MARCELO DOMINGOS

Uma Arquitetura de Referência para Sistemas de Informação e Portais de Serviços de Governo Eletrônico

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal de Santa Catarina como requisito parcial para a obtenção do grau de Mestre em Engenharia de Produção.

Orientador: Prof. Roberto Carlos dos Santos Pacheco, Dr.

FLORIANÓPOLIS
2004

MARCELO DOMINGOS

Uma Arquitetura de Referência para Sistemas de Informação e Portais de Serviços de Governo Eletrônico

Esta dissertação foi julgada adequada e aprovada para a obtenção do grau de **Mestre em Engenharia de Produção** no **Programa de Pós-Graduação em Engenharia de Produção** da **Universidade Federal de Santa Catarina**.

Florianópolis, 27 de setembro de 2004.

Prof. Edson Pacheco Paladini, Dr.
Coordenador do Programa

BANCA EXAMINADORA:

Prof. Roberto Carlos dos Santos Pacheco, Dr.
Orientador

Prof. Vinícius Medina Kern, Dr.

Prof. Rogério Cid Bastos, Dr.

Prof. Fernando Borges Montenegro, M. Eng.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

A demanda por sistemas de software que tornem os serviços públicos acessíveis e transparentes aos cidadãos tem levado governos a investir na construção de Sistemas de Informação e Portais de Serviços. Nossa proposta é definir uma arquitetura de software de referência voltada aos requisitos das aplicações de governo, cujos processos de desenvolvimento estejam baseados em uma estrutura que permita acelerar a produção de tais aplicações a um custo mais baixo e com ganhos na qualidade final. A fim de validar a arquitetura, faz-se uma aplicação à Plataforma Lattes. Como resultado, é desenvolvida uma tecnologia de plug-ins que habilita a inclusão dinâmica de recursos de uma maneira contínua e descentralizada, a qual relativamente promove redução de custos e aumento da qualidade. Recursos como CV-Resume, CV-Perfil e outros foram agregados com sucesso por meio dessa arquitetura.

Palavras-Chave: governo eletrônico, arquitetura de software, sistemas de software, Plataforma Lattes, framework.

Abstract

The demand on software systems which make public services accessible and transparent to citizens has led governments to invest in building Information Systems and Service Portals. Our proposal is the definition of a reference software architecture aiming government applications requirements whose the development process is based on a structure which allows speed up the production of those applications with cost reduction and quality improvement. An application to the Lattes Platform is effected in order to validate the architecture. As a result, it is created a plug-in technology which enable dynamic including of resources to the Platform in a continued and decentralized way, which relatively promote costs reduction and quality improvement. Resources like CV-Resume, CV-Profile and others were aggregated with success by this architecture.

Keywords: electronic government, software architecture, software systems, Lattes Platform, framework.

Abreviaturas

ADL	Linguagem de Descrição Arquitetural
API	Interface de Programa de Aplicação
ASI	Arquitetura de Sistemas de Informação
CASE	Engenharia de Software Auxiliada por Computador
CBD	Desenvolvimento Baseado em Componentes
CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico
CRM	Gerenciamento do Relacionamento com o Cliente
CV-Lattes	Sistema de Currículo Lattes
DAO	Objeto de Acesso Direto
DLL	Ligação de Bibliotecas Dinâmicas
E-GOV	Governo Eletrônico
G2B	Governo para Empresas
G2C	Governo para Cidadãos
G2E	Governo para Empregados
G2G	Governo para Governo
GCJ	Compilador GNU para Java
GNU	Ferramentas e Linguagens Compatíveis com o UNIX
GoF	Gangue de Quatro
GP-Lattes	Diretório de Grupos de Pesquisa no Brasil Lattes
GUI	Interface Gráfica do Usuário
HTML	Linguagem de Marcação de Hipertexto
HTTP	Protocolo de Transferência de Hipertextos
ICONIX	Processo de Desenvolvimento de Software da ICONIX
ID	Identificação
ISO	Organização Internacional para Padronização
JIT	Compilação de Tempo de Execução
LILACS	Literatura Latino-Americana e do Caribe em Ciências da Saúde
MDA	Sistemas Dirigidos a Modelo
MedLine	Literatura Internacional em Ciências da Saúde
MFC	Fundamentação de Classes da Microsoft
MV	Máquina Virtual
MVC	Modelo, Visão e Controle
ODP	Processo Distribuído Aberto
OO	Orientado a Objetos
OSI	Interconexão de Sistemas Abertos
PAC	Controle, Abstração e Apresentação
PDF	Formato de Documento Portável
RIA	Aplicações de Internet Ricas
RPC	Chamada de Procedimentos Remotos
RUP	Processo Unificado da Rational
SciELO	Biblioteca Eletrônica Científica On-line
SGBD	Sistema Gerenciador de Banco de Dados
SI	Sistemas de Informação
SO	Sistemas Operacionais
SOAP	Protocolo de Acesso a Objetos Simples
SWING	Fundação de Classes Java
SWT	Toolkit Padrão de Widgets

TCI	Tecnologias de Comunicação e Informação
UFSC	Universidade Federal de Santa Catarina
UML	Linguagem de Modelagem Unificada
UNISINOS	Universidade do Vale do Rio dos Sinos
URL	Localização Uniforme de Recurso
VCL	Biblioteca de Componentes Visuais
WWW	Vasta Teia Mundial
XHTML	Linguagem de Marcação de Hipertexto Extensível
XML	Linguagem de Marcação Extensível
XSL	Linguagem de Estilo Extensível
XUL	Linguagem de Interface do Usuário em XML

1 Introdução

1.1 Apresentação

Em todo o mundo, governos estão disponibilizando informações importantes on-line, automatizando processos burocráticos e interagindo eletronicamente com seus cidadãos (PCIP, 2002). Tal direcionamento vem da necessidade que cada governo possui de se expressar de forma rápida e transparente diante da sociedade e de obter informações cruciais para a boa gestão dos recursos públicos.

As tecnologias de comunicação e informação (TCI) se tornaram o ferramental necessário e viabilizador para a existência dos governos eletrônicos (E-Gov). A última década apresentou uma considerável evolução nessa área e criou um ambiente capaz de oferecer os mecanismos necessários para a mudança de paradigma na administração pública.

Embora o E-Gov não seja um atalho para o desenvolvimento econômico nem para a diminuição de orçamentos (PCIP, 2002), a infra-estrutura criada por este paradigma aponta um caminho baseado em informações, que se bem gerenciadas, podem viabilizar o acesso a esses objetivos. Por essas e outras razões, muitos governos estão empenhados em automatizar todo o processo burocrático institucional existente (PCIP, 2002).

Do ponto de vista da informática, essa automatização, em última instância, irá transformar todas as decisões governamentais no sentido de implantar E-Gov em peças de software. Isso pode ser observado quando acessamos sites Web, formulários eletrônicos, bases dados, data warehouse e diversas outras representações da TCI que fazem o papel de coletar, processar, informar e gerar conhecimento tanto para o governo quanto para o cidadão.

A principal fonte de informações para E-Gov são provenientes de aplicações de Sistemas de Informação (SI) e Portais de Serviços, as quais se encontram disponíveis na Web ou off-line, que, normalmente, podem ser obtidas diretamente pela Internet ou por disquete nos órgãos competentes. Infelizmente, por razões gerenciais e técnicas, o processo de desenvolvimento desses mecanismos de software tem sido em sua maioria falho. Isso se dá principalmente em virtude de redundâncias de desenvolvimento, “engessamento tecnológico” e falta de padronizações, o que impede que essas aplicações alcancem os atributos de qualidade necessários para a boa gerência dos recursos públicos e a satisfação dos cidadãos. Embora existam equipes competentes que desenvolvem boas soluções, a falta de uma

arquitetura de software voltada aos requisitos desse domínio de aplicações tem sido a principal causa dos problemas identificados.

Todos esses problemas poderiam ser evitados se uma arquitetura de software de referência fosse especificamente elaborada para atender às necessidades desse domínio de aplicação. Essa arquitetura poderia prever os principais problemas que ocorrem na construção das aplicações e, através de suas estruturas, possibilitar que diversos requisitos não-funcionais que promovem a redução de custos e o aumento de qualidade fossem alcançados por essas aplicações. Os requisitos não-funcionais com os quais estamos especialmente interessados são aqueles que simplificam o desenvolvimento, promovem a redução de custos, facilitam o gerenciamento das versões, aumentam a robustez e viabilizam a adequação tecnológica das aplicações ao longo do tempo. Aplicações que satisfazem a esses requisitos atendem às necessidades que os governos vêm sentindo no desenvolvimento de sistemas de software.

1.2 Objetivo geral

O objetivo deste trabalho é conceber e especificar uma arquitetura para sistemas de informação e portais de serviços governamentais que respeitem a arquitetura integrada de governo eletrônico, proposta por Pacheco (2003), e que sejam voltados à captura de informações junto a cidadãos, visando aumentar a qualidade das aplicações e reduzir os custos no desenvolvimento dessas aplicações.

1.3 Objetivos específicos

De um modo específico, pretende-se:

1. apresentar governo eletrônico e evidenciar suas necessidades junto à sociedade do conhecimento;
2. demonstrar a existência da necessidade de padronizações arquiteturais tanto no âmbito gerencial quanto no âmbito das arquiteturas de software utilizadas pelos sistemas de informação construídos para governo;
3. investigar a disciplina Arquitetura de Software e organizar as informações obtidas de maneira a subsidiar a elaboração de uma arquitetura voltada aos requisitos de governo eletrônico;

4. propor uma arquitetura de software de referência que permita o crescimento modular dos sistemas e que seja adequada às necessidades dos sistemas de informação governamentais;
5. aplicar a arquitetura de referência no contexto dos sistemas de captura off-line da Plataforma Lattes do CNPq a fim demonstrar como resultados esperados podem ser alcançados.

1.4 Justificativa

O desenvolvimento de software tem sido uma atividade um tanto quanto heurística e, por isso, muito dependente da qualidade da equipe de desenvolvimento. Boas heurísticas são difíceis de serem elaboradas, e como conseqüência as aplicações ficam prejudicadas quando essas boas heurísticas não são alcançadas.

A atitude mais comum no desenvolvimento de uma nova aplicação tem sido a escolha de um framework de desenvolvimento disponível no mercado e de conhecimento da equipe que irá implementá-lo. Nesse ponto é que começam a aparecer os problemas, como forte acoplamento tecnológico, desperdício de esforços, dificuldade de gerenciamento de versão, uso exagerado e desnecessário de recursos computacionais, problemas estruturais, aumento do custo global, dificuldade de adequação a novas tecnologias, latência na geração de novas versões, etc. Todos esses problemas, naturais do processo de desenvolvimento de software, emperram a fabricação de software nas mais diversas áreas de aplicação.

A disciplina Engenharia de Software tem se dedicado ao estudo de soluções para esses problemas e tem criado métodos, arquiteturas, padrões e uma série de ferramentais lógicos para esse fim. No escopo desta dissertação estamos interessados nas soluções de arquitetura que já foram levantadas pela disciplina e que servem de base para a construção das principais aplicações de software da atualidade.

Entendemos que a arquitetura dos sistemas de software é a peça fundamental para dar um bom andamento a todo o processo de produção de software (ALBIN, 2003). Quando uma arquitetura está bem definida, a construção de aplicações de software fica simples e direta, pois as soluções para os problemas já foram previamente pensadas. Uma arquitetura bem definida estabelece modelos abstratos e precisos que permitem projetar, implementar e manter sistemas de software, avaliando e garantido suas qualidades.

Os sistemas de software em governo eletrônico não fogem à regra de qualquer sistema de software existente. Pelo contrário, essas aplicações, por possuírem uma quantidade de usuários não determinada, possuem requisitos que normalmente não precisam ser tratados pela maioria das aplicações. Por isso, esses requisitos necessitam ser levantados e compreendidos a fim de se elaborar uma arquitetura que seja adequada. Para isso, a arquitetura deve ser empregada em alguma aplicação de governo a fim de se comprovar sua validade.

1.5 Metodologia

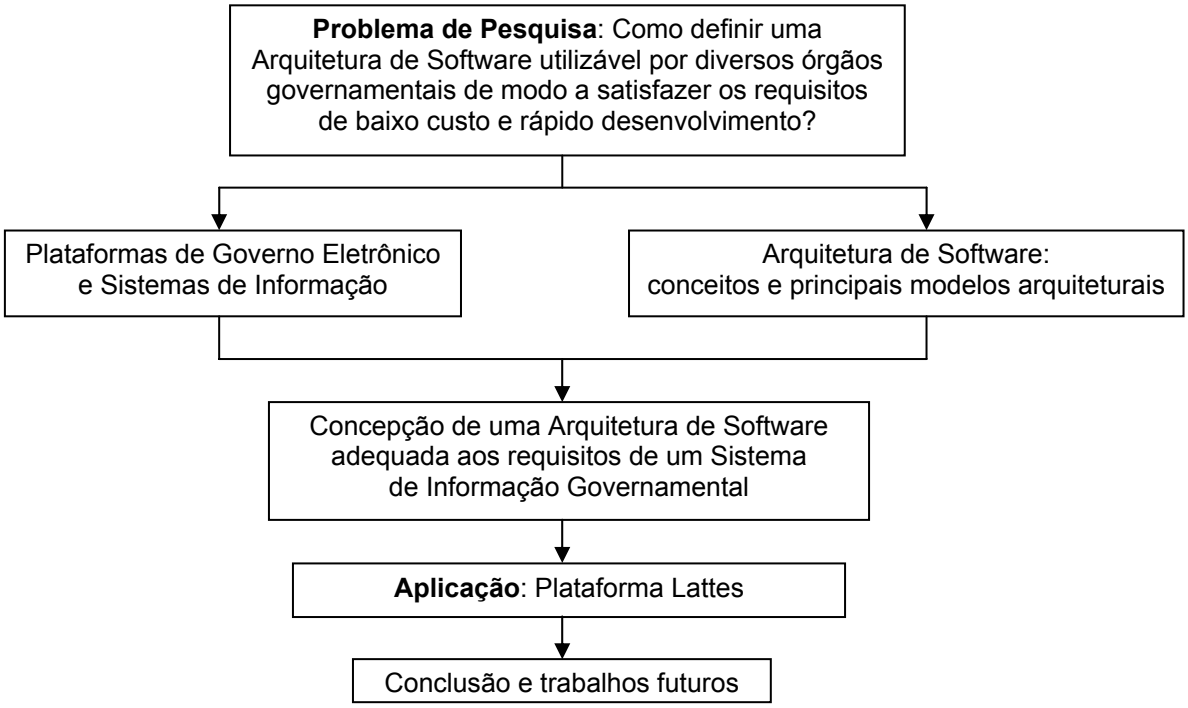

Figura 1.1 - Metodologia

A Figura 1.1 apresenta uma visão esquemática da metodologia adotada no presente trabalho. Para tratar o problema de pesquisa da dissertação, adotou-se a metodologia fundamentada nas etapas a seguir.

1. Estudo sobre plataformas de governo eletrônico e soluções arquiteturais adotadas nos principais sistemas de informação distribuídos no País, além dos principais conceitos de governo eletrônico e como os sistemas de informação implementam esses conceitos. Análise dos aspectos arquiteturais e das soluções tecnológicas adotadas a fim de identificar os pontos fortes e fracos em cada arquitetura.

2. Embasamento teórico da disciplina Arquitetura de Software. Estudo dos principais padrões arquiteturais e de organização do conhecimento em função das necessidades das aplicações de governo eletrônico.
3. Concepção e definição de uma Arquitetura de Software adequada aos requisitos propostos. Proposição de tecnologias e mecanismos que permitam a integração de diversos sistemas governamentais tanto no âmbito do acesso do cidadão quanto no âmbito do órgão governamental.
4. Aplicação da arquitetura proposta na Plataforma Lattes do CNPq. O objetivo é validar a arquitetura e demonstrar os ganhos que podem ser obtidos com a adoção dessa arquitetura.

A Figura 1.1 ilustra, ainda, a conclusão e os futuros desenvolvimentos, descritos ao final do trabalho, como decorrentes da aplicação das quatro etapas metodológicas descritas.

1.6 Delimitações

Para a elaboração da arquitetura de software proposta por esta dissertação, foi necessário estabelecer os elementos delimitadores descritos na seqüência.

- A arquitetura proposta objetiva atender aos requisitos de projetos em sistemas de informação que possam ser representados em aplicações de software tanto para a o ambiente Web quanto para o ambiente Desktop off-line. A uso da arquitetura fora desses contextos pode exigir adaptações que vão além da nossa proposta.
- As linguagens de especificação de arquiteturas de software (ADLs) estão ainda em estágios embrionários e, por isso, não foram utilizadas na especificação da arquitetura.
- A aplicação da arquitetura de referência proposta foi limitada aos sistemas de captura para desktops off-line da Plataforma Lattes, uma vez que se entendeu ser esta aplicação o suficiente para a validação.

1.7 Estrutura do trabalho

A dissertação está estruturada em seis capítulos e três apêndices, incluindo-se o presente capítulo de introdução.

No segundo capítulo apresenta-se a introdução sobre plataformas de governo eletrônico, sistemas de informação e plataformas de governo. O objetivo é apresentar o contexto da aplicação da arquitetura de software proposta na dissertação, concebida a partir da análise dos requisitos que possuem os sistemas de informação para governo eletrônico. Também neste capítulo é apresentada uma extensão ao modelo bidimensional da arquitetura piramidal de plataformas de governo proposta na literatura para um modelo tridimensional, de modo a situar na arquitetura o escopo de atuação da proposta.

No Capítulo 3, é apresentada a fundamentação teórica sobre arquiteturas de software. Objetiva-se investigar as soluções arquiteturais para os problemas das aplicações e estruturar as informações fornecidas na literatura em função das necessidades dos sistemas de software de E-Gov. Discutem-se ainda a definição conceitual de Arquitetura de Software, suas divisões, os processos de desenvolvimento voltados à arquitetura de software, os padrões de projetos e outros temas relevantes para a construção de uma arquitetura de governo eletrônico.

No Capítulo 4, com base nos estudos dos Capítulos 2 e 3, detalha-se a proposta de arquitetura de software. Para isso, são investigados os problemas existentes nas atuais aplicações de governo e as melhores soluções em função dos requisitos levantados nos capítulos anteriores. Dessa forma, são elaborados uma arquitetura de software que define todos os elementos estruturais necessários para os sistemas de software em governo eletrônico e seus comportamentos. Ao final do capítulo, é exibido como essa arquitetura pode ser aplicada nos sistemas de governo já existentes.

No Capítulo 5, mostra-se uma aplicação dos conceitos da arquitetura definida no Capítulo 4 no contexto das aplicações de coleta de dados off-line da Plataforma Lattes de TCI. É especificado um ambiente de desenvolvimento de módulos dinâmicos para esses sistemas, e são também apresentados os resultados obtidos.

No Capítulo 6 apresentam-se as considerações finais e as sugestões para trabalhos futuros.

O trabalho inclui ainda os seguintes apêndices: (a) apêndice A é uma especificação da arquitetura InterLattes atualmente disponível na Plataforma Lattes do CNPq; (b) apêndice B apresenta um manual de como módulos InterLattes devem ser construídos; e (c) apêndice C apresenta o código fontes das units de conexão ao InterLattes. Esses três apêndices permitem que o leitor possa ver na prática os resultados obtidos da aplicação da arquitetura definida no Capítulo quatro 4.

2 Governo Eletrônico e Tecnologia da Informação

As aplicações de Governo Eletrônico possuem alguns requisitos particulares, mas, ao mesmo tempo, compartilham muitos dos requisitos e das soluções das aplicações do setor privado. Assim, é primordial que se estude o contexto em que essas aplicações se encontram inseridas a fim de se evidenciarem os requisitos que devem ser atendidos por uma proposta de arquitetura de software voltada a esse domínio de aplicações. Além disso, neste capítulo, serão contextualizadas, em relação ao modelo de arquitetura de plataformas de governo proposto por Pacheco (2003), as camadas da arquitetura que serão atendidas pela proposta desta dissertação.

2.1 O que é Governo Eletrônico

Seguindo a tendência de adoção da Web pelas aplicações comerciais, as organizações governamentais estão tentando tirar vantagens dessa forma de interação e comunicação (ELSAS, 2001). A definição do que é Governo Eletrônico, portanto, é de difícil detalhamento por envolver conceitos ainda recentes e abrangentes.

De acordo com o conceito de Zweers e Planqué (apud JOIA, 2002, p. 92), pode-se dizer que:

> Governo Eletrônico é um conceito emergente que objetiva fornecer ou tornar disponível informações, serviços ou produtos, através de meio eletrônico, a partir ou através de órgãos públicos, a qualquer momento, local e cidadão, de modo a agregar valor a todos os stakeholders envolvidos com a esfera pública.

Da forma semelhante, Holdes (2001, p. 3) define Governo Eletrônico como sendo:

> O uso da tecnologia da informação, em particular a Internet, para oferecer serviços públicos de maneira muito mais conveniente, orientada ao cidadão e financeiramente viáveis.

Já em Balutis (1999, p. 2) se encontra definido Governo Eletrônico através da seguinte equação:

> Governo Eletrônico = Comércio Eletrônico + Customer Relationship Management (CRM) + Supply Chain Management (SCM) + Gestão do Conhecimento + Business Intelligence (BI) + Tecnologias Colaborativas

Fernandes (2001) diz que o E-Gov envolve basicamente quatro tipos de transações, que ocorrem não apenas por meio da Internet mas também via telefonia móvel, televisão digital, call centers e outros tipos de aplicações ligadas aos computadores pessoais:

1. G2G (Government to Government): a relação entre governo e outros órgãos do governo quando se trata de uma relação intra ou intergovernos;
2. G2B (Government to Business): a relação entre governo e setor privado é caracterizada por transações entre governos e fornecedores;
3. G2C (Government to Constituent - Governo e Cidadão): envolve relações entre governos e cidadãos; e
4. G2E (Government to Employee): a relação governo e o servidor público é caracterizada pelo relacionamento com os servidores públicos.

Em linhas gerais, as funções características do E-Gov são (apud FERNANDES, 2001):

a) prestação eletrônica de informações e serviços;
b) regulamentação das redes de informação, envolvendo principalmente governança, certificação e tributação;
c) prestação de contas públicas, transparência e monitoramento da execução orçamentária;
d) ensino a distância, alfabetização digital e manutenção de bibliotecas virtuais;
e) difusão cultural com ênfase em identidades locais, fomento e preservação das culturas locais;
f) E-procurement, isto é, aquisição de bens e serviços por meio da Internet, tais como licitações públicas eletrônicas, pregões eletrônicos, cartões de compras governamentais, bolsas de compras públicas virtuais e outros tipos de mercados digitais para os bens adquiridos pelo governo; e
g) estímulo aos e-negócios, através da criação de ambientes de transações seguras, especialmente para pequenas e médias empresas.

Segundo Lenk e Traunmüllerv (apud JOIA, 2002), o Governo Eletrônico pode ser analisado sob quatro perspectivas, descritas a seguir.

1. **Perspectiva do Cidadão** - visa oferecer serviços de utilidade pública ao cidadão contribuinte.

2. **Perspectiva de Processos** - visa repensar o *modus operandi* dos processos produtivos ora existentes no governo em suas várias esferas, tais como, por exemplo, os processos de licitação para compras (e-procurement).

3. **Perspectiva da Cooperação** - visa integrar os vários órgãos governamentais e estes com outras organizações privadas e não-governamentais, de modo que o processo decisório possa ser agilizado, sem perda de qualidade, e que se evitem fragmentação, redundâncias, etc., hoje existentes nas relações entre esses vários atores.

4. **Perspectiva da Gestão do Conhecimento** - visa permitir ao governo, em suas várias esferas, criar, gerenciar e disponibilizar em repositórios adequados o conhecimento tanto gerado quanto acumulado por seus vários órgãos.

Essas quatro facetas podem ser mais bem visualizadas na ilustração abaixo.

Figura 2.1 - Perspectivas em Governo Eletrônico
Fonte: (JÓIA, 2002)

2.1.1 As perspectivas do Governo Eletrônico

Adicionalmente, Governo Eletrônico também pode ser entendido através de outras taxonomias, como as expressas por Perri (2001) a seguir.

1. **Fornecimento de serviços eletrônicos**: atualmente, a maior parte dos esforços, recursos e atenção política devotados a Governo Eletrônico se concentram nessa área, que envolve o fornecimento de serviços de utilidade pública para o contribuinte assim como o relacionamento governo-empresas, usando as tecnologias da informação e comunicação como propiciadoras para tal.

2. **Democracia Eletrônica (e-democracy)**: novas legislaturas, como as da Escócia e do País de Gales, estão usando sistemas de votação eletrônica nos seus parlamentos

locais, assim como há no mundo experiências piloto de consulta on-line aos cidadãos. O Brasil vem usando o e-voting – não sem controvérsias acerca da segurança da votação – já há algum tempo, o que o enquadraria nessa tipologia.

3. **E-governance**: segundo Kraemer e Dedrickvii, essa é a área menos estudada de Governo Eletrônico. Ela incluiria, entre outras atividades, todo o suporte digital para elaboração de políticas públicas, tomadas de decisão, *public choices* e workgroup, entre os vários gestores públicos de diferentes escalões.

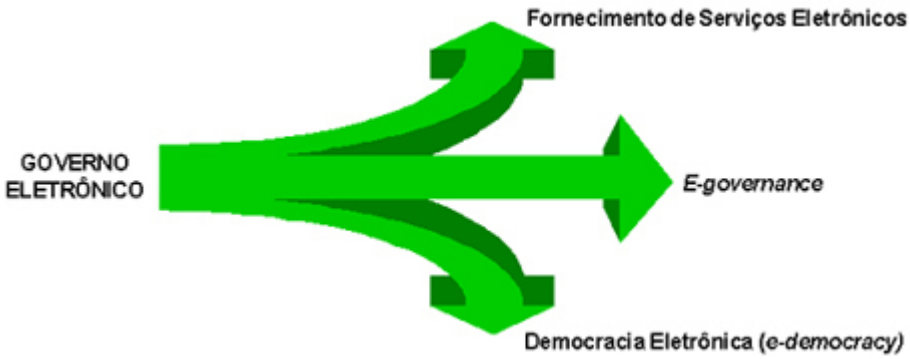

Figura 2.2 - Perspectivas de Governo Eletrônico
Fonte: (JOIA, 2002)

2.1.2 As quatro fases do Governo Eletrônico

No processo de concepção do que é E-Gov, quatro fases foram identificadas e padronizadas: (1) Presença, (2) Interação, (3) Transação e (4) Transformação (OAKLAND, 2003; KELLER, 2000). A transição de uma fase para outra é influenciada por uma série de fatores que podem ser agrupados em quatro áreas: estratégias e políticas, pessoas, processos e tecnologia.

1. **Presença**: a primeira fase caracteriza-se pelo estabelecimento da presença do governo na Internet. Durante esta fase, os sites na Internet são estáticos em sua natureza e somente oferecem informações de propósito geral. Nesse período, ocorre uma proliferação de websites departamentais, cada um deles referenciado por uma URL diferente, oferecendo informações e serviços produzidos por aquele determinado departamento, tais como endereço e telefone para contato, missão da organização, lista de serviços prestados etc., numa espécie de marketing eletrônico. Para ter acesso àquelas páginas, os usuários da Internet devem conhecer os endereços eletrônicos dessas páginas ou fazer uso de sites de busca. Considerando-se que, em média, um governo tem entre 50 a 70 diferentes agências ou

departamentos, pode-se ter uma idéia da dificuldade dos cidadãos para interagirem eletronicamente com o governo.

2. **Interação**: esta segunda fase é caracterizada pelos sites de Internet que fornecem capacidades de pesquisa, formulários para download e links para outros sites relevantes. Em muitos aspectos, este estágio habilita o público a acessar informações críticas on-line, mas requer uma visita ao órgão governamental para completar a tarefa. Nesta fase, percebe-se já um movimento do governo de colocar alguma ordem no caos que se formou na Internet durante a primeira fase. Notando a dificuldade dos cidadãos em lidar com um universo tão fragmentado, os governos criam um site central, a partir do qual os cidadãos podem se conectar a qualquer departamento. O modelo de páginas governamentais ainda é voltado à estrutura formal, e os links para outros departamentos de governo são referenciados de acordo com o organograma. Infelizmente, essa abordagem tem o mesmo efeito que a anterior, uma vez que obriga o cidadão a ter conhecimento prévio de que departamento deve reportar-se para ter um certo tipo de serviço prestado.
3. **Transação**: esta terceira fase é caracterizada por oferecer ao público a possibilidade de ele completar toda uma tarefa on-line. O foco dessa fase está na construção de aplicações self-service para o acesso on-line do público. Aqui há inovações significativas. Ou seja, em vez de organizar suas páginas por departamento, os serviços são agregados num único portal, que, para o governo, é geralmente o principal site governamental na Web.
4. **Transformação**: a quarta fase é caracterizada pela redefinição da disponibilidade dos serviços e das informações governamentais. Esta fase está baseada em robustas ferramentas de gerenciamento do relacionamento com o cliente (CRM), no acesso por dispositivos sem fio e em novos métodos de serviços alternativos que forneçam capacidades de remodelar o relacionamento com o cidadão, as empresas, os empregados e o governo. Essas tecnologias permitirão ao governo transformar os quatro tipos de relacionamentos que são vitais para seu trabalho: G2C, G2B, G2G, e G2E.

2.2 Arquitetura de Sistemas de Informação para E-Gov

Existem dois conceitos do termo “arquitetura” que são complementares, mas que precisam ser conceituados separadamente, a fim de se definir o escopo de abrangência da proposta

arquitetural contida nesta dissertação, a saber: (1) Arquitetura de Sistemas de Informação (ASI) e (2) Arquitetura de Software (AS).

2.2.1 O que é Arquitetura de Sistemas de Informação

O termo "arquitetura" nem sempre foi associado a software. Essa associação passou a ser feita no final dos anos 80 por Zacman (1987) e Richardson et al. (1990). A evolução do uso do termo pode ser vista na tabela abaixo.

Período	**Significado**
Até os anos 80	• Arquitetura tradicional associada com projetos de hardware.
1987	• Arquitetura passa a ser associada com a área de software.
Anos 90	• Arquitetura expressa SI. • Arquitetura de SI (processadores, programas de aplicação de dados x comunicação, gerenciamento de dados). • Arquitetura associada à estratégia de negócios.
Final dos anos 90	• Arquitetura de SI como arquitetura de informação da empresa. • Arquitetura de Software.
Até 2003	• Padrões de Projeto. • Padrão unificado de especificação de modelagem UML. • Desenvolvimento baseado em componentes (CBD).

Tabela 2.1 - Evolução do uso do termo Arquitetura

Fonte: (TAIT, 1999). A linha da tabela que se encontra em cor diferente foi complementada.

O uso do termo não foi acompanhado de um consenso de seu significado, passando a existir diversas definições. Todas essas definições se encaixavam em três visões básicas: (1) arquitetura de dados; (2) arquitetura voltada para os negócios; e (3) arquitetura de infra-estrutura tecnológica (TAIT, 1999).

Ainda segundo Tait (1999), a Arquitetura de Sistemas de Informação possibilita como contribuições básicas: aprimorar as atividades do planejamento estratégico de sistemas de informação; melhorar o desenvolvimento de sistemas de informação computadorizados;

racionalizar a execução das atividades; economizar tempo; estabelecer ordem e controle no investimento de recursos de SI; definir e inter-relacionar dados; fornecer clareza para a comunicação entre os membros da organização; melhorar e integrar ferramentas e metodologias de desenvolvimento de software; estabelecer credibilidade e confiança no investimento de recursos do sistema; e fornecer condições para aumentar a vantagem competitiva.

Portanto, a Arquitetura de Sistemas de Informação preocupa-se em gerir todo o processo da geração de Sistemas de Informação tendo por alvo a otimização e a organização de todos os passos necessários.

2.2.2 Arquitetura de Software X Arquitetura de Sistemas de Informação

Uma arquitetura é uma descrição das estruturas de sistema – decomposição modular, processos, implantação, camadas, etc. – e serve como um veículo para comunicação. É a manifestação das decisões de projetos mais recentes e é também a abstração reusável que pode ser transferida para novos sistemas.

Bass (2003, tópico 2.1) define o termo "arquitetura" da seguinte forma:

> A arquitetura de software de um programa ou sistema de computação é a estrutura ou estruturas do sistema, as quais compreendem elementos de software, as propriedades visíveis externamente desses elementos e o relacionamento entre eles.

Já a Arquitetura de Sistemas de Informação tem se focado em um aspecto mais abrangente e procura estabelecer um conjunto de elementos cuja finalidade é proporcionar um mapeamento da organização no tocante aos elementos envolvidos com o processo de desenvolvimento e implantação de SI (TAIT, 1999), distinguindo-se dessa forma da AS no que se refere ao escopo de abrangência dos objetos de estudo.

2.2.3 Arquitetura de Software e Governo Eletrônico

O E-Gov necessita de uma infra-estrutura capaz de atender às suas necessidades no sentido de se expressar eletronicamente. A massificação do uso da Internet tem criado um ambiente adequado para se praticar E-Gov. Porém, construir aplicações para atender às demandas desse domínio de aplicação exige mais do que simplesmente construir peças de software.

Se for analisada a maneira como as aplicações de E-Gov estão sendo ou foram desenvolvidas, observar-se-á a falta de uma arquitetura global, o que prejudica consideravelmente o desenvolvimento dessas aplicações. Essa arquitetura deveria ser capaz de

abstrair os sistemas dos problemas tecnológicos sem amarrar as respectivas soluções a um momento da tecnologia.

A disciplina Arquitetura de Software apresenta metodologias genéricas e consolidadas que apontam o caminho para a estruturação das aplicações de modo a permitir que evoluam ao longo do tempo e sejam desenvolvidas de uma forma mais racional. Assim, a integração dos conceitos de AS com E-Gov é de fundamental importância no processo de definição de uma arquitetura para atender os requisitos desse domínio.

2.2.4 Longevidade dos sistemas em função da arquitetura escolhida

Embora a demanda por acesso eletrônico e informações públicas tenha feito com que os organismos do governo façam investimentos pesados na construção de projetos E-Gov, isso não tem resolvido de maneira adequada as necessidades desse domínio de aplicação. PCIP (2002) diz que especialmente em países em desenvolvimento, onde os recursos são escassos, seguir adiante com planos mal concebidos pode ser um erro caro, financeira e politicamente. Assim, não basta apenas dispor de recursos. É de fundamental importância criar mecanismos que garantam a concepção correta das aplicações de E-Gov.

A tecnologia é mutante por natureza, e a cada nova concepção tecnológica, surge uma série de soluções oferecidas pela comunidade empresarial e científica. As aplicações que são construídas sobre uma dessas soluções específicas acabam por ficar presas a ela e, num curto espaço de tempo, necessitam ser redesenvolvidas. Os maiores desafios para os organismos de governos têm sido a longevidade dos projetos e a adequação de metodologia e da tecnologia em melhor benefício da comunidade usuária (PACHECO, 2003).

Vemos então que dois fatores são fundamentais para a longevidade de um projeto: (1) boa concepção dos sistemas e (2) soluções que permitam a evolução ao longo do tempo.

2.3 Desenvolvimento de plataformas para sistemas de Governo

A meta de qualquer sistema de governo é alcançar a fase de transformação definida para Governo Eletrônico, tarefa esta que não é fácil. O uso de uma metodologia integrada é fundamental para que essa fase seja naturalmente alcançada pelos projetos de Governo Eletrônico.

Nos últimos cinco anos, o Brasil desenvolveu uma experiência na área de Governo Eletrônico que tem sido referência para governos de outros países, justamente porque entre seus princípios básicos está a arquitetura E-Gov (PACHECO, 2003), baseada na padronização

de unidades de informação e sistemas de informação flexíveis e dinâmicos aos requisitos dos diferentes atores que interagem com os sistemas eletrônicos do governo. Essa experiência consiste na Plataforma Lattes, de gestão em ciência, tecnologia e inovação, desenvolvida para o CNPq pelo Grupo Stela, da Universidade Federal de Santa Catarina.

Pacheco (2003) propõe uma arquitetura e uma metodologia, ilustradas na Figura a seguir, para o desenvolvimento de plataformas de Governo Eletrônico, tendo por objetivo a geração e divulgação de informações e de conhecimento. Essa arquitetura serviu de alicerce para toda a logística utilizada nos processo de obtenção e transformação das informações utilizadas pela Plataforma Lattes.

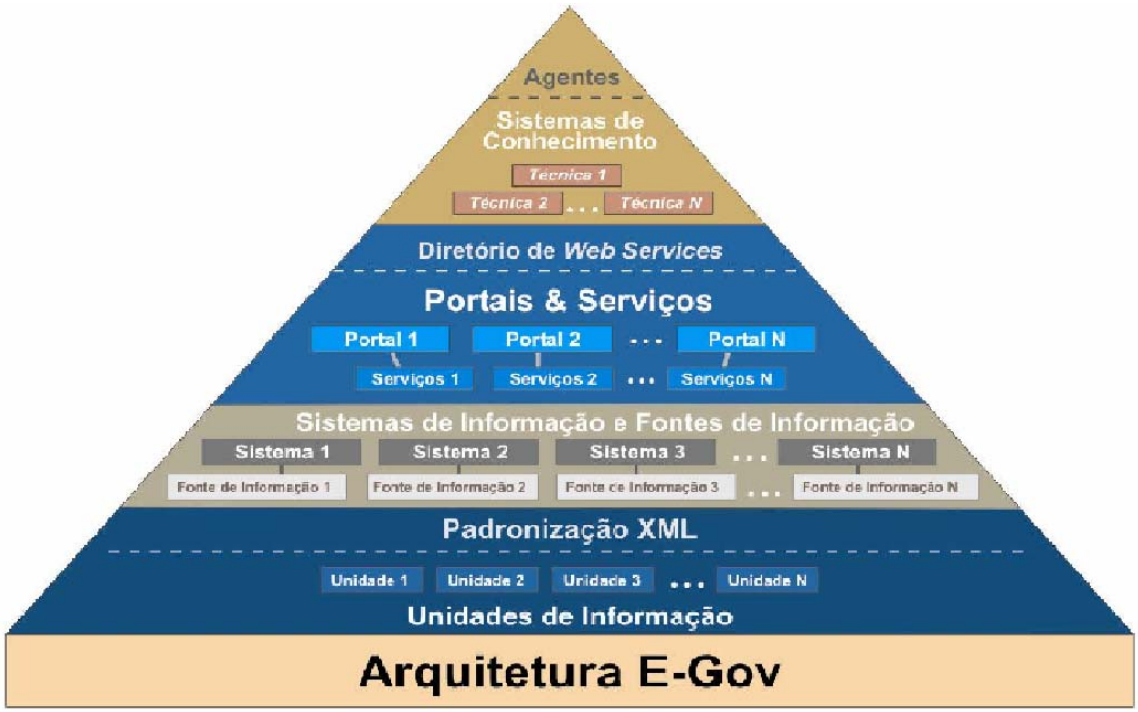

Figura 2.3 - Arquitetura conceitual para projetos E-Gov

Fonte: (PACHECO, 2003)

2.3.1 Camadas da arquitetura conceitual proposta

A arquitetura conceitual proposta tem sua representação na forma piramidal e parte de uma base composta de unidades de informação da Plataforma e segue por camadas de padronização, sistematização e publicação de informações e serviços até chegar ao topo, reservado à gestão, produção e publicação de conhecimento.

As camadas da pirâmide proposta estão assim definidas:

a) Unidades de Informação: descrevem subdomínios da área-fim para a qual a plataforma está sendo desenvolvida. São formadas por classes ou elementos do

domínio da plataforma para os quais estão associados conteúdo, processos e serviços específicos. Uma unidade de informação é composta de diversos elementos específicos (módulos e campos) e tem relação direta com outras unidades de informação do domínio da plataforma.

b) Padronização XML: metadados específicos, definidos em linguagem de marcação para cada unidade de informação que estabelece o domínio de uma aplicação E-Gov.

c) Fontes e Sistemas de Informação: São os repositórios de cada unidade de informação da plataforma e os respectivos sistemas de informação que captam, tratam e armazenam os dados da unidade junto à comunidade usuária.

d) Portais e Serviços: composta dos instrumentos desenvolvidos para apresentação de informações na Web (websites), para publicação de informações dinamicamente atualizadas que interaja com a comunidade usuária (portais Web) e dos recursos de disseminação de serviços de informação de governo na Web (Web Services).

e) Sistemas de Conhecimento: instrumentos projetados para gerar novos conhecimentos a partir das fontes de informação da plataforma e de sua operação por parte da comunidade usuária. A extração desse conhecimento se dá através da Engenharia do Conhecimento e de suas proposições metodológicas.

A arquitetura conceitual de aplicações E-Gov proposta por Pacheco (2003) é exatamente o que entendemos ser o caminho que deve ser seguido para se criar um ambiente que promova a longevidade de projetos em E-Gov.

2.3.2 Extensão do modelo proposto

A proposta de Pacheco (2003) abrange o escopo de organização de projeto e está situada em Arquitetura de Sistemas de Informação. Ao acrescentar a visão de Arquitetura de Software, estaremos estendendo o modelo arquitetural proposto para uma pirâmide tridimensional.

A Figura 2.4 contextualiza a presente dissertação em relação ao modelo arquitetural conceitual proposto e ajuda a entender o que se pretende desenvolver com este trabalho. Ao se propor mais uma dimensão, deseja-se ressaltar que não apenas a estrutura de projetos para E-Gov necessita de padronizações, mas também a forma como as aplicações são desenvolvidas.

No contexto desta dissertação, dar-se-á total atenção às camadas “Sistemas de Informação e Fontes de Informação” e “Portais e Serviços”, justamente por estarem ligadas diretamente a modelos de arquitetura de software que se refletem em aplicações que irão interagir

diretamente com o cidadão. Entende-se que as demais camadas são objetos de outros estudos, que também carecem de padronização e podem servir de temas para outras dissertações.

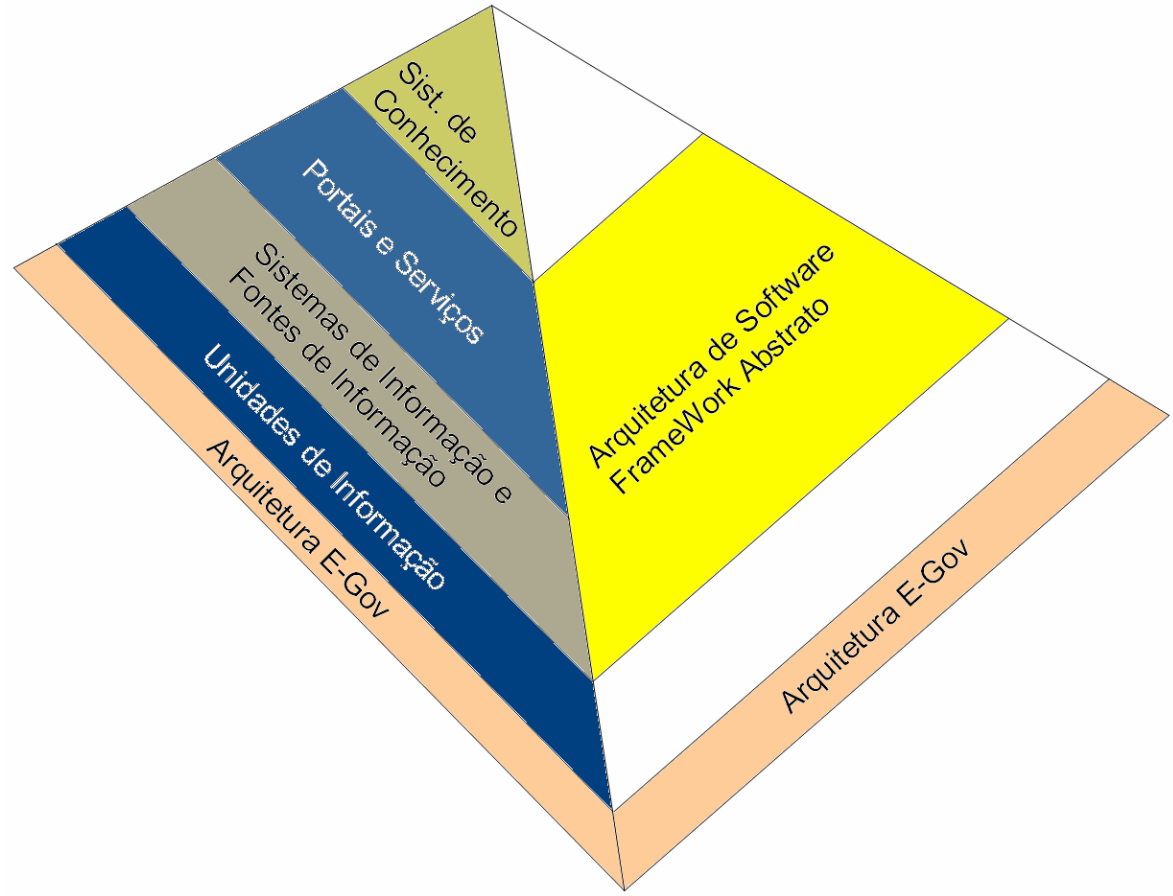

Figura 2.4 - Arquitetura conceitual para projetos E-Gov, Visão 3D

2.4 Futuro e exigências da Sociedade do Conhecimento

Cavalcanti (2000) afirma que as atividades que agregarão mais valor e que gerarão mais riqueza para os indivíduos e para a sociedade serão aquelas oriundas da inovação, que se caracteriza principalmente pela capacidade de usar o conhecimento agregado aos produtos e serviços oferecidos. O autor ainda afirma que o que importa agora para o aumento da produtividade são o trabalho intelectual e a gestão do conhecimento.

Peter Drucker (1993) diz que as atividades que ocupam o lugar central das organizações não são mais aquelas que visam produzir ou distribuir objetos, mas aquelas que produzem e distribuem informação e conhecimento.

2.4.1 Sociedade do Conhecimento

O termo Sociedade do Conhecimento é utilizado para relacionar as mudanças de paradigma na atual sociedade em virtude da agregação de conhecimento advindo do crescimento exponencial da informação organizada capaz de gerar conhecimento.

O conhecimento pode ser definido pela equação $K=(P+I)^3$, onde K significa conhecimento; P significa recursos humanos ou cérebro humano; e I significa tecnologias de informação, potencializadas pelo índice de compartilhamento – indicativo de formação de redes (SICSÚ, 2000). Essa abordagem mostra que o conhecimento tem crescimento exponencial e em função das tecnologias da informação disponíveis.

A tabela abaixo apresenta os cinco aspectos mais essenciais que caracterizam a Sociedade do Conhecimento, o que ajudará a entender a mudança de paradigma em função desse conhecimento:

Atributos	Paradigma Industrial	Paradigma do Conhecimento
• Modelo de Produção	• Economia de escala	• Flexível
• Pessoas	• Mão de obra especializada	• Polivalente e empreendedor
• Tempo	• Grandes tempos de resposta	• Tempo real
• Espaço	• Limitado e definido	• Ilimitado e indefinido
• Massa	• Tangível	• Intangível

Tabela 2.2 - As cinco características essenciais da Sociedade do Conhecimento
Fonte: Centro de Referência em Inteligência Empresarial CRIE - COPPE/UFRJ

2.4.2 O Governo Eletrônico e a Sociedade do Conhecimento

Esta nova sociedade é exigente com relação à qualidade dos sistemas de informação que fazem o interfaciamento entre a ela e o conhecimento. Não basta apenas oferecer sistemas, estes devem ter algo mais, devem ser capazes de gerar conhecimento. Alem disso, o acesso a esses recursos deve ser transparente, direto e o mais desvinculado possível de questões de natureza técnica.

Para satisfazer os requisitos dessa nova sociedade, os governos devem despender mais recursos no sentido de melhorar a concepção dos sistemas de informação e suas respectivas logísticas a fim de criar uma infra-estrutura orientada à geração de conhecimento. O tempo gasto na reformulação de aplicações, motivado por questões de natureza técnica, não será mais aceito nesse novo contexto. O processo de produção de aplicações de software deverá

ser otimizado para que o governo possa se concentrar na atividade primordial dessa nova sociedade.

2.5 Considerações finais

No presente capítulo foram discutidos os conceitos de Governo Eletrônico, as plataformas de Governo Eletrônico e as diferenças conceituais entre a Arquitetura de Sistemas de Informação e a Arquitetura de Software. Também abordou-se o que seria uma arquitetura de software voltada para E-Gov bem como quais as exigências que a Sociedade do Conhecimento tem em relação aos sistemas que fazem o interfaciamento do governo com o cidadão.

No próximo capítulo o tema Arquitetura de Software será aprofundado visando à definição de uma arquitetura específica para Governo Eletrônico que possa atender às exigências de tal domínio de aplicação. A literatura ressaltará os pontos mais relevantes para a construção de arquiteturas voltadas a frameworks.

3 Arquitetura de Software

Desenvolver software não é uma tarefa fácil. Ao longo da história do processo de desenvolvimento de software, praticamente em cada década houve uma mudança de paradigma (ALBIN, 2003). Essas mudanças se deram devido a um processo natural de sucessivo melhoramento da forma como os softwares são desenvolvidos. Albin (2003) acredita que estejamos vivendo a quinta mudança de paradigma no desenvolvimento de software, em que a arquitetura de software é um aspecto importante do ciclo de vida do desenvolvimento de software.

Levando-se em consideração o que diz Albin (2003), cremos que é de suma importância investigar a disciplina Arquitetura de Software a fim de fundamentar as bases de nossa proposta arquitetural para plataformas de governo. Ao longo deste capítulo, estaremos apresentando os tópicos mais relevantes da disciplina e ressaltando os aspectos relacionados com a nossa proposta.

3.1 O que é Arquitetura de Software

Resumidamente, a Arquitetura de Software é uma disciplina da Engenharia de Software que procura ajudar ao gerenciamento da complexidade do desenvolvimento de sistemas (ALBIN, 2003). Propõe o estudo das estruturas dos sistemas e seus relacionamentos de forma a organizar o conhecimento, objetivando alcançar os atributos de qualidade do desenvolvimento (BASS, 2003).

3.1.1 Contextualização do termo

A produção de software de qualidade que seja entregue no tempo, dentro do orçamento e que satisfaça as necessidades do usuário requer um conhecimento minucioso e elaborado, o qual a Engenharia de Software tem procurado detalhar. Através do estudo dos processos de software, do estabelecimento dos princípios de desenvolvimento, das técnicas e notações, a Engenharia de Software define como coordenar todo o processo de desenvolvimento, maximizando as chances de se produzir software de qualidade. A notação UML e metodologias como RUP e ICONIX são exemplos de produtos advindos dessa ciência.

A Engenharia de Software abrange todas as áreas do desenvolvimento de software, desde as áreas voltadas ao negócio até as questões de implementação sobre uma certa tecnologia. Por ser uma disciplina tão vasta e também por uma questão prática vem se sentindo a necessidade de se subdividir a disciplina. Albin (2003) diz que a divisão da Engenharia de

Software é provavelmente necessária, uma vez que abrange uma vasta área de disciplinas. Corroborando essa análise, vemos: estudos em arquitetura de software; metodologias que são uma disciplina por si só, como a UML e seus processos; elaboração de ontologias etc.

Na presente dissertação entende-se que a Arquitetura de Software é uma disciplina completa e autocontida, e, por isso, não serão abordadas questões relativas à Engenharia de Software que não sejam diretamente relevantes para o propósito desta dissertação.

3.1.2 Principais conceitos de Arquitetura de Software

As pesquisas em Arquitetura de Software, durante um estágio inicial, passaram por divergências com relação à definição e ao escopo de abrangência do termo. O termo "arquitetura" é um dos termos mais freqüentemente mal-utilizados (MALVEAU, 2000). Hoje, porém, há uma tendência de se definir Arquitetura de Software em termos de estruturas, elementos e conectores. Contudo, é importante apresentar os diversos conceitos que tentam classificar a disciplina, pois cada definição aborda um aspecto diferente do conceito.

Embora a Arquitetura de Software seja fundamental para a Engenharia de Software, há uma dificuldade de se definir um padrão para a definição de Arquitetura de Software que seja universalmente aceito (BASS, 2003). Albin (2003) diz que Arquitetura de Software está voltada para as questões estruturais e tecnológicas do desenvolvimento e que através de técnicas e métodos ajuda a gerenciar os problemas decorrentes da complexidade desse desenvolvimento.

Em 1994, David Garlan e Mary Shaw sugeriam em um artigo que a Arquitetura de Software era um nível de projeto preocupado com questões que iam além de algoritmos e estrutura de dados da computação. Assim, o projeto e a especificação de uma estrutura de sistema abrangente surgiam como uma nova classe de problema. No artigo os autores diziam que questões estruturais englobavam o "grosso" da organização e da estrutura de controle global e que essas questões diziam respeito a protocolos para comunicação, sincronização e acesso a dados, associação de funcionalidades para elementos de projeto, distribuição física, composição dos elementos de projeto, escalonamento e performance e seleção entre diversas alternativas de projeto.

Mas foi em 1996 que Garlan e Shaw, com a publicação do livro *Software Architecture. Perspectives on an Emerging Discipline* introduziram a Arquitetura de Software como disciplina formal (SHAW, 1996). A partir de então, esse assunto passou a ser tratado com mais formalismo, embora ainda exista uma certa dificuldade de se conceituar completamente

essa disciplina. O termo Arquitetura de Software tem sido utilizado para se referir a duas coisas: (1) estruturas de alto nível de Sistemas de Software e (2) disciplina especialista da Engenharia de Software.

A definição clássica proposta por Garlan e Shaw classifica a Arquitetura de Software em termos de componentes computacionais que se relacionam entre si em um sistema, tal como uma idéia de componentes e conectores.

Booch et al. (1999) conceituaram Arquitetura de Software como um conjunto de decisões significantes com relação aos seguintes aspectos: organização de um sistema de software; seleção dos elementos estruturais e interfaces das quais o sistema é composto, junto com os seus comportamentos, como especificado na colaboração entre aqueles elementos; composição desses elementos estruturais e comportamentais dentro de subsistemas que crescem progressivamente; e estilo da arquitetura que guia essa organização. Resumidamente, Arquitetura de Software engloba todos esses elementos e suas interfaces, suas colaborações e suas composições.

De forma semelhante, Bass et al. (2003) dizem que a arquitetura de software de um programa ou sistema de computação é a estrutura ou estruturas do sistema, as quais compreendem os elementos de software, as propriedades externamente visíveis desses elementos e o relacionamento entre eles.

Assim, ao analisar todos os conceitos apresentados, podemos perceber que a arquitetura de software está intimamente ligada aos elementos estruturais de um sistema e exerce um papel coordenador capaz de simplificar a complexidade do processo de desenvolvimento de software. Além disso, a arquitetura também exerce um papel importante nas decisões de negócio, na medida em que tende a influenciar diretamente no custo total de um projeto de software. A arquitetura de software, então, está situada entre duas partes distintas de um projeto de software – o problema do negócio e a solução técnica – e facilita a ambas o alcance de seus atributos de qualidade.

3.1.3 Arquitetura de Software como disciplina

Como disciplina, a arquitetura de software possui pelo menos umas doze escolas (MALVEAU, 2000). As maiores escolas são as descritas a seguir.

- Framework de Zachman (ZACHMAN, 1987): é um modelo de referência que compreende 30 pontos de vista da arquitetura. O modelo de referência é uma matriz que faz a intersecção de dois paradigmas: jornalismo (quem, o que, quando, por que,

onde e como) e construção (planejador, proprietário, construtor, projetista e subcontratante). Direciona para o modelo procedural.

- Processamento Distribuído Aberto (ODP) (ISO 96): é um padrão da ISO e ITU que define um modelo de referência de cinco pontos de vista: empresa, informação, ambiente computacional, engenharia e tecnologia. Oferece total suporte para Orientação a Objetos (OO) e Desenvolvimento Baseado em Componentes (CBD).
- Análise de Domínio (ROGERS, 1997): define um processo sistemático para alcançar o reuso de software. Procura transformar os requisitos de um projeto específico em requisitos de um domínio mais geral para uma família de sistemas.
- Rational's 4+1 View Model (BOOCH, 1998): é uma técnica baseada em quatro pontos de vista: lógico, implementação, processo e implantação. O "+1" denota o uso de Use Cases para a obtenção dos requisitos da aplicação.
- Arquitetura de Software Acadêmico (BASS, 2003): é o ponto de vista acadêmico da arquitetura de software. É um esforço para se manter o foco nos elementos básicos e fundamentais da matéria. Essa abordagem evita fornecer uma padronização arquitetural procurando estabelecer a fundamentação teórica necessária para se alcançar originalidade, formalidade teórica e outras metas acadêmicas.

Malveau (2000) diz que essas escolas compartilham muitos de seus conceitos e princípios, mas suas terminologias diferem grandemente. O autor diz ainda que existe uma falta de interatividade entre essas escolas, o que faz com que nenhuma delas faça um uso significativo dos resultados das outras.

Além das arquiteturas propostas nas escolas, existem as arquiteturas direcionadas pelo interesse de um fabricante de software, tais como Sun Enterprise Java Beans e Microsoft .Net. De fato, cada fabricante parece ter uma visão arquitetural única fundamentada sobre suas linhas de produtos. As arquiteturas fornecidas pelos fabricantes apresentam um entendimento mínimo das arquiteturas de aplicação (MALVEAU, 2000).

Dessa forma, ao se detalhar a disciplina, procurou-se extrair um denominador comum proveniente das escolas de arquitetura de software, e não das arquiteturas específicas dos fabricantes de software. Há um interesse particular em arquiteturas que permitam a

adaptabilidade (para novas necessidades do negócio e-Gov) e a extensibilidade dos sistemas (para a exploração de novas tecnologias).

3.2 O processo de Arquitetura de Software

As motivações para surgimento de softwares são diversas e vão desde a disponibilização de informações até a automação de oportunidades de negócios. Seja qual forem os motivos, todos esses sistemas irão seguir uma série de etapas até que estejam prontos para uso. Um processo de software é um método para desenvolver ou produzir software (TYRRELL, 2001).

A pesquisa em processo de software lida com métodos e tecnologias estimativas, suporte e melhoria das atividades de desenvolvimento de software, definindo quem faz o que, quando e como (FUGGETTA, 2000). Em outras palavras, o processo de desenvolvimento de software é o conjunto total das atividades necessárias para transformar requisitos do usuário em software.

No processo de desenvolvimento de software existem muitas metodologias que podem orientar o desenvolvimento, umas voltadas a questões de custos, outras, a questões de prazos, e ainda outras que dependem dos requisitos do projeto. Há também uma outra forma de orientar o processo de desenvolvimento, o centrado na arquitetura de software (BOOCH, 1999).

3.2.1 Contextualização do processo em Arquitetura de Software

A Engenharia de Software define uma visão em quatro fases principais que caracterizam o processo de desenvolvimento de software: (1) análise e especificação de requisitos; (2) projeto; (3) implementação e testes; e (4) implantação e manutenção (ALBIN, 2003). Essas quatro fases aparecem em todo tipo de ciclo de vida, pois compreendem as fases envolvidas no processo de desenvolvimento. Já a Arquitetura de Software estabelece uma visão em cinco fases: (1) Pré-projeto; (2) Análise de Domínio; (3) Projeto Esquemático; (4) Desenvolvimento de Projeto; (5) Construção (ALBIN, 2003). Essas cinco fases podem ser paralelas às fases definidas no processo de software tradicional.

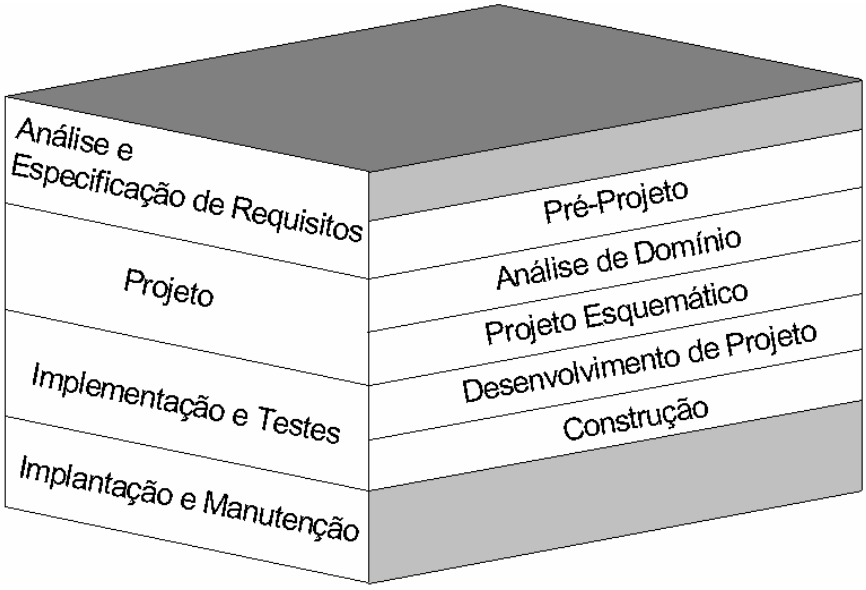

Figura 3.1 - A arquitetura no processo de desenvolvimento de sistemas

Fonte: Figura montada a partir das informações fornecidas por Albin (2003)

A figura acima contextualiza as fases do Processo de Desenvolvimento de Arquitetura em função do Processo de Desenvolvimento de Sistemas. A figura caracteriza a associação que ocorre em projetos de software com ênfase arquitetural, os quais dão início aos dois processos simultaneamente. Quando a arquitetura já se encontra consolidada – por ser um subproduto do sistema ou por ser uma um produto adquirido – o relacionamento com o processo de desenvolvimento de software torna-se diferente, como evidenciado na figura abaixo.

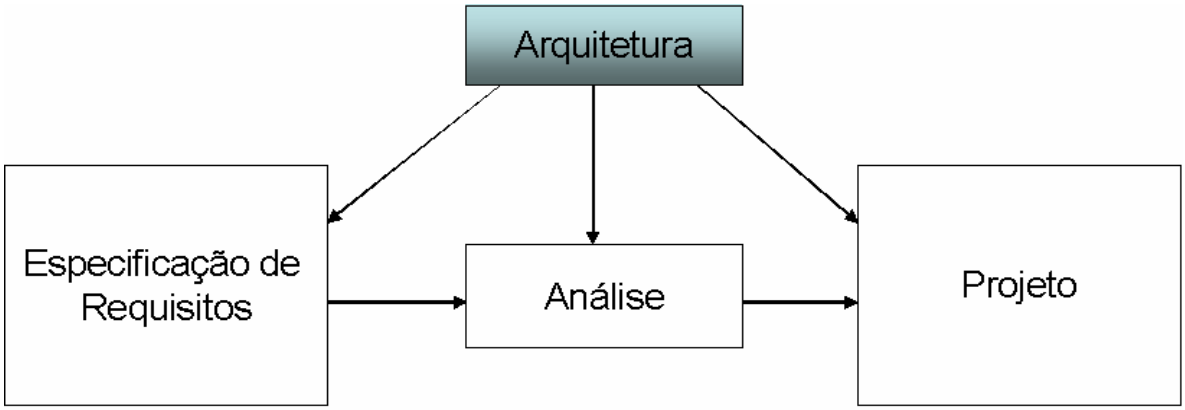

Figura 3.2 - A arquitetura no processo de desenvolvimento de sistemas

Fonte: Figura montada a partir das informações fornecidas por Albin (2003)

Apenas uma fração dos requisitos específicos do sistema tem influência na arquitetura. Por esse motivo, na situação apresentada na Figura 3.2, a arquitetura entra como um recurso de suporte às necessidades do processo de desenvolvimento. A identificação dos requisitos significantes para a arquitetura pode ser respondida através da construção de um framework

conceitual desenvolvido especialmente para um domínio específico, uma vez que essa resposta é muito dependente do domínio (JAZAYERI et al., 2000).

Assim, é notório que no processo de desenvolvimento de software com enfoque arquitetural, a arquitetura é idealizada e construída a partir das atividades complementares desempenhadas durante a construção de um software, visando estabelecer padrões arquiteturais que possam ser utilizados por outros sistemas a serem desenvolvidos pela organização. Esses padrões arquiteturais gerados são conhecidos como produto arquitetural (MALVEAU, 2000).

3.2.2 O ciclo de vida

O ciclo de vida do processo arquitetural é semelhante ao do processo de sistemas, diferenciando no alvo (ALBIN, 2003). Uma arquitetura é concebida para alcançar os atributos de qualidade para o ambiente de desenvolvimento, enquanto um sistema procura atingir os atributos de qualidade para a satisfação dos requisitos do cliente.

Um sistema pode ser desenvolvido sobre uma arquitetura previamente definida ou servir de instrumento de apoio para a definição de uma arquitetura comum aos sistemas. Essa interatividade entre os processos limita os tipos de ciclo de vida viáveis para a definição de uma arquitetura. Gacek et al. (1995) dizem que embora haja vários tipos de ciclo de vida, o ciclo em espiral (interativo incremental) é o que melhor de se adapta ao processo de desenvolvimento com ênfase na arquitetura, pois permite a identificação incremental dos requisitos da aplicação, dos riscos, das restrições e dos objetivos.

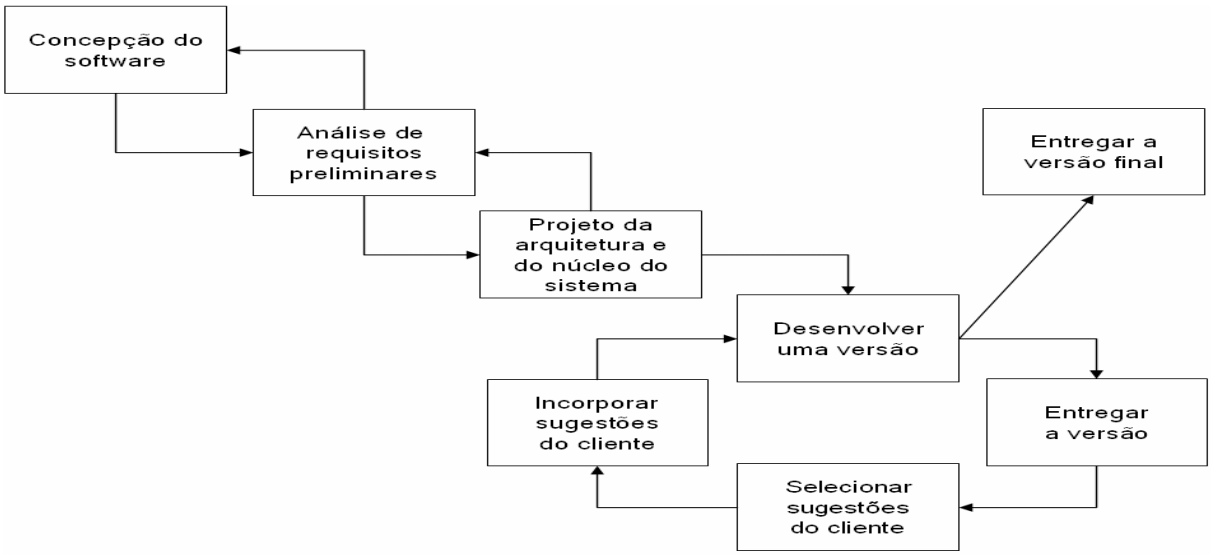

Figura 3.3 - Modelo de ciclo de vida evolucionário

Fonte: (BASS et al., 2003)

O paradigma de modelo espiral é atualmente a abordagem mais realística para o desenvolvimento de softwares e sistemas em grande escala. Utiliza uma abordagem "evolucionária", capacitando o desenvolvedor e o cliente a entender e reagir aos riscos em cada etapa evolutiva da espiral. Também usa a prototipação como um mecanismo de redução de riscos, a qual pode ser aplicada em qualquer ponto evolutivo. Porém, o arquiteto deve estar ciente de que, se um grande risco não for descoberto a tempo, com certeza ocorrerão problemas.

3.2.3 Atividades no processo de Arquitetura de Software

No processo de desenvolvimento de software com enfoque arquitetural, diversas atividades devem ser desempenhadas, entre as quais podem ser destacadas (ALBIN, 2003):

- participar do modelo de negócio (Pré-projeto): essa participação possibilita contribuir com informações técnicas estratégicas referentes à interação entre sistemas, tempo de desenvolvimento e custos técnicos em geral;
- entendimento de requisitos funcionais fundamentais (Análise de Domínio): isso irá permitir a definição do modelo de domínio a ser utilizado pela arquitetura;
- criação ou seleção de uma arquitetura (Projeto Esquemático): dependendo do sistema, existem modelos arquiteturais genéricos que irão direcionar o processo de modelagem arquitetural do sistema. Nesta etapa, deve ser escolhido o modelo que melhor se adapte ao sistema ou deve ser criado um modelo mais adequado;
- escolha de elementos técnicos (Desenvolvimento do Projeto): diz respeito à escolha de tecnologias que suportem a implementação. São questões relevantes à linguagem, às ferramentas de apoio ao desenvolvimento e de documentação, etc.;
- documentação da arquitetura em visões: existem diversos tipos de usuário, e cada um deve ter uma documentação orientada à sua visão. Assim, um gerente deve ter uma documentação da arquitetura diferente da documentação recebida pela pessoa que implementa o sistema;
- implementação orientada pela arquitetura (Construção): a arquitetura irá limitar tecnologias e organizar todo o processo de desenvolvimento, e os desenvolvedores devem se restringir a ela; e
- avaliação da arquitetura: esta atividade deve ser desempenhada durante todo o processo de desenvolvimento, de forma a permitir que os pontos fracos da

arquitetura possam ser detectados e melhorados em versões posteriores da arquitetura em outros sistemas.

3.2.4 Considerações relevantes em Arquitetura de Software

Muitas vezes, por pressão do mercado, suprime-se a fase de modelagem arquitetural, e os desenvolvedores buscam de imediato a implementação e a infra-estrutura na qual o sistema irá operar. Como resultado, geram-se sistemas "engessados", e por muitas vezes não se consegue atingir o objetivo do projeto.

A construção de software baseado em arquiteturas tem a capacidade de melhorar os atributos de qualidade e reduzir a complexidade, mas diversas restrições de projeto podem alterar a organização do processo de desenvolvimento, o que deve ser levado em consideração quando se pensa em introduzir os conceitos de Arquitetura de Software (BASS, 2003). Quando uma arquitetura já está definida, é fácil alcançar esses atributos (WENTZEL, 1994). Contudo, a construção em paralelo de uma arquitetura organizada e genérica pode tomar um tempo não aceitável, dependendo das restrições do projeto.

Um outro aspecto que deve ser levando em consideração é que freqüentemente as arquiteturas são utilizadas como instrumento de vendas, não representando o que se tem de melhor na área. Através de uma forte propaganda, os fabricantes de software tentam demonstrar que seus produtos são os melhores. Malveau (2000) diz que o problema com essas arquiteturas de marketing é que elas são desassociadas do processo de desenvolvimento, e a menos que o arquiteto de software gerencie o modelo computacional, esse tipo de arquitetura não traz nenhum benefício técnico.

Contudo, quando os requisitos de um projeto permitem o desenvolvimento com enfoque arquitetural, os benefícios esperados podem ser alcançados. Booch et al. (1999) dizem que a Arquitetura de Software é o produto do desenvolvimento que dá o maior retorno de investimento com respeito a qualidade, cronograma e custos. Isso se dá devido ao fato de que a arquitetura é estabelecida no princípio do desenvolvimento de um produto e pode ser estendida a outros produtos que futuramente venham a serem desenvolvidos. Ao se elaborar uma boa arquitetura, tem-se como resultado a simplificação do desenvolvimento, das integrações, da fase de testes e das futuras modificações e manutenções (ALBIN, 2003).

Bass et al. (2003) abordam outro aspecto importante no uso de uma arquitetura. Os autores dizem que a arquitetura do software representa uma abstração comum em alto nível de um

sistema. A maioria dos participantes pode usar essa arquitetura como base para criar um entendimento mútuo, formar consenso e comunicar-se com os outros.

Resumidamente, podem ser enumeradas as seguintes vantagens de se ter uma arquitetura de software bem definida:

- um framework que satisfaz os requisitos dos sistemas;
- um ponto de partida técnico para o projeto de sistemas;
- uma forma viável de estimação de custos e gerenciamento do processo;
- uma efetivação do reuso em desenvolvimento; e
- uma base para a análise de consistência e dependência.

3.3 Técnicas em Arquitetura de Software

Diversos motivos podem gerar a necessidade de se construir ou alterar uma arquitetura de software utilizada na infra-estrutura do desenvolvimento. Os motivos mais comuns seriam os seguintes: requisitos de um novo sistema, deficiências na arquitetura atual, evolução tecnológica, etc. Qualquer que seja o motivo, a tarefa de se definir uma nova arquitetura é difícil e demanda muita experiência.

Dessa forma, para a elaboração de uma arquitetura bem-sucedida é de suma importância que se faça uma análise das experiências de outros projetos que alcançaram sucesso. A utilização das experiências anteriores de arquitetos e a semelhança existente entre arquiteturas trazem benefícios substanciais no que diz respeito à reutilização e adoção de estratégias previamente validadas (BASS et al., 2003). Estratégias bem-sucedidas acabam por se tornar princípios e padrões referenciais para o projeto focado na arquitetura de software.

3.3.1 Referências em Arquitetura de Software

No processo de desenvolvimento de um sistema, desde a análise de requisitos até a implementação, se configura um processo contínuo de aproximação entre as informações arquiteturais e as respectivas possíveis soluções, como mostrado na Figura 3.4. No extremo mais abstrato, o sistema se encontra num momento conceitual em que inúmeras soluções são possíveis. No outro extremo, o concreto, existe apenas uma única solução. Foram definidos três pontos nesse processo de aproximação, com a finalidade de se contextualizar o uso de referências na tomada de decisões estruturais para um sistema.

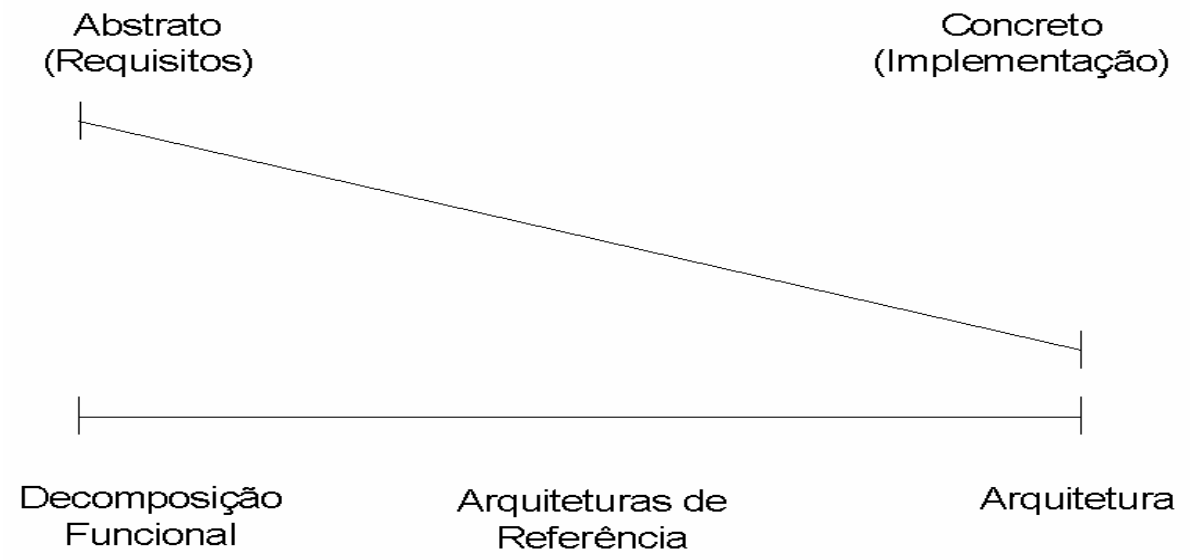

Figura 3.4 - Desenvolvimento de projeto com enfoque arquitetural

Como podemos ver na figura acima, a todo momento há a necessidade de se tomar decisões que levem à escolha de soluções adequadas para os problemas arquiteturais. As decisões tomadas no início de um projeto têm um maior impacto na arquitetura do que aquelas tomadas no final do projeto. Nesse processo, ao se deparar com um problema, as soluções são, prioritariamente, retiradas de um catálogo de arquiteturas de referências. Este catálogo fornece todos os tipos de informações necessárias sobre a solução do problema: o contexto; a solução; uma referência arquitetural; exemplos de uso; e, os benefícios e malefícios do uso desta solução.

Bass et al. (2003) definem os conceitos de Padrão Arquitetural, Modelo de Referência e Arquitetura de Referência como uma forma de classificar os níveis de abstração existentes para a padronização dessas referências. Esses três critérios não são arquiteturas por si mesmos, mas oferecem uma abstração conceitual que direciona o fundamento básico para os próximos passos (BASS et al., 2003). Abaixo é apresentado um exemplo da utilização desses conceitos na concepção de arquiteturas de softwares que ajuda a entender de maneira resumida o relacionamento entre os conceitos.

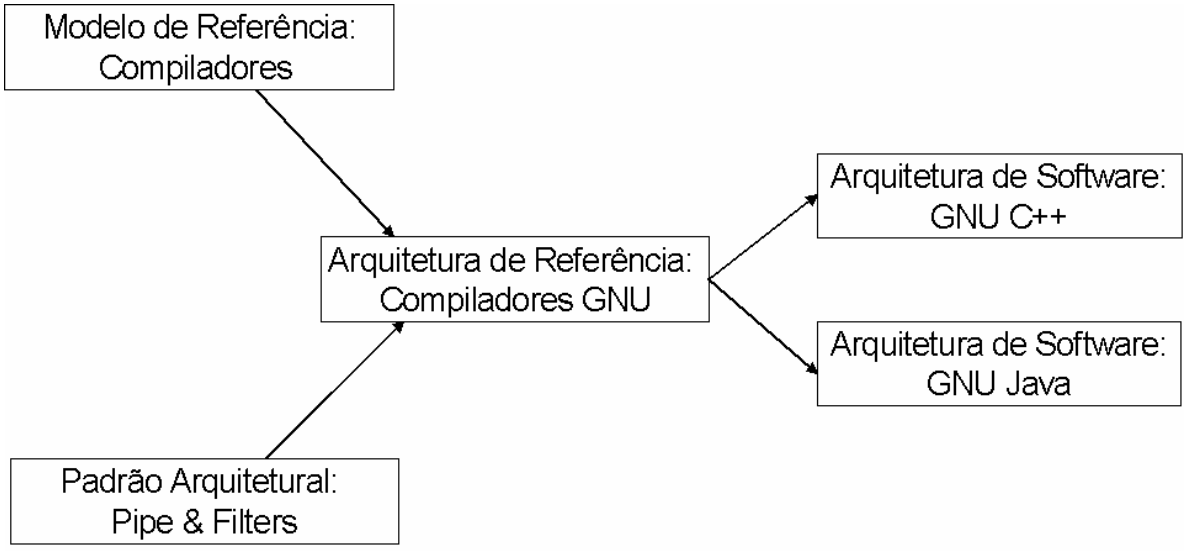

Figura 3.5 - Exemplo de utilização de referências na definição de uma arquitetura

Fonte: Baseado em Bass et al. (2003)

No exemplo acima vemos a concepção de duas arquiteturas com base em uma arquitetura de referência, a qual foi concebida pela composição de padrões arquiteturais tendo em mente um modelo de referência já conhecido, os compiladores.

Ainda na mesma linha de estabelecer referências para o projeto arquitetural, o grau de padronização pode ser ainda mais detalhado. Buschmann (1996) apresenta a existência de padrões de projeto e idiomas. Os padrões de projeto procuram solucionar os problemas das estruturas dos sistemas orientadas a objetos. Já os idiomas são soluções padronizadas para os problemas de implementação em uma linguagem em particular.

3.3.2 Padrão arquitetural

Um padrão descreve uma solução para um problema que ocorre com freqüência durante o desenvolvimento de software, podendo ser considerado como um par "problema–solução" (BUSCHMANN et al., 1996). Projetistas familiarizados com certos padrões podem aplicá-los imediatamente a problemas de projeto, sem ter que redescobri-los (GAMMA et al., 1995). Um padrão é um conjunto de informações instrutivas que possui um nome e que capta a estrutura essencial e o raciocínio de uma família de soluções comprovadamente bem-sucedidas para um problema repetido que ocorre sob um determinado contexto em um conjunto de repercussões (APPLETON, 1997).

Pode-se pensar num padrão arquitetural como uma forma de identificar a essência de uma solução e torná-la suficientemente genérica para ser utilizada em problemas similares. O padrão funcionaria como um elo entre o problema e a solução. Os padrões arquiteturais

seriam o nível mais alto de abstração de padrões para um projeto de software, nível este que estaria situado, no processo de desenvolvimento de software, no projeto estrutural (BUSCHMANN, 1996).

Bass et al. (2003) descrevem padrão arquitetural como sendo uma descrição dos elementos e tipos de relacionamentos juntamente com um conjunto de restrições de como eles podem ser utilizados. Os autores dizem ainda que um padrão arquitetural pode ser pensado como um conjunto de restrições sobre uma arquitetura, sobre seus elementos e seus padrões de interação, e essas restrições definem um conjunto ou uma família de arquiteturas que a satisfazem.

Os padrões arquiteturais expressam esquemas de organização estrutural fundamentais para sistemas de software, pois fornecem um conjunto de sistemas predefinidos, especificam suas responsabilidades e incluem regras e linhas mestras para a organização do relacionamento entre eles (BUSCHMANN et al., 1996).

Para a especificação de um padrão arquitetural é fundamental que sejam abordados os tópicos descritos a seguir.

- Contexto: é uma descrição do domínio de aplicabilidade do padrão.
- Problema: é o detalhamento do domínio do problema de maneira a identificar os elementos componentes da solução e seus requisitos.
- Solução: é uma explicação em termos gerais de como resolver o problema identificado.
- Estrutura: é a especificação do padrão propriamente dito. Define os elementos componentes do padrão juntamente com suas responsabilidades e colaborações. É uma definição formal de como o problema pode ser resolvido.
- Dinamismo: é uma apresentação dos cenários de uso do padrão e serve para melhorar o entendimento desse padrão. Diversos cenários são apresentados de maneira a exaurir as possibilidades dinâmicas do uso do padrão.
- Implementação: é um exemplo de implementação do padrão que serve para concretizar os conceitos apresentados na especificação.
- Variantes: apresenta sucintamente outros padrões derivados do padrão corrente.
- Conseqüências: apresenta os benefícios e malefícios do uso do padrão.

- Usos conhecidos: quando um padrão já é bem conhecido, esse tópico apresenta os principais modelos de referência existentes.

Ao se observarem as especificações de diversos padrões arquiteturais, é possível notar a existência de quatro tipos de propriedades comuns dos padrões, relacionadas a seguir.

1. Os padrões fornecem um vocabulário de projeto – tipos de componentes e conectores tais como pipes, filtros, clientes, servidores, parsers, base de dados, etc.
2. Estabelecem restrições de topologia que determinam as composições permitidas desses elementos.
3. Determinam a interpretação semântica pela qual a composição de elementos de projeto, devidamente restringidos pelas regras de configuração, tem seu significado bem definido.
4. Definem a análise que pode ser feita sobre sistemas construídos naquele estilo.

Segundo Buschmann et al. (1996), os principais padrões utilizados em projetos de software são os que se seguem.

- Pipes & Filters: definem a estrutura de sistemas que utilizam um mecanismo seqüencial de fluxo de dados. Os componentes processadores de dados executam o processamento de maneira seqüencial, de modo que a saída de um componente se torna entrada em outro, e assim sucessivamente (ALBIN, 2003).
- Camadas (layers): propõem a decomposição de um problema em grupos de subtarefas, em que cada grupo se apresenta em um nível particular de abstração.
- Blackboard: são padrões caracterizados por problemas que, quando decompostos em subproblemas, abrangem muitos campos de conhecimento. A solução para problemas parciais necessita de diferentes paradigmas e representações. Vários subsistemas organizam o conhecimento para a construção de uma possível solução aproximada através do uso de uma memória compartilhada. Cada subsistema resolve uma tarefa em particular, e todos trabalham juntos para aquisição da solução. Não existe comunicação entre os subsistemas.
- Broker: padrões que propõem uma decomposição funcional para o problema de objetos distribuídos. O padrão broker promove o desacoplamento da implementação dos objetos que interagem transparentemente por invocações de

serviços remotos. O componente broker é responsável pela coordenação da comunicação, controlando as requisições, os resultados e as exceções.

- Model-View-Controller (MVC): este modelo de padrão remove o acoplamento entre a visão e o modelo por estabelecer um protocolo do tipo subscrição–notificação entre os dois. A visão deve assegurar que ela reflete o estado do modelo. Sempre que o modelo sofre alteração, este notifica as visões que dependem dele. O Controlador define a forma que a interface do usuário reage às interações com esse mesmo usuário (GAMMA et al., 1995).
- Presentation-Abstraction-Control (PAC): define uma estrutura para sistemas na forma de uma hierarquia de agentes cooperativos. Cada agente é responsável por um aspecto específico da funcionalidade da aplicação e é composto de três componentes: apresentação, abstração e controle. Essa subdivisão separa os aspectos de interação homem–máquina dos agentes de seu núcleo funcional e sua comunicação com outros agentes.
- Microkernel: padrões que se aplicam a sistemas de software que devem estar aptos a se adaptarem a mudanças nos requisitos do sistema. Separam um núcleo funcional mínimo das funcionalidades estendidas e das partes específicas do cliente e também servem como um soquete ao qual essas extensões podem ser conectadas e coordenadas.
- Reflection: é um padrão que permite mudar as estruturas e o comportamento de um sistema de software dinamicamente. Suporta a modificação dos aspectos fundamentais, tais como estruturas de tipos e mecanismos de chamada de função. Neste padrão, uma aplicação é dividida em duas partes. Um metanível fornece informações a respeito das propriedades selecionadas do sistema e torna o software consciente disso. Um nível-base inclui a lógica da aplicação. A implementação é feita sobre o metanível. Uma interface é especificada para a manipulação dos metaobjetos do metanível através de um protocolo de metaobjetos.

Embora um padrão defina uma solução para um problema de domínio em particular, normalmente os sistemas estão envolvidos com diversos problemas de domínios diferentes. Dessa forma, a maioria dos sistemas não pode ser estruturado de acordo com um único padrão arquitetural. Diversos requisitos fazem com que os sistemas utilizem vários padrões a fim de

atingir a meta do sistema (BUSCHMANN, 1996). Um sistema Web, por exemplo, normalmente utiliza os padrões Broker, MVC e Layers em sua arquitetura.

É importante ressaltar que o simples uso de diversos padrões arquiteturais não garante a qualidade da solução arquitetural. A devida composição de uma nova arquitetura exige competência e estudo aprofundado para que tenha a eficácia necessária a que se propõe. Mesmo arquiteturas consolidadas, como a dos sistemas Web, apresentam falhas de concepção, como veremos mais adiante no capítulo 4.

3.3.2.1 Categorizações dos padrões arquiteturais

Os padrões arquiteturais podem ser organizados de acordo com as similaridades existentes entre eles. Buschmann et al. (1996) apresentam as categorizações que se seguem.

- From Mud to Structure: padrões que apóiam a decomposição adequada de uma tarefa do sistema em subtarefas que cooperam entre si. Ex.: Layers, Pipes & Filters, Blackboard.
- Sistemas Distribuídos: padrões que fornecem infra-estrutura para sistemas que possuem componentes localizados em processadores diferentes ou em diversos subsistemas e componentes. Ex.: Broker.
- Sistemas Interativos: padrões que ajudam a estruturar sistemas com a interface homem–máquina. Ex.: MVC, PAC.
- Sistemas Adaptáveis: padrões que oferecem a infra-estrutura que apóia a decomposição adequada de subsistemas e componentes complexos em partes cooperativas. Ex.: Microkernel, Reflection.
- Organização do trabalho: padrões que definem como componentes colaboram para fornecer um serviço complexo.
- Controle de Acesso: padrões que guardam e controlam acesso a serviços de componentes.
- Gerenciamento: padrões para lidar com coleções homogêneas de objetos, serviços e componentes.
- Comunicação: padrões que ajudam a organizar a comunicação entre os componentes.

- Manuseio de Recursos: padrões que ajudam a gerenciar componentes e objetos compartilhados.

3.3.3 Modelo de referência

Um modelo de referência é uma divisão de funcionalidades juntamente com o fluxo de dados entre as partes. É uma decomposição padrão de um problema conhecido em partes que cooperativamente resolvem o problema. Advindos da experiência, os modelos de referência são características de um domínio maduro e descrevem em termos gerais como as partes se inter-relacionam para alcançar seu propósito coletivo (BASS et al., 2003).

Quando um problema é bem conhecido, a partes funcionais que o compõem são facilmente identificadas. Quando se pensa em compiladores, automaticamente surge uma decomposição do problema, a qual se caracteriza como um modelo de referência.

Dependendo do escopo de uma arquitetura de referência, o próprio padrão arquitetural pode se tornar modelo de referência. Um exemplo disso pode ser percebido no padrão MVC. Arquiteturas de software como o SWING ou MFC utilizam esse padrão como modelo de referência (veja Tabela 3-1).

Alguns modelos de referência acabam sendo padronizados. Comitês de padronizações formalizam uma decomposição de um problema conhecido. Um exemplo bem consolidado de modelo de referência padronizado é o modelo OSI, que define uma decomposição em camadas do problema de comunicação entre sistemas de software em um ambiente de rede coorporativa.

O uso de modelo de referência é como um atalho para a construção de uma arquitetura de software, o qual permite que a elaboração de uma nova arquitetura não tenha que se preocupar com os elementos funcionais daquele domínio, uma vez que estes já foram identificados.

Muitas vezes, o grau de decomposição e a abrangência de um modelo de referência são tão grandes que promovem a existência de outras decomposições mais simples para o mesmo problema. A Internet, por exemplo, é uma decomposição simplista do mesmo problema resolvido pelo modelo OSI.

3.3.4 Arquitetura de referência

A Arquitetura de Referência consiste em um modelo de referência mapeado sobre elementos de software, que cooperativamente implementam as funcionalidades definidas no modelo de referência e o fluxo de dados entre eles. Enquanto um modelo de referência define

a funcionalidade, uma arquitetura de referência é um mapeamento dessa funcionalidade em uma decomposição de sistema (BASS et al., 2003).

Uma arquitetura de referência é a arquitetura generalizada de diversos sistemas finais que compartilham um ou mais domínios comuns. A arquitetura de referência define a infra-estrutura comum aos sistemas finais e as interfaces de componentes que irão ser incluídas no sistema final. A arquitetura de referência é então instanciada para criar uma arquitetura de software de um sistema específico. A definição de uma arquitetura de referência facilita derivar e estender novas arquiteturas de software para classes de sistemas. Além disso, uma arquitetura de referência exerce um papel dual em relação às arquiteturas de softwares-alvo. Primeiramente, generaliza e extrai funções e configurações comuns. Segundo, ela fornece uma base para instanciar os sistemas-alvo que usam aquela base comum de maneira mais confiável e barata. O conceito de uma arquitetura de referência é muito similar a um framework de aplicação (GALLAGHER, 2000).

Podem existir diversas arquiteturas de referência para um mesmo modelo. Isso se dá justamente porque há muitas maneiras de se transformar um modelo de referência em uma arquitetura de referência e, além disso, alguns modelos de referência são tão genéricos que podem ser aplicados em domínios de problemas diferentes. Pelo mesmo motivo, podem existir diversas arquiteturas de software para cada arquitetura de referência.

A tabela abaixo apresenta alguns exemplos de derivações na concepção de arquiteturas de software.

Modelo de Referência	**Arquitetura de Referência**	**Arquitetura de Software**
Sistemas Distribuídos	CORBA	TOA MICO Visibroker
	COM/DCOM	DCOM da Microsoft EntireX da Software AG
GUIs	MVC	SWING Struts e JSF SOFIA
	Document View	VCL da Borland QT da TrollTech QTK da GNU
Interpretadores	Máquina Virtual	JVM (Java Virtual Machine) CIL (Common Interpreter Language)

Tabela 3.1 - Derivações em Arquitetura de Software

3.3.5 Padrões de projeto

Um padrão de projeto identifica o particionamento das classes e instâncias, suas regras e colaborações e a distribuição das responsabilidades. Cada padrão de projeto toma por foco um problema do projeto orientado a objeto em particular. O padrão descreve onde ele pode ser aplicado e se é possível sua aplicação levando-se em conta outras restrições de projeto, as conseqüências e os custos de seu uso (GAMMA et al., 1995). Também fornece dicas de implementação e exemplos. A solução é um arranjo de objetos e classes que solucionam o problema, sendo então customizada e implementada para resolver uma questão num contexto em particular.

Gamma et al. (1995) organizaram um conjunto de padrões em função das características de criação, de comportamento e de estruturas. Os padrões descritos no livro desses autores passaram a ser conhecidos na comunidade como GoF (Gang of Four), justamente pelo fato de serem quatro os autores e o livro ser uma referência em padrões de projeto. Os padrões da GoF são padrões genéricos que podem ser aplicados durante o projeto de software orientado a objeto. São classificados quanto ao propósito, que reflete o que o padrão faz, e quanto ao escopo, que define se o padrão é adequado primeiramente para classes ou para objetos. A tabela abaixo é um resumo esquemático desses padrões.

		Propósito		
		De criação	**Estrutural**	**Comportamental**
Escopo	**Classe**	Factory Method	Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Flyweight Observer State Strategy Visitor

Tabela 3.2 - Espaço de padrões de projeto

Fonte: (GAMMA et al., 1995)

3.3.5.1 Padrões de criação

- Abstract factory: fornece uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.
- Builder: separa a construção de um objeto complexo de sua representação. Assim, o mesmo processo de construção pode criar diferentes representações.
- Factory method: define uma interface para criar um objeto sem deixar as subclasses decidirem qual classe instanciar.
- Prototype: especifica os tipos de objetos a serem criados usando uma instância de prototipação e cria novos objetos por copiar esses tipos.
- Singleton: garante que uma classe tenha somente uma instância e fornece um ponto de acesso global para essa classe.

3.3.5.2 Padrões estruturais

- Adapter: converte a interface de uma classe em outra interface. Os adaptadores permitem que classes de diferentes procedências trabalhem junto, o que não seria possível em virtude da incompatibilidade de interfaces.
- Bridge: tira o acoplamento de uma abstração de sua implementação para que os dois possam variar independentemente.
- Composite: compõe os objetos em três estruturas para representar uma hierarquia todo–parte. Este padrão permite que os componentes tratem objetos individuais e composição de objetos uniformemente.
- Decorator: vincula responsabilidades adicionais a um objeto dinamicamente. Este padrão oferece uma alternativa flexível para fazer subclasses que estendam as funcionalidades.
- Facade: fornece uma interface unificada para um conjunto de interfaces em um subsistema. Define uma interface de nível mais alto que torne o subsistema mais fácil de ser usado.
- Flyweight: usa compartilhamento para suportar um grande número de objetos de pequena granularidade de maneira eficiente.
- Proxy: toma o lugar de um outro objeto e controle o acesso a ele.

3.3.5.3 Padrões comportamentais

- Chain of responsibility: evita acoplar o emissor de uma requisição a seu receptor por dar a mais de um objeto a chance de manipular a requisição. Restringe o objeto receptor e passa a requisição, junto com a restrição, para o objeto que a manipula.
- Command: encapsula uma requisição como um objeto de tal forma que os clientes possam ser parametrizados com diferentes requisições, filas ou log de requisições e suporta operações que não podem ser desfeitas.
- Interpreter: dada uma linguagem, define uma representação para sua gramática juntamente com um interpretador que usa a gramática para interpretar as seqüências na linguagem.
- Iterator: fornece uma forma de acessar os elementos de um objeto agregado seqüencialmente sem expor sua representação.
- Mediator: encapsula o comportamento coletivo em um objeto mediador separado dos demais objetos. O mediador funciona como um intermediário que evita que os objetos do grupo referenciem uns aos outros explicitamente. Cada objeto comunica-se com o mediador sempre que necessitar se comunicar com outro objeto.
- Memento: sem violar o encapsulamento, captura e externaliza um estado interno do objeto para que este possa ser restaurado depois ao mesmo estado. Define uma relação de dependência 1:N, de forma que os demais (observadores) são notificados quando um certo objeto (assunto) tem seu estado modificado. Possibilita baixo acoplamento entre os objetos observadores e o assunto.
- Observer: define uma dependência um-para-muitos entre objetos para que todas as suas dependências sejam notificadas e atualizadas automaticamente quando um objeto troca de estado.
- State: permite a um objeto alterar seu comportamento quando seu estado interno muda.
- Strategy: define uma família de algoritmos, os encapsula e os faz intercambiáveis. Este padrão permite que o algoritmo seja modificado, independente do cliente que o usa.
- Template method: define um esqueleto de um algoritmo em operação deferindo alguns passos para as subclasses sem mudar a estrutura do algoritmo.

- Visitor: este padrão define elementos estruturais que permite a operacionalização das classes de maneira transparente, isto é, sem que a interface de programação dessas classes seja afetada.

3.3.6 Idiomas

Os idiomas representam os padrões de mais baixo nível. Em contraste com os padrões de projeto, os quais visam padrões estruturais mais abrangentes, os idiomas descrevem como resolver um problema específico de implementação em uma linguagem de programação, tal como gerenciamento de memória em C++. Os idiomas podem também visar à implementação concreta de um padrão de projeto específico. Não existe uma divisão clara entre padrão de projeto e idiomas (BUSCHMANN, 1996).

Os idiomas fornecem soluções para o uso competente de uma linguagem de programação. Com eles, todos os aspectos da programação são catalogados e padronizados de maneira a promover um estilo de programação eficiente e comum.

A definição de idiomas é feita pela tríade descrita a seguir.

- Nome: identificação unívoca do idioma. Esta identificação deve ter uma expressividade capaz de permitir sua rápida localização.
- Problema: apresentação de qual problema está sendo resolvido por este idioma. Tal apresentação deve contextualizar completamente o problema.
- Solução: diz exatamente o que fazer para alcançar a solução do problema. Normalmente vem acompanhado de um ou mais exemplos que demonstram claramente como a solução pode ser alcançada.

As áreas de abrangência abordadas por idiomas são variadas. Existem idiomas para resolver as questões do gerenciamento de memória, formatação de código, criação de objetos, nomeação de variável, classes e métodos, uso eficiente de uma biblioteca de componentes, e assim por diante. A eficiência dos idiomas irá depender diretamente da qualidade de suas definições e da facilidade em acessá-los.

3.3.7 Linguagens para a descrição arquitetural

A arquitetura do software descreve a estrutura e o comportamento de um sistema de software e os elementos não-software com o qual ela faz interfaciamento.

A criação de uma arquitetura de software promove um melhor entendimento do sistema, fornecendo a base para uma análise rigorosa do projeto, o que torna possível a detecção prévia de erros. Dessa forma, é possível melhorar a qualidade do software e ajudar a assegurar a correção desses erros.

Geralmente, as arquiteturas têm sido representadas de maneira informal através de diagramas de caixas e linhas, em que a natureza dos componentes, suas propriedades, a semântica das conexões e o comportamento do sistema em seu conjunto é pobremente definido (BASS et al., 1998).

Uma linguagem de descrição arquitetural (ADL) é um mecanismo formal ou semiformal para se especificar arquiteturas de software. As ADLs fornecem um meio para especificar arquiteturas de software, de modo a permitir que sejam feitas análises com critérios rigorosos. Uma ADL também pode ser gráfica ou um misto entre linguagens textuais e gráficas (SCTG, 2003).

Para que uma linguagem seja considerada ADL é necessário que represente claramente os conceitos de arquitetura de software e suporte padrões; base para análise arquitetural; e, habilidade de descrever interações arquiteturais (Allen, 1997).

Um número considerável de ADLs tem sido proposto, embora nenhuma tenha se tornado um padrão de fato, todas procuram ter expressividade o suficiente para a descrição arquitetural. As ADLs mais conhecidas são:

Proponente	**ADL**
Universidade Carnegie-Mellon	ACME (a mais referenciada)
	AESOP
	UNICON
	WRIGHT
Universidade do Texas	GEN-VOCA
Honeywell	Meta-H
Universidade de Stanford	RAPIDE
SRI Internacional	SADL
Universidade da Califórnia	xADL

Tabela 3.3 - Principais ADLs

Fonte: (SCTG, 2003)

Dependendo do grau de complexidade de uma arquitetura de software, pode não haver a necessidade do uso de uma ADL formal. Contudo, as ADL semiformais gráficas são sempre bem-vindas, uma vez que fornecem elementos gráficos padronizados para a representação da arquitetura.

3.4 Considerações finais

Este capítulo mostrou uma visão geral da disciplina Arquitetura de Software e seus produtos. Foram abordadas a contextualização do termo, o qual muitas vezes tem sido usado erroneamente, juntamente com as definições mais comumente aceitas na literatura. Também foram apresentadas as principais escolas em Arquitetura de Software e foi definida a escola guia para esta dissertação.

Foram relacionados também neste capítulo o processo de arquitetura de software e sua relação com o processo de desenvolvimento de sistemas. Apresentaram-se o ciclo de vida mais adequado ao processo de desenvolvimento de sistemas com enfoque arquitetural e as atividades relacionadas a cada etapa desse ciclo. Foram feitas também importantes considerações sobre a concepção de arquiteturas e as armadilhas que o mercado pode trazer para tais arquiteturas.

Com o objetivo de fundamentar a elaboração da arquitetura de referência proposta, foram apresentadas as principais técnicas utilizadas na concepção de arquiteturas de software. Foram vistos o conceito e exemplos em arquiteturas de referência, padrões arquiteturais, modelos de referência, padrões de projeto e idiomas. Também foi apresentado o estágio atual das linguagens de descrição arquitetural e demonstrado como ainda se encontram num estágio embrionário para serem de fato utilizadas na especificação de arquiteturas.

4 Arquitetura para aplicações de Governo Eletrônico

Nos capítulos anteriores descreveu-se a natureza das plataformas de E-Gov e a arquitetura de software. Esses conceitos são importantes para a compreensão da proposta do presente trabalho. Uma vez apresentados, pode-se mostrar a arquitetura de software elaborada para os requisitos dos sistemas de E-Gov.

Este capítulo está dividido em duas partes: na primeira, apresentam-se os requisitos dos sistemas de E-Gov e o motivo pelo qual a forma como os sistemas são montados atualmente não é adequada; na segunda parte apresentam-se a arquitetura elaborada e todos os seus componentes.

Ao final do capítulo, discute-se como a arquitetura pode ser aplicada na atual conjuntura das aplicações de software de E-Gov.

4.1 Requisitos determinantes das arquiteturas

Durante os últimos 20 anos as aplicações de software têm evoluído dos ambientes de mainframes com terminais "burros" para as aplicações Web de que dispomos hoje. Em cada etapa dessa evolução as aplicações têm se modificado em suas arquiteturas em função da infra-estrutura e das tecnologias existentes.

Ao se observar esse avanço, é possível notar que a arquitetura e as tecnologias associadas tendem a evoluir com base nos seguintes critérios: funcionalidade, infra-estrutura disponível e amigabilidade. Os critérios de funcionalidade e infra-estrutura são impeditivos e, por essa razão, são critérios determinantes da arquitetura das aplicações, como ocorreu na época dos mainframes. O critério de amigabilidade, embora não fundamental, possui uma grande importância no que diz respeito à parte política da aplicação. Por isso, à medida que os equipamentos que apresentavam as aplicações para os usuários evoluíram de terminais "burros" para microcomputadores poderosos, as aplicações também evoluíam no sentido de explorar esse nova infra-estrutura, a fim de torná-las mais amigáveis e, por questões de infra-estrutura, menos consumidoras de processamento dos servidores.

O gráfico abaixo sumariza essa evolução e mostra o que era provido pelos servidores e o que os usuários percebiam do processo. Nota-se que as arquiteturas de desenvolvimento de aplicações têm variado entre concentrar o processamento em uma das pontas da aplicação: no cliente ou no servidor.

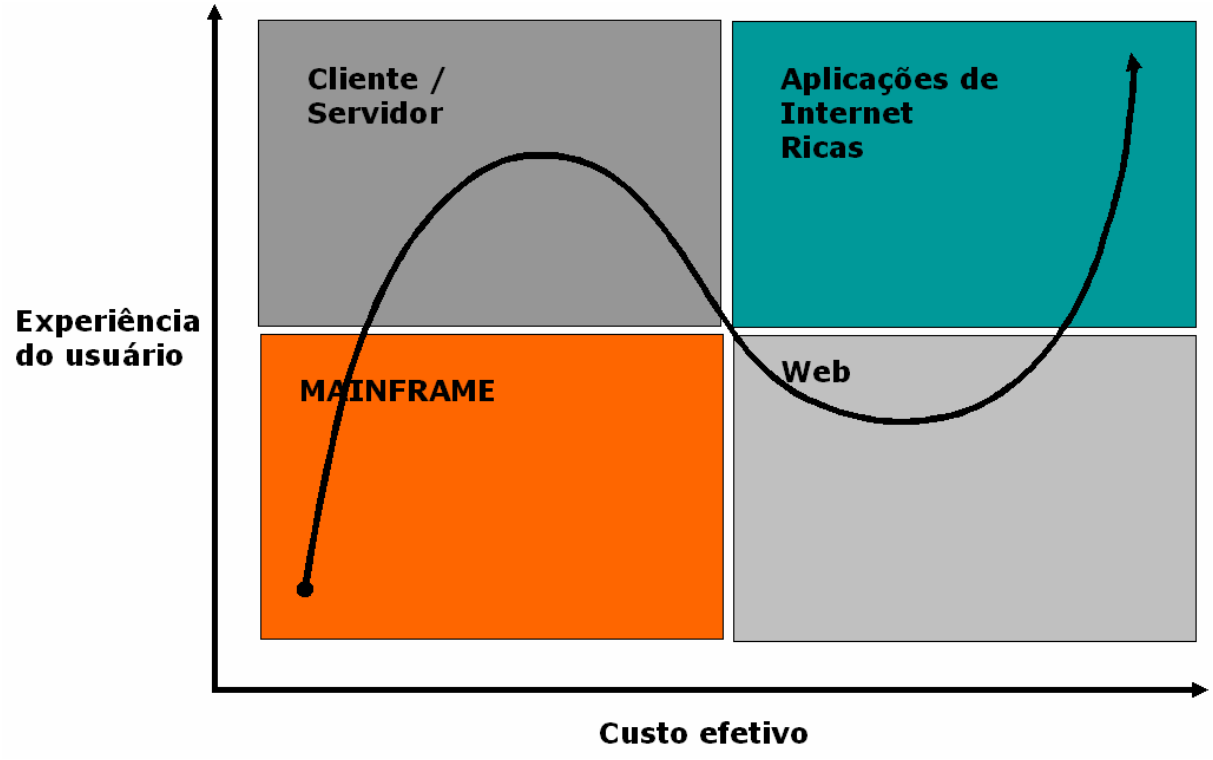

Figura 4.1 - Evolução do desenvolvimento de aplicações

Fonte: Laszlo Systems

Na última década, com o advento de uma infra-estrutura de interconexão mundial de computadores (WWW – World Wide Web), surgiram aplicações on-line com alto grau de disponibilidade (aplicações Web). Tais aplicações demandaram a elaboração de uma nova arquitetura que fosse capaz de atender aos requisitos dessa nova classe de aplicações.

Em todo o processo de evolução tecnológica há um refinamento sucessivo das soluções dadas. Por isso, as primeiras aplicações on-line foram hipertextos que eram disponibilizados através de um navegador de hipertexto conectado à Web, em que o navegador fazia o papel de um terminal "burro". Tanto os navegadores quanto a infra-estrutura têm evoluído e se tornado muito mais poderosos, viabilizado a elaboração de sistemas on-line mais complexos. Na atualidade, existe uma tendência de os sistemas se tornarem Web (GOODMAN, 2002). Essa tendência está baseada principalmente na simplicidade de distribuição que esse tipo de aplicação apresenta.

4.2 Problemas das aplicações

Como pudemos ver, a maioria das aplicações são desenvolvidas em função das tecnologias existentes no momento da concepção de um projeto. As tecnologias que forem mais bem divulgadas acabam por ser as escolhidas, e o projeto segue cegamente os ditames dessas

tecnologias. Muitas vezes os modelos arquiteturais disponíveis nessas tecnologias não são os mais adequados aos requisitos da aplicação, e quando isso é percebido, já é tarde demais.

As soluções tecnológicas mais escolhidas no desenvolvimento de aplicações para governo são aquelas que apresentam uma visão minimalista dos recursos de software existentes no equipamento em que a aplicação irá operar. Isso ocorre porque os grupos que desenvolvem as aplicações não podem confiar no ambiente operacional para qual as aplicações são desenvolvidas, uma vez que o universo de equipamentos e tecnologias de apoio são extremamente diversificados. Então, nesse tipo de contexto, a distribuição das aplicações como blocos monolíticos se torna a solução menos problemática. Infelizmente, as aplicações que utilizam esse modelo de operação apresentam dificuldade para incluírem um requisito muito significativo em aplicações de governo: a atualização automática dos recursos instalados.

Outro aspecto importante é o fato de que cada domínio de aplicação possui requisitos particulares para os quais as tecnologias genéricas normalmente não fornecem solução. Então, os grupos que desenvolvem as aplicações elaboram suas soluções isoladamente, a fim de ter o recurso necessário sobre a tecnologia escolhida. Esse tipo de atitude promove um distanciamento desnecessário entre os grupos e, por essa razão, eleva o custo de desenvolvimento de aplicações de software feitas pelo ou para o governo.

O desenvolvimento isolado, provocado pelas razões citadas, também promove a falta de padronização das aplicações. E, quando as aplicações se apresentam totalmente desorganizadas e sem nenhum padrão entre si, é automaticamente criada uma idéia de incompetência administrativa. Esse pensamento é reforçado quando o cidadão se depara com os seguintes problemas encontrados hoje nas principais aplicações:

- repetição da manipulação das Unidades de Informação: duas ou mais aplicações que deveriam ser complementares entre si acabam por ter de manipular as mesmas informações comuns, uma vez que uma aplicação não pode confiar na outra, devido a incompatibilidades, pois cada aplicação fornece a sua solução para o problema. Por vezes são criados exportadores e importadores para minimizar o problema, mas esse tipo de solução não resolve o fato de a informação estar sendo gerenciada repetidamente por cada aplicação e com requisitos diferentes;
- falta de padronização na Interface Gráfica do Usuário (GUI): um dos requisitos dos sistemas que interagem diretamente com o cidadão é ser “amigável”. Isso significa

que a pluralidade de comportamentos diferentes nas GUIs, para o mesmo propósito, é algo que deveria ser evitado;

- falta de mecanismo para atualização dinâmica dos sistemas: o dinamismo da tecnologia e do negócio faz com que os sistemas precisem ser periodicamente atualizados. Os sistemas on-line, por serem uma espécie de terminal gráfico, acabam tendo esse mecanismo de forma implícita, pois qualquer atualização no servidor é automaticamente refletida no cliente. Os sistemas em E-Gov que residem na máquina do usuário deveriam possuir um mecanismo de atualização dinâmica que oferecesse a mesma facilidade que os sistemas on-line possuem. Poucos sistemas governamentais apresentam essa funcionalidade, e quando apresentam, ela não é explorada totalmente, o que faz com que a responsabilidade da atualização do sistema fique sempre a cargo do cidadão; e
- falta de versões para diferentes Sistemas Operacionais (SO): outro problema é a falta de versões dos sistemas governamentais para Sistemas Operacionais (SO) de domínio público. O Linux, por exemplo, pode ser uma ótima solução para redução de custos da estrutura governamental, e poucos sistemas que funcionam off-line estão prontos para rodar de maneira adequada sobre esse SO.

Para o governo, existe também o problema do custo operacional dos projetos em E-Gov, que acabam se tornando muito mais elevados, uma vez que apresentam os seguintes problemas:

- pluralidade de soluções tecnológicas redundantes utilizadas pelos projetos: isso dificulta a capacitação dos colaboradores participantes dos projetos de aplicações;
- desperdícios de esforços com o desenvolvimento de funcionalidades idênticas e incompatíveis entre os sistemas, o que leva a um considerável aumento de custo na produção das aplicações;
- dificuldade no gerenciamento do legado: fica quase impossível gerenciar um universo de aplicações construídas com as mais diversas tecnologias. A operacionalização de cada sistema depende da tecnologia de suporte, o que cria a necessidade da existência de especialistas; e

- manutenção das soluções proprietárias: como as soluções para os problemas dos projetos são dadas de forma proprietária, torna-se quase impossível um órgão que contratou uma certa aplicação assumir o projeto quando este é finalizado.

Desgaste no relacionamento com o cidadão, dificuldade de operacionalização e aumento do custo de produção e manutenção das aplicações são as principais razões pela qual o modelo de desenvolvimento de sistemas para E-Gov utilizado hoje não deve ser considerado adequado.

4.3 Determinando o modelo operacional da arquitetura

Quando um modelo arquitetural é escolhido para uma determinada aplicação em particular, todos os atributos de qualidade inerentes ao modelo são, por conseqüência, implicitamente herdados pela aplicação. Cada classe de aplicação possui requisitos próprios que irão exigir que os atributos de qualidade oferecidos pelo modelo estejam em conformidade. Dessa forma, ao serem analisados os requisitos necessários para as aplicações do domínio E-Gov, é necessário também identificar os problemas apresentados pelos modelos arquiteturais tradicionalmente utilizados no desenvolvimento desses sistemas.

O primeiro passo no desenvolvimento de um sistema é a escolha de um aspecto arquitetural muito importante: o modelo operacional da arquitetura. Quanto a esse modelo, uma aplicação pode ser desenvolvida de duas formas: (1) monoliticamente ou (2) de maneira compartilhada. Como estamos falando em nível de aplicação de SI, entendemos que o sistema operacional é um componente comum a ambos os modelos.

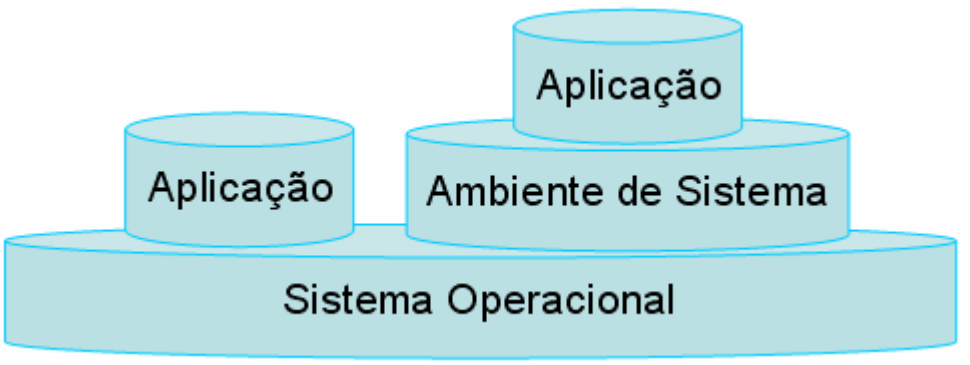

Figura 4.2 - Modelo operacional de arquiteturas

As aplicações desenvolvidas monoliticamente (standalone) fazem poucas suposições a respeito do ambiente no qual irá operar. O sistema operacional é tudo o que uma aplicação desse tipo pode supor, e, se o recurso necessário não estiver disponível, a aplicação terá de

incorporá-lo. As aplicações que são desenvolvidas seguindo essa suposição apresentam os seguintes problemas:

- dificuldade de gerenciamento de versão: os mecanismos de conexão existentes para os componentes de apoio dificultam grandemente a alteração de versão, pois tornam a manutenção da compatibilidade uma tarefa árdua;
- aumento do tamanho da aplicação: para que esses sistemas sejam viáveis de serem desenvolvidos, eles fazem uso de bibliotecas de componentes de terceiros, que são conectadas internamente à aplicação. Como todos os componentes de apoio são levados junto com a aplicação, esta se torna extensa em tamanho, o que pode causar problemas operacionais; e
- aumento da complexidade: os componentes de apoio se confundem com o problema da aplicação, o que resulta num aumento da complexidade da aplicação.

Os sistemas que executam tarefas simples de processamento são bem adaptáveis ao modelo monolítico, mas aplicações de grandes proporções que executam diversas tarefas diferentes sofrem com a característica estática desse modelo.

Já as aplicações desenvolvidas sob a ótica do compartilhamento determinam que deva existir um ambiente de componentes previamente estabelecidos para que os sistemas possam operar. O ambiente suposto está para as aplicações desse ambiente assim como o um sistema operacional está para suas próprias aplicações. Esse modelo separa devidamente os problemas dos componentes de apoio dos problemas da aplicação e oferece características desejáveis para os sistemas de governo, tais como redução da complexidade, aplicações de tamanho físico menor e simplicidade no gerenciamento de versões.

4.4 Determinando o modelo de referência da arquitetural

Uma vez escolhido o modelo operacional da arquitetura, o segundo passo é a escolha do modelo de referência estrutural. Os seguintes modelos de referência estruturais são os mais utilizados em aplicações de E-Gov: terminal “burro”, client-server, n-camadas e briefcase. Para escolher o modelo adequado para aplicações de E-Gov, deve-se levar em consideração o atual contexto tecnológico. Devido à realidade que é a Internet hoje, qualquer solução arquitetural que desconsidere os requisitos desse domínio de aplicação seria inadequada. As aplicações Web evidenciam dois requisitos fundamentais, os quais são descritos a seguir.

- Disponibilidade de recursos operacionais: quando uma aplicação fica exposta a uma quantidade indeterminada de usuários, o uso de recursos do SO necessários para atender às requisições tende a ser insuficiente. Por essa razão, a racionalização do uso de recursos operacionais do servidor deve ser meta indispensável. Nenhuma atividade além das estritamente essenciais deveria ser executada no servidor. Além disso, recursos de escalonamento de servidores tornam-se necessários. O modelo de terminal "burro", utilizado atualmente pela maioria das aplicações Web, definitivamente não atende a esses requisitos.
- Componentes de apoio: as arquiteturas que tentam enriquecer a qualidade da aplicação apresentada na Web, em sua maioria, se equivocam na questão do modelo operacional. Normalmente optam ou por aumentar o processamento do servidor com compilações sobre demanda ou por tornar as aplicações Web arquivos monolíticos extensos. As aplicações que se manifestam via Internet necessitam de um ambiente de componentes adequado a seus requisitos no cliente.

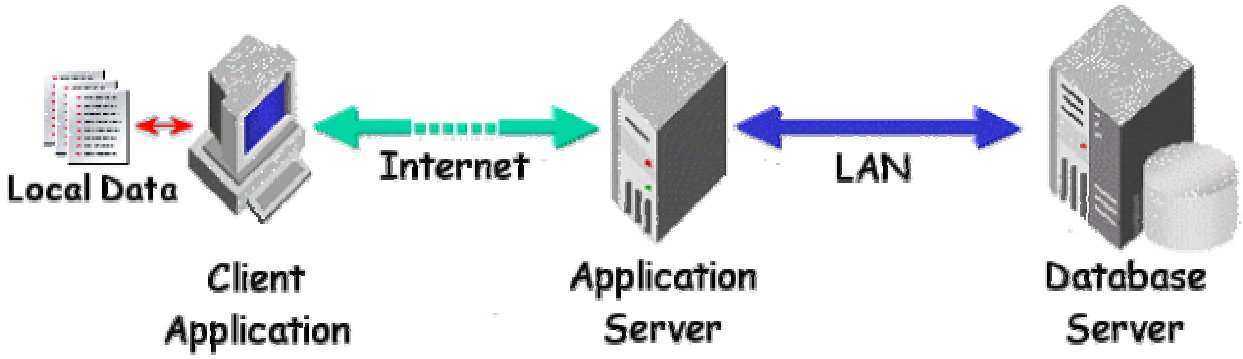

Figura 4.3 - Modelo estrutural Briefcase

O modelo estrutural briefcase, que é uma especialização do modelo n-camadas, é particularmente adequado aos sistemas de E-Gov. Nesse modelo o uso dos recursos remotos é minimizado ao máximo. O modelo admite duas formas de funcionamento: on-line e off-line. No modo off-line as aplicações funcionam normalmente, pois possuem todos os dados necessários para operar. Quando entram no modo on-line, essas aplicações criam uma atividade de sincronização das informações locais com as remotas e disponibilizam para o servidor todos os dados necessários a serem coletados. Da mesma forma, as aplicações podem obter informações do servidor, viabilizando a construção de verdadeiros Sistemas de Informação. Quando devidamente integrado em uma arquitetura, esse modelo pode perfeitamente atender também a aplicações estritamente Web.

4.5 Aspectos tecnológicos essenciais

Como já observado, as arquiteturas de software são influenciadas pelas infra-estruturas tecnológicas existentes e, simultaneamente, influenciam a elaboração de novas infra-estruturas. Por essa razão, devemos observar os aspectos tecnológicos que uma arquitetura adequada a sistemas de governo deve possuir, pois esses aspectos são fundamentais para garantir a viabilidade da solução dada.

4.5.1 Abstração de linguagem

A linguagem de desenvolvimento escolhida pode limitar as possíveis arquiteturas de hardware alvo, além de normalmente limitar os frameworks disponíveis para aquela linguagem/hardware/SO. Por exemplo, a linguagem Object Pascal está disponível apenas para processadores Intel/x86 e nos SOs Windows e Linux, e o framework VCL apenas no Windows.

Esse tipo de limitação pode ser resolvido se a linguagem estiver presente em todas as plataformas relevantes e simultaneamente possuir um framework robusto disponível nas mesmas plataformas. O problema é que nesse critério poucas linguagens se encaixam. Abaixo são elencadas as principais soluções existentes.

Linguagem	FrameWorks
GNU-C++	Qt/Trolltech (Comercial) wxWidget (Freeware)
SUN-Java	SWT (Freeware) SWING (Freeware)

Tabela 4.1 - Elenco de linguagens e frameworks multiplataforma

A situação ideal seria aquela em que o grupo desenvolvedor pudesse escolher a linguagem conforme a adaptabilidade para o tipo de problema a ser resolvido e, independente dessa escolha, o mesmo framework estivesse disponível. Nesse sentido, a tecnologia Dot Net atende completamente ao que se espera. Infelizmente, ainda não existe essa tecnologia de forma robusta em outras plataformas que não Windows.

4.5.2 Abstração de hardware

A abstração de hardware pode ser feita de duas maneiras: pelo framework ou por uma máquina virtual (MV). Em C++ utiliza-se abstração por framework e, no Java, a abstração é feita por MV.

O Java da Sun, relacionado na Tabela 4.1 - Elenco de linguagens e frameworks multiplataforma, é uma excelente solução que já se encontra bem estabelecida em uma vasta gama de plataformas de hardware e SOs (SIDDALINGAIAH, 2000). Infelizmente, a Sun não objetivou a neutralidade de linguagem ao especificar a MV, deixando-a muito específica para a linguagem Java, essa restrição não permite que certas linguagens possam ser totalmente representadas (MANZOOR, 2002). Até existem outras linguagens que geram código para a MV da Sun, mas estas acabam por sofrer algum tipo de adaptação para poder rodar sobre a MV (UDELL, 2000). Esse tipo de limitação é circunstancial e tende a ser solucionado no decorrer do tempo (MANZOOR, 2002).

Tanto o Java quanto o Dot Net rodam sobre uma MV. A MV é uma solução simplista para um problema complicado: a diversidade de hardwares e SOs. Ao se criar uma abstração de um equipamento fictício, elimina-se a necessidade de se ajustar o código-fonte da aplicação às peculiaridades de cada equipamento (MANZOOR, 2002). O problema nessa solução é que ela penaliza a performance e o uso de recursos da máquina.

Tendo-se uma perspectiva do que deve acontecer no futuro com as linguagens que utilizam MV, é possível identificar três possibilidades, as quais são descritas a seguir.

1. Surgimento de equipamentos nativamente compatíveis com a MV a fim de maximizar a performance, elevando assim a performance ao mesmo nível do C++.
2. O poder de processamento dos equipamentos será tal que as aplicações de E-Gov serão plenamente atendidas em seus requisitos de performance.
3. Ambas as situações ocorrendo simultaneamente. Equipamentos servidores, mais caros por natureza, focados na possibilidade 1, e equipamentos clientes focados na possibilidade 2.

Na atualidade, para minimizar os problemas de performance existem soluções como GCJ (GNU Compiler for the Java - freeware) e Excelsior JET. Essas ferramentas convertem o assembler da MV para o assembler da máquina real. A performance e o consumo de recursos são razoavelmente melhorados nesse caso.

4.5.3 Abstração de componentes

Este é o item menos percebido na hora de se definir uma arquitetura de software e seus respectivos componentes, mas no decorrer do tempo nota-se claramente que é um dos mais

importantes. De fato, a falta dessa abstração pode inviabilizar o desenvolvimento contínuo das aplicações.

É comum ao mercado que tecnologias surjam, sejam divulgadas pela mídia como sendo as melhores e, após um certo tempo, sejam abandonadas em detrimento de outras soluções tecnológicas. Por vezes o motivo do abandono é a própria inadequação da solução oferecida ou a pouca força de mídia que uma certa solução possui. Quando isso ocorre, todos os componentes desenvolvidos sobre a tecnologia passam naturalmente a deixar de serem mantidos, e as aplicações já desenvolvidas ficam impedidas de evoluir.

Além da volatilidade das tecnologias de apoio, existe um outro problema. Normalmente, para que uma aplicação seja completamente construída, é necessária a utilização de mais de uma biblioteca de componentes, pois é muito difícil encontrar um que tenha todos os recursos de que uma aplicação precisa de maneira satisfatória. Dessa forma, a própria integração dessas bibliotecas se torna mais um problema a ser manipulado pela aplicação, aumentando ainda mais a sua complexidade.

O dinamismo da tecnologia faz com que esses problemas sejam recorrentes, e deixar a aplicação à mercê é falta grave na arquitetura da aplicação. A solução para isso é criar uma abstração de componentes sobre os quais os sistemas serão desenvolvidos que se responsabilize por esses problemas e, assim, forneça uma camada de isolamento para as aplicações. Dessa forma, quando uma biblioteca não for mais adequada, ela poderá ser substituída sem a necessidade de se refazerem as aplicações.

4.5.4 Extensibilidade dinâmica das aplicações

As aplicações devem permitir que novos recursos sejam acrescentados dinamicamente. Essa é uma característica muito importante, pois permite que sejam feitas customizações à aplicação sem que o projeto principal seja afetado. Assim, a customização não precisa necessariamente ser feita pela mesma equipe que desenvolveu a aplicação principal. Isso abre um leque enorme de possibilidades para que a aplicação se consolide junto aos usuários.

A característica de extensibilidade pode ser observada nos navegadores Web existentes hoje. Quando um certo usuário precisa ver um documento no formato PDF, por exemplo, e o navegador não possui esse recurso, este oferece um meio para que o recurso seja acrescentado. Esse tipo de funcionalidade demanda um mecanismo de plug-in que determine as regras necessárias para que recursos desenvolvidos por terceiros sejam embutidos nas aplicações.

O uso de linguagens de scritps é uma outra maneira de prover recursos de extensibilidade para as aplicações. Os scripts, por serem totalmente dinâmicos, facilitam a inclusão contínua e descentralizada de recursos, uma vez que não dependem de pré-compilação para serem executados. Os navegadores Web, novamente, são exemplos de aplicações que fazem uso extensivo desse recurso.

Não basta, porém, fornecer apenas os mecanismos de extensibilidade para que esta seja viável. É necessário que uma biblioteca de apoio esteja acessível às aplicações que utilizam esse mecanismo a fim de que essas aplicações possam operacionalizar suas funções de maneira minimalista. Nesse sentido, poderíamos dizer que a melhor biblioteca de apoio seria aquela que fosse equivalente à biblioteca abstrata utilizada pela aplicação. Devido à pluralidade de tecnologias, dificilmente esta situação pode ser totalmente alcançada.

4.6 A arquitetura InterStela

InterStela é uma arquitetura de software voltada para o desenvolvimento de sistemas adequada à realidade de aplicações em E-Gov. Sua concepção se originou da compilação de experiências obtidas no desenvolvimento de aplicações desse domínio junto ao laboratório Stela da UFSC. Essa arquitetura consiste em um navegador de aplicação, um mecanismo modular para a inclusão contínua e descentralizada de recursos, abstrações de tecnologias e uma lógica específica, que procuram minimizar a quantidade de informações de controle trocadas entre as partes componentes da arquitetura.

4.6.1 Requisitos não-funcionais

Os principais requisitos não-funcionais que irão garantir a qualidade idealizada para a arquitetura são os que se seguem.

1. Mesmo método de desenvolvimento de aplicações para as aplicações dos tipos Web e off-line.
2. Consumo minimalista dos recursos oferecidos pelos equipamentos servidores de aplicações.
3. Alto grau de independência de tecnologias de terceiros.
4. Desenvolvimento rápido de aplicações.
5. Crescimento dinâmico de funcionalidades.
6. Simplicidade no gerenciamento de versões.

O atendimento de todos esses requisitos pela arquitetura implicará em redução do custo global de produção de aplicativos de software e em aumento de produtividade. Ao contemplar todos esses requisitos, a arquitetura terá certamente alcançado seus objetivos.

4.6.2 A arquitetura

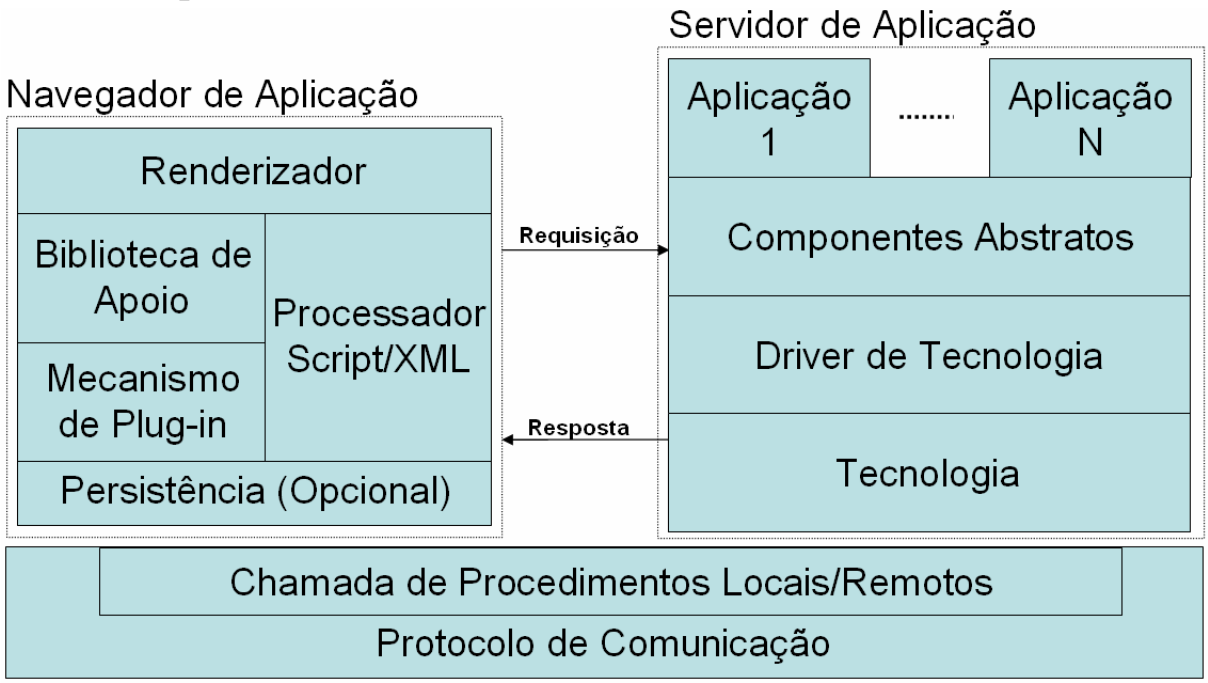

Figura 4.4 - Diagrama da arquitetura InterStela

O diagrama de blocos acima apresenta cada elemento e sua disposição na lógica da arquitetura. Para que tal lógica possa ser entendida é necessário que cada um dos elementos seja individualmente comentado.

4.6.3 Aplicação

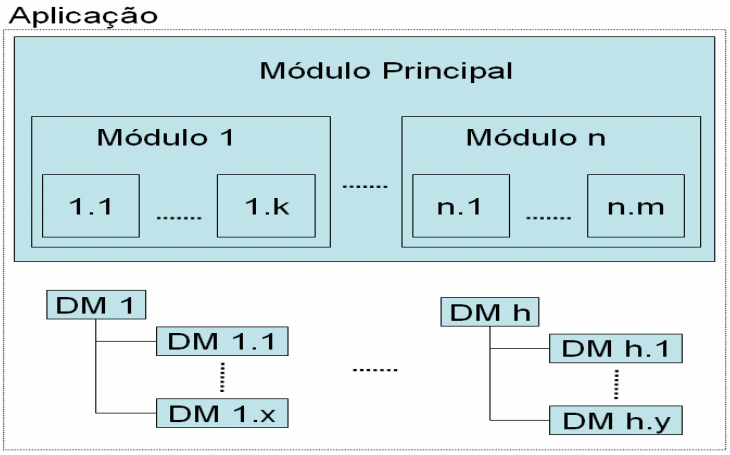

Figura 4.5 - Arquitetura da aplicação

As aplicações devem ser construídas sobre uma biblioteca de componentes abstrata que ofereça todos os recursos que sejam necessários. Uma aplicação para a arquitetura é uma composição hierárquica de componentes dinamicamente interligados por uma lógica de composição inerente à biblioteca abstrata subjacente. São definidas duas classes de componentes para as aplicações: Módulos de Visualização e Modelos de Dados (DM).

- Módulos de Visualização: responsáveis pela parte visual da aplicação. Através de uma composição hierárquica de módulos é possível descrever cada parte de um formulário para uma aplicação. Um módulo é uma composição de componentes oferecidos pela biblioteca abstrata.
- Modelos de Dados: são a representação das unidades de informação para a aplicação. Os modelos de dados abstraem da aplicação o mecanismo de persistência e fornecem a todos os módulos uma maneira comum de interação com os dados.

Sob a ótica do padrão MVC, os Modelos de Dados representam os modelos persistentes e os Módulos de Visualização representam a parte de visualização do padrão. Alguns componentes da biblioteca, que manipulam conjunto de dados específicos, podem exigir a existência de modelos específicos, mas esses casos devem ser tratados por cada componente individualmente. O controle da aplicação é executado pela tecnologia de apoio.

4.6.4 Biblioteca abstrata

A biblioteca é considerada abstrata pois deve definir para as aplicações um conjunto de componentes que ofereça um comportamento idêntico independentemente da tecnologia de apoio utilizada. Os componentes definidos devem ser uma abstração que priorize a facilidade de uso e a legibilidade da aplicação, pois estes são requisitos básicos para reduzir custos de desenvolvimento.

Como se pretende utilizar a mesma biblioteca para o desenvolvimento Web e o off-line, é imperativo que seja seguido o padrão MVC, pois este define um isolamento entre os dados que customizam a utilização e o controle dos componentes. Um grupo de componentes fica responsável pelo controle e pela visualização da aplicação, e outro grupo, que representa a persistência, responsável pela função de obtenção e manipulação dos dados. A existência do isolamento promovido pela abstração permite que a parte da aplicação responsável pelas regras de negócio passe a ser devidamente alocada em função da tecnologia utilizada, inclusive quando esta é para a Web.

No momento em que uma aplicação for executada, a biblioteca abstrata deve permitir que dinamicamente seja selecionada a tecnologia de apoio. Para que isso seja viável, um mecanismo de drivers de tecnologia deve ser estabelecido.

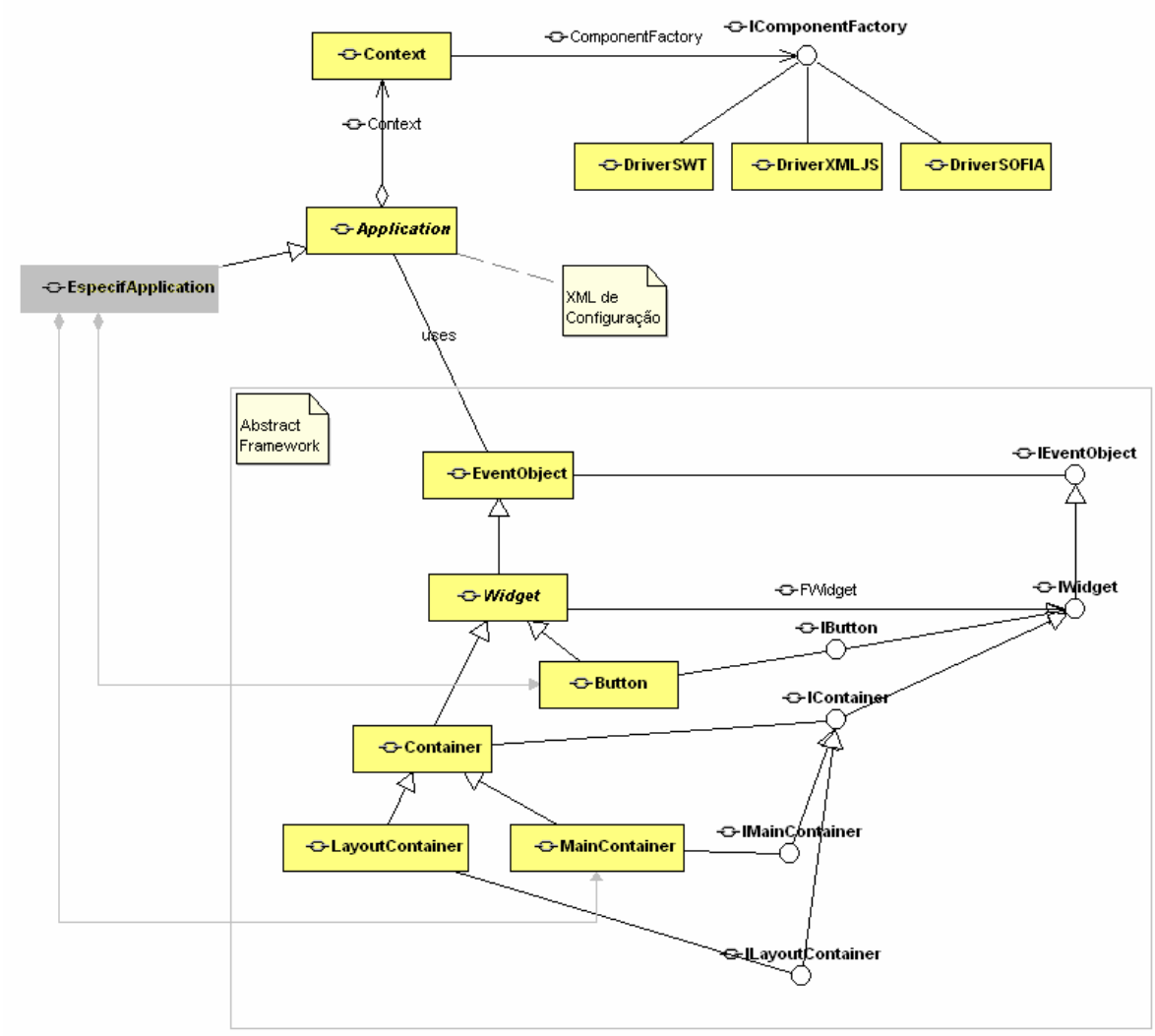

Figura 4.6 - Esboço de um diagrama de classes básicas para uma biblioteca abstrata

O diagrama de classes acima mostra uma possível estruturação de uma biblioteca abstrata que atende aos requisitos da arquitetura InterStela. As diretrizes da composição desse esboço podem ser observadas no próximo bloco da arquitetura: drivers de tecnologia e tecnologia de apoio.

4.6.5 Drivers de tecnologia e tecnologia de apoio

A operacionalização da biblioteca abstrata é feita por intermédio de drivers de tecnologias que são conectados dinamicamente no momento em que a aplicação é executada. As tecnologias de apoio utilizadas podem ser as mais variadas, desde que sejam suficientemente expressivas para representar a biblioteca.

A junção do padrão *Bridge* e o padrão *Abstract Factory* nos permite compor uma arquitetura para a biblioteca que possibilita a existência desses drivers de tecnologia.

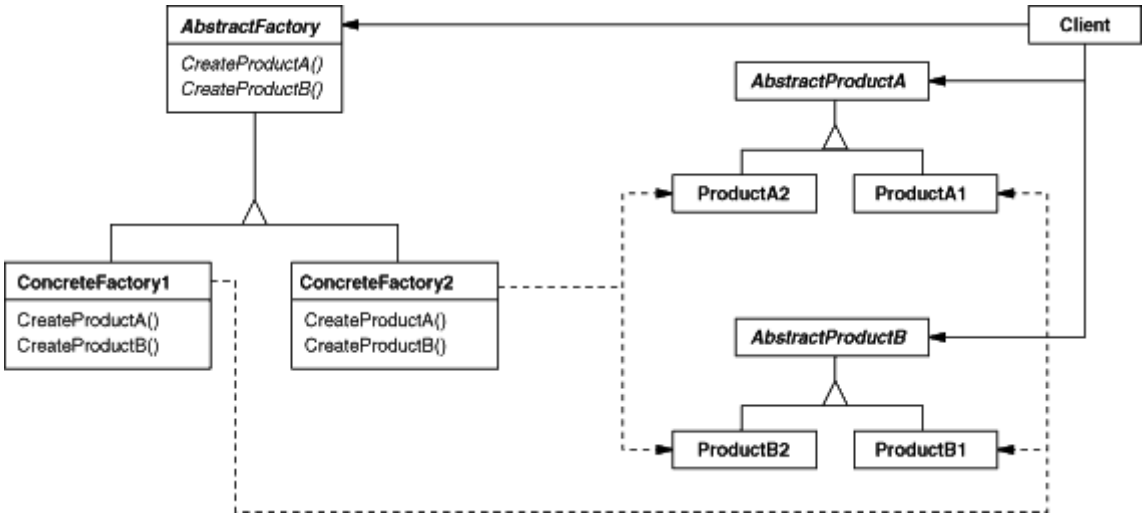

Figura 4.7 - Padrão Abstract Factory

Fonte: (GAMMA et al., 1995)

O padrão *Abstract Factory* é utilizado para criar um mecanismo de drivers para a aplicação. Cada driver é a concretização da abstração de *factory* da biblioteca abstrata sobre uma ou mais bibliotecas de uma tecnologia em particular. Assim, como a aplicação sempre invoca a mesma interface, o comportamento irá depender apenas da versão do driver que no momento estiver sendo utilizado.

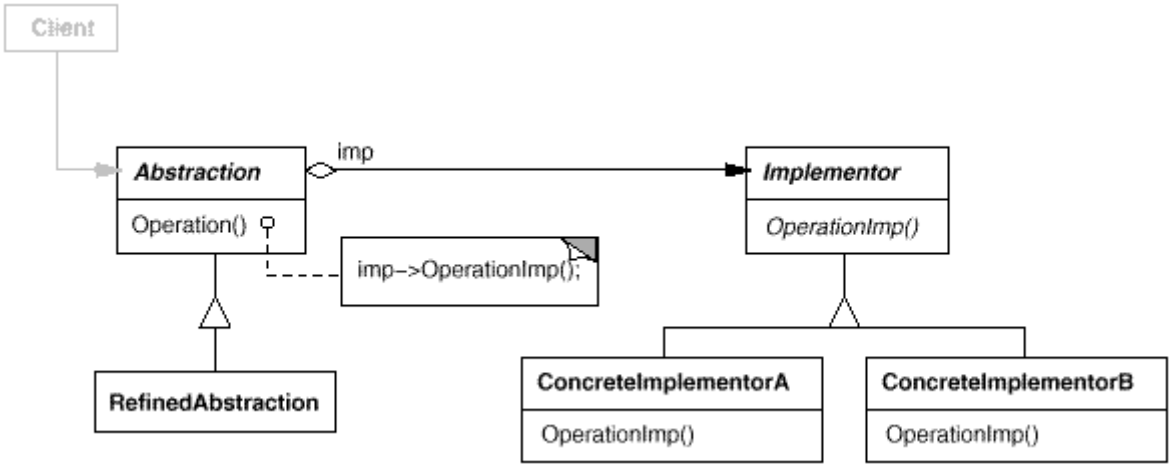

Figura 4.8 - Padrão *Bridge*

Fonte: (GAMMA et al., 1995)

O padrão *Bridge* se faz necessário para ocultar da aplicação os detalhes particulares da negociação da biblioteca com os drivers. Essa ocultação deve ser feita a fim de não misturar os problemas de implementação do driver com as soluções da biblioteca abstrata.

Dependendo de como seja implementado um driver de tecnologia, pode não ser necessária a existência do navegador de aplicação. Tecnologias como o SWT ou Swing já possuem uma lógica de navegação própria e, portanto, a simples utilização dessas tecnologias já dispõe para a aplicação um navegador. Entretanto, o uso de um navegador de aplicação que seja independente das aplicações é aconselhado, uma vez que estabelece um ambiente que favorece a redução dos custos de operacionalização. Além disso, o mecanismo de extensibilidade demanda recursos muito próximos ao de um navegador de aplicação e, portanto, implementá-lo dentro do driver não é a melhor solução.

4.6.6 Navegador de aplicação

A arquitetura define um isolamento forte entre a navegação da aplicação e a aplicação em si. O objetivo é criar um ambiente operacional para aplicações de uma plataforma de sistemas que permita que as tarefas de atualização e disponibilização de novos recursos sejam feitas de forma transparente. Para isso, é necessária a existência de um navegador voltado às necessidades das aplicações.

Toda aplicação necessita de um ambiente que a operacionalize. A operacionalização de uma aplicação é feita por meio de memória, processamento, mecanismos de persistência, mecanismos de comunicação e outros. Normalmente, o sistema operacional (SO) oferece e gerencia esses recursos. Todavia, as aplicações não podem confiar totalmente nos SOs, pois estes são construídos para um propósito mais geral e por isso permitem que sejam ultrapassados os limites desejáveis para uma plataforma de sistemas. Além disso, cada SO apresenta características particulares que são difíceis de serem geridas pelas aplicações. O navegador de aplicação deve oferecer os recursos necessários para as aplicações de maneira independente de SO e, ao mesmo tempo, gerenciar essas aplicações a fim de se certificar que estão cumprindo seus contratos de execução.

Construir um navegador de aplicação com todas essas características é complicado e demanda uma equipe de desenvolvimento experiente. Felizmente, na atualidade os navegadores apresentam muitos dos recursos necessários para representar um navegador de aplicação, e o navegador proposto pode ter suas principais funcionalidades operacionalizadas

através de um navegador Web tradicional. Os recursos disponíveis nos navegadores Web fundamentais para a arquitetura são relacionados a seguir.

- Navegação interna: permite a navegação em partes da página Web sem gerar a necessidade de recarregar toda a página.
- Linguagens de script: praticamente todos os navegadores Web apresentam interpretadores para o processamento de linguagens de script. Dessa forma, o navegador pode executar tarefas como qualquer outra aplicação.
- Biblioteca de componentes: os navegadores mais modernos apresentam uma biblioteca de componentes independente de SO que pode ser explorada a fim de oferecer os mesmos recursos disponíveis na biblioteca abstrata concebida.
- Mecanismos de plug-ins: embora de forma diferenciada entre os navegadores, esse recurso permite a inclusão dinâmica e descentralizada de recursos não previstos pelos navegadores. Com tal mecanismo é possível acrescentar os recursos que porventura estejam ausentes, a fim de implementar todos os requisitos do navegador de aplicação.
- Mecanismos de comunicação: através do protocolo HTTP os navegadores permitem a execução de procedimentos remotos (RPCs) e upload e download de arquivos sobre o meio de comunicação.

Esses cinco recursos juntos aproximam o desenvolvimento de aplicações Web das aplicações off-line tradicionais. Uma aplicação off-line pode ser perfeitamente operacionalizada por um navegador Web quando este considera o mecanismo de comunicação como sendo local. Na arquitetura InterStela a diferença entre uma aplicação off-line e uma Web é apenas a sua forma de distribuição.

4.6.6.1 Renderizador

Quando uma ligação com um módulo é estabelecida, uma série de scripts são trocados entre o renderizador e o servidor de aplicação remoto, no intuito de operacionalizar o módulo. O renderizador é quem controla a cadência desse processo, pois é ele quem interage com o usuário. A interação é feita pela disponibilização da interface gráfica e pela obtenção dos eventos gerados a partir dessa interação.

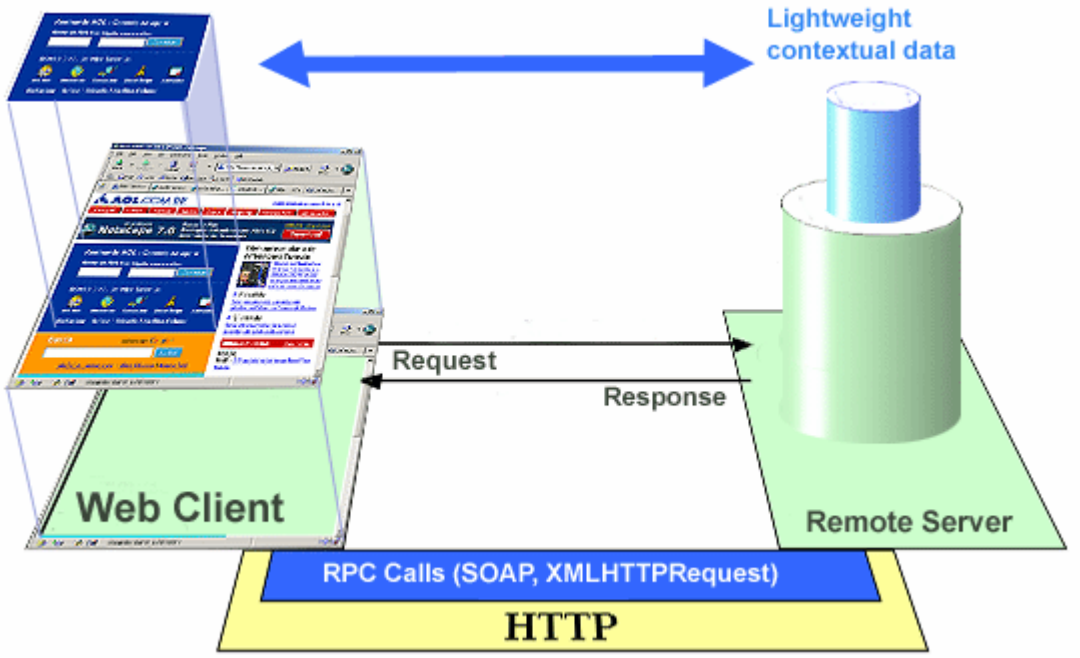

Figura 4.9 - Navegação interna em aplicações Web

Fonte: (GALLI et al., 2003)

O Renderizador é baseado no paradigma da navegação interna. Pelo paradigma da navegação interna o navegar pelas opções de um sistema provoca apenas modificações contextuais leves, não sendo necessário descartar as informações não afetadas. Essa característica reduz a necessidade de obtenção de informações sobre o próximo estado da aplicação, o que minimiza a quantidade de dados trocados com o servidor de aplicação. Essa é uma característica fundamental para a redução do consumo de recursos de processamento e banda. Para as aplicações Web essa característica é ainda mais importante, uma vez que dispõem de um recurso de banda passante normalmente muito limitado.

4.6.6.2 Mecanismos de plug-ins

Durante o processo de renderização, o renderizador considera a existência de uma biblioteca de componentes de apoio que lhe permite operar. Devido à natureza dinâmica dos sistemas, é quase impossível ter uma biblioteca que atenda a todos os requisitos de todos as classes de sistemas existentes. Como o navegador de aplicação deve atender a todo tipo de aplicação, ele deve possuir um mecanismo de plug-ins que lhe permita crescer em funcionalidades conforme o tipo de aplicação que esteja sendo executada.

Esse tipo de mecanismo é encontrado nos navegadores Web. Novos recursos podem ser acrescentados através de uma API padrão e de uma lógica de conexão dinâmica ao navegador. O problema é que cada navegador apresenta um padrão proprietário para esse recurso, e normalmente os mecanismos de atualização dinâmica não ficam transparentes para o usuário,

o que é importante para as aplicações. Como estamos trabalhando com a hipótese de se utilizarem os navegadores Web como navegador de aplicação, o recurso de plug-ins existentes nesses navegadores deve ser padronizado.

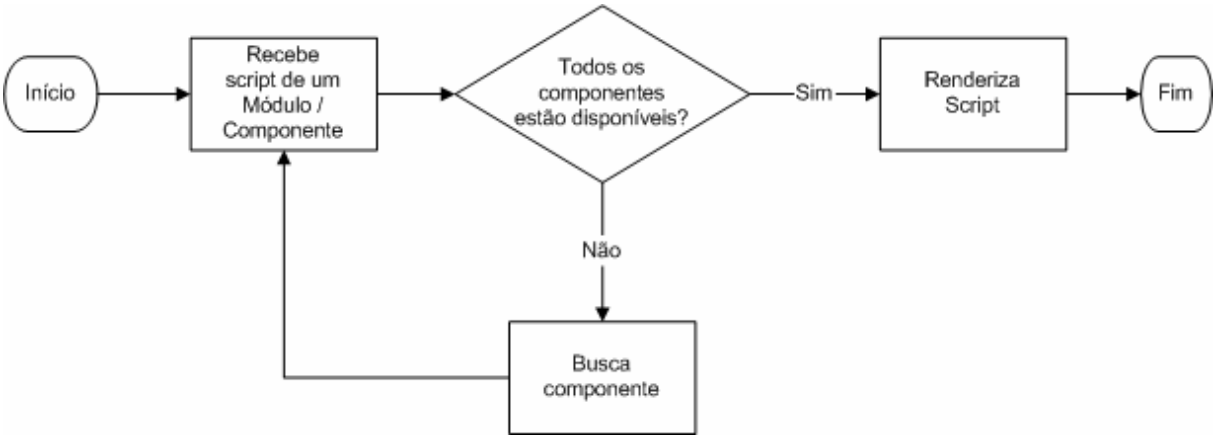

Figura 4.10 - Busca automática de componentes

O mecanismo de plug-ins deve funcionar da seguinte forma:

- quando chegar um script que faça referência a um componente que não se encontre instalado, este deve ser buscado de um repositório de componentes homologados;
- a instalação do componente deve ser feita de maneira totalmente transparente para o usuário;
- as questões de segurança devem ser avaliadas a priori, quando da homologação do componente;
- componentes só podem ser considerados homologados quando forem devidamente testados em todas as plataformas onde o navegador de aplicação estiver disponível; e
- além de os componentes poderem ser construídos nativamente, deve existir também um recurso de interpretação de scripts que permita a construção de componentes multiplataforma. Os componentes nativos, embora apresentem uma melhor performance para as aplicações, são mais trabalhosos de serem construídos e mantidos e, por essa razão, a possibilidade de se construírem componentes sobre interpretadores é fundamental para a redução de custos de operacionalização.

4.6.6.3 Biblioteca de apoio

Em última instância, o navegador de aplicação funciona como um interpretador de linguagem de script. Quando as informações são trocadas com o servidor de aplicação, o que

chega para ser interpretado são scripts customizados para uma certa aplicação. Esses scripts são altamente dependentes da biblioteca de apoio existente no navegador. Por isso, quanto mais a biblioteca de apoio se assemelhar à biblioteca abstrata, menor serão o tamanho e a complexidade dos scripts trocados.

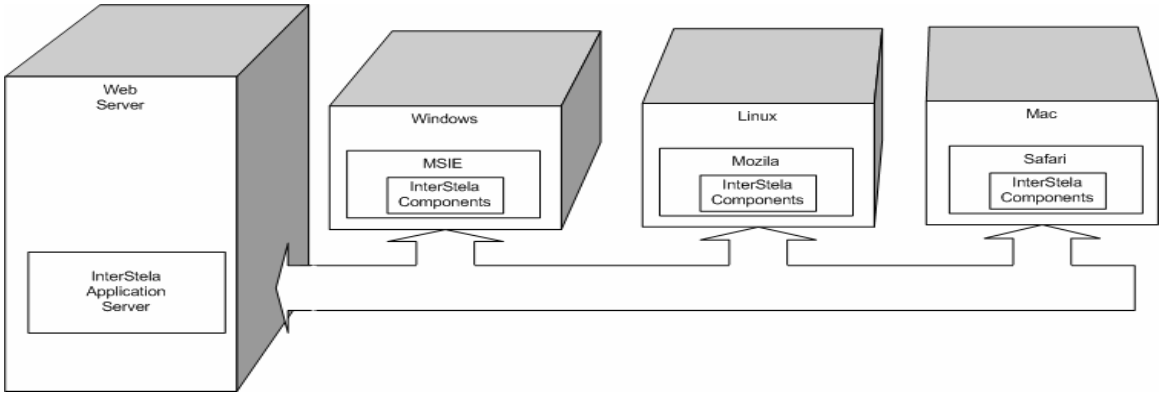

Figura 4.11 - Biblioteca de componentes

A biblioteca de apoio pode ser disponibilizada para os navegadores Web de duas formas, as quais são descritas a seguir.

- Biblioteca Nativa (Estática): pode fazer uso do mecanismo de plug-in e por isso pode oferecer recursos dos mais diversos. Cada componente é construído sobre uma biblioteca de tecnologia em particular, e suas interfaces são disponibilizadas na linguagem de script do navegador. Este tipo de biblioteca utiliza o mínimo de recursos de processamento da máquina cliente.
- Biblioteca Script (Dinâmica): existe uma certa dificuldade de se representarem todos os componentes necessários sobre os recursos limitados dos navegadores. Os navegadores Web mais modernos tendem a ter mais recursos e, por conseguinte, facilitam a construção da biblioteca. Este tipo de biblioteca exige mais recursos de processamento da máquina cliente.

Embora a biblioteca nativa tenha mais atributos desejáveis, ela demanda que o plug-in seja pré-instalado. A obtenção e a instalação desse plug-in podem ser um problema que não se deseja assumir. Nesses casos, a biblioteca de script é mais indicada, embora algumas limitações tenham de ser impostas. Em navegadores Web mais antigos a falta de recursos pode exigir a instalação da biblioteca estática.

4.6.6.4 Processador de script e máquina virtual

O dinamismo exigido pelo navegador de aplicação pode ser mais facilmente alcançado através de linguagens de script. Por essa razão, a arquitetura prevê que o navegador de aplicação ofereça tal recurso para as aplicações.

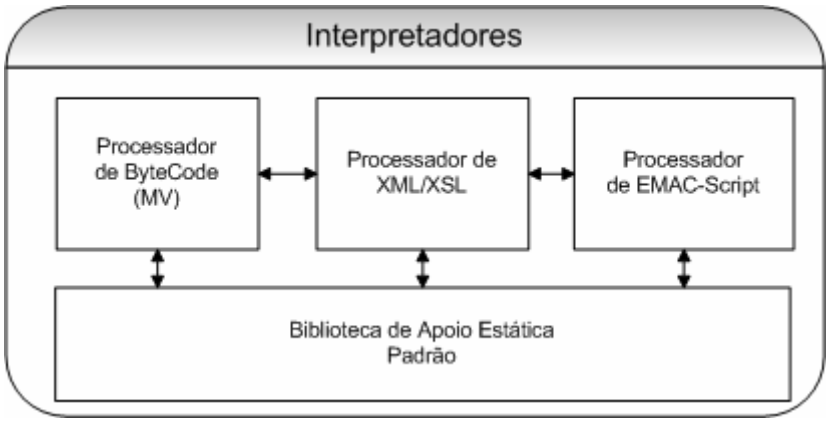

Figura 4.12 – Interpretadores previstos pela arquitetura

A arquitetura define três classes de scripts que devem ser processadas pelo navegador de aplicação, as quais são descritas a seguir.

- *ByteCode*: a linguagem de *ByteCode* é uma linguagem primitiva que permite a utilização de scripts binários multiplataforma, os quais podem ser gerados a partir de qualquer linguagem por intermédio de compiladores apropriados. O objetivo de disponibilizar um processamento de *ByteCode* é o de fornecer uma forma intermediária entre os componentes nativos e os componentes interpretados, minimizando assim a necessidade de se implementarem componentes nativos. A abordagem de interpretadores de ByteCode é bastante utilizada e apresenta como principal vantagem sobre outros tipos de interpretadores a capacidade de JIT (Just-In-Time Compilation), o que aumenta a performance percebida dos componentes.
- XML/XSL: as linguagens de marcação, quando utilizadas para o propósito para o qual foram projetadas, apresentam a propriedade de aumentar a legibilidade e facilitar a compreensão. Por essa razão a arquitetura define que os objetos da biblioteca de apoio possam também ser instanciados a partir de uma descrição XML. Essa abordagem é utilizada em projetos como XUL, XAML, FLEX e outros.
- EMAC-Script: as linguagens de scripts tradicionais, como a EMAC-Script escolhida, permitem criar um ambiente totalmente dinâmico. Esse dinamismo é muito importante para determinados tipos de aplicação e facilita a tarefa de

prototipação de novos sistemas. A linguagem de script deve poder trabalhar de maneira complementar à linguagem de marcação, aos moldes do que é feito hoje nos navegadores Web.

Uma biblioteca de apoio nativa que seja comum a todas as classes de aplicações deve estar disponível para essas linguagens de script. Essa biblioteca estática oferecerá os recursos básicos fundamentais para que os scripts possam compor seus comportamentos. Quando da suposição do navegador Web como navegador de aplicação, o papel da biblioteca nativa é feito pelos componentes HTML disponíveis nesses navegadores. Se o navegador de aplicação for construído, a biblioteca nativa deve ser o mais próximo possível da biblioteca abstrata.

4.6.6.5 Persistência

Embora não fundamental, este mecanismo ajuda a minimizar o acesso ao servidor de aplicação. A arquitetura prevê os usos descritos a seguir para este mecanismo.

- Tabelas basilares: as aplicações normalmente precisam consultar um conjunto de tabelas basilares, tais como unidades da federação, CEP, instituições de pesquisa, etc. Esse tipo de informação tende a mudar muito raramente. Logo, fazer a pesquisa remota dessas informações é desnecessário. Esse tipo de informação poderia ser *cacheado* no navegador, e os componentes apropriados poderiam fazer suas consultas sobre as informações existentes no mecanismo de persistência do navegador.

- Sessão das aplicações: outro uso interessante para o mecanismo de persistência é a possibilidade de manter o estado da sessão da aplicação. Essas sessões poderiam ser posteriormente recuperadas mesmo quando tivessem sido abruptamente interrompidas. Além disso, esse comportamento facilita a utilização do modelo *briefcase*, já que a manutenção do estado da aplicação pode ser feita no lado cliente.

- Disco virtual: a simulação de um disco para a aplicação pode ser muito útil no que diz respeito à simplificação do desenvolvimento das aplicações. Com um disco virtual, as aplicações podem guardar seus arquivos de configuração, imagens e outros recursos diretamente na máquina do usuário, sem que a segurança das aplicações seja afetada.

É importante ressaltar que o acesso ao mecanismo de persistência tem que ser previamente negociado no contrato da aplicação, pois se trata de um recurso que, quando usado de maneira mal-intencionada, pode exaurir o espaço em disco da máquina cliente.

4.6.6.6 Chamada de procedimentos

Como vimos, a arquitetura divide o problema das aplicações em dois grandes blocos: Navegador de Aplicação e Servidor de Aplicação. Esses dois blocos se comunicam através de chamada de procedimentos, a qual pode ser local ou remota, dependendo apenas do modo de distribuição escolhido para a aplicação.

As aplicações que são distribuídas para serem off-line possuem o servidor de aplicação embutido no navegador de aplicação e, portanto, não necessitam de um protocolo de comunicação. A interface de acesso aos recursos do servidor é direta para esses casos.

Já as aplicações que funcionam sobre a Web devem utilizar um protocolo de comunicação nos moldes dos Serviços Web (SOAP, XMP-RPC, etc.). Do ponto de vista do navegador de aplicação, a localização do servidor de aplicação é totalmente irrelevante. Isso se deve ao fato de que todos os recursos devem ser identificados através de uma URL (Uniform Resource Locator).

4.6.7 Validando os requisitos não-funcionais

Na definição da arquitetura foram levantados alguns requisitos não-funcionais que garantiriam a qualidade desejada para a arquitetura e que viabilizariam o alcance dos objetivos da presente proposta. Cabe agora verificar como esses requisitos foram atendidos pela arquitetura.

1. **Mesmo método de desenvolvimento de aplicações para as aplicações dos tipos Web e off-line**: através do uso de uma biblioteca abstrata as aplicações ficam totalmente isoladas da tecnologia de apoio. Essa situação permite que um driver adequado disponibilize a aplicação sobre um navegador de aplicação bem como diretamente sobre o sistema operacional, sem que seja necessário fazer qualquer alteração nas aplicações.
2. **Consumo minimalista dos recursos oferecidos pelos equipamentos servidores de aplicações**: este atributo é primariamente importante para as aplicações Web. Apresenta as seguintes características:
    - o uso do modelo *briefcase*: estabelece que o estado da aplicação deve ser mantido no cliente, o que evita o acesso ao servidor para buscar informações que já foram trazidas;

- o mecanismo de navegação interna: por provocar modificações pontuais na interface, reduz a quantidade de scripts trocados com o servidor na composição de uma tela; e
- o mecanismo de extensibilidade: cria uma biblioteca de componentes no navegador, os quais só são novamente trazidos quando sofrem alterações de versão.

3. **Alto grau de independência de tecnologias de terceiros**: a construção das aplicações sobre a biblioteca abstrata isola totalmente a aplicação da tecnologia utilizada.
4. **Desenvolvimento rápido de aplicações**: a arquitetura define que a biblioteca abstrata deve ser concebida de maneira a tornar direta e simples a composição de aplicações. Isso significa que componentes de comportamentos mais elaborados devem fazer parte da concepção da biblioteca. O uso contínuo dessa técnica tende a criar um acervo de componentes de fácil uso que exponencialmente acelera o desenvolvimento.
5. **Crescimento dinâmico de funcionalidades**: o mecanismo de extensibilidade permite que dinamicamente novas funcionalidades sejam acrescidas à biblioteca de componentes abstrata sem que seja necessário mudar a versão do navegador de aplicação.
6. **Simplicidade no gerenciamento de versões**: a arquitetura define que cada componente estabeleça suas dependências de outros componentes de maneira total. Assim, no momento em que um componente é alterado, todos os componentes que dependem dele, quando usados, são obrigatoriamente atualizados. A não necessidade de instalação da aplicação no cliente, juntamente com a regra de dependência versionada entre componentes, simplifica grandemente o gerenciamento de versões.

4.7 Uso do InterStela em aplicações existentes

Como vimos, a arquitetura InterStela traz uma série de vantagens para as aplicações em SI, tanto Web quanto off-line. Embora os atuais SI não tenham sido projetados sobre a arquitetura, é possível fazer com que utilizem uma versão simplificada da arquitetura e com isso usufruam

um pouco das vantagens oferecidas. Neste tópico serão mostradas duas possibilidades de uso adaptado da arquitetura: uma para uso em sistemas off-line e outra para uso em sistemas Web.

4.7.1 InterStela em aplicações off-line

Os sistemas off-line normalmente são desenvolvidos monoliticamente e, por isso, levam consigo todos os recursos que necessitam para operar. O uso do InterStela em tais sistemas consiste explorar esses recursos e disponibilizá-los de maneira dinâmica para os módulos.

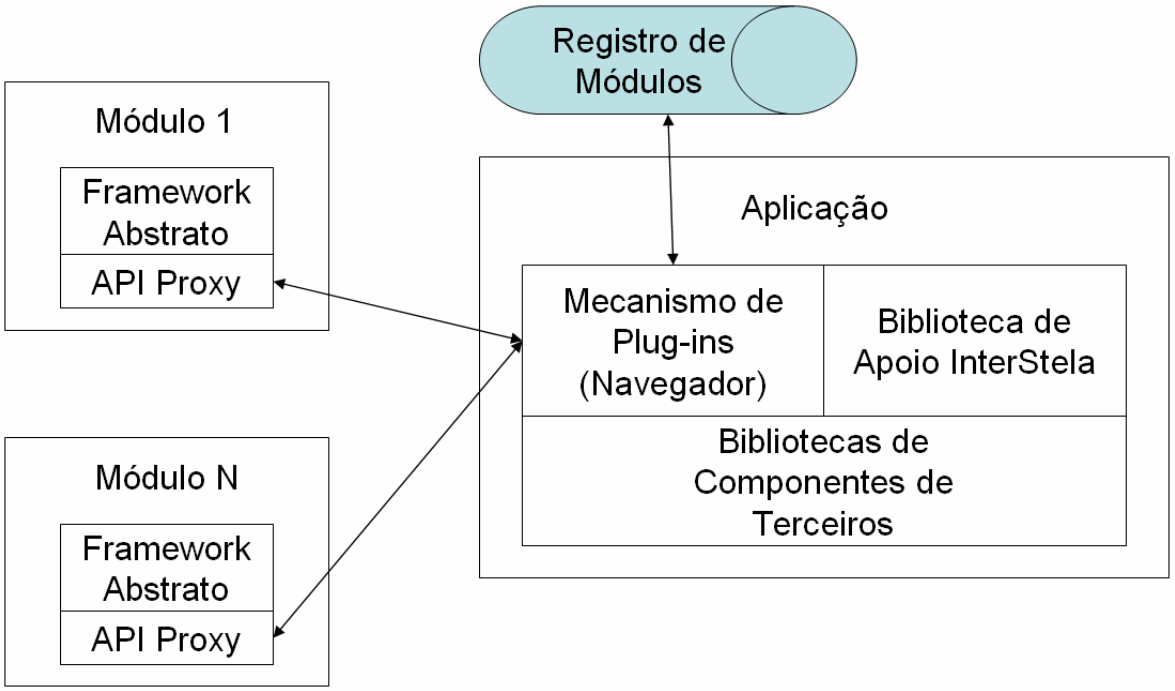

Figura 4.13 - Arquitetura InterStela para aplicações legadas off-line

Para que o InterStela fique disponível para a aplicação é necessário que se faça uma leve reconfiguração na estrutura da aplicação de modo a disponibilizar os elementos fundamentais da arquitetura. As modificações necessárias encontram-se relacionadas a seguir.

- Criar mecanismo de plug-in: permitir que módulos sejam dinamicamente ligados à aplicação. O mecanismo de plug-in deverá funcionar como uma espécie de Navegador de Aplicação, ou seja, deve oferecer aos módulos um ambiente sobre o qual possam operar. Como as aplicações já possuem uma interface gráfica com um esquema próprio de navegação, é necessário que seja estabelecida uma lógica que permita indicar onde serão colocados os enlaces para os módulos.
- Compor uma biblioteca de apoio: construir, sobre as bibliotecas de terceiros, uma biblioteca de apoio que deverá ser disponibilizada para os módulos por intermédio

de uma API Proxy. Essa API deve possuir a propriedade de baixo acoplamento e garantir que futuras versões do aplicativo principal não suprimam funcionalidades.

- Criar framework abstrato para os módulos: os módulos não devem fazer uso direto da API, devendo ser construídos sobre um framework abstrato. O framework é considerado abstrato porque ele de fato não implementa as funcionalidades mas sim repassa as solicitações feitas sobre seus componentes diretamente para a API Proxy que, nesse caso, faz o papel do driver de tecnologia. Quanto mais completo em funcionalidades for o framework abstrato, menor será o tamanho físico dos módulos.

Essa abordagem foi aplicada nos sistemas off-line da Plataforma Lattes do CNPq. No capítulo 5 será mostrada uma versão contextualizada do InterStela, denominada InterLattes, e os benefícios que a arquitetura trouxe para a Plataforma Lattes.

4.7.2 InterStela em aplicações Web

As aplicações Web admitem muitas arquiteturas de desenvolvimento e, por isso, definir uma solução para cada uma das arquiteturas existentes vai além do escopo desta dissertação. Contudo, a maior parte das aplicações Web foram desenvolvidas sobre o paradigma de terminais “burros”, e as aplicações de E-Gov não fugiram à regra. Assim, ao se propor uma solução do uso do InterStela para esse contexto estaremos atendendo a maioria.

As aplicações Web, por serem totalmente baseadas em um navegador, já possuem um mecanismo para adicionar dinamicamente novos módulos, o que irá simplificar o mecanismo de plug-in. Entretanto, a composição das páginas é feita diretamente sobre tecnologias de terceiros, ou mesmo diretamente sobre os scripts específicos dos navegadores (HTML, XHTML, JavaScript, etc.). O InterStela em tais aplicações consiste em se criar um framework abstrato e fornecer um mecanismo para que essas aplicações possam fazer os devidos enlaces.

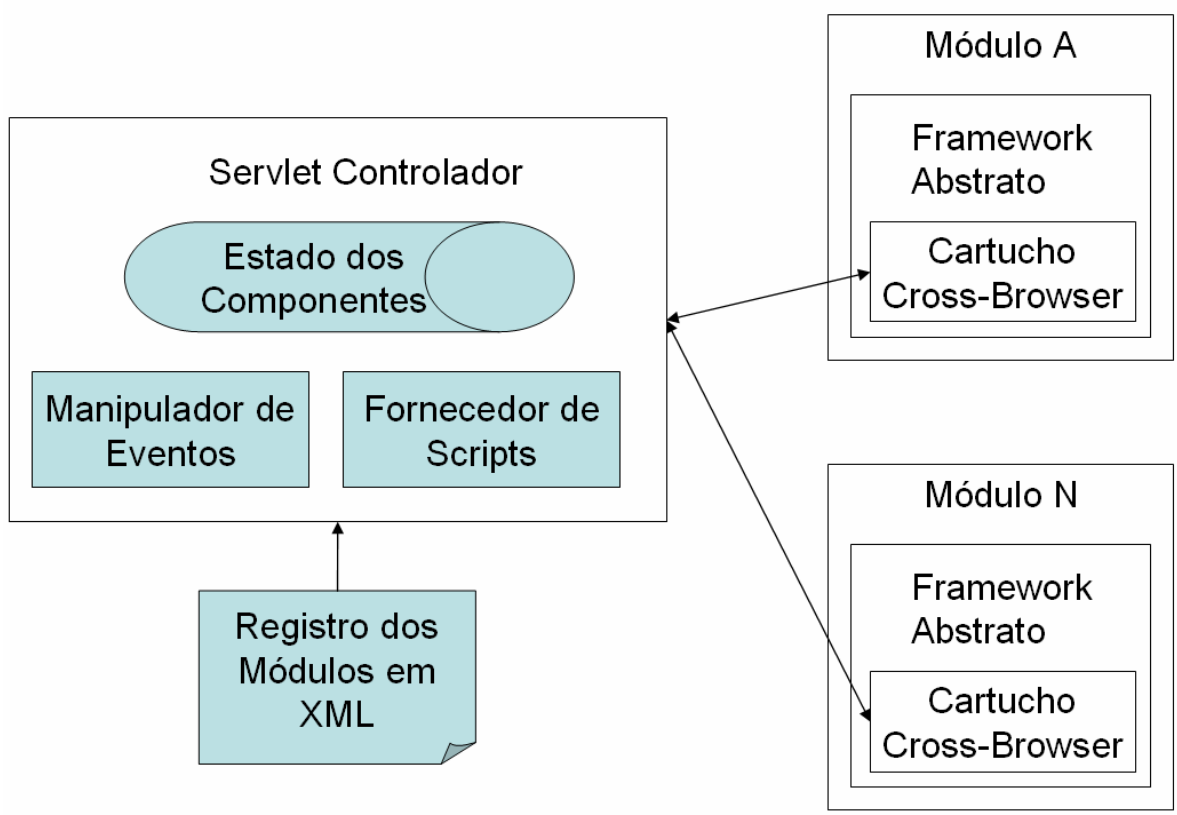

Figura 4.14 - Arquitetura InterStela para aplicações legadas Web

Para que o InterStela fique disponível para as aplicações Web é necessário que sejam criados os elementos da arquitetura descritos a seguir.

- Framework abstrato: criar um framework que isole totalmente os módulos da tecnologia de apoio. É necessário fazer com que este framework utilize drivers e que seja desenvolvido um driver para cada tipo de browser suportado pela aplicação.
- Mecanismo de plug-in: este recurso deve ser implementado através de um *servlet* que irá controlar o acesso aos componentes instanciados pelos módulos bem como o estado desses componentes. Novos módulos podem ser acrescentados ao sistema desde que devidamente registrados em um arquivo de configuração presente no Servidor Web. O enlace para os módulos deve ser fornecido dinamicamente por meio de uma página de *links* Web também fornecida pelo controlador.

A conexão das aplicações Web legadas aos módulos fornecidos pelo InterStela deve ser feita através do uso do mecanismo de navegação interna dos browsers. A aplicação legada deve criar uma página especial onde seja apresentada a página de enlaces de módulos. Essa página deve explorar o mecanismo de navegação interna dos browsers.

4.8 Considerações finais

Este capítulo apresentou uma proposta de arquitetura de software voltada aos requisitos das plataformas de Governo Eletrônico. Os requisitos foram levantados, identificando-se os problemas nos sistemas atuais e verificando-se como esses problemas poderiam ser solucionados.

Observou-se também que a arquitetura deveria estar fundamentada em princípios e padrões consolidados, os quais deveriam ser devidamente escolhidos. Os padrões e princípios escolhidos foram então justificados em função dos requisitos previamente levantados.

Então, apresentou-se a arquitetura de software InterStela. A arquitetura elaborada definiu quais elementos são necessários na construção de sistemas de informação de E-Gov de maneira a garantir que todos os requisitos não-funcionais identificados fossem atendidos. A arquitetura então foi descrita detalhadamente, ressaltando-se o fluxo dos dados entre os elementos bem como o comportamento esperado de cada elemento.

Finalmente, discutiu-se como essa arquitetura poderia ser aplicada na atual conjuntura das aplicações de software. Foram elaboradas arquiteturas simplificadas que poderiam viabilizar o uso do InterStela em tais aplicações, mesmo sem ter sido inicialmente projetadas para isso.

No próximo capítulo apresentaremos o uso da arquitetura proposta no contexto da Plataforma Lattes e procuraremos demonstrar os ganhos promovidos por sua utilização.

5 InterLattes

A capítulo anterior introduziu a arquitetura InterStela como sendo uma arquitetura voltada aos requisitos de aplicações de E-Gov. Na concepção da arquitetura foram levantadas soluções para cada um dos problemas identificados na elaboração e construção de aplicações desse domínio. Cabe agora apresentar uma aplicação da arquitetura proposta de modo a se comprovarem os benefícios esperados.

Este capítulo irá apresentar uma customização da arquitetura InterStela no contexto dos sistemas de captura off-line da Plataforma Lates do CNPq. Para isso, iremos inicialmente apresentar a Plataforma Lattes para justificar o porquê da escolha e, em seguida, apresentaremos detalhadamente todos os artefatos criados pela arquitetura. Ao final do capítulo mostraremos que o InterLattes é uma realidade na atual versão da Plataforma.

5.1 Plataforma Lattes

"... uma verdadeira vitrine e memória da TCI brasileira." (JC, 2004).

A Plataforma Lattes é uma integração de sistemas, fundada na atenção aos interesses de pesquisadores, agências governamentais e instituições de ensino e pesquisa, que mudou a face da gestão de informações sobre TCI (GALIZA, 2004). Foi concebida para integrar os sistemas de informação das agências federais, racionalizando o processo de gestão de TCI (PORTAL LATTES, 2004). Atualmente é composta de um conjunto de sistemas de informação, bases de dados e portais Web voltados para a gestão de TCI (GALIZA, 2004).

Desde o lançamento, em agosto de 1999, a Plataforma Lattes mantém um crescimento contínuo da sua base de dados e comprova, a partir daí, sua maturidade no cenário nacional de TCI. Ela é o mais completo mapeamento de uma comunidade científico-tecnológica existente no mundo. Já recebeu seis milhões de acessos de 67 países desde 2002 quando passou a ser internacional. Ela armazena mais de 350 mil currículos de pesquisadores, docentes, estudantes, gestores e profissionais das mais diversas áreas do conhecimento, e hospeda ainda 20 mil grupos de pesquisa em cerca de 270 instituições de pesquisa (JORNAL DA CIÊNCIA, 2004).

A Plataforma Lattes tem chamado a atenção de outros países e atualmente serve de modelo a uma rede de cooperação internacional conhecida por Rede ScienTI . Esta rede foi idealizada para padronizar e compartilhar informações e metodologias de gestão sobre TCI da América Latina, do Caribe e dos países da Península Ibérica. Outros países também estão sendo

beneficiados, tais como Argentina, Equador, Panamá, Venezuela, Paraguai e Portugal. (JORNAL DA CIÊNCIA, 2004).

A Plataforma Lattes recentemente venceu o "Prêmio E-gov 2004" do Governo Federal, destinado aos melhores serviços oferecidos por órgãos governamentais através da Internet, na categoria Governo para Cidadão (GALIZA, 2004).

5.1.1 Visão geral

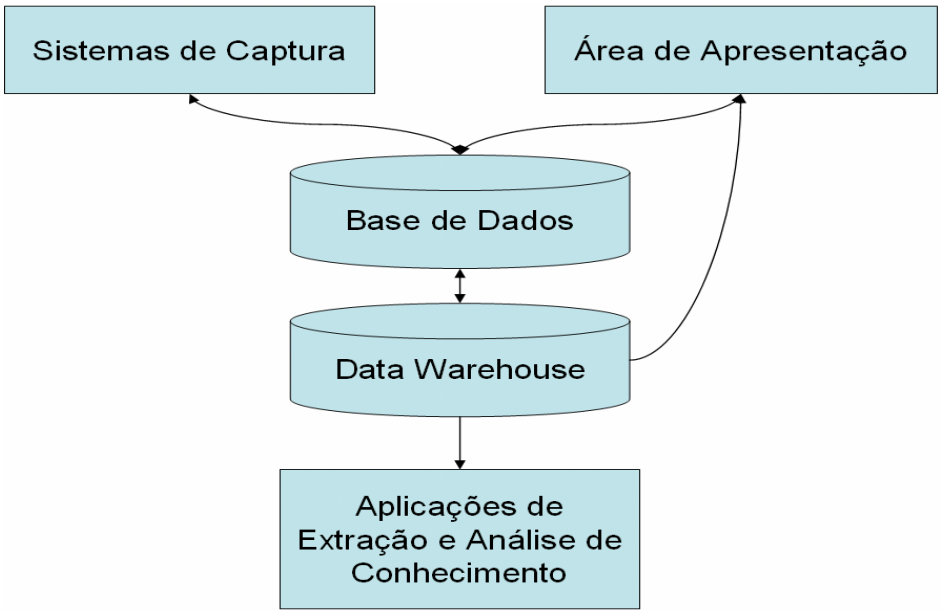

Figura 5.1 - Visão geral da Plataforma Lattes

A Plataforma Lattes está dividida em:

- sistemas de captura: sistema de currículo e de grupos de pesquisa no Brasil são os atuais sistemas de captura, os quais são os responsáveis por coletar as informações para as bases de dados do CNPq e também por estruturar e fornecer informações para os pesquisadores;
- bases operacionais: repositório relacional de tabelas de banco de dados que armazenam todas as informações geradas pelos sistemas de captura;
- data warehouse: consolidações dos dados das bases operacionais em modelos de consulta para atendimento dos requisitos da área de apresentação da Plataforma e para as ferramentas de extração de conhecimento;
- área de apresentação: representada pelo Portal Web, é o ponto de acesso comum aos pesquisadores e é acessível pelo site do CNPq. Através do portal é possível

fazer consultas, operar sobre as versões do sistema Web, obter os instaladores dos sistemas off-line e navegar por informações de propósito gerais.

- aplicações de análise e extração de conhecimento: a Plataforma Lattes faz mais do que simplesmente coletar informações, ela oferece também soluções para a análise e extração de conhecimento. Os atuais instrumentos presentes na Plataforma para esse fim são:
    - **Link Analysis**: possui como objetivo a análise das inter-relações entre atores do processo de TCI de modo a permitir que os tomadores de decisão estejam aptos a inspecionar, de forma dinâmica, a forma com que esses atores interagem em cada área de construção do processo científico e tecnológico do País (LINLK ANALYSIS, 2001);
    - **Lattes Mining**: é uma iniciativa para a geração de instrumentos de caráter investigativo que fazem uso das pesquisas de pós-graduação na área. Estabelece ações de pesquisa e desenvolvimento de métodos e instrumentos de investigação, avaliação e extração de conhecimento a partir das bases de dados da Plataforma (LATTES MINING, 2001);
    - **Redes GP**: é um projeto contínuo de definição, concepção, projeto, prototipação e implementação de instrumentos de análises de redes de pesquisa, que toma por base as informações do Diretório de Grupos de Pesquisa no Brasil (REDES GP, 2001).

As informações coletadas pelos sistemas de captura são armazenadas nas bases operacionais do CNPq, as quais são então ajustadas para bases de Data Warehouse em que podem ser utilizadas pelo Portal Web e pelas aplicações de extração e análise de conhecimento.

5.2 O Ambiente InterLattes

O ambiente InterLattes foi concebido para permitir a inclusão dinâmica, contínua e descentralizada de recursos na Plataforma Lattes. A idéia é facilitar o processo de inserção de novos recursos na Plataforma sem que seja necessário gerar novas versões. Através dele podem-se construir módulos que se acoplam dinamicamente aos sistemas de informação da Plataforma.

5.2.1 Por que a Plataforma Lattes?

Como podemos ver, a Plataforma Lattes representa uma aplicação de governo eletrônico com ampla abrangência, tanto nacional quanto internacional. Por essa razão, a aplicação da arquitetura proposta no contexto da Plataforma representa mais do que uma simples validação da arquitetura, é uma contribuição técnica de grande valor para a Plataforma e para a comunidade de TCI em geral.

Outro fator importante da escolha da Plataforma Lattes é o fato de ela possuir Sistemas de Informação que estão dentro da área de abrangência da arquitetura InterStela e, portanto, adequados a servirem de ambiente de validação da proposta arquitetural sugerida por esta dissertação. Além disso, por trabalharmos para o Grupo Stela, responsável pela Plataforma, temos pleno acesso aos códigos-fonte da Plataforma, quesito este fundamental. Assim, a Plataforma Lattes é o campo de testes mais apropriado para se validar a arquitetura proposta por esta dissertação.

5.2.2 Os sistemas Lattes atendidos

Os sistemas da Plataforma atendidos pelo InterLattes são: (a) Currículo Lattes 1.4.0 ou superior, versão off-line; e (b) Grupo de Pesquisa 5.5 ou superior, versão off-line. Esses sistemas apresentam as seguintes responsabilidades:

- Sistema de Currículos Lattes (CV-Lattes): é o sistema de informação responsável pela coleta das informações que servem de apoio na descrição da pesquisa do indivíduo no País. Fazem uso desse sistema pesquisadores, estudantes, gestores, profissionais e demais atores do sistema nacional de Ciência, Tecnologia e Inovação (PORTAL LATTES, 2004). Este sistema fornece subsídios ao incremento e à manutenção da base de dados curriculares do CNPq (GONÇALVES, 2000), onde as informações obtidas são aplicadas na avaliação da competência de candidatos à obtenção de bolsas e auxílios, na seleção de consultores, de membros de comitês e de grupos assessores, e na avaliação da pesquisa e da pós-graduação brasileiras (PORTAL LATTES, 2004); e
- Diretório de Grupos de Pesquisa no Brasil (GP-Lattes): o Diretório de Grupos de Pesquisa no Brasil é um projeto desenvolvido no CNPq desde 1992. Constitui-se em bases de dados (censitárias e correntes) que contêm informações sobre os grupos de pesquisa em atividade no País (PORTAL LATTES, 2004). Desde 1997 a Plataforma Lattes assumiu os sistemas de informação vinculados a esse projeto e

> passou a coletar e manter as informações em suas bases. Essas informações coletadas dizem respeito aos recursos humanos constituintes dos grupos, às linhas de pesquisa em andamento, às especialidades do conhecimento, aos setores de aplicação envolvidos, à produção científica e tecnológica, e aos padrões de interação com o setor produtivo (PORTAL LATTES, 2004).

Para esses sistemas, podem ser desenvolvidos módulos InterLattes independentes do projeto principal. Essa independência diz respeito inclusive a terceiros, como veremos mais adiante.

5.3 Especificação da ambiente InterLattes

O InterLattes é uma customização da arquitetura InterStela no contexto dos sistemas de captura da Plataforma Lattes. Ele consiste em um mecanismo modular para a inclusão contínua e descentralizada de recursos, abstração das tecnologias utilizadas na Plataforma e um framework de desenvolvimento de módulos.

Abaixo descrevemos os elementos que compõem a estruturam do ambiente bem como os relacionamentos e o fluxo de informações entre eles.

5.3.1 A arquitetura

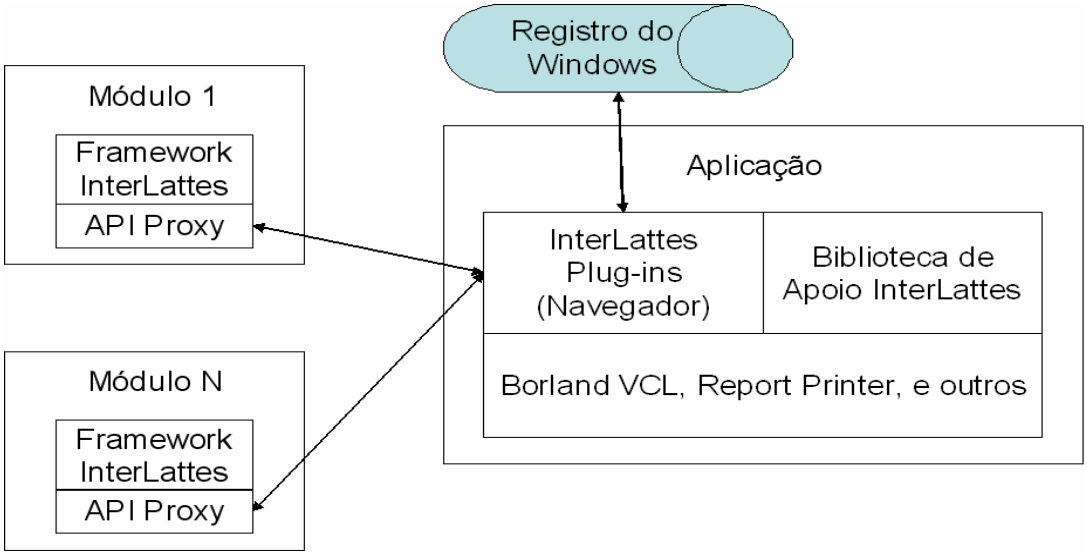

Figura 5.2 - Diagrama da arquitetura InterLattes

O diagrama de blocos acima apresenta cada elemento e sua disposição na lógica da arquitetura. Ele é idêntico ao apresentado no Capítulo 4, exceto pela customização feita à realidade dos sistemas off-line escolhidos. Abaixo detalhamos a arquitetura.

5.3.2 Mecanismo de plug-ins

O mecanismo de plug-ins é o responsável pelo ambiente InterLattes na aplicação. É ele quem gerencia todo o processo de comunicação com os módulos, faz os enlaces de módulos dinamicamente e oferece os recursos de operacionalização aos módulos. Sua concepção e especificação encontram-se descritas a seguir.

5.3.2.1 Escolha da tecnologia adequada

O desenvolvimento de módulos que se acoplam dinamicamente ao sistema exige a utilização de recursos de ligação dinâmica. Esse tipo de recurso encontra-se disponível em quase todos os sistemas operacionais existentes na atualidade, alguns deles chegam até a oferecer mais do que um recurso desse tipo. No entanto, a maior parte desses mecanismos são de baixo nível e apresentam características específicas ao sistema operacional. A escolha da tecnologia adequada é importante para a vida futura do ambiente InterLattes e deve ser feita dentro da realidade das aplicações desenvolvidas para a Plataforma, que deve estar relacionada ao Windows e Linux, já que existem versões das aplicações off-line da Plataforma para esses dois sistemas operacionais.

Para se atingir esse objetivo, foram estudadas as tecnologias mais difundidas de integração dinâmica de módulos existentes nos sistemas operacionais nos quais a Plataforma opera. A seguir apresentamos sucintamente as tecnologias avaliadas.

- ActiveX: é uma tecnologia de ligação dinâmica de módulos e utilizada pelo Internet Explorer. Existem várias ferramentas no Windows que dão suporte direto a esta tecnologia, inclusive o Delphi. Embora esta tecnologia apresente atributos de qualidade desejáveis, no Linux ela é proprietária e poderia onerar muito o tamanho das aplicações off-line da Plataforma. Contudo, alguns elementos dessa tecnologia podem ser utilizados sem que essa dependência seja caracterizada.

- Plug-ins: é uma tecnologia que não possui uma especificação *a priori*. Trata-se de uma forma de desenvolvimento de módulos em que a inclusão de suas funcionalidades ocorre dinamicamente mediante um contrato de API estabelecido pelas regras da aplicação. Assim, cada produto que utiliza esta tecnologia oferece uma solução própria. Os navegadores Web Netscape, Mozila, Opera, etc. que utilizam esta tecnologia estão presentes em várias plataformas S.O./hardware, inclusive o Linux.

Como podemos ver, a criação de uma tecnologia de *plug-ins* própria é a melhor saída, haja vista a realidade tecnológica da Plataforma. Além disso, uma tecnologia de *plu-gins* própria não é complicada e simplifica os requisitos necessários para o desenvolvimento de módulos.

5.3.2.2 O motor de plug-ins

Para que uma aplicação seja compatível com o ambiente InterLattes, precisa agregar um motor de *plug-ins*. Uma vez que o motor esteja devidamente configurado na aplicação, esta terá à sua disposição um ambiente de módulos capaz de localizar e executar módulos registrados para a aplicação.

O motor é uma biblioteca de classes e funções que faz uso de uma tecnologia de *plug-ins* especificamente criada para atender à lógica modular proposta na arquitetura InterStela. Do ponto de vista dos módulos, o motor é totalmente imperceptível; já para a aplicação ele funciona como um componente que se integra a ela.

5.3.2.3 Modelo de objetos dos plug-ins

Todo o processo de comunicação com os módulos é feito através de uma única interface disponibilizada pelo módulo, a saber: o procedimento "GetModuleInfo". Essa interface retorna uma referência para a estrutura que representa cada um dos formulários disponíveis no módulo. É através dessa referência que o motor de *plug-ins* obtém todas as informações de que precisa para interagir com o módulo (detalhes no anexo A).

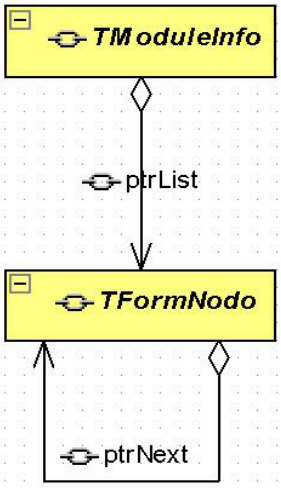

Figura 5.3 - Modelo de objetos de um plug-in

Quando um módulo registrado para aplicação é lido pelo motor de *plug-ins*, as informações contidas nessas estruturas são armazenadas e posteriormente utilizadas para o fornecimento dos enlaces para a aplicação.

5.3.2.4 Enlaces de módulos

As aplicações off-line da Plataforma utilizam um esquema de navegação estático. Isso significa dizer que todos os enlaces presentes no sistema foram colocados durante a fase de desenvolvimento do projeto. O InterLattes necessita que os módulos sejam ligados dinamicamente à aplicação, e, para isso, foi necessário elaborar um mecanismo que oferecesse essa funcionalidade.

Os enlaces dinâmicos de módulos são oferecidos à aplicação através de uma função de busca de enlaces. Quando a aplicação está integrada ao ambiente InterLattes, ela dispõe dessa função de busca, que lhe fornece todas as informações necessárias para que possa relacionar os enlaces aos componentes de navegação. Os enlaces criados são, no contexto das tecnologias utilizadas, apenas referências para métodos, que podem ser facilmente vinculados a qualquer componente da aplicação. No CV-Lattes, por exemplo, os enlaces podem aparecer em três lugares – no menu principal, na barra de ferramentas, e na tela de importação e exportação de dados.

Por questão de simplicidade, foi elaborada uma função que realiza dinamicamente os enlaces no menu principal da aplicação. Essa função é automaticamente chamada quando da iniciação do ambiente InterLattes no sistema hospedeiro (CV-Lattes e/ou GP-Lattes). Em função desse recurso, os módulos podem determinar a posição em que serão ligados neste menu.

5.3.2.5 Navegação interna

A navegação interna é um atributo implícito às aplicações off-line. Qualquer componente quando modificado provoca alterações mínimas e contextualizadas. A disponibilização do recurso de navegação interna para os módulos é feita através de janelas, que no contexto do ambiente InterLattes podem ser:

- via diálogo: abre-se uma janela sobre a aplicação;
- via container: a tela principal da aplicação apresenta um container no qual os módulos do sistema são mostrados. Quando um módulo InterLattes está usando a navegação por container, é neste container que ele é mostrado.

Na atual versão do ambiente InterLattes apenas um módulo pode estar aberto simultaneamente. Os módulos são sempre exibidos de maneira modal, isto é, apenas aquele módulo que estiver aberto aceita navegação.

5.3.2.6 Extensões de funcionalidades

O mecanismo de *plug-ins* também permite que as funcionalidades específicas das aplicações possam ser disponibilizadas para os módulos. Os módulos que utilizam essas funcionalidades só podem ser utilizados para o sistema da Plataforma que foram concebidos. Se necessário, está definida uma classe que, quando derivada pela aplicação, permite que funcionalidades específicas sejam acrescidas.

Infelizmente as aplicações off-line da Plataforma Lattes são aplicações monolíticas, e, por isso, o acréscimo de novas funcionalidades implica na atualização do executável principal do sistema. Esse problema poderia ser minimizado se o motor de *plug-ins* fosse colocado externo à aplicação, mas, para isso, a aplicação teria que estar funcionando sobre bibliotecas dinâmicas, e, como vimos no Capítulo 4, isso causa um problema sério de gerenciamento de versão para a aplicação principal, pois essas bibliotecas usam um mecanismo estático de entrelaçamento. Por outro lado, à medida que os sistemas da Plataforma forem se adequando ao ambiente InterLattes, esse problema será cada vez menos notado, pois os módulos passarão a ter cada vez mais independência das amarrações estáticas.

5.3.3 Biblioteca de apoio

Este é o principal recurso acessível pelos módulos. A biblioteca de apoio é a contraparte do framework abstrato dentro da aplicação. Ela é uma biblioteca de componentes que tem a função de fornecer uma interface robusta e funcional para módulos. O framework abstrato dos módulos faz acesso direto a essa biblioteca.

5.3.3.1 Modelo de API

Esta biblioteca está disponível para os módulos a partir do executável principal e, diferentemente da forma tradicional, não na forma de DLL (*Dynamic Library Linkage*). De acordo com a arquitetura InterStela, é muito importante que os recursos utilizados pela aplicação fiquem centralizados e que a lógica de amarração a esses recursos seja feita pelo framework abstrato.

A utilização dessa lógica nos permite evoluir tanto na aplicação principal quanto em seus módulos sem provocar os problemas tradicionais de gerenciamento de versões em executáveis baseados em DLLs. Além disso, todos os objetos são instanciados, de fato, pelo executável principal, o que viabiliza a utilização do InterLattes no contexto da Plataforma Lattes/Linux.

5.3.3.2 Componentes

A palavra de ordem aqui é disponibilizar a quantidade máxima de componentes para os módulos. Quanto mais componentes estiverem disponíveis para os módulos, menos carga de componentes de terceiros serão necessárias e, conseqüentemente, mais atributos de qualidades serão alcançados.

Entretanto, para o propósito de demonstração conceitual da utilização da arquitetura não há a necessidade de se construir uma biblioteca extensiva. Por essa razão, foram desenvolvidos apenas os componentes que estabelecem a infra-estrutura básica necessária para a inclusão dinâmica de recursos. São eles:

- componentes de persistência: as aplicações off-line da Plataforma Lattes utilizaram o DAO 3.5 (*Direct Acces Object*) como SGBD. Pelo fato de esse recurso ser compatível com o Active X, ele já possuía uma API bem definida para manipular a persistência. A biblioteca de apoio herdou um subconjunto dessa API;
- componentes de janela: oferecem um container para a composição de componentes visuais; e
- componentes de comunicação: componentes que criam um canal de comunicação entre os módulos e a aplicação *host*.

Esta pequena quantidade de componente é o suficiente para se criar um mecanismo de inclusão contínua de recursos, e isso abre um leque enorme de possibilidades de uso para módulos InterLattes, como veremos mais adiante.

5.3.3.3 Funcionalidades genéricas

O componente de janela oferece acesso a uma interface de funcionalidades genéricas aos sistemas da Plataforma Lattes, que são:

- obter informações sobre o registro do módulo;
- retornar a conexão com SGBD;
- navegar entre as janelas de um módulo;
- fechar a janela principal do módulo;
- trocar o modo de abertura da janela: diálogo ou container;
- verificar erro;

- mostrar o status quando a janela está no modo container;
- mostrar progresso quando a janela está no modo container.

5.3.3.4 Funcionalidades específicas

Cada aplicação off-line da Plataforma atendida pelo InterLattes pode possuir funcionalidades específicas, as quais são disponibilizadas para os módulos através do motor de *plug-ins*, como já visto.

As funcionalidades específicas do CV-Lattes são:

- fechar todos os módulos do CV abertos;
- enviar currículo para o CNPq;
- importar currículo;
- salvar currículo;
- salvar XML do currículo;
- obter o ID do pesquisador conectado;
- obter a data da última atualização do currículo;
- obter a versão e o build do sistema;
- abrir formulário do CV-Lattes posicionando sobre um campo em particular.

O InterLattes foi recentemente integrado ao GP-Lattes, e por isso ainda não foi acrescentada nenhuma funcionalidade específica.

5.3.4 Framework InterLattes

A construção de módulos está sempre associada ao framework abstrato concebido para a tecnologia, que doravante vamos chamar Framework InterLattes. É através desse framework que é possível a interação das aplicações construídas por terceiros com os sistemas da Plataforma Lattes.

A arquitetura InterStela define que os módulos devem ser construídos sobre um framework de componentes que seja independente de tecnologias de terceiros, mas que ao mesmo tempo não seja complexo de ser montado. A arquitetura também define a existência de drivers de tecnologia de modo a permitir o melhoramento contínuo dos módulos e da aplicação.

Devido às condições do ambiente de desenvolvimento da Plataforma, o driver de tecnologia pode ser totalmente desenvolvido dentro da aplicação principal. A biblioteca de apoio implementou diretamente as interfaces do driver de tecnologia de tal forma que ela, por simplificação, passou a ser o próprio driver. Essa característica permitiu que os módulos não necessitassem agregar o driver de tecnologia, o que implica numa redução de tamanho para os módulos.

Embora o driver de tecnologia possa ser acessado diretamente nos módulos, esse acesso deve ser evitado porque a interface do driver tem o propósito de fugir dos problemas de incompatibilidade inerentes às bibliotecas que serão encapsuladas pelos drivers e, por isso, não apresenta uma boa legibilidade. Os módulos devem se restringir ao uso do framework InterLattes e deixar os problemas da negociação com o driver de tecnologia a cargo do framework.

5.4 Possibilidades de contribuições

Por se tratar de uma tecnologia aberta, qualquer pessoa pode desenvolver seus próprios módulos InterLattes, acoplá-los aos sistemas da Plataforma Lattes e disponibilizá-los para a comunidade usuária. Essa possibilidade cria um ambiente favorável para a redução de custos e o aumento de disponibilidade de recursos para a Plataforma.

A comunidade de software livre está sempre ávida por desenvolver sistemas de seu interesse. Por tratar de TCI, a Plataforma Lattes tem o acesso facilitado a essa comunidade, pois ela normalmente está ligada às universidades. O InterLattes, se devidamente incentivado, pode criar uma cultura de desenvolvimento de módulos na comunidade de software livre que tende a agradar a ambas as partes: o governo e o cidadão. O governo fica satisfeito porque passa a ter menos gastos no desenvolvimento de funcionalidades, e o cidadão, porque passa a ter o recurso de seu interesse disponível.

O ambiente InterLattes possibilita a construção de diversos recursos para a Plataforma Lattes e, conseqüentemente, para a sua comunidade de usuários. Citam-se apenas alguns exemplos de recursos que poderiam ser adicionados à Plataforma:

- **agentes**: sistemas que interagem entre si e com os dados cadastrados pelos pesquisadores no banco de dados Lattes, procurando integrar a comunidade acadêmica e a científica, e oferecer-lhes informações inusitadas;

- **sistemas adicionais**: sistemas que customizam as funcionalidades das aplicações da Plataforma Lattes para as necessidades específicas das instituições;
- **mecanismos de pesquisa**: módulos que possibilitam aos usuários dos sistemas da Plataforma buscar informações sobre publicações em bibliotecas digitais (como SciELO, LILACS, MedLine, entre outras);
- **recursos de conversão**: módulos que fazem a importação de informações cadastradas em outros sistemas para a Plataforma Lattes, e vice-versa.

A característica mais contundente dessa tecnologia é o fato de ela permitir que novos recursos para cadastro, análise e gestão de informações em TCI possam estar disponíveis em um lugar comum: na Plataforma Lattes.

5.5 Resultados alcançados

O ambiente InterLattes é uma realidade na Plataforma Lattes do CNPq. Qualquer equipe que queira desenvolver um novo módulo para a Plataforma pode hoje acessar o site do CNPq e baixar o kit de desenvolvimento de módulos (INTERLATTES, 2004).

Alguns módulos já se encontram desenvolvidos sobre o ambiente, alguns pelo próprio Grupo Stela, outros por terceiros. Entre os que conseguimos catalogar, podemos citar: Atualizador Basilar, CV-Resume, CV-Perfil, Lattes Institucional, Extensão Incor, ProColeta Professor. A existência desses módulos demonstra o potencial da tecnologia e, para nós, comprova que os objetivos a que nos propusemos foram totalmente alcançados.

Abaixo apresentamos alguns módulos existentes que consideramos interessantes para que se vislumbrem o potencial da tecnologia e os resultados já alcançados.

5.5.1 Módulo de atualização basilar

Este módulo é um mecanismo que permite a atualização dinâmica das unidades de informações estáticas presentes na base local da Plataforma. A atualização oferecida faz mais do que uma simples substituição dos dados, ela também procura por inconsistências na base local e aplica as correções cabíveis. Este módulo foi concebido para permitir que correções na base de dados do CNPq não impliquem em atualizações da versão dos sistemas de coleta de dados instalados na máquina do pesquisador.

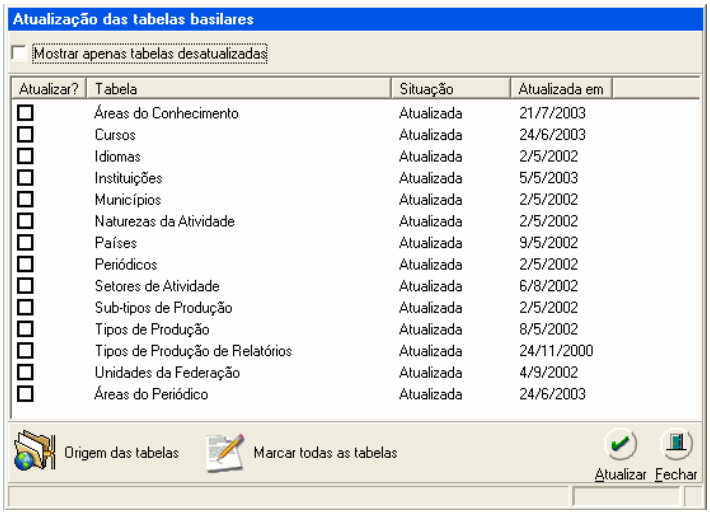

Figura 5.4 - Tela principal do módulo de atualização basilar

5.5.2 CV-Resume

O módulo InterLattes CV-Resume é um aplicativo que se adiciona ao CV-Lattes off-line para gerar o résumé com base nas informações curriculares registradas pelo usuário. Este módulo tem por objetivo a apresentação de texto resumido sobre o perfil curricular do usuário do Sistema de Currículos Lattes, baseado em uma descrição formal das informações disponíveis na base Lattes, automaticamente traduzidas para o formato texto.

O résumé pretende apresentar os principais pontos observados em um currículo, de forma que o usuário possa verificar como futuros avaliadores ou interessados devem interpretar as informações fornecidas à Plataforma Lattes (CV-RESUME, 2004).

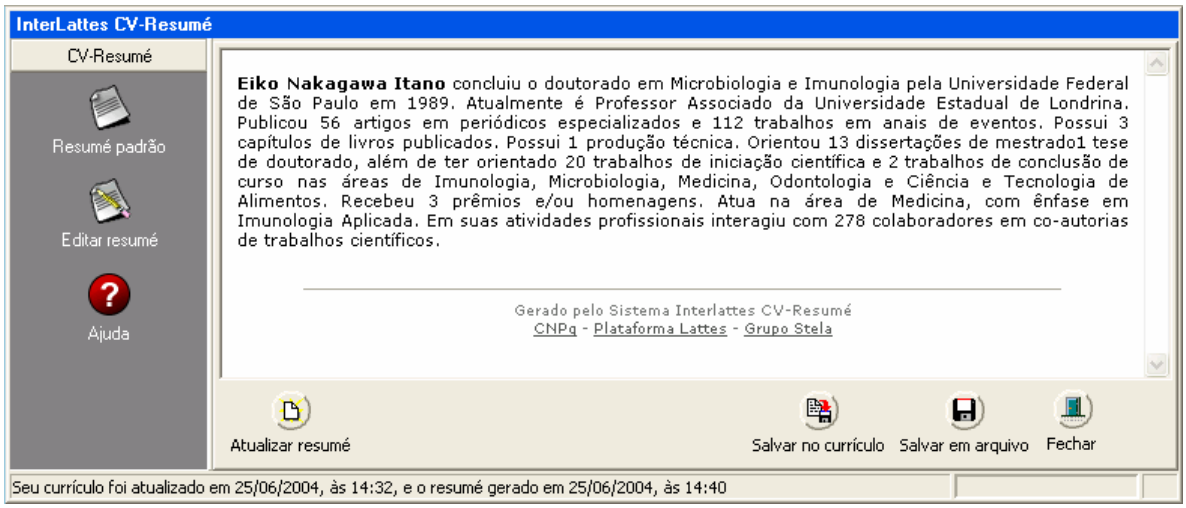

Figura 5.5 - Exemplo das informações geradas pelo CV-Resume

5.5.3 CV-Perfil

Este módulo tem por objetivo permitir que o usuário do Sistema CV-Lattes verifique a qualidade de suas informações registradas em seu currículo e, assim, possa efetivar as

alterações ou as atualizações necessárias para que seus dados curriculares sejam mais fidedignos à caracterização de sua vida profissional (WATANABE, 2004).

O perfil curricular apresentado é resultado da aplicação automática de regras dedutivas e procedimentos de comparações entre as informações fornecidas pelo usuário ao Sistema CV-Lattes. A recuperação das informações e a funcionalidade de ligação dinâmica aos módulos internos do CV-Lattes são recursos fornecidos pelo framework InterLattes.

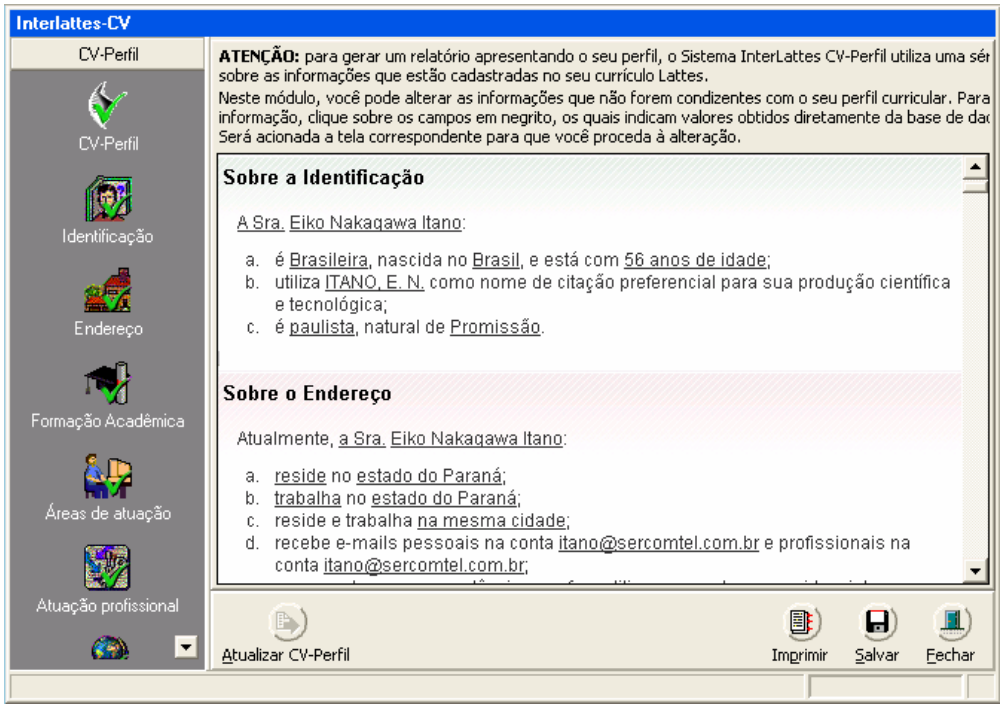

Figura 5.6 - Exemplo de um perfil montado pelo módulo CV-Perfil

5.5.4 Módulos Lattes Institucional

São módulos InterLattes que têm por objetivo adicionar informações que não constam na base do Currículo Lattes e que são de necessidade de uma universidade em particular. Esses módulos se integram aos sistemas da Plataforma Lattes e estabelecem uma comunicação entre o Sistema de Currículos Lattes, as bases de dados da universidade e as bases de dados do CNPq. Esses recursos são oferecidos sem comprometer as informações presentes na base de dados do Currículo.

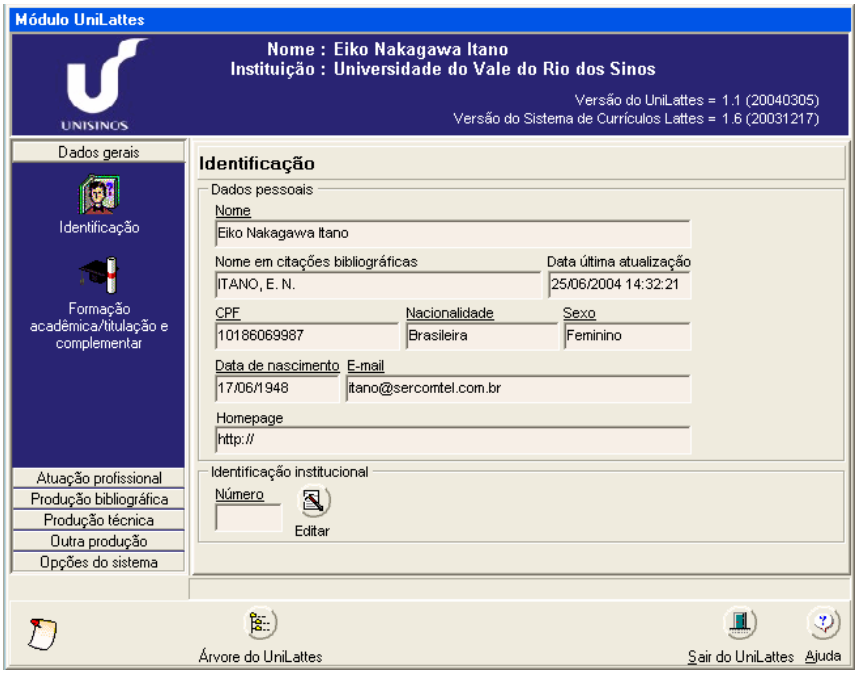

Figura 5.7 - Módulo Lattes Institucional da UNISINOS

5.6 Considerações finais

Este capítulo procurou comprovar a arquitetura InterStela pela aplicação de seus conceitos. Para isso, foi escolhida uma aplicação de E-Gov que fosse consolidada e que estivesse disposta a participar deste trabalho. A aplicação escolhida foram os sistemas de coleta de dados off-line da Plataforma Lattes.

Um relato sobre a Plataforma Lattes e seus sistemas foi então apresentado de modo a contextualizar a aplicação da arquitetura. Esta aplicação, que recebeu o nome de InterLattes, foi então apresentada de maneira estrutural e funcional.

Finalmente, discutiu-se como esse ambiente poderia contribuir para a redução de custos e o aumento da disponibilidade de recursos para a Plataforma. Como prova, foram citados alguns módulos que tiveram seus desenvolvimentos baseados nesse ambiente e que já se encontram disponíveis para qualquer usuário da Plataforma.

6 Conclusões e trabalhos futuros

A pesquisa realizada mostrou que, apesar de existir uma crescente demanda por sistemas de software no contexto de governo eletrônico, esses sistemas são desenvolvidos de maneira inadequada e custosa. Uma das causas, e talvez a principal, é a falta de uma arquitetura voltada especificamente para esse domínio de aplicação. O estudo feito sobre E-Gov mostrou que no setor público o surgimento de novos sistemas de software se dá em todas as áreas de atuação estatal, em suas diversas agências governamentais, e a grande maioria desses sistemas é construída em função das experiências heurísticas das equipes de desenvolvimento. O resultado é o engessamento do processo produtivo de software e o aumento do custo global desses sistemas. A sociedade do conhecimento não pode mais arcar com o custo causado por esse modelo de desenvolvimento, e, para isso, demandam-se soluções.

A característica-chave para se reverter esse quadro é a estruturação lógica da forma como os sistemas de governo eletrônico são montados. Para tal, um dos recursos necessários é a especificação de um padrão estrutural de software para governo que esteja em conformidade com os requisitos desse domínio e, ao mesmo tempo, promova uma redução de custos e um ganho de qualidade nos sistemas de software produzidos. É nesse contexto que se dá a relevância da Arquitetura de Software. Embora essa disciplina seja recente, nos últimos anos ela tem evoluído muito. Através de um estudo minucioso das estruturas dos sistemas, a disciplina tem procurado soluções para os mais diversos problemas de estruturação existentes nos sistemas de software. Para isso, foram catalogados e organizados padrões arquiteturais em todos os níveis de abstração que representam as melhores soluções encontradas para os problemas estruturais. Baseados nesse catálogo de padrões, elaboramos nossa proposta.

Visando estabelecer uma lógica estrutural que promovesse uma redução de custos e um ganho de qualidade na produção de software para governo, este trabalho propõe que a elaboração e a construção de novos sistemas sejam baseadas numa arquitetura de software voltada aos requisitos desse domínio de aplicação. Para tal, foram levantados os problemas dos sistemas de software existentes, os requisitos que essas aplicações devem seguir e as soluções existentes mais adequadas. Com base nesse levantamento, propôs-se uma arquitetura de software para governo eletrônico, que foi denominada InterStela.

O modelo arquitetural proposto foi empregado no contexto das aplicações off-line da Plataforma Lattes, a saber: o Sistema de Currículo Lattes e o Sistema de Grupos de Pesquisas. Como resultado criou-se um ambiente de desenvolvimento de módulos para a inclusão

contínua e descentralizada de recursos para esses sistemas (InterLattes). Na aplicação se evidenciou que o emprego de uma arquitetura adequada à realidade do domínio da aplicação promove a redução de custos e o aumento da produtividade no desenvolvimento de novos módulos. Isso ficou claro em virtude dos resultados que o InterLattes tem promovido para o contexto de TCI da Plataforma Lattes, como o surgimento de módulos (CV-Resume, CV-Perfil e outros).

A proposta arquitetural apresentada no Capítulo 4 é a indicação de um caminho para a construção de sistemas de baixo custo, factíveis aos prazos e de qualidade. A indústria de desenvolvimento de software, que dispõe de recursos humanos e financeiros suficientes, poderia se valer dos conceitos de navegador de aplicação, da biblioteca abstrata, dos drivers de tecnologia, dos modelos de dados voltados a unidades de informação e de tantos outros apresentados pela arquitetura para criar um framework de desenvolvimento que não só atendesse às aplicações em governo eletrônico mas também às aplicações em outros domínios.

6.1 Trabalhos futuros

Esta dissertação propõe o emprego de arquiteturas de software elaboradas sobre um domínio de aplicações na construção de sistemas de software. Um primeiro plano de possibilidades de trabalhos futuros está vinculado à elaboração e ao melhoramento de modelos arquiteturais. Nesse sentido, prevêem-se a:

- elaboração de uma linguagem para a especificação de arquiteturas de domínio;
- criação de metodologia para levantamento de requisitos de domínio;
- concepção de arquiteturas para outros domínios de aplicação.

Além disso, esta dissertação estabelece vários elementos estruturais que representam soluções desejáveis a qualquer desenvolvimento de sistema. Assim, prevemos que os seguintes produtos poderiam ser produzidos:

- um navegador de aplicação, que diferentemente dos navegadores Web, estaria voltado a atender aos requisitos de navegabilidade e gerenciamento transparente de versões de componentes, criando assim um contexto favorável às aplicações Web;
- ferramentas de geração automática de código que traduzam um framework abstrato diretamente sobre uma tecnologia, a exemplo do MDA. A associação de uma

ferramenta desse tipo a uma ferramenta case poderia criar uma solução completa de produção de sistemas de software;

- padrões de frameworks abstratos para domínios específicos de aplicações que fossem detalhados e mantidos por organismos de padronizações.

7 Referências Bibliográficas

(ALLEN, 1997) ALLEN, Robert J.. **A Formal Approach to Software Architecture**. Maio de 1997. Tese de PhD.

(BALUTIS, 1999) Balutis, A. P. - Digital Government - When All is Said and Done, Electronic Government Journal, Vol. 2, No. 6, Novembro.

(BASS; CLEMENTS; KAZMAN, 2003) BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. **Software Architecture in Practice**. 2. ed. ISBN 0-321-15495-9.

(BOOCH; RUMBAUGH; JACOBSON, 1999) BOOCH; RUMBAUGH; JACOBSON. **The UML Modeling Language User Guide.** Addison-Wesley, 1999.

(BUSCHMANN et al., 1996) BUSCHMANN, Frank et al. **Pattern Oriented Software Architecture**: A System of Patterns. Wiley, 1996.

(CAVALCANTI; GOMES, 2000) CAVALCANTI, Marcos; GOMES, Elisabeth. **A Sociedade do Conhecimento e a política industrial brasileira**. Disponível em: <http://www.mdic.gov.br/tecnologia/revistas/artigos/Coletanea/CavalcantiGomes.pdf>. Acesso em: 10 set. 2003.

(CV-RESUME, 2004) Site da Plataforma Lattes. **InterLattes / Módulo CV-Resume da Currículo Lattes Off-line**. Disponível em: <http://lattes.stela.ufsc.br/curriculo/cvresume/>. Acesso em: 22 jun. 2004.

(DRUCKER, 1993) Drucker, Peter. **Post-capitalist Society**. Butterworth-Heinemann. 1993, ISBN 0-7506-2025-0.

(ELSAS, 2001) ELSAS, Alexander. **The Problems of Economic Integration of Ukraine into the European Union: Globalization and New Economy - Consequences for Europe and Ukraine**. In: SEVENTH INTERNATIONAL SCIENTIFIC CONFERENCE - SUMMER SCHOOL.

(FERNANDES, 2001) FERNANDES, Andréa. **Governo no Brasil** - Estudo da Secretaria para Assuntos Fiscais do Banco Nacional de Desenvolvimento Econômico e Social (SF/BNDES).

(FOWLER et al., 2002) FOWLER, Martin et al. **Patterns of Enterprise Application Architecture**. Addison Wesley. 5 nov. 2002. ISBN 0321127420.

(GACEK; BOEHM; ADB-ALLAH, 1995) GACEK, C.; BOEHM, B.; ADB-ALLAH, A. **On the Definition of Software Achitecture**. In: PRIMEIRO WORKSHOP INTERNACIONAL SOBRE ARQUITETURAS PARA SISTEMAS DE SOFTWARE - 17a. CONFERÊNCIA INTERNACIONAL DE ENGENHARIA DE SOFTWARE. Abril, Seattle, EUA. p. 85-95.

(GALIZA, 2004) GALIZA, Mariana. **Plataforma Lattes ganha Prêmio E-Gov**. Assessoria de Imprensa do CNPq. Disponível em: <http://www.cnpq.br/noticias/260504.htm>.. Acesso em: 24 jun. 2004.

(GALLAGHER, 2000) GALLAGHER, Brian P. **Using the Architecture Tradeoff Analysis MethodSM to Evaluate a Reference Architecture**: A Case Study. Nota técnica CMU/SEI-2000-TN-007. jun. 2000.

(GALLI et al., 2003) GALLI, Marcio et al. **Inner-Browsing: Extending Web Browsing the Navigation Paradigm**. Disponível em: <http://devedge.netscape.com/viewsource/2003/inner-browsing/>. Acesso em: 3 mar. 2004.

(GAMMA et al., 1995) GAMMA, Erich et al. **Design Patterns – Elements of Reusable Object-Oriented Software**. Publisher: Addison-Wesley Professional; 1st edition (January 15, 1995). ISBN: 0201633612.

(GONÇALVES, 2000) GONÇALVES, Alexandre L. **Utilização de Técnicas de Mineração de Dados em Bases de C&T**: Uma Análise dos Grupos de Pesquisa no Brasil. 2000. Dissertação (Mestrado em Engenharia de Produção) – Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal de Santa Catarina, Florianópolis, 2000.

(GRIMÁN; ROJAS; PÉREZ, 2002) GRIMÁN, A.; ROJAS, T.; PÉREZ, M. **Methodological approach for developing a KMS: a case study**. CLEI Electronic Journal, v. 5. n. 1, June 2002.

(HOLDES, 2001) HOLDES, Douglas. **E-Gov, na E-Bussiness plan for government**. ISBN 1-85788-278-4. 2001. p. 1-12.

(INTERLATTES, 2004) Site do CNPq. **InterLattes / Desenvolvedores**. Disponível em: <http://lattes.cnpq.br/interlattes/desenvolvedores.jsp>. Acesso em: 21 jun. 2004.

(JOIA, 2002) JOIA, Luiz Antônio. **O que é Governo Eletrônico**. Fundação Getúlio Vargas /EBAPE, 2002. Disponível em: <http://www.ebape.fgv.br/e_government/asp/dsp_oquee.asp>. Acesso em: 7 set. 2003.

(JORNAL DA CIÊNCIA, 2004) Jornal da Ciência. **Plataforma Lattes**: o auto-retrato da C&T. Disponível em: <http://www.jornaldaciencia.org.br/Detalhe.jsp?id=17132>. Acesso em: 19 mar. 2004.

(KELLER, 2000) KELLER, B. **Four phases of e-government**: phase 4 - Transformation. GartnerGroup, 20 dez. 2000. Research Note.

(LATTES MINING, 2001) Grupo Stela. **Subprojeto 4.6: Instrumentos Lattes Mining**. Projeto elaborado em 2001 e aprovado para execução para o período de 2002 a 2004 pelo CNPq.

(LINK-ANÁLISES, 2001) Grupo Stela. **Subprojeto 4.4: Instrumentos de Link-Análises**. Projeto elaborado em 2001 e aprovado para execução para o período de 2002 a 2004 pelo CNPq.

(MALVEAU; MOWBRAY, 2000) MALVEAU, Raphael; MOWBRAY, Thomas J. **Software Architect Bootcamp**. ISBN: 0-13-027407-0. Prentice Hall PTR, 13 out. 2000.

(MANZOOR, 2002) MANZOOR, Kashif. **The Common Language Runtime (CLR) and Java Runtime Environment (JRE)**. Disponível em: <http://www.codeproject.com/dotnet/clr.asp>. Acesso em: 15 jan. 2004.

(OAKLAND, 2003) Oakland County's. **Four Phases of eGovernment**. Disponível em: <http://www.co.oakland.mi.us/egov/about/phases.html>. Acesso em: 22 out. 2003.

(OSBORNE; GAEBLER, 1993) OSBORNE, David; GAEBLER, Ted. **Reinventing Government: How the Entrepreneurial Spirit Is Transforming the Public Sector**. 1993.

(PACHECO, 2003) PACHECO, Roberto. **Uma metodologia de desenvolvimento de plataformas de governo para geração e divulgação de informações e de conhecimento**, 2003, Florianópolis, Grupo Stela.

(PCIP, 2002) Conselho do Pacífico para Políticas Internacionais. **Diretrizes para o Governo Eletrônico no Mundo em Desenvolvimento - 10 Perguntas que os Líderes do Governo Eletrônico Devem Fazer a si Mesmos**. Disponível em: <http://www.pacificcouncil.org>. Acesso em: abr. 2002.

(PERRI, 2001) PERRI. E-governance. **Do Digital Aids Make a Difference in Policy Making? Designing E-Government**, Prints J.E.J. (ed.), Kluwer Law International, p. 7-27.

(PORTAL LATTES, 2004) CNPq. **Portal da Plataforma Lattes.** Disponível em: <http://lattes.cnpq.br>. Acesso em: 1 mar. 2004.

(REDES-GP, 2001) Grupo Stela. **Subprojeto 3.4: Sistemas Redes-GP**. Projeto elaborado em 2001 e aprovado para execução para o período de 2002 a 2004 pelo CNPq.

(RICHARDSON; JACKSON; DICKSON, 1990) RICHARDSON, G.; JACKSON, B. M.; DICKSON, G. W. A. **Principles-Based Enterprise Architecture: Lessons from Texaco and Star Enterprise**. MIS/Quartely, v.14, n.4, p. 385-403, dez. 1990.

(SCTG, 2003) Standards and Conformance Testing Group. **Using Architectural Description Languages to Improve Software Quality and Correctness**. Disponível em: <http://www.itl.nist.gov/div897/ctg/adl/adl_info.html>. Acesso em: 17 dez. 2003.

(searchNetworking.com, 2003) **OSI Reference Model illustrated**. Disponível em: <http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci523729,00.html>. Acesso em: 13 dez. 2003.

(SICSÚ; MELO, 2000) SICSÚ, Abraham Benzaquen; MELO, Lúcia Carvalho P. **Sociedade do Conhecimento: Integração Nacional ou Exclusão Social?** Disponível em: <http://www.mct.gov.br/CEE/revista/Parcerias9/06revista9lucia.pdf>. Acesso em: 15 set. 2003.

(SIDDALINGAIAH, 2000) SIDDALINGAIAH, Madhu. **An Analysis of DotNET**. Disponível em: <http://java.sun.com/features/2000/11/dot-net.html>. Acesso em: 1 jan. 2004.

(TAIT, 1999) TAIT, Tania Fatima Calvi. **Um Modelo de Arquitetura de Sistemas de Informação para o Setor Público: Estudo em Empresas Estatais Prestadoras de Serviços**

Informáticos. 1999. Tese de Doutorado. Disponível em: <http://teses.eps.ufsc.br/Resumo.asp?677>. Acesso em: 9 set. 2003.

(UDELL, 2000) UDELL, Jhon. **Tangled in the Threads - JVM and CLR - Does JVM already deliver what .NET's CLR promises?** Disponível em: <http://udell.roninhouse.com/bytecols/2000-12-13.html>. Acesso em: 15 jan. 2004.

(WATANABE, 2004) WATANABE, Wagner T. **Introdução CV-Perfil.** Mensagem pessoal recebida por watanabe@stela.ufsc.br em 23 jul. 2004.

(WENTZEL, 1994) WENTZEL, K. Software Reuse, Facts and Myths. In: 16A. CONFERÊNCIA INTERNACIAL DE ENGENHARIA DE SOFTWARE, maio 1994, Sorrento, Itália. p. 267-273.

(ZACHMAN, 1987) ZACHMAN, J. A. **A framework for Information Systems Architecture**. IBM System Journal, v. 26, n.3, p. 276-285, 1987.

APÊNDICE A
Especificação da API do Ambiente InterLattes

Este apêndice apresenta a atual API do driver de tecnologia utilizada pelos frameworks InterLattes. Em virtude de os sistemas de captura da Plataforma Lattes terem sido desenvolvidos com a tecnologia ObjectPascal/VCL/Delphi, foram utilizados elementos dessa tecnologia na especificação, mas isso não a limita, em hipótese alguma, a essa tecnologia.

A.1 Resumo técnico da tecnologia

O diagrama abaixo demonstra como funciona a solução do framework InterLattes.

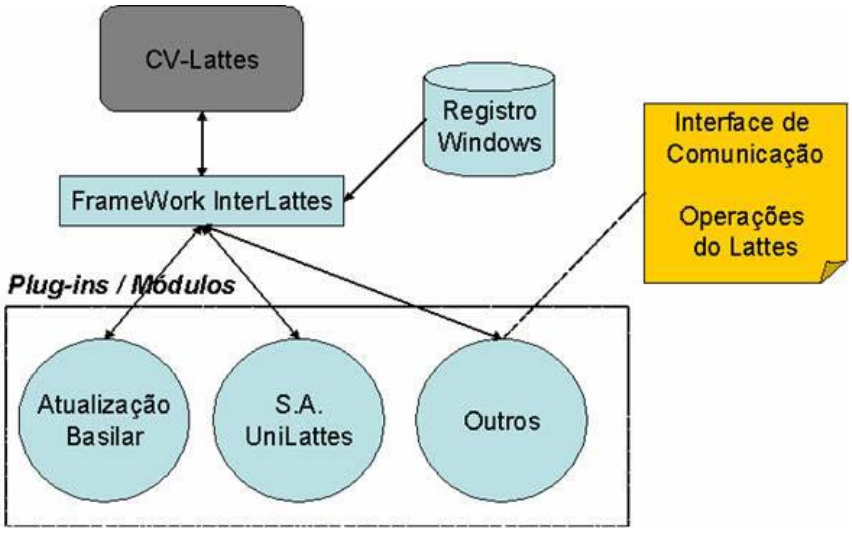

Figura A.1 – Visão esquemática reduzida do Ambiente InterLattes

Nesse diagrama, referenciou-se o Sistema de Currículos Lattes, mas o InterLattes também está presente no Sistema de Grupo de Pesquisa no Brasil da Plataforma Lattes.

O framework do InterLattes só disponibiliza módulos que foram devidamente registrados e que são compatíveis com o próprio framework. Esse processo de verificação ocorre no momento da carga do sistema host (CV-Lattes na Figura A.1 – Visão esquemática reduzida do Ambiente InterLattes). Cada módulo registrado é carregado e liberado nesse momento, a fim de buscar informações de vinculação ao sistema host. Essas informações são necessárias para o gerenciamento do módulo e para a criação de links que habilitam o acionamento do módulo pelo usuário. Esses links são dispostos no menu principal do sistema host mas também poderiam ser colocados em outros locais desse sistema, o que varia de sistema para sistema.

No CV-Lattes, por exemplo, é possível vincular os links na barra de botões e na tela de importação.

Os futuros desenvolvedores de módulos para o InterLattes devem se preocupar em utilizar técnicas que evitem que o módulo de referência seja demasiadamente grande em termos de bytes. Isso porque, como observado, cada módulo é "investigado" pelo InterLattes, no momento da carga do sistema host, na busca de informações necessárias para sua integração no ambiente. Caso esse sistema seja excessivamente grande, pode atrapalhar o processo de carga do sistema e em máquinas com menos recursos provavelmente nem funcione. Se por acaso um módulo tiver que ser grande devido à sua complexidade, o responsável pelo desenvolvimento de módulos deverá utilizar a técnica de módulo de referência pequeno, que consiste em tornar a DLL registrada no InterLattes com o menor tamanho possível, e só no momento de sua execução propriamente dita é que poderá carregar os outros recursos necessários.

Todo esse processo de comunicação é feito através de uma única interface disponibilizada pelo módulo, a saber: o procedimento "GetModuleInfo". Essa API fornecida pela DLL do módulo retorna um ponteiro para a estrutura "TmoduleInfo" (ver Figura A.2). É através dessa referência que o framework do InterLattes irá obter todas as informações de que precisa para interagir com o módulo.

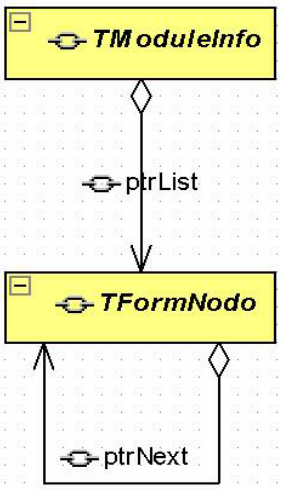

Figura A.2 - Visão da externa da estrutura de Objetos de um Módulo

Os tipos "TmoduleInfo" e "TformNodo" são na realidade uma estrutura registro, pois nem todas as linguagens dão suporte a objetos. Embora não sejam especificamente uma classe, são implementados como se fossem, pois possuem atributos e métodos. Sendo assim, daqui para a frente, esses tipos serão referenciados como classes.

A classe “TModuleInfo”, além de trazer informações sobre o módulo (tais como versão, nome, etc.), aponta para uma lista simplesmente encadeada de formulários fornecidos pelo módulo. Já a classe “TformNodo” descreve cada formulário e oferece métodos para manipulá-los.

Abaixo serão detalhados o significado existente em cada atributo e o método para cada uma dessas classes. Será usada a sintaxe do Object Pascal, por ser uma das linguagens mais difundidas atualmente.

A.2 Especificação da estrutura de objetos

A.2.1 Interface TModuleInfo

```
TModuleInfo = record
            // propriedades
            numVersion: Integer;
            strSystem : PChar;
            strName   : PChar;
            ptrList   : PFormNodo;
            // métodos
             dllModuleIcon: function: IPictureDisp;
end;
```

A.2.1.1 Documentação

Atributo	Descrição
numVersion	Número de versão do módulo. Valor numérico que deve ser maior ou igual a 1000, em que 1000 representa a versão 1.0.
strSystem	Identificadores dos sistemas da Plataforma Lattes que serão compatíveis com o módulo. Se o módulo atender a mais de um sistema, os identificadores deverão estar separados por ponto-e-vírgula. Se o módulo é genérico e pode ser utilizado em todos os sistemas hosts da Plataforma Lattes, o valor desse parâmetro deve ser “LATTES”. Os sistemas compatíveis hoje são: • CUR: Sistema de Currículos Lattes. • GRP: Sistema Grupo.
strName	Nome do módulo a ser apresentado no gerenciador de módulos.
ptrList	Ponteiro para lista de classes que descrevem cada formulário fornecido pelo módulo.

Método	Descrição
dllModuleIcon	Função que retorna um ícone 32x32 que representa o módulo no gerenciador de módulos.

A.2.1.2 Comentários

A instância retornada pelo método "GetModuleInfo" deve ser a de uma variável estática, de preferência que fique dentro do escopo da própria função. É necessário evitar que fique fácil para futuros técnicos que forem dar manutenção ao módulo o acesso direto ao descritor do módulo, procedimento esse que só deve ser gerenciado pelo InterLattes.

O método "dllModuleIcon" retorna uma interface definida pelo Windows aos moldes dos objetos COM. Com o objetivo de facilitar o trabalho dos técnicos, abaixo é apresentada uma implementação de como montar o objeto a ser retornado por essa interface. A função a seguir, embora tenha sido implementada em Object Pascal do Delphi, só utiliza funções da API do Windows, o que a torna viável de ser adaptada para qualquer outra linguagem de programação do SO.

```
function ReadPictureFromInstance(const strPPictureName: string):
IPictureDisp;
var
  PictureDesc: TPictDesc;
  hldLBitmap: HBitmap;
begin
  result := nil;
  hldLBitmap := LoadBitmap(hInstance, PChar(strPPictureName));
  if hldLBitmap <> 0 then
  begin
    FillChar(PictureDesc, sizeof(PictureDesc), 0);
    PictureDesc.cbSizeOfStruct := SizeOf(PictureDesc);
    PictureDesc.picType := PICTYPE_BITMAP;
    PictureDesc.hbitmap := hldLBitmap;
    OleCreatePictureIndirect(PictureDesc, IPicture, true, Result)
  end;
end;
```

É importante lembrar que a API "GetModuleInfo" deve ser exportada com o índice 1, e o nome deve respeitar o caso, pois esse nome é usado de maneira sensitiva.

A.2.2 Interface TFormNodo

```
  TEvtWndHandle = record
    function(Data: Pointer): HWND;
    Data: Pointer;
  end;

  TParamMethod  = record
    function(Data: Pointer;
             numPIDParam: Integer;
             vrtPArgs: OleVariant): OleVariant; stdcall;
```

```
    Data: Pointer;
  end;

  TFormNodo= record
    // propriedades
      hldForm       : TEvtWndHandle;
      strSysMenuPath: PChar;
      strFormCaption: PChar;
      strMenuCaption: PChar;
      strDescription: PChar;
      bolNeedConnect: Boolean;
      bolShowStatus : Boolean;
      ptrNext       : PFormNodo;
      // Métodos
      dllSmallIcon  : function: IPictureDisp;
      dllLargeIcon  : function: IPictureDisp;
      dllFormCreate : function(hldPWinApp: HWND): TEvtWndHandle; stdcall;
      dllFormFree   : procedure(hldPHandle: TEvtWndHandle); stdcall;
      dllHostSolicitation: function(hldPHandle: TEvtWndHandle;
                                    numPIDParam: Integer;
                                    vrtPArgs: OleVariant): OleVariant; stdcall;

      dllModuleSolicitation: procedure(hldPHandle: TEvtWndHandle;
                                       mthPParams: TParamMethod); stdcall;;
end;
```

A.2.2.1 Documentação

Atributo	Descrição
hldForm	Deve ser preenchido com nil. Este atributo é de uso interno do InterLattes.
strSysMenuPath	Determina onde irá ser inserido o link que fará referência ao módulo no menu do sistema. Este atributo é uma *string* que possui uma gramática à moda URL. Deve ser usado o símbolo // para separar cada elemento do diretório representado pelo menu do sistema. Veja os exemplos: *exemplo 1:'//&Arquivo//Exemplo OO 2';* *exemplo 2:'//&Arquivo//Exemplo OO 2[0]'.* Observe que o segundo exemplo tem um "[0]". Essa informação serve para indicar a posição do item de menu dentro do grupo do menu no qual o link será inserido. Valores positivos indicam a ordem de cima para baixo; valores negativos indicam a ordem de baixo para cima. Este parâmetro também pode ser **nil**. Nesse caso, não haverá link no menu do *sistema host* que fará referência a este formulário.

strFormCaption	Nome do formulário a ser apresentado no container InterLattes.
strMenuCaption	O container InterLattes pode apresentar um menu lateral semelhante ao utilizado na interface do CV-Lattes. Cada formulário cria um grupo nesse menu. O texto que referencia o nome desse grupo é inserido nesta propriedade. Caso ela seja **nil**, não será criado um grupo para este formulário.
strDescription	Texto que descreve o formulário. Esse texto deve ser especialmente detalhado para o primeiro módulo do sistema, pois este será o módulo que irá descrever o sistema no gerenciador de módulos.
bolNeedConnect	Informa se, para chamar esse módulo, é necessário estar conectado. Os sistemas do Lattes, quando abertos, podem ou não estar com um Currículo ou Grupo aberto. Essa propriedade irá refletir no estado de habilitação do item de menu que faz referência ao módulo no sistema host.
bolShowStatus	Mostra barra de status no container do InterLattes. Quando se usa o módulo que utiliza o container, este pode tornar visível uma barra de status através dessa propriedade.
ptrNext	Ponteiro para o próximo formulário. Quando essa propriedade for nil, indica que não há outros formulários disponíveis no módulo.

Método	**Descrição**
dllSmallIcon	Retorna o ícone que será usado no menu do sistema. Caso seja **nil**, não terá ícone no link. Esse ícone deve ter a dimensão de 16x16.
dllLargeIcon	Ícone de 32x32 usado pelo gerenciador de módulos para fazer referência ao módulo. Caso seja **nil**, não será mostrado o ícone.
dllFormCreate	Este método é requerido pelo InterLattes quando do acionamento do módulo pelo usuário (via menu, barra de botões, etc.). Ele deve criar uma instância do formulário e retornar uma função que possa também retornar um manipulador de janela para o InterLattes. Caso o valor retornado pela função seja 0, o InterLattes não utilizará seu container-padrão, passando a ser responsabilidade do módulo todo e

	qualquer controle de interface do usuário.
dllFormFree	Este método é responsável pela liberação do objeto de formulário instanciado pelo método "dllFormCreate". Recebe por parâmetro um ponteiro para "TevtWndHandle". O Atributo Data aponta para o objeto instanciado.
dllHostSolicitation	Este método é requerido pelo InterLattes para que o sistema possa fazer solicitações ao módulo, a fim de montar menus, informar cliques e outras funções.
dllModuleSolicitation	Este método é chamado apenas uma vez. A ação é passar o ponteiro para uma estrutura que permitirá ao módulo fazer solicitações ao *sistema host*.

A.2.2.2 Comentários

Da mesma forma como no método "GetModuleInfo", aqui é recomendado criar procedimentos, um para cada módulo, e fazer com que retornem um ponteiro para "TFormNodo". O ponteiro deve referenciar uma variável estática, pois o InterLattes não faz suposições sobre a instância dessa variável.

Os métodos "dllHostSolicitation" e "dllModuleSolicitation" definem o mecanismo de comunicação do InterLattes com o módulo. Existem diversas ações que podem ser chamadas pelo módulo ao sistema host e vários eventos que são disparados pelo InterLattes para o módulo por meio desses métodos. A próxima seção irá detalhar cada uma dessas ações bem como os eventos.

A.3 Especificação do mecanismo de comunicação

A comunicação entre o módulo e o sistema host é feita através de eventos e ações. Os eventos constituem ações disparadas pelo InterLattes para o módulo quando este precisa informar que algo aconteceu ou que necessita de alguma informação. As ações são operações que podem ser solicitadas pelo módulo para obter informações ou solicitar uma tarefa ao sistema host.

A.3.1 Eventos do InterLattes

Os eventos são disparados pelo método "dllHostSolicitation", e as informações são trocadas através dos parâmetros "numPIDParam", "vrtPArgs" e do retorno da função. O

parâmetro “numPIDParam” é o identificador do evento. O parâmetro “vrtPArgs” diz respeito aos parâmetros que esse evento está enviando. O retorno é o resultado esperado pelo evento.

Cada evento possui uma sintaxe própria, conforme descrito abaixo.

<table>
<tr><th>Identificador</th><th>Função protótipo</th></tr>
<tr><td>IMM_GETMENUICONS</td><td>function GetMenuItems: OleVariant;</td></tr>
<tr><td colspan="2">vrtPArgs sempre unassigned;

Informa ao InterLattes quantos itens de menus serão apresentados no container-padrão do InterLattes, quais as etiquetas de cada item e quais as figuras que deverão ser mostradas. Todas essas informações são retiradas do “OleVariant” de retorno. O formato do retorno é um vetor de itens, de acordo com a seguinte regra:

// TItem[0] => string
// TItem[1] => IPictureDisp
TItem = array[0..1] of OleVariant;
TItems = array of TItem;

Este método não precisa ser implementado caso não esteja sendo usado o container-padrão.</td></tr>
<tr><td>IMM_MENUCLICK</td><td>function OnMenuClick(numPItem: integer): boolean;</td></tr>
<tr><td colspan="2">vrtPArgs é um valor inteiro;

Informa ao módulo que um item do menu do container foi pressionado. Esse evento é mandado apenas para o formulário que definiu o item com o número do item correspondente.</td></tr>
<tr><td>IMM_FORMACTIVATED</td><td>procedure OnFormActivated;</td></tr>
<tr><td colspan="2">vrtPArgs sempre unassigned;

Informa ao módulo que o formulário foi ativado pelo InterLattes.</td></tr>
</table>

A.3.2 Ações genéricas disponíveis para o módulo

Essas ações são disponibilizadas ao módulo através do método “dllModuleSolicitation”. Esse método entrega ao módulo um ponteiro para um método de objeto do Interlattes (mthPParams) que processa as solicitações.

Quando uma solicitação é feita ao sistema host, deve ser identificada pelo seu número e passada no parâmetro “numPIDParam”. Já os argumentos da solicitação devem ser passados no parâmetro “vrtPArgs”. A resposta e o retorno da função variam de ação para ação.

Abaixo estão catalogadas todas as ações que valem para qualquer sistema host da Plataforma Lattes:

Identificador	Função protótipo
IMC_MODULEKEY	`function GetModuleKey(): String;`
vrtPArgs sempre **unassigned**. Retorna o path, no registro do Windows, do registro do módulo no InterLattes. Este path é relativo ao grupo LocalMachine.	
IMC_GETDATABASE	`function GetDatabase(): IDispatch;`
vrtPArgs sempre **unassigned**. Retorna o objeto COM que representa o banco de dados. Esse objeto pode variar de *sistema host* para *sistema host*. Os seguintes sistemas da Plataforma Lattes retornam um objeto Database do Access: Sistema de Currículos Lattes e Diretório de Grupos de Pesquisa.	
IMC_GOTOFORM	`function GoToForm(numPForm: integer): Boolean;`
vrtPArgs recebe o número do formulário que se deseja tornar ativo. Essa ação permite que o módulo solicite ao InterLattes tornar outro formulário ativo no container-padrão. Se a ação foi bem-sucedida, irá retornar true como resposta. Essa ação não tem utilidade quando não se está usando container.	
IMC_CLOSE	`function CloseContainer: Boolean;`
vrtPArgs sempre **unassigned**. Fecha o container-padrão do InterLattes fazendo com que o módulo seja, logo que possível, liberado da memória. O retorno é true se a ação foi executada com sucesso. Essa ação não tem utilidade quando não se está usando container.	
IMC_ENABLECONTAINER	`function EnableContainer: Boolean;`
vrtPArgs sempre **unassigned**. Habilita o container-padrão do InterLattes. Retorna true se a ação foi executada com sucesso. Essa ação não tem utilidade quando não se está usando container.	
IMC_DISABLECONTAINER	`function DisableContainer: Boolean;`
vrtPArgs sempre **unassigned**. Desabilita o container do InterLattes. Essa ação é utilizada quando se deseja fazer algum processamento e não se quer permitir que o usuário clique em algum item de menu do container-padrão. Retorna true se a ação foi executada com sucesso.	

Esta ação não tem utilidade quando não se está usando container.

IMC_ERRORSVERIFY	`function CheckError(numPIDSysError: integer): integer;`

vrtPArgs vetor com pelo menos um elemento numérico, no caso numPIDSysError.

Essa ação solicita a operação de verificação de erro do *sistema host*. Alguns sistemas podem exigir futuramente mais parâmetros para outros tipos de verificação de erro, e por isso o parâmetro dessa função é um vetor. O valor para a chamada de erro padrão é "0". O retorno dessa função é a quantidade de erros impeditivos encontrados.

O Sistema de Currículos Lattes aceita dois valores: o "0" para a checagem de erro do CNPq, e o número do ano para a checagem de erro relativa à CAPES.

Esta ação é obrigatória para os módulos que atendem aos requisitos da Plataforma Lattes Institucional. Deve ser invocada antes de qualquer envio de dados.

IMC_SETTEXTSTATUS	`procedure SetStatusText(const strPMensagem: string);`

vrtPArgs string com o texto a ser apresentado na barra de status.

Mostra texto na barra de status do container-padrão. O retorno é sempre um valor true.

Essa ação não tem utilidade quando não se está usando container.

IMC_SETPROGRESS	`procedure SetProgress(numPMin, numPMax, numPPos: Word);`

vrtPArgs vetor com valores numéricos mínimo, máximo e posição na barra de progresso, respectivamente.

Atualiza o indicador de progresso na barra de status do container-padrão. O retorno é sempre um valor true.

Essa ação não tem utilidade quando não se está usando container.

IMC_GETSYSINFO	`function SendSysInfo(numPIDParam: Integer; vrtPArgs: OleVariant): OleVariant;`

vrtPArgs um OleVariant qualquer.

Essa ação permite com que sejam criadas ações específicas para cada *sistema host*. Assim, cada sistema da Plataforma Lattes disponibilizará ações relativas às necessidades que módulos compatíveis possam acionar.

O retorno dessa função é um OleVariant, dessa forma, qualquer valor pode ser retornado ao módulo.

A.3.3 Ações específicas para módulos do Currículo Lattes

O Sistema de Currículos Lattes disponibiliza, além das ações genéricas, as ações abaixo para seus módulos. É importante que os desenvolvedores tenham em mente a seguinte interface:

```
function SendSysInfo(numPIDParam: Integer; vrtPArgs: OleVariant): OleVariant;
```

Onde:

- numPIDParam: identificador da ação requerida;
- vrtPArgs: parâmetros da ação;
- result: resultado da execução da ação.

Identificadores	Função protótipos
IMC_CLOSEALLFORMS	`function CloseAllForms: Boolean;`
vrtPArgs sempre unassigned. Fecha todos os formulários abertos no currículo. Retorna true se bem-sucedido.	
IMC_SENDCURRICULUM	`function SendCurriculum(bolPApagaArqEnviado, bolPMostrarRecibo: Boolean): Boolean;`
vrtPArgs vetor com dois valores booleanos: bolPApagaArqEnviado e bolPMostrarRecibo. Envia o currículo para o CNPq. bolPApagaArqEnviado: Informa se o arquivo deve ser apagado ou não do diretório de trabalho logo após o envio do currículo. bolPMostrarRecibo: Informa se deve ser mostrado o recibo de envio ao término do envio do currículo. NOTA: Para os sistemas da Plataforma Lattes Institucional, esta ação deve ser executada sempre antes do envio do currículo para a instituição.	
IMC_IMPORTCURRICULUM	`function ImportCurriculum: Boolean;`
vrtPArgs sempre unassigned. Chama a tela de importação de currículos Lattes. Retorna true caso o usuário tenha resolvido realmente fazer uma importação, e ela seja bem-sucedida.	
IMC_SAVECURRICULUM	**function** SaveCurriculum: **Boolean**;
vrtPArgs sempre unassigned. Salva o currículo do usuário conectado no disco. Volta true caso a gravação tenha sido realizada.	

IMC_SAVEXMLCURRICULUM	**function** SaveXMLCurriculum(**const** strPTargetFile: **String**): **Boolean**;
vrtPArgs sempre string. Salva o currículo do usuário conectado no formato XML no disco. Volta true caso a gravação tenha sido realizada. O parâmetro deve ser o nome do arquivo alvo a ser gerado.	
IMC_GETNRO_ID_CNPQ	**function** GetNRO_ID_CNPQ: **string;**
vrtPArgs sempre unassigned. Retorna o NRO_ID_CNPQ – número de identificação de pesquisador – do usuário conectado. Caso o usuário não esteja conectado, o valor será uma string nula.	
IMC_GETCURRICULUMDATE	**function** GetCurriculumDate: **Double;**
vrtPArgs sempre unassigned. Obtém a data do currículo do pesquisador conectado. A data é apresentada num formato de número flutuante com a seguinte estrutura: a parte inteira identifica a quantidade de dias decorridos desde 30/12/1899; a parte fracionária desse valor é uma fração das 24 horas do dia especificado. Exemplo: 0 30/12/1899 12:00 am 2.75 1/1/1900 6:00 pm -1.25 29/12/1899 6:00 am 35065 1/1/1996 12:00 am	
IMC_GETGETVERSION	**procedure** GetVersion(**var** strPVersion, strPBuild: **string**);
vrtPArgs sempre unassigned. Retorna a versão e o build do Sistema de Currículos Lattes. Esse valor é apresentado num vetor 'OleVariant' seguindo a ordem versão-build.	

APÊNDICE B
Construção de um framework sobre a API InterLattes

Neste apêndice procuramos mostrar como construir um módulo inteiramente baseado na API do driver de tecnologia. Como havíamos ressaltado no Capítulo 5, a construção de módulos não deve ser feita diretamente sobre essa API, mas sim sobre um framework que abstraia dos módulos a API do driver. Entretanto, é importante mostrar um exemplo de uso da API para permitir que os desenvolvedores entendam a API e assim possam montar seus frameworks abstratos. Esse exemplo foi baseado na ferramenta Delphi da Borland.

B.1 Pré-requisitos

Nesta seção são apresentados os pré-requisitos necessários para todos os interessados em desenvolver módulos para a Plataforma Lattes utilizando o InterLattes.

B.1.1 Perfil da equipe

- Necessário:
    - conhecimento de alguma linguagem de programação usada no Windows;
    - bom conhecimento sobre DLL.
- Desejável:
    - conhecimento sobre programação orientada a objetos;
    - manipulação de objetos COM.

B.1.2 Infra-estrutura

- Hardware:
    - qualquer computador em que rodem os programas do Lattes e também um ambiente de programação capaz de gerar DLL.
- Software:
    - uma linguagem de programação que gere DLL ou que possa ser acessada por uma.

B.2 Passos para a criação de um módulo

A construção de módulos no InterLattes segue os seguintes passos:

1. criar um projeto que gere uma DLL;
2. preencher todos os atributos da estrutura de retorno “TModuleInfo”;
3. exportar o método “GetModuleInfo” que retorne o ponteiro para uma instância estática de “TModuleInfo”;
4. implementar métodos de gerenciamento;
5. implementar métodos de comunicação;
6. registrar o módulo no para o ambiente do InterLattes.

B.3 Criando um projeto que gere uma DLL

No Delphi, crie um projeto para DLL e inclua as units “ugstInterLattes” e “ulatInterLattes”, como mostra o exemplo abaixo.

```
library ModuloExemploProcedural;

uses
  SysUtils,
  Classes,
  ActiveX,
  ugstInterLattesPlugIn in 'ugstInterLattesPlugIn.pas',
  ulatInterLattesPlugin in 'ulatInterLattesPlugin.pas';

{$R Icones.res}

function GetModuleInfo: PModuleInfo; stdcall;
const
  data : TModuleInfo = ( );
begin
  result := @data;

  result.ptrList := nil;
end;

exports
  GetModuleInfo index 1;

end.
```

Vale ressaltar alguns comentários sobre certos detalhes que aparecerem neste trecho do código. Como pode ser notado, foi exportada a API ‘GetModuleInfo’ com o índice 1. Esta API retorna um ponteiro para ‘PmoduleInfo’, conforme a especificação. Observe que foi colocada uma diretiva {$R Icones.res} que serve para incluir, na parte de recursos da DLL, figuras que serão utilizadas pelo InterLattes. Cabe salientar que esse arquivo de recurso também poderia ser utilizado para outras finalidades.

B.4 Preenchendo todos os atributos da estrutura de retorno "TModuleInfo"

A API "GetModuleInfo" retorna um ponteiro para "TModuleInfo" que deve ser preenchido conforme a especificação. Abaixo é apresentado um exemplo de preenchimento dessa estrutura.

```
function ILat_ModuleIcon: IPictureDisp;
begin
  result := ReadPictureFromInstance('APPICON');
end;

function GetModuleInfo: PModuleInfo; stdcall;
const
  data : TModuleInfo = (
    numVersion   : 1000;
    strSystem    : 'CUR';
    strName      : 'Exemplo procedural 1';
    dllModuleIcon: ILat_ModuleIcon;
  );
begin
  result := @data;

  result.ptrList := nil;
end;
```

Note que o método "ILat_ModuleIcon" utiliza a função "ReadPictureFromInstance" e que o parâmetro passado para essa função está definido no arquivo de recurso "Icones.res".

Nesta etapa do processo, já temos um módulo InterLattes, mas para que ele tenha alguma funcionalidade será necessário implementar os procedimentos que retornem ponteiros para formulários (TFormNodo). Cada formulário estará representando um objeto para o InterLattes que poderá ou não conter links na aplicação host. O Ponteiro retornado por esses procedimentos deve ser amarrado ao atributo "ptrList" de "TmoduleInfo" em forma de lista com terminador nulo, de maneira que o primeiro será o formulário 0 (zero), o segundo 1 (um) e assim por diante. Veja abaixo como fica essa amarração de formulários.

```
  result.ptrList := PegaMetaInfoPrimeiraTela();
  result.ptrList.ptrNext := PegaMetaInfoSegundaTela();
  result.ptrList.ptrNext.ptrNext := nil;
```

Os procedimentos "PegaMetaInfoPrimeiraTela()" e "PegaMetaInfoSegundaTela()" retornam ponteiros para "TformNodo" e, da mesma forma como "GetModuleInfo", apontam para uma variável estática definida dentro deles. Veja abaixo.

```
function PegaMetaInfoPrimeiraTela: PFormNodo;
const
  bolLOnce: boolean = false;
  objLPrimeiraTela: TFormNodo = ();
begin
```

```
  if not bolLOnce then
  begin
    bolLOnce := true;

    result := prepFormNodo(@objLPrimeiraTela);

    // Propriedades
    result.strSysMenuPath := '//&Arquivo//Exemplo procedural 1';
    result.strFormCaption := 'Primeiro Exemplo Procedural';
    result.strMenuCaption := 'Primeira';
    result.strDescription := 'Exemplo InterLattes ...';
    result.bolNeedConnect := false;
    result.bolShowStatus  := true;

    // Métodos
    result.dllSmallIcon  := ILat_DllSmallICon;
    result.dllLargeIcon  := ILat_DllLargeICon;
    result.dllFormCreate := ILat_FormCreate;
    result.dllFormFree   := ILat_FormFree;
    result.dllHostSolicitation := ILat_SolicitacaoDoHost;
    result.dllModuleSolicitation := ILat_ModuleSoliticitation;
  end else
    result := @objLPrimeiraTela;
end;
```

Este formulário indica que serão utilizados o container-padrão do InterLattes e também a barra de status deste último. O link no menu do sistema host ficará embaixo do submenu “Arquivo” e se chamará “Exemplo Procedural 1”. O menu interno do container terá o título de “Primeira”, e sua funcionalidade estará descrita na forma de um texto resumido como “Exemplo InterLattes...”.

Agora o InterLattes tem todas as informações de que precisa para utilizar o módulo, mas no momento em que o link no menu do sistema host for invocado não terá métodos que manipulem o objeto, a não ser que sejam implementados todos os métodos de gerenciamento presentes na estrutura “TFormNodo”.

B.5 Implementando os métodos de gerenciamento

Cada formulário possui seis métodos que devem obrigatoriamente ser implementados e que são responsáveis por gerenciar o objeto oferecido pelo módulo.

B.5.1 Métodos que retornam ícones

```
function ILat_DllSmallICon: IPictureDisp;
begin
  result := ReadPictureFromInstance('MENUICON');
end;

function ILat_DllLargeICon: IPictureDisp;
begin
  result := ReadPictureFromInstance('APPICON');
```

```
end;
```

Estes dois métodos são bem simples e servem para informar os ícones a serem associados com o link para o formulário em questão.

B.5.2 Métodos que controlam a instância

```
function ILat_FormCreate(hldPWinApp: HWND): TEvtWndHandle; stdcall;
var
  frmLTela: TfrmPrimeiraTela;
begin
  Application.Handle := hldPWinApp;
  try
    frmLTela := TfrmPrimeiraTela.Create(Application);
    frmLTela.Caption := PegaMetaInfoPrimeiraTela.strFormCaption;
    result := frmLTela.MyCreateWindow;
  except
    result := nil;
  end;
end;

function TfrmPrimeiraTela.MyCreateWindow: HWND;
begin
  result := Self.Handle;
end;
```

O procedimento “ILat_FormCreate” cria a instância do formulário e entrega o ponteiro para o método “MyCreateWindow”. Observe que não está diferente da especificação, embora pareça. Isso porque a maneira como o Delphi manipula objetos é exatamente equivalente à definição utilizada pelo InterLattes, e neste exemplo esta sendo utilizada essa semelhança.

```
procedure ILat_FormFree(hldPHandle: TEvtWndHandle); stdcall;
var
  frmLTela: TfrmPrimeiraTela;
begin
  frmLTela := TfrmPrimeiraTela(TMethod(hldPhandle).Data);
  try
    TMethod(hldPhandle).Data := nil;
    frmLTela.Free;
  except
    // dar mensagen de erro se quiser
  end;
  if Screen.FormCount = 0 then
    Application.Handle := 0;
end;
```

O procedimento “ILat_FormFree” é responsável por liberar a instância e deve se preocupar em fazer outras limpezas de memória se não tiver mais nenhum objeto desse módulo instanciado, pois, nesse caso, o módulo será liberado imediatamente da memória.

B.5.3 Métodos que controlam a comunicação

```
function ILat_SolicitacaoDoHost(hldPHandle: TEvtWndHandle; numPIDParam:
Integer;
                                vrtPArgs: OleVariant): OleVariant; stdcall;
var
  frmLTela: TfrmPrimeiraTela;
begin
  frmLTela := TfrmPrimeiraTela(TMethod(hldPhandle).Data);
  result := frmLTela.SolicitacaoDoHost(numPIDParam, vrtPArgs);
end;
```

Toda vez que o InterLattes precisar de informações ou necessitar disparar uma ação sobre a instância do formulário, ele o fará através desse procedimento. Aqui, por uma questão de organização de código, o tratamento foi transferido para um método da própria instância.

```
procedure ILat_ModuleSoliticitation(hldPHandle: TEvtWndHandle; mthPParams:
TParamMethod); stdcall;
var
  frmLTela: TfrmPrimeiraTela;
begin
  frmLTela := TfrmPrimeiraTela(TMethod(hldPhandle).Data);
  frmLTela.ExecHostFunc := mthPParams;
end;
```

Este procedimento é chamado uma vez para cada instância, no momento imediatamente seguinte à instanciação do formulário. Como o objetivo deste método é apenas passar o ponteiro de um método para que o módulo futuramente possa fazer requisições ao InterLattes, a única ação requerida aqui é guardar o ponteiro para este método na instância do formulário. Como esse procedimento é chamado uma única vez, ele pode ser utilizado para fazer algumas iniciações necessárias à instância. Note que isso também pode ser feito no método de create.

B.6 Implementando os métodos de comunicação

Nessa etapa do desenvolvimento do módulo, implementam-se os métodos que irão verdadeiramente utilizar os recursos do InterLattes e que justificam a sua utilização. Os métodos de comunicação permitem ao módulo solicitar informações e operar a base do sistema host, de modo que o módulo passa a ser uma parte integrante desse sistema.

B.6.1 Ações solicitadas pelo InterLattes

Abaixo temos a implementação das ações solicitadas pelo InterLattes sobre o módulo utilizado como exemplo.

```
function TfrmPrimeiraTela.SolicitacaoDoHost(numPIDParam: Integer;
  vrtPArgs: OleVariant): OleVariant;
begin
  result := unassigned;
  case numPIDParam of
```

```
    IMM_FORMACTIVATED: Self.OnFormActivated();
    IMM_GETMENUICONS:  Result := Self.GetMenuItems();
    IMM_MENUCLICK:     Result := Self.OnMenuClick(vrtPArgs);
  end;
end;
```

Neste método está implementando o comportamento do procedimento 'ILat_SolicitacaoDoHost', em que é feita a distribuição das ações para outros métodos da classe.

```
procedure TfrmPrimeiraTela.OnFormActivated;
begin
  Visible := true;
end;

function TfrmPrimeiraTela.GetMenuItems: OleVariant;
var
  intLImage: IPictureDisp;
  numLIndex: integer;
  strLCaption: string;
begin
  result := varArrayCreate([0, imlCMenuIcons.Count - 1], varVariant);

  strLCaption := '';
  for numLIndex := 0 to imlCMenuIcons.Count - 1 do
  begin
    case numLIndex of
      0: strLCaption := 'Item 1';
      1: strLCaption := 'Item 2';
    end;
    GetImageListPicture(imlCMenuIcons.Handle, numLIndex, intLImage);
    result[numLIndex] := varArrayOf([strLCaption, intLImage]);
  end;
end;

function TfrmPrimeiraTela.OnMenuClick(numPItem: integer): boolean;
begin
  edtCCampo.Text := IntToStr(numPItem);
  result := true;
end;
```

Os métodos acima mostram como implementar o tratamento para as ações disparadas do servidor.

B.6.2 Ações solicitadas pelo módulo

São inúmeras as ações possíveis de serem disparadas pelo módulo. Nesta seção, será mostrado um exemplo de cada um dos tipos de solicitação existentes: solicitação genérica e solicitação específica do sistema host.

Abaixo segue um exemplo de solicitação genérica.

```
function TfrmPrimeiraTela.GetDatabase(): Database;
begin
  if assigned(ExecHostFunc) then
```

```
    Result := IDispatch(ExecHostFunc(IMC_GETDATABASE, unassigned)) as
Database
  else
    Result := nil;
end;
```

Aqui se tem a implementação necessária para o método responsável pelo tratamento das solicitações específicas do sistema host. Note que ele é implementado como uma ação genérica, mas os parâmetros são determinados pelo sistema host. Observe que foi feito um tratamento no parâmetro da ação de modo a garantir que o parâmetro seja um vetor cujo primeiro parâmetro seja o identificador da ação cliente.

```
function TfrmPrimeiraTela.SendSysInfo(numPIDParam: Integer; vrtPArgs:
OleVariant): OleVariant;
var
  vrtLParams: OleVariant;
  numLIndex: integer;
begin
  if assigned(ExecHostFunc) then
  begin
    if VarIsArray(vrtPArgs) then
    begin
      vrtLParams := varArrayCreate(
            [0, VarArrayHighBound(vrtPArgs, 1) + 1], varVariant);
      vrtLParams[0] := numPIDParam;
      for numLIndex := 0 to VarArrayHighBound(vrtPArgs, 1) do
        vrtLParams[numLIndex + 1] := vrtPArgs[numLIndex];
    end else if VarIsEmpty(vrtPArgs) then
      vrtLParams := VarArrayOf([numPIDParam])
    else
      vrtLParams := VarArrayOf([numPIDParam, vrtPArgs]);

    Result := ExecHostFunc(IMC_GETSYSINFO, vrtLParams);
  end else
    Result := unassigned;
end;
```

Finalmente, nessa etapa tem-se a implementação de um evento definido no sistema host para o qual este módulo foi desenvolvido.

```
function TfrmPrimeiraTela.CloseAllForms: Boolean;
begin
  if assigned(ExecHostFunc) then
    Result := SendSysInfo(IMC_CLOSEALLFORMS, unassigned)
  else
    Result := False;
end;
```

B.7 Registrando o módulo no InterLattes

Embora todo esse trabalho tenha sido executado, o módulo não irá aparecer como um módulo do InterLattes se ele não for registrado corretamente. O registro de um módulo InterLattes é simples e direto, mas pode variar um pouco dependendo do sistema host. O CV-

Lattes oferece a capacidade de criar links na barra de ferramentas e na tela de importação; já o Grupo de Pesquisa oferece apenas as opções consideradas padrão. Novas versões desses sistemas poderão oferecer outras possibilidades, e o desenvolvedor do módulo deverá se interar delas se quiser utilizar novos recursos.

Abaixo é apresentado um template de arquivo de registro do Windows que pode ser utilizado como referência para instalação de módulos nos sistemas da Plataforma Lattes.

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Lattes\<sistema host>\InterLattes\Modulos\
<nome da DLL>]
"DiretorioInstalacao"="<caminho completo da localização da DLL>"
"Versao"="<número da versão do sistema>"
"MostraBotaoBarra"="<número do formulário>" // CV-Lattes
"MostraBotaoImportação"="<número do formulário>" // CV-Lattes
```

O Sistema de Currículos Lattes permite que sejam colocados mais dois parâmetros:

- MostraBotaoBarra: valor numérico correspondente ao número do formulário a ser disparado quando do clique no link da barra de ferramentas;
- MostraBotaoImportacao: valor numérico correspondente ao número do formulário a ser disparado quando do clique no link na tela de importação.

APÊNDICE C
Units de conexão ao ambiente InterLattes

Este apêndice apresenta os códigos-fontes mínimos necessários para conectar um módulo ao ambiente InterLattes da Plataforma Lattes. Embora os códigos-fontes estejam em Object Pascal, pode-se facilmente convertê-los para outra linguagem de programação.

C.1 Genérico aos sistemas

```
unit ugstInterLattesPlugIn;

interface

uses Windows, Messages, ActiveX;

type
  TEvtWndHandle = function: HWND of object;

  TParamMethod = function(numPIDParam: Integer; vrtPArgs: OleVariant):
OleVariant of object; stdcall;

  TDllGetIcon = function: IPictureDisp;

  TDllFormCreate = function(hldPWinApp: HWND): TEvtWndHandle; stdcall;

  TDllFormFree = procedure(hldPHandle: TEvtWndHandle); stdcall;

  TDllHostSolicitation = function(hldPHandle: TEvtWndHandle;
                                  numPIDParam: Integer;
                                  vrtPArgs: OleVariant): OleVariant;
stdcall;

  TDllModuleSolicitation = procedure(hldPHandle: TEvtWndHandle;
                                     mthPParams: TParamMethod); stdcall;

  PFormNodo = ^TFormNodo;
  TFormNodo = record
    // propriedades
    hldForm       : TEvtWndHandle;
    strSysMenuPath: PChar;            // Referencia no menu do sistema
    strFormCaption: PChar;            // Titulo do form quando ativado
                                       // no container
    strMenuCaption: PChar;            // Caption no menu do container
    strDescription: PChar;            // Caption no menu do container
    bolNeedConnect: Boolean;          // Informa se para chamar este módulo
                                       // é necessário estar conectado
    bolShowStatus : Boolean;          // Mostrar Status
    ptrNext       : PFormNodo;
    // Métodos
    dllSmallIcon         : TDllGetIcon;      // este icone sera usado
                                              // no menu do sistema
    dllLargeIcon         : TDllGetIcon;      // este icone ficara a
                                              // disposição do sistema host
    dllFormCreate        : TDllFormCreate;
    dllFormFree          : TDllFormFree;
```

```
    dllHostSolicitation  : TDllHostSolicitation;
    dllModuleSolicitation: TDllModuleSolicitation;
  end;

  PModuleInfo = ^TModuleInfo;
  TModuleInfo = record
    // propriedades
    numVersion    : Integer;
    strSystem     : PChar;
    strName       : PChar;
    ptrList       : PFormNodo;
    // métodos
    dllModuleIcon: TDllGetIcon;
  end;

  TfnDllGetModuleInfo = function: PModuleInfo; stdcall;

const
  { IMC - InterLattes Messages Container }
  IMC_GETDATABASE      = -1;
  IMC_GOTOFORM         = -2;
  IMC_CLOSE            = -3;
  IMC_GETSYSINFO       = -4;
  IMC_SETTEXTSTATUS    = -5;
  IMC_SETPROGRESS      = -6;
  IMC_ERRORSVERIFY     = -7;
  IMC_DISABLECONTAINER = -8;
  IMC_ENABLECONTAINER  = -9;
  IMC_MODULEKEY        = -10; // Chave do módulo de configuração
  IMC_USER          = 0;

  { IMM - InterLattes Messages Module }
  IMM_GETMENUICONS   = -1;
  IMM_MENUCLICK      = -2;
  IMM_FORMACTIVATED  = -3;
  IMM_USER           = 0;

  function prepFormNodo(ptrPValue: PFormNodo): PFormNodo;
  function ReadPictureFromInstance(const strPPictureName: string):
IPictureDisp;
  procedure GetImageListPicture(hldPImgList: THandle; idx: integer; out
objPImagem: IPictureDisp);

implementation

const
  cctrl = 'comctl32.dll';

{$EXTERNALSYM ILD_TRANSPARENT}
ILD_TRANSPARENT         = $0001;

{$EXTERNALSYM ImageList_GetIcon}
function ImageList_GetIcon(ImageList: THandle; Index: Integer;
  Flags: Cardinal): HIcon; stdcall; external cctrl name 'ImageList_GetIcon';

function prepFormNodo(ptrPValue: PFormNodo): PFormNodo;
begin
  ptrPValue.hldForm := nil;
  ptrPValue.ptrNext := nil;
  result := ptrPValue;
end;
```

```
function ReadPictureFromInstance(const strPPictureName: string):
IPictureDisp;
var
  PictureDesc: TPictDesc;
  hldLBitmap: HBitmap;
begin
  result := nil;
  hldLBitmap := LoadBitmap(hInstance, PChar(strPPictureName));
  if hldLBitmap <> 0 then
  begin
    FillChar(PictureDesc, sizeof(PictureDesc), 0);
    PictureDesc.cbSizeOfStruct := SizeOf(PictureDesc);
    PictureDesc.picType := PICTYPE_BITMAP;
    PictureDesc.hbitmap := hldLBitmap;
    OleCreatePictureIndirect(PictureDesc, IPicture, true, Result)
  end;
end;

procedure GetImageListPicture(hldPImgList: THandle; idx: integer; out
objPImagem: IPictureDisp);
var
  PictureDesc: TPictDesc;
begin
  FillChar(PictureDesc, sizeof(PictureDesc), 0);
  PictureDesc.cbSizeOfStruct := SizeOf(PictureDesc);
  PictureDesc.picType := PICTYPE_ICON;
  PictureDesc.hIcon := ImageList_GetIcon( hldPImgList, Idx,
ILD_TRANSPARENT);
  OleCreatePictureIndirect(PictureDesc, IPicture, true, objPImagem);
end;

end.
```

C.2 Específico ao Sistema CV-Lattes

```
unit ulatInterLattesPlugin;

interface

uses ugstInterLattesPlugIn;

const
  { IMC - InterLattes Messages Container }
  IMC_GETNRO_ID_CNPQ     = IMC_USER + 1;
  IMC_SENDCURRICULUM     = IMC_USER + 2;
  IMC_IMPORTCURRICULUM   = IMC_USER + 3;
  IMC_SAVECURRICULUM     = IMC_USER + 4;
  IMC_CLOSEALLFORMS      = IMC_USER + 5;
  IMC_GETCURRICULUMDATE  = IMC_USER + 6;
  IMC_SETOKIMPORT        = IMC_USER + 7;
  IMC_SETNRO_ID_CNPQ     = IMC_USER + 8;
  IMC_GETGETVERSION      = IMC_USER + 9;

  IMC_USER_SUBSYSTEMS    = IMC_USER + 10000;

implementation

end.
```