UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM WEB E SISTEMAS DE INFORMAÇÃO

NELSON ROGÉRIO VICTORAZZI

RIA - RICH INTERNET APPLICATIONS

Trabalho de Conclusão apresentado como requisito parcial para a obtenção do grau de Especialista

Prof. Dr. Carlos Alberto Heuser
Orientador

Prof. Dr. Carlos Alberto Heuser
Coordenador do Curso

Porto Alegre, dezembro de 2007.

AGRADECIMENTOS

Gostaria de agradecer aos professores e aos funcionários do Instituto de Informática da Universidade Federal do Rio grande do Sul pelo carinho e dedicação dispensados aos alunos do Curso de Especialização em WEB e Sistemas de Informação período 2006/2007.

Também gostaria de fazer um reconhecimento pessoal ao alto nível dos professores do curso de Especialização em WEB e Sistemas de Informação período 2006/2007. O Brasil deve se orgulhar por ter pessoas tão capacitadas como os professores acima mencionados.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

WEB	World Wide Web.
HTML	Hyper Text Markup Language.
FTP	File Transfer Protocol.
HTTP	Hyper Text Transfer Protocol.
ICH	Interação Humano-Computador
CIC	Cadastro de Inscrição de Contribuinte.
UFRGS	Universidade Federal do Rio Grande do Sul

LISTA DE FIGURAS

RESUMO

As aplicações desenvolvidas para funcionarem na Internet já podem obedecer aos princípios necessários para a boa interação humano-computador. Estas aplicações têm hoje um conjunto de ferramentas e tecnologias que podem proporcionar os meios para a construção de aplicações com uma interface gráfica que leve em conta as necessidades do usuário, recurso que até agora só era encontrado nas ferramentas para desenvolvimento para aplicações *desktop*.

Neste trabalho foram relacionadas quatro tecnologias de fabricantes distintos que oferecem um conjunto de recursos para a construção de aplicações com interface rica. Open Laszlo, Java WEB Start , Abobe Flex 2 e Microsoft Silverlight são as tecnologias que este trabalho identificou suas características, linguagens e arquitetura .

Também foi utilizado no estudo de caso uma aplicação modelo desenvolvida pela Adobe onde são mostrados alguns recursos gráficos que enriquecem a experiência do usuário no uso da aplicação.

Palavras-Chave: Interação humano-computador usabilidade aplicações interfaces rica.

1 INTRODUÇÃO

Invenções humanas, não raramente, tem suas funcionalidades ampliadas ou modificas pelas pessoas que fazem uso delas. Um exemplo simples é o correio eletrônico. Ele foi desenvolvido para trocar mensagens entre pessoas, no entanto é muito comum o seu uso como forma de enviar e receber arquivos. O meio ideal para enviar ou receber arquivos pela rede é o FTP, no entanto a troca de arquivos, recebimento de mensagens contendo arquivos anexos é muito comum.

Outro exemplo é a WEB. Tim Berners Lee que em 1989 trabalhava no CERN desenvolveu um sistema baseado em hipertextos para poder localizar e compartilhar documentos do centro de pesquisa entre seus colegas. O CERN é um centro e de pesquisa com muitos pesquisadores organizados hierarquicamente, mas que compartilhar recursos e informações. Lee desenvolveu o HTTP e o HTML. No inicio utilizado para localizar e compartilhar arquivos localizados nos computadores do centro de pesquisa, hoje em dia sofreu uma grande transformação quanto ao seu uso.

Comércio eletrônico é uma das funcionalidades agregadas a WEB não prevista na idéia inicial do projeto de Lee. Pessoas em seus computadores pessoais acessam servidores das empresas, recebem informações sobre produtos, efetuam transações com cartões de credito e recém em casa os produtos escolhidos.

Transações financeiras, pagamentos de títulos, contratação de empréstimos, investimentos e transferência de dinheiro são algumas das funcionalidades oferecidas por bancos aos seus clientes evitando assim a presença física nas agencias.

Na educação, professores compartilham informações, distribuem materiais, aplicam testes, recebem trabalhos dos alunos permitindo aos alunos cursarem seus cursos a distancia.

Atualmente até aplicações como processadores de texto e planilhas eletrônicas que eram vendidas e instaladas nos computadores dos usuários estão disponíveis na WEB mudando completamente a maneira como as pessoas utilizam seus computadores e aplicativos.

E as mudanças parecem não ter fim. É grandes o número de novas aplicações, com funcionalidades não previstas no projeto de Lee, que estão sendo desenvolvidas e oferecidas a cada dia. Musicas, vídeos, emissoras de rádios e televisão agencias de noticias estão fortemente presentes na WEB.

Grandes empresas, com milhares de computadores ligados em rede vislumbram a troca de suas aplicações que rodam na rede interna por aplicativos que utilizam a WEB. Enxergam a possibilidade de redução de custos na utilização de aplicações rodando na WEB. A migração para aplicativos na WEB mantendo os requisitos de usabilidade das

aplicações cliente-servidor é uma exigência e um desafio para estes os novos aplicativos. Interfaces amigáveis, segurança e eficiência são alguns requisitos destes novos aplicativos.

Segundo ROCHA "dados disponíveis apontam que em 1998 cerca de três bilhões de dólares deixaram de ser ganhos na WEB norte-americana por causa de design mal feito de páginas, que dificultava a compra em vez de facilitar. Essa estimativa dá conta de um debate que ganha cada vez mais espaço: como equilibrar o uso de recursos visuais capazes de atrair a atenção do usuário e ao mesmo tempo tornar os *site*s fáceis de entender e usar? A questão pode ser parafraseada: Como fazer uso da tecnologia disponível e ao mesmo tempo aumentar a usabilidade de *sites* da WEB?".

Ainda segundo ROCHA "Um exemplo muito citado é o da IBM dos Estados Unidos. A empresa constatou que o recurso mais popular em seu *site* era a função de busca, porque as pessoas não conseguiam descobrir como navegar, e o segundo mais popular era o botão de ajuda. A solução foi um amplo processo de redesign, envolvendo centenas de pessoas e milhões de dólares. Resultado: na primeira semana depois do redesign, em fevereiro de 1999, o uso do botão de ajuda caiu 84% enquanto as vendas aumentaram 400%".

A usabilidade dos sistemas atuais ganhou atenção devido as grandes cifras envolvidas. As empresas e profissionais que oferecerem ferramentas e sistemas que levem em conta as necessidades e características dos novos usuários da internet certamente colherão melhores resultados. Os desafios serão maiores que aqueles previstos inicialmente por Lee, mas os resultados poderão ser compensadores.

Este trabalho foi organizado como segue:

Na seção dois, Interação Humano-Computador, foram abordados os conceitos e princípios que norteiam a construção de aplicações e sistemas de informação que levem em conta as características do usuário da aplicação. Estes conceitos e princípios foram estendidos para a construção de sistemas para a WEB. Com base nestes princípios buscou-se ferramentas e tecnologias que pudessem ser usadas para a construção de aplicações para WEB.

Na seção três, Modelos para a construção de Aplicações , foram pesquisados quatro tecnologias que podem ser utilizadas para a construção de aplicações para WEB e que podem oferecer ao programador os recursos necessários para obedecer os requisitos da interação humano-computador de forma satisfatória. OpenLaszlo, Java Web Start, Adobe Flex 2 e Micrososft Silverlight são os quarto modelos abordados.

Na secção quatro, Estudo de Caso, foi utilizada uma aplicação modelo, desenvolvida pela Adobe utilizando o Adobe Flex 2 Builder, que demonstra alguns dos requisitos requeridos no estudo da Interação Humano-computador.

2 INTERAÇÃO/INTERFACE HUMANO-COMPUTADOR

O estudo referente à interação humano-computador (IHC) data do inicio dos anos oitenta e pode ser definido como "a disciplina relativa ao design, avaliação e implementação de sistemas computacionais interativos para uso humano e aos fenômenos que os cercam" (ROCHA). É uma área recente e abrangente de estudo das relações humano-computador da qual serão usados alguns aspectos referentes às características do desenvolvimento de interfaces humano-computador.

Segundo ROCHA interação humano-computador pode ser definida como:

> Primariamente, como já dissemos, se visualiza uma interface como um lugar onde o contato entre duas entidades ocorre (por exemplo, a tela de um computador). O mundo está repleto de exemplos de interfaces: a maçaneta de uma porta, uma torneira, a direção de um carro, etc. A forma das interfaces reflete as qualidades físicas das partes na interação. A maçaneta de uma porta é projetada para se adequar à natureza da mão que irá usá-la, o mesmo acontece com o câmbio de um carro (observe que a localização do câmbio dentro do carro sugere o uso por uma pessoa destra). Existem tesouras de dois tipos uma para pessoas destras e outra para pessoas canhotas. O que muitas vezes é esquecido é que a forma da interface também reflete o que pode ser feito com ela. Tomando o exemplo da maçaneta, podemos ver que no mundo existem diversos formatos de maçaneta e de acordo com o formato sabemos como deve ser aberta uma porta: girando a maçaneta no sentido anti-horário, empurrando a porta, puxando a porta, etc. (Norman, 1988). O mesmo acontece com a forma das torneiras onde se deve girar ou empurrar ou levantar uma alavanca, etc. Nos exemplos anteriores da porta e da torneira que foram feitas para serem abertas por um humano podemos dizer que o humano é o agente e a porta (ou torneira) são os pacientes dessa ação. Mas, temos também as portas, ou torneiras, que abrem automaticamente quando identificam através de um sensor ou uma câmera a presença de alguém (mesmo que esse alguém não queira abrir a porta). Nesse caso o sentimento que temos de quem está controlando a interação é bastante diferente. Em muitos banheiros públicos existem instalados aqueles secadores automáticos de ar quente para mãos e muitas vezes, mesmo não querendo usá-los eles se ligam porque nos encostamos próximos a eles ou sem querer passamos a mão perto do sensor. E as torneiras que sempre se fecham antes de acabarmos de lavar as mãos? Nesses casos, não é mais o humano que está no controle da interação. Portanto, podemos ter como uma definição de base, que uma interface é uma superfície de contato que reflete as propriedades físicas das partes que interagem, as funções a serem executadas e o balanço entre poder e controle (ROCHA, 2000).

Uma abordagem completa sobre IHC envolve múltiplas disciplinas, tais como, segurança, eficiência e produtividade, aspectos sociais e organizacionais (ROCHA). O que intereça neste trabalho é somente os princípios de design de interfaces adotados por ROCHA.

.

- **Visibilidade e *Affordances***

Visibilidade indica o mapeamento entre ações pretendidas e as ações reais. Indica também distinções importantes. A visibilidade do efeito das operações indica se a operação foi feita como pretendida (ROCHA).

No HTML as formas comuns de dar visibilidade é com links de textos e imagens e botões. Se o projetista capturou bem o contexto do usuário é provável que os links serão entendidos pelo usuário e a ação será uma chamada a uma nova pagina HTML ou um script que ficara a cargo do browser executar. Esta operação já não é tão visível ao usuário, pois depende do browser que esta executando, podendo ocorrer um erro inesperado em função do browser não suportar o script.

Affordance é o termo definido para se referir às propriedades percebidas e propriedades reais de um objeto, que deveriam determinar como ele pode ser usado. Uma cadeira é para sentar e também pode ser carregada. Vidro é para dar transparência, e aparenta fragilidade. Madeira dá solidez, opacidade, suporte, e possibilidade de escavar (ROCHA). Link e botões do HTML correspondem muito bem as propriedades percebidas pelos usuários. Ainda é possível o auxilio de textos de ajuda explicando o seu funcionamento.

- **Bom modelo conceitual**

Um bom modelo conceitual permite prever o efeito de ações. Sem um bom modelo conceitual opera-se sob comando, cegamente. Efetua-se as operações receitadas, sem saber que efeitos esperar ou, o que fazer se a coisa não der certa. Conforme as coisas vão dando certo, aprende-se a operar. Agora, quando as coisas dão errado ou quando se depara com situações novas necessita-se de um maior entendimento, de um bom modelo (ROCHA).

No mundo dos sistemas computacionais o modelo conceitual pode fazer com que o usuário aprenda a utilizar o sistema ou simplesmente desista de utilizar o sistema. Se o modelo conceitual estiver fora do contexto do usuário será muito difícil que o usuário perca tempo tentando entender como sistema funciona.

- **Bons mapeamentos**

"Mapeamento é o termo técnico para denotar o relacionamento entre duas entidades. No caso de interfaces, indica o relacionamento entre os controles e seus movimentos e os resultados no mundo" (ROCHA).

Neste principio o HTML pode complicar muito a vida do usuário. Como o mapeamento implica no relacionamento entre o controle e seus resultados esperados e como o feedback não é imediato pode deixar o usuário perdido na ação que tentou realizar. Suponha o uso de controle que necessite do número do CIC do usuário e não é informado o formato que deve ser informado. O usuário poderá tentar varias vezes até adivinhar o formato esperado.

- **Feedback**

"Retornar ao usuário informação sobre as ações que foram feitas, quais os resultados obtidos, é um conceito conhecido da teoria da informação e controle" (ROCHA).

Nos sistemas para WEB o *feedbak* de link, botões ou submissões de formulários é sempre uma chamada a uma nova pagina que pode ser demorada e em situações que o usuário faz um número grande de repetições pode ser enfadonha. Ainda pode causar ao usuário aflição porque a resposta demora algum tempo e neste intervalo o usuário não sabe o resultado da operação.

São de interesse deste trabalho apenas os princípios (aqui chamados de *slogans)* direcionados para sistemas computacionais que levem a um aumento de usabilidade criada por ROCHA. A seguir são listados alguns deles:

- **Sua melhor tentativa não é boa o suficiente**

"É impossível fazer o design de uma interface ótima simplesmente baseado em nossas melhores idéias. Usuários têm um potencial infinito para mal interpretar elementos de interface e para fazer suas tarefas de modo diferente do que imaginamos. Portanto, o design é sempre melhor se trabalhamos baseados no entendimento do usuário e de suas tarefas" (ROCHA).

- **Usuário está sempre certo**

"A atitude do designer quando verifica que o usuário tem problemas de interação com um determinado aspecto da interface, não deve ser a de julgar que o usuário é ignorante ou então, que ele não tentou o suficiente ou ainda, deixar passar que um dia o usuário aprende. O que deve ser feito é aceitar que o texto está mal escrito e que precisa ser reformulado. Portanto, o designer de interfaces deve adquirir uma certa humildade e aceitar a necessidade de modificar uma" grande idéia "de forma a resolver problemas dos usuários" (ROCHA).

- **Usuário não está sempre certo**

"Também não se deve ir ao extremo de construir uma interface somente a partir do que os usuários gostariam. Usuários freqüentemente não sabem o que é bom para eles. Qualquer um de nós teria dificuldades em prever como gostaríamos de interagir com um sistema em potencial com o qual não temos nenhuma experiência. Temos a tendência de rejeitar *a priori* qualquer grande inovação em objetos com os quais estamos familiarizados e que atendem satisfatoriamente nossas necessidades" (ROCHA).

- **Usuários não são designers**

"Uma solução simples para atender a diversidade de usuários seria a de prover interfaces flexíveis que pudessem ser amplamente customizadas e aí cada usuário teria exatamente a interface que melhor lhe satisfizesse. Estudos demonstram que usuários novatos não customizam suas interfaces. Mas existem alguns outros bons motivos para não se dar a customização uma importância indevida: primeiro, customização é fácil somente se puder produzir um design coerente a partir do conjunto de opções

disponíveis; segundo, o processo de customização também vai exigir uma interface e portanto adiciona complexidade; terceiro, muita customização leva a que cada usuário tenha uma interface muito diferente de outro usuário e quarto, usuários nem sempre adotam as decisões de design mais apropriadas" (ROCHA).

- **Designers não são usuários**

"Designers são humanos e certamente usam computadores, mas são diferentes de usuários em diversos aspectos básicos: a experiência computacional e o conhecimento dos fundamentos conceituais do design do sistema. Conseqüentemente o designer olha uma determinada tela ou uma determinada mensagem e acredita que são perfeitamente claras e adequadas, mesmo que sejam incompreensíveis para quem não conhece o sistema. Conhecer sobre um sistema é uma via de mão única, impossível voltar e fazer o papel de um novato" (ROCHA).

- **Menos é mais (*less is more*)**

"Uma das freqüentes soluções de design que têm sido adotadas é colocar no sistema todas as opções e características imagináveis, pois se tudo está disponível então todos ficarão satisfeitos. Essa tendência é verificada nos softwares, como editores de texto, por exemplo, que a cada nova versão tem dobrado de tamanho gerando o fenômeno denominado *fatware*. Cada elemento em uma interface acarreta uma sobrecarga ao usuário que tem que considerar se o usa ou não" (ROCHA).

Este principio na WEB é fundamental porque alem de poluir a pagina pode-se inviabilizar o recebimento da mesma pelo usuário dependendo da velocidade de conexão.

- **Help não ajuda (help doesn´t)**

"Muitas vezes, senão na maioria delas, vemos usuários perdidos tentando encontrar informação na enorme quantidade de material de *help* que acompanha um sistema, e quando a encontra não consegue entendê-la. Também, a existência de *helps* acrescenta mais complexidade à interface e na maioria das vezes sem grande efetividade" (ROCHA, 2000).

Usar o *help* na WEB demanda muito tempo facilitando que o usuário desista da operação.

- **Facilidade de aprendizagem (*learnability*)**

"O sistema precisa ser fácil de aprender de forma que o usuário possa rapidamente começar a interagir. É o mais importante atributo de usabilidade, por ser a primeira experiência que qualquer usuário tem com um sistema. Certamente existem sistemas para aplicações altamente especializadas e complexas onde se prevê um extenso trabalho de treinamento em seu uso, mas na maioria dos casos um sistema deve ser fácil de aprender. Quando se analisa a facilidade de aprendizagem, é preciso ter em mente que geralmente o usuário não aprende toda uma interface antes de começar a usá-la. Pelo contrário, o aprendizado ocorre do uso. Portanto, esse fator é avaliado em função

do tempo que o usuário demora para atingir um suficiente grau de proficiência na execução de suas tarefas" (ROCHA).

- **Eficiência**

"O sistema precisa ser eficiente no uso, de forma que uma vez aprendido o usuário tenha um elevado nível de produtividade. Portanto, eficiência refere-se a usuários experientes, após um certo tempo de uso. Um modo típico de avaliar esse atributo é definir de alguma forma o que significa um usuário experiente e avaliar um grupo desses executando tarefas típicas de um sistema" (ROCHA).

- **Facilidade de relembrar (*memorability*)**

"O sistema precisa ser facilmente relembrado, de forma que o usuário ao voltar a usá-lo depois de um certo tempo não tenha novamente que aprendê-lo. Esse atributo tanto se refere a usuários casuais (que é uma categoria com um número grande de usuários na maioria dos sistemas) como para aqueles sistemas utilitários que são inerentemente usados em períodos específicos como os sistemas para confecção de relatórios de atividades trienais, de imposto de renda, etc. Certamente, aumentar a facilidade de aprendizagem também torna a interface mais fácil de ser relembrada, mas tipicamente usuários que retornam a um sistema são diferentes dos usuários principiantes" (ROCHA).

- **Erros**

"Neste contexto, erro é definido como uma ação que não leva ao resultado esperado, um "engano" portanto. O sistema precisa ter uma pequena taxa de erros, ou seja, o usuário não pode cometer muitos erros durante o seu uso e, em errando, deve ser fácil a recuperação, sem perda de trabalho" (ROCHA).

Na WEB o tratamento de erro geralmente vem acompanhado da perda dos dados enviados na submissão da pagina. O usuário simplesmente enlouquece.

- **Satisfação subjetiva**

"Os usuários devem gostar do sistema, ou seja, deve ser agradável de forma que o usuário fique satisfeito ao usá-lo. Esse atributo é muito relevante quando se considera sistemas usados fora do ambiente de trabalho, tais como jogos, sistemas domésticos em geral, etc " (ROCHA).

- **Clareza na arquitetura da informação**

"É essencial que o usuário consiga discernir o que é prioritário e o que é secundário no *site*. Ou seja, antes de mais nada é preciso chegar a um bom arranjo da informação. Os usuários sempre terão dificuldades em encontrar o que procuram, então devem ser ajudados provendo-se um senso de como a informação está estruturada e localizada. Para se conseguir isso, uma das alternativas adotadas em alguns *site*s, é prover um mapa do *site*, de forma que os usuários saibam onde estão e para onde podem ir" (ROCHA).

- **Facilidade de navegação**

"Uma máxima é que o usuário deveria conseguir acessar a informação desejada no máximo em três clique. E conseguir organizar a informação dentro disso já é um bom princípio" (ROCHA).

- **Simplicidade**

"Quem navega quer encontrar o mais rapidamente possível o objetivo da busca. Portanto, a pirotecnia deve ser evitada, dando ao usuário paz e tranqüilidade para que possa analisar a informação. Cuidados devem ser tomados para que a simplicidade não signifique ausência de informação" (ROCHA).

- **A relevância do conteúdo**

"Se nas revistas ou na televisão, por exemplo, a sedução passa muito pela beleza das imagens, na WEB o conteúdo é o que mais importa para atrair e prender a atenção do usuário. Sempre que questionados sobre *site*s, usuários se referem a qualidade e relevância do conteúdo. Um bom texto para essa mídia tem que ser o mais conciso e objetivo possível, não promocional ou publicitário, como impera hoje, com perda de credibilidade. É preciso alterar o estilo de escrita, de forma a ser otimizado para leitores *online* que freqüentemente imprimem textos e que necessitam páginas bem curtas com a informação secundária deixada para páginas de suporte" (ROCHA).

- **Manter a consistência**

"Assim, como para qualquer outro tipo de software, a consistência é um poderoso princípio de usabilidade na WEB. Quando as coisas acontecem sempre do mesmo jeito, os usuários não precisam se preocupar a respeito do que irá acontecer. Ao contrário, eles sabem o que vai acontecer baseados numa experiência anterior. Isso leva a adoção de procedimentos padrões, como por exemplo, o uso de cores. *Layouts* ambiciosos devem ser abandonados. As fontes a serem usadas devem ser as mais comuns, pois o designer não sabe as fontes que o usuário tem instalado. Outro aspecto bastante verificado e que transparece no design, é o de se gerenciar um projeto para a WEB da mesma forma que qualquer outro projeto corporativo tradicional. Isso conduz a um design com uma interface inconsistente. Ao invés disso, um *Website* deve ser gerenciado como um projeto único de interface com o usuário" (ROCHA).

- **Tempo suportável**

"O tempo de carga das páginas deve ser necessariamente curto. Estudos indicam que 10 segundos é o máximo de tempo antes que as pessoas percam o interesse. Mas na WEB os usuários já têm uma baixa expectativa, então esse limite pode aumentar para 15 segundos e mesmo assim ser aceitável" (ROCHA).

- **Foco nos usuários**

"Novamente, todos os princípios podem ser sumarizados em um só: o foco deve estar nas atividades dos usuários. Deixar-se embevecer pelas últimas tecnologias da

WEB irá atrair uns poucos interessados somente na tecnologia. Como cada vez há um número maior de páginas, as pessoas estão se tornando impacientes com *site*s não usáveis e não tem pudor algum em mudar - afinal, há atualmente outros dez milhões de *site*s para ir e nada impede a livre navegação" (ROCHA).

Não é uma tarefa trivial construir um sistema para WEB levando-se em conta os princípios mencionados por ROCHA. O uso somente de HTML e scripts também é um limitador no desenvolvimento do sistema, pois deixa o desenvolvedor com poucos recursos não podendo assim cumprir boa parte dos princípios acima mencionados.

3 MODELOS PARA CONTRUÇÃO DE APLICAÇÕES

O desenvolvimento de aplicações WEB com um bonito visual e que ofereça as vantagens de usabilidade de um sistema desenvolvido para o *desktop* tem no exemplo citado na introdução deste trabalho um argumento muito forte, ou seja, o argumento financeiro. Um sistema que observe os requisitos de usabilidade e leve em conta as necessidades do usuário pode gerar bons lucros para o seu proprietário.

Então a pergunta que se deve fazer agora não é "se devemos desenvolver aplicações com uma interface rica para WEB" e sim "como podemos desenvolver estas aplicações para WEB".

Outro ponto é que as aplicações que serão desenvolvidas para serem usadas na WEB terão que observar os princípios de usabilidade das aplicações *desktop.* Princípios como visibilidade, feedback, facilidade de aprendizado, eficiência, satisfação subjetiva, facilidade de navegação, simplicidade, tempo suportável e foco no usuário terão que ser observados no desenvolvimento das novas aplicações. E para atender esses requisitos teremos que utilizar novas formas e novas tecnologias para o desenvolvimento dessas aplicações.

Neste trabalho foram abordados quatro formas de desenvolvimento de aplicações para a WEB. O termo RIA (Rich internet Application) usado pela Adobe ou (Rich Interative Application) usado pela Microsoft é usado neste trabalho como um sistema que mantém as características das aplicações *desktop* mas são usados WEB e mantidos em servidores de aplicações. Outro fator que influenciou a pesquisa é que o sistema possa ser usado pelo usuário sem a necessidade de instalação da aplicação a não ser os *plugins* do navegador.

A escolha foi projeto OpenLaszlo inicialmente da IBM, o Java WEB Start da Sun , o Flex 2 da Adobe e o Silverlight da Microsoft e teve como principio de escolha apenas referencias na internet sobre RIA. A seguir relacionamos algumas características de cada um desses projetos.

3.1 Laszlo

Projeto de código aberto chamado de OpenLaszlo e distribuído sob a proteção de de uma OSI (*certified common public licence*) de autoria da IBM. O projeto é composto de uma linguagem (LZX), um servidor, um compilador e *runtimes* como mostra a figura 3.1.

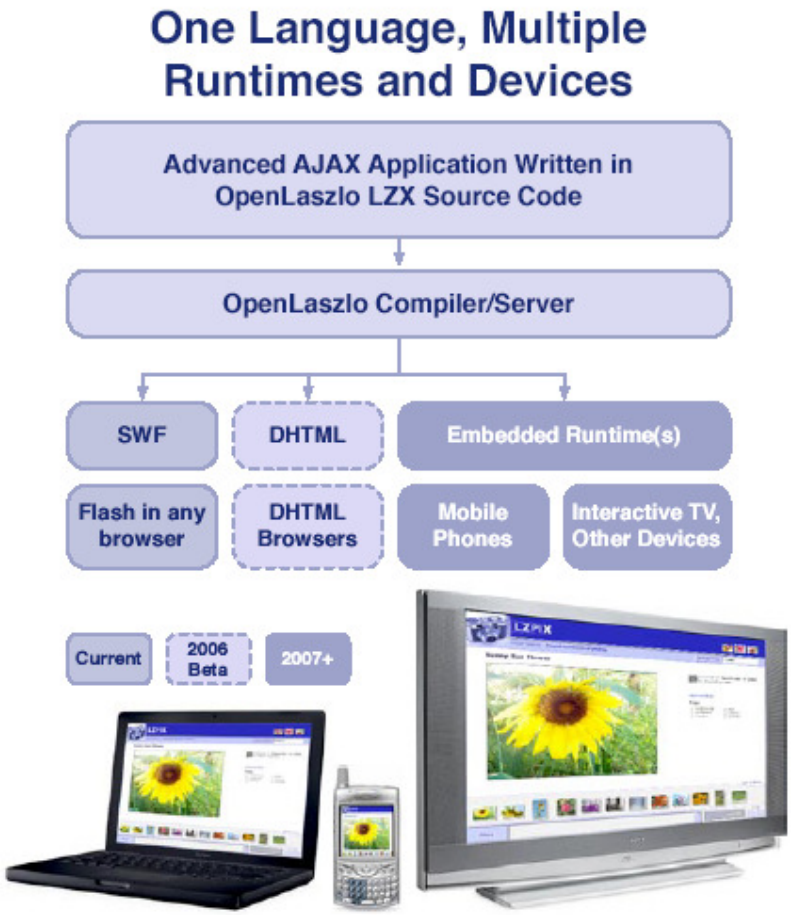

Figura 3.1: Arquitetura Open Laszlo (OPENLASZLO, 2006).

- **Linguagem LZX**

É uma linguagem XML declarativa e de marcação. Declarativa porque define o que são cada componente e é usada como linguagem de marcação do tipo HTML e pode receber parâmetros. Exemplo:

```
<window>
    < button text=”HELLO”/>
</window>
```

Define uma janela e um botão e coloca o botão com um rotulo “HELLO” em uma janela.

Inclui operações primitivas que implementam animações, ligações com bases de dados, comunicação pela rede com o servidor. Utiliza tags XML para descrever a estrutura da aplicação e utiliza *JavaScript* para a lógica da aplicação no lado do cliente. É também uma linguagem orientada a objetos que suporta herança, polimorfismo, encapsulamento. Suporta a inclusão de bibliotecas de componentes. Exemplo:

```
<canvas height="250">
  <dataset name="weatherdata" request="true"
  src="http://www.laszlosystems.com/cgi-pub/weather.
cgi?zip=10022"/>
  <grid datapath="weatherdata:/weather"“contentdatapath="forecast/
day"/>
</canvas>
```

Figura 3.2: Exemplo de Código LZX (OPENLASZLO, 2006).

LZX é desenhado para ser independente de *runtimes.* Pode ser traduzido para o flash *player* ou DHTML e para JME.

Este exemplo de código tem como resultado a figura 3.3.

label	imageurl	desc	temp
TODAY	http://www.srh.noaa.gov/	Mostly Sun	Hi 65°F
Tonight	http://www.srh.noaa.gov/	Chance Rai	Lo 50°F
Wednesday	http://www.srh.noaa.gov/	Breezy	Hi 60°F
Wednesday Night	http://www.srh.noaa.gov/	Mostly Clea	Lo 38°F
Thursday	http://www.srh.noaa.gov/	Breezy	Hi 58°F
Thursday Night	http://www.srh.noaa.gov/	Mostly Clea	Lo 47°F
Friday	http://www.srh.noaa.gov/	Breezy	Hi 63°F
Friday Night	http://www.srh.noaa.gov/	Breezy	Lo 50°F
Saturday	http://www.laszlosystems	Partly Clou	Hi 63°F

Figura 3.3: Resultado do Código LZX exemplo (OPENLASZLO, 2006).

- **OpenLaszlo Server**

É um servidor implementado como um servlet Java e roda dentro de um servlet container ou em servidor J2EE compatível (JBOSS, TOMCAT, WEBSPHERE, etc) e é independente de sistema operacional. Pode mudar a codificação de mídia de um formato para outro, pode ser acessado via SCAP e XML-RPC e faz mapeamento de objetos openlaszlo para objetos Java.

Seu modelo de segurança (*segurity model*) suporta SSL. Trafega dados pela internet com criptografia usando SSL sobre HTTPS. As aplicações no cliente rodam dentro das *sendbox* impedidas de acessar recursos da maquina cliente. *Webservices* e comunicação com base de dados podem usar autenticação por usuário (*per-user authentication model*)

3.2 Java Web Start

Sistema desenvolvido pela Sun Microsistens para distribuir aplicações J2SE via internet. As aplicações J2SE usam em sua interface objetos da biblioteca Swing ou AWT. São componentes desenvolvidos para aplicações *desktop*. As aplicações distribuídas via Java WEB Start (JWS) são na verdade aplicações com interface de *desktop*. A arquitetura JWS é mostrada na figura 3.4.

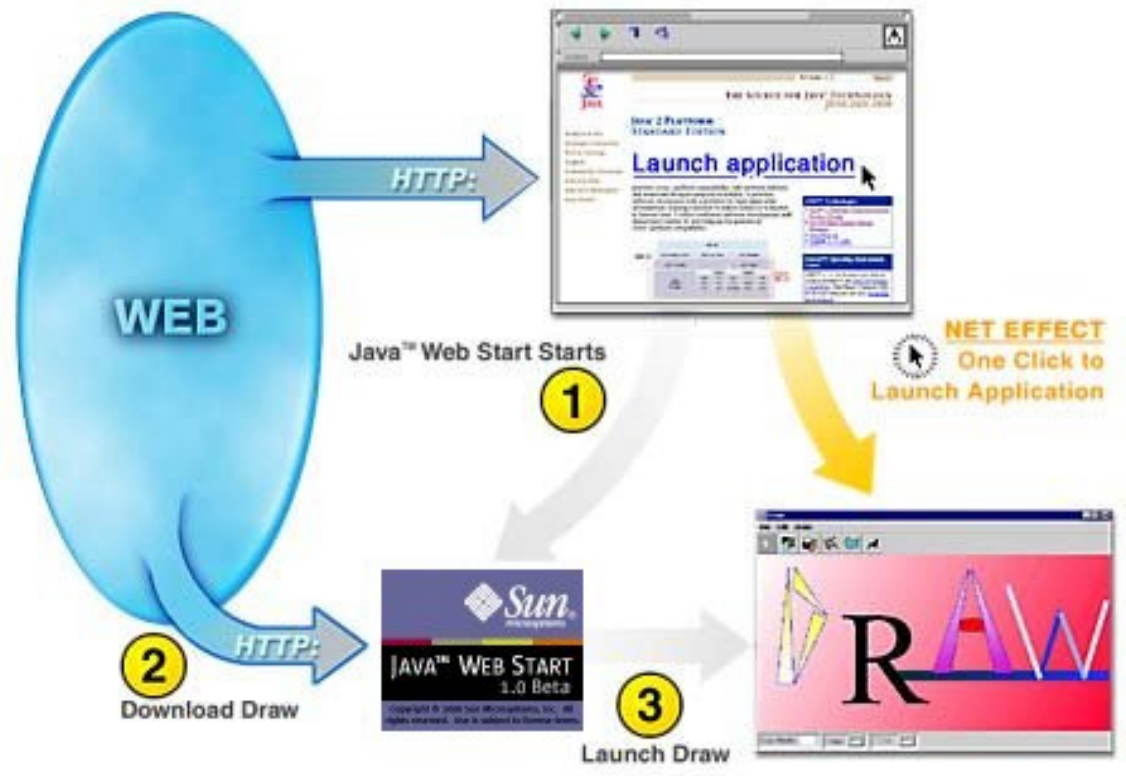

Figura 3.4: Arquitetura Java WEB Start
(http://java.sun.com/products/javawebstart/architecture.html, 2006).

A aplicação distribuída via JWS esta associada a um navegador WEB. Quando o usuário acessa uma pagina com contenha um link para um tipo especial de arquivo (arquivo JNLP) o browser lança o Java WEB Start que por sua vez transfere a aplicação pela rede, coloca em memória e roda a aplicação. Esta operação é executada sem a intervenção do usuário, a não ser o primeiro *click* no link da pagina inicial.

O Jave WEB Start executa exclusivamente aplicações escritas para a plataforma J2SE e o aplicativo pose ser executado nas plataformas Windows 98/NT/2000/ME/XP, Linux e Solaris. Pode também sem executado sem a intervenção do navegador podendo ser acionado através de um atalho ao arquivo JNLP como se fosse uma aplicação local.

A tecnologia por traz do Java WEB Start é o Java Network Launching Protocol e API (JNLP). Foi desenvolvido através do Java Community Process.

Quanto a segurança aplicações disparadas via Java WEB Start são executadas na maquina virtual dentro das chamadas s*andbox,* que limitam o acesso da aplicação aos recursos da maquina local. Caso a aplicação necessite de acesso a recursos da maquina local o Java WEB Start lança um aviso de segurança mostrando as informações sobre o desenvolvedor do sistema. O usuário decide se confia ou não no desenvolvedor e então libera o acesso aos recursos da maquina local.

Uma combinação importante entre Java WEB Start e a plataforma Java Enterprise Edition (JEE) é mostrada na figura 3.5.

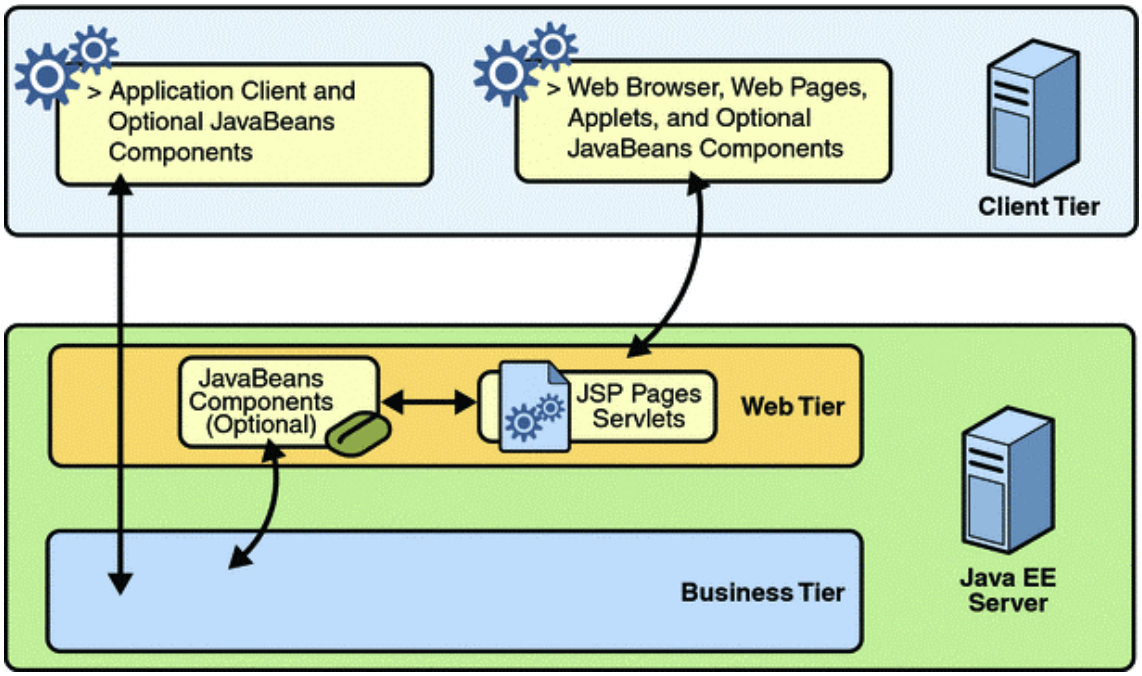

Figura 3.5: Arquitetura JEE http://java.sun.com/javaee/5/docs/tutorial/doc/bnaay.html

A aplicação cliente distribuída pelo Java WEB Start poderá estar associada a uma aplicação JEE. A aplicação cliente terá acesso ao contêiner EJB do servidor de aplicação. No contêiner EJB são colocados os objetos da camada de negocio da aplicação, normalmente *session beans* e *entity beans*.

O arquivo JNLP é um arquivo XML com os seguintes elementos base:

- JNLP
- INFORMATION
- RELATED_CONTENT
- SECURITY
- RESOURCES
- APPLICATION_DESC
- APPLET_DES

Podemos ver um exemplo do arquivo JNLP na figura 3.6:

```
<jnlp
  spec="1.5+"
  codebase="http://my_company.com/jaws/apps"
  href="swingset2.jnlp">
  <information>
    <title>SwingSet2 Demo Application</title>
    <vendor>Sun Microsystems, Inc.</vendor>
    <homepage href="docs/help.html"/>
    <description>SwingSet2 Demo Application</description>
    <description kind="short">A demo of the capabilities
    of the Swing Graphical User Interface.</description>
    <icon href="images/swingset2.jpg"/>
    <icon kind="splash" href="images/splash.gif"/>
    <offline-allowed/>
    <association mime-type="application-x/swingset2-file" extensions="swingset
    <shortcut online="false">
      <desktop/>
      <menu submenu="My Corporation Apps"/>
    </shortcut>
  </information>
  <information os="linux">
    <title> SwingSet2 Demo on Linux </title>
    <homepage href="docs/linuxhelp.html">
  </information>
  <security>
      <all-permissions/>
  </security>
  <resources>
    <j2se version="1.4.2+" java-vm-args="-esa -Xnoclassgc"/>
    <jar href="lib/SwingSet2.jar"/>
  </resources>
  <application-desc main-class="SwingSet2"/>
</jnlp>
```

Figura 3.6: Exemplo do Arquivo JNLP
http://java.sun.com/javaee/5/docs/tutorial/doc/bnaay.html

O JNLP é um protocolo para permitir que aplicações remotas possam ser baixadas e executadas na maquina cliente. Todas os arquivos (arquivos de classes, arquivos de imagens, etc) necessários para execução da aplicação são transferidos para a maquina cliente não sendo necessário que estes arquivos estejam na maquina cliente à *priori*.

O arquivo JNLP é um arquivo XML e é baixado pelo navegador que não sabe o que fazer com este tipo de arquivo. Então passa o arquivo para o JNLP Helper. O JNLP Helper vai ler a lista de recursos requeridos e transferir os arquivos necessários para a execução da aplicação. Porem antes da execução o JNLP Helper verifica a compatibilidade da versão da maquina virtual Java e as regras de segurança e se tudo estiver de acordo inicia a aplicação.

3.3 Adobe Flex 2

Adobe Flex 2 é um pacote de ferramenta e tecnologia desenvolvido pela Adobe para a construção de aplicações WEB. É composto dos seguintes itens:

- Adobe Flex 2 SDK
- Adobe Flex Builder 2
- Adobe Flex Data Services 2
- Adobe Flex Charting 2

O Adobe Flex 2 SDK é a base da tecnologia para a criação de aplicações. Consiste em um conjunto de classes (*framework*), compilador, *debugger*, duas linguagens de programação MXML e Actionscript. Inclui ainda os códigos fontes do conjunto de classes para customização pelo desenvolvedor.

O MXML é uma linguagem XML usada para o desenvolvimento do lay out da aplicação. Fornece *tags* que correspondem às classes do SDK Flex simplificando o desenvolvimento visual da aplicação. O MXML também é usado para o desenvolvimento de componentes não visuais.

O ActionScript é a linguagem para ser usada no lado cliente e é baseada no ECMAScript similar ao JavaScript. É responsável pelo controle e manipulação dos objetos visuais.

O Adobe Flex Builder 2 é o ambiente de programação, design e teste (IDE) para aplicações Flex 2. Possui um compilador de debugger passo a passo. Pode ser usado sozinho ou como um plu-in para o Eclipse. O Eclipse é uma ferramenta de desenvolvimento de código aberto.

Possui editores de MXML, ActionScript e Cascating Style Sheets (CSS). A figura 3.7 mostra a interface do Adobe Flex Builder 2.

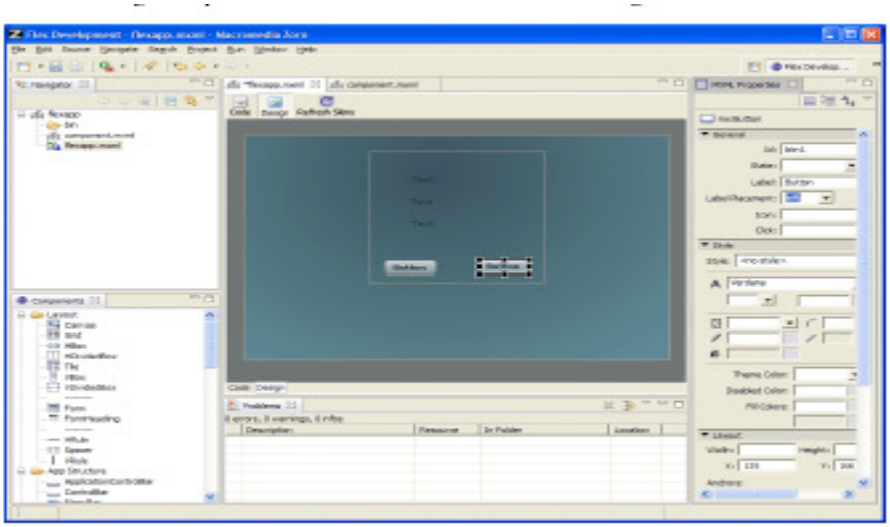

Figura 3.7: Interface Adobe Flex Builder 2.

O Adobe Flex Data Services 2 é um serviço de mensagens que roda em um servidor JEE compatível ou em contêiner Servlet. Possui componente para sincronização de dados entre o servidor e o cliente ou múltiplos clientes. Permite a comunicação entre clientes. Tem um servidor de *streaming* para receber dados em tempo real.

Sua arquitetura possibilita a integração com mecanismos como JMS, Hibernate, EJB entre outros mecanismos de persistência. Permite autenticação de usuário para acesso a recursos do servidor. Acessa objetos remotos usando o protocolo AMF.

A figura 3.8 mostra a arquitetura do O Adobe Flex Data Services 2 :

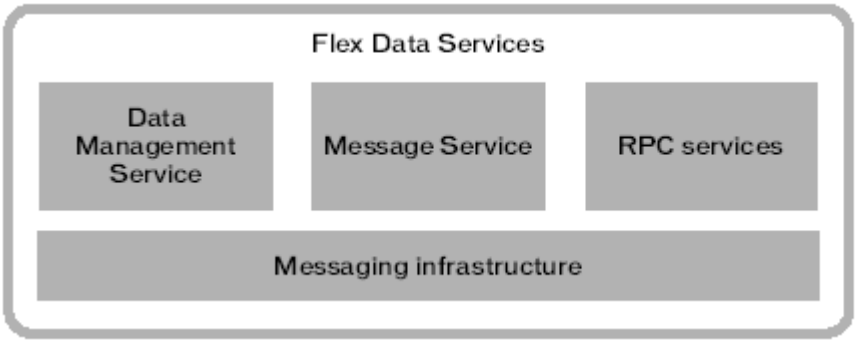

Figura 3.8: arquitetura do Adobe Flex Data Services 2

O Adobe Flex Charting 2 é uma funcionalidade que facilita a visualização de um conjunto de dados no formato de gráficos de duas dimensões. Suporta um conjunto de tipos de gráficos, tais como, barra, pizza entre outros.

O modelo de aplicativo desenvolvido com o Adobe Flex 2 pode ser representado pelo modo como ele é executado no servidor. São três os modos de execução:

- Client-side Only Applications.

A aplicação é executada no cliente e não usa os recursos do servidor. A figura 3.9 mostra este modelo.

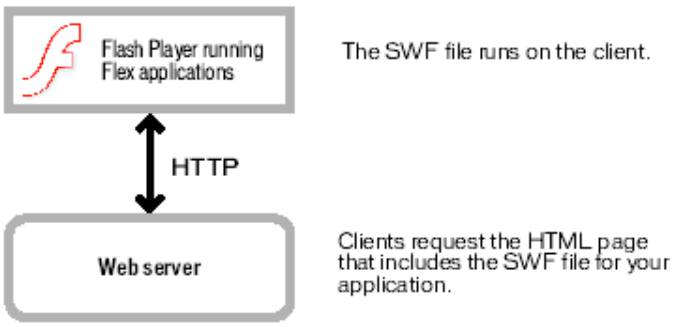

Figura 3.9: Client-side Only Applications

- Data Acess with HTTPServices and WebServices

A aplicação se comunica com o servidor atraves de chamadas HTTP(GET,POST) ou utilizando WEB services (SOAP). A figura 3.10 mostra este modelo.

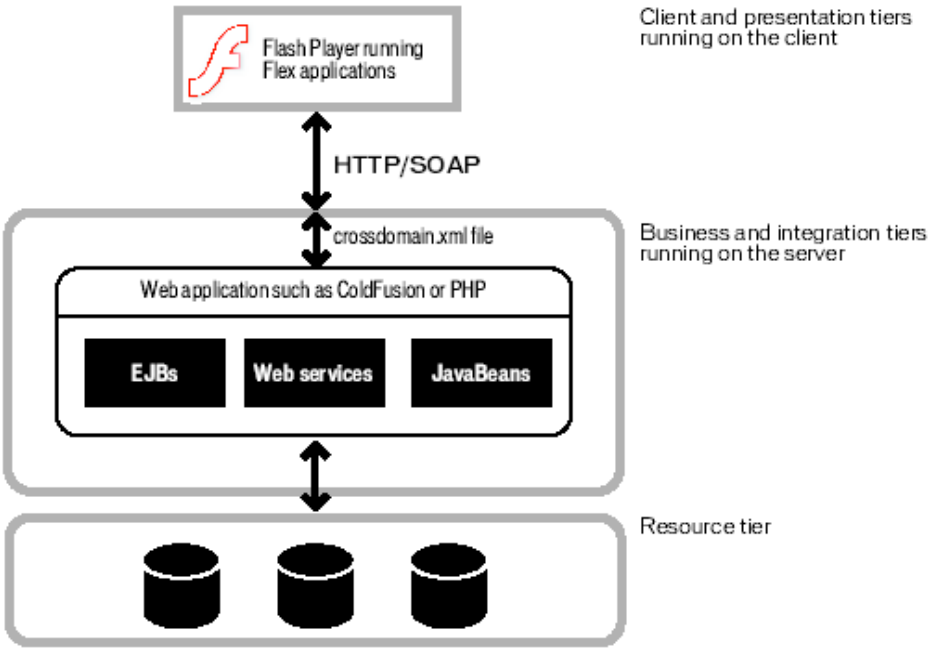

Figura 3.10: Data Acess with HTTPServices and WebService

- Data Acess with Flex Data Services

A aplicação usa a arquitetura Adobe Flex Data Services 2 que proporciona vantagens no acesso ao servidor, tais como, sincronização, segurança e comunicação. A figura 3.11 mostra este modelo.

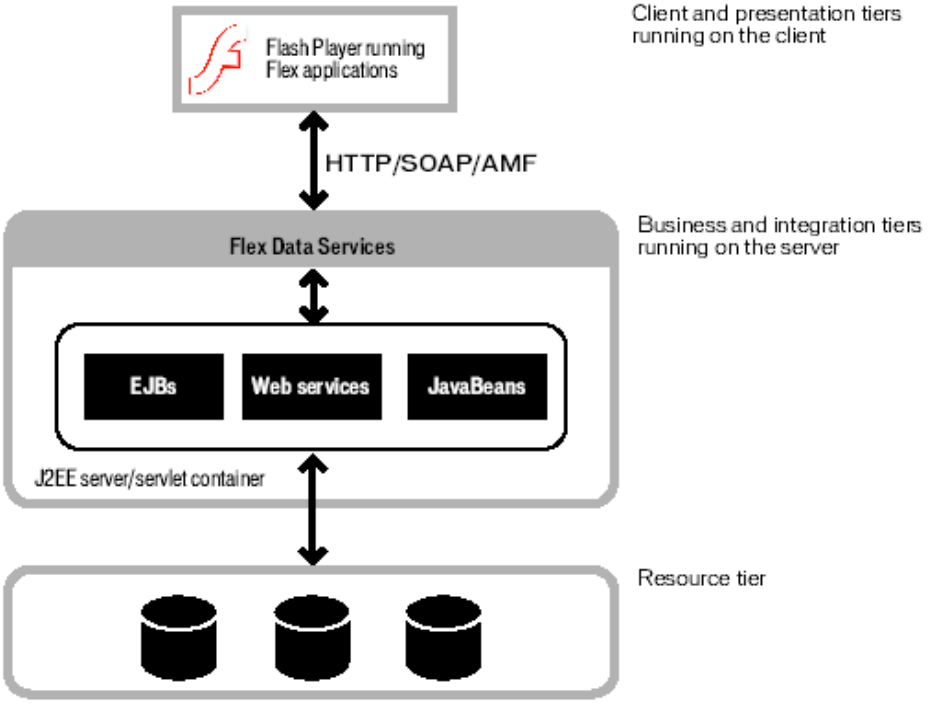

Figura 3.11 : Data Acess with Flex Data Services

3.4 Microsoft SilverLight

O Microsoft SilverLight é um projeto desenvolvido pela Microsoft para a construção de aplicações interativas ricas (Rich Interactive Applications) para uso na WEB. A Adobe usa o termo Aplicações ricas para Internet (Rich Internet Applications) para suas aplicações desenvolvidas com Flex 2.

O Microsoft SilverLight suporta AJAX, Python, Ruby e as linguagens .NET, visual basic e C#. É composta por um run time, um SDK e ferramentas de desenvolvimento e dising. A linguagem usada para o desing é o XAML (Extensible Application Markup Language) que segundo a Microsoft foi usada para o desenvolvimento da interface do Windows Vista. A figura 3.12 mostra a arquitetura do SilverLight.

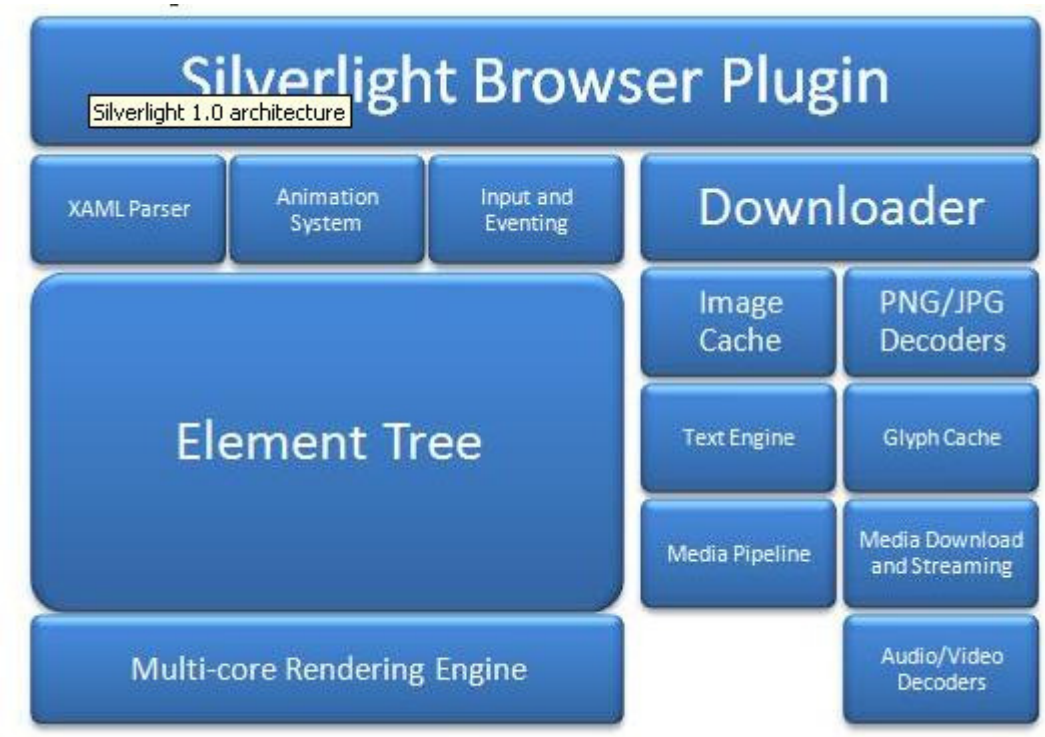

Figura 3.12: Arquitetura Microsoft SilverLight

O runtime do Silverlinght tem suporte para o internet explorer, mozila firefox e Apple safári. O runtime executa o código XAML e utiliza o javascript para manipulação dos objetos Silverlignt. Tem suporte para download incremental aumentando a interatividade.

O Element Tree tem um conjunto rico em elementos gráficos e utiliza o conceito de arvore do HTML e é manipulado com Javascript.

O Rendering Engine é otimizada para renderizar os componentes da arvore Silverlignt de forma incremental e somente os componentes visíveis.

O Input and Eventing fornece um conjunto de eventos disponíveis para os objetos Silverlignt. O Animation System adiciona movimento e interatividade aos elementos gráficos. No Image Cach a imagem é mantida no cliente e pode ser referenciada em varias partes do programa. O Media Pipeline recebe conteúdo progressivo no formato HTTP download e HTTP streaming.

A languagem *Extensible Application Markup Language* (XAML) é uma linguagem declarativa utilizadas para a criação de objetos de interface com o usuário. Cada tag XAML representa uma classe de objeto definida no *Windows Presentation Foudation* (WPF). Uma aplicação WPF normalmente consiste em duas linguagens, uma de marcação e outra para código. A XAML trabalha em conjunto com as linguagens.NET, visual basic e C#. A figura 3.13 mostra integração do XAML com o C#.

XAML Copy Code

```
<Page
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="MyNamespace.MyPageCode">
  <Button Click="ClickHandler" >Click Me!</Button>
</Page>
```

C# Copy Code

```
namespace MyNamespace
{
  public partial class MyPageCode
  {
    void ClickHandler(object sender, RoutedEventArgs e)
    {
      Button b = e.Source as Button;
      b.Background = Brushes.Red;
    }
  }
}
```

Figura 3.13: Exemplo de Código XAML e C#

3.5 Tabela Comparativa

	Laszlo	Java Web Sart	Flex 2	Silverlignt
RunTime	FLASH	JRE	FLASH	Silverlignt
Linguagem Declarativa	LZX(xml)	JAVA(swing/ awt)	MXML(xml)	XAML(xml)
Linguagem de Manipulação	JavaScript	Java	ActionScript	JavaScript

Podemos observar alguns pontos em comum que as novas tecnologias apresentam conforme a tabela acima. São eles:

Utilização de r*untime*. As aplicações "RIA" serão utilizadas no lado cliente e farão uso de *runtimes* instalados nos navegadores dos clientes. O FLASH da Adobe, segundo dados da própria Adobe, esta instalado em 90% dos navegadores o da Microsoft é um projeto mais recente, esta na versão 1.0 e foi lançado em 2007.

Uso de uma linguagem declarativa do tipo XML em conjunto com outra linguagem de manipulação dos objetos. A linguagem declarativa constrói os objetos de interface com o usuário e utiliza outra linguagem de manipulação que roda no cliente e manipula os objetos de interface. O *Javascript* é a linguagem básica utilizada pelas tecnologias estudadas.

4 ESTUDO DE CASO

Para a demonstração de uma aplicação com interface amigável com o usuário foi escolhida uma aplicação desenvolvida o Adobe Flex 2 Builder. A escolha do Adobe Flex para o estudo de caso foi motivada pela grande quantidade de material de consulta na internet.

O Adobe Flex Builder 2 é uma ferramenta de propriedade da Adobe e a versão que foi utilizada é a de demonstração valida por trinta dias. Após este prazo é necessário comprar a ferramenta.

A aplicação exemplo do Adobe Flex Builder 2 foi incorporada a uma aplicação WEB desenvolvida no NetBeans 5.5 e que roda no servidor de aplicação da Sun (Sun Java System Application Server Platform Edition 9.0).

A aplicação exemplo simula uma loja virtual de telefones celulares com três vistas distintas e com funcionalidades diferentes. Avista inicial mostra a vitrine da loja e não tem funcionalidade estudada neste trabalho. A figura 4.12 mostra esta vista.

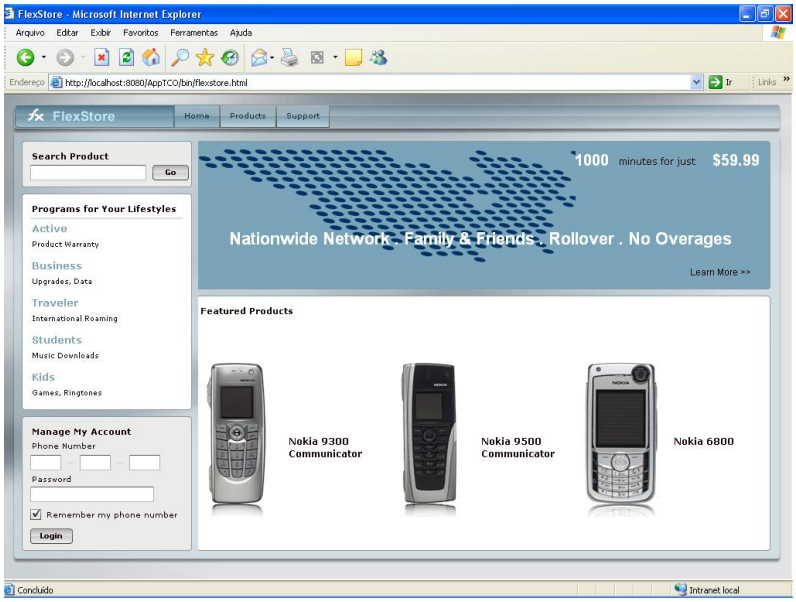

Figura 4.12: Vitrine da Loja

A segunda vista mostra a lista de produtos oferecidos pela loja virtual. Nesta vista é que será observado algum dos recursos avaliado neste trabalho. A figura 4.13 mostra esta vista.

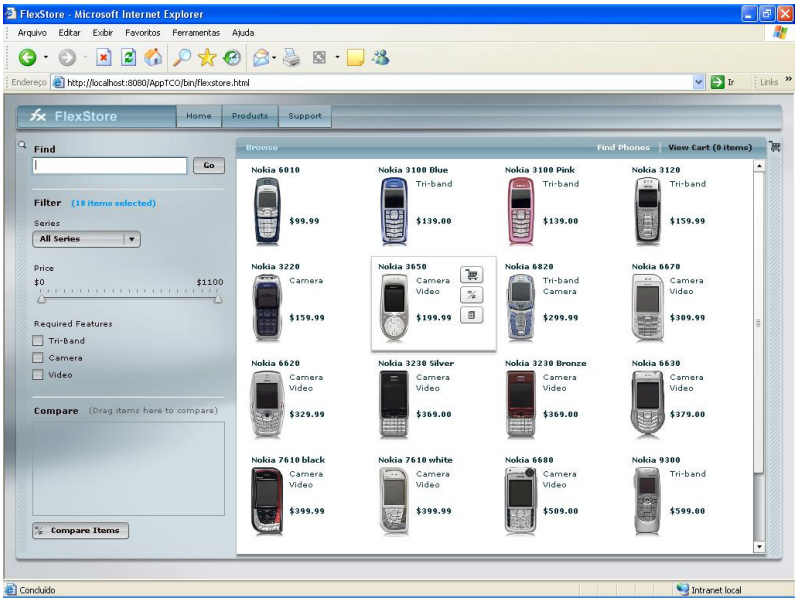

Figura 4.13: Vista de Produtos da Loja Virtual

Um dos requisitos importantes de um sistema é não deixar o usuário perdido. Neste ponto a aplicação modelo é exemplar. O catalogo de produtos e as opções para filtrar-los podem ser executados sem alteração de pagina. Tudo pode ser visto na mesma pagina inclusive o carinho de compras. Nas figuras 4.14 e 4.15 pode-se ver o carinho de compras e o recurso de “arrastar e soltar” da aplicação.

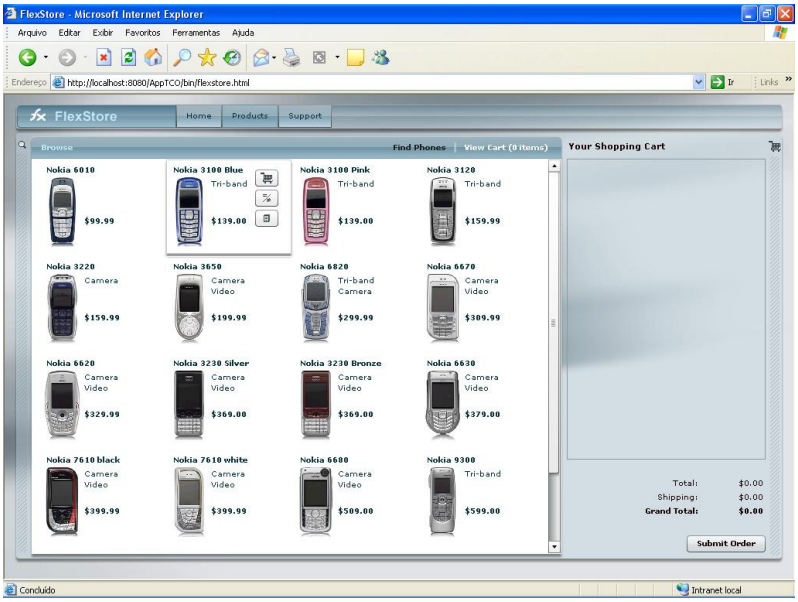

Figura 4.14 Catalogo de Produtos e Carrinho de Compras da Loja Virtual

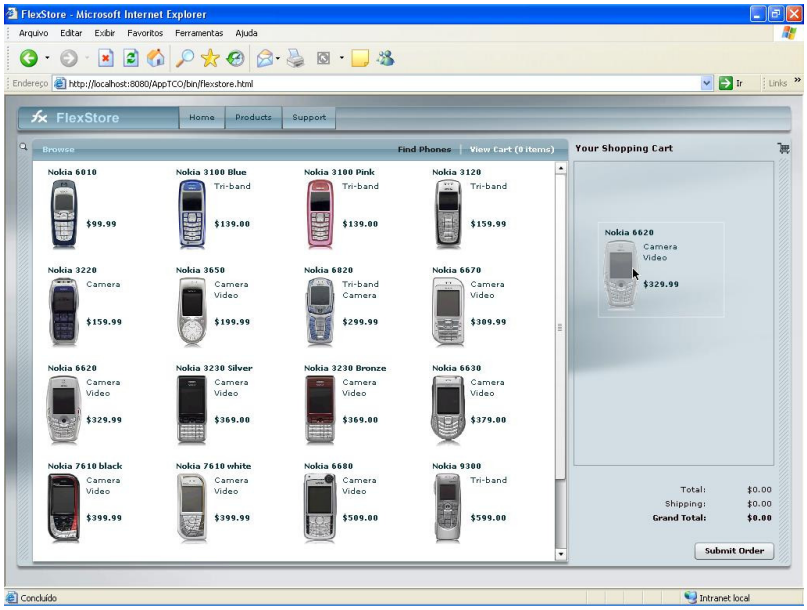

Figura 4.15: Recurso de Arrastar e Soltar da Loja Virtual

Após arrastar e soltar o produto no carinho de compras a aplicação já calcula o valor da compra bem como define a quantidade inicial do produto. A figura 4.16 mostra o resultado.

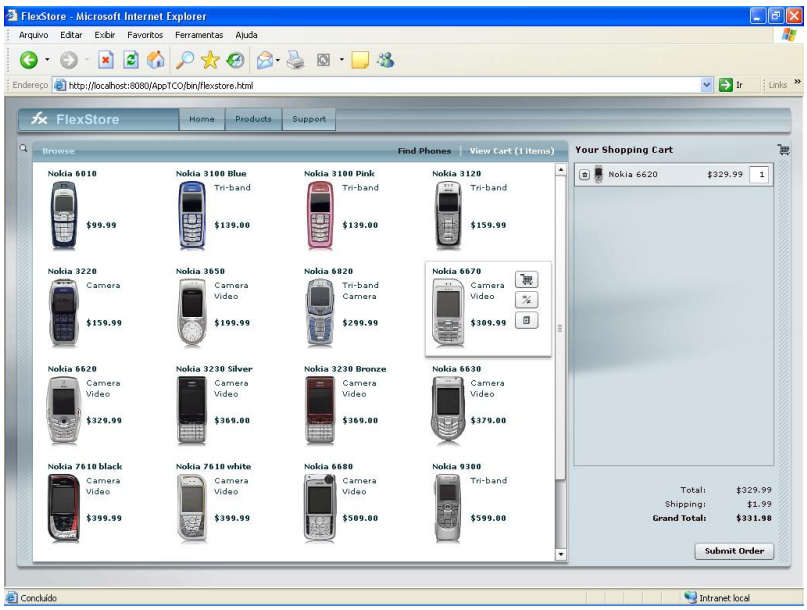

Figura 4.16: Recurso de Arrastar e Soltar da Loja Virtual

Outro recurso da aplicação é a filtragem dos produtos. A filtragem é feita de forma dinâmica não exigindo trocas de paginas. O usuário só observa a mudança no catalogo de produtos. As figuras 4.17 e 4.18 mostram esta funcionalidade.

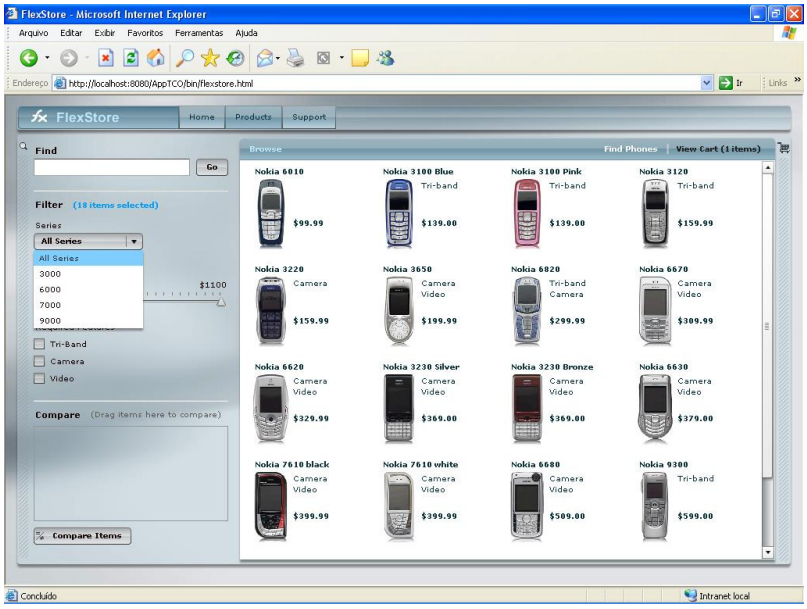

Figura 4.17: Recurso de Arrastar e Soltar da Loja Virtual

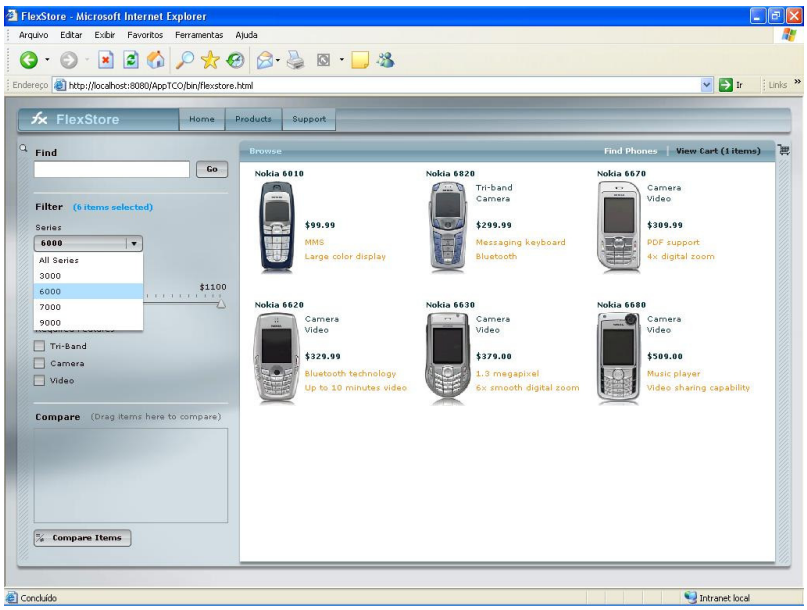

Figura 4.18: Recurso de Arrastar e Soltar da Loja Virtual

Ainda pose-se observar outro recurso aplicação que é a utilização de um vídeo de demonstração do produto. Quando o usuário seleciona um produto a aplicação mostra um painel com duas abas. Na primeira são mostrados a foto e detalhes do produto. Na

segunda é mostrado um vídeo de demonstração do produto. As figuras 4.19, 4.20, 4.21 mostram esta funcionalidade.

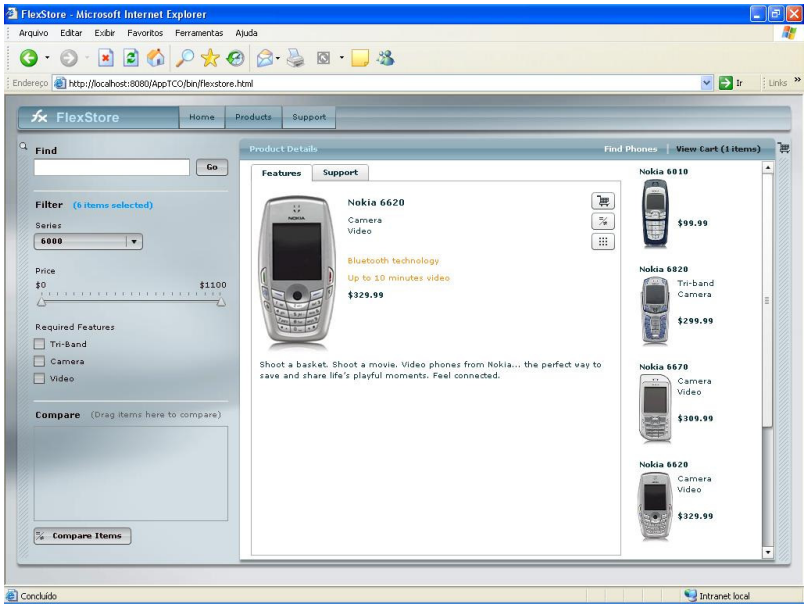

Figura 4.19: Recurso de Arrastar e Soltar da Loja Virtual

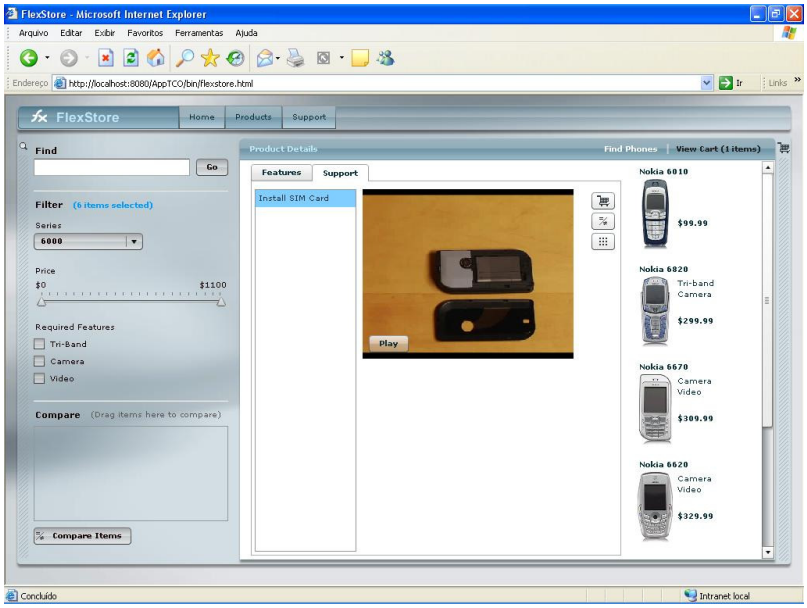

Figura 4.20:Recurso de Arrastar e Soltar da Loja Virtual

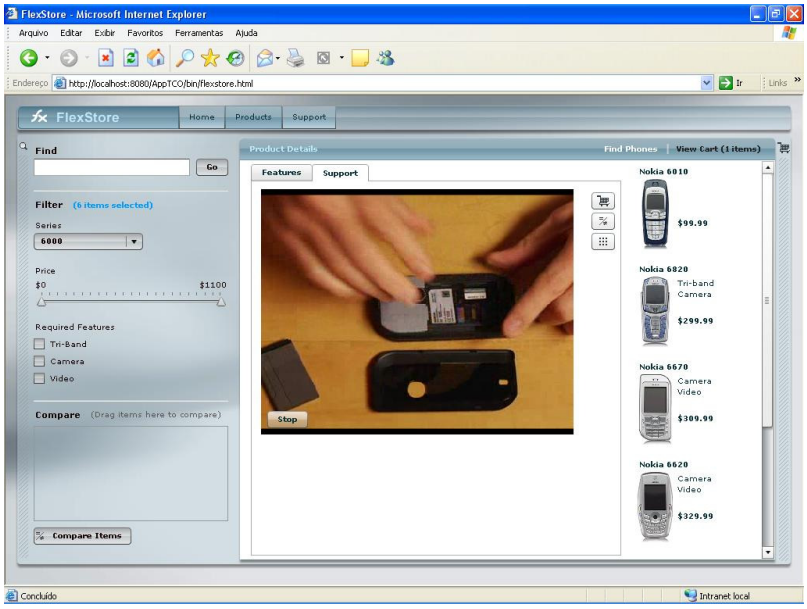

Figura 4.21: Recurso de Arrastar e Soltar da Loja Virtual

Uma importante funcionalidade da aplicação é a mostrar as mensagens na forma "modal", isto é, os recursos da aplicação são desativados até que o usuário pressione o botão da mensagem. A figura 4.22 mostra esta funcionalidade.

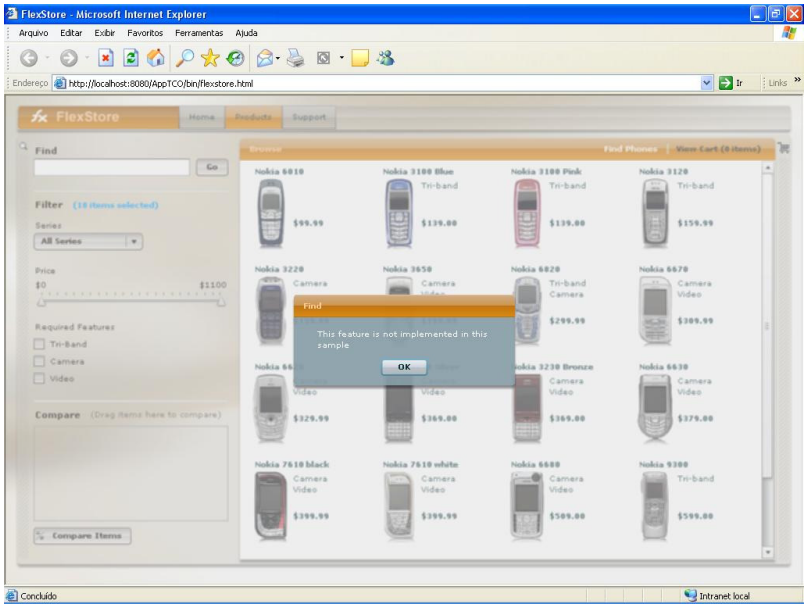

Figura 4.22: Recurso de Arrastar e Soltar da Loja Virtual

5 CONCLUSÃO

Como foi visto na introdução deste trabalho os sistemas desenvolvidos levando-se em conta o usuário podem gerar ganhos financeiros significativos como mostrou o exemplo da IBM.

Quando surgiram as interfaces gráficas com o usuário, os programadores da época acostumados a programar seus sistemas em terminais tiveram resistência a adotar a nova tecnologia. Com as interfaces gráficas, os usuários tinham mais opções, tais como, abrir mais de uma janela, fechar a janela a qualquer momento entre outras, e os programadores tiveram que prever em seus sistemas as novas opções do usuário.

Agora o novo modelo de desenvolvimento de aplicações para a WEB trás novos desafios para os programadores. O uso de mais de uma linguagem de programação, sistemas sendo executados em clientes diversos e a própria internet que um ambiente caótico por natureza são os novos desafios que devem ser vencidos pelos programadores com ganhos significativos para o usuário.

O desenvolvimento de aplicações hoje em dia dispõe de um conjunto de ferramentas para a elaboração de projetos multicamadas. Mapeamento objeto relacional automatizam a persistência dos dados da aplicação. A UML proporciona aos projetistas ferramentas eficientes para representar a aplicação, e finalmente começam a aparecer as ferramentas para elaboração de interfaces gráficas com funcionalidades das linguagens *desktop* para as aplicações WEB.

Como foi visto no estudo de caso, já é possível desenvolver aplicações capazes de oferecer funcionalidades exigidas e orientadas aos princípios da interação humano-computador. Visibilidade , feedback instantâneo, facilidade de aprendizado pelo usuário,eficiência, satisfação subjetiva, facilidade de navegação e foco no usuário são princípios que podem ser aplicados no desenvolvimento das novas aplicações para a WEB.

No entanto esta nova forma de desenvolvimento de aplicações tem o seu preço. O uso de mais de uma linguagem de programação (no estudo de caso foram três, XML, ActionScript e MXML) exige dos programadores um curva de aprendizado mais íngreme e normalmente exige uma equipe multidisciplinar para o bom desenvolvimento da aplicação.

O desenvolvimento de ferramentas e ambientes de desenvolvimento integrados que permitam desenvolver completamente a aplicação em todas as suas camadas também é um desafio a ser vencido.

A velocidade de conexão dos usuários com a internet também é outro fator limitante para adoção das tecnologias aqui estudadas. No entanto esta restrição vem sendo minimizada com o oferecimento de conexões rápidas a um custo razoável.

Grandes grupos da área de software, tais como os mostrados neste trabalho, IBM, Sun, Microsoft e Adobe tem suas estratégias para desenvolver e integrar aplicações com interfaces ricas para o usuário. Certamente a concorrência será acirrada e com ganhos para as aplicações e os usuários.

A arquitetura da aplicação modelo estudada neste trabalho, utilizando uma linguagem para componentes gráficos, uma linguagem para manipulação destes componentes, outra linguagem para o desenvolvimento das demais camadas da aplicação e o uso de *runtime* no navegador do cliente mostra uma tendência de como serão as aplicações futuras já que são adotadas pelos quatro propostas estudadas aqui.

Pela facilidade de uso demonstrada no estudo de caso e pela beleza subjetiva que se pode obter com estas novas técnicas de desenvolvimento de interfaces é possível concluir que aplicações futuras serão mais ricas em recursos visuais, mais fáceis para o usuário usar, mais ricas com adoção de sons e vídeos e certamente novas áreas de aplicações poderão ser desenvolvidas.

REFERÊNCIAS

ADOBE CO. **Start Guide**. Disponível em: < http://www.adobe.com/br/products/flex/>. Acesso em nov. 2007.

MICROSOFT. **Silver Lingt Documentation.** Disponível em: <http://www.microsoft.com/silverlight/>Acesso em nov. 2007.

OPEN LASZLO. **Open Architecture Framework for Advanced Application.** Nov. 2006**.** Disponível em: <http://www.nied.unicamp.br/inicial.php>. Acesso em nov. 2007.

PERSEGONA, M. F. M.; ALVES, I. T. G. **História da Internet:** Origens do E-Gov No Brasil. Disponível em: <http://www.nied.unicamp.br/inicial.php> . Acesso em nov. 2007.

ROCHA, H. V.; BARANAUSKAS, M. C. **Design e Avaliação de Interfaces Humano-Computador.** Núcleo de Informática Aplicada à Educação Unicamp. Disponível em: <http://www.nied.unicamp.br/inicial.php> . Acesso em nov. 2007.

SUN SYSTEM. **Java Network Lauching Protocol.** Disponível em: <http://jan.newmarch.name/java/jini/tutorial.2.08/JNLP.html>. Acesso em nov. 2007.

SUN SYSTEM. **JNLPFileSintax.**Disponível em: <http://java.sun.com/j2se/1.5.0/syntax.html>. Acesso em nov. 2007.

W3.ORG. **The Original Proposal of the WWW, HTMLized**. Disponível em: http://www.w3.org/history/1989/proposal.html>. Acesso em nov. 2007.