Desenvolvimento Web e Comunicação Móvel Aplicados em um Sistema de Gerenciamento de Compromissos

Research · July 2015

CITATIONS
0

READS
824

1 author:

Venilton FalvoJr
University of São Paulo
9 PUBLICATIONS **25** CITATIONS

SEE PROFILE

CENTRO UNIVERSITÁRIO DE ARARAQUARA - UNIARA
DEPARTAMENTO DE CIÊNCIAS DA ADMINISTRAÇÃO E TECNOLOGIA

ENGENHARIA DE COMPUTAÇÃO

VENILTON FALVO JÚNIOR

DESENVOLVIMENTO WEB E COMUNICAÇÃO MÓVEL APLICADOS EM UM SISTEMA DE GERENCIAMENTO DE COMPROMISSOS

ARARAQUARA
2011

VENILTON FALVO JÚNIOR

DESENVOLVIMENTO WEB E COMUNICAÇÃO MÓVEL APLICADOS EM UM SISTEMA DE GERENCIAMENTO DE COMPROMISSOS

Trabalho de Conclusão de Curso apresentado ao Departamento de Ciências da Administração e Tecnologia, do Centro Universitário de Araraquara UNIARA,como parte dos requisitos para obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. Me. Rodrigo Daniel Malara

ARARAQUARA

2011

FOLHA DE APROVAÇÃO

Centro Universitário de Araraquara - UNIARA
Departamento de Ciências da Administração e Tecnologia

Curso de ENGENHARIA DE COMPUTAÇÃO

Aluno: _____

Título do Trabalho: _____

_____

BANCA EXAMINADORA

Orientador Prof. Me. _____

Assinatura:_____

1º Examinador - Prof. Me. _____

Assinatura:_____

Nota: _____

Araraquara, de dezembro de 2011.

Dedico esse trabalho a Deus e a todas as pessoas que me ajudaram direta e indiretamente na conquista da minha graduação, em especial à minha noiva Catherine e minha mãe Marilda pelo amor e apoio incondicionais.

AGRADECIMENTOS

Primeiramente, agradeço minha mãe Marilda Jussara Solcia, por me proporcionar não só este momento de realização pessoal, como também todo o amor e necessários durante todos os anos da minha vida.

Agradeço meu pai Venilton Falvo pelo apoio sem hesitação à minha graduação.

Agradeço a minha noiva Catherine Helen Martins, por ser minha companheira e melhor amiga presente em todos os momentos com total amor, carinho e dedicação. Até mesmo quando eu era quem deveria dá-los.

Alguns dos meus colegas de trabalho também foram fundamentais para um bom desenvolvimento do trabalho, são eles: Andre Luiz Bacaglini, Drielli Peres Costa, Fabio Montezuma, Gustavo Scatolin Bergamim, Lais Garcia e Matheus Rodrigues de Almeida.

Agradeço também aos meus colegas de classe pela amizade e colaboração partilhadas nestes cinco anos.

Ao professor Rodrigo Daniel Malara, pelo conhecimento, apoio e conselhos oferecidos no decorrer deste trabalho e também a todos os professores do curso.

RESUMO

A carência de aplicações *Web* que interagem com dispositivos móveis estimulou o desenvolvimento deste trabalho. Nele são apresentados alguns dos principais conceitos e tendências do desenvolvimento *Web* nas plataformas *Java* e.NET.

O objetivo da utilização de duas plataformas distintas é apresentar seus pontos fortes e particularidades, aplicando-as em um cenário adequado em conjunto com um banco de dados maduro e estável.

A partir do conhecimento obtido, as plataformas em questão foram exploradas e integradas de forma colaborativa. A necessidade de interação com dispositivos móveis fez com que diversos serviços e tecnologias fossem pesquisadas, culminando em um sistema com capacidade de envio de notificações a aparelhos celulares através da tecnologia SMS.

Palavras-chave: *Java,* .NET, SMS.

ABSTRACT

The lack of Web applications that interact with mobile devices has stimulated the development of this work. In it, some of the key concepts and trends of web development using Java and .NET platforms are presented.

The purpose of using two different platforms is to present their strengths and particularities, applying them in a suitable scenario together with a mature and stable database.

From the knowledge gained, the platforms in question were explored and integrated in a collaborative way. The need for interaction with mobile devices demanded research of several services and technologies, culminating in a system capable of sending notifications to mobile phones through SMS technology.

Keywords: *Java,* .NET, SMS.

LISTA DE FIGURAS

Figura 1 – Estrutura de uma página HTML 5
Figura 2 – Estrutura de uma página HTML com CSS 6
Figura 3 – Estrutura de uma página HTML com *JavaScript* 8
Figura 4 – Funcionamento do CLR 11
Figura 5 – Design de uma página ASP.NET 13
Figura 6 – *Code-behind* de uma página ASP.NET 14
Figura 7 – Histórico de crescimento na utilização das linguagens de programação 16
Figura 8 – Exemplo de compilação de uma classe Java 17
Figura 9 – Funcionamento da JVM 18
Figura 10 – Exemplo de um diagrama de classes 21
Figura 11 – Exemplo de um diagrama de componentes 22
Figura 12 – Exemplo de um diagrama de implantação 23
Figura 13 – Consulta SQL 24
Figura 14 – Modelo de dados *Monarch Communication* 27
Figura 15 – Solution aplicação ASP.NET 29
Figura 16 – Exemplo de utilização do plugin *jQuery Validation* 30
Figura 17 – Diagrama de componentes da aplicação ASP.NET 31
Figura 18 – Formulário de cadastro de usuário 32
Figura 19 – *Datepicker jQuery UI* 32
Figura 20 – Formulário de login 33
Figura 21 – Formulário de cadastro de compromissos 33
Figura 22 – Abas com as informações dos compromissos 34
Figura 23 – web.xml 36
Figura 24 – contexto.xml 37
Figura 25 – servico.xml, com seu arquivo de propriedades 38
Figura 26 – dao.xml, com seus arquivos de propriedades 39
Figura 27 – temporizadorQuartz.xml 40
Figura 28 – Diagrama de classes da aplicação *Java* 41
Figura 29 – Console aplicação *Java* 42
Figura 30 – log4j.properties 42
Figura 31 – Diagrama de implantação 43

LISTA DE ABREVIATURAS E SIGLAS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASP	Active Server Pages
CIL	Common Intermediate Language
CLI	Command-Line Interface
CLR	Common Language Runtime
COM	Component Object Model
CSS	Cascading Style Sheets
DLL	Dynamic-Link Librarie
FTP	File Transfer Protocol
GSM	Global System for Mobile Communications
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IIS	Internet Information Services
IP	Internet Protocol
Java EE	Java Enterprise Edition
Java ME	Java Micro Edition
Java SE	Java Standard Edition
JDBC	Java DataBase Connectivity
JVM	Java Virtual Machine
MMS	Multimedia Messaging Service
POO	Programação Orientada a Objetos
SGBD	Sistema Gerenciador de Banco de Dados
SMS	Short Message Service
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UML	Unified Modeling Language
XML	Extensible Markup Language

SUMÁRIO

1. INTRODUÇÃO 1
1.1. CARACTERIZAÇÃO DO TEMA 1
1.2. OBJETIVOS 2
1.3. JUSTIFICATIVA 2
1.4. PROBLEMA E HIPÓTESE DE PESQUISA 3
1.5. ESTRUTURA DO TRABALHO 3
2. DESENVOLVIMENTO WEB 4
2.1. CONSIDERAÇÕES INICIAIS 4
2.2. HTML 4
2.3. CSS 5
2.4. JAVASCRIPT 7
2.5. PLATAFORMA .NET 9
2.5.1. Linguagem C# 9
2.5.2. CLR 10
2.5.3. ASP.NET 12
2.5.5. Visual Studio IDE 14
2.5.6. IIS 15
2.6. PLATAFORMA JAVA 15
2.6.1. Linguagem Java 15
2.6.2. JVM 17
2.6.3. Spring Framework 18
2.6.4. Quartz Scheduler 19
2.6.5. Eclipse IDE 19
2.6.6. Apache Tomcat 20
2.7. UML 20
2.7.1. Diagrama de classes 20
2.7.2. Diagrama de componentes 21
2.7.3. Diagrama de implantação 22
2.8. POSTGRESQL 23
2.9. SMS 24
2.9.1. Detalhes técnicos 25
2.9.2. Mercado brasileiro 25
2.10. HUMAN GATEWAY 26
2.11. CONSIDERAÇÕES FINAIS 26

3. ESTUDO DE CASO 27
3.1. CONSIDERAÇÕES INICIAIS 27
3.2. MODELO DE DADOS 27
3.3. APLICAÇÃO ASP.NET: INTERFACE VISUAL 28
3.3.1. Ambiente de desenvolvimento 28
3.3.2. Configuração 29
3.3.3. Diagrama de componentes 31
3.3.4. Resultados obtidos 32
3.4. APLICAÇÃO JAVA: TEMPORIZADOR 35
3.4.1. Ambiente de desenvolvimento 35
3.4.2. Configuração 35
3.4.3. Diagrama de classes 41
3.4.4. Resultados obtidos 42
3.5. DIAGRAMA DE IMPLANTAÇÃO 43
3.6. CONSIDERAÇÕES FINAIS 44
4. CONCLUSÃO 45
REFERÊNCIAS BIBLIOGRÁFICAS 46

1. INTRODUÇÃO

1.1. Caracterização do tema

Serviços móveis e aplicações de Internet estão entre os meios de comunicação mais utilizados no mundo, graças às facilidades e grandes avanços essas potências se ratificam cada vez mais no topo da preferência mundial.

O SMS (*Short Message Service*) é um método de comunicação que provê a troca de mensagens de texto entre aparelhos de telefonia móvel. O conceito foi criado no final da década de 1980 pelo engenheiro finlandês *Matti Makkonen* e visava à comunicação com a tecnologia digital GSM (*Global System for Mobile Communications*) de forma simples e funcional mesmo quando os aparelhos estivessem desligados ou fora da área de cobertura (MOBILEPRONTO, 2010).

Apesar de o método ser relativamente antigo, o primeiro SMS foi enviado somente em dezembro de 1992 de um computador para um telefone celular na rede GSM da *Vodafone*[1] no Reino Unido. Mais tarde, em 2008, *Makkonen* recebeu o *Prêmio de Inovação* pela paternidade do invento, outorgado pela revista *The Economist* (MOBILEPRONTO, 2010).

Em outro contexto, a Internet é caracterizada pelo conjunto de redes de computadores interligadas que possuem várias regras e serviços em comum, possibilitando assim que pessoas do mundo inteiro possam se conectar para se comunicar e compartilhar informações. Esse conceito passou vários anos como um projeto experimental de comunicação acadêmica, mas em meados da década de 1990 ela deixou de ser uma instituição dessa natureza e passou a ser explorada e difundida a nível mundial.

Antonioli (2011) afirma que o número de usuários de computador vai dobrar até 2012, chegando a 2 bilhões. A cada dia, 500 mil pessoas entram pela primeira vez na Internet, a cada minuto são disponibilizadas 48 horas de vídeo no *YouTube* e a cada segundo um novo *blog* é criado. Foi concluído na pesquisa que 70% das pessoas consideram a Internet indispensável.

[1] Companhia global de telecomunicações

1.2. Objetivos

Este trabalho tem o objetivo de estudar as novas tendências no desenvolvimento *Web*, apresentando seus padrões e boas práticas de desenvolvimento em conjunto com um banco de dados dedicado.

Prestadores de serviços em envio de mensagens de texto serão contatados para viabilização da integração da tecnologia SMS em aplicações *Web*.

Para a utilização de todos os conceitos, a criação de duas aplicações se faz necessária. A primeira será um site de relacionamentos diferente de qualquer outro, onde o foco principal é o controle de compromissos, ou seja, cada internauta poderá agendar seus compromissos e, quando a data agendada chegar, a outra aplicação notificará todos os relacionados com um SMS informativo.

1.3. Justificativa

MobilePronto (2010) afirma que a comunicação via SMS é mais barata que a maioria dos outros meios, além de ser extremamente simples e estar presente em 100% dos celulares.

Um cenário que caracteriza a capacidade desse meio de comunicação é o continente africano, que segundo Rossi (2011), disponibilizou diferentes formas de transformar um simples torpedo em ferramenta para transferir dinheiro, fazer pagamentos, promover campanhas de saúde pública e mapear conflitos, além de acompanhar a movimentação do mercado, efetuar empréstimos bancários e até guardar dinheiro em poupança, tudo por SMS.

Rossi (2011) ainda cita que oito anos atrás, apenas 5% da população africana possuíam celular. Com a disponibilização desses serviços esse número subiu para 41,4% em 2010 e a previsão é de 66% em 2014.

Apesar do potencial já explorado na África, o SMS é pouco utilizado como serviço em um contexto geral, o que incluem as aplicações *Web*, já que praticamente nenhuma explora esse meio tão promissor.

1.4. Problema e hipótese de pesquisa

A escassez de serviços com capacidade de interação com aparelhos móveis cria uma grande lacuna entre um serviço de Internet e um usuário desconectado, esse problema pode ser sanado com a utilização de um servidor[2] dedicado e um site que execute rotinas independentemente do acesso de seus usuários.

Isso é possível quando essa aplicação *Web* incorpora o conceito *batch*, caracterizado por um conjunto de comandos rodados sequencialmente em um determinado espaço de tempo. Esses comandos no caso deste projeto serão as consultas à base de compromissos e a codificação de envio de SMS.

1.5. Estrutura do Trabalho

O trabalho foi dividido em mais três capítulos além deste para abordagem teórica e prática das tecnologias e recursos envolvidos, a apresentação das referências consultadas no estudo virá posteriormente.

No segundo capítulo, será feita uma abordagem geral de todas as tecnologias utilizadas no projeto, já que todas se englobam no contexto de desenvolvimento *Web*. Serão apresentadas desde elementos básicos para criação de páginas, até integração com serviços externos.

O terceiro capítulo é focado em todo o processo de desenvolvimento do sistema-conceito, chamado *Monarch Communication*, o nome é uma referência à capacidade que as borboletas monarcas têm para saber o momento e lugar certos de sua migração. O projeto pretende atingir o mesmo nível de precisão ao enviar mensagens de texto aos usuários.

A conclusão sobre o trabalho de forma geral será apresentada no quarto e último capítulo.

[2] Gerenciador de aplicações, resumidamente é onde os sistemas *Web* são armazenados quando estão prontos para interação com os usuários.

2. DESENVOLVIMENTO WEB

2.1. Considerações iniciais

Existem inúmeras maneiras de desenvolver aplicações *Web*, nesse capítulo todas as tecnologias utilizadas no projeto serão apresentadas. Cada uma delas é fundamental no contexto geral do sistema, por isso a introdução de cada conceito é importante.

Neste capítulo os conceitos de HTML, CSS e JavaScript foram revisados em conjunto com padrões contemporâneos de desenvolvimento *Web*.

2.2. HTML

O HTML (*HyperText Markup Language*) foi criado em 1990 por um cientista chamado *Tim Berners-Lee*. Sua ideia inicial era tornar possível o acesso e compartilhamento de pesquisas entre cientistas de diferentes universidades. O projeto inicial tornou-se um sucesso e fortaleceu as fundações da Internet que conhecemos hoje (HTML.NET, 2011).

Essa tecnologia nada mais é que uma linguagem de marcação, que por sua vez é estruturada através de *tags*, que informam ao navegador como o *site* será exibido.

As *tags* começam com um sinal de menor “<” e terminam com um sinal de maior “>”. Genericamente falando, existem dois tipos de *tags*: de abertura “<comando>” e de fechamento “</comando>”. A primeira é utilizada para dar início a um elemento e a outra determina o limite de atuação do mesmo, portanto tudo nesse intervalo será processado segundo o comando contido na *tag*.

Em algumas *tags* a abertura e o fechamento ocorrem na mesma *tag*. Elas contêm comandos que não necessitam de um conteúdo para serem processados, isto é, são *tags* de comandos isolados, por exemplo, um pulo de linha é conseguido com a *tag*
.

Para uma melhor assimilação da linguagem a ferramenta *Tryit Editor* da *W3C*, empresa especializada em padrões *Web*, foi utilizada para a visualização da codificação juntamente com o resultado obtido.

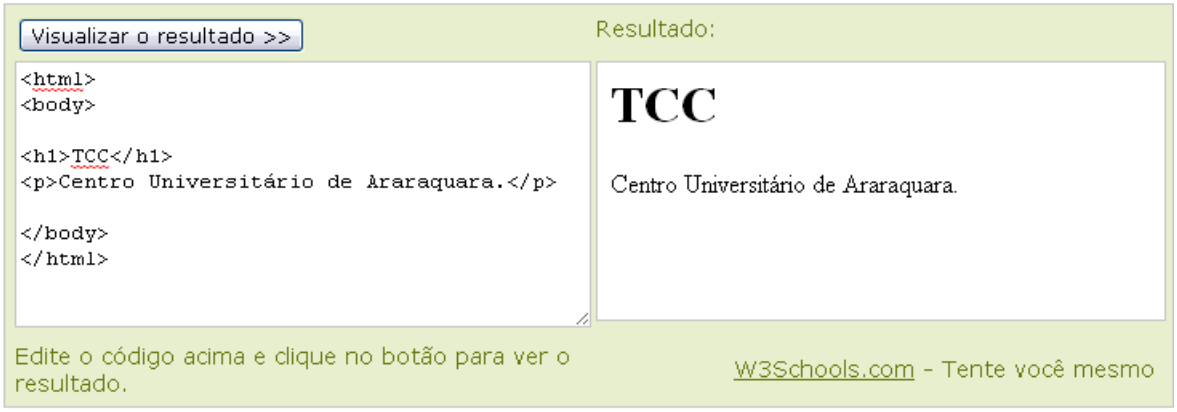

Figura 1 – Estrutura de uma página HTML

Fonte: http://www.w3schools.com/html/tryit.asp?filename=tryhtml_formatting

As tags <html> e </html> definem que o arquivo será interpretado como uma página *Web*.

O corpo dessa página é representado por tudo entre as tags <body> e </body>.

<h1>TCC</h1> indica que as letras TCC serão apresentadas com o estilo de cabeçalho, isso pode ser comprovado à direita da Figura 1.

O texto dentro das tags e será exibido como um parágrafo, que também é exibido à direita da ilustração.

Segundo HTML.net (2011), o HTML foi criado a mais de duas décadas e apesar do passar dos anos ele continua evoluindo, recentemente a versão cinco foi lançada, o objetivo é fazer com que o navegador deixe de ser apenas o aplicativo que exibe as páginas e torne-se um ambiente de execução de aplicações. Essa versão evidencia o armazenamento *offline*[3] de arquivos, modificações de imagens sem a necessidade de editores específicos e elementos multimídia que dispensam *plugins*[4] para execução de áudio e vídeo.

2.3. CSS

HTML.net (2011) indica que o CSS (*Cascading Style Sheets*) é uma linguagem para estilos que define o *layout* [5] de documentos HTML, resumidamente essa linguagem controla fontes, cores, margens, linhas, alturas, larguras, imagens de fundo, posicionamentos e tantas outras propriedades de elementos HTML.

[3] Termo utilizado para definir quando o usuário está desconectado de uma rede, por exemplo, a Internet.
[4] Usado para adicionar funções a programas maiores, provendo alguma funcionalidade especial ou muito específica.
[5] Significa aparência, esse termo é utilizado para se referir ao apelo visual de uma página *Web*.

Esses estilos podem ser aplicados de forma direta a elementos de uma página, esse comportamento pode ser observado na Figura 2. Também existem classes e outros tipos de seletores[6] que podem tornar a customização de uma página muito mais sofisticada e elegante, esse conceito será apresentado junto ao estudo de caso.

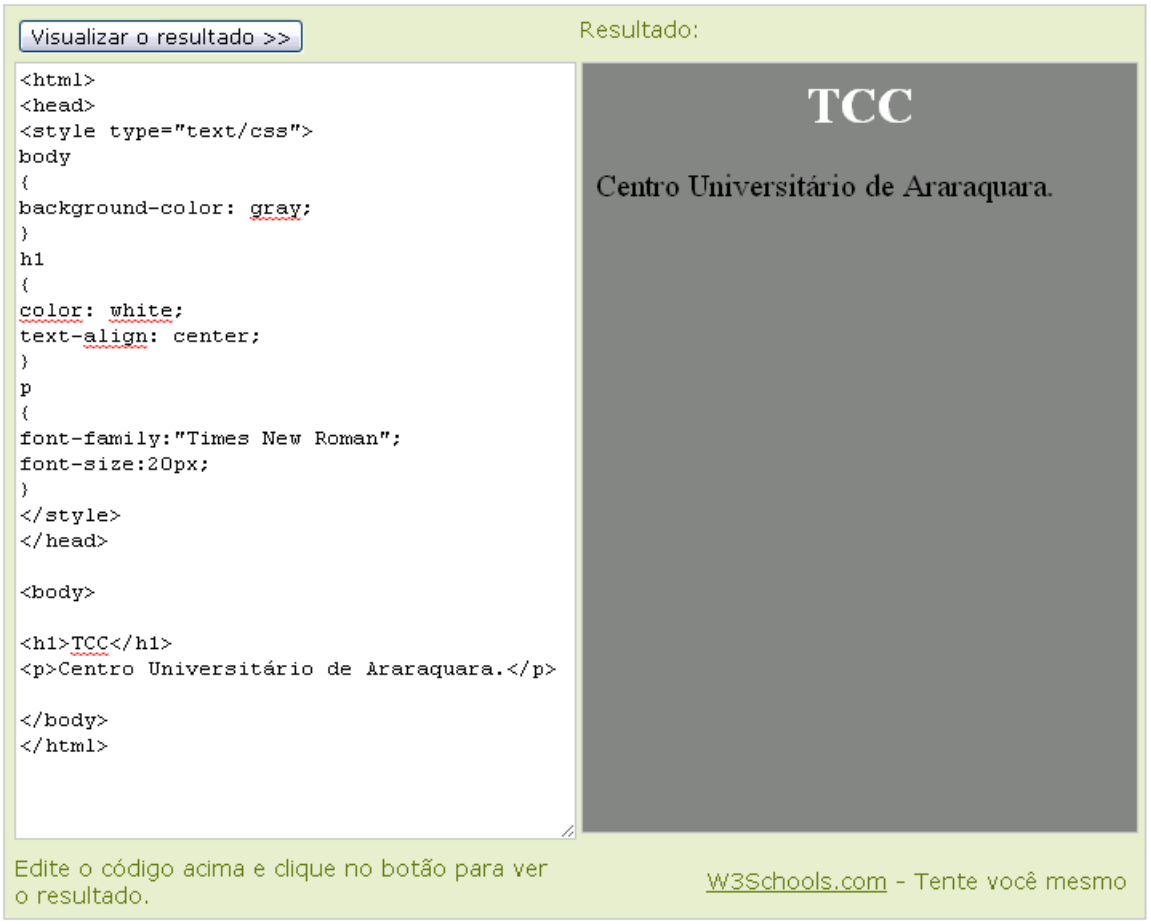

Figura 2 – Estrutura de uma página HTML com CSS

Fonte: http://www.w3schools.com/html/tryit.asp?filename=tryhtml_formatting

O conteúdo entre as tags <style> e </style> definem os estilos aplicados na página exibida no canto direito da Figura 2.

O seletor *body* aplica a cor *gray* ao fundo da página, o que define esse estilo é a propriedade *background-color* com a constante da cor cinza.

No seletor seguinte a *tag* <h1> tem sua cor e alinhamento customizados. A propriedade *color* é vinculada a constante *white* e a constante *center* é atribuída a propriedade *text-align*.

Por fim a *tag* <p> têm o seu tipo de fonte e seu tamanho modificado pelas propriedades *font-family* e *font-size* respectivamente.

[6] Identifica um elemento *HTML* ou define uma classe ou pseudo classe na qual a regra de estilo será aplicada (jQueryMagazine, 2011).

Assim como o HTML, o CSS também vem evoluindo, só que a passos mais lentos, sua última versão, a 3.0 foi lançada no ano de 2001. Grande parte desse tempo sem avanços se deve a falta de suporte a essa tecnologia por parte dos navegadores da época. Hoje as novas versões dos principais browsers do mercado suportam CSS 3.0.

2.4. JavaScript

Segundo Mozilla (2011), o *JavaScript* foi criado pela *Netscape* em 1995, na época tinha o nome de *LiveScript*. Com o sucesso no mercado, recebeu colaboração da *Sun Microsystems*, criadora da linguagem de programação *Java*. Essa parceria gerou alguns frutos, como a semelhança de sintaxe entre as duas linguagens que permite ao desenvolvedor não ter que aprender novos conceitos antes de iniciar a implementação.

Enquadra-se tanto no paradigma estrutural quanto no orientado a objetos. Por se tratar de uma linguagem processada no cliente[7], é muito usada e indicada para fazer validações de formulários antes de submetê-los ao servidor e também para criar interações entre os componentes da página.

É fracamente tipada, dinâmica e implícita, ou seja, as variáveis não têm um tipo definido e durante a execução podem receber múltiplos valores. Sua interatividade com elementos de página permite que características, como atributos e estilos, sejam alterados dinamicamente.

O *JavaScript* coloca o HTML e o CSS como duas das três peças da construção de uma página *Web*. O HTML fornece a estrutura, o CSS adiciona o estilo e o *JavaScript* inicia e faz as coisas acontecerem. Para encontrar o “caminho” para uma página *Web* interativa, você tem que seguir a “trilha” da estrutura (HTML) até o estilo (CSS) e então, a ação (*JavaScript*). Assim como o CSS, o código *JavaScript* geralmente reside na página *Web* (MORRISON, 2008).

[7] Pode ser definido como o usuário, ele tem a iniciativa de começar a comunicação quando deseja algum serviço.

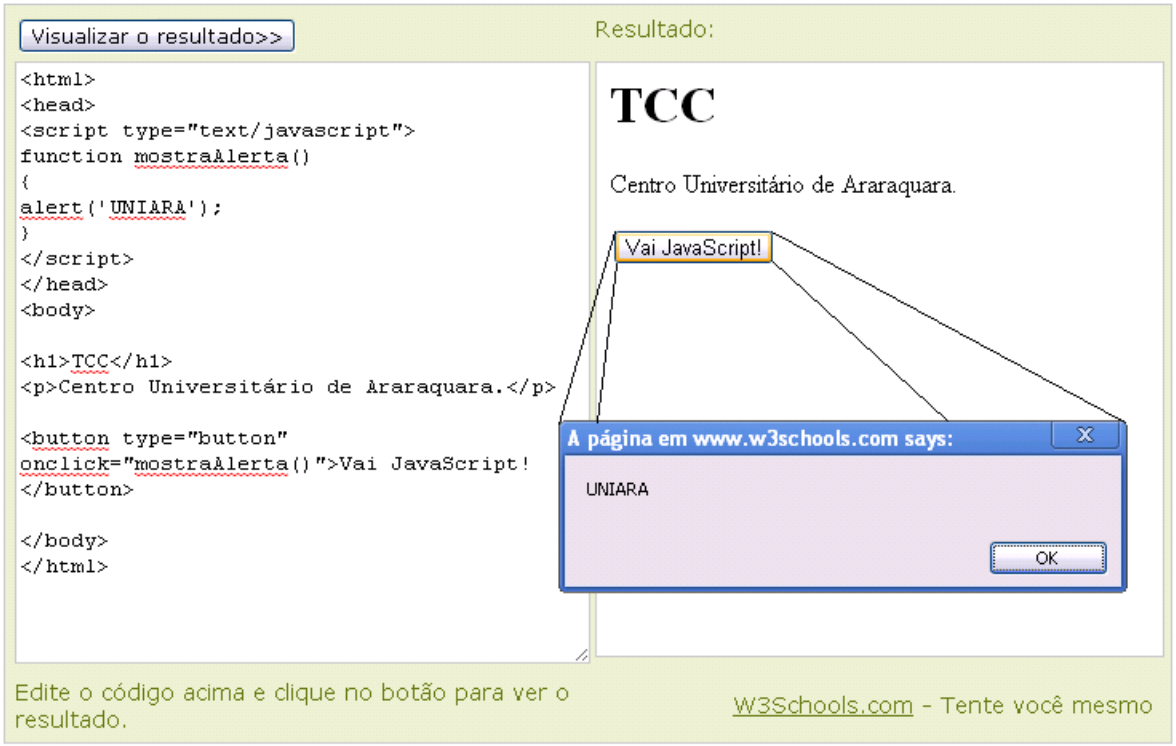

Figura 3 – Estrutura de uma página HTML com *JavaScript*

Fonte: http://www.w3schools.com/html/tryit.asp?filename=tryhtml_formatting

O exemplo acima mostra uma função *JavaScript* definida no corpo da tag <script>, ela está simplesmente lançando um alerta ao usuário. O que faz com que essa função seja chamada é a associação a um evento, nesse caso o *onclick*, que será executado quando o usuário clicar no botão, definido pela tag <input> do tipo *button*.

Frameworks[8] disponíveis no mercado, como o *jQuery*, utilizam ao máximo o *JavaScript* e suas ramificações para criação de aplicações dinâmicas com um visual moderno e elegante em menos tempo.

jQuery é uma biblioteca *JavaScript* que de maneira rápida e concisa simplifica a busca em documentos HTML, manipulação de eventos, animação e interações AJAX (*Asynchronous JavaScript and* XML), culminando em um desenvolvimento web rápido. *jQuery* é projetado para mudar a maneira que você escreve *JavaScript* (JQUERY, 2011).

[8] São abstrações que englobam códigos comuns entre vários projetos de software provendo funcionalidades genéricas.

2.5. Plataforma .NET

Araújo (2006) afirma que a plataforma *Microsoft .NET* tem como objetivo ser uma plataforma única para desenvolvimento e execução de aplicações *Desktop*, *Web* e *Web Services*[9]. Qualquer código gerado para *.NET* pode ser executado em qualquer dispositivo que possua o *.NET Framework*, que de acordo com Thai e Lam (2003) é uma estrutura de desenvolvimento em tempo de execução para a construção de aplicações na plataforma *Windows*.

2.5.1. Linguagem C#

Segundo Castro (2006) o C# (Pronuncia-se “cê sharp”) é uma linguagem de programação criada por *Anders Hejlsberg*, *"Distinguished Engineer"* da *Microsoft*, como parte das ferramentas de desenvolvimento criadas pela empresa. *Anders* trabalhou na *Borland*, onde criou o *Turbo Pascal* e o *Delphi*. Em 1997 foi contratado, junto com sua equipe, pela Microsoft para trabalhar no time que desenvolvia o que viria a ser chamada de *.NET Framework.*

C# é uma evolução das linguagens C e C++, essa linguagem foi criada do zero baseada nas citadas, portanto possui uma base sólida, porém com maior facilidade no aprendizado, pois a Microsoft retirou algumas características mais complexas como ponteiros, mantendo os pontos fortes do C e C++ e eliminando os pontos fracos que tornavam difícil a vida do desenvolvedor (FERGUSON, 2002).

Inicialmente pensava-se que a utilização do símbolo “**#**” (cerquilha) no nome viria da influência do C++, utilizando quatros sinais “+” (C++++) sobrepostos. No entanto o uso da cerquilha é uma alusão ao símbolo musical “♯” (sustenido) que indica o aumento em meio tom de uma nota musical. Como existe uma limitação quanto ao uso de alguns caracteres em algumas fontes e layouts de teclados, o sustenido foi trocado pela cerquilha, facilitando seu uso em diversos meios.

Além das linguagens citadas por Ferguson (2002), Castro (2006) indica que o C# ainda teve influencia de linguagens como *Smalltalk*, bem como *Pascal*, *Delphi* e *Java*, linguagens com as quais *Anders* teve contato antes de trabalhar na *Microsoft*.

Varela (2005) cita as principais características do C*#:*

[9] Soluções utilizadas na integração de sistemas e na comunicação entre aplicações diferentes.

- Por ter sido criada do zero para funcionar especialmente na plataforma *.NET*, a linguagem não tem a necessidade de compatibilidade com código já existente.
- A maior parte das classes do *.NET Framework* foram desenvolvidas em C*#,* até porque o primeiro compilador da plataforma era para essa linguagem.
- Primeira linguagem "orientada a componentes" da família C/C++.
- Segue os padrões de POO (Programação Orientada a Objetos), onde tudo deriva de um ancestral comum, no caso *System.Object*, não existem atributos e métodos autosuficientes, tudo é declarado dentro do escopo da classe, contudo é possível declarar tipos *struct* e *enum* fora do escopo de classes.
- É fortemente tipada, o que ajuda a evitar erros oriundos de uma manipulação imprópria de tipo ou atribuições.
- Suporte à COM (*Component Object Model*), COM+ ou outras DLLs (*Dynamic-Link Libraries*) escritas por linguagens que geram código não gerenciado.
- É *case-sensitive*, ou seja, faz diferenciação entre maiúsculas e minúsculas, por exemplo, tcc é completamente diferente de TCC ou TcC.
- Os programas escritos em C# rodam sob um ambiente gerenciável, o que significa que todo o controle de memória é feito pelo *.NET Framework* e não diretamente pelo programador, reduzindo assim falhas na programação enquanto a alocação e liberação de um objeto na memória.

2.5.2. CLR

Na decisão por qual linguagem utilizar em uma aplicação, diversos pontos são levados em consideração como gerenciamento de recursos, domínio da sintaxe, segurança e tratamento de exceções. Para diminuir as diferenças entre as linguagens, a Microsoft criou seu pacote de desenvolvimento sobre um único ambiente de execução, o CLR (*Common Language Runtime*). Com isso, os recursos das linguagens do *.NET Framework* diferenciam-se basicamente pela sintaxe do código, pois o núcleo de seu funcionamento está no CLR.

Com base nos argumentos de Richter (2010), podemos afirmar que quando um código de uma determinada linguagem é compilado, o compilador transforma esse código em uma linguagem intermediária, a CIL (*Common Intermediate Language*) e *metadados*[10]. Dessa forma, o CLR executa a conversão da CIL em linguagem de

[10] São informações úteis para identificar, localizar, compreender e gerenciar os dados.

máquina e os metadados contêm as informações sobre os tipos de dados e os membros definidos no código fonte, ou em bibliotecas referenciadas pelo mesmo.

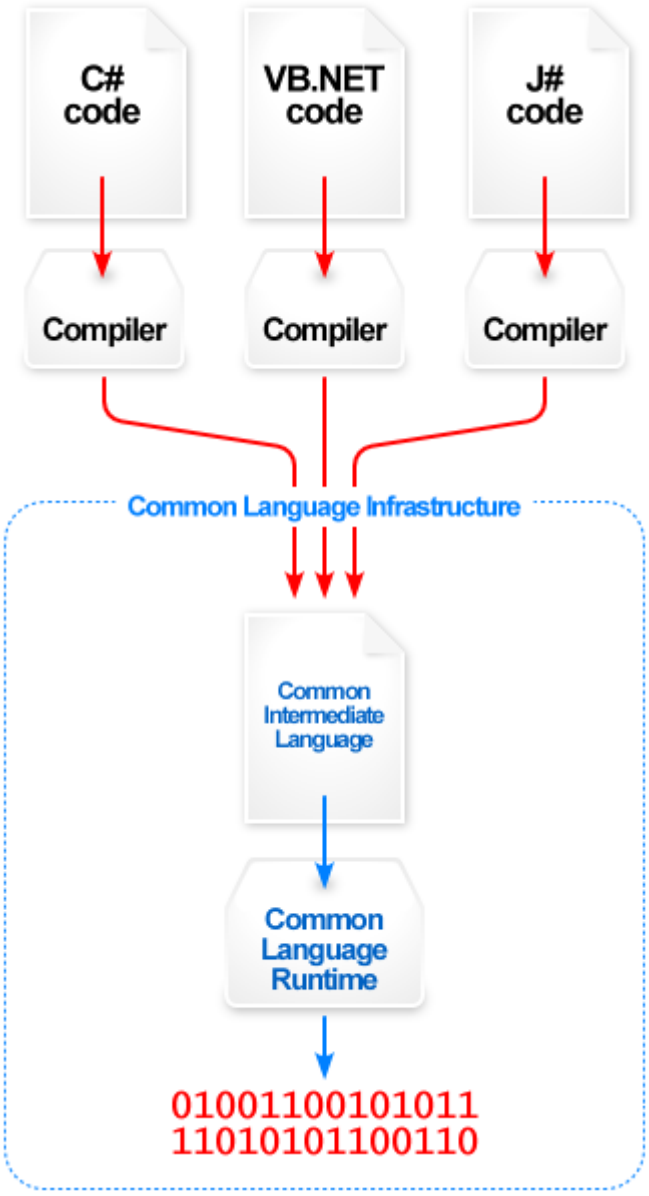

Figura 4 – Funcionamento do CLR

Fonte: http://blog.tangcs.com/2008/01/20/dotnet-clr-overview/

Neste modelo de execução, toda linguagem que se utilizar do CLR deve apenas garantir que o programador escreveu o código de forma correta, dentro da sintaxe que o compilador irá interpretar para gerar a CIL. Toda a execução fica dentro do CLR.

A Microsoft utiliza o CLR nas linguagens C++/CLI (*Command-Line Interface*), C#, VB.NET, F#, *Iron Phyton, Iron Ruby* e a própria CIL. Outras empresas e até universidades criaram compiladores utilizando o CLR para suas linguagens, tais como COBOL, PHP, LUA, LISP, FORTRAN, entre outras.

2.5.3. ASP.NET

O ASP.NET é a evolução do ASP (*Active Server Pages*), tecnologia utilizada anteriormente pela Microsoft para criar aplicações *Web*. Com ele as páginas ficaram mais organizadas e fáceis de entender, já que cliente e servidor podem ser separados, o que não era possível no ASP, que acessa elementos do servidor pela própria página com auxílio das tags diretivas.

Aplicações em ASP.NET podem ser desenvolvidas utilizando qualquer uma das dezenas de linguagens suportadas pelo *.NET Framework*. Os códigos das páginas são compilados automaticamente, ou seja, quando a página é requisitada pela primeira vez ao servidor, ele compila a mesma para dentro de um arquivo *assembly* temporário e armazena este arquivo como uma cópia da página, nas próximas requisições o servidor utilizará a cópia do resultado que foi armazenado, proporcionando assim um aumento no desempenho (SHEPHERD, 2005).

Lotar (2007) diz que para construir as interfaces o ASP.NET utiliza o recurso *server controls* que são controles, componentes reutilizáveis, eles possuem diversas funcionalidades, comportamentos pré-estabelecidos, propriedades, métodos e eventos que podem ser utilizados para o desenvolvimento do código para ser acessado no servidor. Existem recursos que são similares aos elementos HTML, como caixa de texto e botões, mas também há recursos complexos como componentes para realizar conexão com o banco de dados, o diferencial é que todos os elementos podem ser programados como objetos, já que a tecnologia é orientada a objetos.

Na prática é bem simples, uma página tem duas partes fundamentais: o *design*, que é a parte visual da página com todos os seus controles e o *code-behind*, que é onde toda a requisição que necessite de uma interação do servidor será manipulada. Para uma melhor visualização desses fundamentos a visão *Split* do IDE (*Integrated Development Environment*) *Visual Studio 2010* foi utilizada nas Figuras 5 e 6.

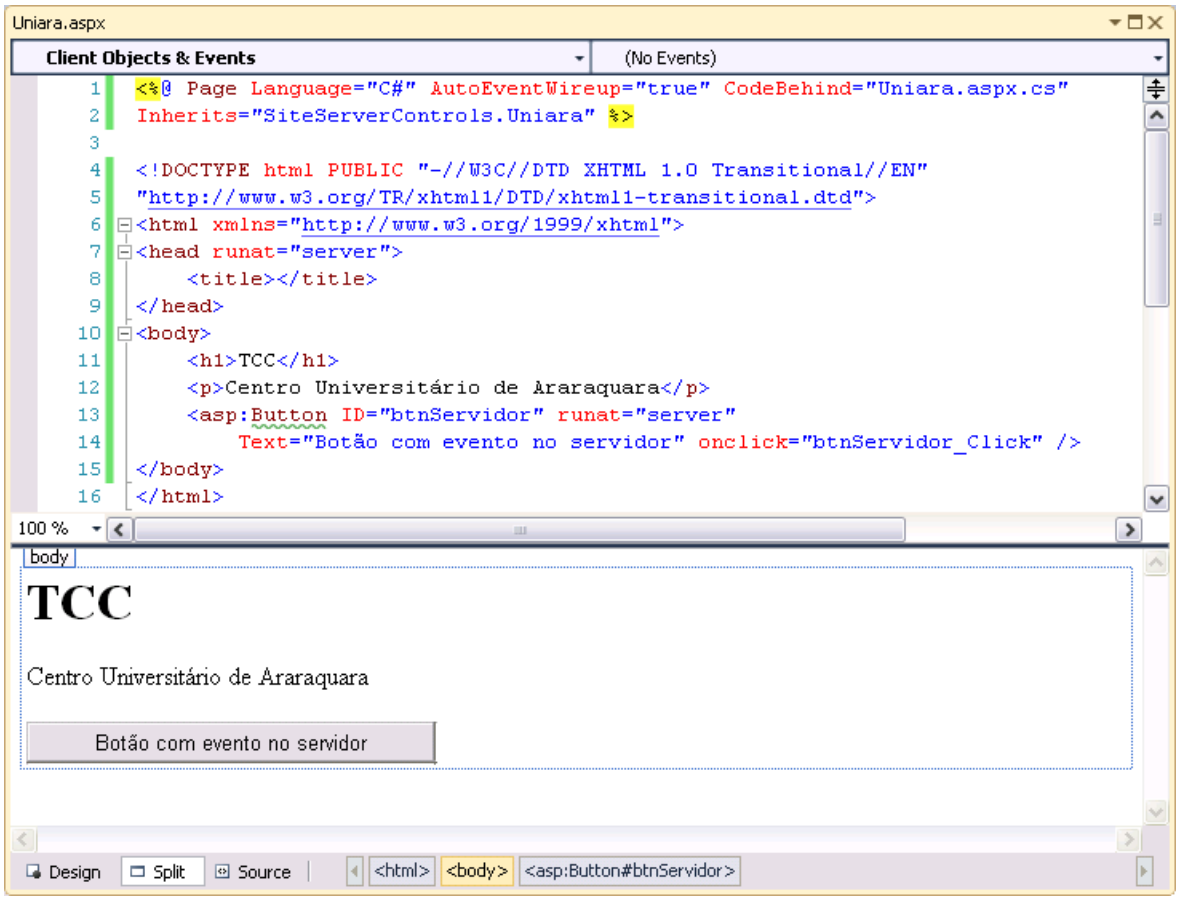

Figura 5 – Design de uma página ASP.NET

Fonte: Elaboração própria

Algumas notações padrões podem assustar, mas é bem mais fácil que parece. Além dos elementos que já conhecemos essa página tem a diretiva <%@ Page %>, que define algumas configurações suportadas de uma página .aspx[11].

A *tag* <!DOCTYPE> indica o padrão adotado no documento, nesse caso a página segue o padrão *Transitional* da versão 4.0 do HTML. Essa versão possui três padrões possíveis:

- *Strict:* a utilização de elementos em desuso é proibida (Padrão internacional).
- *Transitional:* a utilização de elementos em desuso é permitida.
- *Frameset:* é utilizado em documentos que utilizam frames.

A *tag* <asp:Button> representa um *server control* equivalente a um <input> do tipo *button*, esse botão tem um evento associado ao seu *click*[12], esse por sua vez será executado no servidor. O botão pode ser visualizado na parte inferior da Figura 5.

[11] Extensão de páginas *Web* da tecnologia ASP.NET.
[12] É definido quando o usuário clicar com o botão esquerdo do mouse no botão.

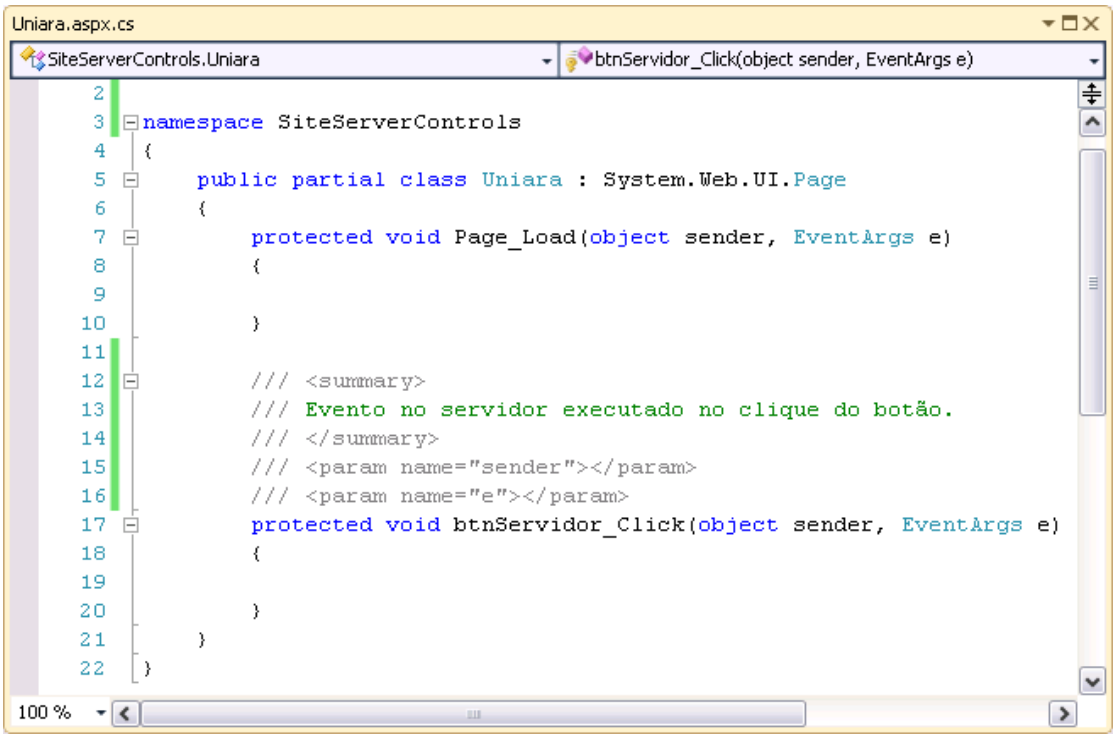

Figura 6 – *Code-behind* de uma página ASP.NET

Fonte: Elaboração própria

O evento associado ao botão deve ser implementado no *code-behind* que nada mais é que a classe, nesse caso escrita na linguagem C#, que controla a página *Web* em questão. A codificação no servidor maximiza as possibilidades de construção do programador.

O ASP.NET vem evoluindo juntamente com o *.NET Framework*, que dispõem da versão 4.0. Várias inovações foram desenvolvidas nesse segmento, fazendo dessa versão o padrão de desenvolvimento *Web* da *Microsoft.*

2.5.5. Visual Studio IDE

A ferramenta *Visual Studio* é um poderoso IDE que fornece recursos que simplificam todo o processo de desenvolvimento, desde o design até a implantação. Com ele é possível criar aplicações *Web* ricas em pouquíssimo tempo (MICROSOFT, 2011).

A exemplo do ASP.NET o *Visual Studio* também tem o *.NET Framework* como base para a disponibilização de seus recursos.

O *Visual Studio* é muito possivelmente o IDE de desenvolvimento de software mais poderoso e abrangente disponível no mercado. Não importa se a disciplina é de arquitetura, desenvolvimento ou testes, ele fornece as ferramentas necessárias para a otimização de cada uma delas (NABOULSI e FORD, 2011).

2.5.6. IIS

O IIS (*Internet Information Services*) é o servidor de aplicação padrão de aplicações desenvolvidas em tecnologias *Microsoft*. Segundo TechNet (2009) ele ajuda organizações e pessoas a atender às suas necessidades comerciais prestando os serviços que oferecem suporte a um servidor *Web* seguro, disponível e escalável no qual são executados os sites e aplicativos.

Apesar de ser indispensável na implantação de um sistema *Microsoft*, é possível que programadores que utilizem o IDE *Visual Studio* desconheçam as tarefas para configuração do servidor, isso se deve ao fato do *Visual Studio* ter um servidor de aplicação nativo que executa os projetos de forma local.

2.6. Plataforma Java

Segundo Oracle (2010) o *Java* além de ser uma linguagem de programação é também uma plataforma computacional.

Essa plataforma poder ser enumeradas em três sub-plataformas principais:

- *Java SE* (*Java Standard Edition*) – Prove a base da plataforma. Inclui o ambiente de execução e as bibliotecas mais comuns.
- *Java EE* (*Java Enterprise Edition*) – Voltada para o desenvolvimento de aplicações corporativas e para Internet.
- *Java ME* (*Java Micro Edition*) – Desenvolvimento de aplicações para dispositivos móveis e embarcados.

2.6.1. Linguagem Java

Em 1991, a *Sun Microsystems* elaborou o *Green Project*, projeto considerado pai da linguagem *Java*. O projeto foi liderado pelos programadores Patrick Naughton, Mike Sheridan, e James Gosling. O propósito deste projeto não era desenvolver uma nova linguagem de programação, mas criar um sistema dinâmico que fosse suportado tanto por um computador quanto por equipamentos e eletrodomésticos (HOMEHOST, 2011).

Homehost (2011) afirma que um ano depois foi apresentado o protótipo **7 (StarSeven)*, um controle remoto com interface gráfica *Touchscreen*. Juntamente com o protótipo, Gosling escreveu e especificou uma nova linguagem de programação denominada OaK, que em 1995 foi adaptada para a Internet e rebatizada como Java.

Desde seu lançamento, a linguagem *Java* vem crescendo e atualmente representa um padrão para o mercado, pois oferece qualidade, performance e

segurança. Com pouco mais de dez anos de existência, foi à linguagem de programação adotada mais rapidamente em toda a história da computação, como podemos observar na Figura 7:

Programming Language	Position Dec 2011	Position Dec 2006	Position Dec 1996	Position Dec 1986
Java	1	1	5	-
C	2	2	1	1
C++	3	3	2	7
C#	4	8	-	-
Objective-C	5	42	-	-
PHP	6	5	-	-
(Visual) Basic	7	4	3	5
Python	8	7	26	-
Perl	9	6	6	-
JavaScript	10	10	25	-
Lisp	13	17	16	2
Ada	17	16	12	3

Figura 7 – Histórico de crescimento na utilização das linguagens de programação
Fonte: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Em meio às características que tornaram esses números possíveis JavaFree.org (2005) destacou:

- Orientação a objeto – Baseado nos modelos das linguagens de programação *Smalltalk* e *Simula67*.
- Fortemente tipada.
- Portabilidade – Independência de plataforma.
- Recursos de Rede – Possui extensa biblioteca de rotinas que facilitam a cooperação com protocolos TCP (*Transmission Control Protocol*) / IP (*Internet Protocol*), como HTTP[13] (*Hypertext Transfer Protocol*) e FTP (*File Transfer Protocol*).
- Segurança – Pode executar programas via rede com restrições de execução;
- Facilidade para criação de programas distribuídos e multitarefa (múltiplas linhas de execução num mesmo programa).

[13] Protocolo de comunicação utilizado para sistemas de informação de hipermídia distribuídos e colaborativos.

- Carga Dinâmica de Código – Programas em *Java* são formados por uma coleção de classes armazenadas independentemente e que podem ser carregadas no momento de utilização.

A Oracle (2011) destacou a liderança do *Java* segundo TIOBE (2011) e apresentou mais os seguintes dados sobre a plataforma:

- 97% das aplicações empresariais rodam *Java*.
- 1 bilhão de downloads do *Java* são realizados por ano.
- 9 milhões de programadores em todo o mundo.
- Mais de 3 bilhões de dispositivos são movidos pela tecnologia *Java*.

2.6.2. JVM

Segundo JavaFree.org (2005) quando uma classe *Java* é compilada, é gerado um arquivo de extensão *class*. Um *.class* não contém código *Java* ou de máquina, mas sim uma linguagem intermediária denominada *bytecode* que é interpretada pela JVM (*Java Virtual Machine*). Esse processo pode ser visualizado na Figura 8:

Figura 8 – Exemplo de compilação de uma classe Java
Fonte: http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html

A JVM traduz em tempo de execução os *bytecodes* em instruções nativas do processador. Isto permite que uma mesma aplicação seja executada em qualquer plataforma computacional que possua uma máquina virtual devidamente configurada. Esse comportamento pode ser visualizado na Figura 9.

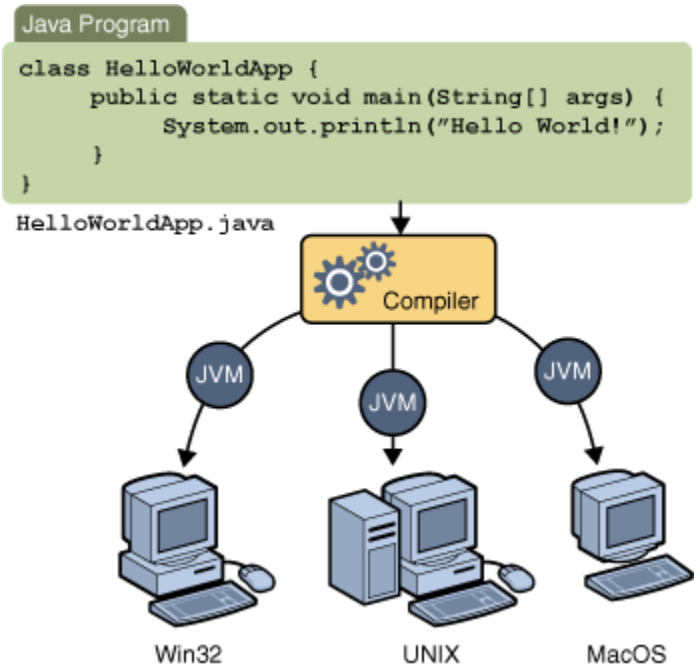

Figura 9 – Funcionamento da JVM

Fonte: http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html

Desde sua primeira versão, esse ambiente de execução vem equipado com gerenciamento automático de memória, realizado por um algoritmo coletor de lixo, o *Garbage Collector*, que liberta o programador das tarefas de alocação e libertação de memória, fonte de muitos erros de programação em linguagens anteriores.

2.6.3. Spring Framework

O Spring é um *framework* de código aberto desenvolvido na plataforma *Java EE* baseado no código publicado em *Expert One-on-One J2EE Design and Development* por Rod Johnson (WROX, 2002).

Atualmente, o Spring é uma divisão da companhia *VMWare* e é distribuída sobre os termos da licença *Apache License, Version 2.0.*

O conceito de *Inversion of Control* (Inversão de Controle) é o que torna uma simples biblioteca de classes diferente de um *framework*. Uma biblioteca consiste em um conjunto de classes em que o usuário instancia e utiliza seus métodos. Entretanto, em um *framework* o cenário é diferente. Para utilizá-lo, o código da aplicação deve ser criado e se manter acessível por ele. O *framework*, então, realiza a chamada ao código da aplicação, ato que justifica o nome do padrão.

O *Spring* disponibiliza o padrão *Dependency Injection* (Injeção de Dependência), que se trata de uma especialização do padrão de inversão de controle, além de mais uma grande diversidade de outros padrões de projeto.

A injeção de dependência providencia meios consistentes de configuração e manuseio de *beans*[14]. O *container* é responsável por gerenciar o ciclo de vida dos *beans* de maneira a criar novos objetos, invocar métodos de inicialização e criar referências entre eles.

Em resumo, a injeção de dependência indica que uma classe não deve criar suas dependências, ou seja, seus atributos não devem ser instanciados internamente. Ao invés disso, a responsabilidade desta tarefa é atribuída ao *Spring*.

2.6.4. Quartz Scheduler

Mantido pela *Terracotta*, subdivisão da empresa *Software AG*, o *Quartz Scheduler* é um serviço de agendamento de tarefas que pode ser integrado com qualquer aplicação *Java EE* ou *Java SE*, desde as aplicações em pequena escala até sistemas de grande porte. Seu código é aberto e é distribuído gratuitamente como *framework* sobre os termos da licença *Apache License, Version 2.0*.

Nele é possível configurar rotinas de execução definidas em classes *Java*, as quais podem ser executadas em ambiente *multithread*[15], disponibilizado pelo *framework.*

2.6.5. Eclipse IDE

O *Eclipse IDE* é um ambiente de desenvolvimento criado em 2001 e disponibilizado à plataforma *Java* de forma gratuita e *open source*. No início do projeto *Eclipse* a *IBM* fez uma contribuição de 40 milhões dólares, desde então o software se firmou como um dos principais IDEs do mercado e hoje tem valor estimado em mais de 800 milhões de dólares (ECLIPSE, 2011).

Apesar da ideia inicial se apoiar apenas à plataforma *Java,* hoje existem diversas distribuições paralelas que permitem a utilização do IDE em linguagens como C/C++, PHP, *Ruby*, entre outras. Essa diversidade é reflexo da disponibilização *open souce*, que permite aos membros da comunidade abrir o código fonte e criar plugins e até mesmo versões customizadas.

[14] São objetos *Java* cujas propriedades são acessadas pelos métodos assessores.
[15] Ambiente ou sistema que suporta múltiplas threads, ou seja, múltiplas linhas de execução.

2.6.6. Apache Tomcat

O *Tomcat* é um servidor de aplicação *Web* voltado à plataforma *Java EE* desenvolvido pela *Apache Software Foundation.* Segundo Apache (2011) o *Tomcat* é uma implementação *open source* dos conceitos *Java Servlet* e *JavaServer Pages*, ambos mantidos pela comunidade de desenvolvimento *Java*.

2.7. UML

A UML (*Unified Modeling Language*) é utilizada para especificar, construir e documentar os artefatos de um sistema, oferecendo uma forma padronizada de especificação, envolvendo elementos conceituais, tais como processos e funções do sistema, como uma classe de uma linguagem de programação, esquema de banco de dados e componentes de software reutilizáveis (RUMBAUGH, JACOBSON e BOOCK, 1999).

2.7.1. Diagrama de classes

Segundo Rumbaugh, Jacobson e Boock (1999) um diagrama de classes é usado para descrever os tipos de objetos de um sistema e os vários tipos de relacionamentos existentes entre eles, bem como atributos e operações de uma classe e suas restrições.

A Figura 10 mostra um diagrama de classes de uma aplicação que simula uma bilheteria. Este diagrama pretende especificar um modelo de venda de bilhetes com suas classes mais relevantes (RUMBAUGH, JACOBSON e BOOCK, 1999).

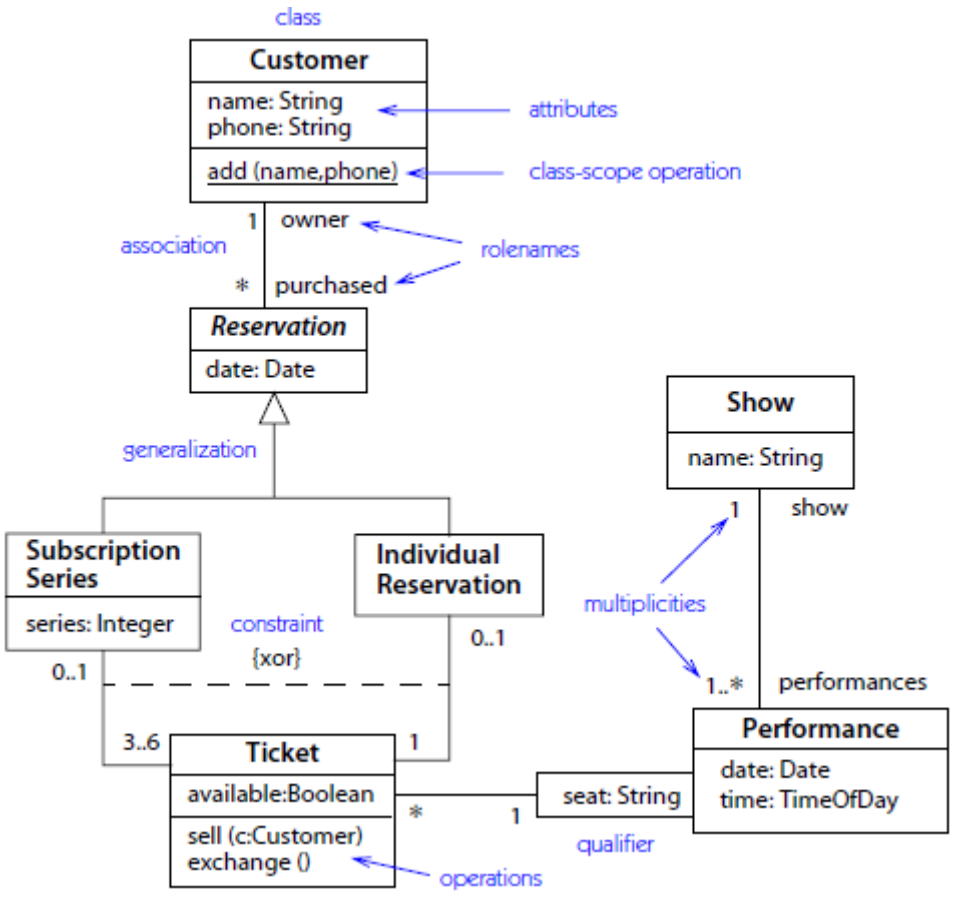

Figura 10 – Exemplo de um diagrama de classes

Fonte: The Unified Modeling Language Reference Manual (RUMBAUGH, JACOBSON e BOOCK, 1999)

2.7.2. Diagrama de componentes

Rumbaugh, Jacobson e Boock (1999) indicam que um diagrama de componentes define os módulos físicos de um software e suas relações. O software se torna um conjunto de unidades modulares e intercambiáveis que compõem um conjunto maior, com as mesmas características.

Componentes podem representar uma classe, página, aplicação, sistema ou subsistema.

A visão da implementação é apresentada em um diagrama de componente. A Figura 11 apresenta o diagrama de componentes do sistema de bilheteria exemplificado anteriormente (RUMBAUGH, JACOBSON e BOOCK, 1999).

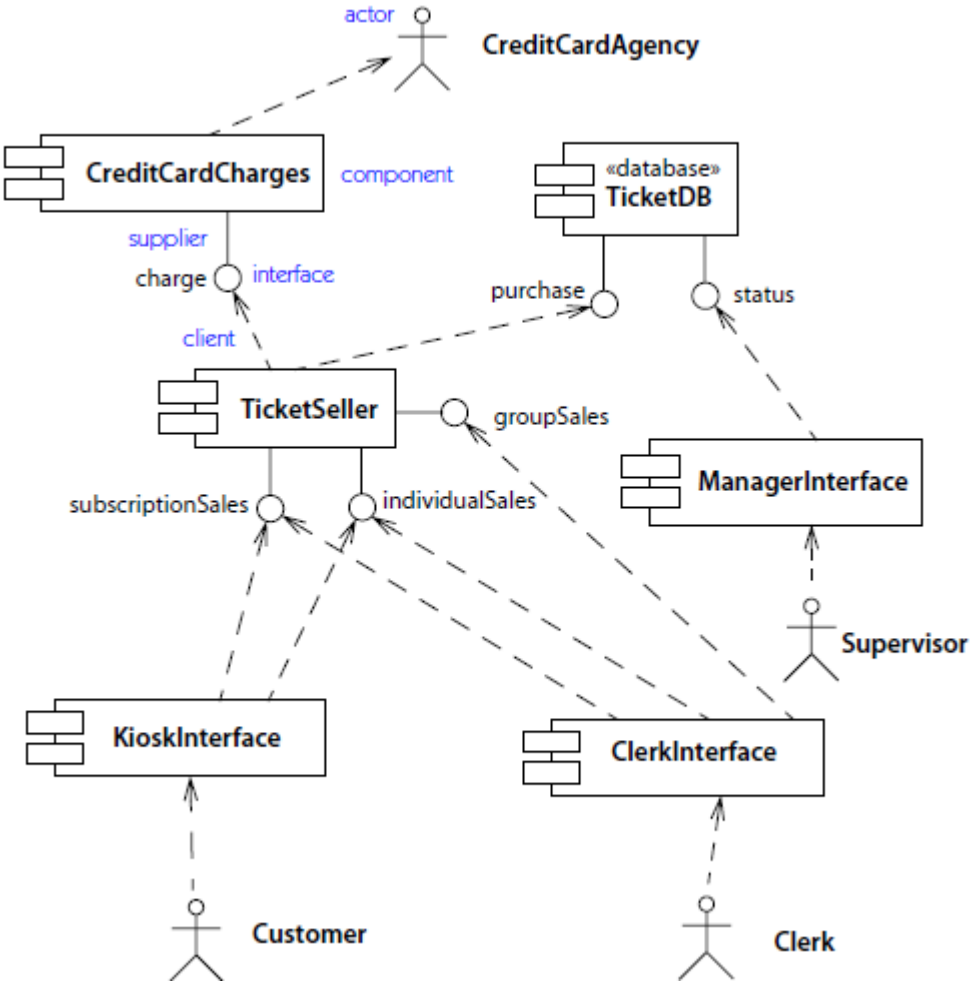

Figura 11 – Exemplo de um diagrama de componentes

Fonte: The Unified Modeling Language Reference Manual (RUMBAUGH, JACOBSON e BOOCK, 1999)

2.7.3. Diagrama de implantação

A partir dos conceitos de Rumbaugh, Jacobson e Boock (1999) pode-se afirmar que o diagrama de implantação foca a questão da organização da arquitetura física sobre a qual um software será implantado e executado em termos de hardware, além de definir como estas máquinas estarão conectadas e através de quais protocolos se comunicarão e transmitirão os dados.

O diagrama de componentes mostra os tipos de componentes do sistema, onde uma configuração particular da aplicação pode ter mais de uma cópia de um componente, como podemos visualizar na Figura 12, que mostra o diagrama de componentes do sistema de bilheteria (RUMBAUGH, JACOBSON e BOOCK, 1999).

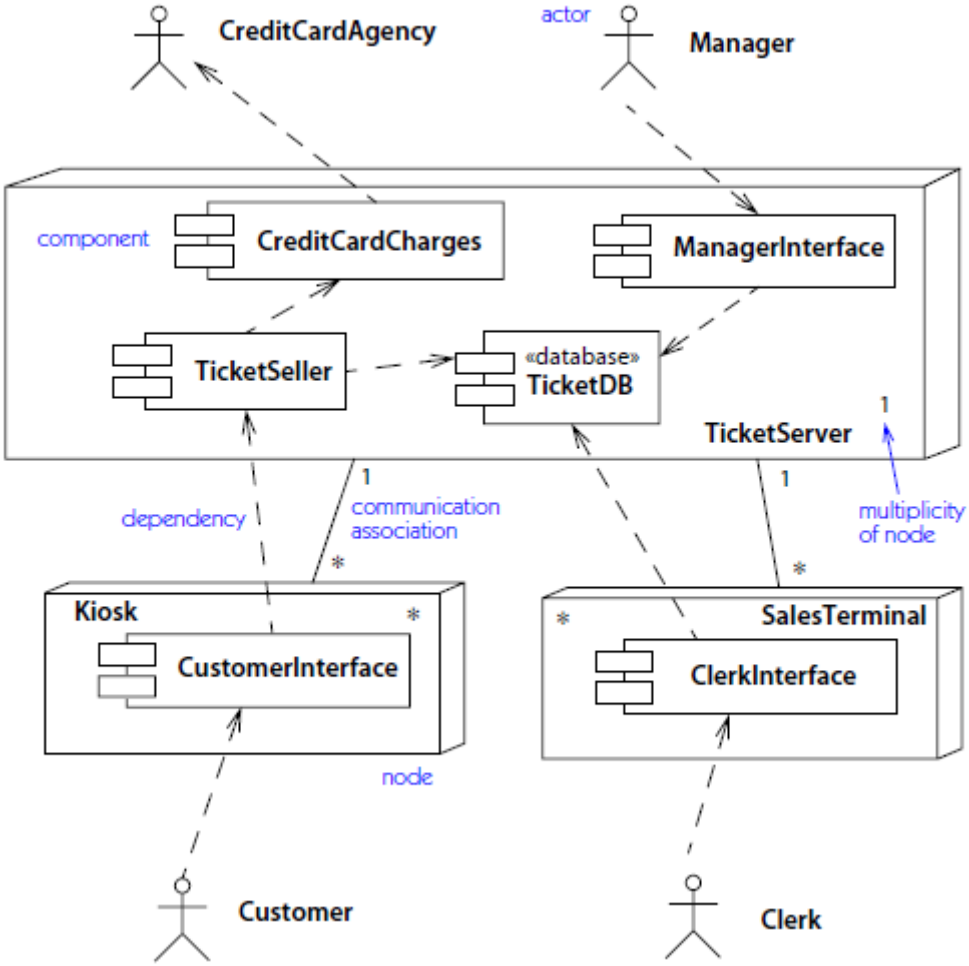

Figura 12 – Exemplo de um diagrama de implantação

Fonte: The Unified Modeling Language Reference Manual (RUMBAUGH, JACOBSON e BOOCK, 1999)

2.8. PostgreSQL

A linguagem SQL (*Structured Query Language*) é o padrão utilizado pela maioria dos bancos de dados do mercado. Isto decorre da sua simplicidade e facilidade de uso. Ela se diferencia de outras linguagens de consulta a banco de dados no sentido de que uma consulta SQL especifica a forma do resultado e não o caminho para chegar a ele. É uma linguagem declarativa em oposição a outras linguagens procedurais. Isto reduz o ciclo de aprendizado daqueles que se iniciam na linguagem (TECWORKS, 2011).

O software *pgAdmin*[16] foi utilizado para exemplificar uma consulta simples a um banco de dados:

[16] Provê uma interface amigável para o gerenciamento e administração de um banco de dados *PostgreSQL*.

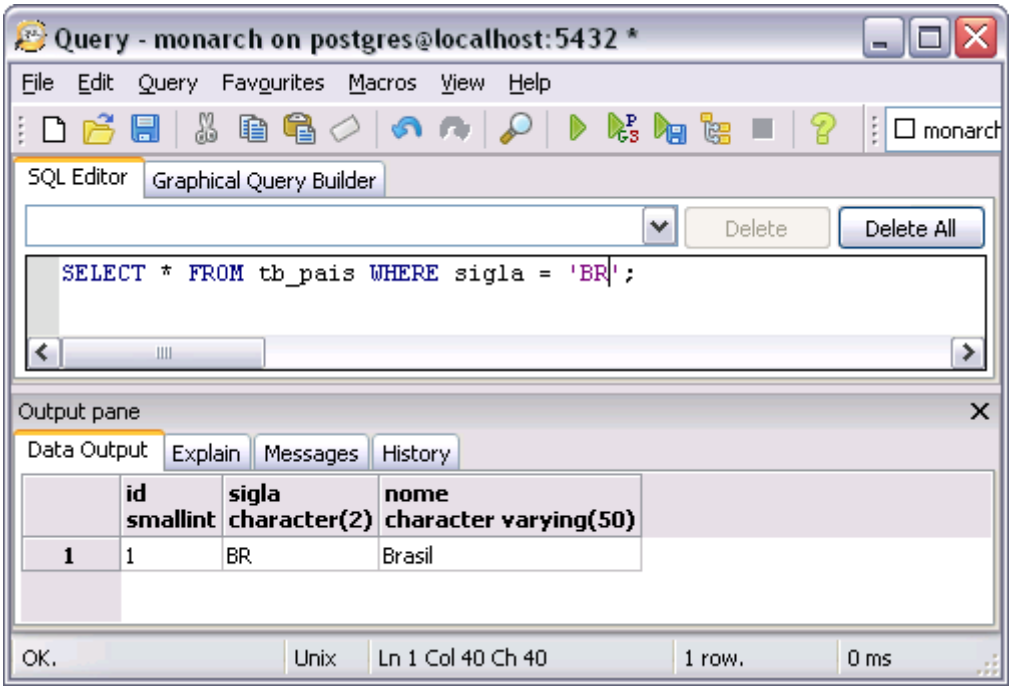

Figura 13 – Consulta SQL

Fonte: Elaboração própria

A consulta executada na Figura 13 pode ser “traduzida” da seguinte forma: busque (*SELECT*) todas as colunas (*) da tabela tb_pais, onde (*WHERE*) sua sigla seja igual a ‘BR’. Veja que na parte de baixo da figura o resultado é exibido.

Em um contexto histórico essa linguagem foi extremamente importante na história do *PostgreSQL*. Em 1996 a primeira versão externa desse SGBD (*Sistema Gerenciador de Banco de Dados*) foi lançada, por conta da padronização do SQL como linguagem de consulta, o projeto foi renomeado de *Postgres95* para o nome atual. A primeira versão renomeada foi a 6.0, lançada em 1997, desde então o *PostgreSQL* é mantido e atualizado por um grupo de desenvolvedores e de voluntários de todo o mundo, coordenados pela Internet (POSTGRESQL, 2011).

2.9. SMS

Segundo Machado (2011), a primeira década do milênio foi radicalmente influenciada pelo SMS, ao se tornar um hábito entre pessoas de todas as idades, transformando-se em uma das principais formas de comunicação em todo o planeta.

Mesmo com a chegada do MMS[17] (*Multimedia Messaging Service*), e-mail e redes sociais nos *smartphones*[18], além de uma infinidade de outras aplicações que permitem

[17] Tecnologia que permite aos dispositivos móveis enviar e receber mensagens multimídia.

envio de mensagens, nenhuma tecnologia chegou perto de abalar o reinado do SMS como a forma mais simples, rápida e eficaz de comunicação móvel instantânea. O motivo é simples: todos os celulares, em uso e futuros sempre serão compatíveis com o SMS, algo que nenhuma outra tecnologia chegou perto de realizar.

2.9.1. Detalhes técnicos

HowStuffWorks (2005) diz que o SMS ou mensagem de texto, é a tecnologia que possibilita a transmissão e o recebimento de mensagens curtas de texto no celular, ele surgiu como parte das especificações da tecnologia *wireless*[19] GSM na Europa e teve o seu primeiro envio bem sucedido em 1992 no Reino Unido. Um padrão garante a entrega das mensagens, caso o usuário estiver fora de área, tentativas de entrega sucessivas são realizadas.

Essa tecnologia foi projetada para entregar pequenos fragmentos de dados. Para evitar a sobrecarga do sistema, seus criadores concordaram com um tamanho máximo de 160 caracteres para cada mensagem.

Mas esse limite não é absoluto e os tamanhos máximos variam de rede para rede, de aparelho para aparelho e de operadora para operadora. Alguns telefones não permitem que se continue digitando após o limite de 160 caracteres, fazendo com que se tenha que enviar a mensagem atual antes de poder continuar. No entanto, outros serviços quebram automaticamente qualquer mensagem que se envie, permitindo que você digite e envie uma mensagem longa, pois ela será entregue como várias mensagens menores.

2.9.2. Mercado brasileiro

Segundo Sanches (2010) pesquisas realizadas por *Anatel, Teleco* e *Acision* apontam o SMS como a tecnologia de comunicação mais utilizada no Brasil, essa afirmação é comprovada pelos seguintes números, também disponibilizados por Sanches (2010):

- 190 milhões é aproximadamente a quantidade de celulares no Brasil.
- 5,7 bilhões de mensagens de texto são enviadas mensalmente no País.
- 58% dos usuários de celular enviam SMS.
- 20% é a possibilidade de redução no absenteísmo com o uso de SMS.

[18] Telefone celular com funcionalidades avançadas que podem ser estendidas por meio de programas executados no seu sistema operacional.

- 10% utilizam celular para serviços de comunicação instantânea.
- 3% utilizam para envio de e-mail.

2.10. Human Gateway

Por ser uma tecnologia originalmente destinada à troca de mensagens entre celulares, a utilização do SMS em aplicações *Web* ainda é pequena. Atualmente existem muitas empresas especializadas na prestação desse tipo de serviço, uma delas é a *Human*, empresa brasileira que possui uma plataforma para integração com sistemas existentes, na qual praticamente qualquer sistema pode ser integrado de forma simples e rápida.

Essa empresa disponibiliza uma vasta documentação sobre os seus serviços, seguem algumas informações sobre o Gateway para envio de SMS:

- Integração feita através de HTTP, HTTPS[20] (*Hypertext Transfer Protocol Secure*), *Web Services* ou e-mail, padrões universais para qualquer linguagem ou banco de dados.
- Envio de SMS a partir do próprio sistema do cliente.
- Envio instantâneo ou agendado.
- Envio individual ou em lote.
- Status de entrega de cada mensagem.
- Relatórios detalhados de uso.
- Apoio técnico na implantação.
- Segurança através de criptografia SSL (*Secure Sockets Layer*).
- Compatível com *Java*, PHP, ASP, *Perl*, .NET, VB, *Delphi*, entre outras.
- Maior biblioteca de exemplos de código do mercado.

2.11. Considerações finais

Neste capítulo foram apresentadas as revisões dos conceitos básicos de desenvolvimento Web, como HTML, CSS e JavaScript, além de conceitos mais refinados como as plataformas *Java* e .NET.

Estes conceitos introduzem todas as tecnologias utilizadas no desenvolvimento do projeto, com isso os próximos capítulos tendem a uma linguagem mais técnica e concisa.

---

[19] Rede que não necessita de cabos ou fios.
[20] Implementação do protocolo HTTP sobre a camada de segurança SSL (*Secure Sockets Layer*) ou TLS (*Transport Layer Security*).

3. ESTUDO DE CASO

3.1. Considerações iniciais

Para exemplificação dos conceitos apresentados até aqui, duas aplicações *Web* foram criadas e implementadas em diferentes ambientes de desenvolvimento, visando explorar o melhor das plataformas *Java* e *.NET*. A esse projeto foi dado o nome *Monarch Communication*.

Esse capítulo abordará desde o modelo de dados utilizado pelas aplicações até a arquitetura e resultados obtidos em cada uma delas.

3.2. Modelo de dados

Para ilustrar as características e relacionamentos do banco de dados do sistema o software *DBVizualizer Free* na versão 8.0.5 foi utilizado, o resultado pode ser observado abaixo:

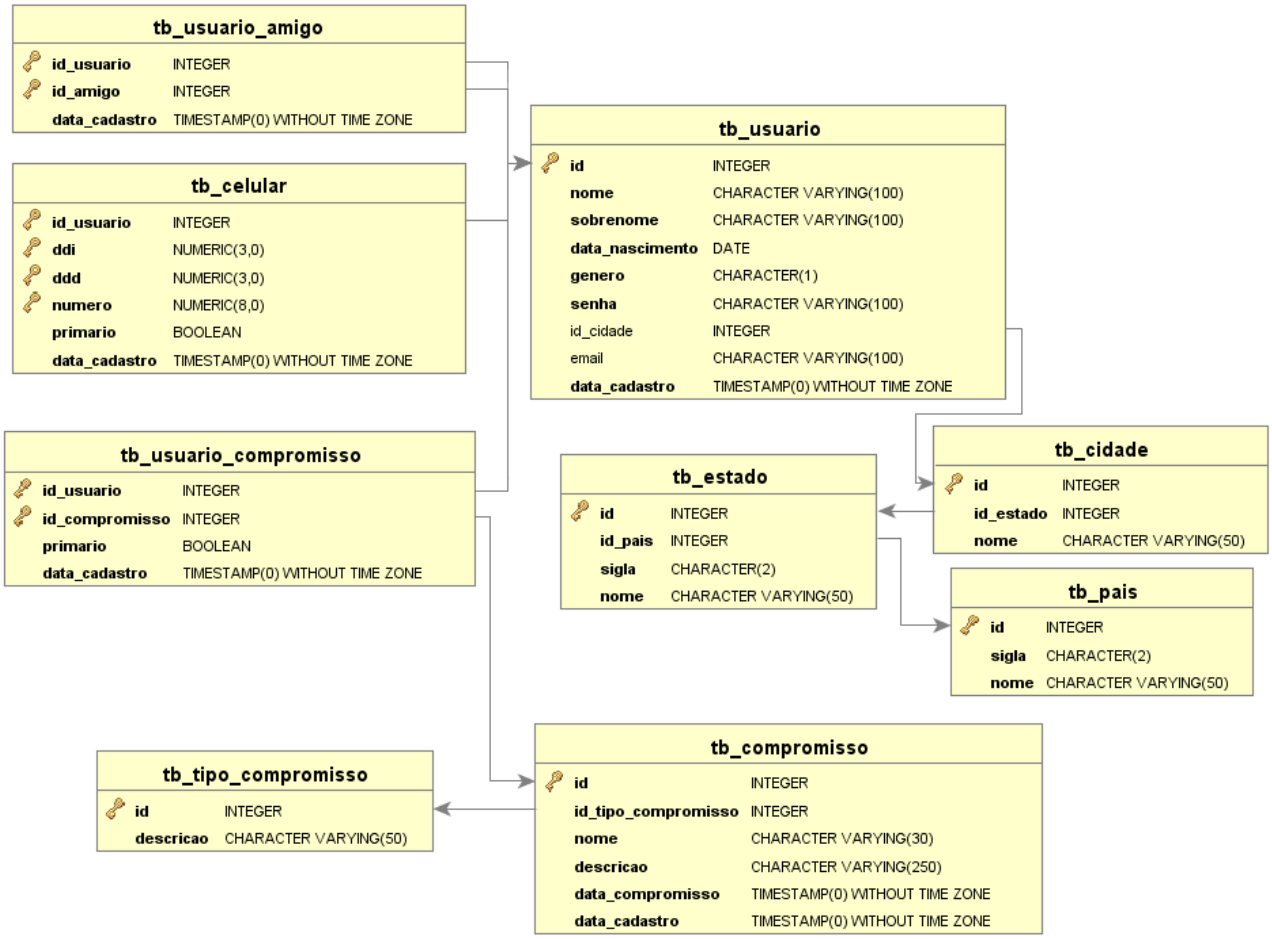

Figura 14 – Modelo de dados *Monarch Communication*

Fonte: Elaboração própria

Quatro das nove tabelas exibidas no modelo são de domínio, ou seja, elas terão uma carga de dados fixa que servirá como base para futuras consultas de localização, são elas: tb_pais, tb_estado, tb_cidade e tb_tipo_compromisso.

A tb_usuario é, com certeza, a tabela mais importante do sistema, já que guarda todos os dados do usuário e ainda exporta informações para outras três tabelas: tb_usuario_amigo, tb_celular, tb_usuario_compromisso. Além disso, ela ainda importa informações sobre a localização do usuário da tabela tb_cidade.

Os compromissos dos usuários serão salvos na tb_compromisso, ela fornece informações à tabela tb_usuario_compromisso e importa o tipo de compromisso da tb_tipo_compromisso.

O relacionamento muitos para muitos das tabelas tb_usuario e tb_compromisso é representado pela tabela tb_usuario_compromisso, seguindo dessa forma uma das regras da normalização de banco de dados.

As informações dos telefones celulares do usuário serão salvas na tabela tb_celular, que se encontra devidamente normalizada e associada à tb_usuario.

A tabela tb_usuario_amigo representa um relacionamento bidirecional na tb_usuario, portanto importará dois identificadores. É essa tabela que gerenciará todas as listas de amigos do sistema.

Todas as tabelas que não são de domínio tem a coluna data_cadastro, essa prática foi adotada para facilitar futuras consultas informativas.

3.3. Aplicação ASP.NET: Interface visual

A aplicação que disponibiliza a interface visual será feita em ASP.NET, essa plataforma em conjunto com seu IDE proporciona diversas facilidades na criação de páginas dinâmicas em ambiente *Web*, agilizando o desenvolvimento e produzindo interfaces visuais de qualidade.

Nessa aplicação o usuário se cadastrará, autenticará, fará vinculo com outros usuários e agendará seus compromissos.

3.3.1. Ambiente de desenvolvimento

O IDE *Visual Studio 2010* deu todo o suporte necessário para a criação de uma aplicação *Web* de interface amigável, além de disponibilizar o servidor de aplicação IIS 7 nativamente. O *.NET Framework* foi configurado na versão 4.0.

Foram criados cinco projetos do tipo *Class Library* para garantir o reaproveitamento de todas as classes de serviço, negócio, acesso a dados, entidades e

infraestrutura. Isso é possível porque esse tipo de projeto pode ser salvo como DLL e posteriormente ser reaproveitado em outro projeto, independentemente do seu tipo (*Web*, *Desktop*, *Mobile,* etc).

Por fim um projeto ASP.NET *Empty Web Aplication* foi criado, nele todas as páginas visíveis ao usuário foram configuradas, além de todas as bibliotecas de apoio.

O *Visual Studio* permite a manipulação dos seis projetos em um único contexto, chamado de *Solution.*

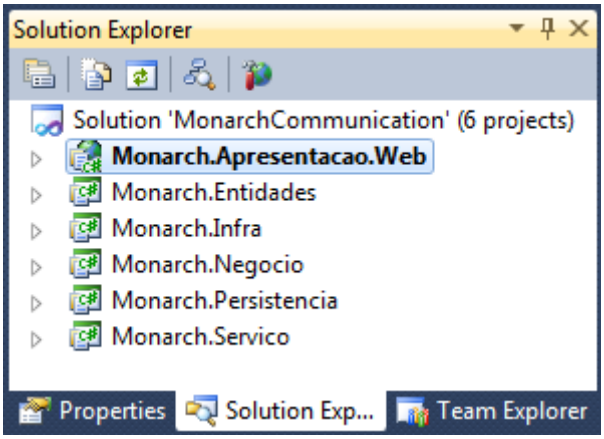

Figura 15 – Solution aplicação ASP.NET

Fonte: Elaboração própria

3.3.2. Configuração

Apesar do IDE possuir todas as ferramentas necessárias para o desenvolvimento de uma aplicação *Web*, algumas bibliotecas *JavaScript* de grande aceitação foram adicionadas à aplicação:

- ***jQuery 1.6.4:*** Biblioteca que agilizará a implementação com *JavaScript* e dará suporte a seus aos plugins e extensões.
- ***jQuery UI 1.8.16:*** Extensão *jQuery* que disponibiliza diversas opções em interfaces visuais, animações e interatividade. Logicamente possui diversos padrões CSS, já que se trata de um *framework* visual. Por conta de seu tamanho foi disponibilizada também em forma modular.
- ***jQuery Validation 1.9:*** Plugin *jQuery* destinado a validações de formulários no lado cliente.
- ***jQuery MaskedInput 1.3:*** Plugin *jQuery* que possibilita a criação de máscaras na aplicação.

Com os arquivos *JavaScript* devidamente adicionados na aplicação *Web*, basta carregá-los nas páginas de acordo com a necessidade. O conceito de *Master Page* foi utilizado para centralizar as importações comuns de arquivos de *JavaScript* e CSS. Uma *Master Page* é uma página padrão, ou seja, todas as páginas que a utilizarem herdarão sua estrutura.

O plugin *jQuery Validation* teve de ser configurado de forma alternativa, já que os componentes ASP.NET seguem um padrão próprio e não customizável para gerar o valor do atributo *name*. Para solucionar esse problema a função *rules* disponibilizada pelo plugin foi utilizada, com ela é possível especificar as regras de validação de forma legível e sem a utilização de blocos de código embutidos na página.

```
jQuery(function ($) {
    // Ativa o plugin Validation para o formulário atual.
    $('form').validate();
    // Seleciona e percorre os elementos a qual a regra de obrigatoriedade deve ser aplicada.
    $(':text[id="TxtNome"],[id="TxtSobrenome"],[id="TxtCelularPrimario"]').each(function (){
        $(this).rules('add', { required: true });
    });
    // Adiciona a regra de obrigatoriade e tamanho mínimo.
    $('#TxtSenha').rules('add', { required: true, minlength: 5 });
    // Adiciona a regra de obrigatoriade, tamanho mínimo e associação comparativa.
    $('#TxtConfirmacaoSenha').rules('add',{required: true, minlength: 5, equalTo: '#TxtSenha'});
});
```

Figura 16 – Exemplo de utilização do plugin *jQuery Validation*

Fonte: Elaboração própria

3.3.3. Diagrama de componentes

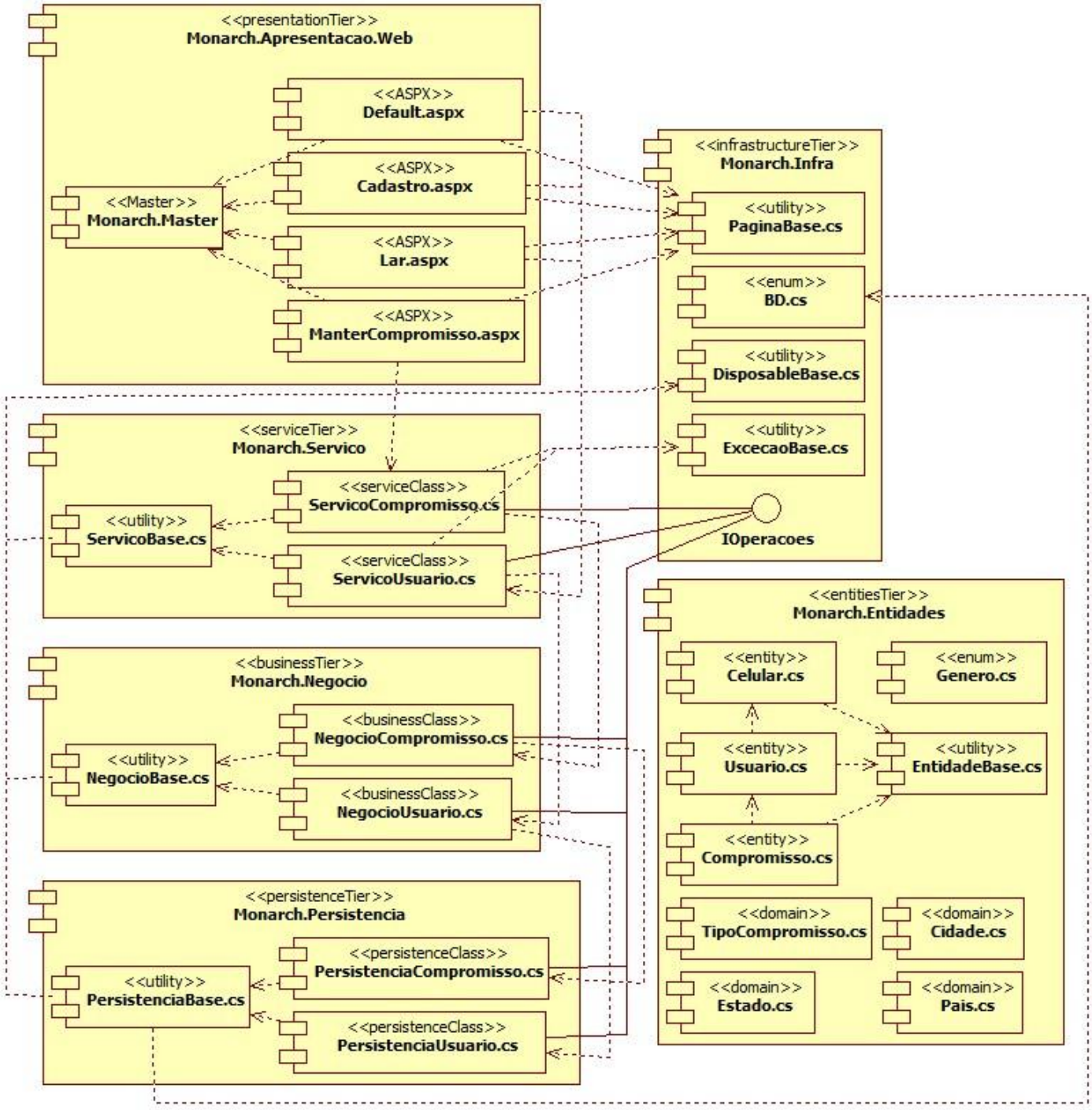

Figura 17 – Diagrama de componentes da aplicação ASP.NET

Fonte: Elaboração própria

Para ilustrar de forma completa a aplicação ASP.NET o diagrama de componentes da Figura 17 foi elaborado, nele todos os relacionamentos entre os projetos definidos na *Solution* podem ser visualizados.

3.3.4. Resultados obtidos

Os resultados obtidos pela aplicação ASP.NET envolvem todas as funcionalidades visíveis ao usuário. As caraterísticas de cada uma delas serão apresentadas a seguir:

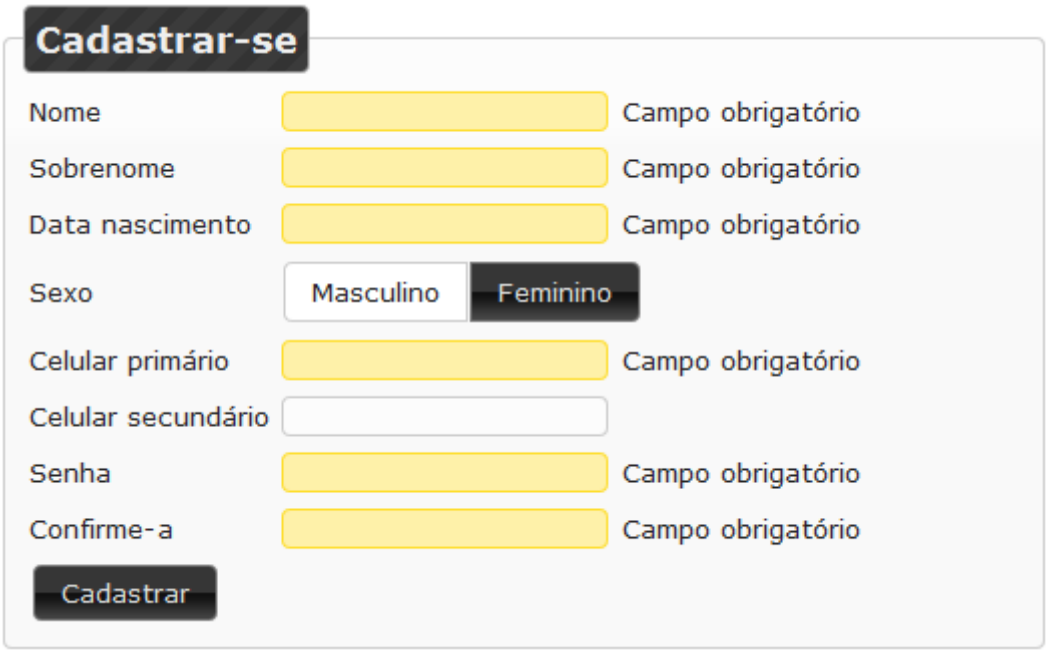

Figura 18 – Formulário de cadastro de usuário

Fonte: Elaboração própria

Figura 19 – *Datepicker jQuery UI*

Fonte: Elaboração própria

O formulário de cadastro de usuário é ilustrado pela Figura 18, nela é possível enxergar o tema visual aplicado via *jQuery UI* e também o estilo adotado para campos inválidos.

Campos do tipo data utilizaram-se do elemento *datepicker* disponibilizado pelo *jQuery UI,* seu padrão visual foi apresentado na Figura 19.

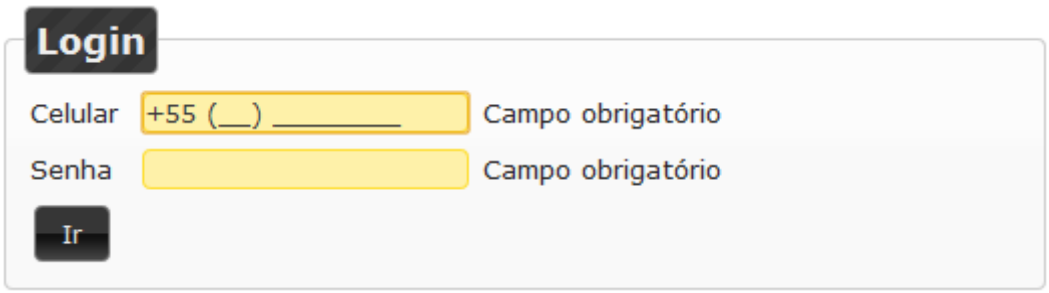

Figura 20 – Formulário de login

Fonte: Elaboração própria

A Figura 20 apresenta o formulário de login da aplicação, nele fica nítido o padrão de máscara para telefones celulares.

Quando um usuário faz a autenticação com sucesso na aplicação seu identificador é armazenado em uma variável de sessão, esse tipo de variável garante que apenas o usuário que a criou indiretamente tenha acesso a ela.

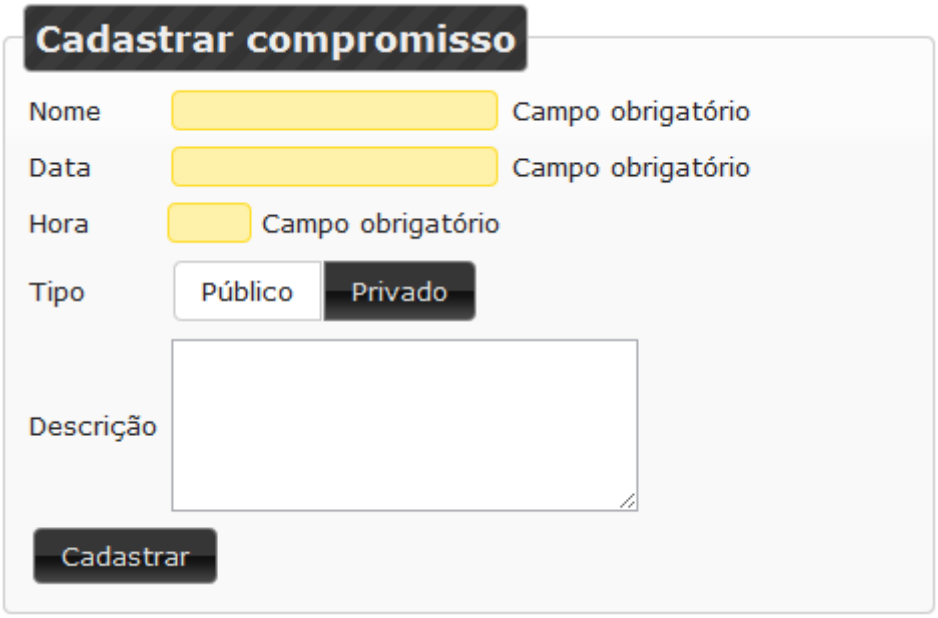

Figura 21 – Formulário de cadastro de compromissos

Fonte: Elaboração própria

O formulário da Figura 21 efetua o cadastro de compromissos dos usuários, além das características já citadas, esse formulário carrega as opções de tipo de compromisso dinamicamente, ou seja, os valores *Público* e *Privado* são obtidos da tabela tipo_compromisso em tempo de execução.

Figura 22 – Abas com as informações dos compromissos
Fonte: Elaboração própria

A Figura 22 ilustra o padrão visual de abas disponibilizado pelo *jQuery UI*, com este componente é possível disponibilizar ao usuário os próximos compromissos, os finalizados e os pendentes em um espaço compartilhado. Isso garante a reutilização de espaços da página de forma elegante e usual.

3.4. Aplicação Java: Temporizador

A execução da rotina de consulta aos compromissos foi implementada utilizando a plataforma *Java* devido à sua característica multiplataforma, que nesse caso, agrega no quesito confiabilidade, já que a aplicação poderá ser disponibilizada em qualquer SO (Sistema Operacional) sem nenhum problema de compatibilidade.

3.4.1. Ambiente de desenvolvimento

O ambiente de desenvolvimento utilizado foi o *Eclipse IDE for Java EE Developers* na versão 3.7.1, chamado pela comunidade de *Eclipse Indigo*. Até o presente momento essa é a versão mais recente desse software, mas apesar disso ainda não suporta os novos recursos disponibilizados na versão 7 do *Java*, por conta dessa limitação, a versão 6 será utilizada no desenvolvimento da aplicação.

Um projeto do tipo *Dynamic Web Project* foi usado como base da aplicação, pois esse tipo de projeto configura automaticamente o ambiente na plataforma *Java EE*, criando diretórios e arquivos de configuração que permitem a utilização otimizada de servidores de aplicação. O servidor *Apache Tomcat* na versão 7.0 foi escolhido devido a sua boa aceitação de mercado e ótima documentação de apoio.

3.4.2. Configuração

Vários *frameworks Java* foram adicionados ao *Build Path*[21] da aplicação, os mais relevantes são:

- ***SLF4J 1.6.2:*** Abstraí vários *frameworks* de log, disponibilizando uma fachada de operações muito mais simples e otimizada. Seu funcionamento em conjunto com o *framework LOG4J* depende apenas da criação e configuração do arquivo log4j.properties em uma pasta ou pacote que o servidor de aplicação carregue.
- ***Spring 3.1:*** Devido a sua característica modular foi possível utilizar apenas as bibliotecas necessárias, são elas: *ASM, BEANS, CONTEXT, CONTEXT.SUPPORT, CORE, EXPRESSION, JDBC, TRANSACTION e WEB.*

[21] Termo usado pelo *Eclipse IDE* para o gerenciamento das bibliotecas utilizadas em um projeto.

- ***Quartz 1.8.5:*** A versão 1.8.5 teve de ser utilizada devido à incompatibilidade das versões posteriores com o *Spring* 3.1.
- ***Human Gateway Client:*** Biblioteca de integração *Java* com o conjunto de classes que tem como principal função fazer requisições HTTP da forma mais organizada e simplificada possível, fazendo com que poucas informações sejam passadas pela aplicação, deixando todos os tipos de configuração para o envio de mensagens de texto (SMS) sob responsabilidade da API. Os SMSs são disponibilizados como crédito, os quais são pagos, porém a *Human* disponibilizou uma ótima quantidade grátis para realização desse projeto.

Devido às características dos *frameworks* utilizados, grande parte do esforço gasto na construção foi direcionado a configuração dos mesmos. Sem dúvida o mais importante deles é o *Spring*, devido a grande utilização do padrão de injeção de dependência.

Para facilitar a compreensão em futuras correções, os arquivos de configuração do *Spring* foram divididos de forma modular, ou seja, um arquivo do tipo XML (*Extensible Markup Language*) foi criado para cada camada da aplicação, cada um deles será detalhado neste tópico.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">
    <display-name>Temporizador</display-name>

    <!-- Contexto de configuração do Spring -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/contexto.xml</param-value>
    </context-param>

    <listener>
        <listener-class>
        org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
</web-app>
```

Figura 23 – web.xml

Fonte: Elaboração própria

A Figura 23 configura o *Listener*[22] do *Spring* que gerencia o contexto da aplicação *Web*, carregando o arquivo contexto.xml como parâmetro.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.1.xsd">

    <import resource="servico.xml" />

    <import resource="dao.xml" />

    <import resource="temporizadorQuartz.xml" />

    <context:property-placeholder location="classpath:*.properties" />

</beans>
```

Figura 24 – contexto.xml

Fonte: Elaboração própria

O arquivo contexto.xml ilustrado na Figura 24 é o arquivo de configuração primário do *Spring*. Nele são importados os arquivos de configuração secundários e é definido o padrão de localização dos arquivos de propriedades.

[22] Objeto que fica esperando que uma ação pré-determinada seja executada, para nesse momento executar a sua ação. Pode-se dizer que é o nome dado a interface *Java* que implementa o padrão de projeto *Observer*.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

    <!-- Configuração da camada de serviço-->

    <bean id="smsService"
        class="com.human.gateway.client.service.SimpleMessageService" >
        <constructor-arg name="account" value="${human.account}" />
        <constructor-arg name="password" value="${human.code}" />
    </bean>

    <bean id="notificacaoService"
        class="monarch.servico.NotificacaoServiceImpl" autowire="byName" />

    <!-- Fim da configuração da camada de serviço -->

</beans>
```

human.properties

```
human.account = venilton
human.code = 3EWLTq7YJF
```

Figura 25 – servico.xml, com seu arquivo de propriedades

Fonte: Elaboração própria

A Figura 25 ilustra o XML de serviços, nele a injeção de dependência é aplicada pela primeira vez, garantido que as classes descritas nos *beans* sejam instanciadas de forma otimizada.

O *bean* smsService referencia a classe *SimpleMessageService* disponibilizada pelo *Human Gateway Client*. Essa classe possui apenas um construtor com dois parâmetros, o que torna a passagem de ambos obrigatória, essas duas constantes são recuperadas a partir do arquivo human.properties, garantindo a injeção de dependência.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

    <!-- Configuração da camada de acesso a dados DAO -->

    <bean id="notificacaoDAO" class="monarch.dao.NotificacaoDAOImpl">
        <property name="dataSource" ref="dataSource" />
        <property name="mensagemPadrao"
            value="${aplicacao.sms.mensagempadrao}"></property>
    </bean>

    <bean id="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="${jdbc.driverClassName}" />
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>

    <!-- Fim da configuração do DAO -->

</beans>
```

aplicacao.properties

```
aplicacao.sms.mensagempadrao = O compromisso...
```

jdbc.properties

```
jdbc.driverClassName = org.postgresql.Driver
jdbc.url = jdbc:postgresql://localhost:5432/monarch
jdbc.username = postgres
jdbc.password = 123456
```

Figura 26 – dao.xml, com seus arquivos de propriedades

Fonte: Elaboração própria

O arquivo dao.xml é fundamental para o funcionamento da aplicação, é nele que a conexão JDBC[23] (*Java DataBase Connectivity*) é configurada a partir do arquivo jdbc.properties. Essa conexão é disponibilizada na classe de acesso a dados e em conjunto com as extensões de suporte JDBC fornece o cenário necessário para a execução de comandos SQL.

A classe de acesso a dados ainda injeta a mensagem padrão, disponível no arquivo aplicacao.properties, essa mensagem é usada como base em todos os SMSs enviados pela aplicação.

[23] Biblioteca escrita em *Java* para o envio de instruções SQL para qualquer banco de dados relacional.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

    <!-- Configuração de tarefas usando CronTrigger do framework Quartz -->

    <bean name="detalheTarefaQuartz"
        class="org.springframework.scheduling.quartz.JobDetailBean">
        <property name="jobClass"
            value="monarch.temporizacao.quartz.CompromissoJob" />
        <property name="jobDataAsMap">
            <map>
                <entry key="notificacaoService" value-ref="notificacaoService" />
            </map>
        </property>
    </bean>

    <bean id="cronTrigger"
        class="org.springframework.scheduling.quartz.CronTriggerBean">
        <property name="jobDetail" ref="detalheTarefaQuartz" />
        <property name="cronExpression" value="0/60 * * * * ?" />
    </bean>

    <bean class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
        <property name="triggers">
            <list>
                <ref bean="cronTrigger" />
            </list>
        </property>
    </bean>
    <!-- Fim da configuração de tarefas usando o Quartz -->
</beans>
```

Figura 27 – temporizadorQuartz.xml

Fonte: Elaboração própria

O arquivo de configuração exibido na Figura 27 integra efetivamente o *framework Quartz* a aplicação, fazendo com que seja possível criar uma expressão cronológica associada a uma *trigger*[24] que indique o intervalo temporal em que a classe *CompromissoJob* será executada. O valor “0/60 * * * * ?” indica que a *trigger* será disparada a cada virada de minuto, por exemplo, 13:10:00, 13:11:00, 13:12:00, etc.

Além disso, também é feita uma referência ao serviço de compromissos declarado no arquivo servico.xml, garantindo sua injeção em tempo de execução.

[24] Englobam um procedimento que é disparado de acordo com uma condição ou evento pré-configurado, no caso das *triggers* do *Quartz* a condição de disparo são as expressões cronológicas.

3.4.3. Diagrama de classes

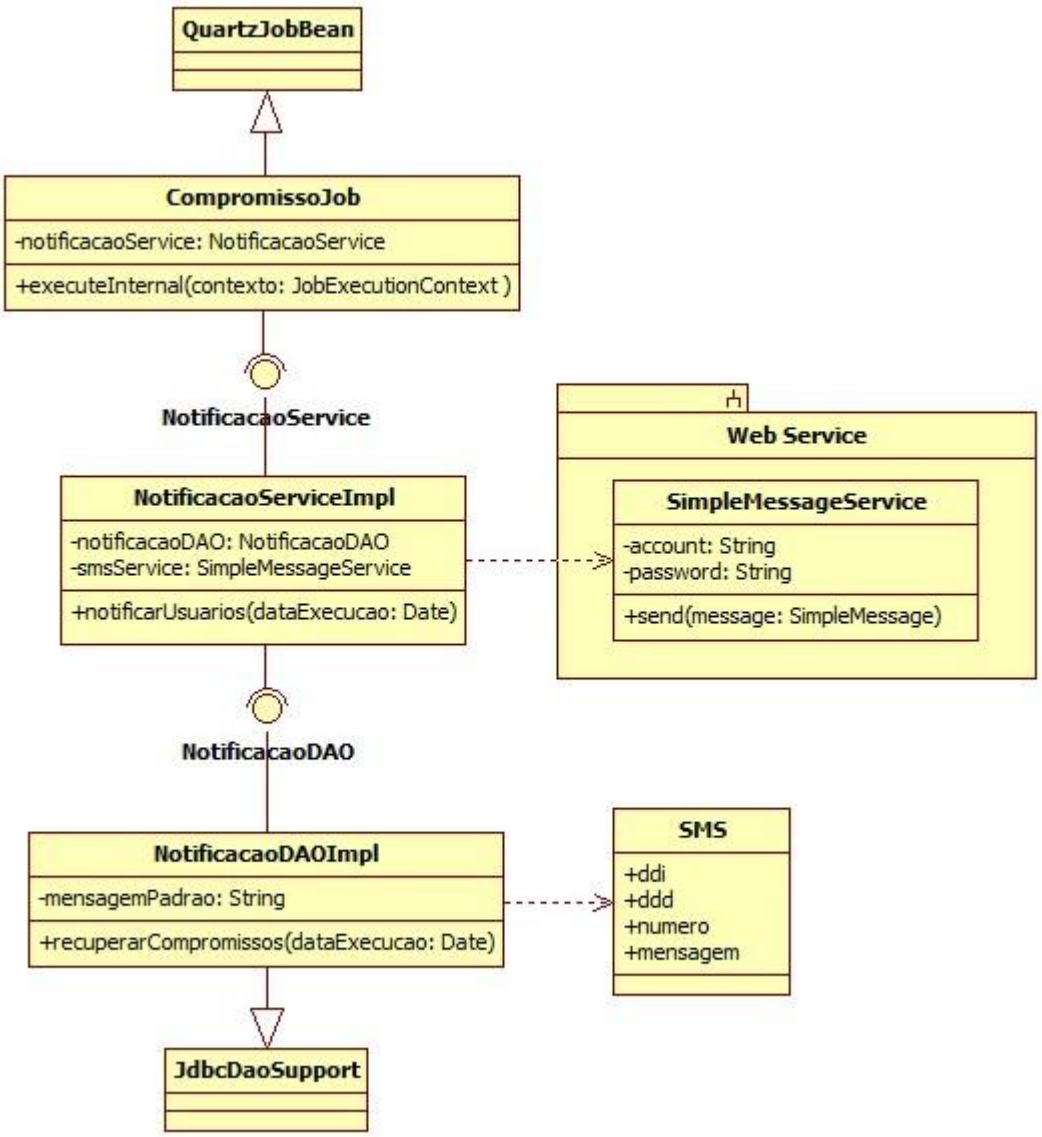

Figura 28 – Diagrama de classes da aplicação *Java*

Fonte: Elaboração própria

O diagrama acima apresenta de forma clara a estrutura do temporizador, lembrando que o *Web Service* é acessado através da biblioteca *Human Gateway Client*, descrita no tópico de configuração.

Uma observação relevante é fato de que todas as propriedades indicadas como privadas no diagrama de classes utilizaram-se do conceito de injeção de dependência.

3.4.4. Resultados obtidos

Com a aplicação devidamente configurada, toda a lógica de consulta de compromissos e envio de SMS foi implementada respeitando cada uma das camadas da aplicação, a divisão dos pacotes e o log de saída podem ser visualizados na Figura 29:

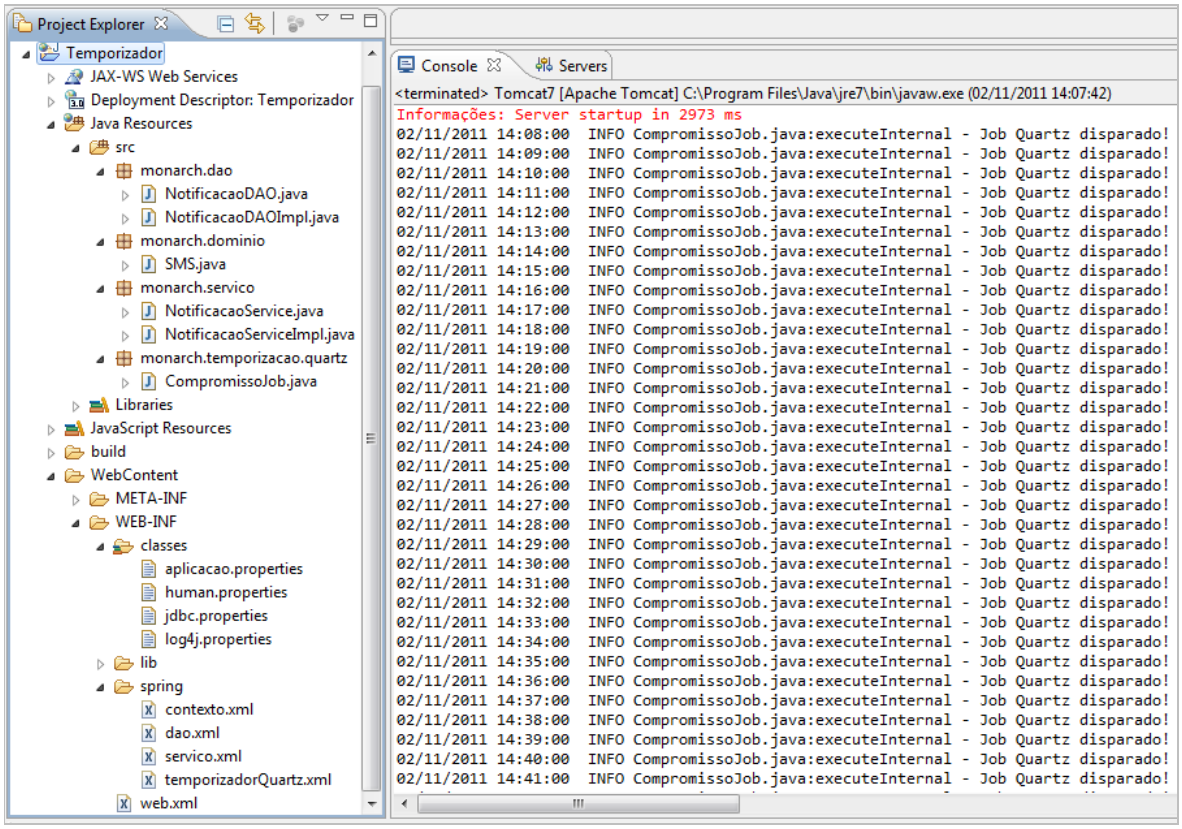

Figura 29 – Console aplicação *Java*

Fonte: Elaboração própria

Perceba que é possível identificar que o método *executeInternal* da classe *CompromissoJob* é executado a cada minuto, esse padrão foi definido no arquivo na propriedade *log4j.appender.stdout.layout.ConversionPattern* do arquivo log4j.properties.

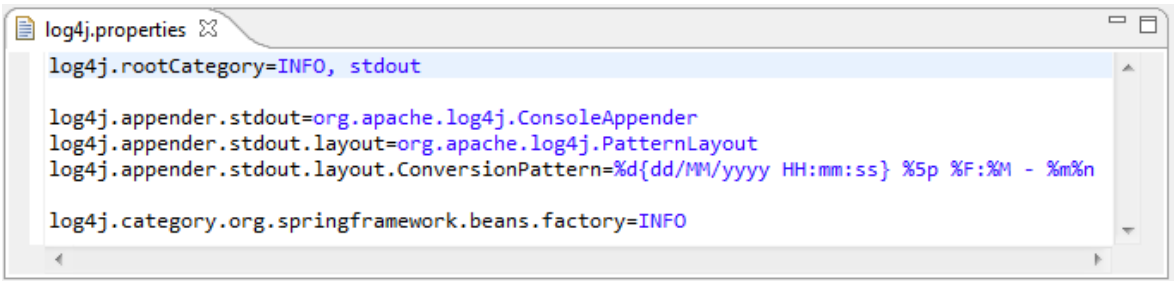

Figura 30 – log4j.properties

Fonte: Elaboração própria

3.5. Diagrama de implantação

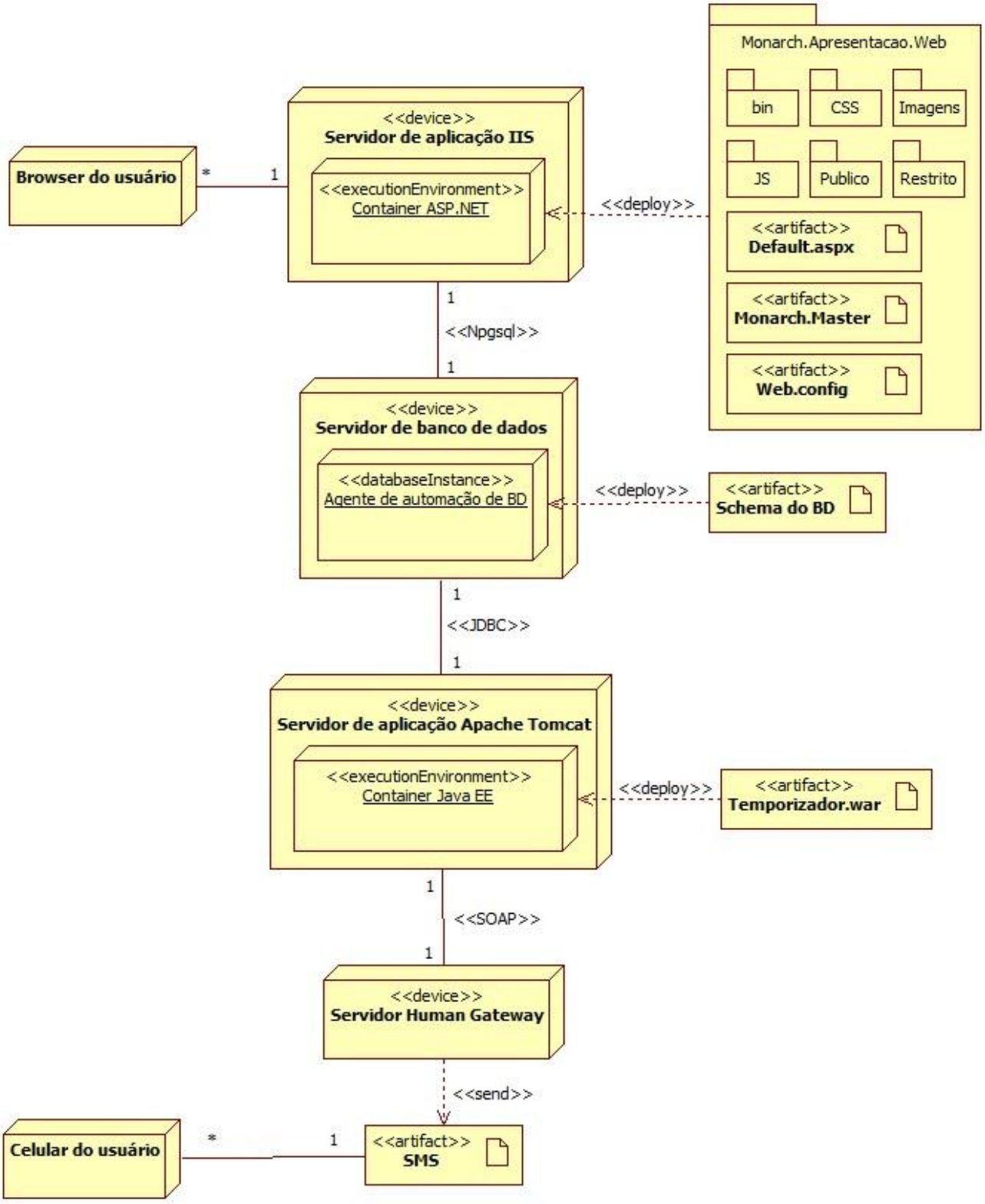

Figura 31 – Diagrama de implantação

Fonte: Elaboração própria

A Figura 31 apresenta o diagrama de implantação do estudo de caso, nele é possível visualizar desde as interações conhecidas pelo usuário até detalhes técnicos da comunicação entre os servidores de aplicação, inclusive o servidor do *Web Service* acessado via SOAP pela biblioteca de integração.

3.6. Considerações finais

As informações apresentadas neste capítulo garantem um maior entendimento das aplicações como um todo, justificando e esclarecendo as arquiteturas propostas.

Neste capítulo foi descrito todo o processo de desenvolvimento de duas aplicações *Web*, desde seus ambientes de desenvolvimento e configurações até diagramas auxiliares e resultados obtidos.

4. CONCLUSÃO

Aplicações Web possuem uma infinidade de possibilidades quanto à arquitetura, por isso o conhecimento das características e ramificações de uma plataforma é crucial para que o resultado esperado seja obtido.

O estudo de caso desenvolvido buscou exaltar as principais características das plataformas *Java* e *.NET* em ambiente *Web*. *Frameworks* e bibliotecas tidos atualmente como padrões de desenvolvimento também foram conceituados e aplicados as suas respectivas plataformas.

É possível afirmar que esse trabalho apresentou um contexto onde plataformas totalmente distintas se comunicavam de forma elegante e harmoniosa através de um mesmo banco de dados.

Web Services garantiram a interoperabilidade para o envio de mensagens de texto em uma das aplicações *Web*. O serviço disponibilizado pela empresa *Human* se mostrou confiável e de integração rápida e customizável.

A notificação de usuários através de SMS mostra que esse serviço é um ótimo diferencial em uma aplicação *Web* e que existem empresas nacionais de qualidade especializadas neste tipo de serviço.

REFERÊNCIAS BIBLIOGRÁFICAS

ANTONIOLI, L. Estatísticas, dados e projeções atuais sobre a Internet no Brasil. **To Be Guanary**, 2011. Disponivel em: <http://tobeguarany.com/internet_no_brasil.php>. Acesso em: 2011 Nov. 08.

ARAÚJO, A. V. **Treinamento Avançado em.NET**. São Paulo: Digerati Editorial, 2006.

CASTRO, U. Introdução e Origem do C#. **PontoNetPT**, 2006. Disponivel em: <http://pontonetpt.org/blogs/bcastro/archive/2006/12/23/P11061.aspx>. Acesso em: 1 Nov. 2011.

ECLIPSE. Eclipse Celebrates 10 Years of Innovation. **Eclipse.org**, 2011. Disponivel em: <http://www.eclipse.org/org/press-release/20111102_10years.php>. Acesso em: 4 Nov. 2011.

FERGUSON, J. **C# Bible**. Indianapolis: Wiley Publishing, 2002.

HOMEHOST. Java: Programando por Orientação à Objeto. Homehost, 2011. Disponivel em: <http://www.homehost.com.br/artigos/java_programando_por_orientacao_a_objeto-051.html>. Acesso em: 15 Nov. 2011.

HOWSTUFFWORKS. HowStuffWorks - Como funciona o SMS. **HowStuffWorks**, 2005. Disponivel em: <http://informatica.hsw.uol.com.br/sms.htm>. Acesso em: 13 Jul. 2011.

HTML.NET. Free tutorials on HTML, CSS and PHP - Build your own website - HTML.net. **HTML.net**, 2011. Disponivel em: <http://html.net>. Acesso em: 14 Jul. 2011.

JAVAFREE.ORG. Java - Java Free.org. **JavaFree.org**, 2005. Disponivel em: <http://javafree.uol.com.br/wiki/Java>. Acesso em: 3 Nov. 2011.

JQUERYMAGAZINE. **jQuery Magazine**, 2011. Disponivel em: <http://www.jquerymagazine.com.br/faq.php>. Acesso em: 10 Dez. 2011.

JQUERY. jQuery. **jQuery**, 2011. Disponivel em: <http://jquery.com>. Acesso em: 11 Jul. 2011.

LOTAR, A. **Como programar com ASP.Net e C#**. São Paulo: Novatec Editora, 2007.

MACHADO, C. B. Vida longa ao SMS. **Human**, 2011. Disponivel em: <http://www.human.com.br/artigos/vida-longa-ao-sms>. Acesso em: 13 Jul. 2011.

MICROSOFT. Microsoft Visual Studio 2010 - O site oficial do Visual Studio 2010. **Microsoft**, 2011. Disponivel em: <http://www.microsoft.com/visualstudio/pt-br>. Acesso em: 13 Jul. 2011.

MOBILEPRONTO. História do SMS. **MobilePronto**, 2010. Disponivel em: <http://www.mobilepronto.org/historia-sms.html>. Acesso em: 14 Jul. 2011.

MORRISON, M. **Use a cabeça JavaScript**. São Paulo: Alta Books, 2008.

MOZILLA. About JavaScript. **Mozilla Developer Network**, 2011. Disponivel em: <https://developer.mozilla.org/en/About_JavaScript>. Acesso em: 14 Jul. 2011.

NABOULSI, Z.; FORD, S. **Coding Faster:** Getting More Productive with Microsoft® Visual Studio®. Sebastopol, California: O'Reilly, 2011.

ORACLE. O que é o Java e por que é necessário? **Java**, 2010. Disponivel em: <http://www.java.com/pt_BR/download/faq/whatis_java.xml>. Acesso em: 4 Nov. 2011.

ORACLE. Saiba mais sobre a tecnologia Java. **Java**, 2011. Disponivel em: <http://www.oracle.com/us/corporate/press/444374#>. Acesso em: 7 Out. 2011.

POSTGRESQL. PostgreSQL: Documentation. **PostgreSQL**, 2011. Disponivel em: <http://www.postgresql.org/docs/>. Acesso em: 14 Jul. 2011.

RICHTER, J. **CLR via C#**. Redmond: Microsoft Press, 2010.

ROSSI, A. A revolução do SMS. **Info Exame**, p. 59-63, 2011.

RUMBAUGH, J.; JACOBSON, I.; BOOCK, G. **The Unified Modeling Language Reference Manual**. Massachusetts: Addison Wesley Longman, 1999.

SANCHES, D. Na palma da mão. **Saúde Business**, 2010.

SHEPHERD, G. **Microsoft Asp.NET 2.0:** Step by step. Washington: Microsoft Press, 2005.

TECHNET. Implantando um Servidor Web IIS 7. **TechNet - Microsoft**, 2009. Disponivel em: <http://technet.microsoft.com/pt-br/library/cc753079(WS.10).aspx>. Acesso em: 03 Nov. 2011.

TECWORKS. Desenvolvimento TecWorks. **TecWorks**, 2011. Disponivel em: <http://tecworks.com.br/desenvolvimento.html>. Acesso em: 10 Nov. 2011.

THAI, T. L.; LAM, H. Q..**NET framework essentials**. Sebastopol: O' Reilly, 2003.

TIOBE. TIOBE Programming Community Index. **TIOBE Software**, 2011. Disponivel em: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>. Acesso em: 19 Out. 2011.

VARELA, M..NET Iniciante: Introdução ao C#. **Linha de Código - A informação no momento em que você precisa**, 2005. Disponivel em: <http://www.linhadecodigo.com.br/Artigo.aspx?id=740>. Acesso em: 3 Nov. 2011.

WROX. Java: Expert One-on-One J2EE Design and Development. **Wrox**, 2002. Disponivel em: <http://www.wrox.com/WileyCDA/WroxTitle/productCd-0764543857.html>. Acesso em: 04 Nov. 2011.