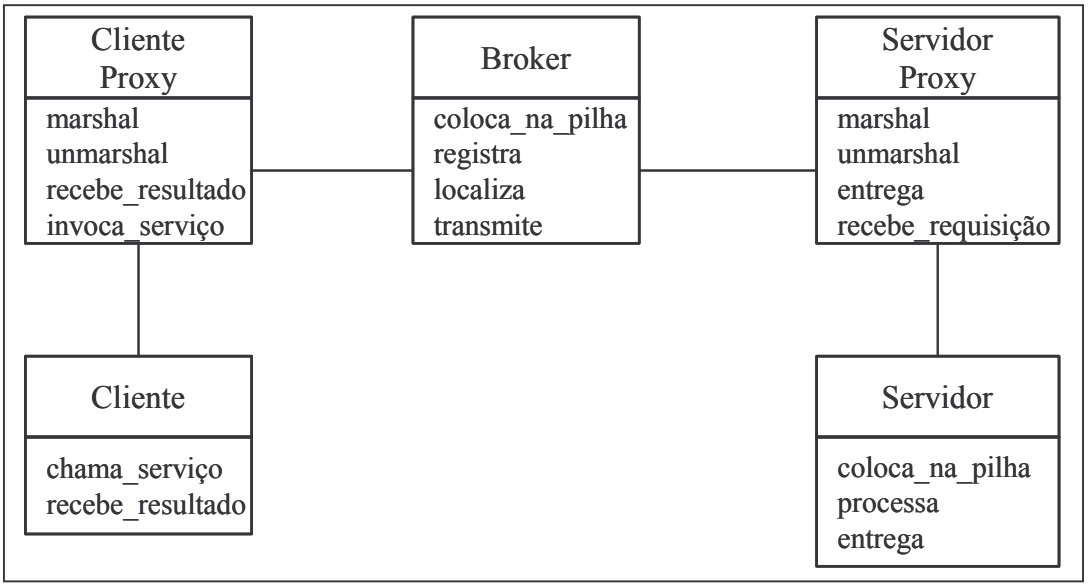


Figura 6.5 – Funções básicas do ambiente de integração.

A Figura 6.5 apresenta algumas funções básicas do ambiente de integração, conforme proposto.

Existem diversos *middlewares* orientados a objetos como CORBA, COM+, JavaRMI, cada um com seus conceitos arquiteturais comuns que diferem entre si em vários aspectos, como por exemplo, protocolos de comunicação e *marshalling*.

A especificação CORBA para a web [47] determina o nível de transporte através da utilização de *firewalls* mediante uma conexão bidirecional IIOP usada para retornos de chamadas a objetos e notificação de eventos.

O transporte de *firewalls* trabalha na camada de transporte (TCP) e define as portas 683 para IIOP e 684 para IIOP sobre SSL(*Security Socket Lay*er) – conexões seguras.

Os objetos CORBA necessitam dar um retorno de uma chamada ou notificar os clientes que o invocaram, para isso, os objetos agem como clientes e o módulo do lado do cliente instancia um objeto, que é chamado de volta em uma invocação em direção inversa. Como as conexões CORBA fazem invocações em

apenas uma direção, o retorno da chamada do objeto requer o estabelecimento de uma segunda conexão TCP para o tráfego na direção inversa, o que não é permitido pelos *firewalls* [12].

Além das portas TCP, é necessário a utilização de um *proxy*, definido pelo protocolo SOCKS, que serve como um canal de dados entre a comunicação cliente e servidor, através de TCP ou UDP. O cliente autentica-se no *proxy* SOCKS, faz suas requisições e o *proxy* conecta a requisição ao servidor de objetos, após isto, o cliente começa a transmitir os dados.

A OMG com o objetivo de promover a padronização no acesso de objetos CORBA através de Web Services, propôs em 2003, uma especificação de mapeamento da linguagem CORBA IDL para Web Services WSDL [13].

As duas propostas entregues a OMG, pelos participantes do consórcio, suportam apenas RPC codificado e não suportam RPC literal para comunicação via protocolo SOAP.

Apesar da recente especificação adotada, todos os tipos de dados CORBA ainda não foram mapeados, como a primitiva TypeCode do IDL que apresentou complexidade para este mapeamento.

A arquitetura proposta adotou um *middleware* baseado em XML/SOAP. A noção de middleware vem da possibilidade de transmissão de mensagens encapsuladas ou XML abstrato, que garante que as mensagens possam ser entendidas entre as aplicações que necessitam da informação.

Por causa do valor do XML, fabricantes de *middleware* têm tentado de alguma forma aproveitar os recursos desta linguagem, aplicando esta tecnologia para domínios de problemas de integração de aplicações.

Habilitar XML em um produto é simplesmente um problema de incorporar um *parser* (tradutor) dentro do *middleware* e determinar que o produto possa ler e escrever documentos XML, conforme representado na Figura 6.6.

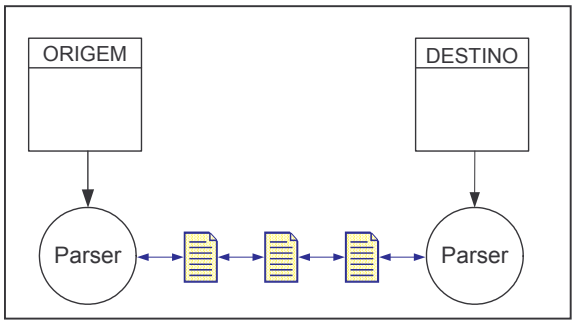


Figura 6.6 – Parser embutido ao *middleware*.

As aplicações corporativas robustas existentes são, na sua maioria, escritas em linguagens obsoletas, o que torna mais difícil a conversão da informação para XML e o uso do XML apenas como mecanismo de troca de informação cria um problema de baixo desempenho. Em várias oportunidades, pode-se converter os dados em padrão XML, no entanto mensagens binárias proporcionam melhor eficiência.

A maioria das soluções de integração de aplicações existentes envolve a criação de interfaces nativas e o uso do XML como um padrão de integração reduz a complexidade do processo. Em função disto, há uma tendência mundial no uso do XML para o desenvolvimento de novas aplicações, já fazendo uma previsão de uma futura necessidade de integração de informações.

O padrão XML apresenta um valor adicional, a inclusão de uma camada comum de metadados que pode existir entre um ou mais parceiros de negócios e também um mecanismo de transformação padrão como XSLT.

Conforme foi mencionado anteriormente, existe uma necessidade na transformação da informação para que elas se movam entre as aplicações. É importante lembrar que documentos XML são como mensagens e cada aplicação tem sua própria

semântica. As estruturas de dados e os conteúdos têm que ser semanticamente traduzidos para estabelecer a comunicação com outra aplicação.

Para transformar esquemas e conteúdos XML, o XSLT tem que desempenhar o processamento do texto e as operações de transformação, que incluem a criação de um formato de dados como: delimitados por vírgula, arquivos PDFs ou txt, conforme ilustrado pela Figura 6.7.

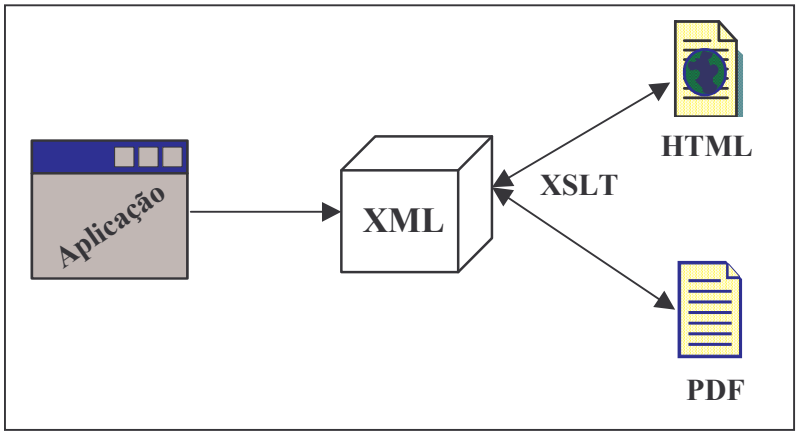


Figura 6.7 – Aplicação de estilos aos documentos XML.

A grande vantagem do XSLT, em EAI, é a possibilidade de definição de serviços comuns entre aplicações B2B. O XSLT é um padrão, desenvolvido pelo W3C, e tem obtido preferência na nova tendência de desenvolvimento de *middlewares* atuais que é a presença da semântica.

A tecnologia utilizada pela maioria dos *middlewares* suporta apenas o transporte de mensagens binárias, que é mais eficiente que a mensagem baseada em texto e é mais fácil de gerenciar e processar, mas não são facilmente adaptáveis à sistemas externos.

A criação de mecanismos de processamento de texto é bastante complexa. O software BizTalk tem começado a implementar algum processamento de mensagens, mas ainda efetua operações bem simples.

Certamente o XSLT irá transformar-se em um padrão para os produtos que trabalhem com textos XML.

Entende-se que há uma tendência híbrida na solução para integração de aplicações intra-companhia e integração B2B. As empresas precisam, em um primeiro estágio, aprender a integrar suas próprias aplicações, para posteriormente incorporar os serviços de seus parceiros de negócios. Para isso, é necessário conhecer bem o domínio do problema, identificar e entender os requisitos e estabelecer a melhor tecnologia para a solução.

A área pioneira no desenvolvimento de padrões[46] de integração é o comércio eletrônico, que tem procurado definir uma infraestrutura comum e entregá-la aos fabricantes de ferramentas.

Para resolver o problema da interoperabilidade é necessário inserir uma camada adicional à arquitetura proposta chamada de camada do adaptador, que é responsável por fazer a comunicação com o objeto legado usando a tecnologia melhor adequada.

A Figura 6.8 apresenta o lay-out da arquitetura global proposta para integrar aplicações empresariais com Web Services.

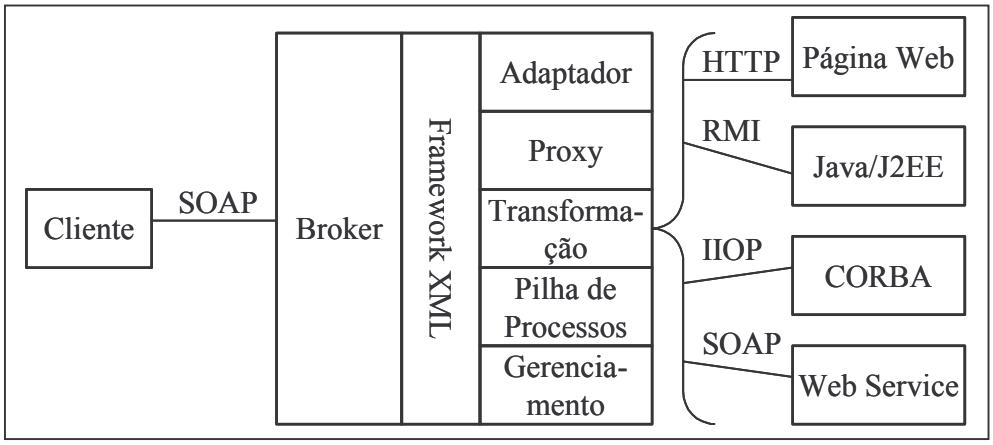


Figura 6.8 – Arquitetura Global EAI/Web Services.

6.1.4 Fase 4: Identificando as Transformações.

Com o entendimento dos dados é necessário aplicar a semântica ao domínio da integração.

Esta fase tem que estabelecer um estilo ou esquema para os dados que serão transformados. Este procedimento é necessário porque um sistema não conseguirá entender outro a não ser que os dados de ambos sejam reformatados para um padrão comum.

Isto também garantirá a consistência da semântica entre os sistemas. A arquitetura deve estabelecer um dicionário comum que contenha as informações de como a aplicação irá exportar seus dados. A camada de transformação contém um interpretador (*parser*) e métodos padrões que descrevem a estrutura do formato da mensagem. Quando a mensagem é quebrada em partes, os campos são recombinados para criar uma nova mensagem.

Existem servidores de aplicação que manipulam muitos tipos de informação, reformatando a informação usando uma interface com o cliente, que pode ser uma API.

Padrões dos Web Services como SOAP e WSDL garantem a interoperabilidade com outras arquiteturas [14]. O SOAP define o envelope de comunicação para troca de dados XML. O WSDL é uma linguagem de definição de Web Services que se deriva de um esquema XML. Na arquitetura proposta, recomenda-se o uso do próprio XML como linguagem de definição, uma vez que é mais fácil a adaptação das interfaces legadas ao XML do que ao WSDL, porque WSDL possui um conjunto de *tags* pré-definidas, e o XML permite maior flexibilidade ao desenvolvedor.

Todos os dados são transportados por um canal SOAP, que transporta a mensagem XML encapsulada, após serem transformados por um processador XSLT. Para o retorno da referência dos dados, a transformação reversa deverá ser feita, esta é a função do *proxy* que está contido no servidor Web.

IDLs suportam chamadas por valor e por referência, através de invocações usando *stubs* e *skeletons*. Com XML as invocações são suportadas através da transformação de parâmetros XML.

O transporte ocorre sobre uma conexão HTTP e ao conector deve a serialização das estruturas e dos estados do documento XML para transmissão pelo canal. O processador XSLT lê, filtra e transforma o documento XML para fazê-lo apropriado ao formato do receptor da mensagem. Um tradutor (*parser*) deve reconstruir a mensagem no receptor, conforme apresentado na Figura 6.9.

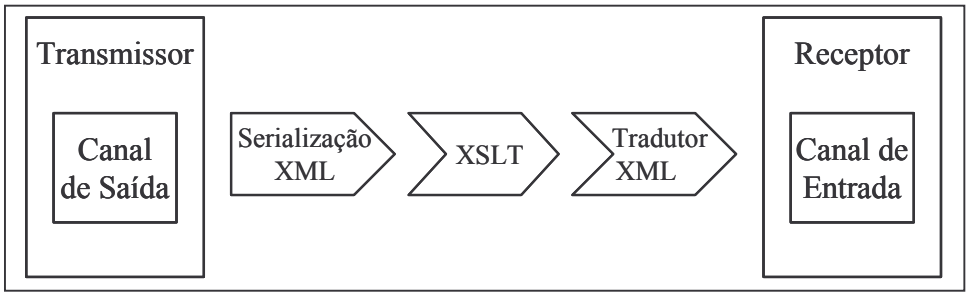


Figura 6.9 – Comunicação entre componentes XML.

Como objetos são simplesmente passados do transmissor ao receptor da mensagem e o Web Service necessita manter o estado da comunicação é necessário a utilização dos proxies que farão o serviço de localização do serviço, a transformação dos parâmetros e o retorno pelo mesmo canal de comunicação, conforme o esquema da Figura 6.10.

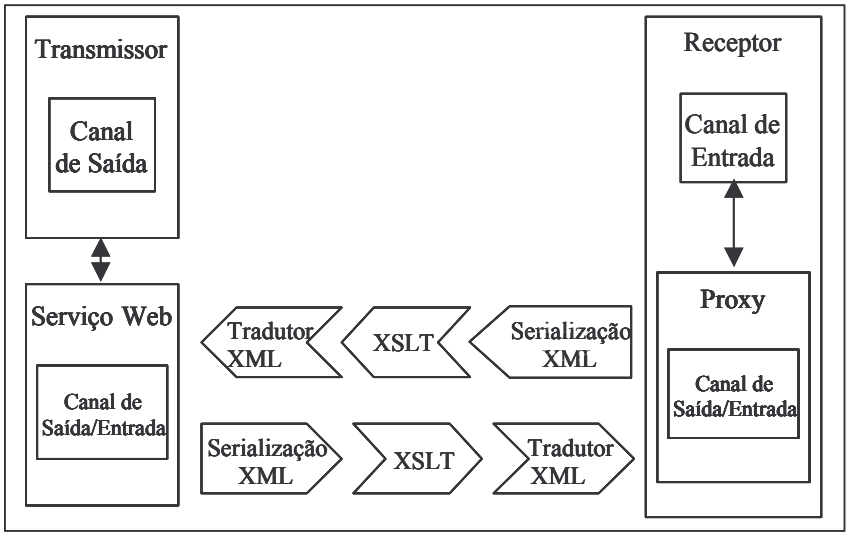


Figura 6.10 - Comunicação entre componentes XML usando *proxy*.

O serviço é representado pela descrição da interface que é derivada da interface da implementação. Os scripts XSLT devem definir a transformação dos serviços. O processador XSLT então fornece a função de serialização e de *proxy* para permitir a recursividade na conexão dos Web Services.

Em [15] é proposta a utilização do XQuery [16] para controle das chamadas e retorno através do XML.

6.1.5 Fase 5: Escolhendo as Ferramentas e Monitorando o Fluxo.

Muitas tecnologias estão disponíveis, incluindo diversos servidores de aplicação, servidores Web, tecnologias de objetos distribuídos, message brokers, etc.

A escolha da tecnologia correta para o domínio do problema é uma conseqüência das fases anteriores. Com base no modelo de arquitetura proposto, o arquiteto responsável possuirá os atributos para escolher a ferramenta ideal, baseado nas necessidades impostas pelos requerimentos detalhados nos modelos do negócio.

Deve-se entender as soluções oferecidas pelos fabricantes, pois um produto pode englobar mais funcionalidades de integração do que outros. Sugere-se a solicitação de versões de demonstração ao fornecedor do produto, sobre a tecnologia semelhante à do problema real. Uma escolha de uma ferramenta ruim põe em risco todo o projeto.

Após a integração das ferramentas é necessária a análise dos fluxos das informações sobre todo o modelo, já integrado, para verificação de persistência dos dados, segurança (controle de acesso), conectividade, performance.

Esta arquitetura utiliza como *middleware*, entre a aplicação legada e a camada de adaptação (*gateway*), o CORBA, por ser um padrão e por prover um mapeamento do IDL para uma grande gama de linguagens de programação existentes.

O *gateway* é necessário para desempenhar a mudança do protocolo de comunicação IIOP para SOAP.

O *gateway* aceita as requisições SOAP, faz a chamada ao serviço correspondente. Embutido no Web Service existe uma chamada a um objeto Web CORBA (através do *applet*) que executa sua função retornando a resposta, via IIOP, ao *gateway*. O *gateway* entrega a resposta ao *proxy* que aplica o estilo e a encaminha ao cliente do Web Service via SOAP.

Estão sendo lançadas no mercado algumas ferramentas que pretendem prover integração de aplicações legada com Web Services, destaca-se o IONA Orbix E2A Integration Platform e o IBM WebSphere Business Integration.

6.1.6 Fase 6: Teste e Implantação

Testes nunca são excessivos. Se a solução de integração não for testada exaustivamente, em todas as suas funcionalidades, o problema pode ficar mascarado e quando aparecer poderá causar grandes prejuízos. A grande vantagem desta arquitetura é a possibilidade de retorno ao ponto inicial sem a interrupção das atividades.

Se a solução proposta, por algum motivo, não satisfizer os requisitos, esta pode ser retirada e refinada até que alcance seus objetivos, tudo sem causar prejuízos na aplicação original, uma vez que foi preservada a integridade de seus códigos fontes.

A implantação deve ser estabelecida por fases: *middlewares*, servidores Web, servidores de aplicação, etc.

O desempenho deve ser avaliado de sistema a sistema, ou seja, desde a requisição do serviço até a resposta exibida no *browser* Web. Caso ocorra uma grande latência, deve ser observado o fluxo dos dados e analisado o tempo de resposta em cada passagem da informação. Alguns servidores de páginas Web possuem um depurador que pode auxiliar nesta tarefa.

7. ESTUDO DE CASO REAL

Com a finalidade de aplicar a metodologia de integração de aplicações empresariais proposta, foi desenvolvida uma solução utilizando o modelo e a arquitetura sugeridos para possibilitar a integração de aplicações legadas com Web Services. O estudo de caso em questão foi realizado em uma Instituição, que necessitava permitir o acesso às rotinas dos aplicativos implementados em *mainframe* através de clientes Web. Ou seja, os usuários dos sistemas corporativos deveriam acessar as rotinas destes aplicativos remotamente, através da Internet, como se estivessem na rede interna local.

A solução apresentada também visa prover o tráfego de informações por uma solução de *firewall* convencional, já implementado na Instituição.

A interface Web foi desenvolvida em XML Web Services, utilizou-se um *gateway* para estabelecer a conexão com o servidor e o protocolo SOAP para permitir a interoperabilidade e o prover a solução do *firewall.*

7.1 O Modelo da Instituição

A instituição em estudo possui todos os seus sistemas corporativos escritos na linguagem estruturada COBOL, rodando sobre o sistema operacional MCP (proprietário da UNISYS) em um mainframe UNISYS modelo Clearpath Series NX4800. Até aproximadamente o ano 2000, os usuários destes sistemas acessavam-no através de terminais “burros” T-27 conectados a uma rede SNA através de modems.

A partir do ano de 2001, foi implantada uma rede de computadores padrão Ethernet usando protocolo TCP/IP com acesso à Internet através de um link privado de comunicação de dados. No mesmo ano a Unisys forneceu um mecanismo de

hardware e software que foi implantado no mainframe para estabelecer a sua comunicação via rede TCP/IP. Juntamente com o suporte ao TCP/IP foi fornecido um emulador, chamado NXView, que possibilitava a emulação do sistema operacional MCP e conseqüentemente o acesso aos sistemas corporativos através de um microcomputador via rede.

O advento da Internet para dentro da Instituição, adicionado à necessidade de oferecer novos serviços a usuários/clientes levou à Instituição a publicar uma página Web. A página foi implementada dinamicamente utilizando a tecnologia ASP (Microsoft), publicado pelo Servidor de Páginas Web "Internet Information Server", também da Microsoft. Como os sistemas corporativos acessam uma base de dados no mainframe em DB2, foi instalado um outro servidor de banco de dados SQLServer (Microsoft) que replica apenas as informações que serão disponibilizadas na Web.

A Figura 7.1 apresenta o lay-out da topologia da rede desta instituição.

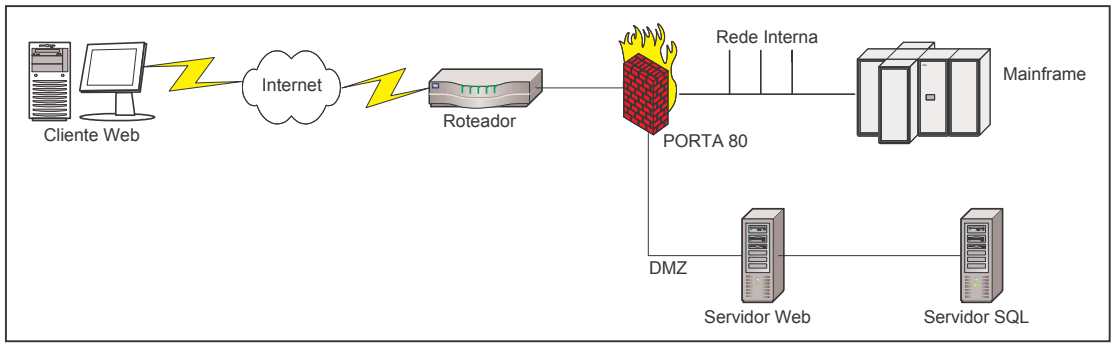


Figura 7.1 – Lay-out da rede corporativa da instituição.

Com a satisfação no uso dos recursos Web por parte dos usuários e clientes, a instituição pretendeu estender a utilização de recursos com a finalidade de agilizar a execução dos processos corporativos.

7.2 A problemática

Os administradores então requerem que usuários possam acessar, através da web, a lógica de negócios implementada nos sistemas corporativos no mainframe, em COBOL.

O usuário deve através de um browser Web executar as mesmas tarefas contidas nos sistemas corporativos, de uma forma segura e eficiente.

O acesso deverá ser feito tanto de máquinas internas, através do browser, quanto através da Web. Medidas de segurança de acesso devem ser levadas em consideração: autenticação, controle de transação, acesso a banco de dados, dentre outras.

Esta solução também objetiva a eliminação do emulador NXView, que até então tem que ser instalado em toda estação cliente que deseje acessar o mainframe.

Não pode-se deixar de citar que este emulador não suporta o Sistema Operacional Windows 2000/ME/XP e Linux, ficando restrito ao Windows 98 (conforma ilustrado na Figura 7.2), além de necessitar de algumas modificações no registro da máquina.

Com a solução Web, qualquer cliente usando um browser, independente do sistema operacional e de configurações adicionais, poderá acessar os serviços oferecidos pelos sistemas.

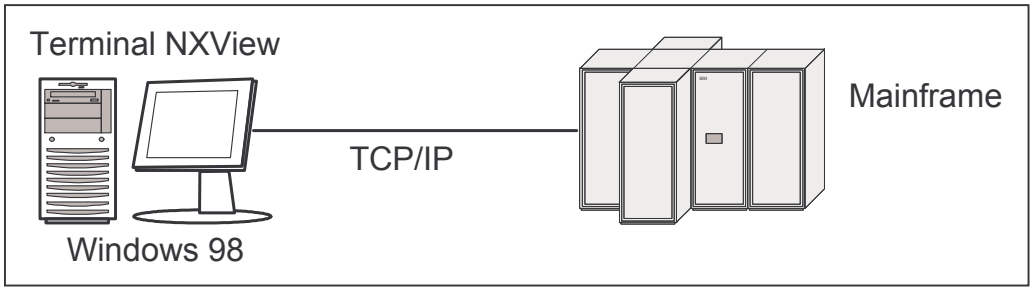


Figura 7.2 – Emulador NXView para redes TCP/IP.

A segurança do NXView é feita da seguinte forma:

- Na configuração do emulador na máquina do usuário é necessária a fixação de um número IP que será mapeado no mainframe.
- No mainframe o nome da máquina do usuário associado ao IP fixo fornecido é mapeado com uma estação cliente, através da inserção destes valores em um arquivo texto.

Alguns problemas podem surgir, por exemplo se o nome da máquina ou IP for alterado, a estação fica impossibilitada de acessar os sistemas.

7.3 Análise do Modelo do Domínio

Deve-se efetivar uma avaliação dos custos envolvidos, das tecnologias a serem empregadas, na viabilidade de execução deste projeto de integração, do tempo gasto, das pessoas que irão se envolver, dos riscos envolvidos, da estratégia de implantação, da performance, etc.

Os sistemas envolvidos são:

- Sistema Legado de Folha de Pagamento, desenvolvido em COBOL, adequado a características próprias da Instituição, em operação há 9 anos. Elevado custo de implantação, estável, exaustivamente testado e adequado à realidade da instituição.
- Sistema Orçamentário e Financeiro, que contém cadastro de contribuintes, fornecedores, prestadores de serviços, controle orçamentário, dívidas a pagar, receitas, cálculos financeiros, previsão, relatórios. Efetua pagamentos pela emissão de DOCs bancários. Em operação há 9 anos e de custo elevado.

- Sistema de Recursos Humanos, administração de pessoal, ativos, aposentados, cargos, funções, etc. Em operação há 6 anos, custo médio.
- Sistema de Atendimento ao Cliente, que contém cadastro dos clientes, emissão de carnês de pagamentos, quitação das dívidas, parcelamento, descontos. Em operação há 6 anos, custo médio.

7.4 Identificando a Solução

Em virtude do número dos sistemas corporativos embutidos no mainframe e dos altos custos com a aquisição deste equipamento e sistemas, é sugestivo a avaliação da possibilidade tecnológica de integração deste domínio com Web Services. Apenas a inviabilidade técnica seria um empecilho para a integração.

Para possibilitar a integração destas aplicações utilizando a metodologia proposta é necessária a criação de um modelo independente da plataforma para expressar o domínio do negócio.

7.5 Criando um Modelo Independente da Plataforma

O domínio do negócio deve apresentar uma representação dos processos do negócio. Pois, a integração será um processo adicional ao fluxo dos processos já existentes. Devem ser avaliados os impactos da inserção deste novo processo.

Após a integração, o sistema será visto como uma coleção de processos que abrange toda a estrutura da informação.

Isto inclui tipos e estruturas de dados, bancos de dados, instâncias de todos os tipos, interfaces clientes, fornecedores, orçamentos, funcionários, computadores, distribuição física de componentes de hardware e software, etc.

Este modelo deve ser independente de plataformas e tecnologia, especificando em uma linguagem formal, neste caso a UML, a mudança no ambiente da Instituição.

Dividir-se-á as responsabilidades em camadas apresentadas na Figura 7.3.

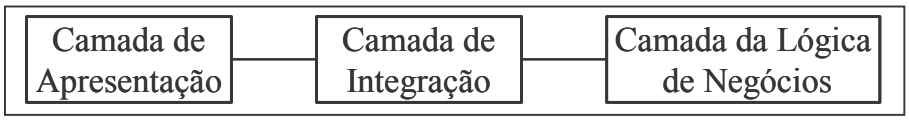


Figura 7.3 – Camadas do processo de Integração.

- Camada de Apresentação: é responsável por manipular as interações com o usuário final. Autenticação e apresentação das interfaces dos sistemas legados são algumas das responsabilidades desta camada.
- Camada de Integração: responsável por prover acesso aos serviços de "back-end", incluindo bancos de dados e lógicas dos sistemas internos, implantação do *middleware*, protocolos de comunicação. Este camada poderá implementar adaptadores, "parsers", "*gateway*s" e servidores Web para estabelecer o processo de integração.
- Camada da Lógica de Negócios: responsável pelo desempenho da lógica do negócio, neste caso, constitui-se nas aplicações legadas existentes.

Existe um conjunto chamado *esteriótipo* em UML, que estende os elementos básicos da modelagem para criar novos elementos. Assim, o conceito de um esteriótipo permite a UML ter um conjunto mínimo de símbolos que podem ser

estendidos para onde for necessário, de forma a oferecer a comunicação de artefatos que tem significado para o sistema em desenvolvimento. Nomes de esteriótipos são incluídos dentro de chaves angulares (<< >>), e colocados junto à linha de relacionamento. Os esteriótipos são usados para criar os relacionamentos de comando de utilização necessários. O esteriótipo <<comunica>> é acrescentado para mostrar uma associação de comunicação. Isto é utilizado porque uma associação é o único tipo de relacionamento permitido entre um ator e um comando de utilização, conforme apresentado na Figura 7.4.

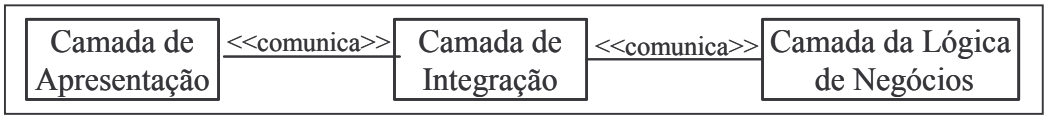


Figura 7.4 – Representação do relacionamento usando esteriótipos UML.

O metamodelo em UML é apresentado na Figura 7.5.

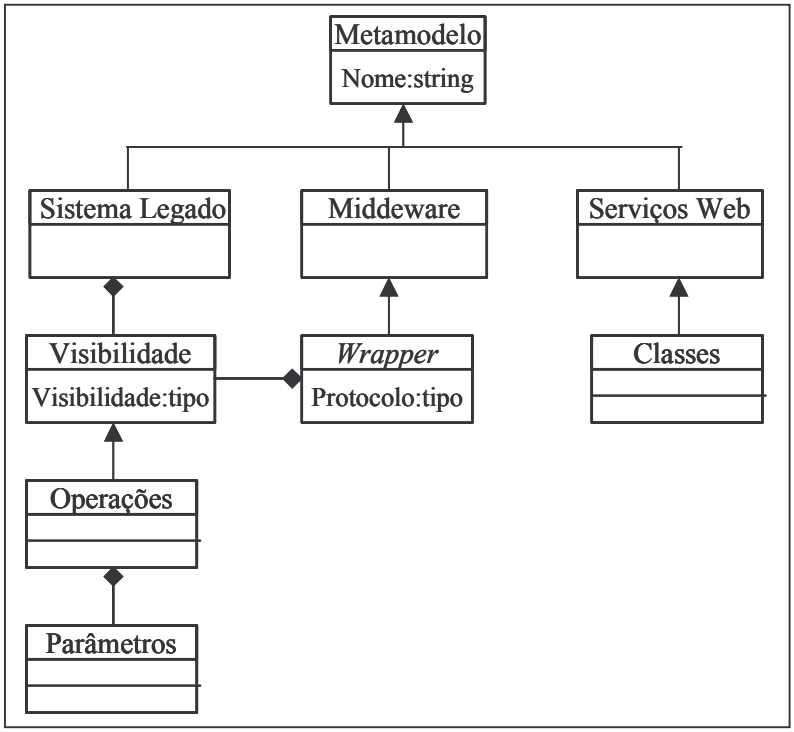


Figura 7.5 – Meta-modelo da integração.

7.5.1 Entendendo o Processo.

A tarefa a ser executada envolve a exposição das interfaces dos sistemas legados como um Web Service. Isto envolve a inserção de alguns componentes para prover funcionalidade, principalmente no quesito comunicação e interoperabilidade. A utilização de um Web Service como interface cliente deve ser baseada em uma linguagem semântica para proporcionar facilidade em futuras integrações ou adaptações.

A comunicação entre as aplicações depende da existência de um mecanismo de *middleware* para rotear as informações entre os ambientes heterogêneos.

Não serão feitas manipulações no código fonte dos sistemas legados, isto evita interrupções de operação do sistema do negócio, além de evitar riscos de danos aos mesmos. Caso a solução apresentada não venha satisfazer os requisitos do negócio, esta é retirada ou substituída do ambiente sem lesar a estrutura existente. Isto é a principal vantagem deste tipo de integração, pois proporciona segurança a todos os envolvidos no processo.

O mecanismo de comunicação deve possuir adaptadores que encapsulem o código necessário e distribua a informação através da rede de computadores, atravesse o firewall e seja acessado de uma conexão remota via Web.

A interface Web tem que ser apresentada como serviços para que usuários e parceiros possam acessá-los no registro dos serviços.

A Figura 7.6 apresenta o diagrama dinâmico do fluxo do processo (diagrama de seqüência).

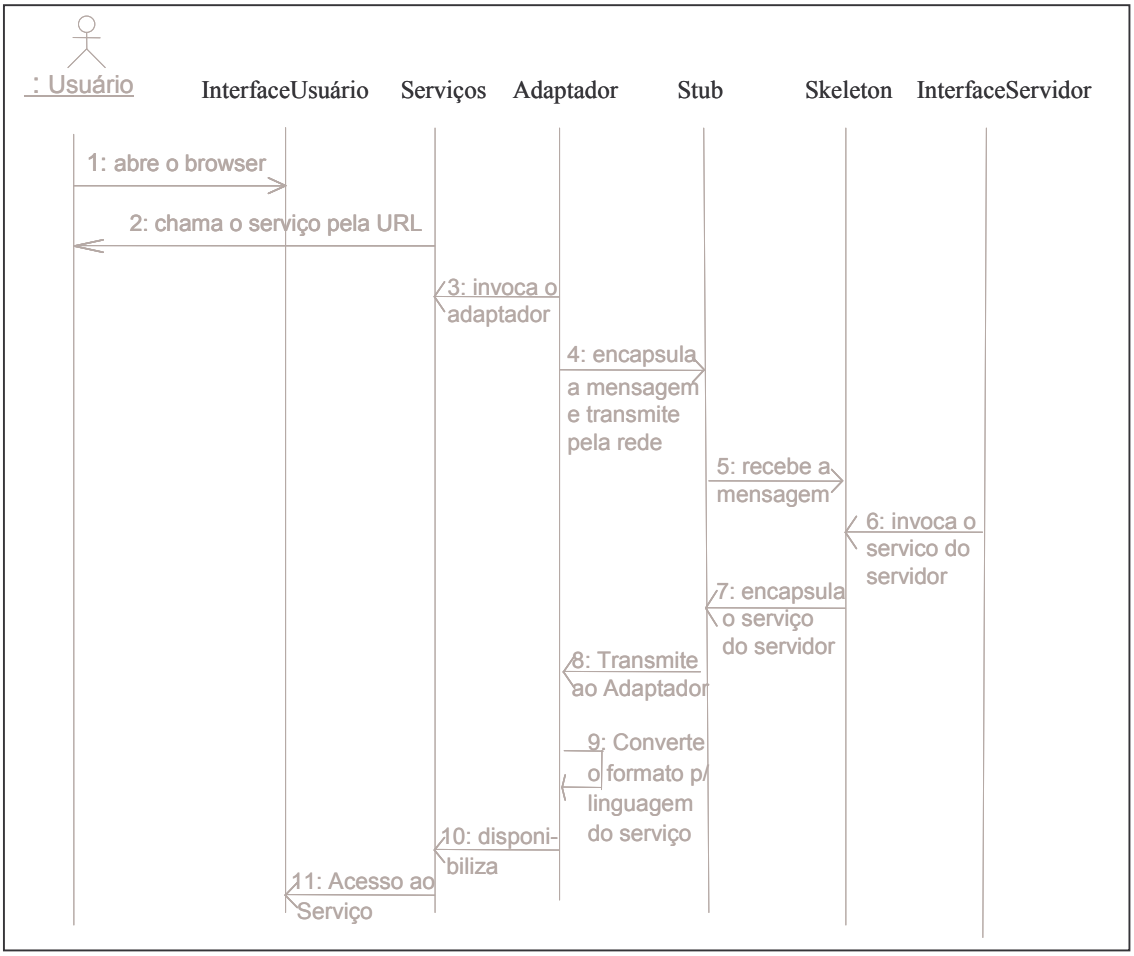


Figura 7.6 – Diagrama de Seqüência do processo de Integração.

7.6 Elaboração da Arquitetura Independente da Plataforma

Baseado no modelo de negócio em estudo apresentar-se-á o lay-out da arquitetura proposta, sem a especificação de plataformas ou linguagem de programação (ver Figura 7.7).

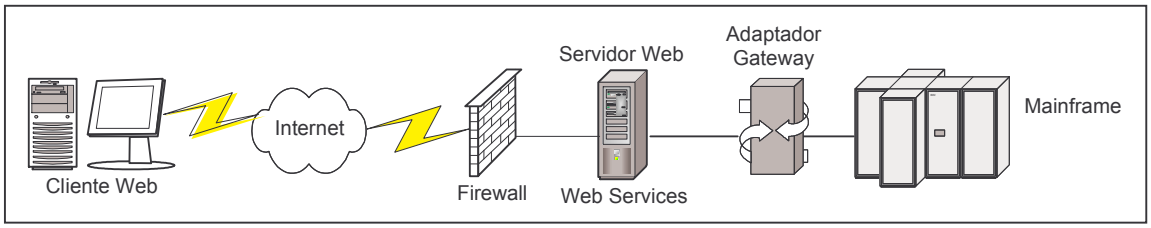


Figura 7.7 – Esboço da arquitetura independente da plataforma.

Os Web Services não são entidades dependentes de tecnologia, eles apenas fornecem um mecanismo para publicação de serviços sobre a Web, além de serem acessíveis usando URL, HTTP e XML que são padrões para o desenvolvimento de aplicações em qualquer linguagem.

O *gateway* é responsável pelo estabelecimento da compatibilidade de protocolos de comunicação, conversão de formato de dados, encapsulamento e transporte de dados entre os Web Services e a aplicação legada.

Baseado na metodologia proposta, que enfoca a semântica e reutilização este *middleware* deve prover funcionalidades comuns a diversas arquiteturas, ou seja, deve possuir uma abstração comum que permita diversos tipos de implementação.

7.7 Identificando as Tecnologias Adaptáveis à Solução do Processo de Integração

CORBA seria a principal tecnologia a ser utilizada neste processo, pelos seguintes fatores:

- Amplamente aplicável a diversas tecnologias.
- Permite invocação remota.
- Possui mapeamento para diversas linguagens, inclusive COBOL.
- Possui muitos serviços verticais disponíveis.

Apesar das vantagens, o CORBA não foi desenvolvido para suportar clientes Web. Na maioria dos casos são necessárias implementações no *firewall*, como abertura das portas 683 para o protocolo IIOP e 684 para IIOP/SSL, que tornam a rede interna passível de invasão.

Outro problema deve-se ao fato de que alguns *firewalls* usam serviços de *proxy* e ainda não existe uma solução IIOP para *proxies*.

Propõe-se a utilização do objeto Corba/Java para estabelecer a chamada dos sistemas em COBOL através de servlets Java. Isto irá resolver o problema interno, o emulador NXView será substituído pelo browser, que chamará um applet Java. Este applet invocará a aplicação servidor que será empacotada como uma página HTML.

O ORB utilizado será o Web Enabler da Unisys, ferramenta esta que permite suporte nativo ao MCP. Este ORB suporta Netscape 6 com Sun Java Plug-in versão 1.3.1_02 e Sun Java Script versão 1.3.1_02.

O Web Enabler funciona como uma console do ambiente MCP disponível para Web. A conectividade é feita através de sockets TCP/IP. O Web Enabler utiliza técnicas de integração, chamadas de "*screen-mapping*" por [17].

Esta técnica determina a localização de cada elemento de dados na tela do sistema legado, ou seja, a posição exata na tela em que a informação aparece ou o local onde um dado é inserido. Esta informação é retirada de um ponto estático da tela como uma seqüência de caracteres, então, esta é traduzida e convertida pelo adaptador.

Algumas vantagens do ORB escolhido são:

- Suporte nativo a Sistema Operacional MCP.
- Não requer modificação nas aplicações.
- É persistente na transação (sincronismo).
- Apresenta boa performance.
- Apresenta funções pré-definidas sob a forma de scripts.

7.7.1 Funcionamento do Web Enabler for MCP

O applet carregado pelo browser inicia uma conexão TCP/IP através do protocolo COMS do MCP, conforme ilustrado na Figura 7.8.

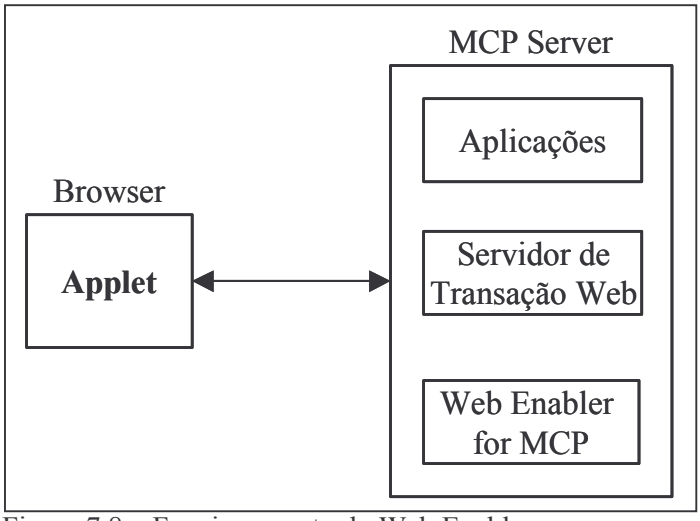


Figura 7.8 – Funcionamento do Web Enabler.

Os parâmetros da conexão são embutidos no applet, juntamente com alguns recursos como:

- Align: left, right, Top, Middle, Baseline, Bottom. Especificam o alinhamento do applet.
- Height e width: definição da resolução da tela.
- Background Color e Background image: definições de cor.

O Servidor de Transação carrega apenas páginas HTML que contêm o applet e provê conexão com o servidor MCP e suas aplicações.

O Web Enabler conecta com o servidor MCP através de uma porta TCP/IP que deve ser configurada no CCF (*COMS Custom Connect Facility*). Alguns atributos devem ser configurados, como mostrado na Figura 7.9.

```
ADD
TRANSPORT=TCPIP,        (Protocolo)
SOCKET=3001,            (Porta de Comunicação)
FRAMING=STANDARD,       (Cerca)
ACVT=LSCANDE,           (Texto)
MYUSE=IO,               (Serviço)
SCREEN=TRUE,            (Tela)
WINDOW=MARC,            (Janela transmitida)
CLOSEACTION=1,          (especifica a ação de fechar o browser)
LOGONREQUIRED=TRUE,     (Senha)
USERCODE=<USUARIO>,     (Usuário)
ENABLE PORT JAVAP3001;
```

Figura 7.9 – Atributos do Web Enabler.

Como a instituição utiliza o *Internet Information Server* da Microsoft como servidor de páginas Web. Será criada uma página ASP para fazer uma chamada ao applet.

O script pode acessar todos os métodos públicos definidos no applet e pode controlar as funções. O script e o applet podem comunicar-se entre si e rodar simultaneamente na mesma página HTML.

Existem algumas funções e parâmetros inerentes ao Web Enabler, a saber:

- *Transmit page in forms*
- *Line at a Time Transmit*
- *Show InfoLine*
- *InsertbyPage*
- *DeletebyPage*
- *Connect*

- *Disconnect*

O Web Enabler prove classes Java em com.unisys.net.Encoder que permite o mapeamento para outros tipos de codificação:

- Converter a string unicode transmitida pela rede, que engloba a interface do servidor, para um byte-array definido pelo usuário. Abaixo segue um exemplo desta implementação:

```
package myPackage;
import com.unisys.net;
import.java.io.UnsupportedEncodingException;

public class MyEncoding {

        public byte[] encode(String source)
          throws UnsupportedEncodingException {

              return <byte array>;
        }

        public String encode(byte[] source, int offset, int length)
        throws UnsupportedEncodingException {

              return <String>;
        }

        public Boolean isDouble(char c) {

              return<Boolean>;
        }
}
```

Uma função estratégica do Web Enabler é que o applet pode fazer *download* de arquivos JAR. Uma vez feito o download, o browser extrai e instala o conteúdo do arquivo Java comprimido no cliente.

Prosseguindo no processo de integração, é necessário promover características de interoperabilidade e extensibilidade ao ambiente e deve-se resolver o problema do acesso de usuários remotos através da Web, uma vez que foi visto que

ORBs CORBA não atendem estas requisições, e que a solução até então provê apenas as funcionalidades internas, ou seja, que não ultrapassem o *firewall*.

Através da metodologia proposta, considera-se que pode-se criar Web Services que encapsulem a chamada dos *applets* e scripts Java, sobre o protocolo SOAP que suporta HTTP. Estes serviços tanto iriam incluir uma espécie de tunelamento para acesso às aplicações legadas, quando englobaria outros serviços adicionais, em uma espécie de *extranet*. A idéia é aplicar semântica aos serviços para permitir sua execução sobre qualquer plataforma, sistema operacional, modelo de browser, etc, além de permitir facilidade de extensão para serviços com necessidades futuras de integração.

Além destas vantagens na criação de Web Services, parceiros podem interagir diretamente através dos serviços publicados em um registro global.

Utilizaremos o XML como linguagem de construção dos Web Services.

O XML Web Service expõe os serviços para os clientes através do protocolo SOAP. Os XML Web Services irão permitir que programas escritos em diferentes linguagens e plataformas possam se comunicar através do HTTP padrão da Web.

A linguagem XML serve como um *middleware* para troca de documentos, que são processos e aplicações, definidos por códigos ASCII, *tags*, comandos formatados, etc.

O código computacional é associado aos documentos XML através das folhas de estilo XSL. As XSL definem as regras da árvore de manipulação do documento e associa o comportamento aos componentes do documento. Estes componentes são ativos e podem ser expressos em diferentes linguagens de programação. Em outras palavras, um documento ativo é constituído de conteúdo, estruturas e comportamentos.

O XSLT suporta os conceitos de parametrização estática através das regras de template XSLT (elemento xsl:template); interação e repetição (elemento xsl:for-each); processamento condicional (elemento xsl:if e xsl:choose). A transformação XSLT pode ser usada com qualquer transformação genérica, que através de pré e pós condições pode especializar a transformação.

O SOAP envelopa (*wrapping*) as requisições e as respostas como documentos XML, o que garante que o cliente e o servidor podem ser escritos em qualquer linguagem de programação, transmite os documentos através da porta 80 (HTTP), suportado por todos os *firewalls*.

No lado do servidor o SOAP converte o documento XML representando a requisição no formato requerido pelo servidor. Depois que o servidor processa a requisição, o SOAP converte o resultado como um documento XML e através do *SOAP-response* é enviado de volta ao cliente, conforme apresentado na Figura 7.10.

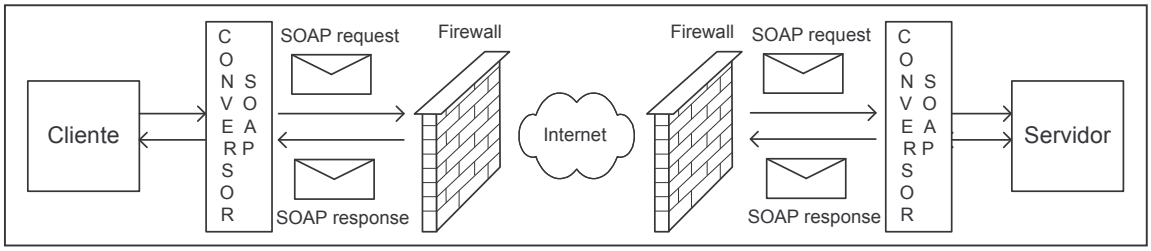


Figura 7.10 – Esquema de comunicação usando SOAP/Cliente Web ao Servidor de Web Services

7.7.2 Apache Tomcat Cocoon

O Tomcat é um contêiner Web usado para implementação de diversas tecnologias, como: tecnologia J2EE e XML. O Apache Tomcat [35] é um ambiente de código aberto que pretende ser um colaborador entre desenvolvedores e fabricantes por

todo o mundo. Ele também permite a inserção de diversas bibliotecas de adaptação a outras tecnologias.

O funcionamento dá-se da seguinte forma: quando inicializado o Tomcat funciona como um serviço, carregando todos os adaptadores relacionados. A partir daí, ele possuirá todo o controle do mapeamento das URLs. Quando chega uma requisição, é checado qual adaptador será responsável pela interpretação da mesma.

A porta padrão, utilizada pelo Tomcat, é 8080 podendo ser alterado caso necessário. O arquivo *server.xml* do Tomcat possui: a lista dos *hosts*, dos ouvintes, dos conectores e interceptadores, das facilidades e a configuração dos componentes instanciados.

O SOAP é utilizado como uma biblioteca de serviços no servidor Apache, também de código aberto, serve para anexar mensagens XML e de especificações Java. Como biblioteca o SOAP provê APIs para clientes invocarem serviços SOAP RPC, permitindo a transmissão e o recebimento de mensagens através deste protocolo.

7.7.3 Apache Cocoon e Tomcat

O Apache Cocoon é um framework de desenvolvimento Web que utiliza conceitos de separação de conteúdos e desenvolvimento Web baseado em componentes.

Estes conceitos são implementados no Cocoon através do conceito de *pipelines*, onde cada componente no *pipeline* é responsável por uma operação em particular.

As soluções Web são construídas e agrupadas no *pipeline* sem a necessidade de integração entre elas, isto provê o desenvolvimento em etapas e a redução de possibilidades de conflitos.

O Apache Cocoon também é um servidor Web para aplicações baseadas em XML, interoperáveis com soluções J2EE, HTML, WML, PDF, dentre outras.

O Cocoon possibilita a transformação de um arquivo XML com um estilo XSL, através de um XSLT embutido no servidor.

O processamento do Cocoon dá-se em três fases:

- A criação do conteúdo XML pelo usuário.
- O processamento do XML pelo Cocoon e da lógica (que está separada do conteúdo).
- Criação do documento aplicando o estilo (*stylesheet*) para o formato requerido: HTML, PDF, etc.

Internamente o Cocoon processa as requisições da seguinte maneira:

- A requisição é recebida em seu contexto original, a requisição deve indicar quem é o cliente que fez o pedido através da URL, o servidor identifica quem será o responsável pelo processamento do pedido, entregando-o para manipulação em um formato XML.
- O arquivo XML é analisado para que seja identificado quem processará a requisição (através das Tags XML).
- A resposta é formatada adequadamente e representada novamente em XML que é entregue ao cliente.

A Figura 7.11 apresenta o funcionamento do Apache Cocoon.

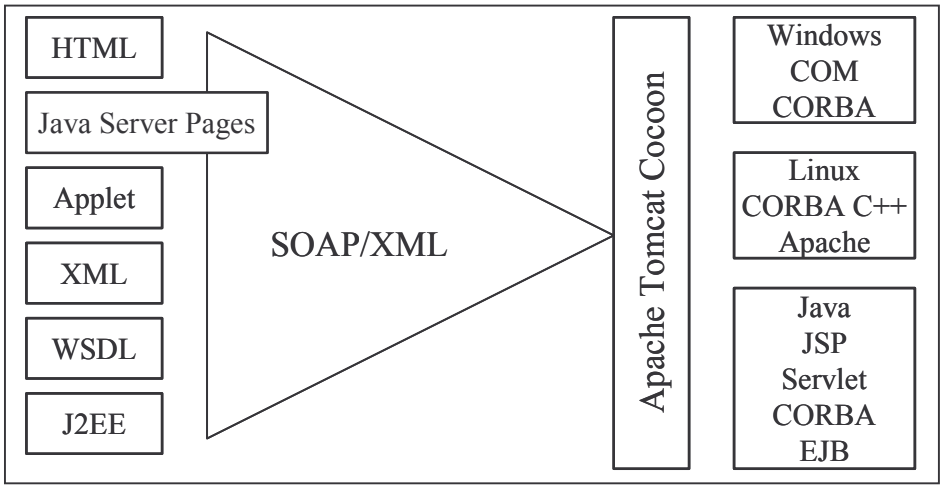


Figura 7.11 – Apache Tomcat Cocoon.

7.8 Modelo Específico da Plataforma

A Figura 7.12 apresenta a arquitetura específica da plataforma a ser implementada ao processo de integração proposto neste estudo de uso.

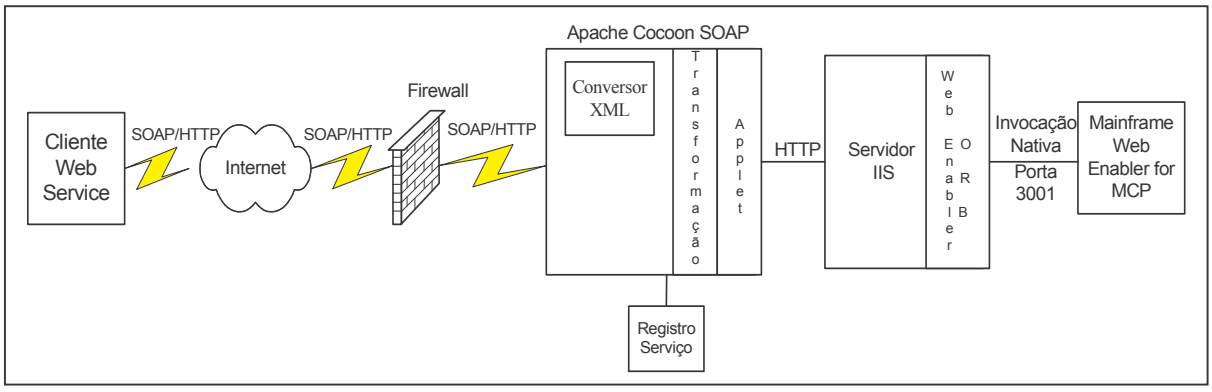
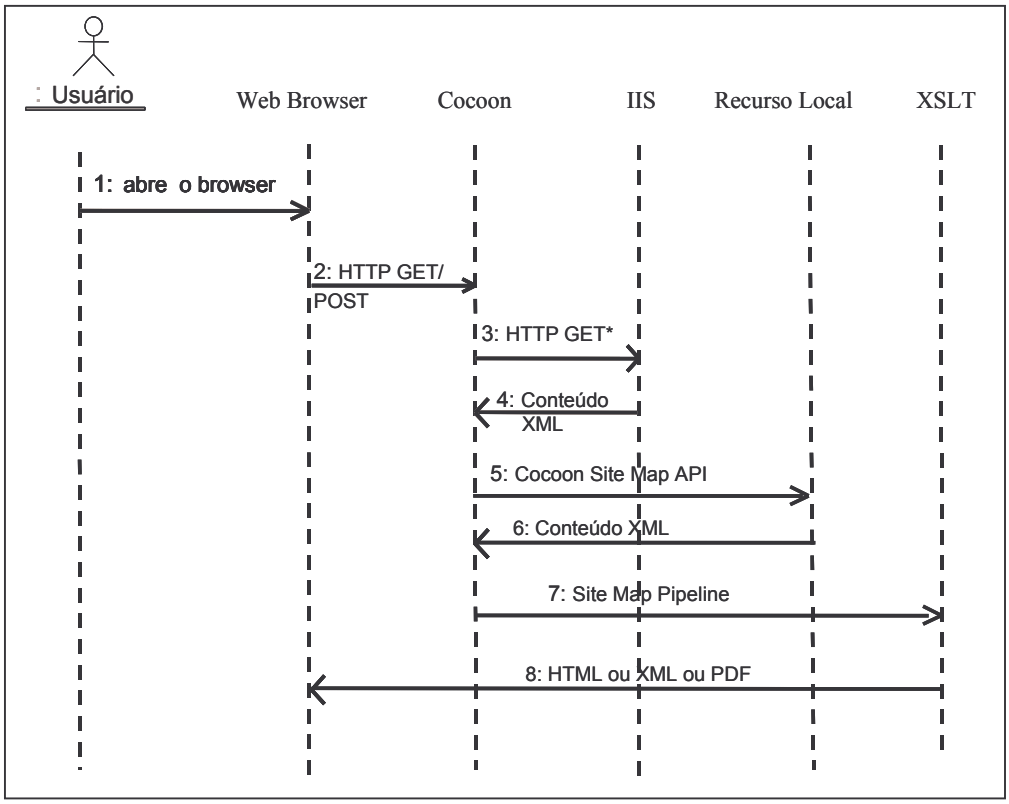


Figura 7.12 – Arquitetura Específica da Plataforma.

7.8.1 Diagrama de Interação do acesso ao Web Service no Cocoon

A Figura 7.13 apresenta o Diagrama de Interação do acesso ao Web Service no Cocoon.



* através do WebServiceProxyGenerator

Figura 7.13 – Diagrama de Interação do acesso ao Web Service no Cocoon.

7.8.2 Funcionamento e Interfaces

Na rede interna da Instituição, a interface dos aplicativos, em COBOL, do mainframe são apresentados pelo *applet*, no browser Netscape (único com suporte à solução Java/CORBA), conforme ilustrado na Figura 7.14. Contudo, a apresentação é impedida pela solicitação da abertura da porta 683 no *firewall*.

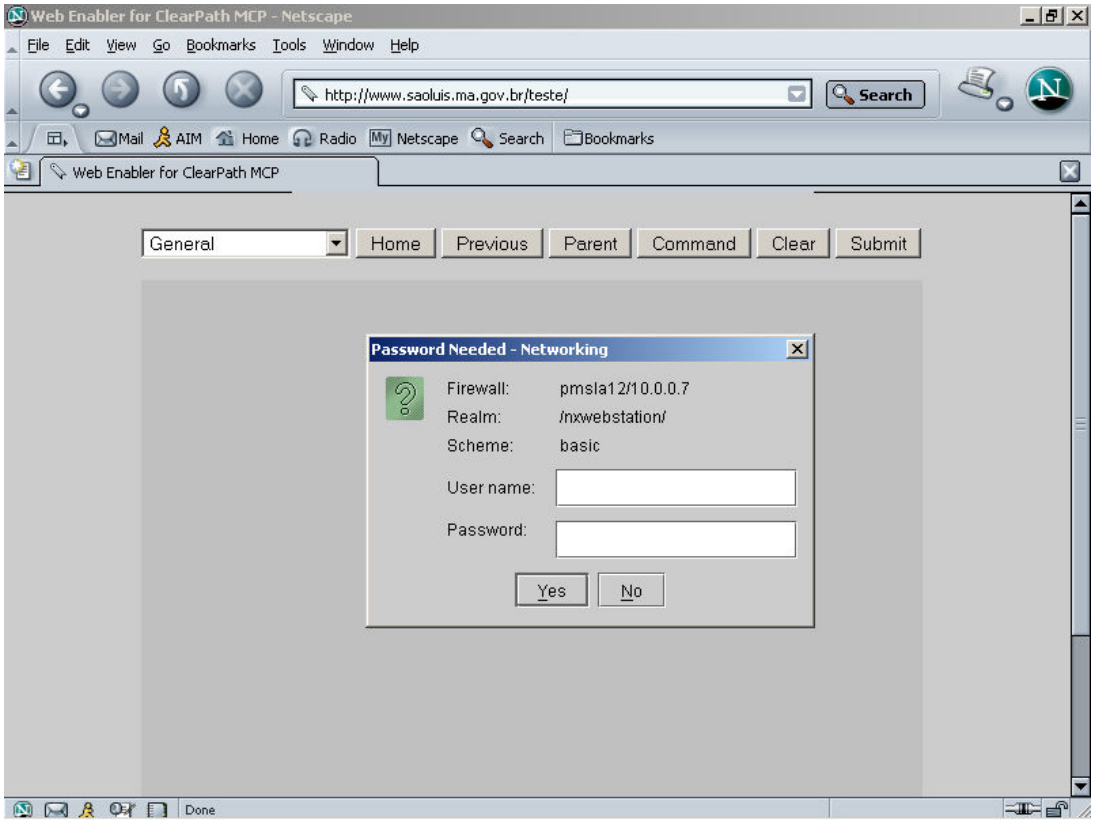


Figura 7.14 – Interface do sistema legado apresentado pelo applet.

Ao digitar a URL onde o Web Service está localizado, é disponibilizada a interface dos serviços. O *applet* que chama o serviço do *mainframe* está encapsulado em um documento XML, onde foi aplicado um estilo XSL. A informação é trafegada através do SOAP sobre HTTP, que soluciona o problema do tráfego através do *firewall*. Isto permite que a conexão Java/Cobol, possa trafegar pela porta 80(padrão HTTP), disponibilizando o acesso através de qualquer browser.

A título de demonstração de outros Web Services, principalmente da capacidade de integração, foi adicionado à mesma URL requisições a serviços disponíveis pelos *sites* da Amazon.com e Apache.com, apresentados sobre estilos XSL diferentes, como mostra a Figura 7.15.

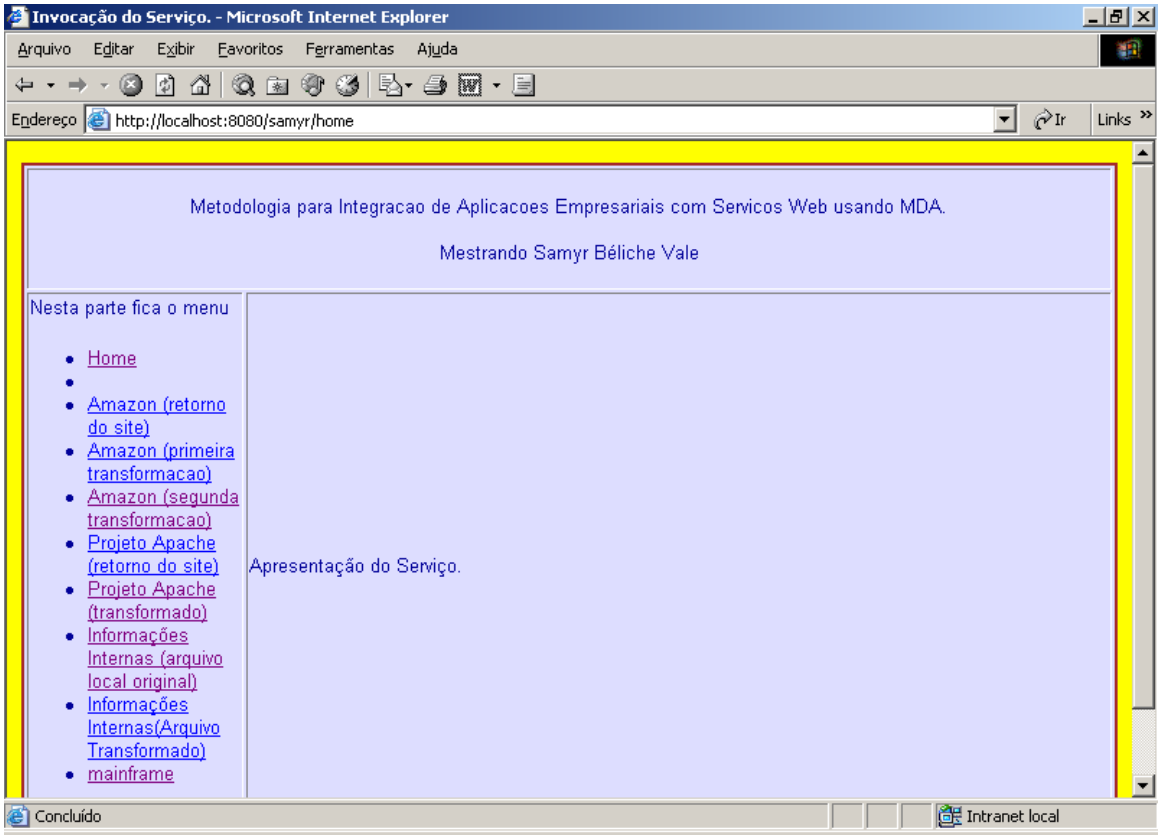


Figura 7.15 – Ambiente dos Web Services: integração com sistema legado, serviços Apache e Amazon.

A interface dos Sistemas COBOL, do mainframe, sendo apresentados como Web Service, encapsulados em documentos XML, como mostram as Figuras 7.16 e Figura 7.17.

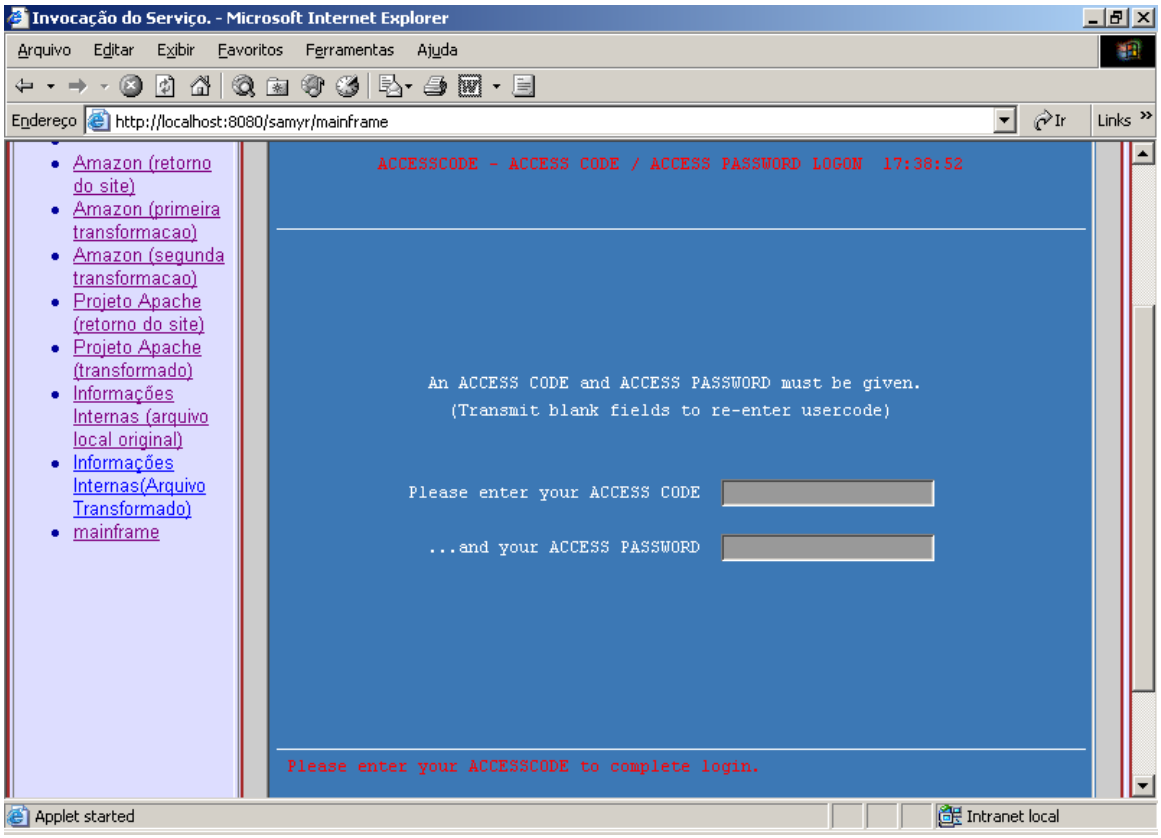


Figura 7.16 – Interface de segurança do sistema COBOL encapsulado no Web Service.

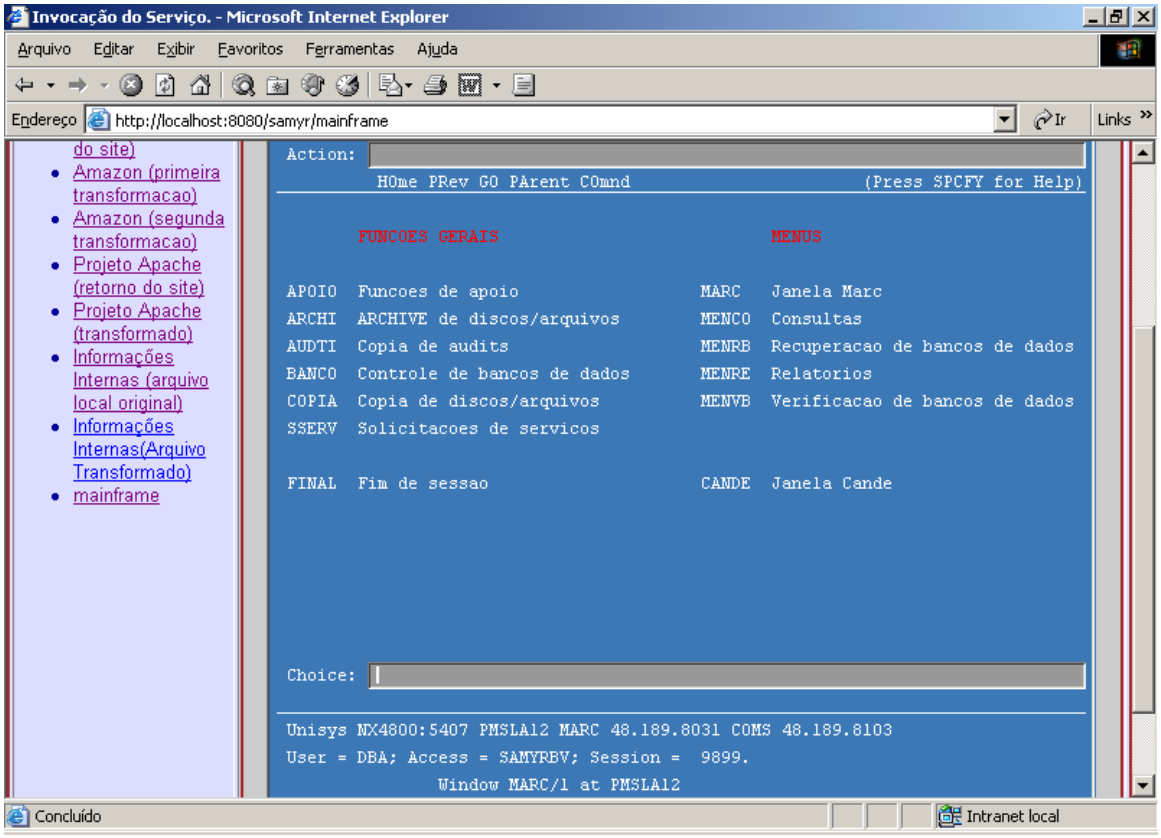


Figura 7.17 - Interface do sistema COBOL encapsulado no Web Service.

7.9 Teste

Alguns testes realizados a fim de verificar se o fluxo do negócio após a integração tem um desempenho satisfatório. Obedecidos os requisitos mínimos de hardware utilizando as tecnologias demonstradas anteriormente, enviou-se um pacote de 1920 bytes para um browser Internet Explorer 5.5. Os dados constituíam-se em uma tela do ambiente MARC do Mainframe, enviado através do Web Enabler for MCP.

O Servidor IIS manipulou cerca de aproximadamente 300 requisições por segundo. Baseado nisto, o Web Enabler com IIS é capaz de suportar aproximadamente 300 sessões.

O fabricante fornece algumas fórmulas para definir a performance:

P * ((300 * (1920 / M)) * T) = S

Onde

P = nº de processadores do mainframe

T = média da transmissão em segundo

M = média do tamanho da mensagem

S = nº máximo de estações concorrentes.

Esta fórmula serve apenas como uma base para estimar o tempo de vida da solução de integração apresentada.

7.10 Implantação

A solução proposta foi implementada, em fase experimental. As interfaces dos aplicativos COBOL contidos no mainframe, trafegam via XML/SOAP, que encapsulam o applet e scripts Java do Web Enabler *middleware*, e são apresentados

ao cliente Web, quando este digita a URL do Web Service. O cliente Web apenas necessita ter um browser e uma conexão Web.

É suportado qualquer browser em qualquer sistema operacional. Não foi necessária qualquer regra adicional ao *firewall* da instituição, o que garante segurança na solução.

A segurança do acesso à interface dos aplicativos é feita pelos mesmos, o acesso é permitido ao usuário através de uma senha. A única restrição é o mapeamento do nome e IP da estação no arquivo "TCP/IP map" do *mainframe* (isto proporciona uma segurança adicional, pois todos os clientes são conhecidos).

Para demonstrar mais recursos dos Web Services, foram adicionados outros serviços na mesma URL que solicitam de um *site* (exemplo, amazon) e transforma as informações recebidas em um estilo XSL (que pode ser modificado por outro cliente/parceiro de negócio que queira utilizar este serviço).

8. CONCLUSÃO

Este trabalho apresenta uma proposta de metodologia que venha determinar os procedimentos necessários para estabelecer um projeto de integração de aplicações com Web Services, que se constitui no enfoque atual em termos de tecnologia de integração.

Não existe uma metodologia padrão para integrar aplicações, pois a gama de técnicas, tecnologias e ferramentas que propõem facilitar o processo de integração acabam confundindo e dificultando a tomada de decisão.

A metodologia proposta apresenta algumas fases peculiares a todos os processos de integração de aplicações desta geração, que envolve aplicações multi-camadas, suporte à Internet, sistemas distribuídos, metodologia orientada a objetos, sistemas de rede, dentre outras.

Esta proposta também visa facilitar e agilizar as tomadas de decisão quanto às tecnologias empregadas, modelo de integração a ser utilizado e infra-estrutura a ser implementada.

Mainframes são plataformas mais poderosas para manipulação de grandes quantidades de dados e um grande número de transações, então, manter aplicações legadas em mainframes ainda é uma boa decisão. Contudo disponibilizar estas aplicações para novos usuários e parceiros através da Internet tem sido um grande desafio para arquitetos e desenvolvedores.

Disponibilizar aplicações legadas como Web Services permite agilidade, flexibilidade e segurança ao projeto de integração. Adaptadores e Gateways têm sido utilizados para prover a integração legados/Web.

Web Services são soluções de software que disponibilizam serviços remotamente via Internet, usando um padrão comum de identificação, descrição, localização e transporte.

Os Web Services são capazes de encapsular outros componentes, o que capacita-os a integrar aplicações legadas. Em outras palavras, os Web Services criam uma topologia de integração, totalmente independente de linguagem, plataforma ou sistema operacional. Esta é a necessidade do mercado atual, sistemas interoperáveis e reutilizáveis, que não fiquem obsoletos com a mudança da versão de uma ferramenta ou inoperantes em outra plataforma.

MDA é um novo princípio de arquitetura de software, extensível, flexível e adaptável a novas tecnologias e padrões. Ele abstrai os conceitos de implementação através da construção de modelos que expressem a lógica do negócio. Isto é ideal para o entendimento do processo de integração entre aplicações. O uso do MDA é parte integrante da metodologia proposta como responsável pela criação de modelos comuns de negócios e compartilhamento de informações entre processos de integração.

Por fim foi realizado em estudo de caso que propunha a disponibilização das interfaces dos sistemas corporativos de uma instituição como Web Services. Foi implementada uma solução através da metodologia proposta, para demonstrar a importância da integração de aplicações.

8.1 Trabalhos Futuros

Sugere-se para trabalhos futuros a utilização de tecnologias de *workflow* na metodologia de integração de aplicações, a fim de automatizar a identificação do

fluxo das informações do sistema integrante, bem como das outras atividades e processos pertencentes ao ambiente do sistema.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] *4. OMG. The Common Object Request Broker: Architecture and Specification. Revision 2.4.2. February, 2001.*

[2] W3C.Extensible Markup Language (XML) 1.0. W3C Recommendation, 1998.

[3] SOLEY, R. and the OMG staff. Model-Driven Architecture. OMG document Available from www.omg.org, November 2000.

[4] LINTHICUM David, Next Generation Application Integration, Addison Wesley, 2003.

[5] BRITTON Chris, IT Architectures and Middleware: Strategies for Building Large, Integrated Systems, Addison Wesley, 2002.

[6] COULOURIS, George, DOLLIMORE, Jean e KINDBERG, Tim. Distributed Systems: Concepts and Design. Second Edition. Addison-Wesley Publishers Ltd., 1999.

[7] TANEMBAUM, Andrew S. Modern Operating Systems. Prentice Hall, Inc., 1992.

[8] ZAHAVI, Ron. Enterprise Application Integration with CORBA: Component and Web based Solutions, OMG Press, 1999.

[9] ROTH, Paul. Moving to a Service Based Architecture, Friday, October 03, 2003.

[10] CAPLAT, G., SOURROUILLE J., Model Mapping in MDA, Workshop in Software Model Engineering – WISME 2002, Dresden, Germany, 2002.

[11] SIEGEL Jon, Using OMG's Model Driven Architecture (MDA) to Integrate Web Services, OMG, 2002.

[12] SIEGEL Jon, CORBA 3 Fundamentals and Programming, OMG Press, 2000.

[13] OMG - Object Management Group, CORBA to WSDL-SOAP Internetworking , orbos,2003.

[14] LOWE, W., and NOGA, M. L. A Lightweight XML-based Middleware Architecture.

[15] CAREY, M., BLEVINS, M., TAKACSI N., TA, Integration, Web Services Style,2003.

[16] W3C, XQuery 1.0: An XML Query Language, W3C Working Draft, August 2002.

[17] LINTHICUM David, Enterprise Aplication Integration, Addison Wesley, 2000.

[18] MICROSOFT. COM- Component Object Model. Microsoft, 2000.

[19] OMG. IDL to Java Language Mapping Specification, Version 1.1, June, 2001.

[20] *5. SUN. Java 2 Platform, Enterprise Edition ,2001.*

[21] *6. W3C. Web Services Description Language (WSDL) 1.1. Note 15 March 2001.*

[22] *7. W3C. Simple object access protocol (SOAP) 1.1. W3C. Note 08 May 2000.*

[23] *8. W3C. Universal Description, Discovery and Integration. - UDDI. Note May 2000.*

[24] *9. W3C. XSL Transformations (XSLT). W3C Recommendation, 1999.*

[25] *10. MICROSOFT .Net. Microsoft, http://www.microsoft.com/net, 2001.*

[26] *11. RATIONAL SOFTWARE CORP., Rational Objectory Process 4.1, September, 1997.*

[27] *12. RUMBAUGH James, Object Oriented Modeling and Design, Prentice Hall, 1991.*

[28]
13. *BOOCH Grady, Object Oriented Analysis and Design with Applications, The Benjamin/Cummings Publishing Company, 1994.*

[29]
14. *JACOBSON Ivar, The Object Advantage – Business Process Reengineering with Object Technology, Addison Wesley Publishing Company, 1994.*

[30]
15. *LAMBERT, John, et al. Distributed Objects in a Commercial Web Environment. IEE Coloquium on Distributed Objects – Technology an Application, vol. 4, 1997, 22 october, pages 1-4.*

[31]
16. *SUN. JDBC Data Access API Home Page, 2000.*

[32]
17. *OMG -Object Management Group, CORBAservices: Common Object Services Specification, Revised Edition, July 1997.*

[33]
18. *OMG -Object Management Group, Common Facilities Architecture, Revision 4.0, November 1995.*

[34]
19. *W3C. XML Schema Part 1: Structures. W3C Recommendation 2 May 2001.*

[35]
20. *W3C. Xerces Java Parser. Apache XML Project. 2002.*

[36]
21. *ORFALI R., HARKEY D., EDWARDS J., Intergalactic Client/Server Computing, Byte, Apr 1995, McGraw-Hill Inc.*

[37]
22. *IRMEN Jong, et al. Web Services/SOAP and CORBA. April27, 2002.*

[38]
23. *CHAPPEL D. A., and Tyler J., Java Web Services, O'Reilly, 2002.*

[39]
24. *HENDRICKS, M., GALBRAITH, B., IRANY, R., MILBERNY, J., MODI, T., TOST, A., TOUSSAINt, A., BASHA, S., and CABLE, S. Professional Java Web Service. Wrox Press, 2001.*

[40]
25. *LEWIS, G., BARBER S. and SIEGEL E., Programming with Java IDL,*

Developing Web Applications with Java and CORBA, Wiley Press, 1998.

[41] 26. *DEITEL, H., DEITEL P., Java - How to Program, Prentice Hall, Inc., 1998.*

[42] 27. *LINTHICUM D., Enterprise Aplication Integration, Addison Wesley, 2000.*

[43] 28. *KLEPPE A., WARMER J., and BAST W., MDA Explained: The Model Driven Architecture: Practice and Promise, Addison Wesley, 2003.*

[44] 29. *EELES P., HOUSTON K., KOZACZYNSKI W., Building J2EE Applications with the Rational Unified Process, Addison Wesley, 2003.*

[45] QUATRANI T., Visual Modeling with Rational Rose 2000 and UML, , Addison Wesley, 2000.

[46] 30. *KUEBLER D. and EIBACH W., Adapting legacy applications as Web services, TA, 2002.*

[47] 31. *OMG- Object Management Group Specification. CORBA /JAVA, 1999.*

[48] 32. *UNISYS. Web Enabler for Clearpath MCP. July 2002.*

[49] 33. *APACHE. Apache Tomcat Cocoon. 2003.*

[50] 34. *VINOSKI, Steve, CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments, Cambridge,MA, Iona Technologies,Inc., 2001.*

[51] 35. *SAHW E GARLAN, Software Architecture, Prentice Hall, ISBN0-13-182957-2*

[52] 36. *VINOSKI, Steve, Integration with Web Services, IEEE Internet Computing, 2003, IONA Technologies, Inc.*

[53] *37. OMG, MDA Guide 1.0.1., 2003.*