

UNIVERSIDADE FEDERAL DE SANTA CATARINA
Curso Superior de Ciências da Computação

INTEGRAÇÃO ENTRE HTML5 E JSF 2.0
EM APLICAÇÕES WEB OFFLINE

ROBERTO JORGE HADDOCK LOBO FILHO

Florianópolis - SC, 2010.

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
Curso Superior de Ciências da Computação

INTEGRAÇÃO ENTRE HTML5 E JSF 2.0
EM APLICAÇÕES WEB OFFLINE

ROBERTO JORGE HADDOCK LOBO FILHO

Trabalho de Conclusão de Curso
apresentado à UFSC como parte dos
requisitos para obter grau de Bacharel em
Ciências da Computação.

Orientador: Leandro José Komosinski.

Florianópolis – SC, junho de 2010.

ROBERTO JORGE HADDOCK LOBO FILHO

INTEGRAÇÃO ENTRE HTML5 E JSF 2.0
EM APLICAÇÕES WEB OFFLINE

Trabalho de Conclusão de Curso
apresentado à UFSC como parte dos
requisitos para obter grau de Bacharel em
Ciências da Computação.

Orientador:

Leandro José Komosinski

Banca Examinadora:

Rodrigo Carvalho Machado

Frank Augusto Siqueira

AGRADECIMENTOS

Meus profundos agradecimentos ao meu pai e minha mãe pelo suporte nos mais diversos aspectos durante toda a minha vida contribuindo para que eu chegasse até aqui.

Ao professor Leandro José Komosinski por ter aceitado, mesmo com o pouco tempo disponível, orientar este trabalho.

Ao meu chefe e colega de trabalho Rodrigo Carvalho Machado e ao professor Frank Augusto Siqueira, por aceitarem participar da banca examinadora.

A todos os professores e demais colaboradores da UFSC que se esforçam em propiciar o ensino com excelência.

Aos amigos e colegas com quem também aprendi e que deixarão saudades.

Ofereço também um agradecimento especial aos conhecidos e anônimos que contribuíram de alguma forma para o fornecimento de fontes de pesquisa e materiais bibliográficos para o desenvolvimento deste trabalho.

Resumo

A constante concepção de novas tecnologias e conceitos de desenvolvimento de software e de sua compatibilidade com as tecnologias já existentes é um dos grandes desafios na área de ciências da computação. Neste contexto, a utilização da WEB – que originalmente foi concebida com a intenção e capacidade de exibição de documentos hipertextos – como interface gráfica ao usuário de sistemas online têm gerado grandes dificuldades e limitações. Objetiva-se com este trabalho desenvolver uma pesquisa sobre as possibilidades de compatibilização entre duas tecnologias modernas relacionadas ao desenvolvimento destes sistemas, tendo como estudo a criação de aplicações web utilizando implementações provenientes da especificação JSF 2.0 – tecnologia padrão do JavaEE – que possam ser disponibilizadas em modo offline – sem que o usuário necessite estar conectado à internet - de acordo com as novas funcionalidades oferecidas pelo HTML 5, assumindo toda complexidade de comportamentos e interações que estas interfaces gráficas abrangem.

Palavras-chave: Desenvolvimento WEB. JSF. Aplicações offline.

Abstract

The constant conception of new technologies and concepts of development of software and its compatibility with the existing technologies is one of the great challenges in the area of computer sciences. In this context, the WEB interface - that originally was conceived with the intention of handling hipertext documents – being used as graphical interface to the user of systems online has generated great difficulties and limitations. This work objective is to develop a research on the compatibility possibilities between two modern technologies related to the development of such interfaces: the creation of web applications using implementations proceeding from the JSF 2.0 specification - standard JavaEE technology - that could be accessed offline - without the need of an internet connection - in accordance with the new functionalities offered for HTML 5, assuming all complexity of behaviors and interactions that these graphical interfaces enclose.

Keywords: WEB Development. JSF. Offline applications.

Lista de Ilustrações

Figura 1: Modelo Cliente-Servidor.....	19
Figura 2: Modelo Cliente Servidor.....	20
Figura 3: Particionamento entre cliente e servidor.....	28
Figura 4: Diagrama de sequência ilustrando os diferentes possíveis comportamentos para diferentes requisições.....	41
Figura 5: Fases do tratamento padrão de requisições pelo JSF 2.0.....	42
Figura 6: Aplicação web JSF 2.0 segundo arquitetura proposta pela especificação JavaEE.....	49
Figura 7: Aplicação web offline com o JSF 2.0 no cliente.....	50
Figura 8: Lógica específica da aplicação.....	51
Figura 9: Separação de lógica da aplicação, representando as porções de lógica executadas no servidor e no cliente offline.....	52
Figura 10: Duas diferentes implementações para a interface dos serviços oferecidos por um EJB, uma pelos processamentos no servidor e a outra pelas ações a serem realizadas no cliente quando offline.....	54
Figura 11: Acesso e armazenamento de dados necessário também pelo cliente offline.....	55
Figura 12: Utilização de plug-in para execução de código Java do cliente offline.....	58
Figura 13: Utilização de Applet Java para execução de código Java do cliente offline.....	60
Figura 14: Motor de sincronização de dados.....	62
Figura 15: Impacto de processamentos relacionados à lógica de negócio sobre dados de uma aplicação.....	63
Figura 16: A não equivalência de dados armazenados no cliente e no servidor, demonstrando a necessidade de armazenamento no cliente de dados já processados no servidor e da execução de processamentos no servidor relativos aos dados coletados no cliente.....	65

Sumário

RESUMO	5
ABSTRACT	6
LISTA DE ILUSTRAÇÕES	7
1 – INTRODUÇÃO	11
1.1 APRESENTAÇÃO DO TEMA.....	12
1.2 MOTIVAÇÃO DA PESQUISA.....	14
1.3 OBJETIVOS	15
1.4 METODOLOGIA E PROCEDIMENTOS.....	16
1.4.1 <i>Pesquisa bibliográfica</i>	16
1.4.2 <i>Levantamento das características relevantes para solução do problema</i>	17
1.4.3 <i>Análise da viabilidade da compatibilização das tecnologias</i>	17
1.4.4 <i>Desenvolvimento de protótipos de teste</i>	17
1.4.5 <i>Análise dos resultados</i>	17
1.5 <i>Estrutura do Texto</i>	17
2 – DESENVOLVIMENTO DE APLICAÇÕES WEB.....	18
2.1 MODELO CLIENTE-SERVIDOR	19
2.1.1 <i>Considerações do modelo cliente-servidor</i>	20
2.2 CLIENT-SIDE SCRIPTING.....	21
2.3 COMUNICAÇÃO CLIENTE-SERVIDOR.....	22
2.3.1 <i>Comunicação síncrona</i>	22
2.3.2 <i>Comunicação assíncrona</i>	22
2.3.3 <i>AJAX</i>	23
2.3.4 <i>Considerações da comunicação entre cliente e servidor</i>	23
2.4 PARTICIONAMENTO DE APLICAÇÕES ENTRE CLIENTE E SERVIDOR.....	23
2.4.1 <i>Mandatário ao servidor</i>	25
2.4.2 <i>Considerações para o servidor</i>	26
2.4.3 <i>Considerações para o cliente</i>	27
2.5 APLICAÇÕES WEB OFFLINE.....	28
2.5.1 <i>Requisitos para disponibilização offline de uma aplicação web</i>	29
3 – HTML5	31
3.1 HISTÓRIA DO HTML	31
3.2 O HTML5 E O DESENVOLVIMENTO DE APLICAÇÕES WEB.....	32

3.2.1	<i>Funcionalidades básicas abordadas pelo HTML5</i>	33
3.3	DESENVOLVIMENTO DE APLICAÇÕES WEB OFFLINE COM O HTML5	34
3.4	CACHEAMENTO DE RECURSOS	34
3.4.1	<i>Manifesto</i>	35
3.4.2	<i>Eventos do cacheamento</i>	35
3.4.3	<i>Caches de aplicação</i>	35
3.4.4	<i>Grupos de caches de aplicação</i>	36
3.5	WEB STORAGE	37
3.5.1	<i>O atributo sessionStorage</i>	37
3.5.2	<i>O atributo localStorage</i>	37
3.5.3	<i>A interface Storage</i>	37
3.6	WEB SQL DATABASE	38
3.6.1	<i>Bases de dados</i>	38
3.6.2	<i>Métodos de acesso</i>	38
3.7	COMPATIBILIDADE DO HTML5	38
4	– JSF 2.0	39
4.1	CICLO DE VIDA DE PROCESSAMENTO DE REQUISIÇÕES	40
4.2	CENÁRIOS DE PROCESSAMENTO DE REQUISIÇÕES	40
4.2.1	<i>Requisição Não-Faces gera Resposta Faces</i>	41
4.2.2	<i>Requisição Faces gera Resposta Faces</i>	42
4.2.3	<i>Requisição Faces gera Resposta Não-Faces</i>	43
4.3	FASES DO CICLO DE VIDA DO PROCESSAMENTO DE REQUISIÇÃO	43
4.3.1	<i>Restauração da visualização</i>	43
4.3.2	<i>Aplicação dos valores da requisição</i>	43
4.3.3	<i>Validação do processamento</i>	44
4.3.4	<i>Atualização dos valores do modelo</i>	44
4.3.5	<i>Invocação da aplicação</i>	44
4.3.6	<i>Renderização da resposta</i>	45
4.4	PROCESSAMENTO DE EVENTOS	45
4.5	CONCEITOS QUE IMPACTAM NAS FASES DO CICLO DE VIDA	45
4.5.1	<i>AJAX</i>	46
4.5.2	<i>Comportamento de componentes</i>	46
5	– INTEGRAÇÃO DO JSF 2 E HTML5 OFFLINE	47

5.1 DESAFIOS INERENTES ÀS TECNOLOGIAS EM QUESTÃO	47
5.1.1 Relativos ao HTML5 e aos navegadores web	47
5.1.2 Relativos ao JSF 2.0 e ao modelo cliente-servidor da web	48
5.1.3 Problemática da compatibilização	48
5.2 CÓDIGO NECESSÁRIO PARA EXECUÇÃO OFFLINE NO CLIENTE	48
5.2.1 JavaServer Faces	49
5.2.2 Particionamento da lógica de negócios entre cliente e servidor	50
5.2.3 Código de lógica de negócios plausível ao cliente	53
5.2.4 Dados da aplicação	54
5.3 EXECUÇÃO DE CÓDIGO INDEPENDENTE DO SERVIDOR NO CLIENTE	56
5.3.1 Interface gráfica e o cacheamento de recursos	56
5.3.2 Código Java da aplicação	57
5.3.3 Plug-ins para navegadores	57
5.3.4 Java Applets	59
5.3.5 JavaScript	61
5.4 INTEGRIDADE E SINCRONIZAÇÃO DE DADOS	62
6 – CONCLUSÕES	66
6.1 INCOERÊNCIA COM CULTURA DE ALTA DISPONIBILIDADE DE SERVIÇOS	67
6.1.1 Inflexibilidade no particionamento de lógica	67
6.1.2 Desafios inerentes às tarefas de sincronização	68
6.2 NECESSIDADE DE COMPLEXOS MECANISMOS ESPECÍFICOS A CADA APLICAÇÃO	68
6.3 DEPENDÊNCIAS NA MÁQUINA DO CLIENTE	69
6.4 NÃO ATENDIMENTO NATIVO A RIA	70
6.5 CONSIDERAÇÕES FINAIS	71
6.6 TRABALHOS FUTUROS	72
7 – REFERÊNCIAS	73
APÊNDICE A - ARTIGO	75

1 – INTRODUÇÃO

O desenvolvimento de aplicações web avança lado a lado com a evolução dos conceitos que propõem novas abordagens de utilização das tecnologias existentes, novas demandas de uso e novos desafios originários da utilização do que já existe como base tecnológica e conceitual para idealização de novas soluções.

A constante concepção de novas soluções permite que novos patamares de complexidade sejam aos poucos simplificados e estabelecidos, possibilitando a ampliação dos limites das tecnologias existentes e abrindo caminho para a necessidade de criação de novos patamares. A web, por si, foi originalmente concebida para a exibição, organização e compartilhamento de documentos hipertexto e, com sua ampla adoção, junto ao crescimento da internet, gerou-se a demanda de sua evolução.

Ao que uma fundação tecnológica é concebida como plataforma, permiti-se a utilização desta para a resolução de problemas de maior complexidade. Do simples fornecimento de páginas estáticas, a web abriu espaço, inicialmente, para gerência de conteúdo dinamicamente gerado, seguidamente provendo a oportunidade de criação de aplicações simples, exibidas estaticamente no cliente, não diferindo de páginas HTML simples. A demanda, então, pelo desenvolvimento de aplicações RIA (Aplicações ricas ao usuário), ou seja, que proovessem funcionalidades avançadas e rica experiência de uso, naturalmente estabeleceu-se, sem possuir nenhuma fundação tecnológica madura, gerando então a necessidade de criação de novas soluções. Este processo traz a criação de novas tecnologias, porém, surge uma grande problemática ligada a questões de compatibilidade. Cada problema atacado estabelece um número de soluções que são naturalmente selecionadas com o tempo e filtradas principalmente de acordo com a aceitação e adoção de seu uso.

Mudanças de paradigmas decorrentes da adaptação de novos conceitos e demandas geram novas necessidades que nem sempre podem ser atendidas com uma base tecnológica já existente. Estas bifurcações entre as soluções conceituais de uma tecnologia e novas necessidades e demandas exige um estudo para determinar a possibilidade de um trabalho de compatibilização. Ao levar em consideração o fato de que soluções propostas podem naturalmente divergir em múltiplas vertentes tecnológicas, o problema de compatibilização se estende ainda

mais. De um ponto de vista mercadológico, a sobrevivência de uma solução não somente pode ser diretamente atrelada à sua capacidade de simplificação do problema que a mesma se propõe a resolver - de acordo com a mitigação das necessidades e requisitos que o problema impõe – mas também à sua adoção. Isto significa que, assim como a compatibilidade entre diferentes versões de uma solução é importante, a capacidade de adaptação de uma tecnologia para o atendimento de novas necessidades mercadológicas pode – mesmo que indiretamente – ser fundamental para sua continuidade.

1.1 Apresentação do tema

A WEB (também conhecida como World Wide Web ou WWW) é um sistema de documentos em hipermídia que são interligados e executados na Internet. Segundo BERNERS-LEE (1989), o intento original do desenvolvimento da web foi tornar mais fácil o compartilhamento de documentos de pesquisas. Inicialmente, o serviço oferecido pela Web disponibilizava hipertextos e a possibilidade da realização de ligações unidirecionais, na forma de referências.

Na época, a maior parte dos sistemas de informação prevalentes utilizava uma estrutura hierárquica fixa para organizar informação, como por exemplo nos grupos de notícias Usenet (do inglês Unix User Network). Tim Berners-Lee descreu, em sua proposta inicial do sistema que viria a se tornar a web, a existência de diversos problemas quanto à perda de informações e quanto à falta de flexibilidade de um modelo hierárquico para a organização de dados para a modelagem do mundo real.

Desde a concepção da web, dada publicamente e oficialmente em 6 de agosto de 1991, sua popularidade cresce aceleradamente acompanhando o crescimento da internet. Seu rápido crescimento, que se mantém continuamente acelerando, age como um meio global de transformação da economia, da veiculação de informação e de comercialização de bens e serviços. Segundo (NETO, 2001), a importância econômica da web cresce junto à sua popularidade.

Atendendo à demanda gerada pela popularização da web - e por sua crescente importância econômica – a geração de páginas web de conteúdo

dinâmico se tornou uma necessidade, criando páginas web que podem ser customizadas de acordo com cada cliente e com o contexto da navegação. Com isso, a capacidade de geração de páginas dinamicamente no servidor permitiu que sistemas pudessem ser criados e que a web seja utilizada como uma interface gráfica, e a possibilidade de execução de linguagens de script no cliente juntos ao desenvolvimento de métodos de comunicação assíncrona com o servidor permitiram que estas interfaces gráficas deixassem de ser simples seqüências de páginas estáticas.

Os primeiros sistemas web de geração de conteúdo dinâmico geralmente objetivavam a simplificação da manutenibilidade e expansibilidade de grandes sistemas de informação. A evolução destes conceitos – e das tecnologias envolvidas – permite que aplicações web tenham cada vez mais capacidade de realizar as mesmas funcionalidades que aplicações desktop nativas.

Neste contexto, diversos frameworks foram criados para facilitar e possibilitar o desenvolvimento de aplicações web, cada um utilizando um conjunto de abordagens para simplificar as complexidades provenientes da adaptação da web – que inicialmente foi concebida com outros propósitos – para as novas demandas e novos requisitos funcionais. O conseqüente amadurecimento do desenvolvimento de aplicações web possibilita a migração de serviços para a web, tornando o conceito de instalação de software em uma máquina desktop obsoleto neste contexto. Esta mudança de paradigma permite que o acesso a serviços e seus respectivos dados seja independente de um computador específico.

O JavaServer Faces 2.0 é a última versão do mais maduro framework de desenvolvimento de aplicações web contido na especificação Java EE 6 (Java Enterprise Edition). Ele é projetado para aliviar significativamente o encargo de escrita e manutenção de interfaces gráficas de aplicativos que rodam em um servidor de aplicativos Java e que geram suas interfaces de usuário dinamicamente para um cliente-alvo.

A busca da continuidade da expansão das capacidades de aplicações web tem, porém, uma grande limitação em comparação a aplicações web. A utilização pura da web como plataforma para aplicações requer que o acesso à internet esteja disponível para que o usuário possa acessar uma aplicação, de acordo com as

próprias características da web. Usuários de aplicações web típicas são capazes de utilizar estas aplicações somente quando conectados à internet.

A especificação do HTML5, linguagem de marcação utilizada para produção de páginas web, propõe uma das abordagens que estão sendo desenvolvidas para diminuir esta limitação, visando permitir a criação de aplicações web que forneçam a capacidade de execução – pelo menos parcial – enquanto offline, sem a necessidade de uma conexão com a internet.

Este trabalho busca, diante deste contexto, estudar a possibilidade de compatibilização do JSF 2.0 – framework de desenvolvimento de aplicações web cuja grande parte de seu processamento ocorre no servidor – com as facilidades oferecidas pela especificação do HTML5 de criação aplicação web offline – que requer execução no cliente do código disponibilizado.

Ante a esta proposta, surgem alguns questionamentos: a compatibilização destas tecnologias é possível? Quais são as possíveis alternativas para possibilitar a execução no cliente de porções de código que no JSF são originalmente processados no servidor? Quais seriam os processamentos do servidor cuja execução seria necessária no cliente para que uma aplicação desenvolvida com a utilização do JSF pudesse ser executada independentemente do servidor? Como definir e isolar a porção de lógica de negócio da aplicação contida no servidor – levando-se em consideração que existem obrigações do servidor cujo isolamento não é possível - que deveria ser trazida ao cliente?

1.2 Motivação da pesquisa

Ante uma acelerada evolução de conceitos de utilização da web e de diferentes técnicas de desenvolvimento de aplicações neste contexto, junto a seu crescente mercado mundial e importância econômica, necessita-se um estudo e análise sobre as possibilidades de compatibilização entre os diversos conceitos e abordagens a fim de divulgá-las, incentivando a pesquisa e inovação nesta área.

Muito há a ser explorado na área de desenvolvimento de software para a web. Problemas como a adaptação das tecnologias existentes decorrentes do desenvolvimento da web para ampliar a capacidade dos sistemas que podem ser criados neste ambiente, a necessidade de que a compatibilidade das mesmas seja

mantida, tanto em relação às versões anteriores quanto às novas necessidades da indústria de desenvolvimento e dos usuários em geral e a simplificação dos processos e dificuldades envolvidos despertam a grande necessidade de um maior estudo nesta área, que vem revolucionando o mundo empresarial e a própria internet.

A criação de aplicações web desencadeou uma série de mudanças de paradigmas na computação, trazendo funcionalidades que originalmente eram disponíveis somente em aplicações desktop tradicionais para a plataforma web na forma de serviços descentralizados acessíveis de qualquer dispositivo que tenha capacidade de navegação na web. Agora, outra mudança de paradigma propõe a disponibilização de aplicações web em determinados computadores, através de armazenamento local de ambos o código de execução e os dados do usuário, permitindo sua utilização - ao menos parcial - independentemente da existência de uma conexão com a internet.

A busca por meios para compatibilizar a última versão do JavaServer Faces – uma importante framework de desenvolvimento de aplicações web – com as capacidades oferecidas pela especificação do HTML5 de disponibilização de aplicações quando offline, ou seja, em ambientes em que o usuário esteja privado da possibilidade de acesso a internet, pode obter resultados significativos nesta área que ainda se encontra em fase longe da maturação.

Há então a necessidade da pesquisa, análise e comparação da possibilidade de compatibilização entre estas tecnologias e conceitos, em termos da viabilidade segundo os princípios básicos do funcionamento do JSF, de complexidade, dos possíveis impactos e requisitos ao usuário, e dos benefícios em relação às diferentes possíveis soluções.

1.3 Objetivos

Este trabalho tem como principal objetivo a pesquisa sobre a aplicabilidade e viabilidade da compatibilização entre o framework JSF 2.0, devido à sua importância no desenvolvimento de aplicações web através da linguagem Java, e as funcionalidades propostas pelo HTML5 de armazenamento local de dados de

execução e de usuário, para permitir a criação de aplicações web que possam ser disponibilizadas e executadas mesmo quando o usuário estiver offline.

Seus objetivos específicos compreendem:

Definir as características compatíveis e incompatíveis do funcionamento do framework JSF em relação ao modelo de disponibilização de aplicações web offline definidos pelas funcionalidades oferecidas pelo HTML5;

Analisar o grau de independência possível entre a aplicação cliente em relação ao servidor, definindo os requisitos para a possibilitação de que aplicações web tradicionais possam ser utilizadas offline junto as obrigações;

Definir o conjunto de possíveis medidas e soluções para a compatibilização destas tecnologias;

Analisar os impactos e a viabilidade das possíveis diferentes alternativas de compatibilização;

Implementar protótipos realizando a prova de conceito sobre a melhor solução encontrada;

1.4 Metodologia e procedimentos

As etapas de realização do projeto compreendem:

Pesquisa bibliográfica;

Levantamento das características relevantes para solução do problema;

Análise da viabilidade da compatibilização das tecnologias;

Desenvolvimento de protótipos de teste;

Análise dos resultados.

1.4.1 Pesquisa bibliográfica

Esta etapa compreende o levantamento de informações necessárias para a compreensão dos problemas relacionados ao desenvolvimento de aplicações para a web e relacionados à disponibilização das mesmas em modo offline, assim como o estudo do funcionamento e das características do HTML5 e do JSF 2.0 – e outras tecnologias envolvidas – com o objetivo de viabilizar o estudo proposto por este trabalho.

1.4.2 Levantamento das características relevantes para solução do problema

O levantamento das características relevantes ao problema é realizado a fim de determinar as possibilidades, as limitações e os desafios estabelecidos pela utilização das tecnologias escolhidas, ou seja, a fim de determinar o cenário e as possíveis abordagens para viabilização da compatibilização proposta.

1.4.3 Análise da viabilidade da compatibilização das tecnologias

Compreende a análise da viabilidade prática de compatibilização do JSF 2.0 para a possibilitação da criação de aplicações web offline de acordo com as limitações tecnológicas impostas pelas características do problema e as possíveis abordagens para resolução dos desafios existentes.

1.4.4 Desenvolvimento de protótipos de teste

O desenvolvimento de protótipos de teste é feito a fim de validar a possibilidade de adaptação das tecnologias estudadas para a resolução do problema.

1.4.5 Análise dos resultados

Os resultados são analisados e interpretados, para a documentação dos mesmos.

1.5 Estrutura do Texto

Os subseqüentes capítulos deste trabalho compreendem os assuntos relacionados ao desenvolvimento de aplicações web, ao HTML 5 e as suas funcionalidades especificadas para a viabilização de criação de aplicações web offline, o framework JavaServer Faces 2.0, a problemática e as possíveis abordagens de integração entre estas tecnologias e as conclusões deste estudo.

2 – DESENVOLVIMENTO DE APLICAÇÕES WEB

Aplicações web é um vago termo que descreve aplicações cuja utilização é disponível através da web, seja através de uma rede - como a Internet ou uma Intranet - ou por acesso local. O termo inclui também qualquer software que é apresentado em um ambiente controlado por um navegador web, ou codificado em linguagens e tecnologias suportadas por estes navegadores.

A demanda do desenvolvimento de aplicações web vem crescendo e acelerando constantemente, conseqüentemente criando uma alta demanda de viabilização e maturação de tecnologias para suporte a este processo. Muitos dos desafios existentes na área provêm do fato que o ambiente web não foi concebido com o intuito de prover base tecnológica alguma para aplicações e para o seu desenvolvimento.

A web foi originalmente concebida com o intento de facilitar o compartilhamento de documentos de pesquisas. O conceito de aplicações com sua execução integral ou parcial em um cliente web é extremamente recente. A web por si teve sua concepção em 1990, a primeira linguagem que permitiu que algum código fosse executado no cliente foi criada em 1995, e somente em 2005 o termo AJAX foi cunhado e seu conceito popularizado. A plataforma web ainda é um conceito muito abstrato.

Este modelo cliente-servidor ainda está em fase de maturação e levanta constantemente novos questionamentos arquiteturais, tecnológicos e conceituais. As possibilidades criadas com sua viabilização e os próprios conceitos necessários que isto seja possível ainda estão sendo moldados, e na verdade, muitos deles certamente ainda nem foram idealizados. Como deve ser realizada e mantida a comunicação entre cliente e servidor? Quais as possíveis abordagens para que novas tecnologias possam ampliar as capacidades de definição e renderização de interfaces gráficas de aplicações web, mantendo a compatibilidade entre navegadores e suas versões antigas? Quais são as conseqüências de longo prazo nos processos de desenvolvimento quanto às diferentes abordagens de particionamento do código entre o cliente e o servidor?

Diferentes abordagens tecnológicas e conceituais desenvolvidas na forma de frameworks propõem respostas às limitações e aos desafios existentes devido à falta de uma base bem definida para desenvolvimento de software para a web.

Neste contexto, tanto os problemas existentes quanto as soluções propostas são muito recentes. Ao contrário do desenvolvimento de softwares nativos, não existem abordagens padronizadas quanto ao desenvolvimento para a web.

"Desenvolvimento web" é um termo amplo que faz referencia a qualquer atividade de desenvolvimento que envolve a web. O termo pode abranger desde a criação de páginas estáticas simples de texto plano até o desenvolvimento de complexas aplicações e serviços baseados na web.

2.1 Modelo cliente-servidor

O modelo cliente-servidor descreve a relação entre dois programas de computador em que um programa - o cliente - faz uma solicitação de serviço de outro programa - o servidor - que responde o pedido. Os clientes, por conseguinte, iniciam sessões de comunicação com os servidores que aguardam e escutam as requisições de entrada.

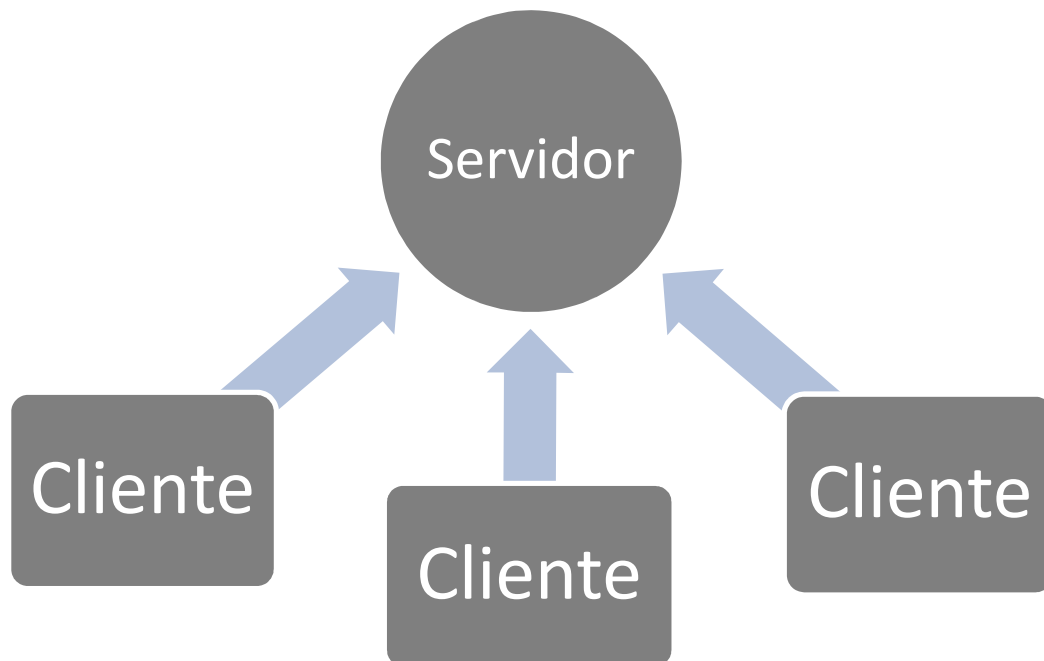


Figura 1: Modelo Cliente-Servidor.

A web foi concebida utilizando a arquitetura cliente servidor. O cliente, comumente chamado de Navegador Web ou browser, interage com um ou mais

servidores através do protocolo HTTP (*HyperText Transfer Protocol*) realizando requisições de recursos e dados. Servidores web, que normalmente – mas não necessariamente - operam sobre uma rede em hardware separado do cliente, ao receberem uma requisição de recurso, ou devolvem um recurso estático ou geram a resposta dinamicamente. Sob a perspectiva do cliente, não existe diferenciação entre ambos os casos, e a resposta será tratada da mesma maneira, ou seja, será renderizada diretamente no cliente ou processada por algum script executado no navegador.

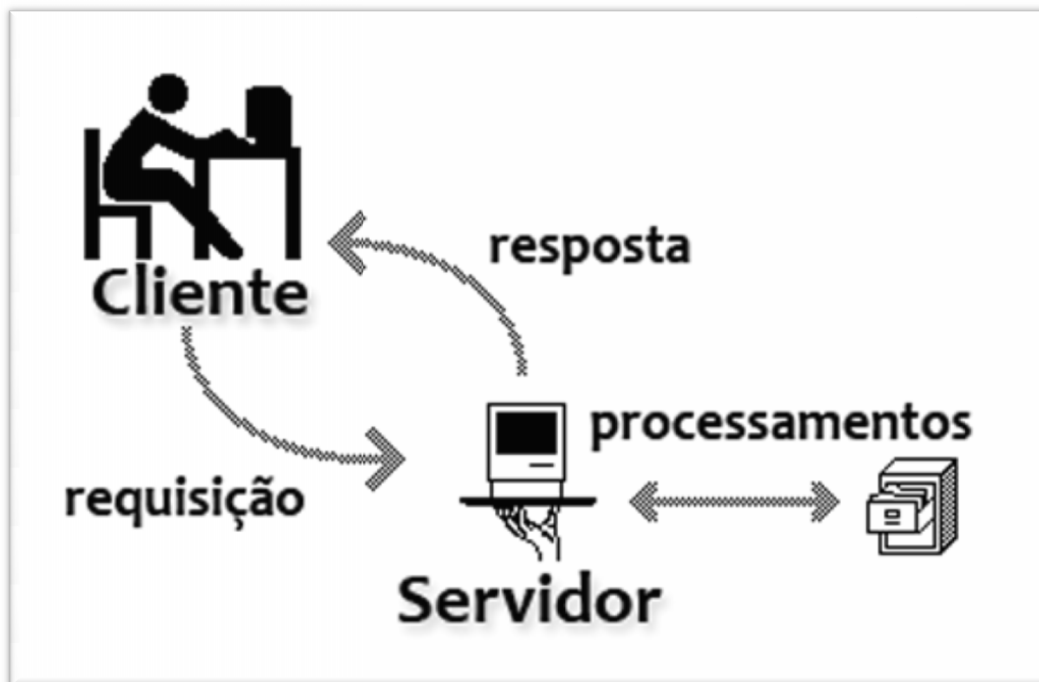


Figura 2: Modelo Cliente Servidor.

2.1.1 Considerações do modelo cliente-servidor

A utilização do modelo cliente servidor sobre a qual a web atua traz consigo algumas considerações que impactam na arquitetura de uma aplicação.

Na maioria dos casos, uma arquitetura cliente servidor permite que os papéis e as responsabilidades de um sistema de computação possam ser distribuídos entre vários computadores independentes que são conhecidos por si só através de uma rede. Isso cria uma vantagem adicional para esta arquitetura: uma maior facilidade

de manutenção. Por exemplo, de acordo com o grau de independência de um cliente, é possível substituir, reparar, atualizar ou mesmo realocar um servidor enquanto seus clientes continuam tanto sem a consciência da manutenção e sem serem afetados por essa mudança. Para isto, porém, o cliente tem que ser mais do que um simples exibidor de conteúdo gerado pelo servidor.

O armazenamento de dados é realizado nos servidores, o que geralmente garante maiores níveis de segurança, dado que este cenário oferece controles de segurança ao acesso de recursos, garantindo que apenas os clientes, com as permissões adequadas possam acessar e alterar dados pertinentes ao seu contexto. A centralização do armazenamento de dados também facilita a administração e atualização de dados compartilhados comparativamente à utilização de outros paradigmas de comunicação de rede, como por exemplo o P2P, mas também restringe o grau de independência possível de um cliente, amarrando o acesso a estes dados à condição de acesso ao servidor.

2.2 Client-side scripting

Client-side scripting se refere à classe de programas de computador na web que são executados no lado do cliente, pelo navegador do usuário da web, em vez do lado do servidor (no servidor web). Este tipo de programação de computadores é uma parte importante do conceito de HTML dinâmico (DHTML), permitindo que as páginas da web possam executar *scripts*, ou seja, possam produzir e alterar o seu conteúdo dependendo da entrada do usuário, das respostas do servidor, e de outras variáveis, de acordo com seu estado interno.

A execução de *scripts* no cliente é fundamental para a criação de aplicações web que exerçam mais funções do que simplesmente a exibição de páginas geradas no servidor, pois permite uma maior interatividade com o usuário, o processamento e validação de dados, o estabelecimento de troca destes dados com o servidor e a criação de estruturas lógicas específicas ao domínio da aplicação.

Dentre as diferentes linguagens de script que podem ser executadas em um navegador web, a linguagem mais estabelecida - compatível entre diferentes navegadores - e utilizada é o JavaScript, um dialeto do padrão ECMAScript que é

caracterizada como uma linguagem dinâmica e funcional, imperativa e estruturada, fracamente tipada e orientada a objetos baseada em protótipos.

2.3 Comunicação cliente-servidor

O HTTP (*Hypertext Transfer Protocol*) é o protocolo de camada de aplicação que implementa a *World Wide Web*. Embora a web em si possua muitas facetas diferentes, o HTTP endereça apenas uma função básica: a transferência de documentos hipertexto e outros arquivos de servidores web para clientes web. Em termos reais de comunicação, os clientes estão principalmente interessados em fazer requisições para os servidores, que por sua vez respondem a esses pedidos.

2.3.1 Comunicação síncrona

Tradicionalmente, a comunicação entre cliente e servidor na web é implementada utilizando uma abordagem síncrona, ou seja, até que a resposta de uma requisição seja retornada, qualquer processamento no navegador web ou interação com o usuário do mesmo não podem ser realizados. No contexto de aplicações web, interfaces gráficas de usuário síncronas lançam uma requisição e congelam até que os dados de resposta sejam eventualmente recebidos.

Neste processo, quando os dados são recebidos, a página inteira é recarregada exibindo uma nova tela cheia. A utilização deste processo em todas as diferentes comunicações com o servidor gera diversas limitações, pois diminui a interatividade do usuário, não permitindo um fluxo de execução da interface gráfica contínuo e gradual.

2.3.2 Comunicação assíncrona

Comunicação assíncrona é uma forma de comunicação mediada em que ambos o emissor e o receptor não estão simultaneamente envolvidos na comunicação. Utilizando tecnologias assíncronas de comunicação, uma aplicação cliente pode realizar uma requisição e continuar seu fluxo de execução normalmente, sem interromper a responsividade de sua interface gráfica, através do estabelecimento de um método para o tratamento da resposta.

2.3.3 AJAX

Muito da possibilitação do desenvolvimento de aplicações web interativas, que realizam mais que somente a exibição de um conteúdo gerado no servidor, é devido à ampla adaptação de tecnologias assíncronas. Nesta abordagem, clientes requisitam dados do servidor de forma assíncrona. Quando a resposta chega, o cliente pode atualizar apenas as partes da página que realmente apresentam os dados.

O primeiro protótipo de tecnologia para comunicação assíncrona para a web foi criado em 1998, pela empresa Microsoft, porém só em meados de 2004 a utilização deste conceito tecnológico foi popularizado em aplicações web, e atualmente o AJAX é a técnica dominante para a implementação de serviços assíncronos na web.

O AJAX, no entanto, não é uma tecnologia única, mas sim um termo genérico que cobre várias tecnologias existentes e referencia a maneira como eles são usados em conjunto. O termo AJAX significa JavaScript e XML assíncrono, e como o nome sugere, permite a troca de dados de código e de domínio com o servidor, assim possibilitando – em conjunto com a execução de scripts no cliente – a atualização dinâmica de páginas e seu conteúdo em aplicações Web.

2.3.4 Considerações da comunicação entre cliente e servidor

A fim de manter a capacidade de resposta de um aplicativo, no contexto de desenvolvimento para a web, a comunicação com o servidor deve ser minimizada o tanto quanto possível, e quando necessária, deve ser realizada de forma assíncrona. Isto significa que para que a latência resultante da troca de dados seja minimamente perceptível, a troca deve ser realizada de forma a não travar o funcionamento do cliente até a chegada de uma resposta, e que mesmo assim - de acordo com o contexto da aplicação - a quantidade de vezes em que se é necessário estabelecer comunicação deve ser mínima.

2.4 Particionamento de aplicações entre cliente e servidor

O desenvolvimento de aplicações web está no meio de uma mudança de paradigma. Usuários estão se acostumando com aplicações web com conteúdo dinâmico e experiência mais interativa do usuário. Interfaces gráficas não mais estão

presas à necessidade de atualizar telas inteiras de uma vez, e servidores são capazes de alimentar os dados a elas espontaneamente.

Do ponto de vista dos utilizadores, as aplicações web estão, portanto, se tornando mais e mais como aplicações desktop tradicionais, mas enquanto as interfaces gráficas de aplicações web estão se tornando mais usáveis, os padrões e protocolos subjacentes não estão evoluindo no mesmo ritmo. As tecnologias existentes assim estão sendo levadas ao seus limites, e as aplicações web se tornam cada vez mais complexas de serem construídas.

Existem centenas de ferramentas e frameworks que aliviar a carga de complexidade para os desenvolvedores, porém todas elas trabalham sobre um nível de abstração cujos objetivos originais de concepção não vislumbravam mais que a exibição de documentos de hipertexto. Qualquer uma destas abordagens, portanto, depende de adaptações para o seu correto funcionamento.

A maioria dos atuais frameworks web focam na simplificação da implementação e manutenção de aplicações web através do fornecimento de um conjunto de ferramentas e convenções para os desenvolvedores. Isso ajuda os desenvolvedores a organizar e implementar o seu código fonte, arquivos binários e arquivos de recursos, mas eles realmente não melhoram a tecnologia por trás deles. O layout produzido da aplicação ainda é praticamente o mesmo se tivesse sido feito manualmente. Até então, as interfaces gráficas ainda são rederizadas na tela usando HTML4, CSS e JavaScript, enquanto o servidor fornece as fundações para os processamentos principais, lidando com todos os pedidos e gerando respostas em HTML, JSON, ou XML. No entanto, convenções bem estabelecidas para o particionamento da lógica da aplicação entre o cliente e o servidor ainda não existem, além da abordagem em que tudo é executado no servidor e em que o cliente simplesmente exibe os dados.

O particionamento da lógica de negócio entre cliente e servidor, ou seja, a definição da porção de código que ficará em cada uma das distintas camadas de uma aplicação, influencia diretamente a dependência entre o grau de interatividade de uma interface gráfica web e o modo em que a comunicação das duas partes deve ser feita. As decisões de particionamento de código, portanto, definem a dependência do cliente para com o servidor, o que significa que conseqüentemente

definem também as possibilidades de que uma aplicação possa ser executada em modo offline - sem conexão com o servidor.

No contexto do desenvolvimento de uma aplicação web complexa, na qual são desejados uma grande interatividade com o usuário e o mínimo de dependência com o servidor - até talvez com o intuito de disponibilização do funcionamento parcial da aplicação em situações em que o usuário não possui conexão com a internet - temos como requisito indireto que a criação da camada de implementação do cliente deve poder lidar com eventos do usuário o tanto quanto possível no cliente.

Neste cenário estabelecido, isto significa que seria ideal que o cliente possuísse o máximo controle possível sobre a execução da lógica de uma aplicação. Tal abordagem não só aumenta a capacidade de resposta e robustez da interface do usuário, mas também torna o desenvolvimento muito mais simples, pois resultaria em um número reduzido de requisições necessárias ao servidor - e o conseqüente tempo de latência relacionado a cada uma delas - garantindo assim maior independência e fluência da interface de usuário.

A utilização de uma única linguagem comum para toda a aplicação, com a aplicação transparente de um gateway para comunicação entre o servidor e o cliente, poderia facilitar o processo de desenvolvimento, aproximando-o ao desenvolvimento de aplicações tradicionais. Além disto, esta estratégia também permitiria fácil balanceamento de código entre o cliente e servidor - sem a necessidade de grande retrabalho - ampliando a flexibilidade da aplicação e a sua manutenibilidade.

2.4.1 Mandatário ao servidor

Existem determinadas operações que aplicações web não pode realizar sem o servidor, independentemente de como a aplicação foi particionada entre cliente e o servidor. Com o suporte da linguagem HTML5 e dos navegadores que a suportam, ou com a ajuda de ferramentas como o Google Gears, pode-se possibilitar a construção de aplicações web que possam ser executadas em modo offline. Mesmo assim, para muitas aplicações, não há possibilidade de eliminar todas as dependências com o servidor.

Seguem abaixo as mais comuns responsabilidades inerentes do servidor:

- Persistência - Refere-se aos dados que não são apagados no encerramento da execução do programa que o criou. Embora em navegadores modernos seguindo a especificação HTML5 seja possível o salvamento de dados localmente, muitas vezes existe também a necessidade de que estes dados estejam disponíveis também para outros computadores na rede.
- Comunicação cliente-cliente - Ocorre entre uma ou mais aplicações web rodado em diferentes clientes. Por questões de segurança - para prevenir ataques *Cross-site Scripting* - nos navegadores o JavaScript segue a política de mesma origem, o que impede clientes de evitar dados a destinos diferentes do servidor em que o cliente tenha sido carregado inicialmente.
- Dados compartilhados - Significa que dados necessitam estar acessíveis a mais de um cliente simultaneamente. O servidor é utilizado para prover uma base de dados centralizada e compartilhada entre diferentes clientes.
- Tarefas agendadas - Pode-se existir a necessidade da execução de tarefas e processamentos durante o período em que não exista garantia que o cliente esteja sendo executado.

2.4.2 Considerações para o servidor

De acordo com a natureza da aplicação, a execução de algumas tarefas e processamentos no servidor pode ser desejável, vantajosa ou até necessária. Enquanto as tendências da evolução da web em relação à execução de aplicativos interativos e dinâmicos dá preferência a (é coerente com) um modelo no qual o cliente possa lidar, na medida do possível, com os eventos gerados na interface gráfica, o contexto de regras de negócio da aplicação deve ser sempre levado em consideração.

- Navegação - o modo na qual a navegação de documentos na web evoluiu permite que o usuário utilize botões para avançar e retroceder entre o conjunto de páginas acessadas. Usuários estão acostumados a aplicações web simples nas quais se pode navegar com segurança

mesmo quando utilizadas estas funções. Esta característica é perdida quando uma aplicação inteira é implementada como uma única página dinâmica. Soluções para este problema envolvem uma abordagem híbrida em que diferentes URLs identifiquem diferentes partes de uma aplicação ao mesmo tempo em que estas transições de estado possam ser realizadas de forma dinâmica e parcial, ou seja, através da adaptação do conteúdo entre diferentes estados sem a necessidade de um recarregamento total da aplicação.

- Cálculos pesados - ou outras operações de uso intensivo da CPU podem ser, em alguns casos, melhor executados no servidor devido ao fato deste poder ser um ambiente um pouco mais controlado onde se possui conhecimento do hardware utilizado e sua capacidade. A execução de código no servidor também possibilita o uso de linguagens mais poderosas para certos domínios, como por exemplo o de cálculos lógicos ou matemáticos.
- Segurança - devido ao fato de que o código executado no cliente pode ser acessado e modificado, existem certas operações cuja a execução possa ser obrigatória no servidor. Um exemplo inclui a necessidade de validação de dados fornecidos pelo usuário, que pode ser facilmente burlada por um usuário mais experiente.
- Confiabilidade - pode ser aumentada através da execução de operações no servidor. Por exemplo, usuários podem fechar seus navegadores da web durante uma operação, ou no caso de operações dependentes de horário, comportamentos inválidos podem ocorrer devido à má configuração do relógio do cliente.

2.4.3 Considerações para o cliente

A natureza da plataforma web, de acordo com a sua evolução, se deu de tal forma na qual os clientes mais simples de uma aplicação são basicamente constituídos da exibição de uma interface gráfica simples e estática, praticamente sem a execução de nenhuma lógica. A criação de clientes mais complexos vem sendo um desafio que aos poucos é possibilitado pelas abstrações tecnológicas

fornecidas por frameworks e ferramentas de desenvolvimento, para facilitar a criação de interfaces dinâmicas, que proporcionem uma maior interatividade com o usuário e maior poder em termos de capacidades que uma aplicação web pode apresentar.

A disponibilização de dinamicidade e de funcionalidades no software cliente requer ou que o mesmo possua lógica em termos de ambas regras de negócio e integração entre os componentes da visualização de acordo com o domínio da aplicação ou que uma comunicação extremamente ativa com o servidor seja estabelecida para que o servidor possa determinar os detalhes de como o cliente deve responder a cada evento. Em outros termos, para a criação de aplicações web, é necessário – proporcionalmente a sua complexidade - ou trazer parte da sua lógica ao cliente ou aumentar a dependência da comunicação de um cliente simples e “burro” com o servidor. Esta dualidade estabelece claramente duas diferentes – e quase opostas – abordagens, uma propondo a utilização de um cliente fino e simples administrado pelo servidor e outra propondo a criação de um cliente mais especializado contendo parte da lógica da aplicação. A figura 3 expõe as principais características de cada abordagem.

	Cliente fino	Cliente grosso
Dependência do servidor pelo cliente	Grande	Baixa
Dependência da interatividade pela latência de rede	Grande	Baixa
Controle do servidor sobre o estado do cliente	Grande	Baixo
Carga e demanda sobre o servidor	Grande	Baixo
Trabalho no desenvolvimento do cliente	Pequeno – a UI pode ser gerada	Grande – a UI tem que ser criada especificamente de acordo com o cliente.

Figura 3: Particionamento entre cliente e servidor.

2.5 Aplicações web offline

O progresso do desenvolvimento de aplicações para a web vem permitindo que este tipo de aplicações tenha um crescente alcance em termos do que pode ser feito neste ambiente, trazendo para a plataforma web não só serviços e funcionalidades que originalmente eram disponíveis somente em aplicações desktop tradicionais,

mas também novas capacidades que são somente possíveis em um modelo que pode centralizar informações pertinentes referentes a uma massa crítica de usuários, ou seja, dados não disponíveis no contexto das aplicações desktop, no qual o uso é relativamente isolado entre diferentes instâncias de um software. Ao mesmo tempo, este modelo de computação em nuvem oferecido pela plataforma web também oferece limitações aos usuários, dentre as quais se pode ressaltar a necessidade quase incondicional de um meio de acesso à internet pelo usuário para que este possa ter acesso a estes serviços. A utilização da web tem como requisito fundamental a presença de uma conexão à internet.

A criação de mecanismos para o desenvolvimento de aplicativos web offline tem como intuito mitigar a limitação - a qual muitas aplicações desktop não possuem - imposta pela necessidade de uma conexão à internet. O esforço da criação de aplicações web offline tem como intento garantir que usuários tenham acesso básico a funcionalidades e dados, seja a aplicação, por exemplo, desde um cliente de e-mail até uma agenda de compromissos. O acesso offline cria uma camada a mais de confiabilidade, estende o alcance de aplicações web e pode melhorar o seu desempenho movendo parte dos dados para mais perto de seus usuários.

2.5.1 Requisitos para disponibilização offline de uma aplicação web

Os requisitos mínimos necessários para que uma aplicação web possa ser disponibilizada offline podem ser sintetizados e classificados em duas categorias básicas:

- Armazenagem de recursos da aplicação - que implica em um mecanismo de armazenamento local de código da aplicação, junto a qualquer lógica e a outros recursos que esta venha a necessitar para o seu funcionamento. Este mecanismo deve também deslumbrar uma maneira de direcionar qualquer requisição da aplicação para os respectivos recursos armazenados localmente caso não exista acesso aos recursos reais na internet. Além disso, é necessário o estabelecimento de um meio para a determinação de todos os recursos que devem ser salvos localmente.

- Armazenagem de dados do usuário, relativos ao uso da aplicação – indica mecanismos de armazenamento local de quaisquer tipos de dados possíveis gerados pela aplicação que necessitam ser salvos para posterior utilização, seja para sincronização posterior com o servidor, para o uso local do usuário ou para ambos.

3 – HTML5

O HTML foi concebido essencialmente como uma linguagem para descrever semanticamente documentos científicos, embora a sua concepção geral e adaptações ao longo dos anos têm-lhe permitido ser usado para descrever uma série de outros tipos de documentos. A principal área que não foi devidamente tratada por HTML é um vago assunto denominado “Web Applications” (aplicações web).

3.1 História do HTML

Por seus cinco primeiros anos (1990-1995), o HTML passou por uma série de revisões e experimentou uma série de extensões, principalmente hospedado primeiro no CERN, e depois no IETF. Com a criação do W3C, o desenvolvimento do HTML foi alavancado por uma série de ampliações de sua especificação até a versão 4, concluída em 1998.

Até então estas especificações eram principalmente voltadas à definição de documentos de hipermídia e o avanço de outras tecnologias parecia muito mais promissora quanto à facilitação do desenvolvimento de recursos mais avançados para a criação de interfaces de aplicações na web. O interesse na evolução da especificação do HTML perdeu força com a descrença de seu potencial em soluções profissionais, mas devido a sua grande difusão, seu uso continuou a ser explorado e expandido nestas práticas.

Em torno do tempo em que a evolução do HTML foi interrompida, em 1998, partes da API para HTML desenvolvidas por fabricantes de navegadores foram especificadas e publicadas sob o nome de DOM1 (em 1998) e DOM2 Core e DOM2 HTML (a partir de 2000 e culminando em 2003). Esses esforços foram em seguida depreciados com a publicação de algumas especificações do DOM3 em 2004 que, porém, nunca foram concluídas.

Provas de conceito demonstrando que era possível estender HTML4 de forma a fornecer muitas das características de outras tecnologias propostas, como o XForms, foram com o tempo introduzidas. Tais provas de conceito demonstraram também que estas ampliações eram possíveis sem a necessidade da implementação de "motores" de renderização incompatíveis com as atuais páginas da Web em HTML.

A idéia de que a evolução do HTML deveria ser reaberta foi testada em um workshop do W3C, em 2004, onde alguns dos princípios que fundamentam o HTML5 foram apresentados, conjuntamente pela Mozilla e Opera. Porém, com o fundamento de que a proposta entrava em conflito com a direção previamente escolhida para a evolução da Web, a proposta foi rejeitada pela equipe do W3C.

Pouco tempo depois, a Apple, Mozilla e Opera em conjunto anunciaram sua intenção de continuar trabalhando no esforço sob a égide de uma nova instituição chamada WHATWG. A lista de discussão pública foi criada, bem como o projeto foi transferido para o site WHATWG.

O WHATWG foi baseado em vários princípios fundamentais, em sumo determinando que é necessário que novas tecnologias sejam compatíveis com versões anteriores, que as especificações e implementações necessitar estar em sincronia mesmo que isso signifique mudar a especificação em vez das implementações, e que as especificações devem ser detalhadas o suficiente para que as implementações possam atingir interoperabilidade completa sem a engenharia reversa de outra.

O último requisito exigido em particular requeria que o escopo da especificação HTML5 incluísse o que tinha sido previamente especificado em três documentos distintos: HTML4, XHTML1 e DOM2 HTML. Isto também significou a inclusão de um detalhamento significativamente mais completo do que anteriormente tinha sido considerado a norma.

Em 2006, o W3C manifestou interesse em participar no desenvolvimento do HTML5, e em 2007 se juntou definitivamente ao grupo de trabalho da especificação do HTML5.

3.2 O HTML5 e o Desenvolvimento de Aplicações Web

A popularidade da web quanto ao desenvolvimento de aplicações cresce continuamente e, para que este progresso seja possível, diferentes sistemas e frameworks vêm ganhando grande complexidade na tentativa de simplificação do desenvolvimento e na mitigação dos desafios envolvidos.

Até a versão 4 do HTML, sua especificação era voltada essencialmente na criação e exibição de documentos diversos, e não visava praticamente nenhum

suporte a aplicações para a web. Todas as soluções criadas remediam esta situação criando seus próprios "blocos básicos" necessários ao desenvolvimento de aplicações, lidando com uma grande falta de congruência entre especificações e suas respectivas implementações, e também entre diferentes implementações.

A falta de padronização e de garantia de que um mesmo sistema se comportaria da mesma maneira em diferentes navegadores web é um dos maiores desafios encontrados no desenvolvimento de software da plataforma web. Existe uma crescente preocupação com a maneira pela qual os sistemas baseados na Web são concebidos, principalmente quanto à sua qualidade, integridade e manutenção.

A criação do HTML 5 tem sua motivação na falta de uma base mais sólida na web no qual softwares possam se estabelecer. Sua especificação visa mitigar a falta de padronização entre diferentes implementações, aumentar o grau de fidelidade entre os resultados desejados e obtidos na exibição e comportamento de componentes de software, e por fim, visa também fornecer os recursos e funcionalidades básicas necessárias para que aplicativos web possam cada vez mais ter as mesmas capacidades de aplicações nativas.

3.2.1 Funcionalidades básicas abordadas pelo HTML5

Com o início da disponibilização e utilização prática do HTML 5 nos navegadores web, as diferentes abordagens para criação de software podem potencialmente ser simplificadas com a utilização de recursos mais estáveis e bem definidos. A disponibilização de recursos e capacidades oferecidos "nativamente" nos navegadores, ou seja, em um nível mais baixo de abstração, elimina também a necessidade de soluções secundárias e artimanhas.

Dentre as soluções propostas pelo HTML5 podemos destacar algumas adições de grande importância e impacto sobre o desenvolvimento de aplicações web.

- Capacidades gráficas ampliadas - com suporte ao canvas, um elemento que representa uma área baseada em pixels, e a gráficos vetoriais escaláveis;
- Exibição de vídeos e animações - sem a necessidade de plug-ins externos;

- Localização - a capacidade de relacionar informações de localização geográfica a aplicações;
- Workers - possibilidade de execução de scripts em plano de fundo;
- Web sockets - possibilita essencialmente a criação de canais bidirecionais de comunicação entre o cliente e o servidor. Estabelece e garante uma maneira padronizada para o que o servidor envie dados ao cliente sem a necessidade de técnicas de polling.
- Cacheamento de arquivos - permitindo o cacheamento local controlado de recursos e arquivos de aplicação;
- Base de dados relacional local - possibilidade de armazenamento local de dados para seu uso em aplicações, tanto online quanto offline, oferecido na forma de uma base de dados relacional.
- Armazenamento de dados local – oferecendo o armazenamento local de dados para seu uso em aplicações, tanto online quanto offline;

3.3 Desenvolvimento de aplicações web offline com o HTML5

O HTML5 contempla todas as funcionalidades e APIs necessárias para comportar os requisitos mínimos para a disponibilização de aplicações web em um contexto offline. Sua especificação estabelece dois mecanismos de cacheamento de recursos que podem ser programaticamente configurados - de maneira semelhante aos cookies de sessão HTTP - para o armazenamento de dados estruturados no cliente, e um conjunto de APIs para a manipulação e armazenamento no cliente de bases de dados relacionais através do uso de SQL.

3.4 Cacheamento de recursos

A fim de permitir que documentos, aplicativos da web e seus respectivos recursos, como imagens, folhas de estilo CSS e scripts, possam ser disponibilizados, mesmo quando uma conexão de rede não está disponível, o HTML5 especifica um manifesto – cuja responsabilidade de interpretação e execução é do navegador - no qual é possível declarar uma lista de arquivos dos quais o navegador do usuário deve manter uma cópia para uso offline.

3.4.1 Manifesto

Um manifesto é constituído de um arquivo simples, que deve ser servido pelo servidor como “text/cache-manifest”, listando todas as informações sobre os outros recursos necessários. Sua declaração é ligada a documentos HTML através de um atributo “manifest” no elemento “html”, e a partir do primeiro acesso de um usuário a esta página, o seu navegador realizará automaticamente o cacheamento dos arquivos e os tornará disponíveis para quando o usuário estiver offline, garantindo também que nenhum dos recursos expirará até que o manifesto seja atualizado, ou seja, exceto pela requisição do usuário ou da aplicação.

3.4.2 Eventos do cacheamento

No momento em que uma página - que declara um manifesto de cacheamento - é visitada por um usuário, o navegador inicia um processo com objetivo de atualização do cache. Isto é realizado através da obtenção de uma cópia do manifesto, seguida de uma comparação entre o estado proposto e o estado atual do cache. Caso alguma mudança seja detectada, o navegador requisita e armazena uma nova cópia de todos os arquivos declarados. Durante todo este processo, existe uma seqüência de eventos que podem ser disparados, cuja finalidade é prover à aplicação uma sinalização de todo o percurso deste processo e seus resultados.

Com o lançamento destes eventos, a aplicação pode não somente indicar o estado da sincronização e realização do cacheamento, mas também pode tomar atitudes e realizar ações de acordo. A API especificada também permite a checagem do estado atual do cache da aplicação e a chamada de funções relacionadas, como por exemplo, a re-checagem da validade do cache atual. A existência destes métodos e eventos provê todo o controle sobre o processo realizado pelo navegador necessário para que uma aplicação possa programaticamente estabelecer sua lógica.

3.4.3 Caches de aplicação

Um cache de aplicação é um conjunto de recursos armazenados que é constituído de:

- Um ou mais recursos, identificados por URLs, nos quais cada recurso pode ser enquadrado em uma das seguintes categorias:
 - Entradas principais - documentos que foram adicionados ao cache porque um contexto de navegação foi "navegado" em algum documento que declarava um manifesto. O documento que declara um manifesto, mesmo que não seja declarado neste manifesto, é sempre adicionado ao cache.
 - O manifesto - o recurso referido por um atributo "manifest" em uma entrada principal correspondente à URL de origem. Todas as entradas principais têm a mesma origem que o seu manifesto.
 - Entradas explícitas - são os recursos que tiveram seu cacheamento explicitamente declarados em um manifesto. Entradas explícitas também podem ser marcadas como *foreign*, palavra-chave que estabelece que no conteúdo do manifesto existe a declaração de outros manifesto, fazendo a declaração de mais recursos.
 - Entradas indiretas - são recursos que foram listados no manifesto do cache em uma sessão de *fallback*.
- Zero ou mais namespaces de *fallback* (entradas indiretas) - URLs, usadas como padrões de correspondência de prefixos, cada uma mapeada a uma entrada indireta.
- Zero ou mais URLs que formam uma lista de *namespaces* permitidos quando online.

3.4.4 Grupos de caches de aplicação

Um grupo de caches da aplicação é o conjunto de caches de aplicação identificados pela URL absoluta de um manifesto de recursos. Cada grupo tem um próprio indicador de estado do processo de atualização, que pode ser *idle*, *checking* ou *downloading*. Múltiplos caches de aplicação em diferentes grupos podem conter os mesmos recursos, por exemplo quando mais de um manifesto referencia o mesmo recurso.

3.5 Web Storage

A especificação do Web Storage define uma API no cliente para o armazenamento persistente de dados baseados em pares de chave-valor. Dois mecanismos relacionados são introduzidos, ambos voltados ao armazenamento de dados estruturados.

3.5.1 O atributo *sessionStorage*

O atributo *sessionStorage* foi planejado para quando o usuário está conduzindo uma transação única, mas poderia estar conduzindo - ao mesmo tempo - múltiplas transações em diferentes janelas. Neste cenário, a utilização de cookies não difere os diferentes contextos de sessão, e como o processamento dos dados pode ocorrer diretamente no cliente – sem a interferência do servidor como meio centralizador – os dados provenientes das múltiplas instâncias não se distinguiriam. Endereçando este problema, o atributo *sessionStorage* permite que dados sejam salvos e carregados separadamente de acordo com as diferentes sessões no cliente.

3.5.2 O atributo *localStorage*

O atributo *localStorage* foi criado para guardar dados globais de uma aplicação, disponibilizando-os entre todas as diferentes sessões, mesmo após o encerramento das mesmas. Este mecanismo tem como principal finalidade o armazenamento de dados não limitados em tamanho, ou seja, dados que podem alcançar a magnitude de dezenas de Megabytes, específicos de um domínio. Mais uma vez, a utilização de cookies neste cenário também não é recomendada, pois cookies necessitam ser enviados ao servidor a cada requisição de recurso.

3.5.3 A interface *Storage*

Ambos os mecanismos, o armazenamento de sessão e o armazenamento local compartilhado, implementam a mesma interface básica *Storage*. Cada objeto *Storage* provê acesso a uma lista de pares chave-valor, no qual qualquer string é uma chave válida (incluindo a string vazia) utilizada para referenciar e mapear qualquer estrutura de dados clonável.

3.6 Web SQL Database

O HTML5 oferece também um conjunto de APIs para a manipulação e armazenamento no cliente de bases de dados através do uso de um dialeto próprio de SQL. A especificação destas APIs tem como objetivo facilitar a gerência de dados relacionais e possibilitar o uso de funções de busca e filtragem dos mesmos no lado do cliente.

3.6.1 Bases de dados

Cada origem, ou seja, domínio de aplicação, pode possuir um conjunto de bases de dados isoladas. Aplicações de diferentes domínios não possuem acesso algum às bases de dados de outras aplicações. Cada base de dados possui um valor de versão, para que a aplicação possa ter controle sobre as possíveis atualizações na estrutura de tabelas de diferentes clientes, eliminando erros devido à eventual existência de bases desatualizadas.

3.6.2 Métodos de acesso

As bases de dados podem ser acessadas de modo síncrono ou assíncrono. Devido à maneira na qual os *scripts* são executados no navegador, e às características de tais linguagens de *scripting*, o acesso assíncrono é o mais recomendado, mantendo a interatividade da aplicação e evitando o congelamento da responsividade da sua interface gráfica durante o processamento requisitado.

3.7 Compatibilidade do HTML5

A especificação do HTML5 ainda está em desenvolvimento, e nenhum navegador atualmente implementa totalmente a sua versão corrente. As funcionalidades propostas pelo HTML5 estão sendo gradualmente implementadas em diversos navegadores diferentes, segundo a prioridade dos responsáveis por cada um deles, porém, as funções necessárias para a disponibilização de um documento ou aplicação em modo offline já está completa na maior parte dos navegadores mais utilizados, como o *Internet Explorer 8*, e as mais recentes versões do *Mozilla Firefox* e do *Google Chrome*.

4 – JSF 2.0

O JavaServer Faces é o principal framework de interface gráfica ao usuário para aplicações Java web na especificação do JavaEE 6 . O JSF é uma framework MVC orientada a requisições, fundamentado no modelo de design de interfaces gráficas baseadas em componentes, através da utilização de arquivos XML chamados de view templates (templates de visualização) ou Facelet views. O framework gera o conteúdo para o cliente (geralmente em HTML) e realiza o processamento das subseqüentes requisições.

Requisições são processadas pelo FacesServlet que, em resumo, carrega o template de visualização apropriado, monta a sua árvore de componentes, processa eventos e renderiza a resposta para o cliente. A arquitetura do JSF estabelece um modelo em que quase a totalidade dos processamentos da aplicação são executados no servidor, não somente incluindo as funções de gerenciamento necessárias, mas incentivando que a lógica de negócio da aplicação não seja levados ao cliente. Esta abordagem promove o cliente como uma camada fina e delega ao servidor toda a gerência das responsabilidades e funcionalidades do framework. Algumas das responsabilidades e funcionalidades assumidas pelo JSF incluem:

- Gerenciamento do estado dos componentes da UI entre requisições;
- Suporte do processamento de formulários, incluindo de múltiplas páginas e também de mais de um por página;
- Um modelo de eventos fortemente tipado que permite a criação de tratadores personalizados no lado do servidor para lidar com eventos gerados no cliente;
- Validação de dados de requisições provendo um mecanismo para relação de erros;
- Permite a conversão de tipos entre os valores definidos em markup e os objetos de dados da aplicação;
- Gerenciamento de erros e exceções, permitindo que os mesmos sejam reportados de forma inteligível ao usuário da aplicação na UI;
- Manipulação de navegação página-a-página em resposta a eventos da UI e das interações do modelo;

4.1 Ciclo de vida de processamento de requisições

Interfaces gráficas web de usuário geralmente seguem um padrão no qual o agente do usuário envia um ou mais requisições ao servidor, com o intuito de exibição da UI. No caso dos navegadores web, uma requisição inicial HTTP GET ou POST é realizado ao servidor, que responde com um documento que é então interpretado pelo navegador automaticamente gerando subseqüentes requisições. Cada uma das respectivas respostas representa recursos, como imagens, arquivos de script, folhas de estilo e outros artefatos que fazem parte do documento original. O JSF trata cada requisição com um ciclo de vida de processamento bem definido, constituído por diferentes fases, para a geração da resposta equivalente à interação dos valores passados com o modelo e lógica da aplicação.

4.2 Cenários de processamento de requisições

No contexto de aplicações desenvolvidas utilizando o JavaServer Faces, existem quatro diferentes cenários possíveis quanto ao tratamento de uma requisição em uma resposta. Considerando que uma Resposta Faces seja uma resposta criada pela execução da fase de renderização de resposta do ciclo de vida do JSF, e que uma Requisição Faces seja uma requisição enviada a partir do conteúdo de uma Resposta Faces previamente gerada, existem três cenários que necessitam ser considerados, e um que pode ser desconsiderado. O processamento de uma Requisição Não-Faces para a geração de uma Resposta Não-Faces não representa nenhum processamento no escopo do JSF.

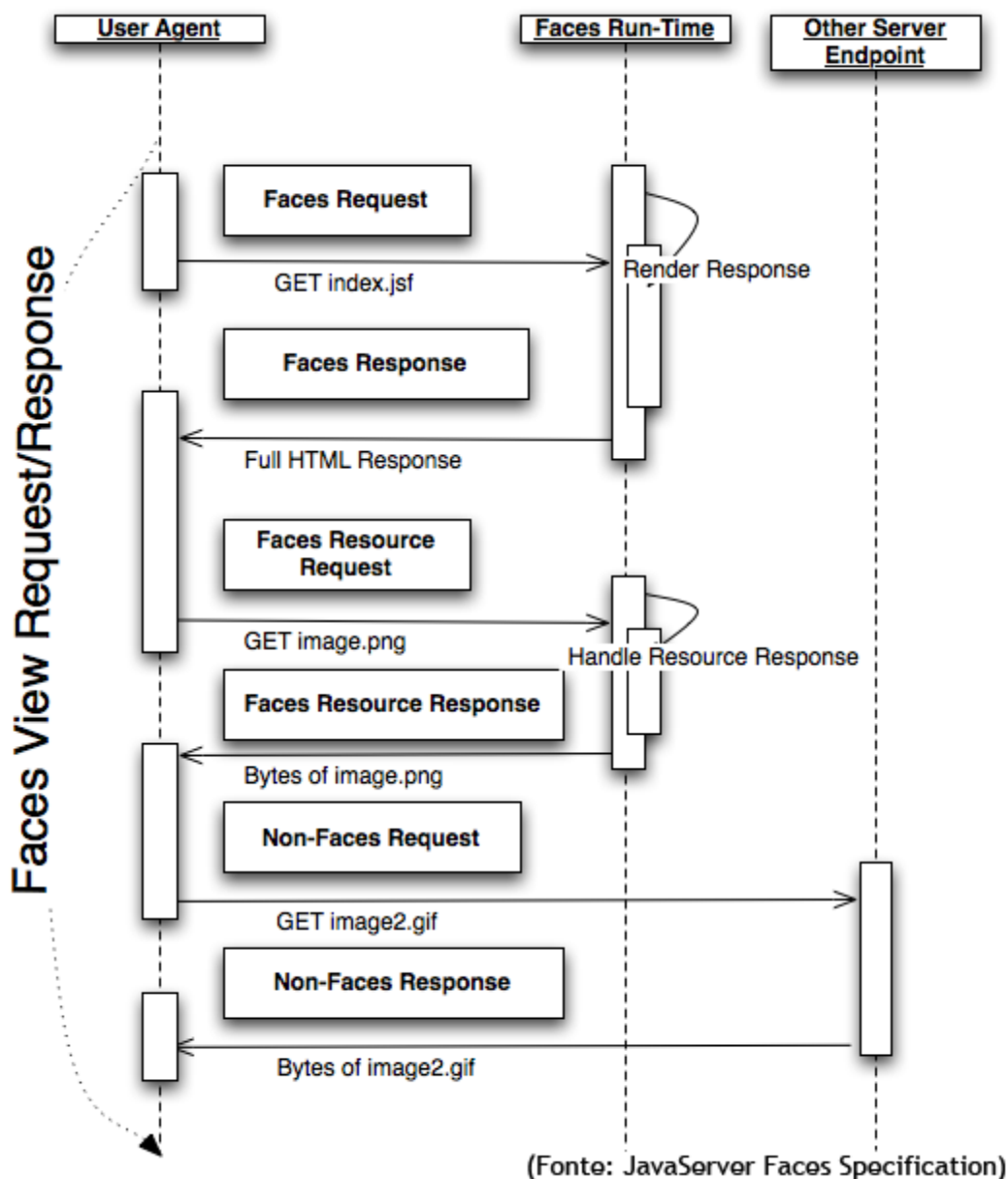


Figura 4: Diagrama de sequência ilustrando os diferentes possíveis comportamentos para diferentes requisições.

4.2.1 Requisição Não-Faces gera Resposta Faces

No caso em que uma requisição é enviada a um componente da aplicação – como, por exemplo, um servlet ou uma página JSP - ao invés de diretamente à view do Faces, a aplicação tem como responsabilidade criar uma nova view e a

configurá-la no contexto correto, permitindo a execução do fluxo normal do ciclo de vida de processamento de requisições do JSF para a geração da resposta apropriada.

4.2.2 Requisição Faces gera Resposta Faces

O mais comum ciclo de vida é o caso no qual uma Resposta Faces inclui controles de UI que podem submeter requisições subseqüentes de volta à aplicação, utilizando uma URI que está mapeada para o controlador da implementação do JSF em uso. O framework então gerencia todo o processamento da requisição em resposta, invocando quando necessário os mecanismos de tratamento de eventos, de validação e a própria lógica da aplicação, quando necessário.

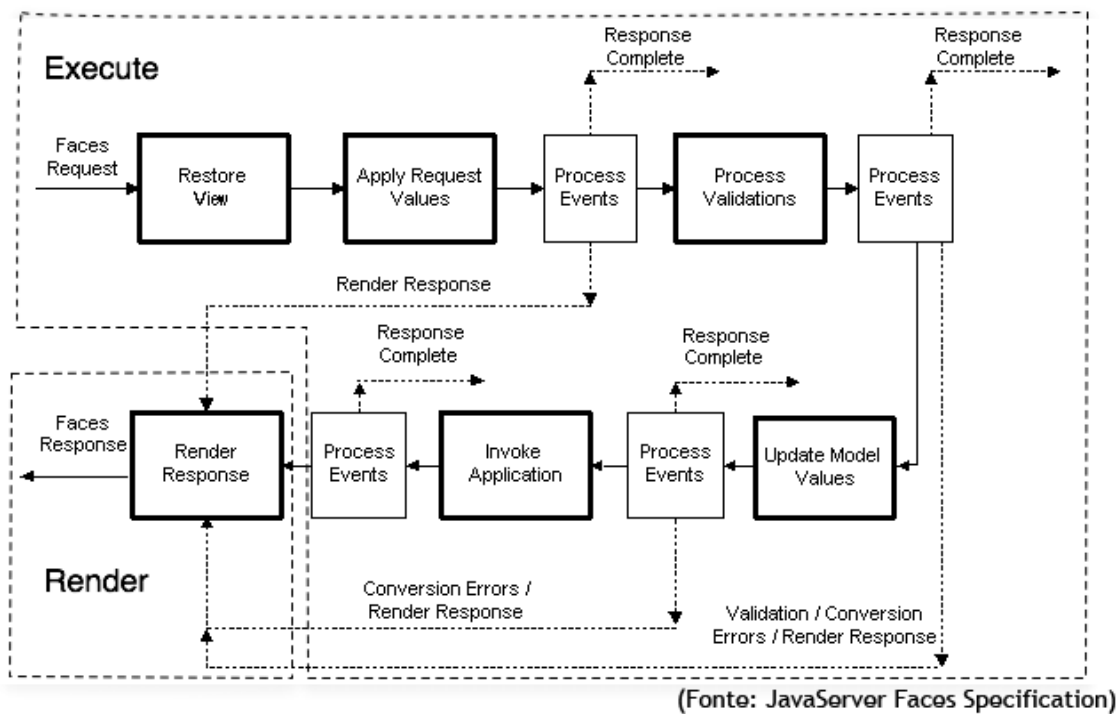


Figura 5: Fases do tratamento padrão de requisições pelo JSF 2.0.

4.2.3 Requisição Faces gera Resposta Não-Faces

Requisições de recursos são tratadas de forma transparente pelo JSF. Causa a geração de uma resposta servindo os bytes de um recurso, como uma imagem, um arquivo ou até recursos produzidos com outra tecnologia (como um servlet).

4.3 Fases do ciclo de vida do processamento de requisição

4.3.1 Restauração da visualização

O JavaServer Faces armazena todas as informações relativas ao estado de uma interface gráfica gerada por seus mecanismos, chamada de visualização, para possibilitar o futuro processamento de requisições subseqüentes. A primeira fase do ciclo de vida é responsável pela recuperação das informações que refletem as configurações salvas pela visualização gerada pela última resposta Faces, ou pela criação de uma nova visualização criada para esta sessão, no caso da não existência de uma resposta precedente.

Dentre as informações recuperadas, se pode destacar a estrutura de árvore de componentes que originou e reflete o estado atual da interface gráfica, utilizada para processamento das requisições geradas a partir da mesma.

4.3.2 Aplicação dos valores da requisição

O propósito da fase de aplicação dos valores da requisição é fornecer uma oportunidade de atualização do estado atual a cada componente da visualização, de acordo com as informações disponibilizadas pela requisição corrente. Nesta ocasião, assim que novas informações são examinadas por um componente para a atualização de seu estado, o componente as armazena como “valores locais”.

Neste momento do processamento, componentes que implementem *ActionSource*, ou seja, que possam gerar ações no sistema, que reconheçam sua ativação, registram a execução de uma ação. O registro da execução de ações é feita através de eventos que, de acordo com suas propriedades, podem ser efetuadas em diferentes pontos do ciclo de vida.

4.3.3 Validação do processamento

A fase de validação do processamento realiza a validação do novo estado proposto pela requisição. Como parte da criação ou recuperação da visualização relativa à atual requisição, zero ou mais instâncias de validadores podem ter sido registradas para cada componente - de acordo com a definição da interface gráfica pelos desenvolvedores da mesma. Além disto, cada componente pode também implementar suas próprias lógicas de validação.

De forma recursiva, a validação de cada componente da visualização é realizada, utilizando seus respectivos “valores locais”. Erros de validação podem tanto gerar mensagens para sinalização de um problema sem interromper o fluxo normal de processamento, quanto podem indicar a necessidade da geração de uma resposta sem a continuação do ciclo de vida.

4.3.4 Atualização dos valores do modelo

Caso esta fase do ciclo de vida do processamento da requisição seja atingida, é assumido que o valor local de cada componente foi atualizado, que a requisição seja válida (de acordo com as validações feitas) sintaticamente e semanticamente e que é apropriado atualizar o modelo de dados da aplicação, preparando-a para a execução dos eventos da aplicação registrados na fase de aplicação dos valores da requisição. A ocorrência de algum erro durante a atualização dos valores do modelo ou durante a execução dos eventos da aplicação pode também gerar a interrupção do fluxo normal de processamento.

4.3.5 Invocação da aplicação

A invocação da aplicação é realizada através da distribuição e processamento no sistema de todo e qualquer evento criado que ainda não tenha sido tratado. Assim como nas fases do ciclo de vida precedentes, erros durante esta fase pode interromper o fluxo normal de processamento.

4.3.6 Renderização da resposta

A geração da resposta é a última fase do ciclo de vida de processamento de uma requisição Faces, e possui como responsabilidades, respectivamente, a renderização de uma resposta ao cliente e o salvamento do estado da resposta, possibilitando o processamento de requisições subseqüentes.

A implementação de diferentes abordagens para a criação de uma resposta é suportada pelo JSF, possibilitando a geração de respostas customizadas. É possível criar *RenderKits* especializados que intercalem os resultados da codificação dos componentes com conteúdo dinamicamente gerado ou pela lógica da aplicação ou por um template estático.

4.4 Processamento de eventos

O JSF implementa um modelo para notificação de eventos e de registro de tratadores dos mesmos baseado nos padrões de design da especificação JavaBeans. Eventos podem ser emitidos denotando mudanças de estado significantes. Cada um destes eventos são então difundidos entre os tratadores que registraram interesse no recebimento de um tipo indicado pela classe que implementa cada evento. Existe também a possibilidade de que os próprios tratadores de eventos lancem outros eventos ao sistema.

Durante a execução das diversas fases do ciclo de vida de processamento de requisições existe a possibilidade de que eventos sejam lançados influenciando diretamente o processamento e a resposta gerada. Neste contexto, eventos podem, por exemplo, indicar erros diversos - como de validação ou de conversão de valores - resultando em até a própria finalização do processamento da requisição - caso esta se faça necessária - indicando a geração de uma resposta conforme a situação.

4.5 Conceitos que impactam nas fases do ciclo de vida

Uma série de diferentes conceitos internos do modelo definido pelo JSF podem exercer influência sobre o processamento de uma requisição Faces,

podendo inclusive modificar a ordem padrão do fluxo de processamento. Todas as fases relacionadas à manipulação de valores de uma requisição podem, por exemplo, ser antecipadas, de acordo com propriedades configuradas nos componentes utilizados para a geração da interface gráfica de uma aplicação. Do mesmo modo, a gerência de estado, o tratamento de recursos e os comportamentos de componentes podem todos sofrer customizações. Questões relacionadas à internacionalização e localização de software também influem no conteúdo utilizado para geração de uma resposta.

4.5.1 AJAX

O JSF 2.0 introduz a utilização de requisições AJAX à especificação. Por alguns anos o JSF e técnicas de AJAX têm sido aplicados juntos através de frameworks complementares, cada uma destas contribuindo para uma experiência mais dinâmica a interação do usuário através de diferentes abordagens.

As variações entre as diferentes maneiras em que estes frameworks operam podem ocasionalmente causar problemas de compatibilidade de componentes em uma aplicação. A especificação 2 do JSF então, a partir de conceitos provindos de uma variedade de frameworks AJAX para JSF, padroniza a utilização destas técnicas, definindo uma biblioteca Javascript responsável pelas operações básicas AJAX, como o envio de requisições e o processamento das respostas conseqüentes.

Além de, neste contexto, padronizar os aspectos relacionados ao cliente, o JSF 2 também padroniza os aspectos relacionados ao servidor, tratando requisições utilizando o ciclo de vida padrão de processamento de requisições JSF.

4.5.2 Comportamento de componentes

Os comportamentos de componentes desempenham um papel similar aos validadores e conversores do JSF, porém também podem ter impacto no cliente, na forma de scripts que podem ser anexados à árvore de tratamento de eventos DOM.

5 – INTEGRAÇÃO DO JSF 2 E HTML5 OFFLINE

Este capítulo trata da compatibilização do JavaServer Faces 2.0 com as capacidades de criação de aplicações offline oferecidas pelo HTML5, segundo as características definidas nos capítulos 3 e 4, analisando em termos da viabilidade a problemática envolvida, as possíveis soluções e seus requisitos e impactos ao usuário.

5.1 Desafios inerentes às tecnologias em questão

5.1.1 Relativos ao HTML5 e aos navegadores web

O HTML5 é a primeira especificação desta tecnologia que padroniza e define mecanismos e recursos básicos diretamente visando à criação de aplicativos sobre a plataforma web. Dentre outros mecanismos, esta especificação determina meios para solucionar todas as necessidades para disponibilização de aplicações sem a necessidade de comunicação com o servidor, ou seja, sem acesso a internet. Todas estas soluções, porém, não endereçam questões relacionadas à execução de código em linguagens de programação que os navegadores atuais não possuem capacidade de processar, como o Java, linguagem na qual o JavaServer Faces é implementado e executado. Isto significa que não existe nenhuma maneira padronizada entre os diferentes navegadores web que permita – ou mesmo vislumbre - a execução de código Java relacionado às especificações existentes do JSF para a supervisão da aplicação quando offline.

O JavaScript é – devido à sua popularidade e difusão entre diferentes implementações dos navegadores atuais - a principal linguagem existente para execução de processamentos no lado do cliente de uma aplicação (client side scripting). Isto implica que, para que uma aplicação web desenvolvida com o JSF possa ser executada em modo offline de modo padrão entre navegadores modernos que suportam o HTML5, seus processamentos e sua lógica de negócio devam ser executados em JavaScript. Apesar disto, também existem alternativas – não padronizadas na plataforma web – em que navegadores podem executar códigos em diferentes linguagens.

5.1.2 Relativos ao JSF 2.0 e ao modelo cliente-servidor da web

A arquitetura determinada pela especificação do JSF 2.0, por sua vez, estabelece um modelo em que o servidor assume toda a responsabilidade sobre o controle e sobre a lógica de negócio da aplicação. É estabelecida, assim, uma estrutura rígida que força que todas as funções de gerenciamento de uma aplicação, desde a gerência do estado da interface gráfica e do modelo até o processamento e manipulação de valores, sua validação, e a geração de uma resposta, sejam executadas no lado do servidor.

Como conseqüência, o cliente da aplicação age como uma camada fina representando somente uma interface gráfica, sem inteligência e auto-suficiência, com alta dependência sobre a rede para acesso ao servidor.

5.1.3 Problemática da compatibilização

De acordo com os requisitos propostos por este trabalho - relativos aos desafios pertinentes à compatibilização do HTML5 e do JSF 2.0 – algumas importantes questões devem ser respondidas para a determinação das possíveis soluções e suas respectivas viabilidades: Como trazer código do servidor para execução no cliente? Qual a porção de código que deve ser trazida ao cliente? Como sincronizar dados e processamentos entre o cliente e o servidor, garantindo a integridade dos dados e operações?

5.2 Código necessário para execução offline no cliente

A primeira problemática a ser trabalhada, no contexto de compatibilização de aplicações web desenvolvidas utilizando-se o JSF junto aos mecanismos de disponibilização offline oferecida por mecanismos do HTML5, refere-se à determinação do código da aplicação necessário para que seja possível a execução offline no cliente de forma independente.

Os requisitos mínimos para disponibilização offline de uma aplicação web, abordados anteriormente neste trabalho, estipulam a necessidade do armazenamento local dos recursos da aplicação, que incluem o código e recursos

da interface gráfica, código base de gerenciamento da aplicação, código contendo lógica de negócio e os dados relativos ao domínio.

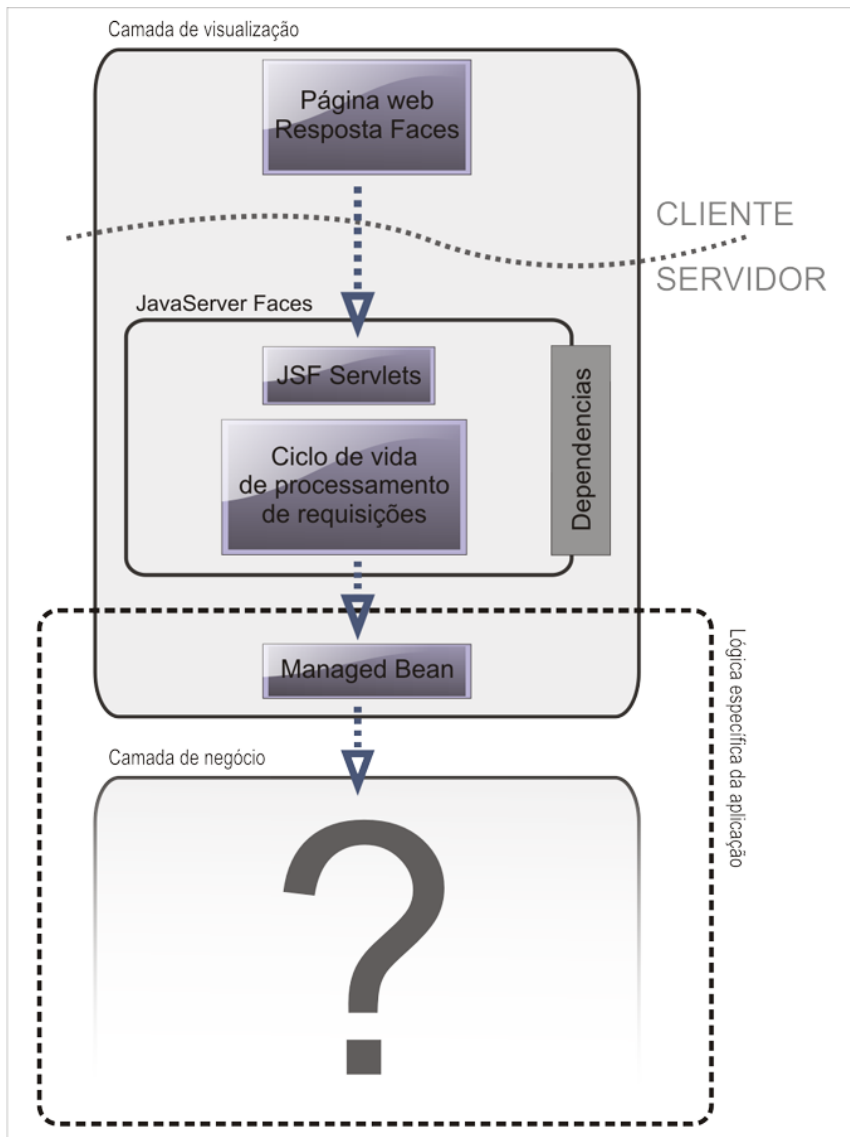


Figura 6: Aplicação web JSF 2.0 segundo arquitetura proposta pela especificação JavaEE.

5.2.1 JavaServer Faces

As responsabilidades do JSF englobam grande parte da geração da interface gráfica e seu gerenciamento, fornecendo uma porta de entrada ao resto do sistema. O ciclo de vida de processamento e tratamento de requisições, junto à geração de respostas, pode ser definido como o principal processo do framework, regendo todo o seu funcionamento. Todas as suas fazes realizam papéis importantes de acordo

com este contexto, sendo responsáveis pela renderização de toda a interface gráfica, pela gerência do estado da aplicação, validação e atualização de novos valores de entrada e também pela invocação da aplicação. Assim sendo, a execução de cada uma destas fases é absolutamente necessária em um contexto offline, diretamente e indiretamente implicando na dependência de praticamente todo o resto do framework.

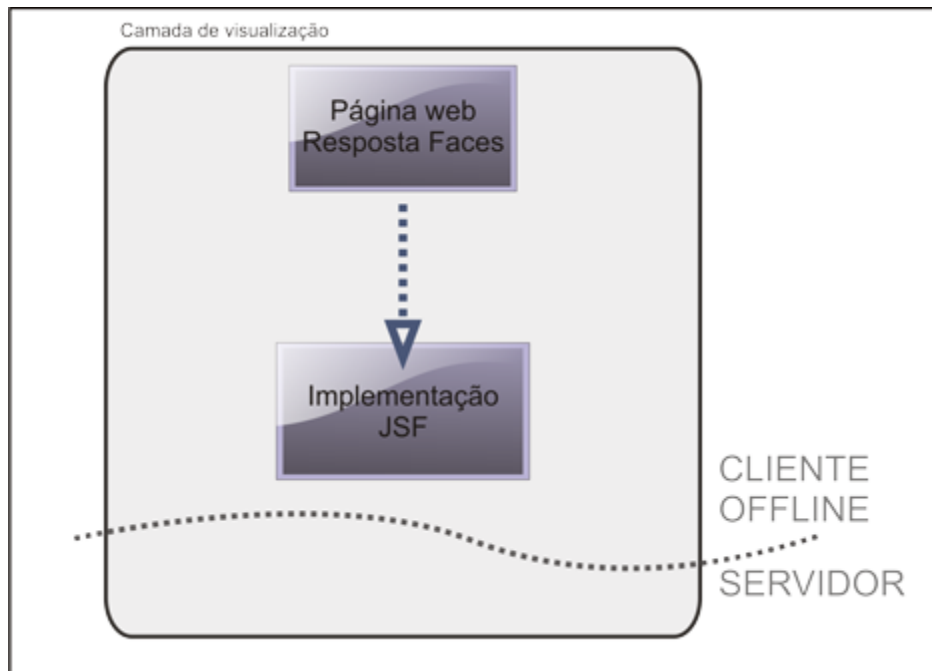


Figura 7: Aplicação web offline com o JSF 2.0 no cliente.

5.2.2 Particionamento da lógica de negócios entre cliente e servidor

No particionamento de aplicações web entre cliente e servidor, como anteriormente estabelecido no capítulo 2 (página 23), pode ser inviável a execução integral dos processamentos relativos ao servidor no cliente, pois existe uma série de responsabilidades inerentes do servidor que podem não ser realizadas de modo independente pelo cliente, de acordo com a natureza da aplicação.

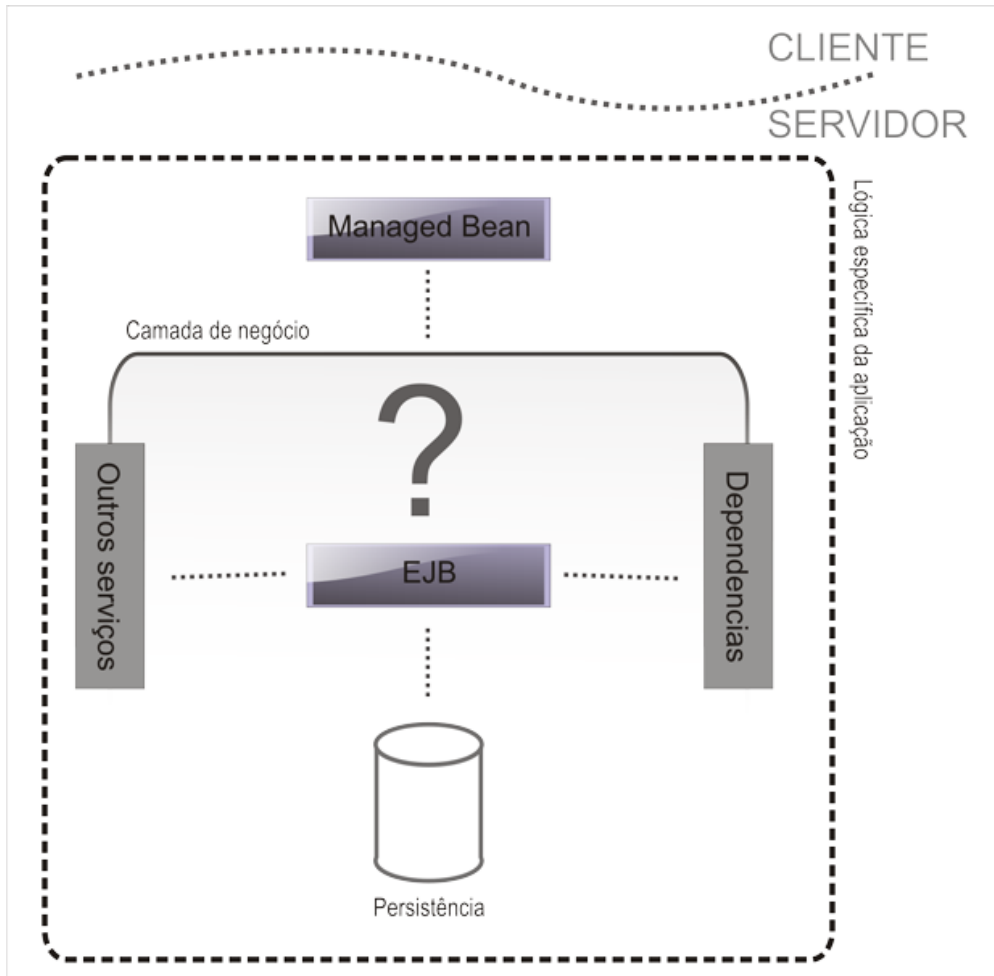


Figura 8: Lógica específica da aplicação.

Assim como qualquer outro software web offline que possua sua lógica particionada, e que quando online se comunique com o servidor, se faz necessário um mecanismo que possibilite a separação de lógica de negócio, aqui referido como *Server-mock*, determinando assim as ações a serem tomadas desde quando operações e dados não estiverem disponíveis até quando não possam ser delegados.

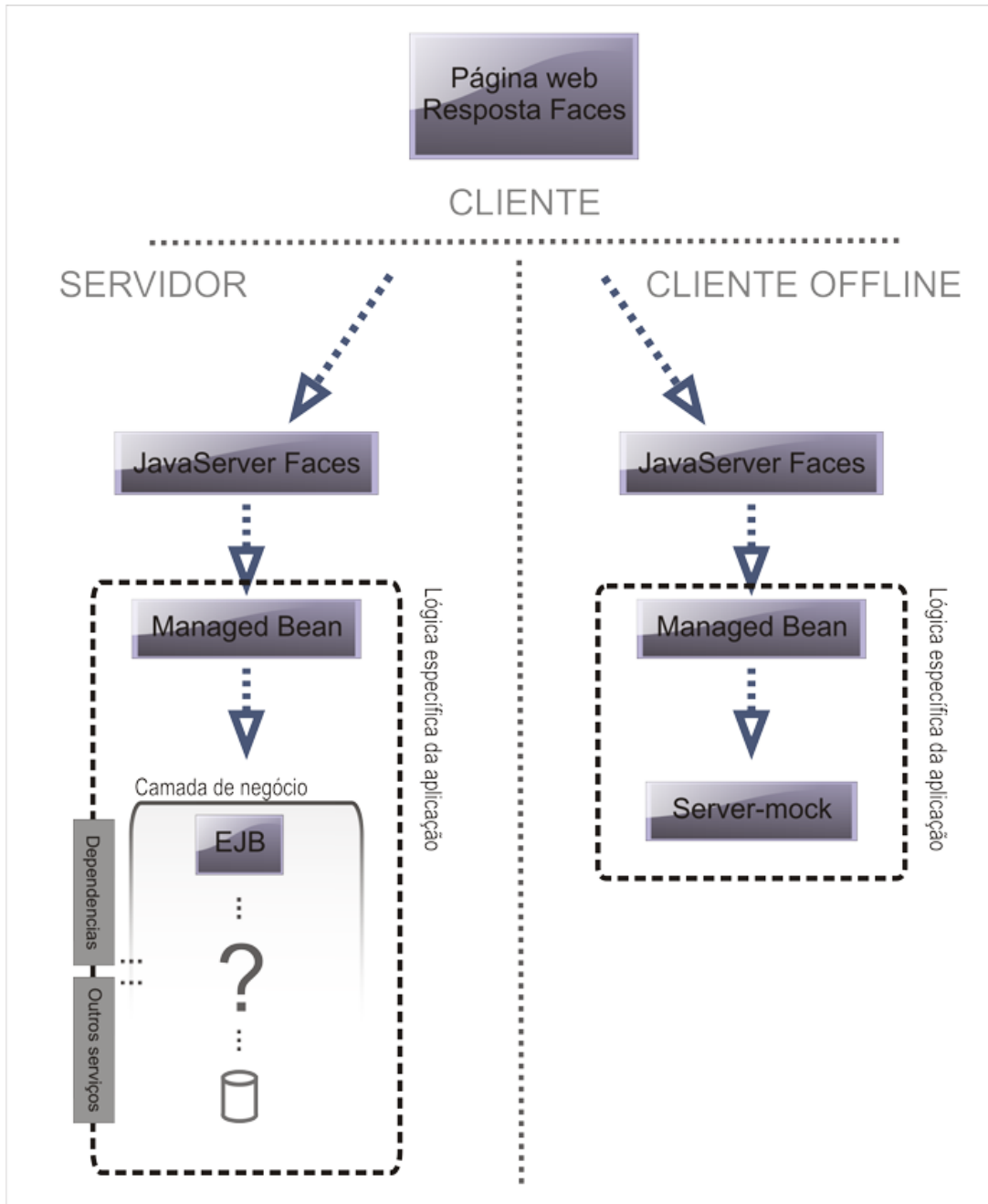


Figura 9: Separação de lógica da aplicação, representando as porções de lógica executadas no servidor e no cliente offline.

É importante ressaltar que uma vez definidas as características deste mecanismo, bem como suas necessidades e todas as tecnologias de suporte as quais a mesma teria acesso, sua implementação seria de inteira responsabilidade dos desenvolvedores do cliente web offline, de acordo com a natureza da aplicação, sua lógica e peculiaridades.

5.2.3 Código de lógica de negócios plausível ao cliente

A determinação da porção de código de lógica de negócio de uma aplicação necessário - e cabível - ao cliente de uma aplicação web é um problema de grande complexidade, pois - de acordo com decisões arquiteturais - seu código pode estar distribuído das mais diferentes maneiras, nas mais diversas partes de um software. Mesmo assim, esta tarefa de delimitação é de fundamental importância e deve ser realizada de maneira clara e objetiva, de modo a não inviabilizar a compatibilização proposta.

Durante o processamento de uma requisição, seguindo o contexto deste trabalho, o JSF depende de pontos de entrada de invocação a aplicação, normalmente referidos como *Managed Beans*. Sua principal responsabilidade resume-se em intermediar a comunicação entre as páginas da interface gráfica web (componentes do JSF) e o modelo da aplicação, fornecendo uma fachada de acesso a dados, escutando eventos, validando entradas e delegando ações e informações à camada de negócios, assim também representando função essencial em um cliente offline.

A camada de negócio, por sua vez, reúne – por concepção - a maior parte da lógica de negócio e, segundo os padrões propostos pelo JavaEE, deve ser implementada de forma totalmente independente da camada de visualização, comumente através do EJB. Devido a isto, esta camada deve não ser incluída em código levado ao cliente, no navegador do usuário, tanto porque pode conter lógica, dependências, referência de acesso a outros serviços online, e outros processamentos de execução ou mandatória no servidor ou não possível no cliente, quanto para manter coesão e coerência de design do software impostos pelos padrões existentes.

Estabelece-se assim, que a porção de código de lógica que deve ser trazida ao cliente pode ser delimitada pela separação entre a camada de visualização e a camada de lógica de negócios de uma aplicação JSF, determinando as fachadas de serviços EJB acessíveis a este código como as interfaces a serem tratadas pelo mecanismo responsável, o *Server-mock*, conforme a figura 10.

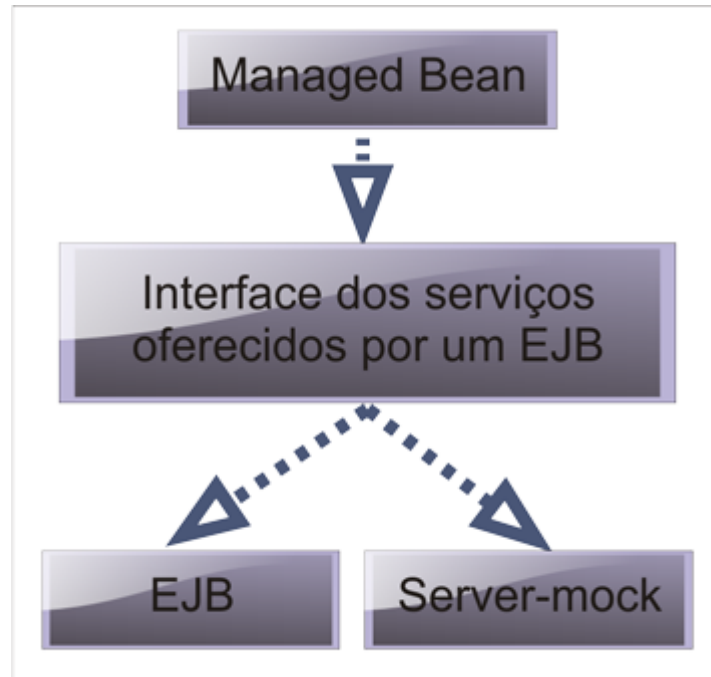


Figura 10: Duas diferentes implementações para a interface dos serviços oferecidos por um EJB, uma pelos processamentos no servidor e a outra pelas ações a serem realizadas no cliente quando offline.

5.2.4 Dados da aplicação

O particionamento de código entre servidor e cliente também implica na carência de processamentos e dados - normalmente disponibilizados pelo conjunto de serviços do servidor - não acessíveis pelo cliente offline, que podem comumente representar fundamental importância para o funcionamento da aplicação. Assim, a implementação alternativa dos serviços indisponíveis, ou seja, o Server-mock, deve ser responsável pelo acesso de - ao menos - todo o conjunto de dados fundamentais, e para que tal função seja possível, os mesmos devem estar armazenados e disponíveis de alguma maneira, em contexto offline.

Do mesmo modo, todo o conjunto de dados gerados pelo cliente offline, bem como informações de representação de todos os processamentos requisitados, deve ser armazenado, para que possam ser posteriormente atualizados e executados no servidor, de modo a manter a pertinência e integridade do estado da aplicação como um todo. Ambas a disponibilização dos dados da aplicação em ambientes offline e a atualização do servidor segundo qualquer mudança depende da existência de um

mecanismo responsável pela sincronização bilateral de dados entre cliente e servidor, assim como dependem de meios para o armazenamento e acesso a dados persistentes no cliente.

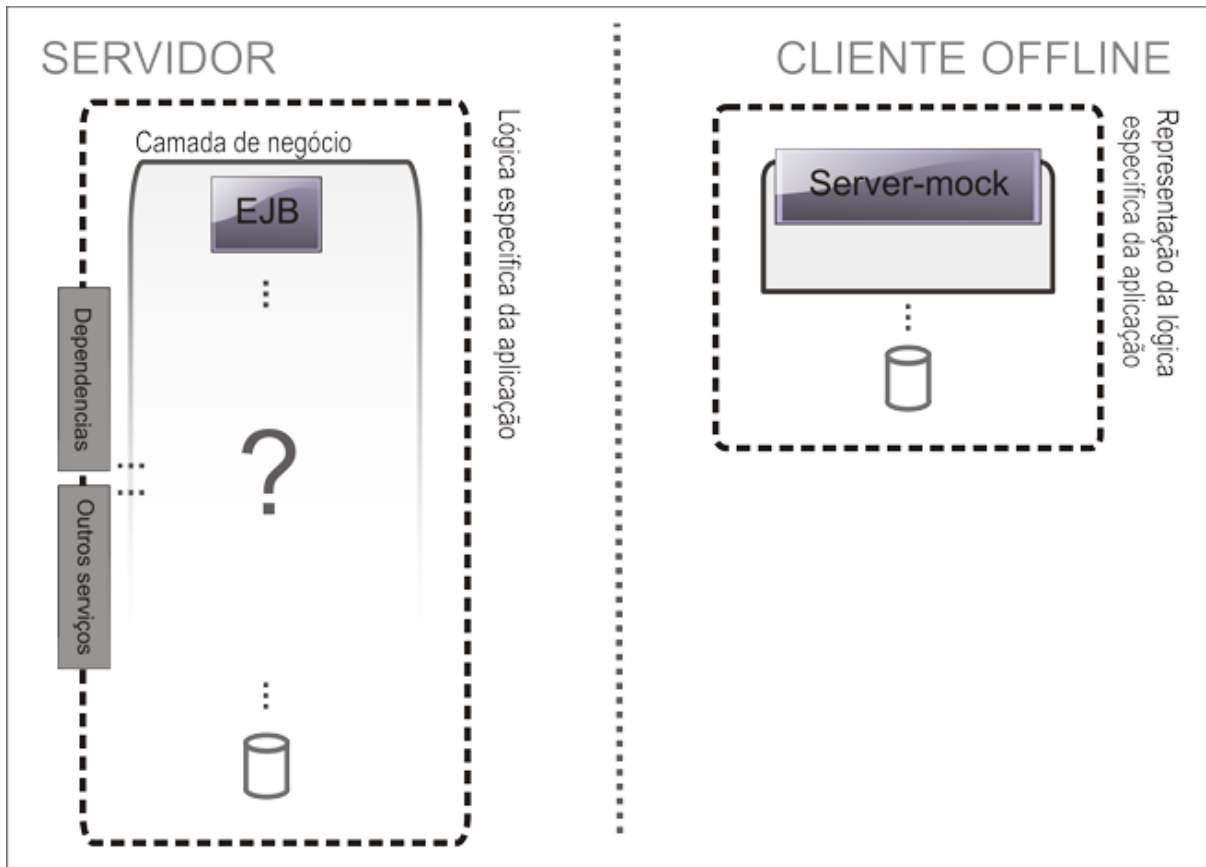


Figura 11: Acesso e armazenamento de dados necessário também pelo cliente offline.

O HTML5, segundo sua especificação, fornece dois mecanismos distintos primários para o armazenamento de dados de uma aplicação, disponibilizando seu acesso tanto quando online quanto quando offline, como descrito em detalhe anteriormente neste trabalho, no capítulo 3. Ambos os mecanismos são acessíveis diretamente pelo navegador e através do uso de JavaScript, provendo acesso a um conjunto de APIs para a manipulação de bases de dados relacionais no cliente e para o armazenamento - e acesso - persistente de dados baseados em pares de chaves-valor.

Em conjunto, as duas tecnologias viabilizam o armazenamento - para acesso futuro - de todos os tipos de dados da aplicação que podem vir a ser necessários neste cenário, incluindo dados salvos durante acesso online, para acesso posterior

offline, e dados salvos durante a utilização offline do software, que necessitem ser posteriormente disponibilizados para o servidor.

5.3 Execução de código independente do servidor no cliente

O ambiente web foi concebido com a finalidade de representação e exibição de documentos, e embora esteja tomando novos rumos, ainda provê muitos desafios ao desenvolvimento de aplicações. Uma vez delimitadas as porções de código nos quais as execuções se fazem necessárias em clientes offline, é preciso determinar os meios para que sua execução seja possível. Cada diferente tecnologia utilizada na compatibilização do HTML5 com o JSF 2.0 requer diferentes soluções. É necessário viabilizar o armazenamento e processamento de código em linguagem Java, da interface gráfica e de recursos diversos.

5.3.1 Interface gráfica e o cacheamento de recursos

A interface gráfica de aplicações web JSF já é naturalmente gerada para exibição nos navegadores modernos, e através do uso do mecanismo de cacheamento de recursos do HTML5, poderia ser cacheada junto a seus recursos através da disponibilização de um manifesto.

É importante, porém, notar que - deste modo - se faz necessário o estabelecimento prévio de todas as páginas e recursos a serem salvos para posterior uso. Conseqüentemente, nesta abordagem, quaisquer recursos que devam ser gerados dinamicamente tornam-se estáticos, ou seja, toda e qualquer página web JSF - que faça uso de dados variáveis ou relacionados à execução de uma lógica - seria armazenada de acordo com o seu estado no momento em que o cacheamento de recursos fosse realizado.

Esta restrição, que em uma análise superficial pode aparentar um baixo impacto na compatibilização de uma aplicação, então representa, segundo os conceitos arquiteturais do JSF e a fina camada de interface gráfica conseguinte, a impraticabilidade do cacheamento da totalidade de páginas de interfaces gráficas JSF. Isto ocorre, pois mesmo que - com restrições - seja possível armazenar algumas páginas geradas por repostas Faces, a exibição de respostas decorrentes

do processamento de subseqüentes requisições Faces depende diretamente da geração de novos conteúdos, ou seja, conteúdos dinâmicos. Por mais que em experimentos o cacheamento de páginas JSF tenha sido possível, verificou-se que a dependência nos dados relacionados às páginas de resposta, necessita explicitamente da geração dinâmica do conteúdo e impossibilita qualquer tentativa de cacheamento de recursos não estáticos.

Assim sendo, estabelecendo-se a existência de dependência de geração dinâmica de conteúdo junto à execução dos processamentos do JSF e regras de negócio envolvidas, se faz desnecessário o cacheamento parcial de respostas Faces de forma estática. Todos outros recursos, como figuras e outros arquivos estáticos podem perfeitamente utilizar estes mecanismos de armazenamento fornecidos pelo HTML5.

5.3.2 Código Java da aplicação

O JavaServer Faces é, bem como comumente também é toda a lógica de negócio em projetos JEE, implementado em Java e, como abordado previamente, não existe nenhum método padronizado para execução de código desta linguagem em navegadores web. Porém, existem algumas alternativas não triviais que poderiam viabilizar a solução deste problema: a utilização de Java Applets para disponibilização de código Java no cliente junto à criação de Render Kits do JSF especializados na criação de páginas web offline que utilizem o Applet ao invés do servidor; a criação de plug-ins para navegadores web que executem o código Java necessário quando offline, interceptando chamadas feitas por páginas da aplicação; e a compilação de código Java para JavaScript, junto à criação de mecanismos de controle simplificados do framework, possibilitando a utilização do cacheamento de recursos para o armazenamento de códigos equivalentes ao processamento de requisições e geração de respostas Faces.

5.3.3 Plug-ins para navegadores

Todos os navegadores modernos, que também suportam o HTML5, fornecem mecanismos para a criação e uso de plug-ins, permitindo assim a extensão de suas

capacidades, modificando funcionalidades existentes ou adicionando novos comportamentos a sua lógica. Esta característica viabiliza – dentre outras coisas - o tratamento de novos tipos de conteúdo, a interceptação de chamadas HTTP a servidores e a execução de código Java.

Através do desenvolvimento de provas de conceito, pode-se estabelecer que a criação de plug-ins para o suporte a aplicações offline desenvolvidas com o JavaServer Faces poderia prover todo o apoio necessário ao funcionamento das mesmas. Cada aplicação necessitaria um plug-in específico contendo o código necessário para seu funcionamento, incluindo, através da realização de detecção da indisponibilidade de acesso ao servidor, a interceptação de chamadas destinadas ao servidor, permitindo a execução do ciclo de vida de processamento de requisições Faces, dos respectivos códigos de lógica de negócio e da geração da resposta equivalente.

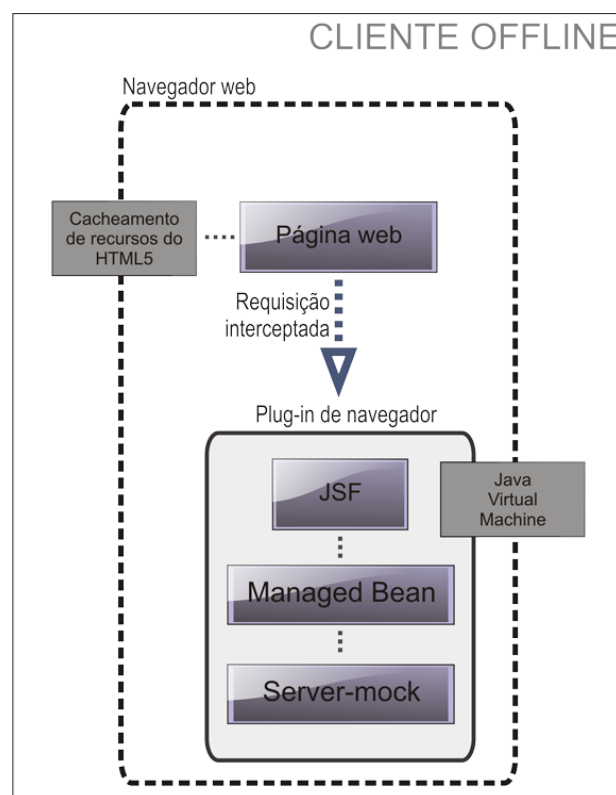


Figura 12: Utilização de plug-in para execução de código Java do cliente offline.

Contudo, a aplicação desta abordagem levantou algumas conseqüências que impactam negativamente a sua viabilidade, dentre as quais se pode citar:

- Necessidade de criação de plug-ins específicos para cada navegador, devido às disparidades entre suas implementações, principalmente relacionadas às tecnologias e mecanismos suportados para a utilização de plug-ins.
- Impossibilidade de criação de um único plug-in - por navegador - não específico a uma única aplicação, decorrente das configurações e da lógica de negócio particulares de cada problema.
- Grande dificuldade relacionada à automatização da confecção de cada plug-in necessário, tarefa não trivial que necessitaria ser realizada a cada modificação, tanto no ambiente de produção quanto na aplicação em si.
- Tamanho demasiadamente grande de cada plug-in, considerando os padrões de velocidade de transferência de arquivos na internet em relação ao tamanho das bibliotecas que seriam necessárias, como o JSF, que possui – de acordo com suas diferentes implementações – entre seis e nove Megabytes, se levando em conta também a necessidade de download, pelo sistema do cliente, a cada nova versão, para armazenamento em seu computador.
- Possui como pré-requisito e condição básica a dependência de uma JVM (Java Virtual Machine) compatível na máquina cliente.

5.3.4 Java Applets

Um Applet é um programa escrito na linguagem Java que pode ser incluído em uma página HTML, de maneira muito semelhante a que uma imagem pode ser incluída e, segundo sua documentação, são passíveis de cacheamento - podendo ser armazenados no navegador - para utilização posterior quando offline. Os applets têm a capacidade de interação bidirecional com JavaScript e também permitem a inclusão de dependências empacotadas como arquivos JAR.

Sua utilização, no contexto da problemática abordada por este trabalho, permitiria a execução da porção de código Java levada ao cliente na própria página web. Para que tal abordagem seja possível, dadas as características e desafios do

problema, seria necessário o desenvolvimento de alguns módulos de gerência e integração das tecnologias envolvidas, assim como adaptações ao framework JSF.

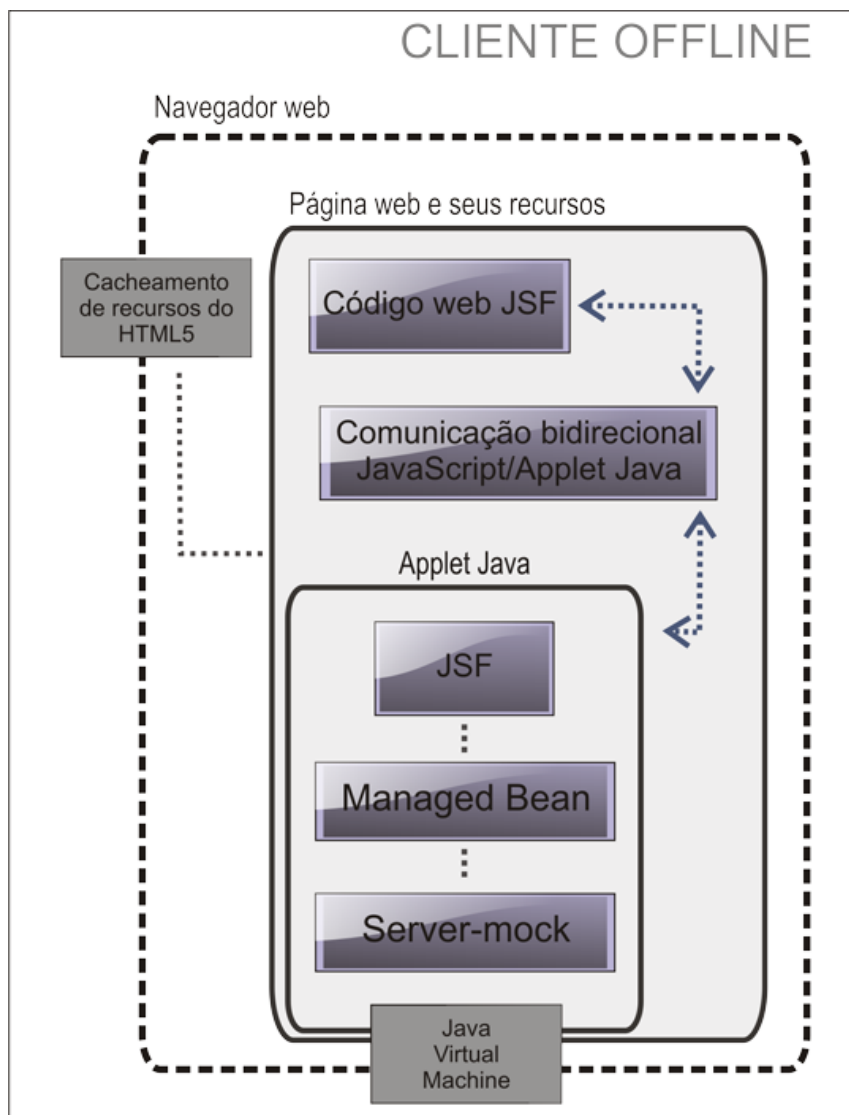


Figura 13: Utilização de Applet Java para execução de código Java do cliente offline.

Conforme representado pela figura 13, far-se-ia preciso o cacheamento de uma série de páginas provendo o acesso inicial – de acordo com a requisição feita pelo usuário - ao mecanismo de controle offline, um applet cuja função seria a execução do código Java necessário para processamento das requisições e renderização das páginas requisitadas, implementando uma fachada de acesso ao JSF alternativa à existente - que faz uso de servlets. Um mecanismo de integração entre o JavaScript e o Applet Java realizaria a comunicação entre ambas as

tecnologias, repassando os dados equivalentes de cada requisição ao applet e aplicando a resposta ao estado atual da interface gráfica.

Também seria imprescindível, dando suporte a este processo, a customização da renderização de respostas, através da criação de um RenderKit especializado, para que requisições não fossem destinadas ao servidor, mas sim ao mecanismo de tratamento correspondente.

A utilização de Java Applets como alternativa para execução da porção de código Java dos processamentos necessários no cliente possui as seguintes considerações e implicações:

- Criação de um applet por aplicação, contendo configurações e da lógica de negócio particulares da mesma, sendo que, um único applet pode ser utilizado em diferentes navegadores.
- Necessidade de automatização da confecção de cada applet de suporte a aplicação offline, tarefa não trivial que necessitaria ser realizada a cada modificação da aplicação.
- Applets teriam relativamente grande tamanho, pois teriam que armazenar código relativo ao JSF, lógica de negócios da aplicação e outros módulos complementares. De acordo com uma estimativa grossa teriam no mínimo seis Megabytes e necessitariam ser baixados no sistema do cliente a cada nova versão.
- Possui como pré-requisito e condição básica a dependência de uma JVM (Java Virtual Machine) compatível na máquina cliente.

5.3.5 JavaScript

A utilização de JavaScript, a linguagem padrão para execução de código no cliente, suportada por todos os navegadores modernos, implicaria na criação de uma solução que reimplementasse os processamentos relativos ao JSF, que incluem o ciclo de vida de processamento de requisições Faces, e conseqüentemente, necessitaria a implementação de seus componentes, tratamentos, validações e lógica. Esta alternativa é inviável devido à alta complexidade envolvida em todos estes requisitos, mesmo que se fizesse possível a compilação de código de lógica de negócio de Java para JavaScript – problema abordado por diferentes projetos.

5.4 Integridade e sincronização de dados

A criação de aplicações web que possam capacidade de funcionamento também quando offline, sem o acesso ao servidor, implica no desenvolvimento de clientes que possam ser executados de forma independente de rede. Conseqüentemente, estes clientes offline necessitam formas alternativas para o acesso a dados requeridos para seu funcionamento – que normalmente seriam disponibilizados pelos serviços oferecidos pelo servidor – e para o armazenamento de dados que devem posteriormente ser atualizados no servidor, permitindo que sessões de uso offline por usuários se façam válidas no contexto dos serviços do próprio servidor.

Faz-se necessário, então, um mecanismo que realize todos os métodos de sincronização de dados entre cliente e servidor, sendo responsável pela manutenção da integridade do estado da aplicação, disponibilização de dados de servidor a clientes e realizando a execução de serviços do servidor de acordo com as ações requisitadas por usuários.

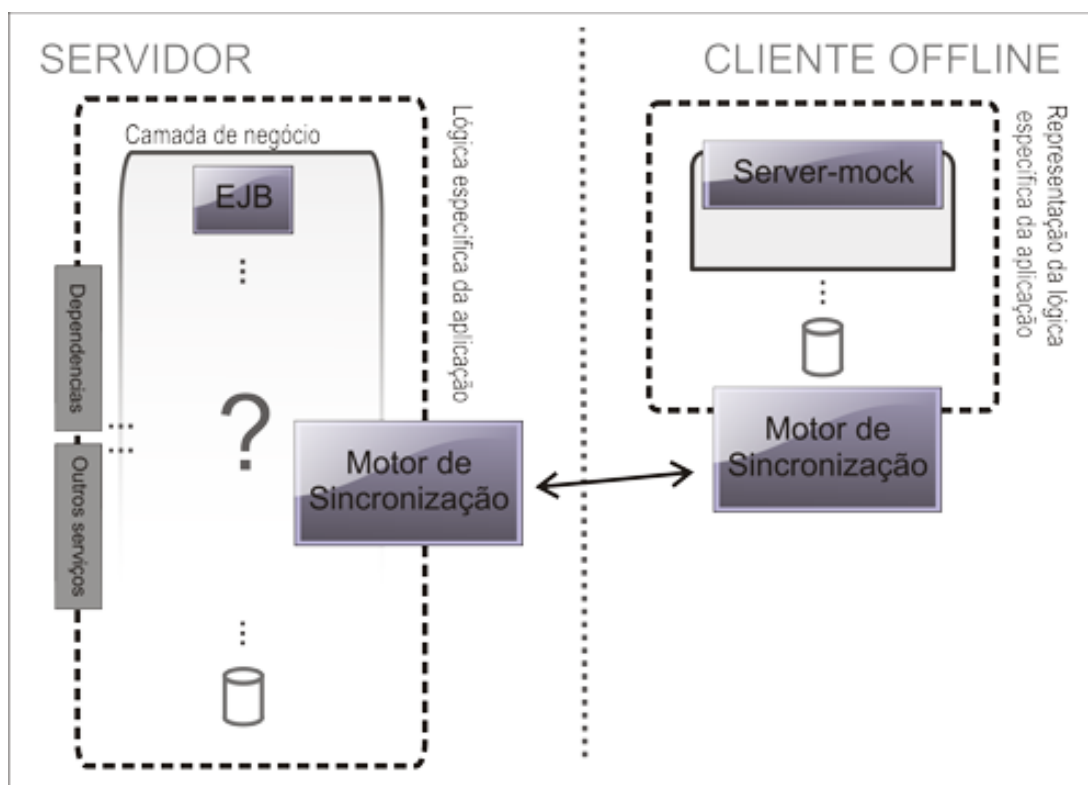


Figura 14: Motor de sincronização de dados.

Processos de sincronização são diretamente dependentes do domínio da aplicação, seus serviços e das decisões arquiteturas do software no particionamento da aplicação. A simples divisão de processamentos entre cliente e servidor exerce influência na implementação da sincronização de dados. Processamentos têm como pré-condição a manipulação e transformação de dados e, por conseguinte, a não disponibilização integral da lógica de uma aplicação em um cliente offline - e as próprias escolhas de particionamento da mesma – determinam a necessidade de gerenciamento de um conjunto de dados intermediários, ou seja, dados equivalentes às entradas e retornos dos processamentos disponíveis somente no servidor.

A concepção deste mecanismo de sincronização bilateral entre cliente e servidor é, portanto, uma tarefa dependente da natureza da aplicação na qual o mesmo deverá atuar, não sendo possível que sua implementação seja realizada de forma genérica, para diferentes aplicações.

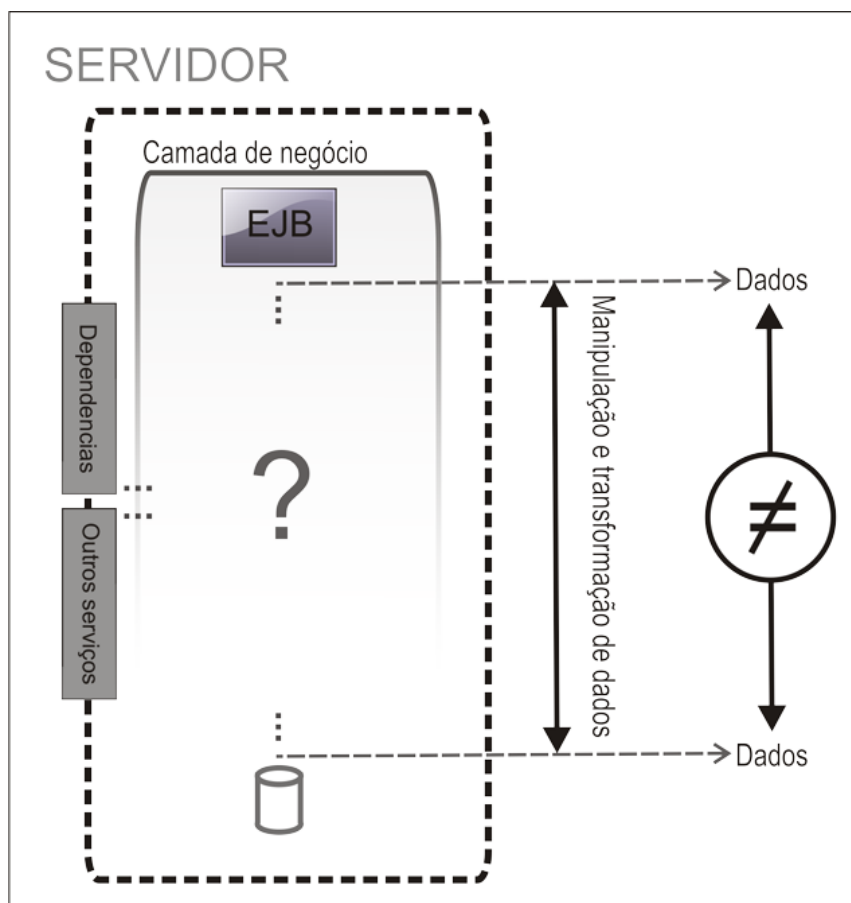


Figura 15: Impacto de processamentos relacionados à lógica de negócio sobre dados de uma aplicação.

Tarefas de sincronização devem também ser desenvolvidas prevendo conflitos decorrentes da limitada camada de lógica disponível no cliente e, também por conta disto, das restrições impostas aos dados que podem ser processados. Considerando que a ordem de uma série de processamentos pode exercer influência no resultado final de uma transação, tem-se que a execução de métodos de implementação alternativos de processamentos do servidor - responsáveis pela determinação das ações a serem executadas durante a ausência do acesso aos processantes originais – deve ser refletida no servidor de forma seqüencial. Como resultado, temos que a simples existência de conflitos na execução de qualquer uma destas operações de sincronização pode, então, resultar na inconsistência do estado do cliente, ou seja, inviabilizar todo o trabalho realizado em um cliente durante uma sessão offline, impossibilitando a realização de sincronização.

Da mesma maneira, temos que a utilização do Server-mock para implementação alternativa de processamentos simplificados pode dificultar o processo de sincronização, pois a realização de processamentos parciais aumentaria a chance de inconsistência de estado da aplicação. Esta pratica é, então, não recomendada, e caso necessária, ocasiona na necessidade de criação de processos capazes de “tradução” dos resultados destes processamentos simplificados nos resultados equivalentes aos processamentos reais.

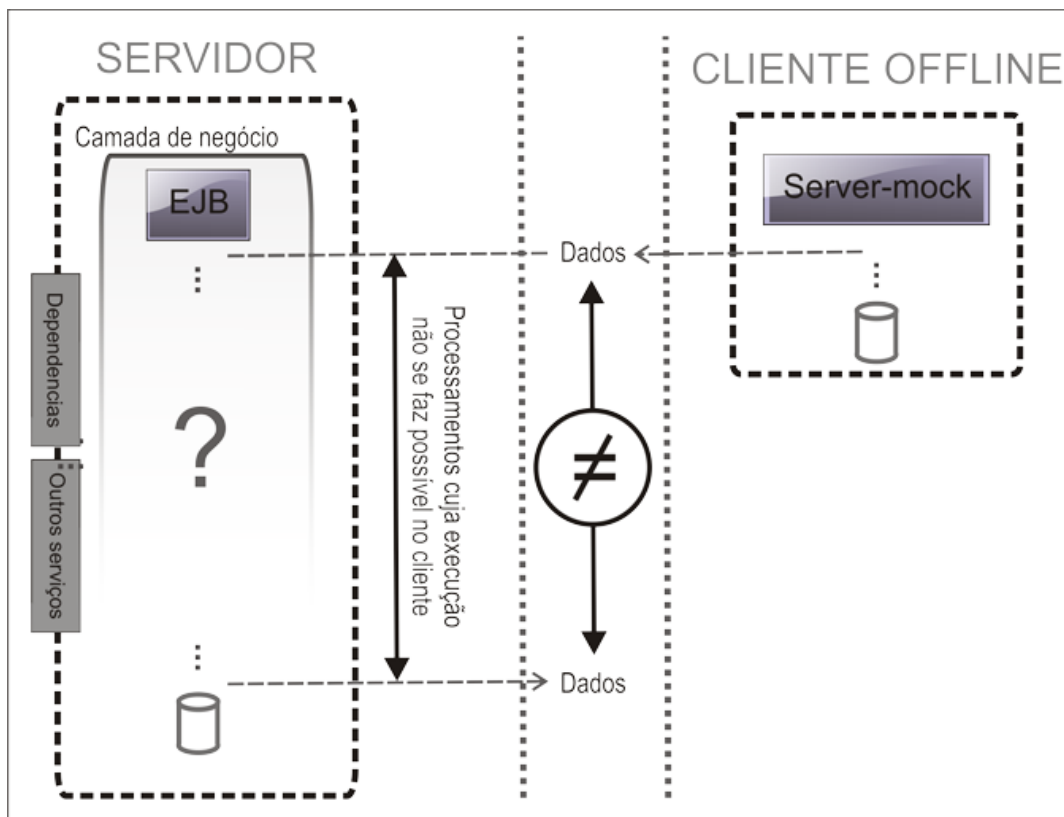


Figura 16: A não equivalência de dados armazenados no cliente e no servidor, demonstrando a necessidade de armazenamento no cliente de dados já processados no servidor e da execução de processamentos no servidor relativos aos dados coletados no cliente.

É, também, plausível estabelecer que possam existir modificações de dados em mais de uma das partes envolvidas entre a realização de operações de sincronização. Cada software deve determinar previamente, se possível, um conjunto de regras para o tratamento de erros causados por diferenças de conjuntos de dados como, por exemplo, em casos em que processamentos requisitados em clientes offline utilizem dados não atuais ou não mais existentes.

A disponibilidade limitada de lógica no cliente agrava ainda mais este problema. A própria possibilidade de diferença entre os tipos de dados passíveis de manipulação e armazenamento em um cliente e no servidor, como demonstrado na figura 16, gera grande complexidade na gerência de processos de sincronização, em alguns casos, de acordo com a natureza e complexidade da aplicação, pode tornar impossível a detecção e tratamento de conflitos.

6 – CONCLUSÕES

Através do estudo das possibilidades de compatibilização de aplicações web desenvolvidas utilizando-se o framework JavaServer Faces 2.0 com as facilidades de disponibilização de aplicações offline oferecidas pelo HTML5, junto à implementação de diversas provas de conceito, foi possível concluir que a efetivação dos fins contemplados pelo estudo realizado por este trabalho é possível, porém, somente apresentando resultados extremamente limitados e demandando enorme esforço, não gerando, portanto, grande valor, ou seja, que a integração das tecnologias é tecnicamente factível porém inviável em sua prática.

Durante a realização deste trabalho, pôde-se determinar o conjunto de desafios e de soluções relativos à integração das tecnologias propostas. Neste processo, através da criação de experimentos práticos, cada solução pôde ser validada e estudada, assim não só realizando-se a prova das conjecturas teóricas feitas ao longo deste trabalho, mas também permitindo que cada abordagem pudesse ter suas características e conseqüências elencadas e avaliadas, determinando-se as vantagens e desvantagens de cada elemento da solução.

A integração das tecnologias propostas implica em um conjunto de limitações relacionado à necessidade de adaptação do próprio framework JSF 2, da especificação e criação de uma série de complexos mecanismos de suporte às funcionalidades fornecidas pelo HTML5 de acordo com os desafios existentes no problema de compatibilização, da impossibilidade de realização de escolhas de particionamento da lógica de negócio da aplicação e, principalmente, da quebra dos padrões estabelecidos pelo JEE e da enorme complexidade para sincronização que seriam resultantes da implementação de qualquer lógica não trivial no cliente da aplicação.

Assim, a criação de uma aplicação JSF offline poderia prover poucas funcionalidades e uma pobre experiência ao usuário em relação ao alto custo para seu desenvolvimento, o que faz com que a sua própria concepção deva ser questionada: a da criação de aplicações web offline pode não fazer sentido quando os próprios serviços que justificariam a sua criação seriam limitados a pobres implementações.

Dentre os fatores que desfavorecem a utilização do JSF como tecnologia Web cliente, executando em conjunto com navegadores ao invés de com containers web no servidor, pode-se ressaltar:

- Incoerência com a cultura de alta disponibilidade de serviços;
- Necessidade de implementação de complexos mecanismos específicos ao modelo de dados e serviços oferecidos por cada aplicação;
- Dependências na máquina do cliente;
- Não atendimento nativo a RIA (*Rich Internet Applications*);

6.1 Incoerência com cultura de alta disponibilidade de serviços

A cultura de utilização do JSF promove alto acoplamento entre as camadas de visualização e de serviços. Este acoplamento torna praticamente intangível a execução de uma aplicação JSF concebida para ser executada no servidor como cliente offline e, mesmo que possível diante do emprego de restrições, a indisponibilidade de serviços ocasionaria em estorvos para a experiência do usuário, inclusive em aplicações simples.

6.1.1 Inflexibilidade no particionamento de lógica

De acordo com as características impostas pelo JSF, o particionamento de código entre cliente e servidor - conforme estabelecido previamente pelo item 5.2.3 (página 53) – define a inviabilidade de execução de maior parte da lógica de negócio de uma aplicação em clientes web, implicando na impossibilidade de execução de quaisquer serviços e processamentos não implementados ou na camada de visualização.

Estas limitações dos clientes offline não somente estabelecem restrições ao número de processamentos disponíveis, mas também sua possível complexidade, tornando impraticável o desenvolvimento de aplicações JSF sofisticadas para ambientes offline.

6.1.2 Desafios inerentes às tarefas de sincronização

O alto acoplamento de camadas promovido pelo JSF também tem como consequência inúmeros problemas de alta complexidade relacionados à sincronização de dados e processamentos.

A arquitetura de software, encorajada pela separação de camadas ditada pelas convenções do JavaEE e do JSF, estabelece grande dependência da visualização aos serviços da camada de lógica da aplicação para o acesso a dados e realização de processamentos. Em um ambiente offline, para que exista a possibilidade de atualização posterior do estado do servidor, se faz necessário o armazenamento e sincronização de toda e qualquer operação realizada na visualização durante uma sessão. Para garantir a consistência do estado de uma aplicação durante a consumação destes processos, determinam-se também dependências relacionadas à ordem de execução de processamentos e da congruência dos dados resultantes entre serviços no cliente e no servidor.

Como consequência, temos que uma grande complexidade surge devido às possibilidades de divergência entre a realização de operações em um ambiente offline e no servidor, que podem ser ocasionados por uma ampla diversidade de situações comuns. Como exemplo, pode-se citar como causa de conflitos a comum possibilidade de que dados mantidos em um cache não estejam atualizados e a não disponibilização, simulação ou até previsão de resultados de serviços e processamentos no cliente. Esta problemática é tão crítica que, de acordo com os requisitos estabelecidos, a simples ocorrência de conflito, erro ou discrepância de resultados de processamentos durante a execução de tarefas de sincronização, causaria a perda da sessão offline ou, se não tratado, na inconsistência do modelo de dados da aplicação.

6.2 Necessidade de complexos mecanismos específicos a cada aplicação

A integração de aplicações web JSF com ambientes web offline também tem como requisito a implementação de uma série de mecanismos com a finalidade de suporte aos diferentes aspectos de compatibilização. Além das necessidades implícitas da adoção de qualquer uma das abordagens estudadas para execução de

código Java no cliente, faz-se necessário também implementar complexos mecanismos diretamente dependentes da natureza de cada aplicação.

O motor de sincronização, por exemplo, possui relação intrínseca com a lógica de negócios, o modelo de dados e com as decisões arquiteturais de uma aplicação e, seu desenvolvimento, pode abranger grandes complexidades podendo até ser impossível, vide o item 5.4 na página 62. Também dependente da natureza de cada aplicação, temos que a implementação do Server-mock, como referido no capítulo 5, deve ser desenvolvida de acordo com os serviços oferecidos por um servidor, responsabilizando-se, em suma, pela gerência dos mecanismos de armazenamento de dados offline e pela execução de simples processamentos para contenção da falta de acesso a serviços reais.

Além disto, a viabilização de execução do código originário do servidor em um cliente somente pode ser realizado, no contexto deste trabalho, através da criação de plug-ins para navegadores ou de applets Java. A primeira alternativa requer a criação de diferentes plug-ins específicos para cada navegador web, cada um sendo responsável por interceptar chamadas ao servidor indisponível, reproduzir as características de um servidor de aplicações, provendo o suporte inclusive à utilização de servlets e a todos os requisitos necessários para a execução do JSF. A utilização de Applets Java, por sua vez, requer adaptações ao framework JavaServer Faces e a implementação de mecanismos que provenham comunicação entre código JavaScript e Java da aplicação.

6.3 Dependências na máquina do cliente

Todas as duas abordagens viáveis elencadas para a mitigação da necessidade de execução de código Java no cliente possuem características que impactam negativamente sua aplicabilidade, em termos de requisitos às máquinas nas quais clientes offline seriam executados. Independentemente da alternativa selecionada como solução para desenvolvimento de uma possível aplicação web offline, ou seja, destacando a intersecção de requisitos dentre todas as abordagens, temos como necessidade:

- A disponibilidade de uma máquina virtual Java (JVM) compatível instalada na máquina do cliente.

- Download de um pacote de recursos - seja na forma de plug-in para navegador ou de Applet Java - a cada mudança na aplicação, considerando-se o um grande tamanho devido à necessidade de empacotamento de uma implementação do JSF junto a todas as duas dependências, os mecanismos cuja criação é imperativa para execução do cliente offline e todos os recursos que devem ser cacheados.
- Criação de mecanismos de suporte.

Todos estes requisitos impõem barreiras no processo de adoção por usuários, tornando a utilização da aplicação web não natural, desfavorecendo-a. Dada a possibilidade de criação de aplicações web offline através de outras tecnologias, que sob o ponto de vista do usuário necessitariam configuração zero para seu funcionamento, o uso destas abordagens poderia ser considerado proibitivo, quando o público alvo não for versado em tecnologia, de acordo com pontos de vista mercadológicos.

6.4 Não atendimento nativo a RIA

A iniciativa de criação de aplicações web, junto à de disponibilização das mesmas em ambientes sem acesso à internet, se deve à possibilidade de criação de aplicações de internet ricas (RIAs), que permitem a execução de funcionalidades equivalentes às fornecidas por softwares tradicionais do tipo Desktop e/ou provêm uma experiência rica ao usuário, seguindo a filosofia Web 2.0.

A tecnologia JSF, por sua vez, foi criada para atender aplicações Web 1.0, nas quais toda a renderização das interfaces gráficas é realizada no servidor, dada uma requisição HTTP. Através de grandes esforços, se fez possível a compatibilização do JSF – pelo desenvolvimento de frameworks suplementares e, posteriormente, pela criação da especificação 2.0 – para a viabilização de aplicações que pudessem prover experiência de páginas dinâmicas ao usuário. Contudo, a filosofia de geração clientes finos, gerenciados e renderizados pelo servidor, impõe amplas limitações para com o dinamismo de interfaces gráficas.

Assim, pode-se estabelecer que a compatibilização de aplicações JSF em ambientes offline não pode, por si só, atender a seus objetivos, não influenciando no fornecimento de experiências ricas ao usuário. Considerando-se, então, que a

própria compatibilização de aplicações JSF também se faz possível somente através de amplas e severas limitações em suas funcionalidades, pode-se finalmente concluir que absolutamente não existem vantagens na realização da migração desta tecnologia e que quaisquer esforços nesta direção seriam improdutivos.

6.5 Considerações finais

A busca pelo estabelecimento de um patamar maduro para desenvolvimento de aplicações sobre a plataforma web permanece como um desafio constante para desenvolvedores e cientistas da computação em geral.

Este trabalho procurou contribuir com esta busca incansável através da compatibilização de uma tecnologia importante e bem estabelecida – o JSF – com o conceito de criação de aplicações web offline. Todo o estudo foi realizado baseando-se nas características impostas pelas tecnologias envolvidas, assim como pela própria plataforma web e das conseqüências das possíveis alternativas de soluções relacionadas ao escopo deste trabalho. Com base nas informações coletadas, pôde-se estabelecer a inviabilidade prática da criação de aplicações JSF web offline, de acordo com os desafios elencados e de análises conceituais dos conceitos correlacionados.

De maneira geral, concluo que o JSF é uma excelente tecnologia para Web 1.0, fim para o qual foi projetada. As grandes vantagens da utilização desta tecnologia, porém, desaparecem quando se está em ambiente cliente, pois mesmo que tecnicamente possível qualquer esforço na migração desta tecnologia seria em vão, visto que existem tecnologias muito mais maduras para entregar aplicações Web 2.0.

A demanda de desenvolvimento de aplicações para a internet cresce continuamente, assim como o nicho destas aplicações. Acredito, então, que existe espaço o suficiente para que mais de um framework de criação de aplicações web seja utilizado pela especificação JavaEE, ou que, pelo menos, não se cultive a crença de que a mesma determina as melhores tecnologias a serem utilizadas em todo tipo de projeto.

6.6 *Trabalhos Futuros*

Uma proposta de continuidade para esta pesquisa é o estudo em torno da problemática de compatibilização dos mecanismos oferecidos pelo HTML5 para criação de aplicações offline com outros frameworks de desenvolvimento de aplicações para a web e, se possível, a criação dos respectivos protótipos. O GWT (Google Web Toolkit), por exemplo, aparenta ser arquiteturalmente compatível com o modelo necessário para a criação de aplicações cliente com grande independência do servidor.

7 – REFERÊNCIAS

APPLET Caching. **Developer Resources for Java Technology**. Disponível em: <http://java.sun.com/javase/6/docs/technotes/guides/plugin/developer_guide/applet_caching.html>. Acesso em: 20 Março 2010.

BERNERS-LEE, T. Information Management: A Proposal. **W3C**, 1989. Disponível em: <<http://www.w3.org/History/1989/proposal.html>>. Acesso em: 2 Janeiro 2010.

BERNERS-LEE, T. WorldWideWeb: Proposal for a HyperText Project. **W3C**, 1989. Disponível em: <<http://www.w3.org/Proposal>>. Acesso em: 05 jan. 2010.

BURNS, E.; KITAIN, R. **JavaServer Faces Specification**. Sun Microsystems Inc. Santa Clara. 2009.

CLIENT-SIDE Scripting and HTML. **W3C**, 1997. Disponível em: <<http://www.w3.org/TR/WD-script-970314>>. Acesso em: 05 jan. 2010.

GINIGE, A.; MURUGESAN, S. Web Engineering: A Methodology for Developing Scalable, Maintainable Web Applications. **Cutter IT Journal**, Julho 2001. 24-35.

HICKSON, I. HTML5. **W3C**, 2010. Disponível em: <<http://dev.w3.org/html5/spec/>>. Acesso em: 20 Fevereiro 2010.

HOLDENER, A. **Ajax: The Definitive Guide**. 1 Edição. ed. Sebastopol: O'Reilly, 2008.

KUUSKERU, J.; MIKKONEN, T. **Partitioning Web Applications Between the Server and the Client**. 2009 ACM symposium on Applied Computing. Honolulu, Hawaii: ACM. 2009. p. 647-649.

MIKKONEN, T.; TAIVALSAARI, A. **Web applications: spaghetti code for the 21st century**. Sun Microsystems, Inc. Mountain View, p. 1-19. 2007.

MURUGESAN, S.; DESHPANDE, Y. **Web Engineering: Managing Diversity and Complexity of Web Application Development**. Heidelberg: Springer-Verlag, v. LNCS 2016, 2001.

NETO, J. C. C. **The Economic and Social Importance of Digital Information Networks as a Mean for Development and Commerce**. Rio de Janeiro. 2001.

TAIVALSAARI, A. et al. **Web browser as an application platform: The Lively Kernel Experience**. Sun Microsystems, Inc. Technical Reports; Vol. SERIES13103. Menlo Park, p. 1-19. 2008. (SMLI TR-2008-175).

APÊNDICE A - ARTIGO

INTEGRAÇÃO ENTRE HTML5 E JSF 2.0 EM APLICAÇÕES WEB OFFLINE

ROBERTO JORGE HADDOCK LOBO FILHO¹

¹UFSC – Universidade Federal de Santa Catarina
INE – Departamento de Informática e Estatística
Florianópolis (SC), Brasil
rhlobo+ufsc@gmail.com

Resumo: A constante concepção de novas tecnologias e conceitos de desenvolvimento de software e de sua compatibilidade com as tecnologias já existentes é um dos grandes desafios na área de ciências da computação. Neste contexto, a utilização da WEB – que originalmente foi concebida com a intenção e capacidade de exibição de documentos hipertextos – como interface gráfica ao usuário de sistemas online têm gerado grandes dificuldades e limitações. Objetiva-se com este trabalho desenvolver uma pesquisa sobre as possibilidades de compatibilização entre duas tecnologias modernas relacionadas ao desenvolvimento destes sistemas, tendo como estudo a criação de aplicações web utilizando implementações provenientes da especificação JSF 2.0 – tecnologia padrão do JavaEE – que possam ser disponibilizadas em modo offline – sem que o usuário necessite estar conectado à internet - de acordo com as novas funcionalidades oferecidas pelo HTML 5, assumindo toda complexidade de comportamentos e interações que estas interfaces gráficas abrangem.

Palavras Chaves: Desenvolvimento WEB. JSF. Aplicações offline.

1 Introdução

A WEB (também conhecida como World Wide Web ou WWW) é um sistema de documentos em hipermídia que são interligados e executados na Internet. Segundo BERNERS-LEE (1989), o intento original do desenvolvimento da web foi tornar mais fácil o compartilhamento de documentos de pesquisas. Contudo, sua ampla adoção, junto ao crescimento da internet, gerou-se a demanda de sua utilização como plataforma para aplicações.

Os primeiros sistemas web de geração de conteúdo dinâmico geralmente objetivavam a simplificação da manutenibilidade e expansibilidade de grandes sistemas de informação. A evolução destes conceitos – e das tecnologias envolvidas – permite que aplicações web tenham cada vez mais capacidade de realizar as mesmas funcionalidades que aplicações desktop nativas.

Neste contexto, diversos frameworks foram criados para facilitar e possibilitar o desenvolvimento de aplicações web, cada um utilizando um conjunto de abordagens para simplificar as complexidades provenientes da adaptação da web para as novas demandas e novos requisitos funcionais. O conseqüente

amadurecimento do desenvolvimento de aplicações web possibilita a migração de serviços para a web, tornando, sob estas perspectivas, o conceito de instalação de software em uma máquina desktop obsoleto neste contexto.

A busca da continuidade da expansão das capacidades de aplicações web tem, porém, uma grande limitação em comparação a aplicações web. A utilização pura da web como plataforma para aplicações requer, de acordo com as próprias características da web, que o acesso à internet esteja disponível para que o usuário possa acessar uma aplicação. Usuários de aplicações web típicas são capazes de utilizar estas aplicações somente quando conectados à internet.

Há então a necessidade da pesquisa, análise e comparação da possibilidade de compatibilização entre tecnologias de desenvolvimento web e os conceitos existentes que visam à disponibilização de aplicações web offline, segundo os princípios básicos do funcionamento da web e dos possíveis impactos e requisitos ao usuário final.

Este trabalho tem como principal objetivo a pesquisa sobre a aplicabilidade e viabilidade da compatibilização entre o framework JavaServer Faces 2.0 0 – a última

versão do mais maduro framework de desenvolvimento de aplicações web contido na especificação Java EE 6 – devido à sua importância no desenvolvimento de aplicações web através da linguagem Java, e as funcionalidades propostas pelo HTML5 de armazenamento local de dados de execução e de usuário, que possibilitam a capacidade de execução – pelo menos parcial – enquanto offline de aplicações web, sem a necessidade de uma conexão com a internet.

2 Desenvolvimento de Aplicações Web

Desenvolvimento web é um termo amplo que faz referência a qualquer atividade de desenvolvimento que envolve a web, e pode abranger desde a criação de páginas estáticas simples de texto plano até o desenvolvimento de complexas aplicações e serviços baseados na web, ou seja, softwares que são apresentados em um ambiente controlado por um navegador web, ou codificados em linguagens e tecnologias suportadas por estes navegadores. A plataforma web ainda é, portanto, um conceito muito abstrato, principalmente devido à falta de uma base bem definida para desenvolvimento de software para a web. Mesmo assim existe uma alta demanda para viabilização e maturação de tecnologias para suporte a este processo.

2.1 Modelo cliente-servidor

O modelo cliente-servidor utilizado pela web descreve a relação entre dois programas de computador em que um programa - o cliente - faz uma solicitação de serviço de outro programa - o servidor - que responde o pedido. O cliente interage com um ou mais servidores através do protocolo HTTP (HyperText Transfer Protocol) realizando requisições de recursos e dados. Servidores web, que normalmente – mas não necessariamente - operam sobre uma rede em hardware separado do cliente, ao receberem uma requisição de recurso, ou devolvem um recurso estático ou geram a resposta dinamicamente. Sob a perspectiva do cliente, não existe diferenciação entre ambos os casos, e a resposta será tratada da mesma maneira, ou seja, será renderizada diretamente no cliente ou processada por algum script executado no navegador.

2.2 Particionamento de aplicações entre cliente e servidor

O desenvolvimento de aplicações web está no meio de uma mudança de paradigma. Usuários estão se acostumando com aplicações web com conteúdo dinâmico e experiência mais interativa do usuário. Interfaces gráficas não mais estão presas à necessidade de atualizar telas inteiras de uma vez, e servidores são

capazes de alimentar os dados a elas espontaneamente. No entanto, convenções bem estabelecidas para o particionamento da lógica da aplicação entre o cliente e o servidor ainda não existem, além da abordagem em que tudo é executado no servidor e em que o cliente simplesmente exibe os dados.

O particionamento da lógica de negócio entre cliente e servidor, ou seja, a definição da porção de código que ficará em cada uma das distintas camadas de uma aplicação, influencia diretamente a dependência entre o grau de interatividade de uma interface gráfica web e o modo em que a comunicação das duas partes deve ser feita. As decisões de particionamento de código, portanto, definem a dependência do cliente para com o servidor, o que significa que conseqüentemente definem também as possibilidades de que uma aplicação possa ser executada em modo offline - sem conexão com o servidor.

Neste cenário, o ideal seria que o cliente independência possível do cliente, possuindo o máximo controle possível sobre a execução da lógica de uma aplicação. Tal abordagem não só torna o desenvolvimento muito mais simples, mas também aumenta a capacidade de resposta e robustez da interface do usuário, pois resultaria em um número reduzido de requisições necessárias ao servidor - e o conseqüente tempo de latência relacionado a cada uma delas - garantindo assim maior independência e fluência da interface de usuário.

A utilização de uma única linguagem comum para toda a aplicação, com a aplicação transparente de um gateway para comunicação entre o servidor e o cliente, poderia facilitar o processo de desenvolvimento, aproximando-o ao desenvolvimento de aplicações tradicionais. Além disto, esta estratégia também permitiria fácil balanceamento de código entre o cliente e servidor - sem a necessidade de grande retrabalho - ampliando a flexibilidade da aplicação e a sua manutenibilidade.

2.3 Responsabilidades inerentes ao servidor

Para muitas aplicações, seus requisitos às prendem a algumas responsabilidades inerentes do servidor:

- Persistência - Refere-se aos dados que não são apagados no encerramento da execução do programa que o criou. Embora em navegadores modernos seguindo a especificação HTML5 seja possível o salvamento de dados localmente, muitas vezes existe também a necessidade de que estes dados estejam disponíveis também para outros computadores na rede.

- Comunicação cliente-cliente - Ocorre entre uma ou mais aplicações web rodado em diferentes clientes.
- Dados compartilhados - Significa que dados necessitam estar acessíveis a mais de um cliente simultaneamente. O servidor é utilizado para prover uma base de dados centralizada e compartilhada entre diferentes clientes.
- Tarefas agendadas - Pode-se existir a necessidade da execução de tarefas e processamentos durante o período em que não exista garantia que o cliente esteja sendo executado.
- Segurança - devido ao fato de que o código executado no cliente pode ser acessado e modificado, existem certas operações cuja a execução possa ser obrigatória no servidor. Um exemplo inclui a necessidade de validação de dados fornecidos pelo usuário, que pode ser facilmente burlada por um usuário mais experiente.
- Confiabilidade - pode ser aumentada através da execução de operações no servidor. Por exemplo, usuários podem fechar seus navegadores da web durante uma operação, ou no caso de operações dependentes de horário, comportamentos inválidos podem ocorrer devido à má configuração do relógio do cliente.

2.4 Considerações para o servidor

De acordo com a natureza da aplicação, a execução de algumas tarefas e processamentos no servidor pode ser desejável, vantajosa ou até necessária. Enquanto as tendências da evolução da web em relação à execução de aplicativos interativos e dinâmicos dá preferência a (é coerente com) um modelo no qual o cliente possa lidar, na medida do possível, com os eventos gerados na interface gráfica, o contexto de regras de negócio da aplicação deve ser sempre levado em consideração.

- Navegação - o modo na qual a navegação de documentos na web evoluiu permite que o usuário utilize botões para avançar e retroceder entre o conjunto de páginas acessadas. Usuários estão acostumados a aplicações web simples nas quais se pode navegar com segurança mesmo quando utilizadas estas funções. Esta característica é perdida quando uma aplicação inteira é implementada como uma única página dinâmica. Soluções para este problema envolvem uma abordagem híbrida em que diferentes URLs identifiquem diferentes partes de uma aplicação ao mesmo tempo em que estas transições de estado possam ser realizadas de forma dinâmica e parcial, ou seja, através da adaptação do conteúdo entre diferentes estados sem a necessidade de um recarregamento total da aplicação.
- Cálculos pesados - ou outras operações de uso intensivo da CPU podem ser, em alguns casos, melhor executados no servidor devido ao fato deste poder ser um ambiente um pouco mais controlado onde se possui conhecimento do hardware utilizado e sua capacidade. A execução de código no servidor também possibilita o uso de linguagens mais poderosas para certos domínios, como por exemplo o de cálculos lógicos ou matemáticos.

2.5 Considerações para o cliente

A natureza da plataforma web, de acordo com a sua evolução, se deu de tal forma na qual os clientes mais simples de uma aplicação são basicamente constituídos da exibição de uma interface gráfica simples e estática, praticamente sem a execução de nenhuma lógica. A criação de clientes mais complexos vem sendo um desafio que aos poucos é possibilitado pelas abstrações tecnológicas fornecidas por frameworks e ferramentas de desenvolvimento, para facilitar a criação de interfaces dinâmicas, que proporcionem uma maior interatividade com o usuário e maior poder em termos de capacidades que uma aplicação web pode apresentar.

A disponibilização de dinamicidade e de funcionalidades no software cliente requer ou que o mesmo possua lógica em termos de ambas regras de negócio e integração entre os componentes da visualização de acordo com o domínio da aplicação ou que uma comunicação extremamente ativa com o servidor seja estabelecida para que o servidor possa determinar os detalhes de como o cliente deve responder a cada evento. Em outros termos, para a criação de aplicações web, é necessário – proporcionalmente a sua complexidade - ou trazer parte da sua lógica ao cliente ou aumentar a dependência da comunicação de um cliente simples e “burro” com o servidor. Esta dualidade estabelece claramente duas diferentes – e quase opostas – abordagens, uma propondo a utilização de um cliente fino e simples administrado pelo servidor e outra propondo a criação de um cliente mais especializado contendo parte da lógica da aplicação.

2.6 Aplicações offline

A criação de mecanismos para o desenvolvimento de aplicativos web offline tem como intuito mitigar a limitação - a qual muitas aplicações desktop não

possuem – imposta pela necessidade de uma conexão à internet. Os requisitos mínimos necessários para que uma aplicação web possa ser disponibilizada offline podem ser sintetizados e classificados em duas categorias básicas: a armazenagem para futuro acesso de recursos da aplicação – que implica em um mecanismo de armazenamento local de código da aplicação - e a armazenagem de dados do usuário, relativos ao uso da aplicação.

3 HTML5

O HTML foi concebido essencialmente como uma linguagem para descrever semanticamente documentos científicos, embora a sua concepção geral e adaptações ao longo dos anos têm-lhe permitido ser usado para descrever uma série de outros tipos de documentos, incluindo sendo utilizado para a exibição de interfaces gráficas de sistemas. Até a versão 4 do HTML, sua especificação não visava praticamente nenhum suporte a aplicações para a web.

A criação do HTML 5 tem sua motivação na falta de uma base mais sólida na web no qual softwares possam se estabelecer. Sua especificação visa mitigar a falta de padronização entre diferentes implementações, aumentar o grau de fidelidade entre os resultados desejados e obtidos na exibição e comportamento de componentes de software, e por fim, visa também fornecer os recursos e funcionalidades básicas necessárias para que aplicativos web possam cada vez mais ter as mesmas capacidades de aplicações nativas.

O HTML5 contempla, dentre outras coisas, diversas funcionalidades e APIs que se propõem a comportar os requisitos mínimos para a disponibilização de aplicações web em um contexto offline. Sua especificação estabelece um mecanismo de cacheamento de recursos declarativo, dois mecanismos para o armazenamento de dados estruturados no cliente, e um conjunto de APIs para a manipulação e armazenamento no cliente de bases de dados relacionais através do uso de SQL.

4 JavaServer Faces 2.0

O JSF, o principal framework de interface gráfica ao usuário para aplicações Java web na especificação do JavaEE 6, é uma framework MVC orientada a requisições, fundamentado no modelo de design de interfaces gráficas baseadas em componentes, através da utilização de arquivos XML chamados de *view templates* (templates de visualização) ou *Facelet views*.

O framework realiza a facilitação da criação de interfaces gráficas web de aplicações escritas em Java, provendo, dentre uma gama de funcionalidades, o

gerenciamento de estado de componentes de UI, a manipulação de navegação página-a-página em resposta a eventos provindos do cliente e do controle de validações e aplicação de valores. Ele é responsável por gerar o conteúdo para o cliente (geralmente em HTML) e realiza o processamento das subseqüentes requisições.

A arquitetura do JSF estabelece um modelo em que quase a totalidade dos processamentos da aplicação são executados no servidor, não somente incluindo as funções de gerenciamento necessárias, mas incentivando que a lógica de negócio da aplicação não seja levados ao cliente. Esta abordagem promove o cliente como uma camada fina e delega ao servidor toda a gerência das responsabilidades e funcionalidades do framework. Requisições são processadas pelo *FacesServlet* que, em suma, carrega o template e árvore de componentes de visualização apropriado, aplica os novos valores fornecendo a oportunidade de atualização do estado de cada componente, realiza as devidas validações, processa eventos, atualiza os valores no modelo, invocando em seguida a aplicação para que se possa realizar os devidos processamentos, e então renderiza a resposta apropriada para o cliente. O cliente, portanto, funciona como uma simples interface gráfica, e é estritamente dependente do servidor.

5 Integração do JSF 2.0 e o HTML5 Offline

A compatibilização do JSF 2.0 para criação de aplicações web offline oferece uma série de grandes desafios, e deve ser analisada em termos da viabilidade das possíveis soluções, seus requisitos e os impactos ao seu uso e usuário final.

A arquitetura determinada pela especificação do JSF 2.0 estabelece um modelo em que o servidor assume toda a responsabilidade sobre o controle e sobre a lógica de negócio da aplicação. É estabelecida, assim, uma estrutura rígida que força que todas as funções de gerenciamento de uma aplicação, desde a gerência do estado da interface gráfica e do modelo até o processamento e manipulação de valores, sua validação, e a geração de uma resposta, sejam executadas no lado do servidor, através da utilização da linguagem Java. Ao mesmo tempo, não existe nenhuma maneira padronizada entre os diferentes navegadores web que permita – ou mesmo vislumbre - a execução de código Java relacionado às especificações existentes do JSF para a supervisão da aplicação quando offline.

É, portanto, necessário não só determinar a porção de código que deve ser trazido ao cliente, mas também maneiras para que se possa executá-lo no ambiente dos navegadores web modernos, e como sincronizar os

processamentos entre o cliente o servidor, garantindo a integridade dos dados dos mesmos.

5.1 Código da aplicação necessário para execução offline no cliente

Os requisitos mínimos para disponibilização offline de uma aplicação web, abordados anteriormente neste trabalho, estipulam a necessidade do armazenamento local dos recursos da aplicação, que incluem o código e recursos da interface gráfica, código base de gerenciamento da aplicação, código contendo lógica de negócio e os dados relativos ao domínio.

5.2 Particionamento de lógica entre aplicação cliente e o servidor

A determinação da porção de código de lógica de negócio de uma aplicação necessário - e cabível - ao cliente de uma aplicação web JSF é um problema de grande complexidade, pois a concepção e modelagem das mesmas não foram idealizadas com este propósito. Neste contexto, é necessário elencar tanto os processamentos cuja execução seria imprescindível em clientes offline quanto as porções de código que devem residir somente no servidor. Assim sendo, temos:

- O próprio framework JSF como um todo é absolutamente necessário em um contexto offline e necessita ser levado ao cliente. Suas responsabilidades englobam quase que em sua totalidade a geração da interface gráfica e seu gerenciamento, bem como fornece as portas de entrada ao resto do sistema. Como todas as fases do seu ciclo de vida de processamento e tratamento de requisições realizam papéis importantes de acordo com este contexto, o mesmo não poderia ser dividido.
- Os pontos de entrada de invocação a aplicação, normalmente referidos como *Managed Beans*, também se fazem necessários em um cliente offline. Sua principal responsabilidade resume-se em intermediar a comunicação entre as páginas da interface gráfica web (componentes do JSF) e o modelo da aplicação, fornecendo uma fachada de acesso a dados, escutando eventos, validando entradas e delegando ações e informações à camada de negócios, assim também representando função essencial em um cliente offline.
- A camada de negócio reúne – por concepção - a maior parte da lógica de negócio e, segundo os padrões propostos e enforçados pelo JavaEE, deve ser implementada de forma totalmente

independente da camada de visualização, na forma de serviços em EJBs. Devido a isto, esta camada não deve, e não pode, ser incluída em código levado ao cliente. Esta camada pode também conter lógica, dependências, referência de acesso a outros serviços online, e outros processamentos de execução mandatórios no servidor.

Estabelece-se assim, que a porção de código de lógica que deve ser trazida ao cliente pode ser delimitada pela separação entre a camada de visualização e a camada de lógica de negócios de uma aplicação JSF. O particionamento de lógica faz então necessário um mecanismo - criado de acordo com a natureza da aplicação - que possibilite a separação de lógica de negócio, aqui referido como Server-mock, representando a implementação disponível somente no servidor. Este mecanismo é responsável por, dentre outras coisas, determinar as ações a serem tomadas desde quando operações e dados não estiverem disponíveis, estabelecendo procedimentos ao implementar fachadas de serviços EJB acessíveis ao código portado, ou seja, fornecendo uma implementação alternativa as interfaces da lógica indisponível no cliente.

5.3 Dados da aplicação

O particionamento de código entre servidor e cliente implica na carência de processamentos e dados - normalmente disponibilizados pelo conjunto de serviços do servidor - não acessíveis pelo cliente offline, que podem comumente representar fundamental importância para o funcionamento da aplicação. Assim, a implementação alternativa dos serviços indisponíveis, ou seja, o Server-mock, deve ser responsável pelo acesso de - ao menos - todo o conjunto de dados fundamentais, e para que tal função seja possível, os mesmos devem estar armazenados e disponíveis de alguma maneira, em contexto offline.

Do mesmo modo, todo o conjunto de dados gerados pelo cliente offline, bem como informações de representação de todos os processamentos requisitados, deve ser armazenado, para que possam ser posteriormente atualizados e executados no servidor, de modo a manter a pertinência e integridade do estado da aplicação como um todo. Ambas a disponibilização dos dados da aplicação em ambientes offline e a atualização do servidor segundo qualquer mudança depende da existência de um mecanismo responsável pela sincronização bilateral de dados entre cliente e servidor, assim como dependem de meios para o armazenamento e acesso a dados persistentes no cliente.

O HTML5, segundo sua especificação, fornece dois mecanismos distintos primários para o armazenamento de dados de uma aplicação, disponibilizando seu acesso tanto quando online quanto quando offline. Ambos os mecanismos são acessíveis diretamente pelo navegador e através do uso de JavaScript, provendo acesso a um conjunto de APIs para a manipulação de bases de dados relacionais no cliente e para o armazenamento - e acesso - persistente de dados baseados em pares de chave-valor. Em conjunto, as duas tecnologias viabilizam o armazenamento - para acesso futuro - de todos os tipos de dados da aplicação que podem vir a ser necessários neste cenário, incluindo dados salvos durante acesso online, para acesso posterior offline, e dados salvos durante a utilização offline do software, que necessitem ser posteriormente disponibilizados para o servidor.

5.4 Execução do código Java da aplicação no cliente

Existem algumas alternativas não triviais que poderiam viabilizar a execução do código Java necessário em navegadores web, dentre elas podemos ressaltar:

- Java Applets para disponibilização de código Java no cliente junto à criação de Render Kits do JSF especializados na criação de páginas web offline que utilizem o Applet ao invés do servidor;
- Criação de plug-ins para navegadores web que executem o código Java necessário quando offline, interceptando chamadas feitas por páginas da aplicação; Necessitaria a criação de plug-ins específicos para cada navegador, para cada aplicação.

Ambas as soluções poderiam viabilizar o problema atacado por este estudo, contudo, ambas possuem algumas considerações importantes:

- Teriam relativamente grande tamanho, considerando os padrões de velocidade de transferência de arquivos na internet em relação ao tamanho das bibliotecas que seriam necessárias, como o JSF, que possui - de acordo com suas diferentes implementações - entre seis e nove Megabytes, se levando em conta também a necessidade de download, pelo sistema do cliente, a cada nova versão, para armazenamento em seu computador.
- Possuem como pré-requisito e condição básica a dependência de uma JVM (Java Virtual Machine) compatível na máquina cliente.

5.5 Sincronização entre cliente e servidor

Com a concepção de uma aplicação web com capacidade de funcionamento offline faz-se necessário um mecanismo que realize todos os métodos de sincronização de dados entre cliente e servidor, sendo responsável pela manutenção da integridade do estado da aplicação. No âmbito da compatibilização de aplicações JSF 2.0 a complexidade deste mecanismo é amplamente agravado.

O alto acoplamento de camadas promovido pelo JavaEE e a arquitetura de software ditada pelas convenções do JSF estabelecem grande dependência da visualização aos serviços da camada de lógica da aplicação para o acesso a dados e realização de processamentos.

Em um ambiente offline, para que exista a possibilidade de atualização posterior do estado do servidor, faz-se necessário o armazenamento e sincronização de toda e qualquer operação realizada na visualização durante uma sessão. Para garantir a consistência do estado de uma aplicação durante a consumação destes processos, determinam-se também dependências relacionadas à ordem de execução de processamentos e da congruência dos dados resultantes entre serviços no cliente e no servidor.

Neste contexto, qualquer lógica que seja levada ao cliente agrava as dificuldades de sincronização. A execução de qualquer lógica particionada entre o cliente e o servidor tem como consequência a geração de dados parcialmente processados, cujo controle de versão e conflitos em relação aos dados do servidor pode ser até impossível, devido à diferente natureza dos dados.

Como consequência, temos que uma grande complexidade surge devido às possibilidades de divergência entre a realização de operações em um ambiente offline e no servidor, que podem ser ocasionados por uma ampla diversidade de situações comuns. Como exemplo, pode-se citar como causa de conflitos a comum possibilidade de que dados mantidos em um cache não estejam atualizados e a não disponibilização, simulação ou até previsão de resultados de serviços e processamentos no cliente. Esta problemática é tão crítica que, de acordo com os requisitos estabelecidos, a simples ocorrência de conflito, erro ou discrepância de resultados de processamentos durante a execução de tarefas de sincronização, causaria a perda da sessão offline ou, se não tratado, na inconsistência do modelo de dados da aplicação.

6 Conclusões

Através do estudo das possibilidades de compatibilização de aplicações web desenvolvidas utilizando-se o framework JavaServer Faces 2.0 com as facilidades de disponibilização de aplicações offline oferecidas pelo HTML5, junto à implementação de diversas provas de conceito, conclui-se que a efetivação dos fins contemplados pelo estudo realizado por este trabalho é possível, porém, somente apresentando resultados extremamente limitados e demandando enorme esforço, não gerando, portanto, grande valor, ou seja, que a integração das tecnologias é tecnicamente factível porém inviável em sua prática.

A cultura de utilização do JSF promove alto acoplamento entre as camadas de visualização e de serviços, restringindo não somente o número de processamentos disponíveis, mas também sua possível complexidade, tornando impraticável o desenvolvimento de aplicações JSF sofisticadas para ambientes offline. Além disto, a integração das tecnologias propostas implica também em um conjunto de limitações relacionado à necessidade de adaptação do próprio framework JSF 2, da especificação e criação de uma série de complexos mecanismos específicos à natureza da aplicação, como de suporte às funcionalidades fornecidas pelo HTML5 e da enorme complexidade para sincronização que seriam resultantes da implementação de qualquer lógica não trivial no cliente da aplicação.

Para que as aplicações resultantes pudessem contemplar qualquer tipo de processamento offline contendo lógica de negócios, por mais simples que os mesmos fossem, se faria necessária a quebra dos padrões estabelecidos pelo JEE para desenvolvimento de aplicações web.

Assim, a criação de uma aplicação JSF offline poderia prover poucas funcionalidades e uma pobre experiência ao usuário em relação ao alto custo para seu desenvolvimento, o que faz com que a sua própria concepção deva ser questionada: a da criação de aplicações web offline pode não fazer sentido quando os próprios serviços que justificariam a sua criação seriam limitados a pobres implementações.

De maneira geral, o JSF é uma excelente tecnologia para Web 1.0, fim para o qual foi projetada. As grandes vantagens da utilização desta tecnologia, porém, desaparecem quando se está em ambiente cliente, pois mesmo que tecnicamente possível trazer esta tecnologia originalmente servidor, seu esforço é demasiado grande em comparação aos seus potenciais resultados, visto que existem tecnologias muito mais maduras para entregar aplicações Web 2.0.

7 Referências

- BURNS, E.; KITAIN, R. JavaServer Faces Specification. Sun Microsystems Inc. Santa Clara. 2009.
- CLIENT-SIDE Scripting and HTML. W3C, 1997. Disponível em: <<http://www.w3.org/TR/WD-script-970314>>. Acesso em: 05 jan. 2010.
- GINIGE, A.; MURUGESAN, S. Web Engineering: A Methodology for Developing Scalable, Maintainable Web Applications. Cutter IT Journal, Julho 2001. 24-35.
- HICKSON, I. HTML5. W3C, 2010. Disponível em: <<http://dev.w3.org/html5/spec/>>. Acesso em: 20 Fevereiro 2010.
- HOLDENER, A. Ajax: The Definitive Guide. 1 Edição. ed. Sebastopol: O'Reilly, 2008.
- KUUSKERU, J.; MIKKONEN, T. Partitioning Web Applications Between the Server and the Client. 2009 ACM symposium on Applied Computing. Honolulu, Hawaii: ACM. 2009. p. 647-649.
- MIKKONEN, T.; TAIVALSAARI, A. Web applications: spaghetti code for the 21st century. Sun Microsystems, Inc. Mountain View, p. 1-19. 2007.
- MURUGESAN, S.; DESHPANDE, Y. Web Engineering: Managing Diversity and Complexity of Web Application Development. Heidelberg: Springer-Verlag, v. LNCS 2016, 2001.
- TAIVALSAARI, A. et al. Web browser as an application platform: The Lively Kernel Experience. Sun Microsystems, Inc. Technical Reports; Vol. SERIES13103. Menlo Park, p. 1-19. 2008. (SMLI TR-2008-175).