Tópicos em Multimídia, Hipermídia e Web

Book · November 2013

CITATIONS
0

READS
426

4 authors, including:

Renato Bulcão Neto
Universidade Federal de Goiás
39 PUBLICATIONS **172** CITATIONS
SEE PROFILE

Iwens I. G. Sene
Universidade Federal de Goiás
22 PUBLICATIONS **68** CITATIONS
SEE PROFILE

Cássio V. S. Prazeres
Universidade Federal da Bahia
59 PUBLICATIONS **131** CITATIONS
SEE PROFILE

TÓPICOS EM MULTIMÍDIA, HIPERMÍDIA E WEB

Edição
Renato de Freitas Bulcão Neto (UFG)
Iwens Gervásio Sene Júnior (UFG)

Editora
Sociedade Brasileira de Computação

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

VIRTUTE SPIRITUS 1808
UFBA

UNIVERSIDADE SALVADOR
UNIFACS
UNIVERSIDADE SALVADOR
LAUREATE INTERNATIONAL UNIVERSITIES

MINICURSOS

XIX Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia)

5 a 8 de Novembro de 2013

Salvador-BA

TÓPICOS EM MULTIMÍDIA, HIPERMÍDIA E WEB

Edição

Sociedade Brasileira de Computação

Coordenação de Minicursos

Renato de Freitas Bulcão Neto (UFG)

Iwens Gervásio Sene Júnior (UFG)

Coordenação Geral

Cássio Vinicius Serafim Prazeres (UFBA)

Paulo Nazareno Maia Sampaio (UNIFACS)

Promoção

Sociedade Brasileira de Computação

XIX SIMPÓSIO BRASILEIRO DE SISTEMAS MULTIMÍDIA E WEB
(WebMedia)

5 a 8 de novembro de 2013
Salvador – BA

Tópicos em Multimídia, Hipermídia e Web

Minicursos

Coordenação de Minicursos
Renato de Freitas Bulcão Neto (UFG)
Iwens Gervásio Sene Júnior (UFG)

Coordenação Geral
Cássio Vinicius Serafim Prazeres (UFBA)
Paulo Nazareno Maia Sampaio (UNIFACS)

Sociedade Brasileira de Computação
Bahia
2013

FICHA CATALOGRÁFICA

Tópicos em multimídia, hipermídia e Web : minicursos / coord.

R. F. Bulcão Neto, I. G. Sene Júnior ; coord. geral C. V. S. Prazeres, P. N. M. Sampaio. - Bahia : Sociedade Brasileira de Computação, 2013.

p.

Minicursos realizados durante o XIX Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia), 5 a 8 de novembro de 2013.

ISBN: 978-85-7669-281-2

1. Multimídia 2. Hipermídia 3. Web I. Bulcão Neto, Renato de Freitas II. Sene Júnior, Iwens Gervásio III. Prazeres, Cássio Vinícius Serafim IV. Sampaio, Paulo Nazareno Maia V. t.

CDD 006.7

Índice

Prefácio

Este livro é uma coletânea de minicursos, eventos de curta duração, com resultados de atividades de pesquisa científica ou tecnológica. Os minicursos visam apresentar tópicos atuais de pesquisa e/ou tecnologias de interesse da comunidade de Sistemas Multimídia e Web. Pretende-se oferecer oportunidades de aprendizagem e de utilização de novos conhecimentos nas áreas de atuação ou de pesquisa dos participantes dos minicursos.

A XIX edição do Simpósio Brasileiro de Sistemas Multimídia e Web – WebMedia 2013 – contou com a submissão de 10 propostas de minicursos, com a abrangência das 3 trilhas principais de interesse do evento: Web e Redes Sociais, Multimídia e TV Digital, Computação Móvel e Ubíqua. A revisão das propostas foi feita por uma equipe de 8 pesquisadores com atuação nas referidas trilhas de pesquisa do evento. Cada proposta de minicurso foi submetida à avaliação de, pelo menos, 2 revisores, segundo os critérios de originalidade e adequação ao tema do evento, o potencial para atrair o público-alvo proposto e a própria limitação imposta pelo reduzido número de minicursos previsto no programa do evento. Dessa maneira, 4 propostas foram selecionadas para apresentação no evento, bem como para publicação do conteúdo proposto na forma de capítulo de livro.

O primeiro minicurso deste livro aborda o tema acessibilidade na Web, de fundamental importância para que o acesso à Web seja possível também a pessoas com deficiência. O minicurso descreve recomendações e técnicas sobre como tornar conteúdo Web e multimídia mais acessível, com técnicas específicas para múltiplos tipos de mídia e elementos estruturais e de navegação, tomando o contexto educacional como pano de fundo.

No capítulo 2 deste livro são discutidos padrões emergentes para o desenvolvimento de aplicações Web, como a linguagem *HTML5*, as Folhas de Estilo em Cascata versão 3 (*CSS3*) e as bibliotecas Javascript *JQuery* e *JQueryMobile*. Os autores do minicurso demonstram o uso combinado dessas tecnologias na construção de um aplicativo para automação residencial, notadamente no desenvolvimento de interfaces com usuário para controle de hardware por meio da Web.

O minicurso descrito no capítulo 3 deste livro trata o desenvolvimento de aplicações voltadas para Interação Natural. Os autores abordam desde conceitos básicos dessa área de pesquisa e dispositivos de interação, como os sensores do *Microsoft Kinect*, até o arcabouço *OpenNI* como plataforma de desenvolvimento promissora para aplicações em cenários que demandam interação natural, como gestos. Por fim, são apontados desafios e oportunidades para desenvolvedores de software e produtores de dispositivos que envolvem interação natural.

Finalizando o livro, o capítulo 4 apresenta uma abordagem baseada em tecnologias Web para tratar desafios inerentes ao desenvolvimento de jogos para dispositivos móveis, como limitações de recursos e heterogeneidade de hardware, bem como a heterogeneidade de plataformas operacionais. Essa abordagem envolve extrair o potencial das tecnologias HTML5, CSS3 e JavaScript para desenvolver aplicativos para dispositivos móveis, objetivando o equilíbrio entre independência de plataforma e desempenho, requisito característico de jogos.

Os editores agradecem a todos os autores que se esmeraram em produzir os capítulos de livro segundo o formato que lhes foi sugerido e em se deslocar até Salvador para ministrar seus respectivos cursos. Registramos também nosso reconhecimento frente ao trabalho minucioso e de qualidade realizado pela equipe de revisores: Dilvan de Abreu Moreira (USP), Fernando Antônio Mota Trinta (UFC), Iwens Gervásio Sene Júnior (UFG), Mário Antonio Meireles Teixeira (UFMA), Renato de Freitas Bulcão Neto (UFG), Renan Gonçalves Cattelan (UFU), Ricardo Couto Antunes

da Rocha (UFG) e Samyr Béliche Vale (UFMA). Por fim, agradecemos também a todos os colegas, instituições e patrocinadores que contribuíram para o sucesso do WebMedia 2013 e para a viabilidade da edição deste livro.

Renato de Freitas Bulcão Neto (UFG)

Iwens Gervásio Sene Júnior (UFG)

Coordenadores dos Minicursos do WebMedia 2013

TÓPICOS EM MULTIMÍDIA, HIPERMÍDIA E WEB

Capítulo 1

Acessibilidade de Conteúdo Web e Multimídia: técnicas e exemplos do contexto educacional

André Pimenta Freire, Raphael Winckler de Bettio, Elaine das Graças Frade, Fernanda Barbosa Ferrari, José Monserrat Neto e Helena Libardi

Abstract

Developing accessible Websites is essential to enable disabled people to have access to content and day-to-day services. As stated by Tim Berners-Lee, the inventor of the Web, "access to the Web by everyone regardless of disability is an essential aspect". This short course presents techniques to help design more accessibility Web and multimedia content for people with different types of disabilities. This chapter presents examples from educational contexts, including issues with text, images, audio, video, structural elements and navigation. For different content types, the chapter includes practical examples focussed on how accessibility issues may affect users with different types of disabilities, and how to improve the accessibility of different types of content.

Resumo

O desenvolvimento de sítios Web acessíveis é fundamental para que pessoas com deficiência possam ter acesso a conteúdo e a serviços essenciais do dia-a-dia. Como indicado por Tim Berners-Lee, o inventor da Web, "o acesso à Web por todas as pessoas, independente de deficiências, é um aspecto fundamental". Este capítulo apresenta técnicas para o design de conteúdo Web e multimídia para torna-los mais acessíveis a pessoas com diferentes tipos de deficiência. O capítulo traz exemplos do contexto educacional, incluindo questões de acessibilidade para texto, imagens, áudio, vídeo, elementos estruturais e de navegação. Para todos esses tipos de conteúdo, são mostrados exemplos práticos com foco em como as questões de acessibilidade podem afetar usuários com diferentes tipos de deficiência e como melhorar a acessibilidade de diferentes tipos de conteúdo.

1.1. Introdução

A inclusão de pessoas com deficiência nas mais diversas esferas da sociedade é um grande desafio que deve ser enfrentado na sociedade brasileira. É importante que

pessoas com deficiência tenham acesso às mais diversas oportunidades de educação, trabalho, entretenimento, entre outros. Pessoas com deficiência são especialmente beneficiadas pelo acesso a serviços online, uma vez que diversos recursos online podem fornecer meios para que elas vivam de forma independente e desempenhem tarefas de forma facilitada (Hanson et al., 2009). Para que os sítios Web sejam mais inclusivos, é importante reconhecer que o público alvo incluirá pessoas que podem ter alguma deficiência visual, auditiva, física, cognitiva, ou dificuldades de aprendizado específicas, tais como a dislexia. As necessidades de cada um desses grupos devem ser levadas em conta no design de sítios Web para que todos possam ter acesso ao conteúdo e serviços disponibilizados. De maneira particular, órgãos públicos no Brasil tem responsabilidade legal de tornar seus sítios Web acessíveis, como determinado pelo Decreto/Lei 5.296 de 2004.

De fato, a inclusão de pessoas com deficiência no uso de sítios Web é um dos desafios de pesquisa incluídos nos grandes desafios da computação do Brasil definidos pela Sociedade Brasileira de Computação, na linha de garantir "acesso participativo e universal do cidadão brasileiro ao conhecimento".

Para que os sítios Web sejam mais inclusivos, é importante reconhecer que o público alvo incluirá pessoas que podem ter alguma deficiência visual, auditiva, física, cognitiva, ou dificuldades de aprendizado específicas, tais como a dislexia. As necessidades de cada um desses grupos devem ser levadas em conta no design de sítios Web para que todos possam ter acesso ao conteúdo e serviços disponibilizados. Por exemplo, usuários cegos normalmente utilizam tecnologias assistivas para síntese do conteúdo textual em voz. Caso páginas Web em um sítio dependam unicamente em características visuais e não tenham recursos técnicos como descrição de imagens e marcação de seções de links corretos, usuários cegos podem encontrar grandes dificuldades para utilizar um sítio.

Nos últimos anos, o desenvolvimento de projetos para inclusão de pessoas com deficiência tem sido o objetivo do Plano "Viver Sem limites" do Governo Brasileiro (2013). De acordo com os resultados preliminares do Censo 2010 (IBGE, 2013), cerca de 23,9% da população afirmou ter algum tipo de deficiência.

Apesar da importância de tornar sítios Web acessíveis, diversos estudos tem mostrado que muitos sítios Web ainda apresentam problemas para o uso por pessoas com deficiência (Coyne e Nielsen 2001, Disability Rights Commission 2004, Leuthold et al. 2008, Petrie and Kheir 2007, Theofanos and Redish 2003). No maior desses estudos, conduzido pela Disability Rights Commission of Great Britain (2004), foi verificado que usuários cegos eram capazes de completar somente 53% das tarefas que eles tentaram fazer, mostrando que problemas de acessibilidade impediam seriamente que usuários tivessem acesso a uma quantidade significativa de informações e serviços na Web. Esses resultados ressaltam a criticidade de tornar sítios Web mais acessíveis e criar sítios Web que usuários com deficiência possam usar efetivamente.

Um dos fatores que pode contribuir para a baixa acessibilidade de sítios Web é o baixo conhecimento de desenvolvedores sobre como pessoas com deficiência utilizam a Web e sobre como fazer o projeto e implementacão desses sítios de forma a evitar que problemas de acessibilidade ocorram (Disability Rights Commission, 2004; Freire et al., 2008). Desta forma, é fundamental que desenvolvedores de sítios e aplicações Web

tenham conhecimento e adotem boas práticas de design inclusivo que favoreçam o acesso a pessoas com deficiência.

Neste capítulo, são apresentados conceitos básicos sobre acessibilidade para conteúdo Web e multimídia, bem como conceitos sobre recursos de Tecnologia Assistiva e como pessoas com diferentes tipos de deficiência utilizam sítios Web. O capítulo apresenta, em seguida, recomendações e técnicas sobre como tornar conteúdo Web e multimídia mais acessível, com técnicas específicas para diferentes tipos de mídia, como textos imagens, áudio e vídeo, e elementos estruturais e interativos de páginas, tais como tabelas, cabeçalhos, links, formulários e navegação.

1.2. Acessibilidade em Sistemas Web e Multimídia

A definição do termo "acessibilidade" tem gerado discussão no meio científico, em particular em relação ao limiar entre usabilidade e acessibilidade (Petrie e Kheir, 2007). As questões do uso de sistemas computacionais são relacionadas ao atributo de usabilidade de sistemas. O conceito de acessibilidade para pessoas com deficiência, como definido pela norma ISO 9242-Parte 141 (ISO, 2008) traz o conceito de acessibilidade mais próximo ao conceito de usabilidade. Segundo a ISO 9241 - *Standard on Ergonomics of Human System Interaction*- Part 11 (ISO, 1998), o conceito de usabilidade é definido como:

> "A medida que um produto [um software ou ambiente] pode ser utilizado por um grupo específico de usuários para atingir objetivos específicos com eficácia, eficiência e satisfação em um contexto específico de uso".

Seguindo a mesma linha, a Parte 171 da ISO 9241 (ISO, 2008) define acessibilidade de software como sendo:

> "a usabilidade de um produto, serviço, ambiente ou funcionalidade por pessoas com os mais variados tipos de habilidades e capacidades".

Essa definição estende a definição de usabilidade para pessoas com os mais variados tipos de capacidades, em particular aquelas que têm alguma deficiência.

Pela definição do *World Wide Web Consortium*, o conceito de acessibilidade na Web significa que pessoas com diferentes tipos de limitação podem perceber, entender, navegar, interagir e contribuir para a Web (Caldwell et al., 2008). Em contraste, usamos o termo "barreira" de acessibilidade, para designar qualquer obstáculo que dificulte ou impossibilite pessoas com deficiência de usar a Web.

Em 1999 um grupo de profissionais da *World Wide Web Consortium* (W3C) se integraram na Iniciativa para Acessibilidade na Web (WAI - *Web Accessibility Initiative*) e definiram a primeira versão do documento "Diretrizes de Acessibilidade ao Conteúdo da Web" (WCAG 1.0 - *Web Content Accessibility Guidelines*) (Chisholm et al., 1999). O WCAG 1.0 é composto por um conjunto de 14 diretrizes que prometem solucionar problemas de acessibilidade encontrados nos conteúdos Web. Estas diretrizes são divididas basicamente em duas categorias:

1. Garantir a transformação harmoniosa das páginas;
2. Criar conteúdos de fácil navegação e compreensão.

A primeira categoria é composta de 11 diretrizes, e para satisfazê-las deve-se obedecer aos principais conceitos a seguir:

a) Separar a estrutura do aplicativo de sua apresentação;
b) Fornecer texto (incluindo texto equivalente). O texto pode ser renderizado nas formas em que está disponível para quase todos os dispositivos e acessível a quase todos os usuários;
c) Criar documentos que sejam interpretados mesmo que o usuário não possa ver e/ou ouvir;
d) Criar documentos que não dependam de um tipo de hardware.

Já a segunda categoria define que o desenvolvedor deve fazer uso de linguagem simples e clara, além de disponibilizar mecanismos para auxiliar a navegação dentro de um conteúdo ou entre as páginas que compõem o conteúdo.

Como dito anteriormente, o WCAG 1.0 é composto por um conjunto de 14 diretrizes. Cada diretriz é descrita por meio de pontos para verificação, que são classificados por três níveis de prioridade. O primeiro nível, Prioridade 1, determina os pontos em que o desenvolvedor Web tem a necessidade de satisfazer, evitando que os usuários com deficiências fiquem impossibilitados de compreender as informações contidas no sítio. A Prioridade 2 indica os pontos que devem ser satisfeitos pelo desenvolvedor para evitar que o usuário tenha dificuldades para acessar todas as informações contidas na página, evitando barreiras significativas ao conteúdo. Por fim, a Prioridade 3, apresenta os pontos em que o desenvolvedor Web pode satisfazer para melhorar o acesso ao conteúdo.

Além disso, o WCAG 1.0 define os níveis de conformidade, que representam uma classificação que varia de acordo com a satisfação das diretrizes propostas. São definidos 3 níveis de conformidade:

- Nível de conformidade A: Todos os pontos de verificação de Prioridade 1 são satisfeitos;
- Nível de conformidade AA: Todos os pontos de verificação de Prioridade 1 e de Prioridade 2 são satisfeitos;
- Nível de conformidade AAA: Todos os pontos de verificação de Prioridade 1, 2e 3 são satisfeitos.

Por praticamente uma década, a primeira versão do WCAG foi utilizada como um padrão de fato para a acessibilidade na Web. Apesar disso, o impacto dessas diretrizes no aumento da acessibilidade na Web não foi muito significativo nesse período. Sendo assim, a maior preocupação sobre o WCAG 1.0 era a falta de evidências comprovando que uma página com nível de conformidade AAA era realmente mais acessível às pessoas com deficiência do que as páginas com nível A de conformidade. A partir dessa preocupação, foram realizados estudos para analisar a eficiência do WCAG. A Disability Rights Commission (DRC) conduziu uma investigação envolvendo mais de 100 páginas Web que foram avaliadas por especialistas e um grande número de pessoas com diferentes tipos de deficiência (Disability Rights Commission, 2004). O estudo mostrou que 45% dos problemas encontrados pelos usuários com deficiência não eram cobertos pelo WCAG 1.0. Com base nesses resultados, observou-se que essa falta de cobertura da totalidade dos problemas encontrados por usuários com deficiência pelo WCAG 1.0 era um importante problema a ser abordado.

Em 2008 o Web Accessibility Initiative (WAI) lançou uma nova versão do WCAG. Essa nova versão, o WCAG 2.0, tinha como objetivo resolver muitos dos problemas encontrados na primeira versão. Esse novo conjunto de diretrizes (Caldwell et al., 2008) foi organizado de uma maneira hierárquica. A nova versão considera quatro princípios sobre a acessibilidade na Web, e segundo esses princípios, o conteúdo deve ser:

- Perceptível;
- Operacional;
- Compreensível;
- Robusto.

Através da utilização desses princípios, as diretrizes são agrupadas em uma hierarquia mais estruturada em relação à hierarquia presente no WCAG 1.0. Além disso, as diretrizes sob cada um desses princípios têm sido reformuladas com o objetivo de solucionar necessidades específicas dos usuários. Ainda, para cada diretriz existe um Critério de Sucesso. Os critérios de sucesso do WCAG 2.0 são escritos como declarações testáveis, que não são especificamente tecnológicas. Orientações sobre satisfazer o critério de sucesso em tecnologias específicas, bem como informações gerais sobre interpretações das regras, são concedidas em documentos separados (Caldwell et al., 2008). A conformidade de uma página Web é medida com base nos critérios de sucesso, sendo que cada critério de sucesso possui um nível de prioridade, nível A, AA ou AAA, de maneira similar ao que ocorre no WCAG 1.0.

Apesar de todas as mudanças, estudos encontrados na literatura mostram que muito problemas em relação à utilização do WCAG 1.0 por desenvolvedores e avaliadores ainda persistem no WCAG 2.0 (Alonso et al., 2010; Brajnik et al., 2010; Petrie et al., 2011). Além disso, ainda faltam evidências empíricas demonstrando que a conformidade com o WCAG 2.0 resulta em páginas mais acessíveis para usuários com deficiência. Esse fato contrasta com diretrizes de usabilidade na Web (U.S. Department of Health & Human Services, 2006) e diretrizes de acessibilidade que foram definidas e validadas para grupos específicos de usuários (Leporini and Paternó, 2008; Leuthold et al., 2008), em particular para usuários com deficiência visual.

No contexto de recomendações de acessibilidade para Web no Brasil, o principal conjunto de recomendações está no e-MAG (Modelo de Acessibilidade do Governo Eletrônico), criado a partir da sanção do Decreto-Lei 5.296 foi criado em janeiro de 2005 pelo Ministério do Planejamento, Orçamento e Gestão juntamente com a Secretaria de Logística e Tecnologia da Informação. O e-MAG consiste em um "conjunto de recomendações a ser considerado para que o processo de acessibilidade dos sítios e portais do governo brasileiro seja conduzido de forma padronizada e de fácil implementação" (e-MAG, 2011). Em maio de 2007, institucionalizou-se o e-MAG no âmbito do sistema de Administração dos Recursos de Informação e Informática – SISP, tornando sua observância obrigatória nos sítios e portais do governo brasileiro (e-MAG, 2011).

A elaboração do e-MAG foi realizada com base em diversos padrões e técnicas, tais como a Section 508 do governo dos Estados Unidos, os padrões da CLF do Canadá, Stanca Act da Itália e outras diretrizes de outros países, no entanto, a que mais teve influência no conteúdo do e-MAG foi a WCAG 1.0.

Assim, o e-MAG é uma iniciativa brasileira para a elaboração de recomendações de acessibilidade, em conformidade com os padrões internacionais. Foi formulado para orientar profissionais que tenham contato com publicação de informações ou serviços na Internet a desenvolver, alterar e/ou adequar páginas, sítios e portais, tornando-os acessíveis ao maior número de pessoas possível (e-MAG, 2011).

O e-MAG foi disponibilizado em sua versão 1.0, mas após alterações que lhes foram propostas, em 14 de dezembro de 2005 foi disponibilizada a versão 2.0. Esta era composta por duas partes (e-MAG, 2011):

- A **cartilha técnica**: apresenta a proposta de implementação das recomendações práticas de acessibilidade em sítios do governo e é destinada aos desenvolvedores de sítios.

- A **visão do cidadão**: apresenta o modelo de acessibilidade de forma mais intuitiva, mais compreensível e simples e é destinada aos cidadãos brasileiros.

Por outro lado, esta divisão sofreu alguns problemas durante o período de disseminação do Modelo, tais como a dificuldade das pessoas para entenderem a Visão do Cidadão e aplicar a acessibilidade. Verificando essas falhas e com o lançamento do WCAG 2.0 tornou-se necessário a revisão do Modelo (e-MAG, 2011).

A revisão do modelo e a elaboração da versão 3.0 foram desenvolvidas pela parceria entre o Departamento de Governo Eletrônico e o Projeto de Acessibilidade Virtual da RENAPI (Rede de Pesquisa e Inovação em Tecnologias Digitais) baseando-se na versão anterior do e-MAG e na WCAG 2.0 e também, considerando as novas pesquisas em acessibilidade Web. O e-MAG 3.0 foi desenvolvido para atender às prioridades brasileiras e manter-se alinhado ao que existe de mais atual na área (e-MAG, 2011).

A versão 3.0 do e-MAG é apresentada em um único documento, não existe mais a divisão entre cartilha técnica e visão do cidadão. Os níveis de prioridade A, AA e AAA, anteriormente presentes, deixaram de existir para não permitir as exceções com relação ao cumprimento das recomendações, uma vez que é voltado para as páginas do Governo. Nesta versão, a fim de padronizar os elementos de acessibilidade nos sítios e portais do governo, foi adicionada uma seção chamada “Padronização de acessibilidade nas páginas do governo federal” (e-MAG, 2011).

O e-MAG recomenda três passos para melhorar a acessibilidade de sítios Web (e-MAG, 2011):

- **Seguir os padrões Web**: o código deve estar em conformidade com os padrões Web internacionais definidos pelo W3C. Uma página que está de acordo com estes deve estar dentro das normas HTML (HyperText Markup Language), XML (eXtensible Markup Language), XHTML (eXtended HyperText Markup Language) e CSS (Cascading Style Sheets).

- **Seguir as diretrizes ou recomendações de acessibilidade**: as diretrizes e recomendações de acessibilidade auxiliam os criadores de conteúdo Web e aos programadores de ferramentas para a criação do conteúdo como tornar o conteúdo Web acessível a todos.

- **Realizar a avaliação de acessibilidade**: após a construção do ambiente Web é necessário realizar a avaliação de acessibilidade, a qual pode ser realizada através de

uma validação automática (avaliadores), os quais são softwares ou serviços online que avaliam o sítio e verificam se este está de acordo com as diretrizes e recomendações de acessibilidade. Porém, esses avaliadores automáticos por si só não determinam de forma completa se um sítio é ou não acessível. Para uma avaliação bem sucedida é necessário também a validação manual após a automática.

O e-MAG possui quarenta e cinco recomendações, as quais possuem grande importância e devem ser seguidas pelos autores de páginas, projetistas de sítios e aos desenvolvedores de ferramentas para criação de conteúdo Web acessível por todos (e-MAG, 2011).

Apesar de utilizar as WCAG 2.0 como referência, o e-MAG 3.0 foi desenvolvido e elaborado para as necessidades locais, visando atender as prioridades brasileiras e mantendo-se alinhado ao que existe de mais atual neste segmento (e-MAG, 2011).

1.3. Tecnologia Assistiva

Um dos aspectos mais importantes para tornar sítios Web mais acessíveis para pessoas com deficiência é entender a forma como elas interagem com computadores e com sítios Web. Um dos principais aliados para a utilização de computadores por esses usuários são recursos de Tecnologia Assistiva, que fornecem adaptações e auxílios para permitir que pessoas com deficiência possam desempenhar suas atividades.

No Brasil, a definição de Tecnologia Assistiva dada pelo Comitê de Ajudas Técnicas é de "uma área do conhecimento, de característica interdisciplinar, que engloba produtos, recursos, metodologias, estratégias, práticas e serviços que objetivam promover a funcionalidade, relacionada à atividade e participação de pessoas com deficiência, incapacidades ou mobilidade reduzida, visando sua autonomia, independência, qualidade de vida e inclusão social". Cook e Polgar (1995) definem Tecnologia Assistiva como "uma ampla gama de equipamentos, serviços, estratégias e práticas concebidas e aplicadas para minorar os problemas encontrados pelos indivíduos com deficiências".

A utilização de recursos de Tecnologia Assistiva por pessoas com deficiência tem grande influência sobre a forma como elas interagem com sítios Web. É muito importante que desenvolvedores conheçam as funcionalidades, vantagens e limitações de recursos de Tecnologia Assistiva para que seus sítios Web sejam desenvolvidos de forma a funcionar de maneira adequada com tais recursos.

Diferentes recursos de Tecnologia Assistiva podem ser utilizadas por pessoas com vários tipos deficiência, e incluem recursos tecnológicos simples, tais como uma bengala, óculos, regletes para escrita de Braille para cegos, e outros recursos tecnológicos computacionais, como sintetizadores de voz para o conteúdo da tela para cegos, ampliadores de tela para pessoas com baixa visão, teclados e dispositivos alternativos para pessoas com deficiência motora, dentre muitos outros.

Usuários cegos com pouca ou nenhuma visão residual normalmente utilizam softwares leitores de tela. Tais softwares possuem um sintetizador de voz que obtém informações textuais contidas na tela do computador e reproduz tais informações na forma de voz. Existem diversos softwares leitores de tela, tais como o Jaws® da

FreedomScientific, o Window-Eyes® da GW Micro, o software livre NVDA[1], e os brasileiros Virtual Vision da Micropower e o Dosvox, desenvolvido pelo NCE da Universidade Federal do Rio de Janeiro.

É muito importante compreender como se dá a interação de usuários cegos com computadores. Eles realizam a interação com o computador primordialmente utilizando o teclado e recebendo a saída por meio de voz sintetizada. Assim, é muito importante que os desenvolvedores considerem meios para que os usuários possam interagir com suas aplicações e sítios Web somente com o uso do teclado. Se alguma funcionalidade só é disponível com o uso do mouse, usuários cegos podem ter problemas para utilizá-las. O mesmo problema acontece com usuários que tem alguma deficiência motora e que não podem utilizar o mouse.

A leitura feita pelo leitor de tela normalmente é sequencial, geralmente começando da parte superior à esquerda e seguindo pela direita em páginas Web. Entretanto, a maioria dos usuários cegos não interage com os elementos somente com o conteúdo que aparece na ordem em que é lido. Os usuários tem uma série de recursos em softwares leitores de tela que os permitem navegar por diferentes partes da tela ou de páginas Web, tais como saltos por links ou por cabeçalhos. Desta forma, a disponibilização de elementos estruturais como cabeçalhos explicativos pode auxiliar usuários cegos a encontrar a informação que desejam muito mais rápido. Por outro lado, caso sejam colocados links que não fazem sentido quando lidos fora do contexto, tais como "clique aqui", os usuários que utilizam uma lista de links podem ficar perdidos. Na Figura 1.1 é ilustrada a funcionalidade de listagem de links de uma página Web simulada pelo plugin *Fangs* do navegador Firefox.

Em estudos realizados com usuários cegos utilizando sítios Web, foram identificados diversos problemas que precisariam ser corrigidos para melhorar a interação de sítios Web com esses usuários. No estudo feito pela *Disability Rights Commission* (DRC) *of Great Britain* (2004), os principais problemas encontrados por usuários cegos em sítios Web foram relacionados à incompatibilidade entre leitores de tela e os sítios Web, quando não eram detectados links ou não era possível acessar algum texto com o leitor, quando não era possível identificar o propósito de links, elementos de formulários e de *frames*, páginas abarrotadas e complexas, textos alternativos de imagens não existentes ou inadequados, e mecanismos de navegação confusos. No estudo realizado por Freire (2012), os cinco problemas mais críticos em termos de frequência e severidade encontrados por usuários cegos foram controles ou elementos de formulários que não eram acessíveis via teclado, falta de audiodescrição de vídeos, estruturas de navegação inadequadas, falta de identificação adequada da função de controles e elementos de formulários, e falta de *feedback* dado de forma acessível sobre a conclusão de ações efetuadas.

[1] Disponível em http://www.nvaccess.org, acesso em julho de 2013

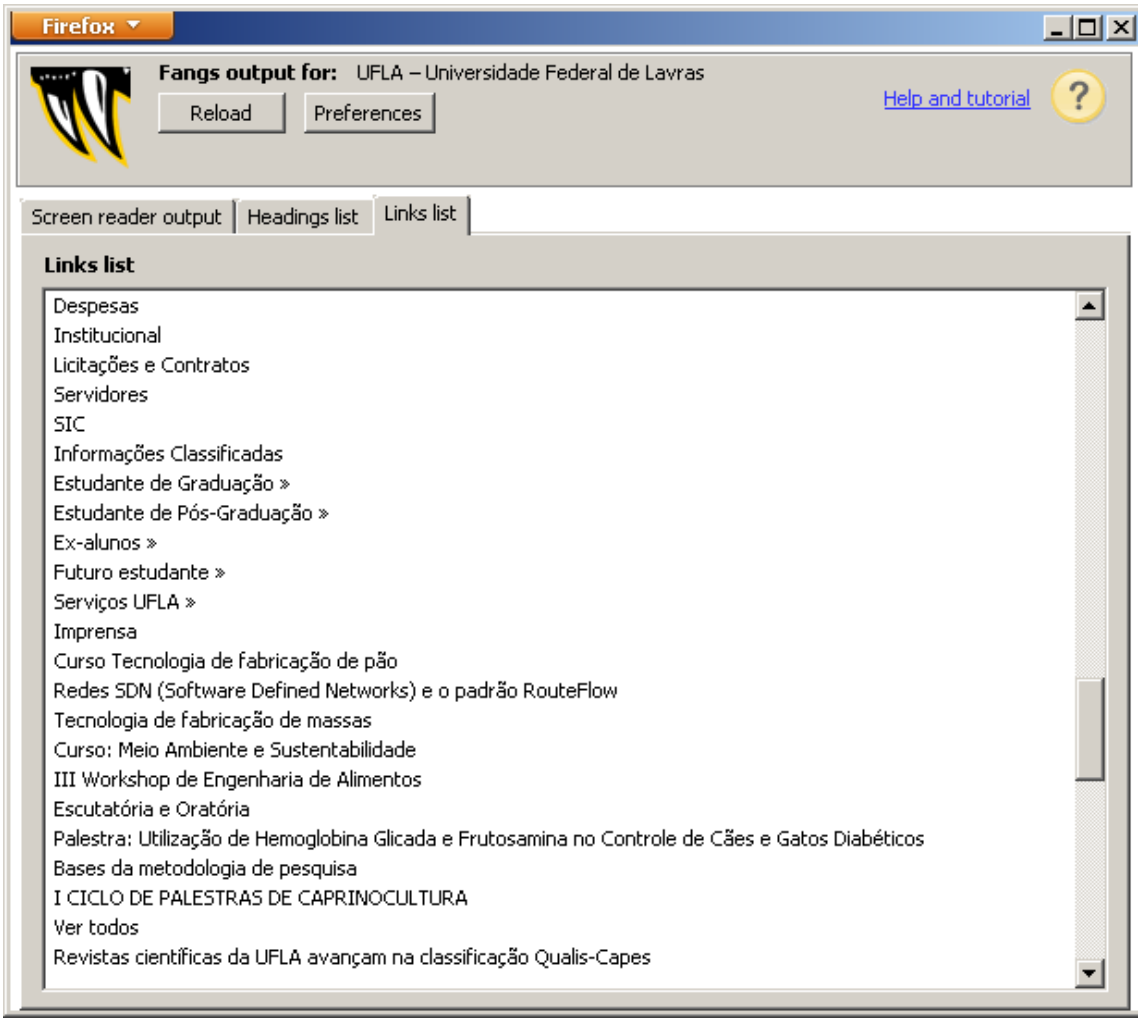


Figura 1.1: Visão de uma lista de links como vista por um leitor de telas utilizado por usuários cegos

Usuários com baixa visão tem formas de interação distintas de usuários cegos. Dependendo da condição da visão dos usuários, muitos deles interagem utilizando mouse e teclado, e podem ou não utilizar softwares leitores de tela. A maioria dos usuários com baixa visão utiliza alguma forma de adaptação da exibição do conteúdo na tela, seja por meio da alteração da resolução, uso de lentes e ampliadores, ou alteração do contraste das cores exibidas. Existe uma grande variação entre os tipos de configurações que podem ser utilizadas, e é importante investigar como usuários com diferentes configurações utilizam sítios Web, pois isso tem influência nos tipos de necessidade que podem surgir para adaptar os sítios Web. Diversos softwares ampliadores de tela estão disponíveis no mercado, tais com o ZoomText® da AiSquare, o Magic da FreedomScientic, o Supernova da Dolphin e o ampliador que acompanham o sistema operacional Windows. Na Figura 1.2, é ilustrado um exemplo de um trecho de uma página com uma lente ampliadora móvel com texto aumentado e cores invertidas pelo software Supernova da empresa Dolphin.

Para muitos usuários, se o contraste entre cores não for bom o suficiente, pode ser impossível enxergar conteúdo em alguns lugares da tela. Da mesma forma, se alterações no tamanho não puderem ser realizadas ou se forem feitas de maneira

insatisfatória (imagens pixeladas, por exemplo), muitos usuários com baixa visão não poderão ler o conteúdo de páginas Web.

Figura 1.2: Exemplo da ampliação de texto e inversão de cores com o ampliador de telas Supernova da empresa Dolphin

No estudo realizado pela DRC (2004), os principais problemas encontrados por usuários com baixa visão foram o uso de cores inapropriadas para contraste entre fundo e o conteúdo, incompatibilidade entre software de ampliação de tela e as páginas, layout de páginas confusos, mecanismos de navegação confusos e tamanhos de gráficos e textos muito pequenos. No estudo realizado por Freire (2012), os problemas mais críticos encontrados por usuários com baixa visão foram a apresentação inadequada de elementos de formulários e de controles de interface, de imagens, e texto, navegação confusa e não encontrar as informações nos lugares onde eram esperados.

Usuários com deficiência motora também tem uma ampla gama de necessidades que podem variar de acordo com o perfil de cada usuário. Usuários com deficiência em membros superiores podem não utilizar o mouse, utilizar o teclado com velocidade reduzida (com um só dedo, por exemplo), ou caso não tenham movimentos nas mãos, podem precisar utilizar uma ponteira de cabeça, como ilustrada na Figura 1.3 ou um software que acompanhe o movimento dos olhos para interagir com o computador. Outros usuários com tremores nas mãos ou dificuldade para utilizar somente uma tecla do teclado por vez podem utilizar recursos como o "teclado colmeia", ilustrado na Figura 1.4, que possui uma cobertura de acrílico com furos nas teclas, para evitar que duas teclas sejam pressionadas ao mesmo tempo.

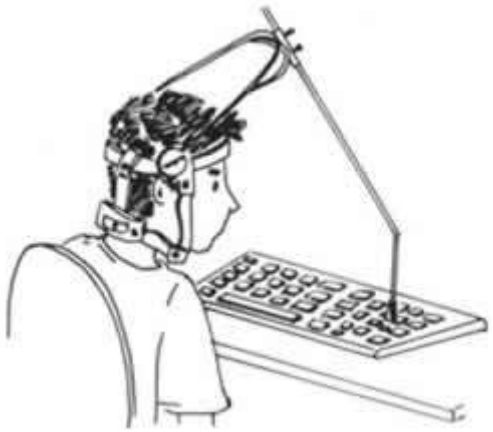


Figura 1.3: Ponteira de cabeça utilizada por usuários com deficiência motora para utilizar o teclado - Fonte: http://www.portalsaofrancisco.com.br

Figura 1.4: Teclado colmeia, com restritores de acrílico sobre as teclas para evitar o pressionamento simultâneo indesejado
Fonte: http:// www.civiam.com.br

Para diversos usuários com deficiência motora, a falta de acessibilidade de recursos interativos por meio do teclado pode ser um fator impeditivo para a interação. Exigir que os usuários tenham uma grande destreza motora para efetuar operações pode ser problemático para diversos grupos de usuários com deficiência.

No estudo realizado pela DRC (2004), os principais problemas encontrados por usuários com deficiência motora foram mecanismos de navegação confusos, layout de páginas confusos, falta de acessibilidade de elementos pelo teclado, e problemas com tamanho de fonte e de elementos interativos.

Para usuários com deficiência auditiva, é importante ressaltar que existem diferentes grupos com diferentes tipos de deficiência. Um importante grupo corresponde a pessoas cuja primeira língua é a LIBRAS (Língua Brasileira de Sinais), e que tem o português ou outra língua como segunda língua. Para essas pessoas, a opção mais favorável para comunicação será sempre sua primeira língua – a LIBRAS. Sempre que possível, é importante disponibilizar conteúdo em LIBRAS para esses usuários, principalmente em conteúdo multimídia com áudio e vídeo. A utilização de legendas também é muito importante para esses usuários. Para muitos usuários que tem LIBRAS como primeira língua, o uso de termos complexos em páginas também pode apresentar sérias barreiras para o entendimento do conteúdo.

Também há usuários surdos que não necessariamente são usuários de LIBRAS, e que terão preferência pelo uso de legendas em conteúdo de áudio e multimídia. Dessa forma, para sítios Web com conteúdo multimídia, é importante que as duas alternativas de conteúdo sejam disponibilizadas.

Um grupo que muitas vezes não é considerado por desenvolvedores Web é o de usuários com dificuldades específicas de aprendizado, tais como a dislexia. A dislexia é relacionada a dificuldades com a codificação e decodificação de texto e outras tarefas relacionadas a processamento verbal, podendo ocorrer conjuntamente com outras dificuldades de processamento de informações espaciais, coordenação motora entre outros.

Usuários com dislexia podem encontrar diversas dificuldades com páginas Web que não apresentem certas características para adaptar o conteúdo para facilitar a leitura e compreensão por esses usuários. Por exemplo, diversos aspectos relacionados à formatação de texto tem um forte impacto na facilidade de leitura desses usuários, como o espaçamento entre linhas, o tamanho da fonte, contraste de cores, dentre outros. É importante que páginas Web sejam feitas de forma flexível para facilitar tais configurações. Muitos usuários com dislexia também podem recorrer a softwares leitores de tela para auxiliar na leitura de conteúdo, tornando importante também para esse grupo de usuários a acessibilidade de conteúdo de forma que possa ser acessado por leitores de tela. A organização de páginas com excesso de informações e falta de organização clara e lógica também pode prejudicar seriamente o acesso a páginas Web por esses usuários.

1.4. Técnicas para Produção de Conteúdo Web e Multimídia Acessível

A produção de conteúdo Web e multimídia acessível é muito importante para que usuários com diferentes tipos de deficiência possam ter acesso e utilizar informações de maneira eficaz. É importante que os desenvolvedores conheçam e utilizem tais técnicas durante a produção de suas aplicações.

Caso tais técnicas sejam utilizadas desde os estágios iniciais do desenvolvimento, a incorporação de melhorias na acessibilidade tem seu custo reduzido significativamente, em comparação com a aplicação tardia de técnicas no desenvolvimento.

Nesta seção, são apresentadas algumas das técnicas mais importantes para tornar diferentes tipos de conteúdo Web e multimídia mais acessíveis por pessoas com deficiência, incluindo para mídias de texto, imagens, áudio e vídeo, e para elementos estruturais e de navegação em sistemas Web. Para cada técnica, são fornecidos exemplos trazidos do contexto educacional e são explicados os impactos da utilização das técnicas para a melhoria da acessibilidade para diferentes grupos de usuários.

1.4.1. Conteúdo Textual

O uso de conteúdo na forma textual é uma das formas mais comuns de disponibilizar informações na Web, sendo um dos principais meios de disponibilização de informações, especialmente em blogs, wikis e redes sociais. A utilização de conteúdo textual traz diversas vantagens para certos grupos de usuários com deficiência, tais como usuários cegos, uma vez que o conteúdo textual pode ser mais facilmente transformado em síntese de voz ou transformado em outros tipos de saída para usuários.

Entretanto, é necessário tomar os devidos cuidados para tornar conteúdo textual para usuários com outros tipos de deficiência, tais como usuários com baixa visão, ou para usuários com dificuldades específicas de aprendizado, como a dislexia.

Mesmo para usuários cegos que utilizam leitores de tela, é importante que sejam tomados os devidos cuidados para que a marcação de conteúdo textual seja feita de maneira correta para que seja possível entender o conteúdo. Por exemplo, quando se utiliza marcações de negrito ou de ênfase, é importante que elas sejam feitas utilizando as *tags* corretas que deem sentido semântico ao texto. Ao utilizar *tags* como *strong* ou *em(phasis)*, softwares leitores de tela são capazes de detectar que é necessário mudar a forma como uma parte do texto é pronunciada para denotar a ênfase. Por outro lado, caso seja utilizada somente uma formatação com efeito visual (via CSS, por exemplo) sem marcação com semântica associada, os usuários podem não ter acesso ao signficado do texto. No trecho de código (X)HTML a seguir, a marcação de palavras importantes em uma instrução de um trabalho de uma disciplina de Psicologia na Educação:

```
<p> Para a escrita da <strong>Introdução</strong>
do trabalho em grupo, vocês deverão seguir as
orientações ertinentes a uma <strong>Pesquisa
Qualitativa</strong>, pois consideramos que a
construção do trabalho feito por vocês se
constituiu no registro de uma pesquisa.</p>
```

Uma outra questão importante para tornar textos mais acessíveis são atributos de apresentação, como tamanho e texto. Usar tamanhos adequados de texto e permitir que o tamanho possa ser alterado de acordo com as necessidades dos usuários é importante para diversos grupos de usuários. No estudo realizado pela DRC (2004), foi verificado que não só usuários com baixa visão tinham problemas com texto em tamanho muito pequeno, mas também usuários com dislexia e outros grupos de usuários. Esse resultado também foi encontrado em outros estudos com usuários com deficiência (Coyne e Nielsen, 2001; Freire, 2012).

Muitos usuários podem precisar alterar o tamanho do texto em páginas Web utilizando funcionalidades de seus navegadores ou recursos de Tecnologia Assistiva como ampliadores de tela. Para tanto, é importante que os desenvolvedores utilizem marcação que permita o reajuste do tamanho. A utilização de tamanhos com medidas fixas, tais como tamanhos em pixels (px) pode prejudicar o funcionamento dessas funcionalidades, fazendo com que o layout de páginas seja seriamente prejudicado ao fazer tais alterações. A utilização de tamanhos relativos, como em porcentagem ou outras unidades ajustáveis é mais adequada para permitir a adaptação do tamanho. Na Figura 1.5 é ilustrado um exemplo do redimensionamento feito no navegador Firefox sobre o tamanho da fonte de uma barra de navegação de um curso de educação à distância no qual ainda não havia sido feito os ajustes para tamanhos com medidas relativas. O tamanho fixo da barra impede a acomodação dos elementos, e os textos ficam sobrepostos, dificultando a leitura.

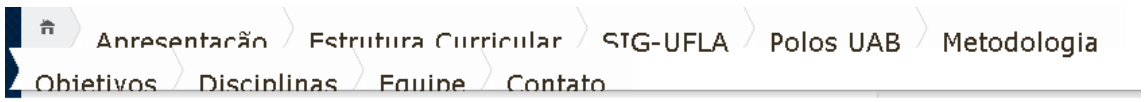


Figura 1.5: Barra de navegação com textos sobrepostos devido a tentativa de ampliação de elementos com medidas absolutas de tamanho

O critério de sucesso 1.4.4 do WCAG 2.0 recomenda que o conteúdo textual de páginas Web possa ter seu tamanho ampliado em até 200% do tamanho original sem perda de conteúdo. Essa regra também é mencionada pela Recomendação 30 do e-MAG 3.0 do governo brasileiro.

A cor do texto também é outro fator muito importante para diversos grupos de usuários. Diversos estudos apontaram para a importância do uso de contrastes entre cor de texto e cor de fundo para usuários com baixa visão, por exempo (e.g. DRC, 2004; Freire, 2012; Theofanos e Redish, 2005). Muitos desses usuários podem ter problemas para ler conteúdo se o contraste não for grande o suficiente. Alguns desses usuários podem precisar alterar os esquemas de cores para cores invertidas, cores de alto contraste, ou utilizar somente cores em preto e branco. Deve-se garantir que os usuários possam ler o conteúdo adequadamente mesmo quando é necessário fazer tais alterações por meio de recursos de Tecnologia Assistiva.

Na Figura 1.6, é ilustrado um exemplo de um texto em uma página de um curso de educação à distância com as disciplinas a serem cursadas pelos alunos na qual ainda não haviam sido feitos os ajustes para melhoria da acessibilidade. O contraste entre um cinza escuro na cor de fundo do número total de horas ao final da tabela e o texto em preto dificulta a leitura por pessoas com baixa visão, assim como o texto em cinza escuro sobre um fundo em azul escuro.

1º MÓDULO			
Disciplina	Código da Disciplina	Carga Horária	Créditos
Educação a Distância	DGAE101	30	2
Teorias da Administração I	DGAE103	60	4
Metodologia de Estudo e de Pesquisa em Administração	DGAE106	60	4
Matemática Básica	DGAE102	60	4
Filosofia e Ética	DGAE104	60	4
Psicologia Oganizacional	DGAE105	60	4
Seminário Integrador	DGAE107	30	2
Total de horas/aula: 360			

Figura 1.6: Exemplo de problemas com contraste de cores na tabela de disciplinas de um curso à distância

Na Figura 1.7, é ilustrada a figura da mesma tabela mostrada na Figura 1.6 após a exibição utilizando um software ampliador de telas que aplica uma inversão nas cores. Verifica-se que nesse caso também é difícil visualizar textos em algumas partes da figura, em particular para usuários com baixa visão. É muito importante que desenvolvedores façam testes utilizando suas páginas com diferentes configurações de

cores para garantir que haja contraste bom o suficiente para que a página possa ser visualizada em diferentes configurações.

1º MÓDULO			
Disciplina	Código da Disciplina	Carga Horária	Créditos
Educação a Distância	DGAE101	30	2
Teorias da Administração I	DGAE103	60	4
Metodologia de Estudo e de Pesquisa em Administração	DGAE106	60	4
Matemática Básica	DGAE102	60	4
Filosofia e Ética	DGAE104	60	4
Psicologia Oganizacional	DGAE105	60	4
Seminário Integrador	DGAE107	30	2
Total de horas/aula: 360			

Figura 1.7: Exemplo de problemas com contraste de cores na tabela de disciplinas de um curso à distância com exibição com cores invertidas

Nas recomendações do WCAG 2.0, recomenda-se que seja feito um cálculo da diferença entre as cores de frente e fundo considerando sua luminosidade, de forma que a proporção entre a cor de frente e fundo seja de 3:1, 4,5:1 ou 7:1, dependendo do tamanho do texto e do destaque e do nível de conformidade com as guidelines que se deseja atingir. A recomendação 28 do e-MAG 3.0 recomenda que o nível de contraste seja de no mínimo 3:1 ou 4,5:1 dependendo do tamanho e do destaque.

Entretanto, é importante tomar cuidado com a utilização desses critérios como única recomendação para utilização de cores em textos em páginas. Estudos com usuários (Freire, 2012) mostraram que há diversos casos em que usuários com baixa visão encontraram problemas com o contraste de cores mesmo em elementos que estavam de acordo com os requisitos mínimos exigidos pelas recomendações de acessibilidade. Desta forma, é importante que os desenvolvedores não se contentem em fornecer somente o mínimo necessário em termos de contraste.

Contraste de cor também é importante para usuários com dislexia. Muitos usuários podem encontrar problemas para ler textos com fonte em preto em fundo branco. Os motivos para os problemas são diferentes daqueles encontrados por usuários com baixa visão. Para esses usuários, certas configurações de cor com muito brilho podem causar perturbações visuais que dificultam a decodificação do texto no momento da leitura. Muitos usuários podem preferir mudar a cor de fundo para uma outra cor mais fosca que facilite a leitura. Desta forma, é importante que os desenvolvedores utilizem codificações de cores que permitam a alteração por recursos de Tecnologias Assistivas ou de navegadores Web. Na Figura 1.8, é ilustrado um exemplo de alteração

da cor de fundo de uma página de uma disciplina de um curso à distância originalmente com fundo branco para um fundo mais fosco por meio do navegador Web.

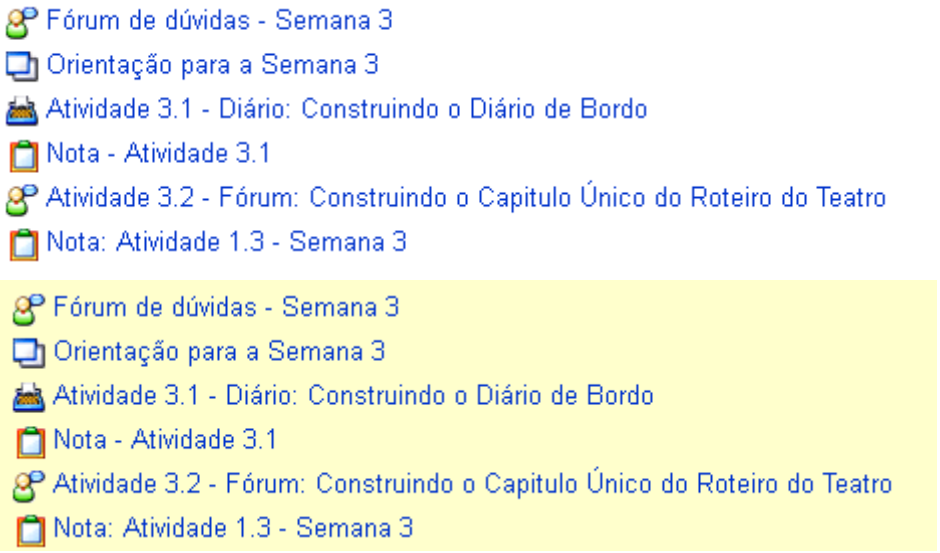


Figura 1.8: Exemplo de alteração de cor de fundo para cor mais fosca para facilitar leitura para usuários com dislexia

Usuários com dislexia também podem encontrar dificuldades para leitura de texto com certos tipos de formatação. Infelizmente, as recomendações de acessibilidade do WCAG 2.0 e do e-MAG 3.0 não incluem muitas recomendações para esse público. Entretanto, outras recomendações incluem importantes indicações sobre como tornar o conteúdo de textos mais acessíveis a usuários com dislexia (Bradford 2005, British Dyslexia Association 2011, Kolatch 2000, Zarach 2002). Essas recomendações incluem questões relacionadas a tamanho de fonte e cor de fundo, e outras recomendações relacionadas a evitar o uso de itálico (que dificulta a decodificação dos caracteres), evitar alinhamento justificado (que despadroniza o espaçamento entre caracteres), evitar parágrafos longos, dentre outros. É muito importante que desenvolvedores permitam que sejam feitas adaptações no conteúdo para atender a diferentes usuários.

A quantidade de texto e a organização deste também é fundamental para diversos grupos de usuários. Estudos encontrados na literatura (DRC, 2004; Freire, 2012) indicaram que muitos usuários podem encontrar dificuldade quando há uma quantidade muito grande informações exibidas de uma só vez, ou quando não há um agrupamento adequado entre as informações para facilitar a leitura.

A leitura do texto por usuários com dificuldades especificas de aprendizado, por usuários com deficiência cognitiva ou por usuários com baixo letramento também é impactada pela dificuldade de leitura. É importante escrever o texto levando em conta as diferentes necessidades do público-alvo. O critério de sucesso 3.1.5 do WCAG 2.0 e a recomendação 25 do e-MAG 3.0 dão recomendações para facilitar a leitura e compreensão de texto, com indicações que incluem:

- Desenvolver apenas um tópico por parágrafo;
- Utilizar sentenças organizadas de modo simplificado para o propósito do conteúdo (sujeito, verbo e objeto, preferencialmente);
- Dividir sentenças longas em sentenças mais curtas;

- Evitar o uso de jargão, expressões regionais ou termos especializados que possam não ser claros para todos;
- Utilizar palavras comuns no lugar de outras pouco familiares;
- Utilizar listas de itens ao invés de uma longa série de palavras ou frases separadas por vírgulas;
- Fazer referências claras a pronomes e outras partes do documento;
- Utilizar, preferencialmente, a voz ativa.

1.4.2. Imagens

O uso de imagens em sítios Web pode ter diferentes finalidades. Páginas podem conter imagens que são *informativas* ou imagens que são meramente *decorativas*. O uso de imagens que passam informações importantes para os usuários merece especial atenção para garantir sua acessibilidade. Essas imagens podem passar informações importantes, ou até mesmo ser utilizadas para elementos interativos, como links, por exemplo.

Usuários cegos que utilizam leitores de tela são seriamente afetados por problemas com baixa acessibilidade de imagens. Os leitores de tela só são capazes de sintetizar em voz conteúdo na forma de texto. Dessa forma, é necessário informar um texto alternativo que descreva o conteúdo de imagens que contenham informações em sítios Web de maneira eficaz para esses usuários. Infelizmente, é muito comum que desenvolvedores deixem o texto alternativo de imagens sem nenhuma informação ou que coloquem o nome do arquivo de uma imagem em sua descrição.

É importante que a descrição das imagens contenha todas as informações que o desenvolvedor ou criador de conteúdo deseje que seja transmitida para seus usuários. Em sítios Web com conteúdo educacional, esse requisito é de particular importância, uma vez que é comum o uso de imagens para transmitir conceitos em gráficos, diagramas e ilustrações de conteúdo educativo.

A inserção de texto alternativo em (X)HTML é feita por meio dos atributos *alt* e *longdesc* do elemento img. O atributo *alt* é utilizado para uma descrição mais curta sobre o que a imagem contém, enquanto que o atributo *longdesc* pode ser utilizado caso seja necessário fornecer uma descrição mais longa.

Na Figura 1.9, é mostrado um exemplo de uma propaganda contra o fumo feita pelo Ministério da Saúde do Brasil utilizada em uma disciplina de Leitura e Produção de Textos de um curso à distância. Para a codificação dessa figura e descrição em texto alternativo do conteúdo para acesso por usuários cegos, ela poderia ser codificada da seguinte forma em uma página Web:

```
<img src="propaganda_cigarro.png"
alt="Cartaz com título: O que eles vendem não é o que você leva. Fumar causa câncer na boca. Cigarro faz mal até na propaganda"
longdesc="Autor: Brasil. Ministério da Saúde. Fonte: [Brasília]: Disponível em: http://bvsms-bases.saude.bvs.br/cgibin/wxis.exe/iah/ms/Acesso em: agosto de 2013" />
```

Figura 1.9: Exemplo de imagem e texto alternativo descritivo utilizado em uma disciplina de Leitura e Produção de Textos
Fonte: http://bvsms-bases.saude.bvs.br/cgibin/wxis.exe/iah/ms/

No caso de imagens *decorativas* que não fornecem nenhuma informação para o usuário, e tem a finalidade única de decoração, pode ser fornecido um texto alternativo em branco no código (X)HTML. Desta forma, ao encontrar uma imagem com um texto alternativo em branco, um leitor de telas para cegos irá simplesmente saltar a imagem e continuar a leitura do texto. O exemplo seguinte mostra como uma imagem decorativa poderia ser codificada:

```
<img src="imagemdecorativa.png" alt="" />
```

O fornecimento de conteúdo alternativo textual para imagens e outros conteúdos não textuais é tratado pelo critério de sucesso 1.1.1 do WCAG 2.0 e pela recomendação 1 do e-MAG 3.0.

1.4.3. Áudio e Vídeo

O uso de recursos multimídia em sítios Web é cada vez mais comum, em particular com a difusão de serviços como o YouTube e do uso de recursos multimídia em redes sociais baseadas na Web. Em diversos países, até mesmo canais de televisão utilizam a Web como plataforma para disponibilização de parte de sua programação gravada ou ao vivo para seus usuários. No contexto educacional, o uso de video-aulas tem se tornado cada vez mais comum, principalmente em casos de cursos à distância.

Contudo, é muito importante que os produtores de conteúdo multimídia estejam atentos à acessibilidade para pessoas com deficiência. Diversos usuários podem ter acesso seriamente restrito a conteúdo multimídia se não tiverem os recursos para comunicação adequados.

A adequação de conteúdo multimídia para televisão tem sido alvo de muitos esforços, em particular na elaboração de legislação que reforça a necessidade da produção de conteúdo com legendas, interpretação em língua de sinais e de audiodescrição para pessoas com deficiência visual.

1.4.3.1. Legendas e Tradução em Língua de Sinais

Um grupo de usuários que precisa de atenção especial para conteúdo multimídia é o de usuários com deficiência auditiva, tanto os que tem alguma perda auditiva quanto aqueles considerados surdos profundos. É importante que sejam feitas adaptações no conteúdo para que esses usuários tenham acesso ao conteúdo disponível em áudio. Os meios mais comuns para fazer essas adatapções são o uso de legendas, tradução do conteúdo em Língua Brasileira de Sinais (LIBRAS) ou a transcrição das falas em um conteúdo unicamente em áudio na forma de texto.

A disponibização de conteúdo falado com legendas é fundamental para usuários que não podem ouvir. Legendas são essenciais para usuários com perda auditiva ou com surdez profunda. É muito importante que as legendas sejam preparadas apropriadamente para que não haja imprecisões no conteúdo passado a pessoas surdas.

Deve ser feito um planejamento para a geração de legendas dependendo do tipo de transmissão de conteúdo em vídeo que é feita, seja ao vivo ou com vídeos pré-gravados. Em transmissões ao vivo, a confecção das legendas pode ser mais desafiadora do que em videos pre-gravados, necessitando da presença de profissionais treinados que possam transcrever o que é dito rapidamente para disponibilização das legendas.

Mesmo em sessões pre-gravadas, é importante atentar-se às estratégias utilizadas para transcrever o conteúdo que será disponibilizado nas legendas. Para vídeoaulas utilizadas em educação à distância, por exemplo, apesar da existência de roteiros para a gravação dessas aulas, professores precisam ter liberdade para conduzir suas aulas e fazer intervenções de acordo com as necessidades do momento. Desta forma, pode ser mais adequado confeccionar as legendas após a gravação das sessões, de forma a retratar fielmente nas legendas o que aconteceu na sessão, e não só o que estava presente no roteiro.

As legendas normalmente são colocadas na parte inferior do vídeo, podendo ser ocasionalmente posicionadas na parte superior. É importante que as cores da legenda tenham um bom contraste com a cor de fundo do vídeo, para que os usuários possam ler confortavelmente. Na Figura 1.10, é mostrado um exemplo de uma cena de um vídeo com uma legenda na parte inferior da tela contendo as falas das personagens no vídeo. O vídeo também tem tradução em Língua Brasileira de Sinais (LIBRAS).

A disponibilização de tradução em Língua Brasileira de Sinais é muito importante para usuários surdos que tem a LIBRAS como primeira língua. Apesar de grande parte dos surdos poderem utilizar legendas em língua portuguesa, a disponibilização de tradução em sua primeira língua é preferida por esses usuários e é

mais efetiva para a comunicação. Como mostrado na Figura 1.10, o vídeo de um intérprete LIBRAS pode ser posicionado junto ao vídeo ou sobreposto a ele em um dos cantos da tela.

Figura 1.10: Exemplo de uma cena de vídeo com legenda e tradução em Língua Brasileira de SINAIS (LIBRAS)
Fonte: http://www.youtube.com/watch?v=KWzHiZZUc20, acesso em agosto de 2013
Dicas de Convivência, Instituto Mara Gabrilli

A ABNT, por meio da norma NBR 15290 (ABNT, 2005) provê um conjunto de recomendações para acessibilidade em comunicação na televisão. Dentre as recomendações estão diretrizes para a gravação de "janela de LIBRAS" para programas de TV que podem ser utilizados como guia para a confecção dessa janela para vídeos disponibilizados em sítios Web.

De acordo com a NBR 15290, a recomendação para a gravação da janela LIBRAS é que o estúdio para a gravação da imagem do intérprete tenha espaço suficiente para que o intérprete não fique muito próximo ao fundo, para evitar o aparecimento de sombras, que a iluminação seja suficiente e adequada para que a câmera possa capturar o intérprete com qualidade, ficando apoiada em um tripé fixo, e que haja marcação no solo para delimitar o espaço de movimentação do intérprete.

Em relação à janela de LIBRAS, a NBR 15290 recomenda que os contrastes sejam nítidos, quer em cores ou em fundo preto e branco, e que haja contraste entre o pano de fundo e os elementos do intérprete. É recomendado também que o foco da janela abranja toda a movimentação e gesticulação do intérprete, além de haver iluminação adequada para evitar o aparecimento de sombras nos olhos e/ou seu ofuscamento.

Quando a interpretação ocorre em recortes no vídeo, a recomendação da NBR 15290 é que a altura da janela seja no mínimo metade da altura da área utilizada para o

vídeo todo, e que a janela ocupe no mínimo um quarto da largura da área do vídeo. Sempre que possível, o recorte deve estar localizado de modo a não ser encoberto pela tarja preta da legenda.

Em relação à interpretação em LIBRAS em si, a NBR 15290 recomenda que a vestimenta, pele e cabelo do intérprete sejam contrastantes entre si e com o fundo, e que sejam evitados fundo e vestimenta em tons próximos ao da pele do intérprete. Também recomenda-se que a janela ou recorte para LIBRAS não sobreponha outras imagens.

A disponibilização de legendas para vídeos é parte das recomendações do WCAG 2.0, nos critérios de sucesso 1.2.2 (somente para pré-gravados) e 1.2.4 (para vídeos transmitidos ao vivo), bem como pela recomendação 33 do e-MAG 3.0.

A transcrição de conteúdo em áudio na forma de texto é tratada pela recomendação 34 do e-MAG 3.0 e pelos critérios de sucesso 1.2.1, 1.2.2 e 1.2.6 do WCAG 2.0.

Em relação à disponibilização de tradução em língua de sinais, a recomendação 33 do e-MAG 3.0 também recomenda que haja interpretação em LIBRAS para conteúdo de vídeo. O critério de sucesso 1.2.6 do WCAG 2.0 também recomenda que haja tradução em língua de sinais para videos pré-gravados.

1.4.3.2. Audiodescrição

Fornecer a descrição de imagens que estão na tela em vídeos para usuários com deficiência visual é muito importante para que eles possam ter acesso ao conteúdo de vídeos. Em vídeos que não tem audiodescrição, usuários cegos podem perder muitas informações que são apresentadas só visualmente, como cenas sem diálogos, quem são as personagens falando em cada cena, e outras informações importantes que podem auxiliar a compreender a mensagem contida em vídeos.

A audiodescrição é feita por meio de uma narração nos intervalos em que não há falas em vídeos com descrições de personagens, cenário ou fatos que ocorrem e não são narrados pelas falas.

A utilização de audiodescrição ainda é bastante restrita, até mesmo na televisão. Entretanto, é cada vez mais comum a disponibilização de filmes em cinemas com audiodescrição, e as redes de televisão deverão gradativamente aumentar a quantidade de programas que tem audiodescrição.

A relevância da audiodescrição foi um resultado importante encontrado em um estudo realizado com usuários com deficiência visual (Freire, 2012; Power et al., 2012). Nesse estudo, problemas relacionados à falta de audiodescrição estiveram entre os problemas com maior nível de gravidade encontrados por usuários cegos. Isso mostra que, apesar dos custos associados à produção de vídeos com audiodescrição, esse recurso é considerado fundamental para que usuários cegos possam ter acesso ao conteúdo.

Na Figura 1.11 são mostradas duas cenas do vídeo “Dicas de Convivência”, produzido pelo Instituto Mara Gabrilli. Na primeira, é mostrado um diálogo entre um homem e uma mulher em uma cadeira de rodas. Na segunda, dois homens carregam a mulher para cima de uma escada, mas não há nenhum diálogo. Para usuários cegos que

utilizem esse vídeo, caso não houvesse audiodescrição, eles não teriam acesso ao que acontece no vídoe nos momentos em que não há falas.

(a) Imagem de cena de vídeo em que homem conversa com moça cadeirante

(b) Imagem de cena em que dois homens carregam a mulher na cadeira de rodas, não há conversa no vídeo. Neste trecho, a audiodescrição é fundamental para explicadr o que acontece no vídeo.

Figura 1.11: Dois instantes de um vídeo indicando a importância de audiodescrição de trechos em que não há falas. Fonte: http://www.youtube.com/watch?v=KWzHiZZUc20, acesso em agosto de 2013 Dicas de Convivência, Instituto Mara Gabrilli

Para tornar o vídeo mais acessível, ele foi gravado com audiodescrição das cenas e eventos que acontecem no vídeo. A seguir está transcrita a sequência de falas com a narração da audiodescricão intercalada nos momentos de silêncio das personagens:

Rapaz: Oi, tudo bem?

Narradora: Rapaz vem até uma mulher que está na cadeira de rodas.

Rapaz: Tá precisando de ajuda, né? Vou te ajudar.

Mulher: Não, não precisa.

Rapaz: Eu vou resolver seu problema.

Mulher: Ô mocinho, é que tem uma rampa aqui, e eu estou só esperando uma amiga, com quem eu combinei de me encontrar exatamente aqui, mas não precisa me ajudar.

Rapaz: Não custa nada, vai? Ó, uma mão lava a outra. Hoje é você amanhã pode ser eu, tudo bem, tá? Eu vou te ajudar, eu vou resolver seu problema, OK?

Mulher: Eu entendi, é que realmente não precisa.

Rapaz: Amigo, me dá uma forcinha aqui?

Narradora: Ele acena para um rapaz que passa, e juntos carregam a cadeira de rodas para um patamar um pouco mais alto.

Rapaz: Custou alguma coisa? Que isso, bom ajudar os outros, né?

Narradora: Tela escrita: Dê uma ajudinha a si mesmo, reveja seus conceitos. A garota desce a rampa e volta para o lugar onde estava.

A norma NBR 15290 da ABNT (2005) também fornece recomendações para a confecção de audiodescrição para vídeos. Segundo a recomendação, a descrição em áudio de imagens e sons deve transmitir de forma sucinta o que não pode ser entendido sem a visão. A norma recomenda que se evite a monotonia e exageros. A descrição fornecida deve ser compatível com o tipo de vídeo, sendo mais objetiva para vídeos para adultos e poética para vídeos infantis. Em filmes de época ou outros vídeos mais complexos, devem ser fornecidas informações que facilitem a compreensão do vídeo. A norma também sugere que se evite a subjetividade excessiva na descrição.

A disponibilização de audiodescrição de vídeos é indicada pelos critérios de sucesso 1.2.3, 1.2.5 e 1.2.7 do WCAG 2.0 e pela recomendação 35 do e-MAG 3.0.

1.4.4. Cabeçalhos e elementos estruturais de sítios Web

A utilização de cabeçalhos e outros elementos estruturais de páginas Web é muito importante para que usuários com deficiência possam interagir eficientemente. Diversos recursos de Tecnologia Assistiva oferecem recursos que permitem que os usuários tenham acesso a diferentes partes de uma página por meio do uso desses elementos estruturais.

1.4.4.1. Cabeçalhos

Os leitores de tela para usuários cegos normalmente lêem o conteúdo de cima para baixo, e da esquerda para direita. Entretanto, ao navegar na Web, a maioria dos usuários cegos não espera que todo o conteúdo seja lido para que eles encontrem a informação desejada. Esses usuários usam de uma série de recursos para saltar por diferentes partes da página e explorá-la para encontrar a informação que desejam.

Uma das possibilidades utilizadas por usuários cegos é utilizar teclas de atalho para saltar por elementos marcados como cabeçalhos. Com isso, os usuários podem saber quais são as diferentes seções e o conteúdo disponível em uma página para reduzir o escopo de onde eles precisam procurar pelo conteúdo que desejam encontrar.

Entretanto, para que isso seja possível, é necessário que os elementos estruturais estejam marcados corretamente utilizando a sintaxe de (X)HTML adequada. Por exemplo, um elemento de cabeçalho marcado com a *tag* <p> e formatado com negrito e fonte grande pode parecer visualmente como um cabeçalho. Contudo, um software

leitor de telas não é capaz de identificar que ele é um cabeçalho a não ser que ele esteja marcado com as *tags* corretas de <h1>, <h2> ou outro nível, de acordo com sua importância. Na Figura 1.12 é ilustrado um exemplo de uma página do curso à distância de Filosofia da UFLA com notícias relacionadas. Ao utilizar marcação de cabeçalho de segundo nível <h2> (o primeiro nível é o título), a página possibilita que usuários cegos "saltem" pelos conteúdos sem ter que esperar a leitura da página toda.

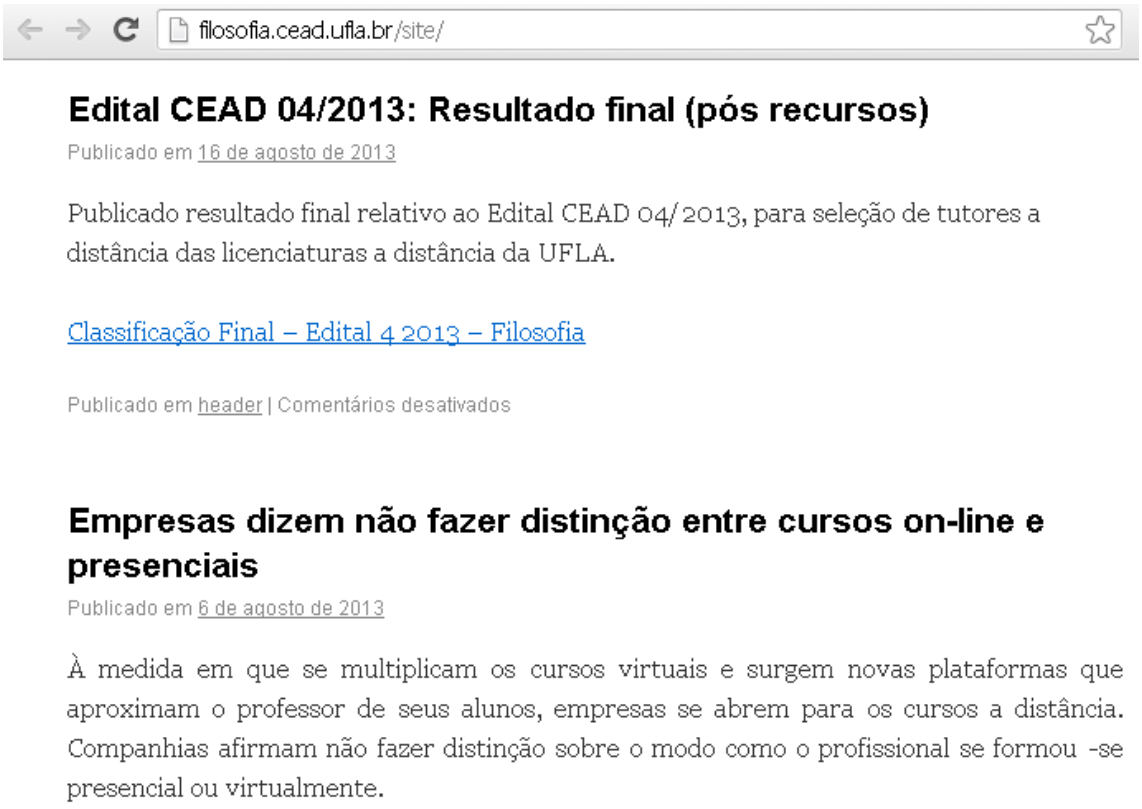


Figura 1.12: Página de notícias de curso à distância com cabeçalhos identificando as notícias.

Para que a funcionalidade de identificação de cabeçalhos seja possível, é utilizado o seguinte código (X)HTML para marcar os cabeçalhos da página:

```
<h2><a href="http://filosofia.cead.ufla.br/site/?p=866">
Edital CEAD 04/2013: Resultado final (pós
recursos)</a></h2>
```

A utilização correta de cabeçalhos em páginas é indicada pela recomendação 3 do e-MAG 3.0 e pelo critério de sucesso 1.3.1 do WCAG 2.0.

O critério de sucesso 2.4.10 do WCAG 2.0 também recomenda que os cabeçalhos sejam utilizados adequadamente para demarcar as diferentes seções de uma página. Dessa forma, ao ler os títulos dos cabeçalhos, é possível ter uma visão geral sobre as seções e o conteúdo disponível nas páginas.

1.4.4.2. Listas

Da mesma forma que cabeçalhos, a marcação correta de elementos como listas também é importante para que usuários de recursos de Tecnologia Assistiva tenham acesso à organização do conteúdo em uma página. É muito importante que elementos de listas (ordenadas ou não) sejam marcados utilizando as *tags* corretas do (X)HTML de <ul> para listas não ordenadas, <ol> para listas ordenadas, e <li> .... para elementos contidos em listas.

Em disciplinas de cursos à distância utilizando o Moodle, um exemplo do uso de listas é a estruturação dos tópicos e organização das etapas dos cursos. A seguir, é mostrado o código com a lista das semanas e períodos de uma disciplina de História da Filosofia:

```
<ol>
<li>Semana 1: 18/03/2013 a 25/03/2013</li>
....
<li>Semana 2: 25/03/2013 a 01/04/2013</li>
...
<li>Semana 3: 01/04/2013 a 08/04/2013</li>
...
<li>Semana 4: 08/04/2013 a 15/04/2013</li>
...
</ol>
```

A utilização correta de listas é um dos requisitos do critério de sucesso 1.3.1 do WCAG 2.0.

1.4.4.3. Tabelas

Tabelas podem prover um meio muito eficiente de exibir informações visualmente de forma organizada. Entretanto, a apresentação de dados em tabelas por meio de áudio pode ser uma tarefa muito árdua, principalmente para usuários cegos que utilizam leitores de tela. A leitura de tabelas é feita de forma *linearizada*. Os leitores de tela leem os dados linha por linha. No caso de um dado no meio de uma tabela com uma sequência de números, a leitura de um número fora de contexto sem associar aos respectivos cabeçalhos pode fazer com que ele não tenha sentido nenhum para quem o ouve.

Desta forma, é muito importante que a marcação de elementos de tabela seja feita de forma adequada, principalmente identificando os elementos que são cabeçalhos na tabela. Em código (X)HTML, isso pode ser feito utilizando os elementos <*th*> para células que são cabeçalhos, tanto de linhas quanto de colunas. Da mesma forma que na marcação de cabeçalhos no texto, não basta utilizar um elemento de marcação de célula convencional <td> com uma formatação que indique que ele seria um cabeçalho de

linha/coluna somente visualmente. É importante que haja uma marcação no código que indique a semântica do texto em relação ao seu papel na tabela. Assim, ao ler o conteúdo de uma célula que só contém um número, por exemplo, o software leitor de telas pode buscar a coluna ou linha correspondente àquela célula que pode fornecer um cabeçalho para identificar o que significa aquele dado.

Na Figura 1.13, é ilustrado um exemplo de uma tabela contendo a lista de disciplinas a serem realizadas em um curso à distância de Filosofia no segundo módulo, contendo o número de créditos, carga-horária e se a disciplina é obrigatória ou não.

Módulo 2 (2012/2)

Disciplinas	Créditos	Carga-Horária	Natureza
Ética I	4	60	Obrigatória
Lógica I	4	60	Obrigatória
Lógica II	4	60	Obrigatória
História da Filosofia Medieval I	4	60	Obrigatória
Total	**16**	**240**	

Figura 1.13: Exemplo de tabela de disciplinas de um semestre de um curso de educação à distância em Filosofia

A codificação da tabela para identificar os cabeçalhos de linha e coluna precisam identificar cada um dos itens. Também é importante fornecer um resumo para a tabela, descrevendo sua organização para que usuários cegos possam saber como utilizá-la.

```
<h2>Módulo 2 (2012/2)</h2>
<table summary="Tabela de disciplinas, contendo nome da disciplina, número de créditos, carga-horária e natureza">
<tr>    <th> Disciplinas</th>
<th>Créditos</th>
<th>Carga-Horária</th>
<th>Natureza</th>          </tr>
<tbody>
<tr>       <th>Ética  I</th><td>4</td>     <td>60</td>
<td>Obrigatória</td>        </tr>
<tr>    <th>Lógica  I</th>    <td>4</td>    <td>60</td>
<td>Obrigatória</td>          </tr>
<tr>    <th>Lógica  II</th>    <td>4</td>   <td>60</td>
<td>Obrigatória</td>       </tr>
```

```
<tr>  <th>História da Filosofia Medieval I</th>
<td>4</td>       td>60</td>       <td>Obrigatória</td>
</tr>
<tr>  <th>Total</th>      <td>16</td>      <td>240</td>
<td> -  </td>        </tr>
</tbody>
</table>
```

Com essa codificação, ao ler um número “4”, por exemplo, o leitor de tela seria capaz de associá-lo ao cabeçalho de coluna “Créditos” e à disciplina “História da Filosofia Medieval I”, por exemplo.

Para tabelas mais complexas, com mais de um nível, por exemplo, pode ser necessário utilizar outros recursos para marcação que relacionem explicitamente a quais cabeçalhos de linha e coluna uma certa célula se relaciona.

A utilização de marcação correta para tabelas é recomendada pelo critério de sucesso 1.3.1 do WCAG 2.0 e pela recomendação 23 do e-MAG 3.0.

Para efetuar a leitura de tabelas, leitores de tela dedicam um grande esforço para contextualizar onde está cada célula de uma tabela em relação aos cabeçalhos e onde inicia cada célula, por exemplo. Desta forma, quando as tabelas são utilizadas com o único fim de posicionar elementos visualmente na tela, usuários cegos podem ter a interação prejudicada com a leitura de diversos elementos que anunciam o início de uma célula e tentam relacionar com um cabeçalho quando, na verdade, o elemento lido não corresponde a uma célula de tabela, mas sim a algo que está alinhado em uma certa posição da tela.

Tabelas não devem ser utilizadas somente para efeitos de layout para evitar que isso aconteça. A forma mais adequada para tratar de questões de alinhamento e posicionamento de elementos é com elementos para agrupar itens e efetuar o posicionamento com codificação em folhas de estilo CSS. A indicação para evitar o uso de tabelas para diagramação é dada pela recomendação 7 do e-MAG 3.0.

1.4.5. Links e Navegação

A estrutura de links e de navegação em sítios Web forma um dos principais alicerces de suas características baseadas em hipertexto. Para os usuários, é fundamental que eles consigam utilizar as estruturas de navegação de maneira eficiente para transitar entre diferentes páginas e ter acesso à informação que desejam.

Uma boa estrutura de navegação é essencial para que qualquer usuário possa ter um acesso satisfatório a páginas Web. Entretanto, problemas com navegação podem ter graves consequências para usuários com certos tipos de deficiência. Estruturas de navegação confusas podem impactar usuários cegos, por exemplo, ainda mais gravemente do que outros usuários, uma vez que o uso do leitor de telas consome mais tempo para esses usuários para localizar as páginas onde querem chegar.

Por exemplo, muitas páginas utilizam uma barra de navegação que é fixa em todas as páginas de um sítio Web. Ao navegar para diferentes partes de um sítio Web, usuários cegos precisam de alguma forma para “saltar” o conteúdo que eles já ouviram antes, e ir logo para a parte da página que contém o conteúdo que eles desejam. Uma

das formas de atingir esse objetivo é prover um link no início da página que "salte" para a posição onde se inicia o conteúdo, conforme a recomendação 6 do e-MAG 3.0 e técnicas do critério de sucesso 1.4.1 do WCAG 2.0. A seguir, é descrito o código (X)HTML para a implementacão deste tipo de link:

```
<a href="#conteudo">Ir para conteúdo</a>
<p><a name="conteudo" id="conteudo" accesskey="1">Início do conteúdo da página</a></p>
```

A utilização de links com destino claramente identificado é muito importante para diversos usuários de sítios Web, mas é fundamental para usuários cegos com leitores de tela. Muitos desses usuários navegam por um sítio utilizando a tecla TAB para ter uma visão geral dos links e botões, ou mesmo usam uma funcionalidade dos programas leitores de tela que só listam os links em uma página.

Quando os usuários leem a lista de links fora de contexto, é muito importante que o destino de um link seja claramente identificado. Por exemplo, quando o usuário cego encontra um link que lê apenas "clique aqui" ou "leia mais", não é possível saber para onde ele irá. Na Figura 1.14 é ilustrado um trecho de uma página que não havia ainda sido adequada a requisitos de acessibilidade com uma lista de cursos à distância oferecidos e links para vídeos e informações detalhadas. Os links são identificados somente por "clique aqui". Para saber para onde vão esses links, é necessário que o usuário veja o contexto do parágrafo onde ele se encontra e o cabeçalho que o antecede com o nome do curso.

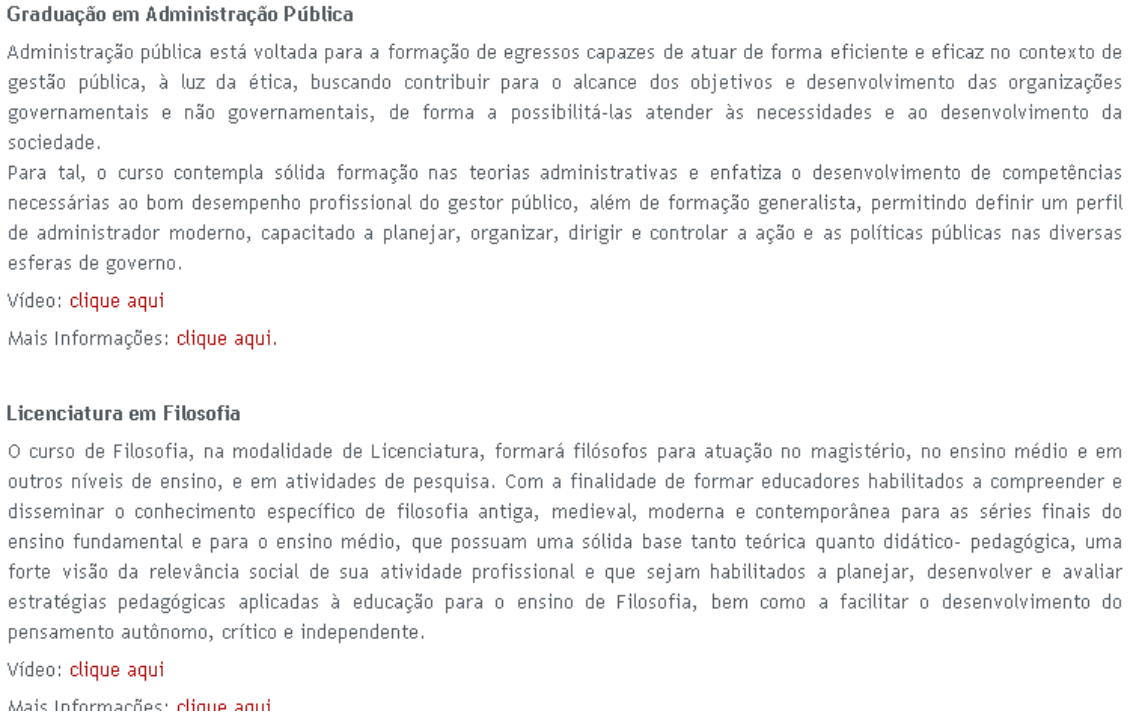


Figura 1.14: Lista de cursos com vídeos e informações detalhadas com links identificados somente por "clique aqui"

Ao utilizar uma lista que só mostra os links da página para facilitar a navegação, usuários cegos iriam ver uma série de links “clique aqui”, sem saber que tipo de informação será aberta ao clicar no link. Na Figura 1.15, é ilustrada a simulação da exibição de uma lista de links pelo plugin do Firefox Fangs, que mostra como seria exibida a lista de links para um usuário de leitor de telas.

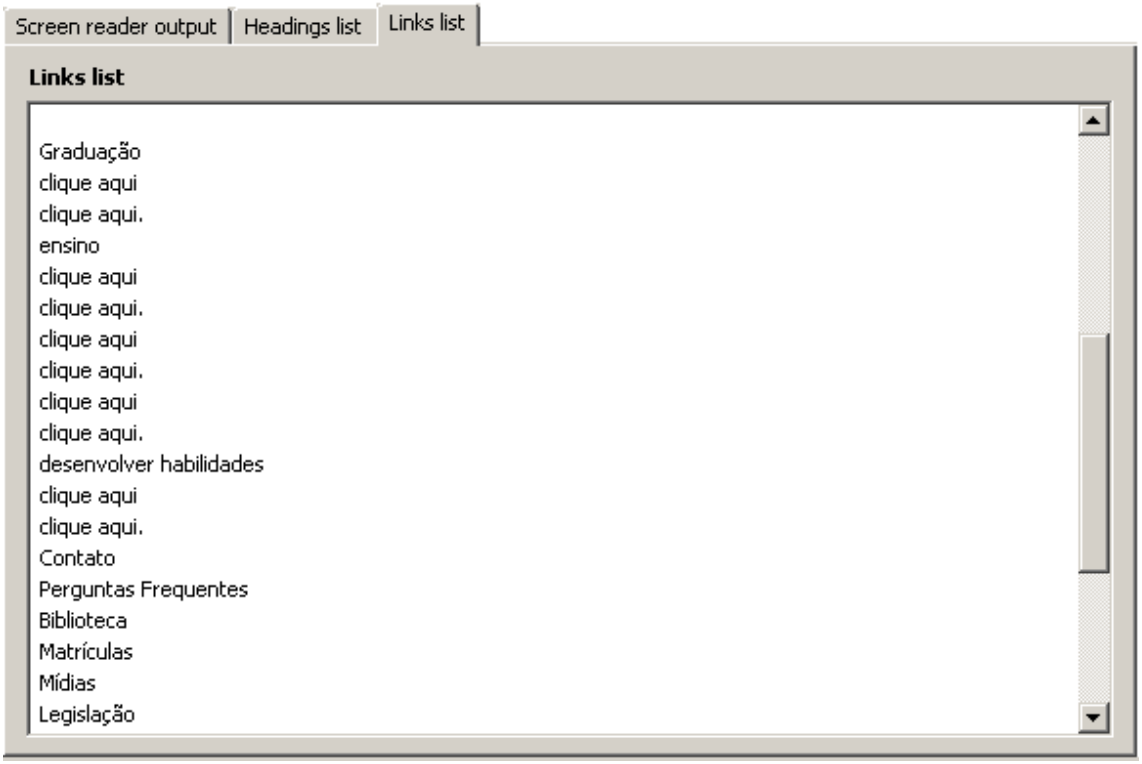


Figura 1.15: Lista de links mostrada pelo plugin Fangs do Firefox, com diversos links identificados somente como “clique aqui”

Diretrizes para criação de links acessíveis são dadas nos critérios de sucesso 2.4.4 e 2.4.9 do WCAG 2.0 e pela recomendação 19 do e-MAG 3.0. É importante ressaltar, entretanto, que para o nível de conformidade A, o WCAG 2.0 permite que somente o critério de sucesso 2.4.4 seja satisfeito. Nesse critério, é permitido que um link não tenha que necessariamente fazer sentido quando lido fora de contexto. O critério permite que o destino do link seja determinado ao ler uma informação de contexto, como parágrafos, listas, cabeçalhos e outros. Contudo, em estudos realizados por Power et al. (2011), foi verificado que o uso dessas técnicas não é tão eficaz quanto a disponibilização de links que façam sentido mesmo quando fora de contexto.

Uma outra característica importante de links é evitar que novas janelas sejam abertas sem o conhecimento dos usuários. Esse problema impacta principalmente pessoas com deficiência visual (cegos ou com baixa visão), que nem sempre tem uma visão geral de todas as janelas abertas, seja porque o ampliador de telas nem sempre mostra todas as partes da tela ou porque o usuário utiliza um leitor de tela com voz. Nesses casos, quando novas janelas são abertas sem avisar, os usuários podem ficar

perdidos sem saber qual é a janela atual, tentar voltar a páginas que já visitou sem sucesso, ou mesmo fechar janelas indevidamente e perder informações importantes.

1.4.6. Formulários e elementos interativos

O uso de elementos interativos e formulários em páginas Web provêem muitas capacidades para que diversas funcionalidades sejam implementadas, fazendo com que essas páginas tenham recursos para funcionar como aplicativos utilizando a Web como plataforma. Muitos recursos podem ser adotados utilizando linguagens de programação que são executadas tanto no servidor quanto no navegador do computador cliente, utilizando tecnologias como JavaScript, Flash e outros.

Prover meios para que usuários com diferentes tipos de deficiência utilizando diferentes tecnologias para entrada e saída de dados possam utilizar tais recursos interativos é de fundamental importância para a acessibilidade de aplicações Web.

1.4.6.1. Acesso utilizando o teclado

É muito importante que usuários que interagem utilizando somente o teclado sejam capazes de ter acesso a qualquer funcionalidade em uma página. Caso alguma funcionalidade só possa ser operada utilizando o mouse, usuários cegos que só utilizam teclados, ou alguns usuários com deficiência motora que não utilizam o mouse podem ter sérios problemas para utilizar certas funcionalidades.

Na Figura 1.16 é ilustrado um exemplo de um vídeo com um botão para acionar que só pode ser utilizado com o mouse. Sem poder ter acesso a esse botão com o teclado, usuários de leitor de tela ou usuários que utilizam outros tipos de dispositivos de entrada incompatíveis com o mouse não podem ter acesso a esse conteúdo.

Figura 1.16: Página com exemplo de botão para acionar vídeo que só pode ser ativado utilizando o mouse

Para aplicações que são desenvolvidas com linguagens como JavaScript, é muito importante que qualquer evento que dependa de uma entrada do usuário por meio de um dispositivo de entrada tenha tanto eventos baseados em mouse (*onmouseclick*) quanto para teclado (*onkeypress*).

A recomendação de que toda funcionalidade seja acessível por teclado é indicada pelos critérios de sucesso 2.1.1 e 2.1.3 do WCAG 2.0 e pela recomendação 5 do e-MAG 3.0.

Além de fornecer acesso por teclado, é importante garantir uma boa usabilidade da interface para aqueles que vão acessá-la utilizando primordialmente o teclado. É importante que haja uma ordem lógica na tabulação dos elementos. Para usuários que navegam por elementos interativos utilizando a tecla TAB para saltar de um por um, é importante que haja uma ordem lógica e que seja possível alcançar rapidamente os elementos que se deseja utilizar.

1.4.6.2. Identificação da funcionalidade de elementos interativos

É muito importante que desenvolvedores identifiquem claramente qual é a funcionalidade de cada elemento interativo disponível em uma página Web. Em muitas páginas, alguns elementos interativos são identificados unicamente por meio de ícones gráficos ou que dependem de cores, o que pode tornar impossível para um usuário com deficiência visual identificar qual é o botão que deve escolher para utilizar uma funcionalidade.

Em aplicações utilizando *Flash*, por exemplo, pode-se criar um botão que toca um vídeo ou avança um vídeo em uma página Web, por exemplo e utilizar um símbolo de tocar (uma seta para frente). Entretanto, é necessário fornecer uma descrição textual alternativa para um botão que utilize um ícone visual para que, ao focar este elemento, leitores de tela possam ler para o usuário qual é a sua funcionalidade. Na Figura 1.17, por exemplo, é ilustrado um caso em que os botões com ícones em um aplicativo embarcado em Flash tem descrições textuais que não correspondem à funcionalidade dos botões (Unlabelled button 1, unlabelled button 2 e unlabelled button 3). Seria impossível para usuários cegos com leitores de tela para descobrirem qual é o botão correto a ser pressionado para conseguir tocar o podcast.

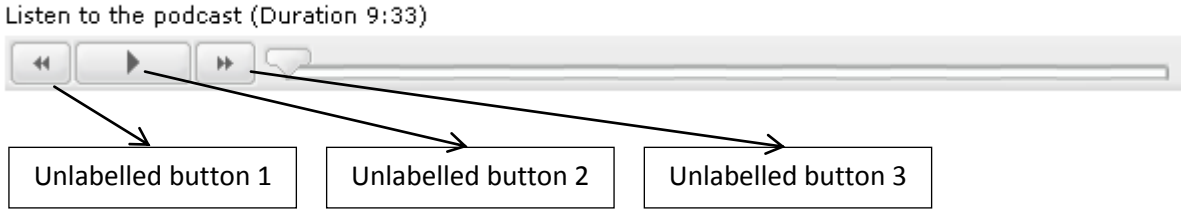


Figura 1.17: Player de um podcast em uma página web educacional – botões sem identificação textual da funcionalidade

1.4.6.3. Identificação da funcionalidade de elementos interativos

Um outro fator importante para usuários com deficiência é saber qual é a função de campos de formulário. Em muitos casos, desenvolvedores só colocarm uma identificação de qual é um campo de forma visual. Por exemplo, a descrição do que

precisa ser colocado em um campo pode estar visualmente na frente ou acima de um campo. Porém, ao chegar em um campo de formulário, um usuário com um leitor de tela não saberá o que é um campo a não ser que haja uma relação explicítica por meio de marcação apropriada relacionando o campo à sua etiqueta. Essa relação é feita por meio do elemento <label> no (X)HTML. O identificador do campo a que ele se refere deve estar explicitamente relacionado por meio do atributo *for* do elemento label da etiqueta. Na Figura 1.18, é ilustrado um exemplo do formulário de contato com o Centro de Educação à distância da Universidade Federal de Lavras.

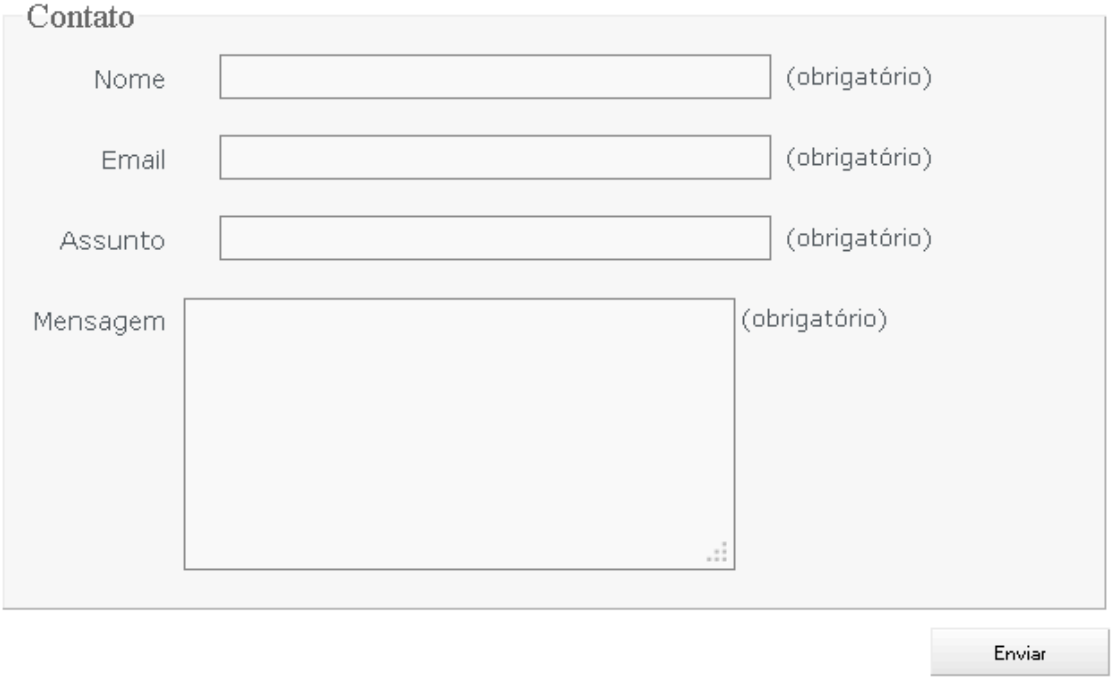


Figura 1.18: Página Web com formulário de contato com o Centro de Educação à Distância da UFLA

No formulário ilustrado na Figura 1.18, a identificação dos campos é feita relacionando os campos com os elementos <label>, conforme descrito a seguir:

```
<form                    enctype="multipart/form-data"
action="/portal/?page_id=138#usermessagea"
method="post" id="cformsform">

<fieldset class="cf-fs1">

<legend>Contato</legend>

<ol>

<li><label for="cf_field_2">Nome</label>

<input type="text" name="cf_field_2" id="cf_field_2"
value=""/>                                    <span
class="reqtxt">(obrigatório)</span></li>

<li><label       for="cf_field_3">Email</label><input
type="text"     name="cf_field_3"     id="cf_field_3"
value=""/>                                    <span
class="emailreqtxt">(obrigatório)</span></li>
```

```
<li><label      for="cf_field_4">Assunto</label><input
type="text"     name="cf_field_4"     id="cf_field_4"
value=""/>                                        <span
class="reqtxt">(obrigatório)</span></li>

<li><label  for="cf_field_5">Mensagem</label><textarea
cols="30"  rows="8"  name="cf_field_5"  id="cf_field_5"
class="area              fldrequired"></textarea><span
class="reqtxt">(obrigatório)</span></li>

</ol>

</fieldset>

<input type="submit" name="sendbutton" id="sendbutton"
class="sendbutton"   value="Enviar"   onclick="return
cforms_validate('', false)"/></p></form>
```

Além da identificação dos campos por meio de marcação que relacione as legendas, é muito importante que os textos que identificam a função dos campos seja claro e que explique de maneira satisfatória a função de cada campo.

Os critérios de sucesso 3.3.2 e 1.3.1 do WCAG 2.0 tratam de questões relacionadas ao uso de marcação de etiquetas para campos de formulários, assim como a recomendação 39 e 42 do e-MAG 3.0.

1.5. Avaliação de Acessibilidade de Sítios Web

A avaliação da acessibilidade de sítios Web pode ser feita utilizando diversos métodos, alguns envolvendo usuários com deficiência reais tentando realizar tarefas e outros métodos que envolvem inspeções por especialistas que revisam os sítios Web de acordo com um conjunto de princípios e recomendações, normalmente com o auxílio de ferramentas automatizadas.

A avaliação de um sítios Web para verificar sua conformidade com as recomendações de diretrizes como o WCAG ou o e-MAG é uma forma de mensurar a acessibilidade de um sítio Web. Inspeções de acessibilidade por meio de uma avaliação de conformidade consiste em checar se as funcionalidades de um sítio Web estão de acordo com as recomendações especificadas em um conjunto de diretrizes. Para desenvolvedores que são familiarizados com acessibilidade, este é provavelmente o método mais comum de avaliação de acessibilidade, principalmente devido à popularidade das diretrizes devido à legislação que as recomendam em diversos países.

Em uma avaliação de conformidade, um especialista em acessibilidade Web analisa todas as funcionalidades de uma página de acordo com os critérios do conjunto de diretrizes. A avaliação pode ser realizada por meio de *testes de conformidade* conduzidos através da combinação de testes feitos por ferramentas automatizadas e inspeções manuais nas quais os especialistas comparam a implementação de uma página com as recomendações das diretrizes. Quando essas avaliações são realizadas, alguns critérios, tais como a presença de um texto alternativo para imagem, podem ser verificados automaticamente por uma ferramenta. Em outros casos, tais como critérios relacionados à clareza de conteúdos textuais alternativos para imagens, a avaliação somente pode ser feita por meio de uma verificação manual.

Ferramentas automáticas podem ser muito úteis para auxiliar os avaliadores a identificar questões que seriam muito tediosas de ser fazer manualmente. Por exemplo, as ferramentas podem auxiliar a verificar a validade de marcação (X)HTML e o uso de folhas de estilo. A verificação também pode incluir o uso de outros recursos, tais como o aninhamento correto de elementos em tabelas e cabeçalhos, e o uso apropriado de tecnologias recomendadas pelo W3C. Este primeiro passo na avaliação auxilia a garantir que uma página Web tem os elementos básicos que permitem que recursos Tecnologia Assistiva possam ler o conteúdo. Ferramentas para realizar testes automatizados de acessibilidade de páginas Web podem ser usadas desde os primeiros estágios de desenvolvimento de protótipos iniciais no desenvolvimento (Petrie e Bevan, 2009).

Além das questões técnicas de marcação básica de código, ferramentas automatizadas também podem ser úteis para detectar outros aspectos, tais como a presença ou ausência de recursos, como texto alternativo para imagens e cabeçalhos. Essas ferramentas também podem ser úteis para verificar se certos aspectos estão definidos de acordo com certos valores pré-definidos, tais como o nível de contraste de cores entre texto e fundo. Os resultados de testes automáticos normalmente são mostrados na forma de um relatório com uma lista detalhada de problemas apresentados para o usuário em uma página Web ou em uma ferramenta externa. A Figura 1.19 mostra um exemplo de um trecho do relatório da avaliação realizada pela ferramenta ASES, do Governo Brasileiro, indicando problemas identificados na página inicial da Universidade Federal de Lavras realizada em agosto de 2013 com a falta de identificação de elementos de tabela.

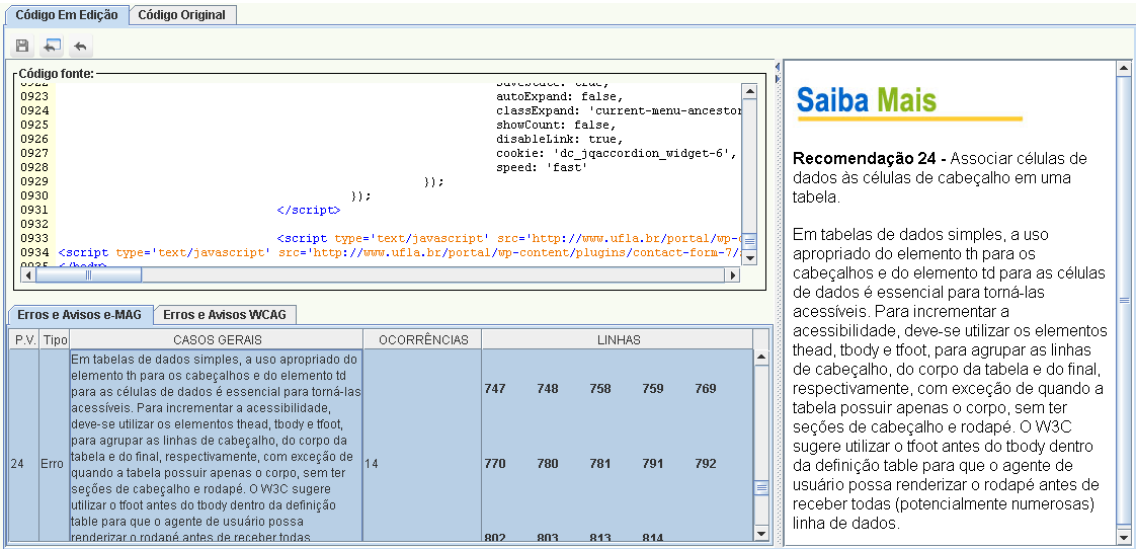


Figura 1.19– Exemplo de relatório produzido pela ferramenta de avaliação automatica de acessibilidade ASES

Juntamente com os testes automáticos, é importante que as inspeções manuais de acessibilidade sejam incorporadas de forma eficaz no desenvolvimento de sítios Web. O uso de métodos de inspeção manual é importante para detectar problemas que podem passar desapercebidos em avaliações automaticamente. Apesar de não encontrarem

todos os problemas que seriam encontrados por usuários, as inspeções manuais podem servir para identificar problemas importantes em estágios iniciais do desenvolvimento.

Inspeções manuais podem ser efetuadas com o auxílio de ferramentas de apoio para auxiliar na realização de testes específicos, tais como a verificação de contraste de cores, simular a visualização de páginas Web em condições específicas (como em diferentes cores, tamanhos de fonte, habilitando ou desabilitando Javascript, por exemplo). Inspeções manuais também devem envolver testes com recursos de Tecnologia Assistiva utilizadas por pessoas com deficiência, tais como leitores de tela, ampliadores de tela, e navegar por uma interface utilizando somente o teclado.

Além de ferramentas de avaliação automática de avaliação de guidelines, existe uma série de outras ferramentas específicas que auxiliam a fazer inspeções de pontos específicos. Essas ferramentas incluem algumas que efetuam diversos tipos diferentes de verificações, como a Web Developer Toolbar[2] do Firefox e a Web Accessibility Tool Bar[3] para Internet Explorer e Opera, desenvolvida em uma parceria entre a Vision Australia[4], The Paciello Group[5] e o Web Accessibility Tools Consortium[6]. Essas ferramentas fornecem funcionalidades como o redimensionamento de páginas Web, mostrar textos alternativos de imagens, desabilitar imagens, salientar informações sobre formulários, dentre outras. Na Figura 1.20, é ilustrado o uso do plugin Web Developer Toolbar do Firefox para auxiliar na verificação dos textos alternativos de imagens presentes em uma página Web. Na Figura 1.21, é ilustrada a ferramenta ColorChecker, um plugin do Firefox que efetua testes de contraste de cores de acordo com as recomendações do WCAG 2.0.

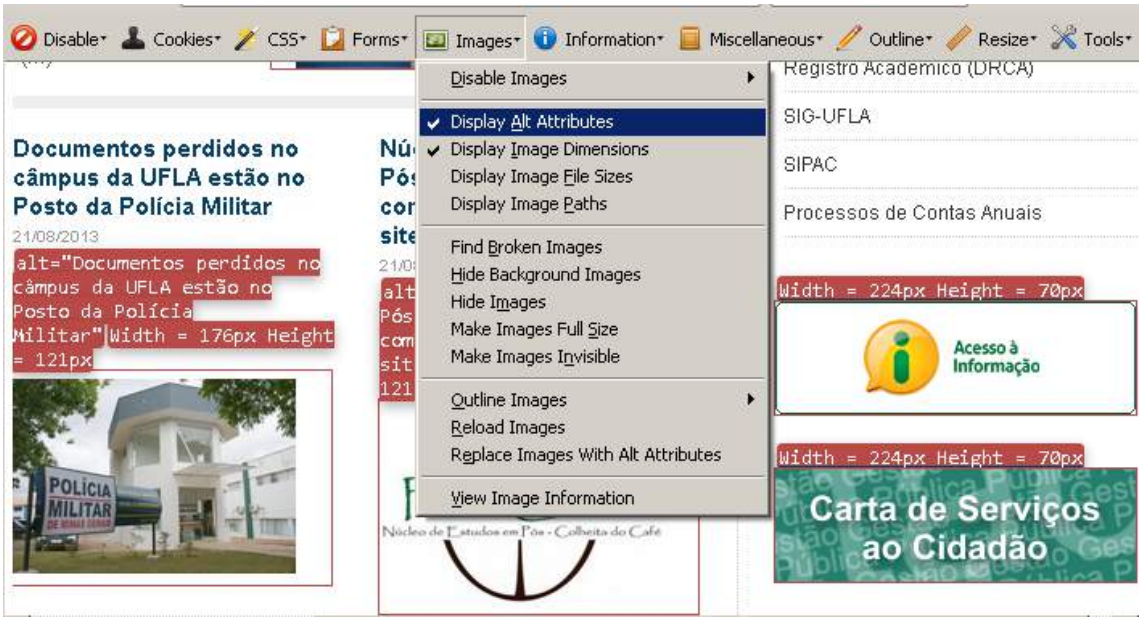


Figura 1.20– Utilização do plugin Firefox Web Developer Toolbar para checar textos alternativos de imagens

[2] Disponível em https://addons.mozilla.org/en-US/firefox/addon/web-developer/
[3] Disponível em http://www.visionaustralia.org.au/ais/toolbar/
[4] Disponível em http://www.visionaustralia.org/
[5] Disponível em http://www.paciellogroup.com/
[6] Disponível em http://www.wat-c.org/

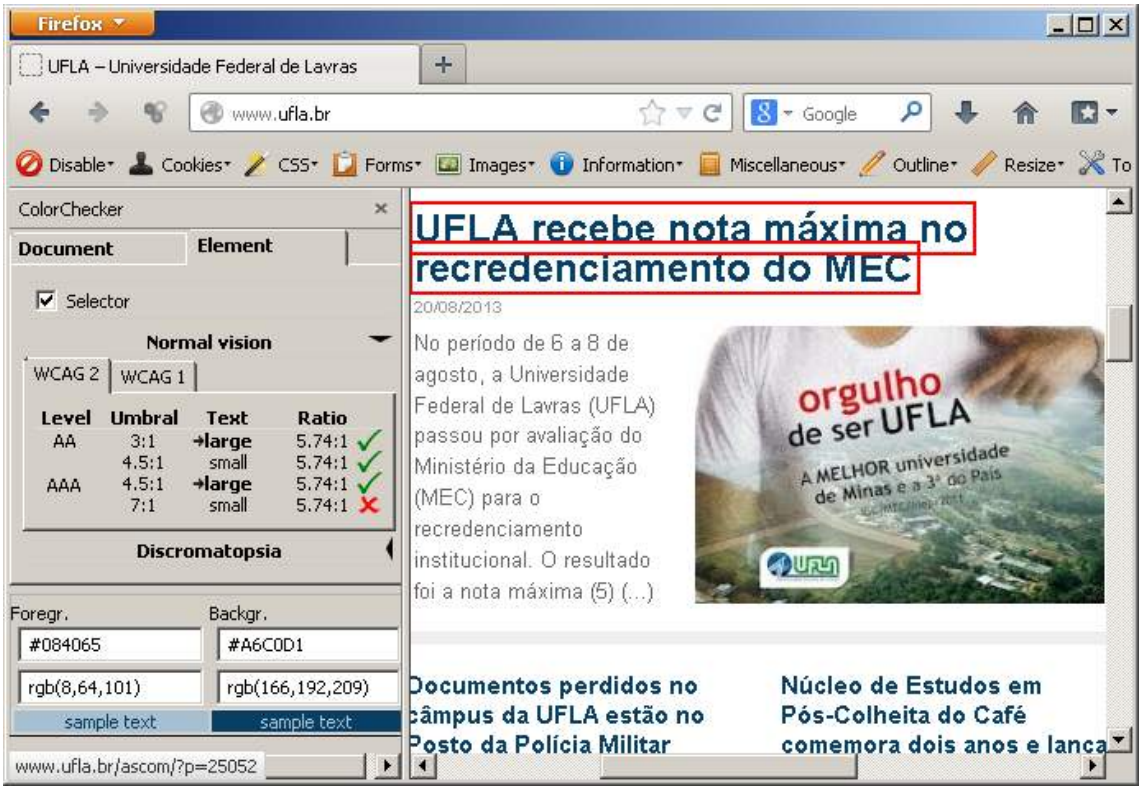


Figura 1.21– Ferramenta ColorChecker e a análise do contraste de um elemento da interface de acordo com as regras do WCAG 2.0

A combinação de métodos de avaliação automática e inspeções manuais pode auxiliar a descobrir diversos problemas importantes de acessibilidade que podem ser corrigidos tão logo sejam descobertos pelas equipes de avaliação. Entretanto, conforme apontado por diversos estudos na literatura (Disability Rights Commission, 2004; Freire, 2012; Power et al., 2012; Rømen e Svanæs 2008, 2012), o uso de recomendações e *guidelines* técnicas só cobre uma parte dos problemas que são encontrados por usuários com deficiência em sítios Web. É muito importante que sejam feitos testes envolvendo usuários com diferentes tipos de deficiência para encontrar e solucionar de maneira mais eficaz os problemas de acessibilidade que podem ocorrer em sítios Web.

1.6. Conclusão

O desenvolvimento de sítios Web acessíveis a pessoas com diferentes tipos de deficiência é de grande importância para garantir que todos possam ter acesso a informações disponibilizadas em sítios Web dos mais diversos contextos.

Neste capítulo, foram apresentados conceitos sobre acessibilidade Web, recursos de Tecnologia Assistiva utilizados por pessoas com deficiência e um conjunto de técnicas e recomendações para auxiliar desenvolvedores a produzir conteúdo Web e mais acessíveis para usuários com deficiência, além de uma visão geral de técnicas para avaliação da acessibilidade de conteúdo Web utilizando recomendações técnicas.

Foram apresentadas técnicas e recomendações para a acessibilidade de conteúdo textual, imagens, áudio e vídeo, bem como técnicas para a implementação correta de elementos estruturais como cabeçalhos, tabelas, listas, e para a acessibilidade de links, elementos de navegação, formulários e elementos interativos. A maioria dos conceitos foi apresentada com exemplos do contexto educacional, com exemplos de bons e maus usos de técnicas para produção de sítios Web e conteúdo multimídia.

A partir dos exemplos mostrados, espera-se que desenvolvedores Web e produtores de conteúdo possam ter um ponto de partida para criar conteúdo que seja mais acessível para pessoas com deficiência e que possam ter fundamentos para aprofundar-se em questões de acessibilidade para outros tipos de conteúdo e funcionalidades comumente utilizadas na Web.

Agradecimentos

Agradecemos ao apoio dado pelo Centro de Educação à Distância (CEAD) da Universidade Federal de Lavras (UFLA) para o desenvolvimento deste trabalho. Também agradecemos ao apoio do NAUFLA – Núcleo de Acessibilidade da UFLA.

Referências

Associação Brasileira de Normas Técnicas (2005) ABNT NBR 15290 – Acessibilidade em Comunicação na televisão.

Alonso, F., Fuertes, J., González, Á., and Martínez, L. (2010) "On the testability of WCAG 2.0 for beginners". In Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A), artigo 9. ACM Press.

Bradford, J. (2005) "Designing Web pages for dyslexic users", Dyslexia Online Magazine, Disponível online em: http://www.dyslexia-parent.com/mag35.html, acesso em agosto de 2013.

Brajnik, G., Yesilada, Y., Harper, S. (2010) "Testability and validity of WCAG 2.0: the expertise effect". In Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility, pages 43-50. ACM Press.

British Dyslexia Association (2013) "Dyslexia Style Guide". Disponível em: http://www.bdadyslexia.org.uk/about-dyslexia/further-information/dyslexia-style-guide.html, acesso em agosto de 2013.

Caldwell, B., Cooper, M., Reid, L. G., Vanderheiden, G. (2008) "Web Content Accessibility Guidelines 2.0", Web Accessibility Initiative (WAI), World Wide Web Consortium (W3C), disponível em: http://www.w3.org/TR/WCAG20, acesso em maio de 2013.

Chisholm, W., Vanderheiden, G., Jacobs, I. (1999) "Web Content Accessibility Guildeines 1.0". Web Accessibility Initiative (WAI), World Wide Web Consortium (W3C). Disponível em: http://www.w3.org/TR/WAI-WEBCONTENT/ acesso em julho de 2013.

Coyne, K. P., Nielsen, J. (2001) "Beyond ALT Text: Making the Web Easy to Use for Users with Disabilities". Relatório do Grupo Nielsen e Norman.

Disability Rights Commission (2004) “The Web: access and inclusion for disabled people: A formal Investigation conducted by the Disability Rights Commission”, London: The Stationery Office.

e-MAG (2011) “Modelo de Acessibilidade de Governo Eletrônico”. Ministério do Planejamento, Orçamento e Gestão, Secretaria de Logística e Tecnologia da Informação; Ministério da Educação, Secretaria de Educação Profissional e Tecnológica. Brasília. Disponível em: <http://www.governoeletronico.gov.br/acoes-e-projetos/e-MAG>. Acesso em agosto de 2013.

Freire, A. P., Russo, C. M., Fortes, R. P. M. (2008) “The perception of accessibility in web development by academy, industry and government: a survey of the brazilian scenario”. New Review of Hypermedia and Multimedia 14 (2), 149-175.

Freire, A. P., Petrie, H., Power, C. (2011) “Empirical Results from an Evaluation of the Accessibility of Websites by Dyslexic Users”. In: 13th IFIP TC13 Conference on Human-Computer Interaction, 2011, Lisbon, Portugal. Proceedings of the Workshop on Accessible Design in the Digital World 2011. Aachen, Germany : Sun SITE Central Europe, 2011. v. 792. p. 41-53.

Freire, A. P. (2012) “Disabled People and the Web: User-based Measurement of Accessibility”. Tese de doutorado, Universidade de York, Inglaterra.

Governo Brasileiro (2013) “Viver Sem Limites: Plano Nacional dos Direitos das Pessoas com Deficiência”. Disponível em http://www.brasil.gov.br/viversemlimite, acesso em agosto de 2013.

Hanson, V. L. (2009) “Age and web access: the next generation”, in Proceedings of the 2009 International Cross-Disciplinary Conference on Web Accessibililty (W4A), Madrid, Spain, 1535658: ACM, 7-15.

Instituto Brasileiro de Geografia e Estatística. Resultados preliminares do Censo 2010. Disponível em http://www.ibge.gov.br/home/estatistica/populacao/censo2010/resultados_preliminares_amostra/default_resultados_preliminares_amostra.shtm, acesso em agosto de 2013.

International Standards Organization (2008) 'ISO 9241-171: “Ergonomics of human-system interaction. Part 171: Guidance on software accessibility”.

International Standards Organization (1998) “ISO 9241-11 - Ergonomic requirements for office work with visual display terminals (VDTs)-Part 11: guidance on usability”.

Kolatch, E. (2000) “Designing for users with cognitive disabilities”, The Universal Usability Guide, University of Maryland, College Park, Disponível online em http://www.otal.umd.edu/UUGuide/erica, acesso em agosto de 2013.

Leuthold, S., Bargas-Avila, J. A., Opwis, K. (2008) “Beyond web content accessibility guidelines: Design of enhanced text user interfaces for blind internet users”, Int. J. Hum.-Comput. Stud., 66, 257-270.

Motta, L. M. V. M., Romeu Filho, P. (2010) “Audiodescrição: Transformando Imagens em Palavras”. Secretaria dos Direitos da Pessoa com Deficiência do Estado de São Paulo.

Petrie, H., Kheir, O. (2007) “The relationship between accessibility and usability of websites”, in *Proceedings of the SIGCHI conference on Human factors in computing systems*, New York, NY, USA, ACM, 397-406.

Petrie, H. and Bevan, N. (2009) “The evaluation of accessibility, usability and user experience” in Stephanidis, C., ed. The Universal Access Handbook, CRC Press, 20-I - 20-XXX.

Petrie, H., Power, C., Swallow, D., Velasco, C., Gallagher, B., Magennis, M., Murphy, E., Collin, S., and Down, K. (2011) “The value chain for web accessibility: challenges and opportunities”. Proceedings of the Workshop on Accessible Design in the Digital World 2011.

Power, C., Freire, A. P., Petrie, H. (2010) “Integrating Accessibility Evaluation into Web Engineering Processes” in Spiliotopoulos, T., Papadopoulou, P., Martakos, D. and Kouroupetroglou, G., eds., Integrating Usability Engineering for Designing the Web Experience: Methodologies and Principles, IGI Global, 55-77.

Power, C., Petrie, H., Freire, A. P., Swallow, D. (2011) “Remote evaluation of WCAG 2.0 techniques by web users with visual disabilities”, in Proceedings of the 6th international conference on Universal access in human-computer interaction: design for all and eInclusion - Volume Part I, Orlando, FL, 2022625: Springer-Verlag, 285-294.

Power, C., Freire, A. P., Petrie, H., Swallow, D. (2012) “Guidelines are only half of the story: accessibility problems encountered by blind users on the web”. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '12). ACM, New York, NY, USA, 433-442.

Rømen, D., Svanæs, D. (2008) “Evaluating web site accessibility: validating the WAI guidelines through usability testing with disabled users”. In Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges (NordiCHI '08). ACM, New York, NY, USA, 535-538.

Rømen, D., Svanæs, D. (2012) “Validating WCAG versions 1.0 and 2.0 through usability testing with disabled users”, Universal Access in the Information Society, 11:375–385.

Theofanos, M. F., Redish, J. (2003) “Bridging the gap: between accessibility and usability”, interactions, 10(6), 36-51.

Theofanos, M. F., Redish, J. (2005) “Helping Low-vision and Other Users with Web Sites That Meet Their Needs: Is One Site for All Feasible?”, Technical Communication, 52(1), 9-20.

Zarach, V. (2002) “Ten Guidelines for Improving Accessibility for People with Dyslexia”, CETIS University of Wales Bangor. Disponível em http://wiki.cetis.ac.uk/Ten_Guidelines_for_Improving_Accessibility_for_People_with_Dyslexia, acesso em agosto de 2013.

Capítulo

2

Explorando HTML5, CSS3 e JQueryMobile no Controle e Monitoramento de Casas Inteligentes

José Antonio Camacho Guerrero[1] e Alessandra Alaniz Macedo[2]

[1]Innolution Sistemas de Informática Ltda. {jose.camacho@innolution.com.br}
[2]FFCLRP/Universidade de São Paulo, Campus Ribeirão Preto {ale.alaniz@usp.br}

Abstract

This chapter is about the use of recent web patterns (HTML5, CSS3, JQuery and JQueryMobile) to develop web applications in different scenarios. As a use case, these technologies are exploited in order to develop an application to domotics context. In the past, automatic environments were limited to big industries. Nowadays this scenario is rapidly changing. Home Control System (HCS) are became frequent and part of modern home. Users of HCS are looking for quality of life and sustainability. We are planning to present and discuss the following items: 1) Development of human computer interface of HCS using web: html5, CSS3, JQuery and JQueryMobile. 2) Concepts and components of HCS. 3) Examples of HCS and Primary Health System (PHS) 4) Use Cases.

Resumo

Neste capítulo, pretende-se apresentar e exemplificar o uso dos padrões mais recentes, HTML5, CSS3, JQuery e JQueryMobile, para o desenvolvimento de aplicações web em diferentes contextos. Como estudo de caso, essas tecnologias são exploradas na construção de um aplicativo para automação e criação de casas inteligentes. No passado, a automação se limitava a indústrias, porém hoje possui crescente e diversificado uso em espaços comunitários como shoppings, restaurantes e até mesmo em residências. Automação busca o aumento da qualidade de vida e a sustentabilidade por meio do uso eficiente de equipamentos, energia e serviços. Planeja-se apresentar e discutir os seguintes itens: (1) Desenvolvimento de interfaces de usuário para controle de hardware via web: HTML5, CSS, JQuery e JQueryMobile, (2) Conceitos e componentes físicos de automação, (3) Exemplos de automação residencial e para a área de saúde, e (4) Estudo de Caso para treinamento.

2.1 Introdução

Automação visa integração de funções de um sistema para que um dispositivo seja controlado. A cada dia, a automação está mais presente nos ambientes buscando aumento da qualidade de vida e sustentabilidade por meio do uso eficiente de equipamentos, energia e serviços. Há alguns anos, a automação se limitava a indústrias, porém hoje possui ampla utilização em residências, shoppings e outros tipos de instituições comunitárias.

Os sistemas de controle domésticos (*Home Control System* – HCS) estão se tornando mais comuns e parte integrante de habitações modernas. O controle computadorizado de alarmes, sistemas de climatização e outras aplicações para habitações são tecnologias que podem favorecer residências em todas as classes sociais. Algumas tecnologias, como os sistemas para controle de iluminação, estão presentes em casas, apartamentos e escritórios de médio e alto padrão, além de grandes empresas, teatros, hotéis e hospitais.

A área da saúde mostra-se necessitada da automação de processos, visto que a maioria dos locais de prestação de serviços e administrativos ainda realiza procedimentos de forma manual, dificultando o controle de dados e o gerenciamento de informações. Pesquisadores da Universidade de Hertfordshire, no Reino Unido, desenvolveram um projeto para cuidar de idosos, por meio da identificação de batimentos cardíacos e outros sinais. Sensores foram instalados para captar os sinais vitais e, em caso de alterações graves, são emitidos avisos para parentes e/ou centrais médicas[1]. No Brasil, agentes de saúde integrantes do programa APS (Atenção Primária a Saúde) utilizam um aplicativo PHCS (*Primary Health Care System*) instalado em um dispositivo móvel, sob a infraestrutura de rede, para realização de acompanhamento e ações preventivas em seus pacientes [Barros et. al, 2012].

Os vários ramos de automação têm em comum os mesmos princípios de controle, utilizando softwares e hardwares, controladores lógicos e linguagens de programação, além de diversos tipos de dispositivos sensores e de atuadores. Sob esse aspecto, o conceito de automação deve estabelecer condições para que todos os subsistemas envolvidos (controles de iluminação, segurança, ar condicionado, controle de energia, incêndio, etc) possam trabalhar em conjunto e de forma otimizada. Para isso é preciso conhecer todas as possíveis potencialidades de utilização nos lares e elencar as tecnologias disponíveis para aplicá-las. O fato de a web estar presente em diferentes hardwares fez com que esse ambiente se torne alvo de profissionais relacionadas com automação.

Durante muito tempo, a ideia de desenvolvimento web ficou exclusivamente associada à construção de páginas que ofereciam aos usuários o acesso a um determinado conteúdo. Desenvolvimento web era normalmente associado ao desenvolvimento de web sites, na internet ou intranet. Associado ao conteúdo, o desenvolvimento web pode ser usado para se referir ao projeto visual das páginas e ao desenvolvimento de comércio eletrônico. Porém, com a popularização da internet, novas necessidades foram surgindo em diversas áreas. As novas necessidades englobam

[1] www.nipponstudios.com.br/automacao-a-favor-da-saude

ferramentas de comunicação, emails, jogos, redes sociais, e-commerce, gerenciamento a distancia de segurança, automação e telemetria de indústrias, prédios e residências.

Liderado por Tim Berners-Lee, o W3C (Consórcio World Wide Web) foi criado com o intuito de definir padrões, protocolos e diretrizes para desenvolvimento de aplicações na web.

Neste capítulo, pretende-se apresentar e exemplificar o uso dos padrões mais recentes para o desenvolvimento de aplicações web: HTML5, CSS3, JQuery e JQueryMobile. Essas tecnologias auxiliam a visualização e o controle de informações estáticas e dinâmicas. Como estudo de caso, planeja-se discutir e aplicar tecnologias na construção de um aplicativo para controle de automação de residências.

2.2 Desenvolvimento de Interfaces de Usuário para Controle de Hardware via Web

Desde que foi concebida por Tim Bernes-Lee na década dos 1990, a web visava à disponibilização de informação na internet. No entanto, rapidamente se torno um meio para efetivar a comunicação, o comércio, o entretenimento, o monitoramento remoto de segurança e diversos outros serviços.

Para atender a essas necessidades, surgiram diversas abordagens utilizando HTML (*Hypertext Markup Language*) e linguagens de programação gerenciadas pelo servidor web, como Python, PHP, ASP entre outras.

Em 2002, o conceito de RIA (*Rich Internet Application*) foi introduzido pela Macromedia para denominar aplicações web com características e funcionalidades de softwares para desktop. A diferença entre HTML e linguagens gerenciadas pelo servidor é que as aplicações RIA transferem o processamento das interfaces para os navegadores no lado do cliente e mantém o conteúdo no servidor, proporcionando maior rapidez e possibilitando o uso de interfaces mais complexas. Soluções utilizando Applets em Java, Java EE, Adobe Flex, Microsoft Silverlight, VBScript e JavaScripts são exemplos de aplicações RIA .

Visando a criação de padrões, protocolos e diretrizes para a web foi criado o W3C (Consórcio World Wide Web) [W3C 2013]. Atualmente o W3C desenvolve especificações técnicas e orientações para aplicações web, arquitetura web, web Semântica, web Services, entre outros.

2.2.1 Linguagem HTML5

HTML (*Hypertext Markup Language*) é uma linguagem para estruturação e apresentação de conteúdo para web criada por Tim Bernes-Lee na década de 1990 visando a comunicação de resultados de pesquisas científicas.

Após a versão 4.01 do HTML, a W3C focou esforços em outra linguagem para web, o XHTML. O XHTML (eXtensible Hypertext Markup Language) é uma reformulação da linguagem de marcação HTML, baseada em XML a qual combina as tags de marcação HTML com regras da XML. O objetivo da W3C com este novo padrão era melhorar a acessibilidade de páginas web através de diversos dispositivos.

Enquanto o W3C trabalhava na segunda versão do XHTML, desenvolvedores das empresas Mozilla, Opera e Apple criaram o grupo WHATWG (*Web Hypertext Application Technology Working Group*) para dar continuidade ao padrão HTML com intuito de fornecer maior flexibilidade na produção de sistemas baseados na web [WHATWG 2013].

Em 2006, W3C e o WHATWG uniram esforços adicionando novos recursos e corrigindo eventuais problemas das versões de HTML dando origem à versão HTML5. Novas funcionalidades como semântica e acessibilidade, com novos recursos, antes só possíveis por meio de outras tecnologias (plugins proprietários e APIs), foram especificados em HTML5. A especificação para HTML5 também introduz marcações e interfaces de programação de aplicativos (APIs) para aplicações web complexas, projetadas para tornar mais fácil a inclusão e a manipulação de conteúdo gráfico e multimídia e para enriquecer o conteúdo semântico dos documentos web.

Na nova versão do HTML foi adicionado um padrão universal para a criação de seções comuns e específicas como rodapé, cabeçalho, sidebar, menus e etc. Também foi criado um padrão de nomenclatura de IDs, classes e tags. Elementos e atributos sofreram modificações na sua função e significado e que agora podem ser reutilizados de forma mais eficaz.

2.2.1.1 Por que HTML5?

O HTML5 permite criar aplicativos multiplataforma, capazes de rodar em uma variedade de dispositivos. Aplicativos em HTML5 além das funcionalidades de acesso remoto podem ser executados como aplicativos nativos permitindo a exibição de vídeo, música e animações. Adicionalmente permite armazenar dados locais, para que fiquem disponíveis mesmo quando não há acesso a internet.

Outro ponto importante para tornar o HMTL5 o padrão principal de desenvolvimento é que muitas companhias importantes participam do grupo de trabalho que definiu o HTML5 no W3C. A lista inclui fabricantes como Samsung, LG e Apple; operadoras como AT&T e France Telecom; produtoras de software como Microsoft, Adobe e Zynga; empresas de TI, como IBM e HP; e da internet, como Google e Netflix.

2.2.1.2 Compatibilidade

Atualmente, há uma grande diversidade de dispositivos pessoais desde o tradicional desktop até dispositivos portáteis como tablets e smartphones que permitem o acesso a web. Essa variedade de dispositivos traz como consequência o uso de uma variedade de navegadores para acessar a web e consequentemente problemas de compatibilidade. Uma forma segura para manter o código compatível é nivelar o desenvolvimento de acordo com os motores de renderização de cada navegador. Esses motores são responsáveis pelo processamento do código da página. Na Tabela 2.1 é apresentada uma lista de motores de navegação utilizados na grande maioria dos dispositivos.

O Webkit é o motor que suporta o maior número de especificações dos padrões do HTML5 e do CSS3. O Webkit foi criado pela Apple, de acordo com o motor de renderização de código aberto KHTML, presente em browsers para Linux, como o Konqueror. A Apple modificou o código, fazendo melhorias e aperfeiçoando o Webkit para criar o browser Safari. Contudo, outros motores já estão trabalhando para atender as novas especificações do HTML5 e do CSS3.

Tabela 2.1 Motores de Renderização

Motor	Browser
Webkit	Safari, Google Chrome
Gecko	Mozilla: SeaMonkey, Camino, Firefox, Thunderbird
Trident	Internet Explorer
Presto	Opera

Para solucionar problemas de incompatibilidade existem técnicas para verificar se o navegador suporta ou não os elementos do HTML5. É possível verificar se uma determinada propriedade existe em objetos globais como WINDOW ou NAVIGATOR. Outra maneira de detecção de compatibilidade é criar um elemento e consultar suas propriedades ou executar algum de seus métodos e avaliar seu retorno.

O Modernizr é uma compacta biblioteca Javascript, criada por Faruk Ates e Paul Irish. Essa biblioteca tem como principal finalidade verificar no navegador a presença de propriedades que permitam o suporte de tags HTML5 e CSS3. Para utilizar a biblioteca Modernizr é necessário fazer sua chamada no *head* do documento. O Modernizr roda automaticamente assim que é referenciado.

O código da Figura 2.1 mostra como verificar se o browser tem suporte da funcionalidade de vídeo utilizando a biblioteca Modernizr.

```
if (Modernizr.video) {
   alert(“Aceita”)
} else {
   alert(“Não Aceita”)
}
```

Figura 2.1 Verificação de compatibilidade utilizando Modernizr

2.2.1.3 Sintaxe e Estrutura do HTML5

A estrutura básica do HTML5 segue a mesma especificação que suas versões anteriores. Um documento é composto por elementos que possuem tags, atributos, valores e conteúdo. Cada elemento deve obrigatoriamente conter um tag, pode conter atributos e valores e seu conteúdo pode ser um texto simples ou outro elemento. Os elementos básicos de HTML, cuja presença é altamente recomendada nas páginas, são representados pelos seguintes tags:

- **<html>:** define o início de um documento HTML e indica ao navegador que todo conteúdo posterior deve ser tratado como uma série de códigos HTML. Este tag possui um atributo importante, o LANG, utilizado para que agentes identifiquem a linguagem principal do documento.
- **<head>:** é onde ficam os metadados que contém informações sobre a página e o conteúdo publicado.

- **<body>**: define o conteúdo principal, o corpo do documento. Esta é a parte do documento HTML que é exibida no navegador.

No tag **<head>** do cabeçalho, pode-se encontrar os seguintes elementos:

- **<title>:** define o título da página, que é exibido na barra de título dos navegadores.
- **<style type="text/css">:** define formatação em CSS. Recomenda-se o uso de arquivo separado para definir a formatação em CSS. Para isso, o arquivo na tag <**link**> é referenciado da seguinte forma: <*link rel="stylesheet" type="text/css" href=" login.css" />***.**
- **<script type="text/javascript">:** define programação de certas funções em página com scripts, podendo adicionar funções de JavaScript.
- **<link>:** define ligações da página com outros arquivos como feeds, CSS e scripts.
- **<meta>:** define propriedades da página, como codificação de caracteres, descrição da página, autor, etc. Um exemplo de uso do tag <meta> seria a declaração de tabela de caracteres utilizada, <meta charset=”utf-8” >.

```
<!DOCTYPE html>
 <html  lang =”pt-br”>
    <head >
          <meta charset =”UTF -8">
           <title >Exemplo da estrutura básica de um documento HTML </ title >
   </ head >
   <body >
       <p style="bottom:2%; left:80%" > Este é o elemento parágrafo</p>
   </ body >
</ html >
```

Figura 2.2 Exemplo de Estrutura do HMTL

A primeira observação do exemplo apresentado na Figura 2.2 diz respeito à declaração do Doctype. O Doctype não é um elemento do HTML, mas é uma instrução para que o browser tenha informações sobre qual versão de código a marcação foi escrita. O Doctype deve ser a primeira linha de código do documento antes do tag HTML, uma vez que indica para o navegador e para outros meios a especificação de código a ser utilizada. Em versões anteriores, era necessário referenciar o DTD diretamente no código do Doctype. Com o HTML5, a referência por qual DTD utilizar é responsabilidade do browser.

Na Figura 2.2 é utilizado o elemento principal HTML, representado pelo tag <HTML> a qual é uma série de elementos em uma árvore cujos elementos são filhos de outros. Essa árvore é denominada árvore DOM.

Os atributos de um elemento são definidos dentro da declaração do tag. No exemplo anterior, o tag <p> tem definido o atributo “style” com valor “bottom:2%;left:80%” e de conteúdo um texto simples: “Este é o elemento parágrafo”.

De acordo com a especificação do W3C, cada elemento tem um propósito específico que deve ser respeitado para a correta interpretação do conteúdo das páginas web. Nesta seção são apresentados os elementos, atributos e tipos mais utilizados no desenvolvimento de aplicativos web.

a) Modelos de Conteúdo

Os elementos no HTML podem ou não fazer parte de um grupo de elementos com características similares. Assim, foram criadas as seguintes categorias de elementos:

- **Metadata content:** Estes elementos estão antes do corpo da página, formando uma relação com o documento e seu conteúdo com outros documentos que distribuem informação por outros meios.
- **Flow content:** Estes elementos suportam o fluxo de informação. Fazem parte desta categoria todas as categorias seguintes.
- **Sectioning content:** Estes elementos definem um grupo de cabeçalhos e rodapés. Basicamente são elementos que juntam grupos de textos no documento.
- **Heading content:** Estes elementos se encarregam da apresentação de cabeçalhos, *h1, h2, h3, h4, h5, h6 e hgroup* e podem estar contidos na categoria Sectioning content**.**
- **Phrasing content:** Estes elementos permitem o tratamento de informação. Eles fazem parte da categoria de elementos que marcam o texto do documento, bem como os elementos que marcam um texto dentro do elemento de parágrafo.
- **Embedded content:** Estes elementos fazem referencia a objetos multimídia que importam outra fonte de informação, áudio, vídeo, canvas, imgens, objetos SVG, etc.
- **Interactive content:** Estes elementos permitem a interação com usuário, menus, botões, entradas, seleção, entre outros.

Dentre todas as categorias de modelos de conteúdo, existem dois tipos de elementos: elementos de linha e elementos de bloco. Os elementos de linha marcam, na sua maioria das vezes, texto, por exemplo, **a, strong, em, img, input, abbr, span**. Já os elementos de blocos dividem o conteúdo nas seções do layout como, por exemplo, **header, nav, article, aside, footer e address**.

Na Figura 2.3 é apresentado como se relacionam as categorias de elementos. Esse relacionamento permite identificar quais elementos podem conter outros elementos dependendo das categorias. Os elementos de linha podem conter outros elementos de linha. Entretanto, os elementos de linha nunca podem conter elementos de bloco.

Elementos de bloco sempre podem conter elementos de linha. Elementos de bloco podem conter elementos de bloco, dependendo da categoria que ele se encontra. Por exemplo, um parágrafo, elemento **p**, não pode conter um elemento **div**, mas o contrário é possível.

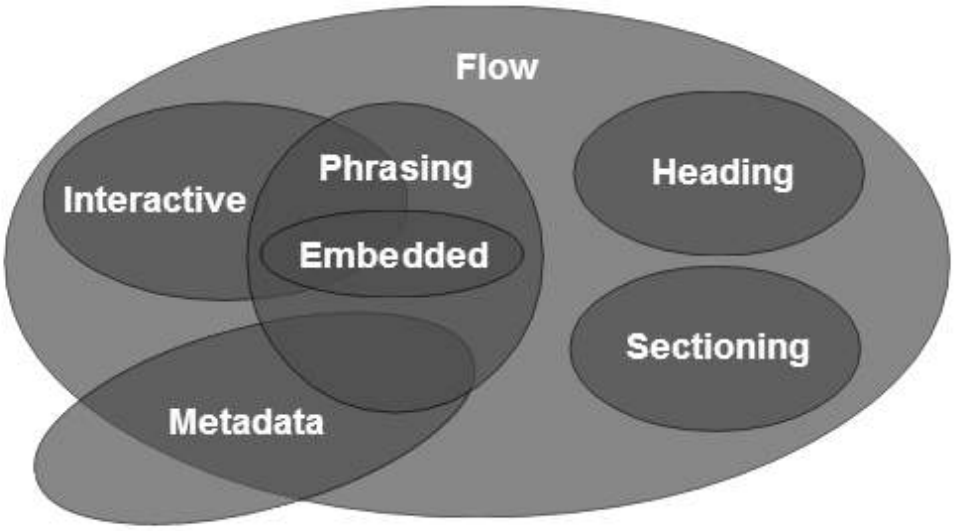


Figura 2.3 Relacionamento entre categorias de acordo com WHATWG

(fonte: http://www.whatwg.org/)

b) Elementos, Atributos e Tipos

Como apresentado na Seção 2.2.1.3, documentos HTML são formados por uma série de elementos aninhados os quais tem atributos com valores e conteúdo. Os elementos permitem que desenvolvedores estruturem a informação que será apresentada no navegador. Versões anteriores do HTML permitiam marcar diversos elementos do layout, estruturando a página de forma que as informações ficassem em suas áreas específicas, contudo careciam de significado semântico.

Na versão 5 do HTML, foram adicionados elementos e atributos que permitem diferenciar semanticamente áreas importantes do documento. No exemplo da Figura 2.4, é apresentado o código em HTML5 que define um formulário para cadastro de contatos. Nesse exemplo, foram utilizados elementos, atributos e tipos novos que auxiliam na segmentação, na apresentação e dão significado ao conteúdo da página.

O elemento **<header>** indica ao navegador que a palavra "Contatos" é um cabeçalho da página. Já o atributo **"placeholder"** utilizado nos elementos **<input>** permite mostrar ao usuário o *que* e *como* se espera que seja digitado o conteúdo desse campo. Também no elemento **<input>** do exemplo, o atributo **"type"** foi utilizado. Esse atributo existe em versões anteriores de HTML, mas na versão 5 são especificados novos valores que permitem comportamentos diferenciados para cada elemento, como por exemplo, *tel, search, email, url, color,* entre outros. Exemplos do uso desses novos tipos são apresentados nas seções seguintes.

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8" />
  <title>Exemplo HTML5- Testando novos elementos, atributos e tipos</title>
  <link href="style.css" rel="stylesheet" type="text/css" media="screen" />
 </head>
<body>
 <div id="contact">
  <header> <h1>Contatos</h1> </header>
   <form action="#" method="post">
            <fieldset>
              <label for="campoNome">Nome:</label>
              <input type="text" id="campoNome" placeholder="Digita teu nome" />
              <label for="campoEmail">Email:</label>
              <input type="email" id="campoEmail" placeholder="email@servidor"/>
              <label for="campoPhone">Phone:</label>
              <intput type=”tel” id=”campoPhone” placeholder=”(ddd)1234.5678”/>
              <input type="submit" value="Send message" />
            </fieldset>
  </form>
 </div>
</body>
</html>
```

Figura 2.4 Exemplo do uso de elementos HTML5

c) Novos Elementos, Atributos e Tipos.

Nesta seção são apresentados os elementos, atributos e tipos especificados na nova versão do HTML. Para cada elemento é descrito brevemente sua funcionalidade. As Figuras 2.5 e 2.6 apresentam exemplos dos elementos.

ELEMENTOS

section- permite dividir documento em segmentos ou seções. Por exemplo, uma loja virtual de música pode classificar seus álbuns por tipo de ritmo, rock, jazz, etc.

article representa uma parte da página que poderá ser distribuída e reutilizável em FEEDs, por exemplo. Isto pode ser um post, um artigo, um bloco de comentários de usuários ou apenas um bloco de texto comum.

aside- é um bloco de anotação para conteúdos de outros elementos próximos.

header- representa um grupo de introdução ou elementos de navegação. O elemento header pode ser utilizado para agrupar índices de conteúdos, campos de busca ou até mesmo logos.

hgroup- agrupa elementos de título (h1 até h6) organizando-os em uma hierarquia de títulos e subtítulos.

```
<section id=”rock”>
  <article>
    <hgroup>
       <h1>Titulo 1</h1>
       <aside>Este titulo faz parte do um album Teste de rock lançado em 2007<aside>
       <h2>Subtitulo 1</h2>
   </hgroup>
   <p>Descricao</p>
 </article>

 <article>
   <hgroup>
      <h1>Titulo 2</h1>
      <h2>Subtitulo 2</h2>
   </hgroup>
   <p>Descricao</p>
 </article>

   <!-- vários elementos article podem vir aqui -->
</section>

<!-- vários elementos section podem vir aqui -->
```

Figura 2.5 Exemplo do uso dos elementos section, article, aside, header e hgroup

footer- é o último elemento a ser apresentado no navegador antes de fechar a tag HTML. O elemento footer ou rodapé pode ser declarado em qualquer lugar do HTML não necessariamente no final de uma seção.

nav- representa uma seção da página que contém links e pode ser utilizado para criar um menu de navegação do site.

time- representa um horário ou uma data precisa no calendário gregoriano.

```
< footer  data-position="fixed">

  < nav class="siam-navbar">
      <a href="#inicio"    class="home-json-bt"  data-iconpos="notext"></a>
     <a href="#ambientes" class="envs-json-bt"  data-iconpos="notext"></a>
     <a href="#perfis"    class="perf-json-bt"  data-iconpos="notext"></a>
     <a href="#cameras"   class="cam-json-bt"   data-iconpos="notext"></a>
     <a href="#panico"    class="panic-json-bt" data-iconpos="notext"></a>
 </nav>

 <p>Publicado em <time> 2013-08-18</time> </p>

</footer>
```

Figura 2.6 Exemplo do uso dos elementos footer, nav e time do HTML5

ATRIBUTOS

Na nova versão HTML5, também foram especificados novos atributos para alguns elementos, auxiliando na interação e na apresentação. Os atributos que foram adicionados são:

autofocus- permite a seleção automática do elemento que especifica este atributo. Os elementos que possuem este atributo são *input, textarea, select* e *button*.

media- é um atributo já utilizado pelo elemento *link*, o qual permite especificar para qual media ou dispositivos o documento que aponta o link está otimizado.

novalidate- permite o envio de dados do elemento **form** sem validação.

reversed- é um atributo do elemento *ol* o qual permite apresentar a lista na ordem inversa.

disabled- já utilizado em outros elementos e agora pode ser utilizado no elemento **fieldset**, permitindo em uma única indicação a desativação de os filhos do **fieldset**.

placeholder- permite apresentar nos elementos de entrada, *inputs* e *textareas*, formatos ou dicas do conteúdo esperado para esse campo.

hreflang e rel- utilizado no elemento *link* em outras versões estes atributos podem ser utilizados no elemento **area.** Esses novos atributos permitem especificar o idioma e a relação do documento apontado pela url.

TIPOS

O elemento **input** do HTML, utilizado para solicitar informação que será inserida por usuários, foi estendido permitindo a formatação e a apresentação de seu conteúdo de forma mais representativa. Para permitir comportamentos diferenciados foram adicionados no elemento **input** os seguintes valores para o atributo **type:** *tel, search, email, url, color, datetime, date, month, week, time* e *datetime-local*, *number* e *range.*

Os tipos de *data e hora, number* e *range* permitem o uso do atributo **step** utilizado para modificar o conteúdo do elemento na quantidade definida. Exemplos de uso de tipos e do atributo **step** são apresentados respectivamente na Figura 2.7 e na Figura 2.8.

```
<!DOCTYPE html>
<html lang=”pt-BR”>
   <head>
        <meta charset=”UTF-8” />
        <title>Number type</title>
  </head>

  <body>
       <input name=”valuex” type=”number” value=”12.4” step=”0.2” min=”0” max=”20” />
       <input name=”valuex” type=”range” value=”12” step=”1” min=”0” max=”20” />
       <input type="date" name="data">
  </body>
</html>
```

Figura 2.7 Uso dos novos tipos do elemento input

Figura 2.8 Visualização dos elementos no navegador

2.2.1.4 Drag-n-Drop

A funcionalidade de arrastar e de soltar especificada no HTML5 é das mais importantes em termos de interação. O HTML5 permite por meio de atributos, tornar um objeto plausível de ser arrastado para outro elemento que permite objetos serem jogados dentro de sua abrangência. Ao adicionar o atributo **draggable=”true”** em um elemento, este pode ser arrastado para outro elemento.

A API Drag-n-Drop implementa eventos que podem ser monitorados e tratados, **dragstart, drag, dragend, dragenter, dragleave, dragover e drop.** Esses eventos permitem desenvolvedores implementar métodos que atuem em determinados estados de interação com as páginas web. Por exemplo, o evento **dragstart** será disparado imediatamente se um elemento **draggable** for arrastado.

Na Figura 2.9 são declarados dois elementos, o *img* e *div*. O primeiro dos elementos, na declaração do elemento *img* é adicionado o atributo **draggable=”true”,** permitindo que a imagem seja arrastada, e o atributo **ondragstart=”drag(event)”**, que avisa o navegador que quando seja detectado o evento **dragstart** execute a função drag.

O segundo elemento vai permitir que o objeto seja adicionado como conteúdo. Para isso foram criadas funções em JavaScript para serem executados quando seja detectado um evento **drop** nesse segundo elemento.

```
<html lang="en">
    <head>
        <title>Drag and Drop - Exemplo 1</title>
        <style type="text/css">
            #DivDestino { width:350px; height:70px; padding:10px; border:1px solid #aaaaaa; }
        </style>
        <script type="text/javascript">
            function allowDrop(ev){
                ev.preventDefault();
            }

            function drag(ev){
                ev.dataTransfer.setData("Text",ev.target.id);
            }

            function drop(ev){
                var data = ev.dataTransfer.getData("Text");
                ev.target.appendChild(document.getElementById(data));
                ev.preventDefault();
            }
        </script>
    </head>
    <body>
        <img id="Drag" src="logo.png" draggable="true" ondragstart="drag(event)" />
        <div id="DivDestino" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
    </body>
</html>
```

Figura 2.9 Exemplo do uso de Drag-n-Drop

2.2.1.5 Elementos de Áudio, Vídeo e Canvas

O elemento de ***audio*** representa um stream de áudio a ser exibido. Atributos como, **src, type, preload, autoplay, loop e controls** podem ser utilizados neste elemento. No atributo **type** do elemento é definido o formato de áudio a ser utilizado e pode ter os valores *Ogg, MP3* e *Wav*. O atributo **preload** indica ao navegador como será carregado o arquivo de áudio ou vídeo quando a página carrega, completo, unicamente metadados ou só quando for acessado pelo usuário.

Para evitar erro de compatibilidade no navegador, pode-se codificar quatro opções, como ilustrado na Figura 2.10. Pode-se observar que a última alternativa será apresentada ao usuário, caso o navegador não suporte nenhum dos formatos de áudio.

```
<audio controls="controls"><br />
      <source src="http://server/audio.ogg" type="audio/ogg" /><br />
      <source src="http://server/audio.mp3" type="audio/mpeg" /><br />
      <source src="http://server/audio.wav" type="audio/wav" /><br />
      Você pode  baixar a musica <a href="http://server/audio.mp3">aqui</a><br />
</audio>
```

Figura 2.10 Exemplo de utilização de um player de áudio

O elemento ***video*** é um dos recursos de HTML5 que tem sido mais utilizado e bem aceito pelos usuários e desenvolvedores. A principal vantagem é a integração natural com os outros elementos e camadas como, por exemplo, CSS e JavaScript. Os formatos suportados são **webm**, **mp4** e **ogv**. Este elemento utiliza os mesmos atributos que o elemento **audio** e pode-se da mesma forma declarar todos os formatos para evitar erros de compatibilidade, ver exemplo na Figura 2.11.

```
<video>
   <source src="movie.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'/>
   <source src="movie.webm" type='video/webm; codecs="vp8, vorbis"'/>
  <source src="movie.ogb" type='video/ogg; codecs="theora, vorbis"' />
  Você pode baixar o video <a href="http://server/video.mp4"> aqui</a><br />
</video>
```

Figura 2.11 Exemplo de tag <video>

2.2.1.6 HTML5 Avançado

Em HTML5 é possível fazer com que objetos interajam com outros objetos por meio de atributos, eventos ou interações com usuários. Nesta seção são ilustradas algumas possibilidades utilizando o elemento ***video*** apresentado anteriormente. Foi escolhido este elemento para demonstrar o impacto que as novas funcionalidades trazem ao desenvolvimento de páginas web. Em versões anteriores do HTML o uso de vídeos era muito complexo. Eles tinham que ser utilizados em applets externos e os codificadores tinham que escrever enormes funções e, JavaScript para interagir com os vídeos. Atualmente é possível manipular desde a apresentação até o conteúdo de um vídeo com comandos simples.

a) Utilizando o Elemento Vídeo em Conjunto com outro Elemento HTML

Novos atributos no elemento de vídeo podem ser usados, os quais também são comuns ao áudio, como, por exemplo, *loop* e *autoplay*. O atributo *pôster* indica qual imagem será mostrada quando o vídeo estiver começando a ser carregado. E o atributo *preload* permite fazer download do vídeo em segundo plano. O uso destes atributos é ilustrado na Figura 2.12.

```
<video poster="star.png" autoplay loop controls tabindex="0">
  <source src="movie.webm" type='video/webm; codecs="vp8, vorbis"' />
  <source src="movie.ogv" type='video/ogg; codecs="theora, vorbis"' />
</video>
```

Figura 2.12 Definição do elemento vídeo avançado

b) Utilizando o Elemento Vídeo e o Manipulado com JavaScript

O elemento de vídeo possui um conjunto de atributos, métodos e eventos que permite controlar o vídeo por meio de código JavaScript. Na Figura 2.13, ***listener*** é adicionado no JavaScript ao evento ***canplay*** para começar a manipular o vídeo.

```
video.addEventListener('canplay', function(e) {
  this.volume = 0.4;
  this.currentTime = 10;
  this.play();
}, false);
```

Figura 2.13 Uso de eventos do elemento vídeo em JS

c) Utilizando o Elemento Vídeo em Conjunto com o Elemento Canvas

Canvas é outro elemento HTML5 com muitas funcionalidades. Com o uso de canvas é possível explorar métodos para edições e apresentações avançadas em vídeo. No exemplo da Figura 2.14, dois recursos do elemento <canvas> para importar e exportar imagens são utilizados.

O método *drawImage* importa imagens a cada execução. Posteriormente, o quadro atual do vídeo é importado e processado no canvas. O segundo recurso utilizado é o método *toDataURL*, que permite exportar o conteúdo do canvas para uma imagem. No exemplo é gerada uma imagem do vídeo a cada 1,5 segundo.

```
video_dom.addEventListener('play', function() {
    video_dom.width = canvas_draw.width = video_dom.offsetWidth;
    video_dom.height = canvas_draw.height = video_dom.offsetHeight;
    var ctx_draw = canvas_draw.getContext('2d');
    draw_interval = setInterval(function() {

// import the image from the video
    ctx_draw.drawImage(video_dom, 0, 0, video_dom.width, video_dom.height);

 // export the image from the canvas
    var img = new Image();
    img.src = canvas_draw.toDataURL('image/png');
    img.width = 40;
    frames.appendChild(img);
  }, 1500)
}, false);
```

Figura 2.14 Explorando o elemento canvas para interagir com vídeo

2.2.2 Estilos CSS3

Paralelamente com o lançamento do HTML4 em 1997, foram criadas as folhas de estilo CSS (*Cascading Style Sheets*). CSS são utilizadas para especificar aspectos de apresentação separadamente da estrutura de páginas web. A separação de estrutura dos aspectos da apresentação simplifica a manutenção e a extensão das páginas web [Deitel & Deitel 2008].

Basicamente existem duas maneiras de descrever os estilos CSS: no arquivo HTML ou em um arquivo CSS. A descrição de estilos em arquivos separados permite que desenvolvedores utilizem esses estilos em diferentes páginas criando uniformidade na apresentação de conteúdos de um mesmo site web.

2.2.2.1 Sintaxe e Propriedades do CSS3

A declaração de estilos em CSS é formada por seletores e as propriedades, conforme Figura 2.15. O *seletor* representa uma estrutura, usada como uma condição para determinar quais elementos de um grupo de elementos serão formatados. A declaração das propriedades dos seletores deve respeitar a sintaxe CSS e devem ser separadas por ponto e vírgula.

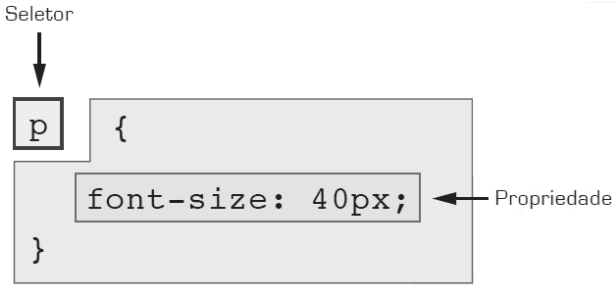


Figura 2.15 Sintaxe CSS

Existem três tipos de seletores, os encadeados, os agrupados e os complexos. Os encadeados são aqueles seletores que representam um elemento que está contido em outros elementos na sequência. Por exemplo, para formatar um link (elemento **a**) que se encontra dentro de um elemento **strong**, que sua vez está dentro do elemento **p** e este encontra-se em um elemento **div,** utiliza-se o seguinte seletor encadeado:

```
div p strong a {
      color: red;

}
```

Os seletores agrupados permitem declarar mais de um elemento com as mesmas propriedades. Os elementos, que são parte do agrupamento, são separados por vírgula. Por exemplo:

```
strong, em, span {
      color: red;

}
```

Já os seletores complexos permitem acessar elementos de difícil acesso pelos tipos anteriores, por exemplo, acessar descendentes, irmãos, filhos, que contenham atributos específicos ou pertençam a uma classe definida. Na Tabela 2.2 é apresentada uma tabela com os seletores complexos, seu significado e a versão do CSS onde foi especificado.

Tabela 2.2 Seletores complexos CSS

PADRÃO	SIGNIFICADO	CSS
elemento[atr]	Elemento com um atributo específico.	2
elemento[atr=”x”]	Elemento que tenha um atributo com um valor específico igual a “x”.	2
elemento[atr~=”x”]	Elemento com um atributo cujo valor é uma lista separada por espaços, sendo que um dos valores é “x”.	2
elemento[atr^=”x”]	Elemento com um atributo cujo valor começa exatamente com string “x”.	3
elemento[atr$=”x”]	Elemento com um atributo cujo valor termina exatamente com string “x”.	3
elemento[atr*=”x”]	Elemento com um atributo cujo valor contenha a string “x”.	3
elemento[atr|=”en”]	Um elemento que tem o atributo específico com o valor que é separado por hífen começando com EN (da esquerda para direita).	2
elemento:root	Elemento root do documento. Normalmente é o HTML.	3
elemento:nth-child(n)	Seleciona um objeto N de um determinado elemento.	3
elemento:nth-last-child(n)	Seleciona um objeto N começando pelo último objeto do elemento.	3
elemento:empty	Seleciona um elemento vazio, sem filhos, incluindo elementos de texto.	3
elemento:enabled elemento:disabled	Seleciona um elemento de interface que esteja habilitado ou desabilidade, como selects, checkbox, radio button etc.	3
elemento:checked	Seleciona elementos que estão “checados”, como radio buttons e checkboxes.	3
E > F	Seleciona os elementos E que são filhos diretos de F.	2
E + F	Seleciona um elemento F que precede imediatamente o elemento E.	2

2.2.2.2 O Novo CSS3

a) Gradiente

O uso de imagens para definir layout e background de páginas web gera requisições de download do servidor. Esse comportamento faz com que navegadores demorem mais para carregar as páginas quando o tráfego da rede fica congestionado. Com CSS3, é possível gerar gradientes unicamente com comandos próprios de CSS. Sendo que o CSS é processado pelo motor de renderização no cliente-side, diminui o número de requisições ao servidor.

Um gradiente é utilizado na propriedade background de qualquer elemento. Quando o CSS é processado pelo motor de renderização do navegador, é necessário fazer chamadas diferenciadas para cada motor, como ilustrado na Figura 2.16.

```
/* Para Mozilla/Gecko (Firefox etc) */

background: -moz-linear-gradient(top, #666, #fff) repeat-X;

/* Para WebKit (Safari, Google Chrome etc) */

background: -webkit-gradient(linear, left top, left bottom, from(#666), to(#fff)) repeat-X;

/* Para IE 8 */

-ms-filter:
"progid:DXImageTransform.Microsoft.gradient(startColorstr=#666,endColorstr=#FFFFFF)";

 /* Para IE 5.5 - 7 */

filter:progid:DXImageTransform.Microsoft.gradient(startColorstr=#666,endColorstr=#FFFFFFF);

/* Exemplo de gradiente complexo utilizando core e linhas intercaladas*/

background-color:#312d28;

background-image: linear-gradient(to right, rgba(255,255,255,1) 1px, transparent 1px), linear-
gradient(to right, rgba(255,255,255,.3) 1px, transparent 1px);

background-size: 50px 50px, 10px 30px;

background-repeat: repeat-x;
```

Figura 2.16 Declaração de gradiente suportados nos diferentes navegadores

Para definir o gradiente no exemplo da Figura 2.16, foi definido o tipo (linear), a direção (top, bottom), o início (#666) e o fim (#FFFFFF) das cores utilizadas na transição. Existem outros tipos de gradiente, como gradiente radial para fazer a transição desde um ponto inicial, aumentando o rádio do círculo em cada transição de cor.

Na Figura 2.17 são apresentados diferentes fundos utilizando os gradientes declarados na Figura 2.16.

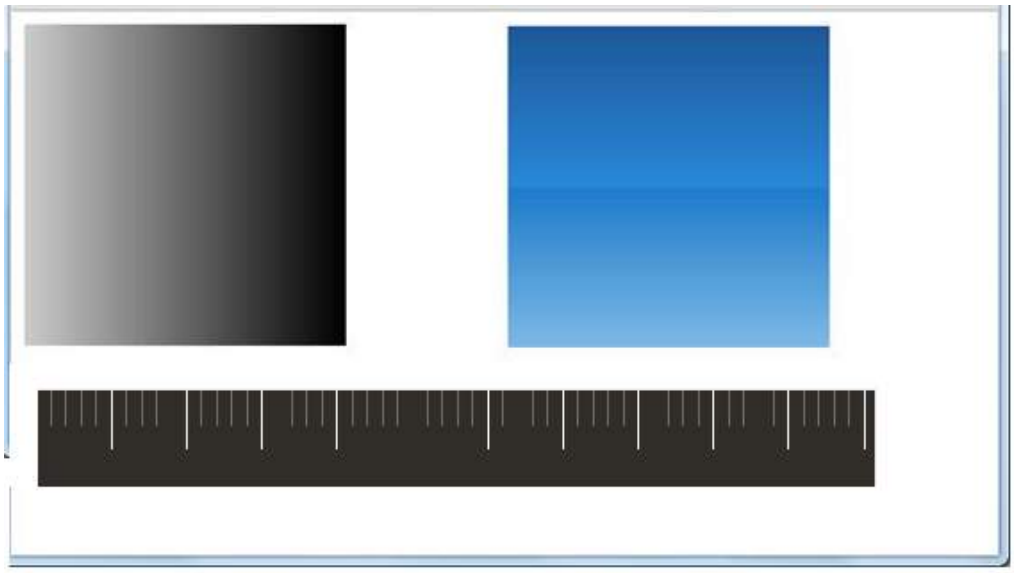


Figura 2.17 Exemplo de uso de gradientes

b) Bordas

A propriedade CSS *border-radius* é responsável por aplicar bordas arredondadas em um elemento HTML. A sintaxe da propriedade *border-radius* permite a especificação do grau de curvatura dos cantos das bordas. A curvatura pode ser igual nos quatro cantos da borda, especificando um único valor, ou cada borda pode ter seu grau de curvatura, como ilustrado na Figura 2.18.

```
# main {
  border - radius : 10 px; // Quatro cantos iguais
 }
# menu {
      border - radius : 10 px 20 px; // Os dois cantos superiores com 10px e os inf com 20px.
 }
# side {
  border - radius : 40 px 30 px 20 px 10 px;//Cada canto com curvaturas distintas.
}
```

Figura 2.18 Exemplo do uso de border-radius

c) Sombras

No CSS3, existem dois tipos de sombras: a sombra que é aplicada a "caixa" do elemento e a sombra que é aplicada no texto de um elemento.

O parâmetro *box-shadow* é responsável por aplicar uma sombra na parte externa ou interna da "caixa" de um elemento. A "caixa" é limitada pelo retângulo definido pela borda do elemento de acordo com o *box model*. Referências de uso do box model podem ser encontradas na especificação CSS. O parâmetro *box-shadow* recebe os valores que contêm informação de deslocamento, desfocamento, propagação e cor. A Figura 2.19 exemplifica como todos os elementos de classe "cam-json-bt" terão uma imagem de tamanho mínimo de 75 px e com sombreamento externo deslocado na horizontal em 1px e sem deslocamento na vertical, desfocado 1px na cor preta.

```
.cam-json-bt{
        background-image: url("images/camerasButtonNormal@2x.png");
        min-height:75px;
        min-width:100%;
        background-repeat: no-repeat;
        background-size:96% 98%;
        background-position:top center;b
        box-shadow:1px 0px 1px #FFFFFF;
}
```

Figura 2.19 Declaração de propriedades para a classe "cam-json-bt"

Para gerar sombras na parte interna da "caixa", deve-se adicionar o valor *inset* na propriedade *box-shadow*. Exemplo: box-shadow: *inset* 1px 0px 1px #FFFFFF.

d) Transformações

A propriedade *transform* manipula o translado, escalonamento, distorção, perspectiva e rotação do elemento do HTML ao ser apresentado. No CSS3, os seguintes valores para a propriedade *transform* foram introduzidos:

- ***scale*** modifica a dimensão do elemento.
- ***skew*** modifica os ângulos dos elementos.
- ***translation*** move o elemento no eixo X e Y.
- ***rotate*** rotaciona o elemento levando em consideração seu ângulo, especialmente quando o ângulo é personalizado com o transform-origin.

Na Figura 2.20 são aplicadas transformações de escalonamento e de rotação para elementos *img.* Essa transformação está sendo realizada para todos os motores de renderização.

```
img {
-webkit-transform: scale(1.5) skew(30deg); /* para webkit */
-moz-transform: scale(1.5) skew(30deg); /* para gecko */
-o-transform: scale(1.5) skew(30deg); /* para opera */
transform: scale(1.5) skew(30deg); /* para browsers sem prefixo */
}
```

Figura 2.20 Exemplo de uso de transformações

2.2.3 Interatividade para páginas Web

2.2.3.1 Transições e animações

As propriedades *transition* e *animation,* e a regra *keyframe* entre outras apresentadas nas seções anteriores, mudaram completamente o uso do CSS. Atualmente é possível fazer interfaces de interação animadas utilizando unicamente regras CSS.

A propriedade *animation* tem uma série de propriedades que podem ser resumidas em um *shortcode* simples. A Tabela 2.3 apresenta as propriedades e os significados.

Tabela 2.3 Propriedades para animação

Propriedade	Definição
animation-name	Especifica o nome da função de animação
animation-duration	Define a duração da animação. O valor é declarado em segundos.
animation-timing-function	Descreve qual a progressão da animação a cada ciclo de duração. Os valores possíveis são: *ease, linear, ease-in, ease-out, ease-in-out, cubic-bezier(<number>, <number>, <number>, <number>) [, ease, linear, ease-in, ease-out, ease-in-out, cubic-bezier(<number>, <number>, <number>, <number>)]** O valor padrão é *ease*.
animation-interation-count	Define o número de vezes que o ciclo deve acontecer. O padrão é um. Para ser declarado como infinito com o valor *infinite*.
animation-direction	Define se a animação irá acontecer ou não no sentido inverso em ciclos alternados. Ou seja, se a animação está acontecendo no sentido horário, ao acabar a animação, o browser faz a mesma animação no elemento, mas no sentido anti-horário. Os valores são alternate ou normal.
animation-play-state	Define se a animação está acontecendo ou se está pausada. Desenvolvedores web poderão, por exemplo, via script, pausar a animação se ela estiver acontecendo. Os valores são running ou paused.
animation-delay	Define quando a animação irá começar, ou seja, o desenvolvedor web define um tempo para que a animação inicie. O valor 0 significa que a animação começará imediatamente.

2.2.3.2 JavaScript, JQuery e JQueryMobile

Uma grande maioria de regras e de funções de interatividade com páginas Web já é possível deixar sob responsabilidade do CSS. No entanto, páginas web não podem existir sem ter scripts que gerenciem e manipulem informação vinda de elementos do HTML. Nesta seção são apresentadas, de forma sucinta, a linguagem JavaScript e as bibliotecas JQuery e JQueryMobile e as informações básicas para habilitar o uso no desenvolvimento de páginas interativas.

a) JavaScript

JavaScript é uma linguagem de script orientada a objetos do tipo client-side. Era originalmente codificada como parte dos navegadores web para que scripts pudessem ser executados do lado do cliente e interagissem com o usuário.

b) JQuery

JQuery é uma biblioteca JavaScript desenvolvida para simplificar os scripts client-side que interagem com o HTML e solucionar problemas de incompatibilidade entre navegadores [JQuery 2013]. JQuery também permite reutilização de código por meio do uso de plugins de outros desenvolvedores.JQuery manipula AJAX e DOM de forma simples e se integra facilmente aos recursos de CSS3. Na Figura 2.21 é comparada a manipulação de elementos.

```
<head>
  <script type =" text / javascript " src =" https :// ajax . googleapis . com / ajax / libs / jquery-
     /1.7.2/ jquery . min .js">
  </ script >
</head>
// Javascript para atribuir a uma id o valor "5".
document.getElementById( 'Teste' ).value = 5;
// O mesmo código em jQuery.
$( '#Teste' ).val( 5 );
```

Figura 2.21 Manipulação de Elementos, JavaScript vs. JQuery

Para utilizar a biblioteca JavaScript JQuery, é necessário adicionar a referência para o arquivo js na tag <script>, conforme Figura 2.21.

A sintaxe básica para executar uma ação sobre determinados elementos é: *$(seletor).acao()*. O símbolo *$* é um método para criar o objeto JQuery, o *(seletor)* serve para consultar e encontrar os elementos HTML e a *acao()* define a operação JQuery que será executada nos elementos. O exemplo da Figura 2.21 apresenta a atribuição do valor 5 no elemento "*Teste"* do HTML.

O *(seletor)* pode ser uma expressão para encontrar um ou mais elementos do HTML. Por exemplo, para selecionar todos os elementos de um documento se utiliza a expressão *$(*)* ou, para selecionar todos os elementos, utiliza-se *div, $(div)*. Além disso, podem ser utilizadas expressões complexas como por exemplo, *$('input [name ^=" curso "] ')$* para selecionar todos os elementos <*input* > cujo atributo *name* comece com a palavra "curso". Informações detalhadas e tutorias da linguagem podem ser encontradas em http://jquery.com/.

c) JQueryMobile

JQueryMobile é um framework para desenvolvimento web, otimizado para interação por toque e que se destina à criação de aplicações para smartphones e tablets. Seu desenvolvimento está voltado para fornecer mecanismos capazes de criar sistemas unificados de interface de usuário, baseados em HTML5 e CSS3. Os scripts em

JQueryMobile são capazes de rodar em todas as plataformas móveis tendo como base de construção as bibliotecas jQuery e jQuery UI.

Para utilizar a biblioteca JQueryMobile, deve-se obrigatoriamente declarar um link para a biblioteca JQuery, outro para o framework JQueryMobile e um outro para a folha de estilo CSS3, padrão do framework, como apresentado na Figura 2.22 .

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Exemplo JQueryMobile</title>
    <link rel="stylesheet" href="http://code.jquery.com/mobile/latest/jquery.mobile.min.css “ />
    <script src="http://code.jquery.com/jquery.min.js"></script>
    <script src="http://code.jquery.com/mobile/latest/jquery.mobile.min.js “></script>
  </head>
  <body>
  </body>
</html>
```

Figura 2.22 Declarações mínimas para uso de JQueryMobile

A *metatag viewport,* declarada no exemplo da Figura 2.22, é fundamental para o funcionamento das páginas em dispositivos móveis, informando ao navegador como deve ser renderizado o conteúdo do documento HTML. É possível determinar a largura e a altura da *viewport* e atributos de escalabilidade das interfaces.

Informações detalhadas e tutorias da linguagem podem ser encontradas em http://jquerymobile.com/.

2.3 Conceitos e Componentes Físicos para Automação

Os sistemas controle domésticos (*Home Control System* – HCS) ou automação residencial é composto de uma rede de comunicação que permite a interconexão de uma série de dispositivos, equipamentos e outros sistemas. O objetivo principal da automação é melhorar a qualidade de vida, aumentar a segurança e viabilizar o uso racional dos recursos para seus usuários. Para atingir este objetivo, os equipamentos ligados na automação enviam informações sobre o ambiente residencial e o meio em que ele se insere, efetuando determinadas ações a fim de supervisioná-lo ou gerenciá-lo (BOLZANI, 2004).

2.3.1 Automação Residencial

A automação residencial é iniciou-se suportada por a automação industrial, porém, em virtude da diferente realidade entre os dois cenários, atualmente têm sido criados sistemas dedicados para ambientes onde não se dispõe de espaço para grandes centrais controladoras e extensos sistemas de cabeamento. Na automação residencial pode-se

encontrar maior diversidade de dispositivos. Os dispositivos residenciais abrangem desde controle de luzes, equipamentos multimídia até o tráfego de dados de telemetria. No ambiente residencial, diferente do ambiente industrial, essas tecnologias são utilizadas por pessoas que não necessariamente possuem qualquer conhecimento técnico. Isso torna primordial o desenvolvimento de interfaces de usuário simples e intuitivas.

Automação residencial traz a possibilidade de facilitar tarefas diárias como irrigar jardins, estabelecer níveis de temperatura e de iluminação, evitar preocupações com janelas em dias de chuva ou portas abertas ao sair de casa, dentre outros. O auxilio ou a total execução destas tarefas contribui para oferecer níveis de conforto e ao mesmo tempo conduzir a economias de eletricidade, gás e água. A automação residencial também é eficiente quanto à segurança, detectando ou sinalizando situações de emergência ou situações de intrusão, podendo acionar imediatamente as centrais de segurança (Dias, C. L. A, Pizzolato N. D. ,2004).

Os sistemas de automação podem ser desenvolvidos em dois tipos de arquiteturas: centralizadas ou distribuídas. Na arquitetura centralizada, todos os dispositivos são conectados a uma unidade central de controle a qual é responsável pelo controle e gerenciamento das reações aos eventos recebidos e das ações ou comandos a serem executados pelos dispositivos.

Na arquitetura distribuída ou descentralizada, os diversos dispositivos são capazes de processar ações com critérios próprios. Cada ação uma função específica dentro das inúmeras necessidades do sistema de automação, sendo interligados por uma rede, comunicando-se e enviando sinais entre sensores e atuadores. Esses sensores e atuadores podem se encontrar próximos ou integrados ao ponto de controle e monitoração (Dias, C. L. A, Pizzolato N. D. ,2004).

Um sistema de automação residencial deve ser desenvolvido de acordo com os requisitos propostos pelo usuário. Somente o usuário pode decidir suas prioridades e o quanto vai investir. Assim a automação residencial é específica para cada casa a ser automatizada. Na Figura 2..23 são apresentados os dispositivos mais comuns que podem ser monitorados e controlados pela automação.

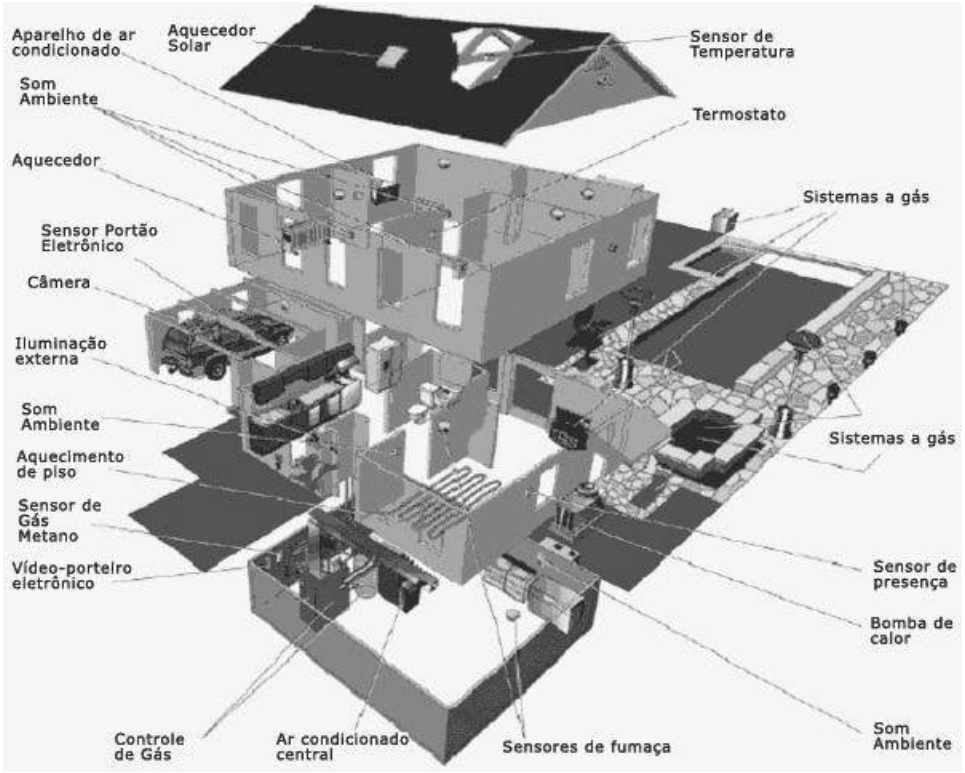


Figura 2..23 - Automação Residencial (Fonte: www.idealhome.com acesso em 20/08/2013).

A representação da automação residencial da Figura 2.23 ilustra uma automação organizada como todo sistema de automação, que tem uma central considerada o cérebro da casa. Os componentes mais comuns utilizados para criar a central de automação residencial são PIC, Arduino, CLP (comando lógico programável). Esses componentes tem inteligência para gerenciar regras de execução. Eles que detêm toda a lógica da automação.

Os *dispositivos de controle e monitoramento* têm um endereçamento ou id e tem funções específicas. Existem também *atuadores liga/desliga*, *atuadores dimmer, atuadores para motores*, *atuadores infra-vermelho*, *interfaces para sensores para botões* e *controladoras de acesso* (RFID, Teclados, Biometria, Controle remoto, etc). Os *sensores* são os que leem informação da casa. Eles podem ser de movimento (para liga/desliga de luzes, câmeras e som ambiente), luminosidade (para abertura de persianas), abertura (para portões e janelas), gás (para controle de gás), temperatura (para, por exemplo, aparelho de ar condicionado, aquecimento de piso, aquecedor), umidade (para ligar/desligar irrigação), nível de água (para bomba de água e irrigação), consumo de energia (para gerar relatórios de consumo por mês, liga/desliga automático de luzes), etc.

Existe uma rede de comunicação entre os sensores, que normalmente utiliza o protocolo RS232. Atualmente existe uma associação, a KNX, que está criando padrões de comunicação no modelo OSI para dar mais segurança na troca de informação. Basicamente, a central envia comandos aos atuadores se uma regra for atingida. Existem regras de horários, cenários, ou para interpretação das informações vindas dos sensores ou controladores de acesso.

Os dispositivos de controle de acesso e segurança permitem identificar os usuários do sistema possibilitando ao sistema de automação gerenciar o comportamento de ações personalizadas e a restringir áreas da residência. Por exemplo, prestadores de serviço podem ter acesso apenas a uma porta específica em um determinado período, um visitante pode ter acesso apenas a um ambiente e os moradores têm acesso a todos os ambientes.

Os sistemas de controle domésticos estão se tornando mais comuns e parte integrante de habitações modernas. Algumas tecnologias, como os sistemas para controle de iluminação, estão presentes em casas, apartamentos e escritórios de médio e alto padrão, além de grandes empresas, teatros, hotéis e hospitais.

2.3.2 Automação para Health Care

As áreas industrial, comercial e residencial se mostram mais familiarizadas com a prática de automação. No entanto, a área da saúde ainda é pouco automatizada[2]. As grandes motivações da automação na área da saúde são o ganho de conforto e a segurança que ela pode proporcionar. A maioria dos procedimentos de Health Care ainda é realizada manualmente, tornando-os lentos e mais propícios a erros.

Seguindo a linha de monitoramento de pacientes, Murakami et al. (MURAKAMI, 2006) desenvolveram o vMon- Gluco que implementa o monitoramento em tempo real dos níveis de glicose dos pacientes. Esse sistema foi desenvolvido sobre dispositivos móveis, sendo utilizado em Unidade de Terapia Intensiva (UTI). Esse trabalho permite que pacientes com altas taxas de glicose possam ser monitorados de forma automatizada e com uma frequência maior e mais precisa, melhorando a qualidade do atendimento do paciente. Outra vantagem do sistema é possibilitar o escalonamento mais eficiente da equipe médica, uma vez que um processo que demandava tempo e recursos humanos foi automatizado.

Um aspecto importante da automação na saúde é a usabilidade do sistema. O sistema deve ser o mais fácil e intuitivo possível. Os profissionais da saúde usam vários dispositivos que se encontram em diversas áreas, ou seja, o ambiente de atuação é volátil. Um sistema de baixa usabilidade exigiria uma nova autenticação a cada ambiente, o que dificulta o uso do sistema, criando muitas vezes rejeição da tecnologia (BARDRAM, 2005). Nesse sentido, a tecnologia RFID presente na automação residencial se mostra eficiente para a autenticação dos usuários na saúde. A identificação por radiofrequência (RFID) é uma tecnologia para identificação automatizada de objetos e pessoas (MOORE, 2011). Dessa forma, não é necessário que o profissional se identifique a cada novo ambiente, o RFID o identifica.

A automação na saúde pode utilizar várias tecnologias provenientes da automação residencial. No entanto, antes de submeter a automação em hospitais e clínicas, as prioridades e áreas mais críticas devem ser devidamente analisadas (Saúde Automatizada, 2005). Na área da saúde, a preocupação com a segurança de dados e a intolerância a falhas são requisitos essenciais.

[2] http://www.bpsolutions.com.br/bpmaga/edicao11/16_17_DICAS.pdf Dicas de automação em saúde.

2.3.3 Estudo de Caso - Automação Residencial

Como estudo de caso é utilizado o sistema de automação residencial SIAM[3] (Sistema Integrado de Automação e Monitoramento). SIAM é também uma indústria brasileira que desenvolve equipamentos e software para automação, telemetria, controle de acesso e vigilância remota.

Os produtos da SIAM são voltados para aumentar a comodidade, o controle e a segurança de seus clientes em relação às suas residências ou locais de trabalho. Por meio de uma rede de dados, é possível automatizar funções e oferecer segurança em diversas aplicações, principalmente, automação e telemetria.

2.3.3.1 Arquitetura do Sistema

Os produtos SIAM são suportados por uma arquitetura distribuída, onde cada sensor e cada atuador têm identificador único e ações associadas para monitorar e controlar dispositivos. Os sensores e atuadores são interligados por uma rede de dados que permite a troca de informação com a central de gerenciamento. A Figura 2.24 ilustra a arquitetura do sistema SIAM que suporta a comunicação de softwares e hardwares.

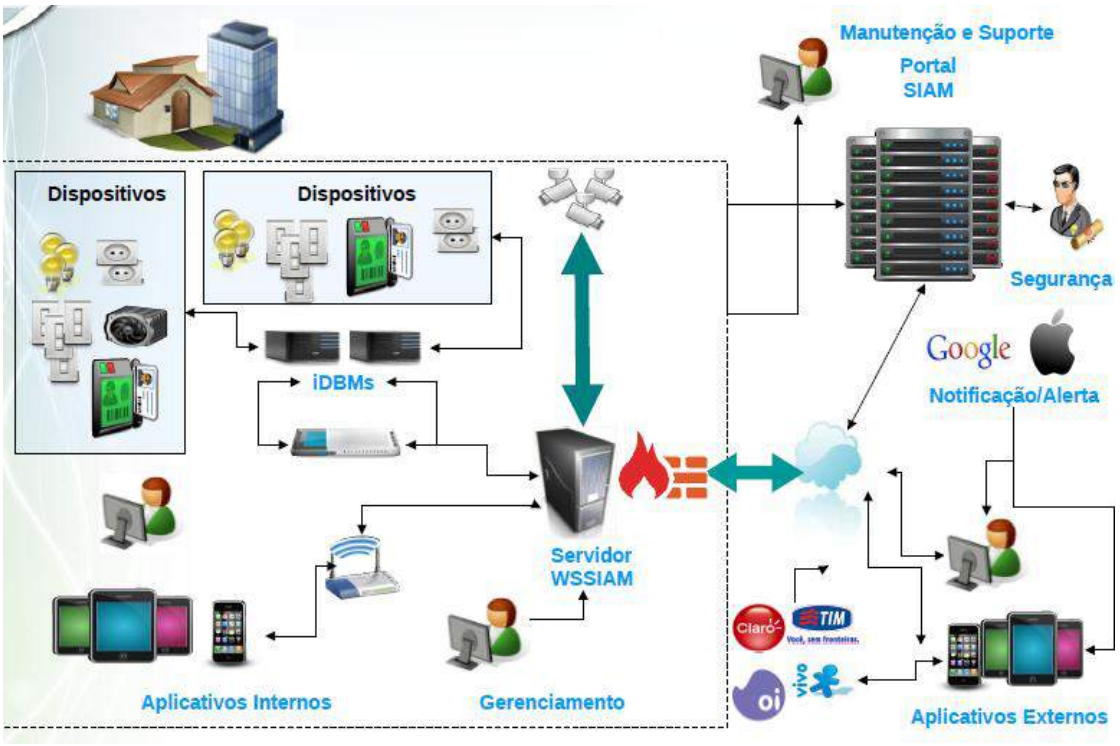


Figura 2.24 - Arquitetura do sistema de automação residencial SIAM

O Sistema Integrado de Automação e Monitoramento (SIAM) é formado pelos seguintes componentes de hardware e de software:

- **iDBM:** é a central de controle que gerencia a informação gerada pelos sensores e dispositivos. De acordo com essa informação, a iDBM executa as regras

[3] http://siam1.hospedagemdesites.ws/Site/Enterprise/Enterprise.aspx

definidas pelo instalador e comunica as ações a serem executadas pelos dispositivos.

- **WSSIAM:** é um servidor desenvolvido em Java com interfaces para web em Adobe Flash. O WSSIAM é responsável por monitorar e gerenciar a comunicação e a configuração de hardwares e de softwares. Este servidor pode ser instalado na residência e/ou em local remoto mantendo sincronização dos dados em todas suas instâncias.

- **SIAM-Mobile:** é interface do sistema com usuário final. O sistema desenvolvido com tecnologia HTML5, CSS3, Ajax e JQueryMobile, permite o controle de ações e de leitura de eventos ocorridos no sistema de automação de forma remota.

- **VRC:** é um servidor de Vigilância Remota que permite o acesso a câmeras de segurança e de controle dos ambientes onde as câmeras estão instaladas.

- **Sensores:** são utilizados para diversos tipos de monitoramento como temperatura, umidade, níveis de água, luminosidade, presença, consumo de energia, detecção de incêndio, entre outros.

- **Atuadores:** são equipamentos utilizados para realizar ações como ligar alarme, ativar irrigação, ligar/desligar luzes, abrir portas, entre outros.

- **Controle Segurança:** são dispositivos para permitir a identificação de usuários por meio de sensores de RFID, teclados para senhas, biometria e controle remoto.

2.3.3.2 Sistema SIAM-Mobile

Nesta seção vamos apresentar as interfaces desenvolvidas com as técnicas apresentadas neste curso, HTML5, CSS3 e jQueryMobile são discutidas.

Antes do desenvolvimento das interfaces web, foi necessário criar uma interface de comunicação entre os serviços que compõem o sistema de automação. As diretrizes definidas nas camadas de comunicação, apresentada na Figura 2.25, foram seguidas. Foi criado um servidor para autenticar as requisições web o qual instancia as *capabilities* que um determinado usuário tem no sistema. Uma vez definidas as *capabilities*, uma linha de comunicação é vinculada entre os serviços e o hardware de automação.

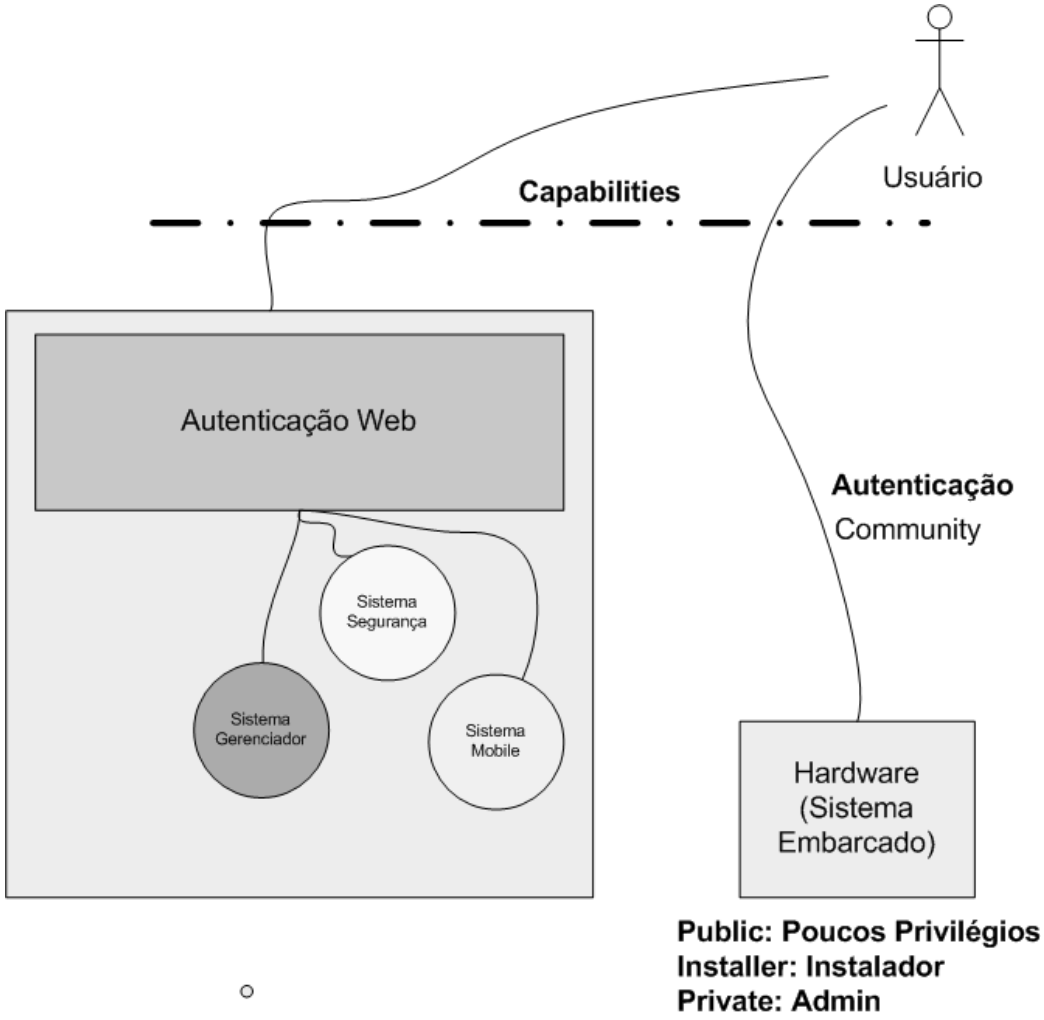


Figura 2.25 - Camadas de Comunicação do SIAM

Na Figura 2.26 é apresentada a preparação da página HTML5 do sistema web de automação. Na página principal do sistema web móbile foi definido inicialmente o atributo *lang="pt-br"* no elemento principal <HTML> para indicar o navegador que a linguagem principal do documento é português do Brasil. Em seguida definimos a metatag *charsert="utf-8"* foi criada para informar a codificação do conteúdo. E o *viewport* é de dimensão fixa do tamanho da tela do dispositivo.

O elemento <*link*> foi utilizado para relacionar imagens que serão utilizadas quando um shortcut à página seja criado. Em seguida, foram relacionados os arquivos para uso das bibliotecas de JQuery, JQueryMobile e Storage. A biblioteca Storage permite o armazenamento e a recuperação de dados no cliente-side. Posteriormente, foram relacionados os arquivos para as folhas de estilo CSS. É importante observar que foi utilizado o atributo *media* no elemento *link* que faz referência ao CSS para determinar qual arquivo será utilizado. Se o dispositivo tiver uma largura da tela maior a 615px será utilizado o arquivo landscape.css, caso contrário o arquivo portrait.css.

Finalmente foram relacionados os arquivos que contem os scripts desenvolvidos para tratamento de informação, eventos e interação com usuário.

```
<!DOCTYPE html>
<html lang=”pt-br”>
  <head>
        <meta charset="utf-8" />
        <title>SIAM Mobile</title>
        <meta name="viewport" content="width=device-width,initial-scale=1,maximum-scale=1"/>
        <!--
        Icones para Shortcut
        -->
        <link rel="apple-touch-icon" sizes="72x72" href="src/css/images/icon@2x.png" />
        <link rel="app-icon" sizes="24x24" href="src/css/images/icon.png" />
        <link rel="shortcut icon" href="src/css/images/icon-ipad.png"/>
        <!--
        Bibliotecas java script utilizadas
        -->
        <script src="lib/jquery-1.7.2.min.js"></script>
        <script src="lib/jquery.mobile-1.2.0.js"></script>
        <script src="lib/jquery.Storage.js"></script>
        <link href="src/css/jquery.mobile.structure-1.1.0.css" rel="stylesheet">
        <link href="src/css/jquery.mobile.theme-1.1.0.css" rel="stylesheet">
        <!--
        Referencias para arquivos customizados
        siam_x.css define a fonte, localização, tamanho e todos os atributos
        -->
        <link rel="stylesheet" href="src/css/siam_landscape.css" type="text/css" media="only all and (min-width:615px)" title="no title" />
        <link rel="stylesheet" href="src/css/siam_portrait.css" type="text/css" media="only all and (max-width: 615px)" title="no title" />
        <!--
        Link para arquivo de metodos em JavaScript para comunicação com a o WSSIAM
        -->
        <script src="src/js/siam_ws.js"></script>
        <script src="src/js/siam.js"></script>
  </head>
  <body>…….....</body>
</html>
```

Figura 2.26 Definições da página inicial do sistema de automação

```
<div id="login" data-role="page" class="type-home" data-theme="a" >
   <header id="header-div" class="ui-siam-header" data-role="header" data-position="fixed" data-tap-toggle="false" data-theme="a">
        <span id="header-title" class="head-label" >Login SIAM </span>
    </header>
        <div id="main-page" data-role="content" data-tap-toggle="false">
            <form id="main-content" class="content-secondary" action="#inicio">
                <label for="usuario">Usu&aacute;rio</label>
                <input type="text" name="usuario" data-role="none">
                <label for="senha">Usu&aacute;rio</label>
                <input type="password" name="senha" data-role="none">
                <input type="submit" value="submit" onClick="login();">
             </form>
        </div>
       <footer data-position="fixed" data-tap-toggle="false">
           <nav data-role="navbar" class="siam-navbar">
            <ul>
              <li><a href="#inicio"    class="home-json-bt"  data-iconpos="notext"></a></li>
              <li><a href="#ambientes" class="envs-json-bt"  data-iconpos="notext"></a></li>
            </ul>
           </nav>
        </footer>
</div>
```

Figura 2.27 Conteúdo da página inicial

Na Figura 2.27 é apresentada a definição dos elementos que compõem a página de login e de início. Nessa página foram utilizadas as tags <header> para definir o cabeçalho e <*footer*> para instanciar um rodapé. Dentro do elemento <*footer*>, foi criado um elemento <*nav*> para referenciar o menu de navegação do sistema. No elemento <*footer*> foi utilizado o atributo *data-position="fixed"* que fixa o elemento para não ser afetado pelo scroll do navegador.

Nas Figura 2.28 e na Figura 2.29 são exibidas as interfaces de usuário correspondentes ao código da Figura 2.27 no navegador desktop e no smartphone, respectivamente.

Figura 2.28 Páginas de Login e Inicio em navegador Desktop

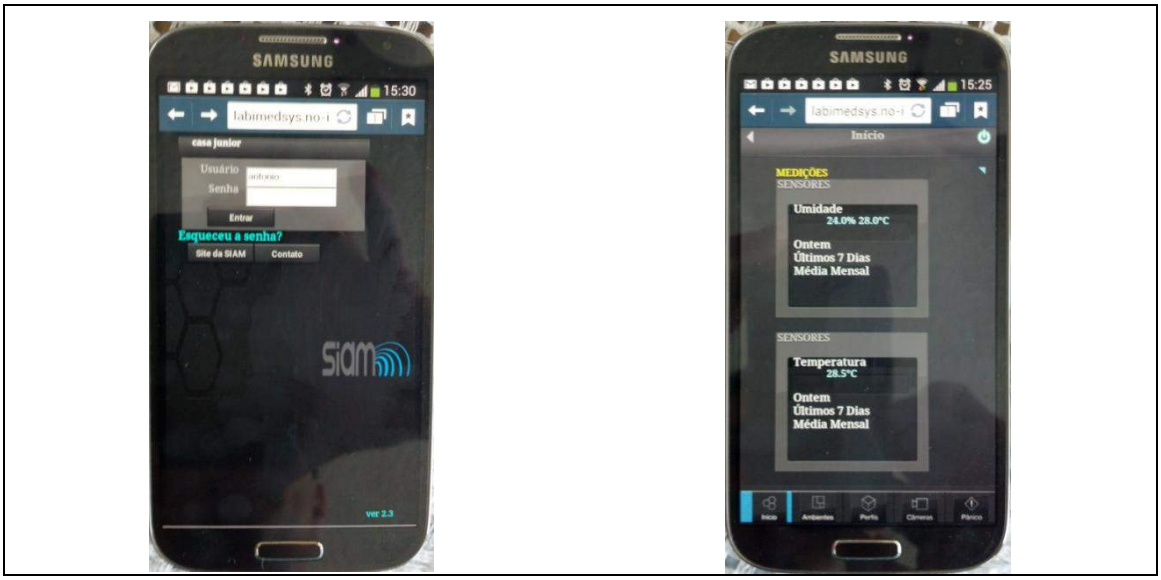


Figura 2.29 - Páginas de Login e Inicio em SmartPhone

Na Figura 2.30 são exibidos elementos de *vídeo* e elementos para controle de iluminação e acesso a portas. Quando pressionado um botão, o evento *touchstart* é acionado pelo navegador. Esse evento está sendo monitorado por um método codificado no arquivo js. Quando o evento é acionado, o script dispara uma chamada ao serviço de comandos o qual validará a permissão e, posteriormente, enviará uma requisição para o hardware de controle.

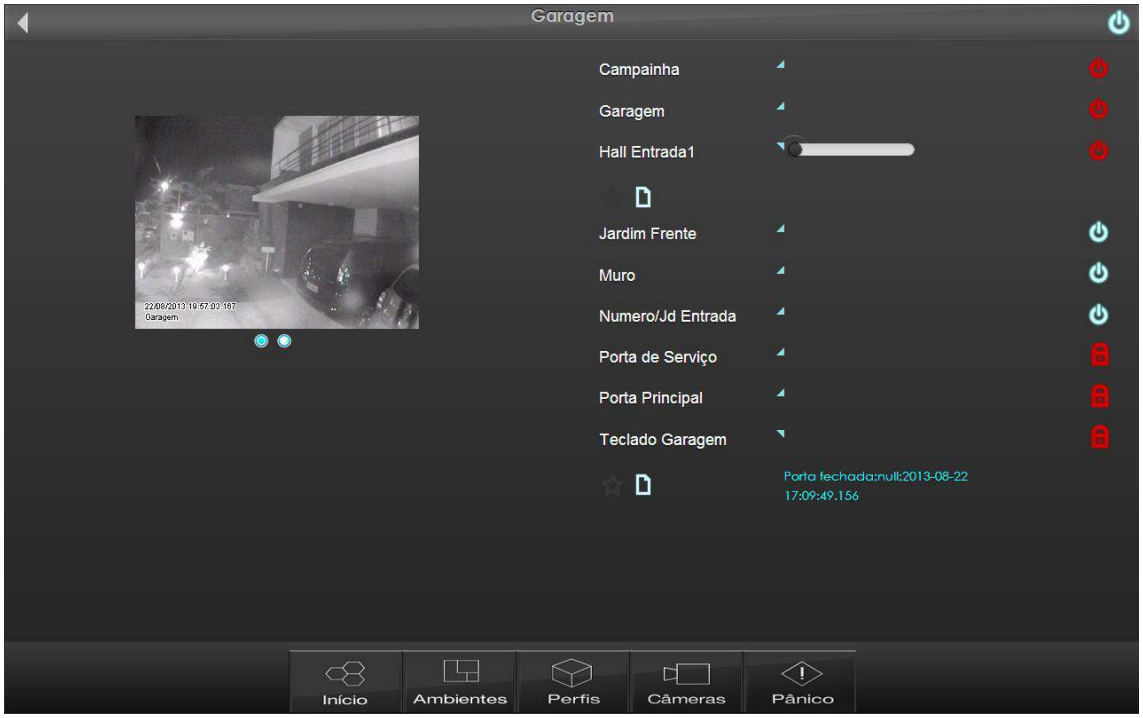


Figura 2.30 Página exibindo Vídeo e Dispositivos de controle

2.4 Biografias Resumidas dos Autores

2.4.1 José Antonio Camacho Guerrero

José Antonio Camacho Guerrero é formado em Engenharia de Sistemas Computacionais pelo Instituto Tecnológico y de Estudios Superiores de Monterrey - México (1997) e possui título de mestre em Ciência da Computação e Matemática Computacional pela Universidade de São Paulo (2002). Atualmente é fundador e coordenador de projetos das empresas Innolution - Sistemas de Informática Ltda e da i-Medsys. Tem experiência na área de Ciência da Computação, com ênfase em Gerenciamento de Informação, atuando principalmente nos seguintes temas: recuperação de informação, hipermídia, web semântica, automação residencial, software de apoio e acompanhamento médico e equipamentos móveis.

2.4.2 Alessandra Alaniz Macedo

Alessandra Alaniz Macedo possui graduação em Ciência da Computação pela Universidade Estadual de Londrina (1996), mestrado em Ciência da Computação pela Universidade de São Paulo (1999), doutorado em Ciência da Computação pela Universidade de São Paulo (2004). Atualmente é professora na Universidade de São Paulo, Ribeirão Preto. Esta pesquisadora coordenou nos últimos anos dois projetos FAPESP: Projeto Jovem Pesquisador e TIDIA-Ae. Esta pesquisadora tem desenvolvido trabalhos e publicado artigos nas áreas extração e relacionamento de informação.

2.5 Agradecimentos

Os autores agradecem à FAPESP pelo apoio na apresentação do trabalho e a empresa SIAM pelo ambiente proporcionado.

2.6 Referencias

Barros, V. F. A. and Pinto JR, J and Borges, R. C. (2011) "*Aplicativo Móvel para Automação e Monitoração do Sistema de Atenção Primária a Saúde"***.** Cadernos de Informática V.6. Universidade Federal de Goiás. Goiânia, Goiás, p 241-244.

Bardram, J. E. (2005) "The trouble with login on usability and computer security". Proceedings on Ubiquit Computing, p 355-367.

Bolzani, C. A. M. (2004) "Residências Inteligentes: um curso de Domótica" (2004). 1. ed. São Paulo: Editora Livraria da Física, p. 332.

Deitel, P. J. and Deitel H. M. (2008) "AJAX, Rich Internet Applications, and Web Development for Programmers". Person Education Inc.

Dias, C. L. A. and Pizzolato, N. D. (2004) "Domótica – Aplicabilidade e Sistemas de Automação Residencial". Vértices, volume 6, setembro/dezembro.

Murakimi, A. and Gutierrez, M. A.and Lage, S. H. G. and Rebelo, M. F. S.and Ramires, J. A. F. (2006) “A Continuous Glucose Monitoring System in Critical”. IEEE Computers in Cardiology, v. 32, 2006, p. 10-14.

Moore, B. (2011) “A Health(care) conscious Technology”. Disponível em: www.rfid.org . Acesso dia 20/08/2013.

W3C. (2013) “Consórcio World Wide Web”. Url: http://w3c.com/. Acesso dia 20/08/2013.

WHATWG. (2013) “Web Hypertext Application Technology Working Group” (2013). url: http://www.whatwg.org/. Acesso dia 20/08/2013.

Capítulo

3

Desenvolvimento de Ambientes Virtuais Interativos usando Java e Kinect

Almerindo Nascimento Rehem Neto, Celso Alberto Saibel Santos, Lucas Aragão de Carvalho e Clebeson Canuto dos Santos

Abstract

This course has a mission to show the new concepts of the framework OpenNI – Open Natural Interaction – version 2.2 - Open Natural Interaction and the development of applications using hardware capture depth, such as the Microsoft Kinect sensors and Carmine 1.08 PrimeSense's so adherent the standards of OpenNI.

Resumo

Este curso tem a missão de mostrar os novos conceitos do framework OpenNI – Open Natural Interaction – versão 2.2 e o desenvolvimento de aplicações que utilizem hardware de captura de profundidade, como os sensores Kinect da Microsoft e o Carmine 1.08 da Primesense, de forma adepta aos padrões do OpenNI.

3.1. Introdução

A evolução das tecnologias para captura de interações dos usuários, muitas vezes de forma ubíqua, criou novos paradigmas para a concepção de interfaces para aplicações interativas. O *Wiimote* [Nintendo 2009] e o *Kinect* [Microsoft 2010, OpenKinect 2011, PrimeSense 2011] são os dois principais ícones comerciais que caracterizam essa nova forma de se pensar em interfaces, facilitando a comunicação entre usuário e computador por meio de interações naturais.

O *Wiimote* (ou *Wii Remote*), lançado em 2006, baseado em acelerômetros, é o dispositivo principal de interação do console *Nintendo Wii*. O *Kinect*, fabricado pela *PrimeSence*, é o dispositivo principal de interação com o console *Xbox* 360, lançado em 2010 pela *Microsoft*, que trouxe como grande inovação uma interface baseada no reconhecimento de gestos. Apesar do foco inicial na área de jogos interativos,

Este material de suporte do minicurso é uma reprodução autorizada pela SBC do Capítulo 3 do Livro "Tutoriais do X Simpósio Brasileiro de Sistemas Colaborativos e XII Simpósio Brasileiro sobre Fatores Humanos em Sistemas Computacionais" **apresentado no SBSC IHC 2013.**

pesquisadores e desenvolvedores têm utilizado estas plataformas como base para a construção de interfaces naturais para aplicações interativas [Shape Quest 2010, Saffer 2009, Shiratori and Hodgins 2008, Nakra et al 2009].

No caso do *Kinect*, em particular, menos de uma semana após seu lançamento oficial, a empresa *Adafruit* lançou o desafio "*Open KinectChallenge*", que oferecia U$3.000 para o primeiro desenvolvedor de um driver de código aberto para o novo dispositivo [Adafruit 2011]. Com o desafio e a recompensa lançados, não demorou para que a primeira implementação *opensource* de drivers para o *Kinect* (a biblioteca *libfreeneck* [OpenKinect 2011]) fosse apresentada, abrindo o caminho para o desenvolvimento de aplicações com novas formas de interação fora da área de jogos.

3.2. Interação Natural

Rehem et al. [Rehem 2011] consideram que a interação natural procura formas de tornar uma Interface o mais natural possível, com objetivo de fazer com que as pessoas manipulem uma máquina sem perceber qual(is) artefato(s) é (são) utilizado(s) e sem necessariamente aprender um novo vocabulário correspondente ao conjunto de instruções desta interface. Uma das principais motivações da área de Interação Humano-Computador (IHC) é melhorar a adaptação dos sistemas computacionais às necessidades do usuário [Hewett et al 1996]. É exatamente neste sentido que caminham as chamadas interfaces de interação natural .

Para Heckel [Heckel 1993], a interface deve procurar reproduzir a linguagem natural do usuário de computador ou da máquina, tentando prover o uso de palavras e expressões conhecidas, respeitando o vocabulário do usuário. Esta tentativa terá seus reflexos na redução do uso da memória e na diminuição do esforço cognitivo do usuário e, consequentemente, influirá numa interação mais simples, agradável e natural.

Valli [Valli 2007] define a interação natural em termos experimentais: as pessoas naturalmente se comunicam através de gestos, expressões, movimentos, além de explorarem o mundo real através da observação e manipulação de objetos físicos. E como uma tendência cada vez mais forte, as pessoas querem interagir com a tecnologia da mesma forma como lidam com o mundo real no cotidiano. A criação de novos paradigmas de interação e padrões alternativos de mídia que explorem as novas capacidades dos sensores presentes nas máquinas, e ainda, respeitem a espontaneidade inerente ao modo como o ser humano descobre e interage com o mundo é o principal desafio da construção de novas interfaces para interação. A tendência de quebra do paradigma tradicional das interfaces é evidenciada através de diversos experimentos usando novas propostas de interação surgindo cotidianamente na Web [Kinecthacks 2010] e do interesse de empresas como a Microsoft, que começa a dar importância na exploração dessas novas formas de interação [Toyama 1998]. Desta forma, os usuários têm sido cada vez mais direcionados a um cenário de computação ubíqua no qual os serviços digitais (ou computacionais) são disponíveis e acessíveis a partir de qualquer lugar, permitindo que as pessoas interajam de forma bastante natural com esses serviços [Cordeiro 2009], [Gregory et al 2000], [Weiser and Brown 1996].

Saffer [Saffer 2009] considera um gesto na interação natural usuário computador como qualquer movimento físico que um sistema digital possa reconhecer e ao qual possa responder. Um som, um inclinar de cabeça ou até mesmo uma piscada de olho podem ser considerados gestos. Um dos problemas enfrentados neste tipo de

reconhecimento é estabelecer o contexto das interações de forma a facilitar o reconhecimento dos gestos e a possibilitar a construção de interfaces naturais para cada tipo de cenário de utilização.

Gallud et al. [Gallud et al 2010] definem a comunicação não verbal como aquela que utiliza pistas e sinais sem estrutura sintática e separa essa comunicação em três tipos:

- Paralinguagem: Refere-se aos elementos que acompanham expressão linguística, consistindo de pistas e sinais que sugerem uma interpretação diferente do que está sendo dito, percebe-se pela intensidade e/ou altura da voz do emissor bem como chorar, rir, controle respiratório, etc.
- Linguagem corporal: Caracteriza-se pela utilização de gestos, porém seguida de características distinguíveis utilizando o tronco, extremidade e cabeça também pela proximidade entre emissor e receptor, expressões de estado como felicidade, honestidade, cumplicidade, etc.
- Linguagem sonora: Acompanha os gestos e caracteriza-se pela interpretação qualquer som que expresse emoção, contextualiza uma cena e, até mesmo, o silêncio pode ter significado em uma mensagem.

A detecção e interpretação de tantos sinais para uma resposta adequada dependem de análise e contextualização extremamente rápidas dos sinais recebidos. Assim, inúmeros fatores podem influenciar o entendimento correta de uma mensagem na comunicação usuário computador.

Se for possível utilizar linguagem corporal e falada como intervenções de usuário, abre-se a possibilidade de se criarem interfaces através das quais o usuário e a máquina podem se comunicar de forma mais natural. Por outro lado, a criação de interfaces baseadas na comunicação por gestos e voz traz consigo um novo tipo de problema: Quais gestos de interação e comandos associados devem ser utilizados para realizar uma determinada tarefa? Como facilitar o desenvolvimento de aplicações que necessitem utilizar gestos a principal interação com o sistema? Alguns trabalhos focam nestes problemas, ainda em aberto, para tentar auxiliar os desenvolvedores na redução de esforço e tempo de trabalho, como é o caso do *Touch The Air Framework* [Rehem, Santos and Carvalho 2013], que utiliza como base o *framework* desenvolvido pela *PrimeSense* [OpenNI 2011].

A proposta deste capítulo é apresentar uma forma padronizada de desenvolver módulos de componentes que somados viabilizariam a construção de ambientes virtuais interativos baseados na comunicação por gestos. Através da abordagem apresentada, os componentes gerados podem ser compartilhados e reutilizados para aplicações de diferentes finalidades, além de permitirem a criação de um ecossistema de bibliotecas e aplicações para auxiliar os desenvolvedores da área em questão.

3.3. Os Novos Dispositivos de Interação Humano-Computador

O *Wiimote* e o *Playstation Move* se parecem muito em termos de funcionamento. Eles são baseados essencialmente em emissão e detecção de luz infravermelha e uma câmera de RGB tradicional. A luz infravermelha é detectada pela câmera RGB e assim, é possível detectar a posição do ponto de referência: no caso do *Wiimote* esse ponto é o próprio controle e no caso do Playstation Move, é a peça com uma extremidade

luminosa que melhora a precisão que é dada pela câmera RGB. Este processo pode ser feito de forma 2D resultando numa posição na tela, exatamente como um ponteiro de mouse, ou de forma 3D, onde para detectar a distância do controle até a tela é utilizada uma triangulação. Além disso, quando há um acelerômetro disponível ao conjunto de sensores, além de posição e distância, as inclinações em qualquer eixo também podem ser detectadas. As inclinações definem os chamados graus de liberdade (*Degrees Of Freedom* – DOF) para os movimentos do controle (ver Figura 3.1).

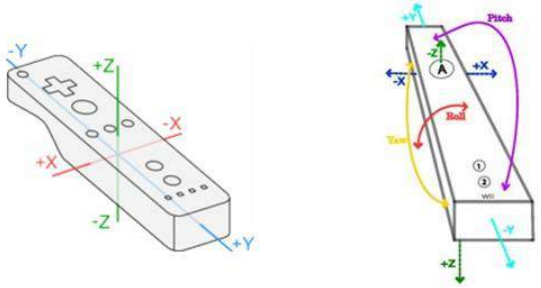


Figura 3.1.Eixos de referência e (b) os seis graus de liberdade do Wiimote. Fontes: [Wiiremote 2010],[Twiky 2010]

De fato, estas técnicas já eram conhecidas desde a década de 80 quando a Nintendo lançou o controle *Power Glove* (Figura 3.2) que funcionava de maneira muito similar aos dispositivos *Wiimote* e *Playstation Move*, com a diferença que, ao invés do infravermelho, utilizavam-se sensores de ultrassom. Estes sensores permitiam medir o tempo entre a reflexão das ondas emitidas por um aparelho posicionado na TV e o material da luva para determinar a posição da luva.

Figura 3.2. Nintendo Power Glove. Fonte: [BlaGames 2008]

A principal inovação trazida pelo *Kinect* é a geração de um mapa de profundidade, que possibilita ao dispositivo entregar informações 3D da cena completa, incluindo o jogador/ator e não somente informações do controle. Para se conseguir este resultado, utiliza-se o conceito de luz estruturada: uma luz com um padrão de projeção bem definido, que permite o cálculo das informações da cena partindo da combinação entre os padrões do que é projetado e da distorção da projeção em função dos obstáculos encontrados pela luz (ver Figura 3.3).

Figura 3.3. Luz estruturada (a) usada no mapeamento da face [18] e (b) Nuvem de pontos de luz emitidos a partir do sensor IR do Kinect.

Para conseguir esses resultados, o Kinect possui um sofisticado algoritmo de processamento paralelo, embarcado no chip (*SoC – System on Chip*) desenvolvido pela *PrimeSense*, necessário para extrair o mapa de profundidade [PrimeSense 2011]. Para ter maior precisão nos dados dos sensores, as imagens são alinhadas pixel a pixel, ou seja, cada pixel de imagem colorida é alinhado a um pixel da imagem de profundidade. Além disso, o chip *SoC* sincroniza (no tempo) todas as informações dos sensores (profundidade, cores e áudio) e as entrega através do protocolo USB 2.0 [Crawford 2010],[PrimeSense 2011].

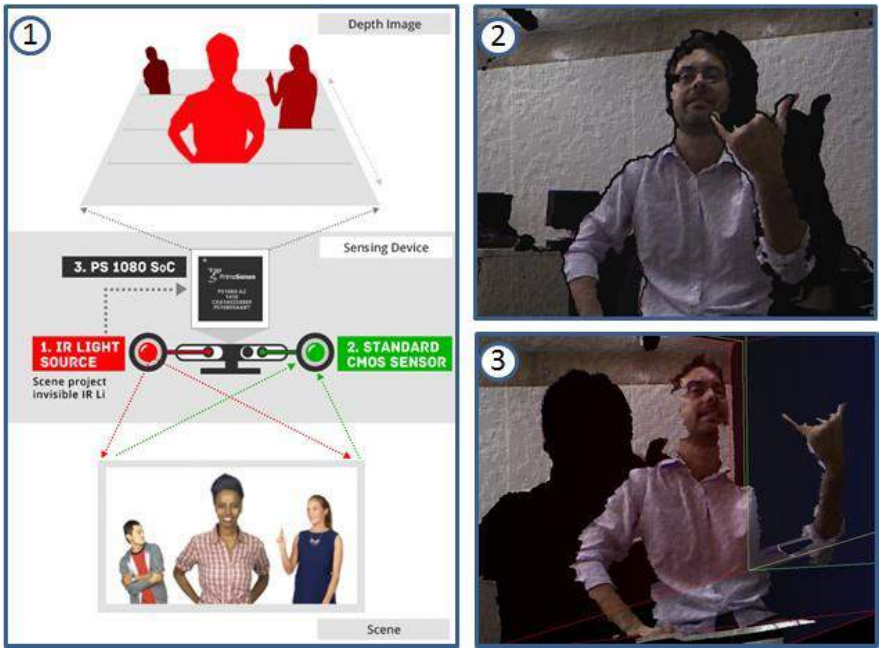


Figura 3.4. (1) Exemplo do diagrama do sensor da PrimeSense [PrimeSense 2011]. (2) e (3) são imagens RGB com seus pixels associados à profundidade.

A Figura 3.4 mostra um diagrama de funcionamento de sensores 3D baseados no chip da *PrimeSense* e exemplificados em três passos. O primeiro e o segundo passo, ilustram os sensores de IR e RGB tendo seus dados capturados e processados pelo chip *SoC*, que faz todo o processamento necessário para definir e entregar o mapa de profundidade ilustrado, como resultado final, o segundo e o terceiro itens da figura.

O *Kinect* dispõe de vários recursos (som, imagem, profundidade, infravermelho, motor de movimentação) com um alto índice de precisão e sincronismo em um único dispositivo. Estes recursos oferecem diversas possibilidades de interação entre os usuários e serviços e aplicações computacionais. Contudo, as formas e tecnologias para acesso a esses recursos podem ser as mais variadas possíveis. Na tentativa de se resolver o impasse, a *PrimeSense* disponibilizou o *framework OpenNI* (*Open Natural Inrterface*) e o módulo NiTE (*Natural Integration Middleware*) [PrimeSense 2011]. As tecnologias, tratadas na sequência do texto, permitem construir interfaces de aplicações sofisticadas, em um nível mais alto de abstração, tendo como base a combinação dos recursos disponíveis em sensores 3D e sem se restringir somente à plataforma *Kinect*.

3.4. O Framework *OpenNI*

O termo *OpenNI* designa uma instituição sem fins lucrativos e também uma marca registrada da *PrimeSense* [PrimeSense 2011]. A *OpenNI* busca promover a

interoperabilidade e compatibilidade entre aplicações, *middleware* e dispositivos de interação natural, atuando na verificação e certificação da conformidade das soluções de diferentes fabricantes e desenvolvedores aos padrões definidos pelo *framework OpenNI*.

O principal propósito da instituição *OpenNI* é especificar uma API padrão para a comunicação, tanto para dispositivos (sensores), quanto para aplicações e *middlewares*. Com isso, busca-se quebrar a dependência entre os sensores e os *middlewares*, além de permitir o desenvolvimento e portabilidade de aplicações para diferentes módulos de *middlewares* com menor esforço adicional (conceito "*write once, deploy everywhere*"). Mais ainda, a API fornece: (i) o acesso direto aos dados brutos dos sensores aos desenvolvedores e (ii) a possibilidade de se construir dispositivos que funcionem em qualquer aplicação compatível com o padrão proposto aos fabricantes.

O *framework OpenNI* habilita aos desenvolvedores acompanharem as cenas do mundo real (cenas em 3D), utilizando tipos de dados processados a partir da captura de um sensor de entrada (por exemplo, a representação do contorno de um corpo humano ou a localização de uma mão numa cena). Graças ao padrão *OpenNI* essas cenas são construídas com um alto nível de desacoplamento, isto é, podem ser construídas de forma independente dos fabricantes de sensores ou de um determinado *middleware*. A Figura 3.5 oferece uma visão abrangente das camadas da arquitetura *OpenNI* e dos conceitos associados ao padrão. Estas camadas são divididas em *Top*, *Middle* e *Botton*. Elas podem ser descritas da seguinte forma:

- ***Top***: representa os *softwares* que implementam aplicações de Interação Natural seguindo o padrão *OpenNI*;
- ***Middle***: representa o próprio *framework OpenNI* e oferece interfaces de comunicação para interagir tanto com os *middlewares*, quanto com os dispositivos (sensores);
- ***Bottom***: ilustra a camada de mais baixo nível, onde se situam os dispositivos de hardware responsáveis por capturar elementos visuais e auditivos da cena.

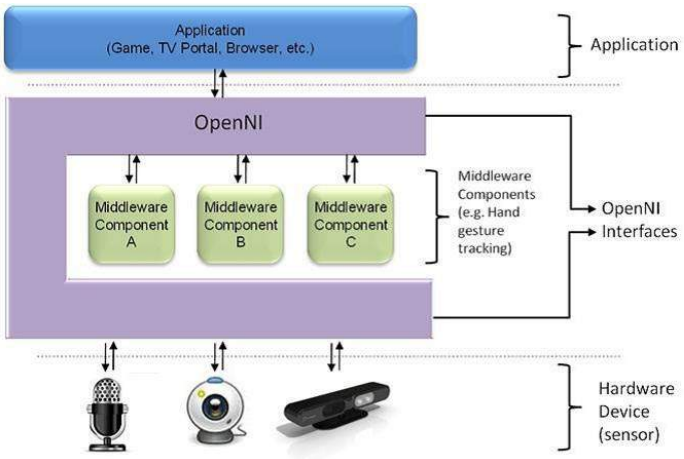


Figura 3.5. Abstração e componentização do *OpenNI*. Fonte: [PrimeSense 2011]

Pelo exposto até aqui, o conceito de *framework* está associado a uma camada de abstração que fornece interfaces para os componentes de softwares (*middlewares*) e para os dispositivos físicos através de uma API. Esta API habilita registros de múltiplos componentes (módulos) no *framework* que são responsáveis por produzir ou processar dados dos sensores físicos de forma flexível. Atualmente, os componentes suportados

pela arquitetura *OpenNI* são agrupados em dois tipos, os módulos de sensores e *middlewares*, descritos como se segue:

Módulos de sensores:

1. **Câmera RGB**: dispositivo responsável por gerar imagens coloridas;
2. **Sensor 3D**: dispositivo capaz de gerar o mapa de profundidade;
3. **IR câmera**: dispositivo de infravermelho;
4. **Dispositivo de áudio**: um ou mais microfones.

Middlewares:

1. **Análise de corpo**: componente de software capaz de processar os dados de entrada dos sensores e retornar informações relacionadas ao corpo humano. Como por exemplo, uma estrutura de dados que descreva as articulações, orientação e o centro de massa de uma pessoa na cena;
2. **Análise de mão**: componente capaz de processar dados dos sensores, reconhecer e retornar uma coordenada de localização do centro de uma mão em cena;
3. **Detecção de gestos**: componente capaz de identificar gestos predefinidos e alertar a aplicação todas as vezes que houver ocorrência desses gestos;
4. **Análise de cena**: componente capaz de analisar os dados da cena e produzir informações como: (i) a separação dos planos foreground e do background; (ii) as coordenadas do chão e (iii) a identificação de atores em cena.

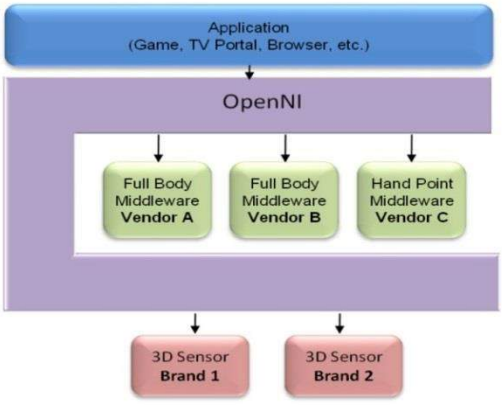


Figura 3.6. Exemplo de uso da arquitetura onde são registrados cinco módulos simultâneos em uma única instância do *OpenNI*. Fonte: [PrimeSense 2011]

A Figura 3.6 ilustra um exemplo de aplicação dos componentes da arquitetura para construir um cenário onde cinco módulos são registrados para trabalhar em uma única instância do *framework OpenNI*. Dois dos módulos registrados são sensores 3D conectados fisicamente ao *host*. Os outros três, são componentes de *middleware*, sendo que os dois primeiros módulos retornam dados relacionados ao corpo humano (*full body*) e o último, dados relacionado à mão (*hand point*).

Atualmente, o *framework* está passando por um redesenho de sua arquitetura. A versão estável anterior do *OpenNI* (versão 1.5), embora tenha impulsionado a comunidade científica e o desenvolvimento de diversas aplicações, ao ganhar popularidade, sofreu diversas críticas pela complexidade do seu uso. A versão 1.5, além de possuir conceitos com um nível de complexidade que dificultam o entendimento do

funcionamento do *framework*, também exigia o uso de muitas linhas de código para a concepção de tarefas simples. Após diversas sugestões de melhorias no *framework* e visando atender às necessidades da comunidade científica e dos desenvolvedores, a *PrimeSense*, principal mantenedora da *OpenNI*, iniciou a concepção de uma nova arquitetura (denominada *OpenNI* 2.x), em tese, mais simples e fácil de se utilizar.

Os próximos tópicos irão detalhar a API *OpenNI* em sua versão 2.2 (a mais atual no momento). É importante ressaltar que, devido ao seu lançamento recente, no momento ainda não é possível encontrar uma documentação detalhada sobre a nova arquitetura *OpenNI* e sua integração com a linguagem Java. Por consequência, a concepção deste texto exigiu estudos e análises cautelosas de todas as classes (e métodos associados) da versão 2.2 do *framework OpenNI*. Como contribuição, o texto traz uma série de comentários e explicações que buscam facilitar o entendimento das classes e dos principais métodos da nova versão do *framework*, além de apresentar exemplos práticos que auxiliam a instalação e demonstração do seu funcionamento.

3.4.1 Entendendo a estrutura da API Java *OpenNI*

Este tópico trata das classes (e de seus respectivos métodos) que fazem parte do *wrapper* Java, disponível na versão 2.2 do *OpenNI*, além de detalhar o funcionamento desses métodos, com o objetivo de possibilitar um melhor entendimento e utilização dos recursos disponibilizados.

Tabela 3.1.Relação das classes do *wrapper Java OpenNI2.2*

CLASSES	CLASSE DE APOIO	DESCRIÇÃO
CoordinateConverter.java	SIM	Usada para converter pontos entre sistemas de coordenadas diferentes.
CropArea.java	SIM	Classe de apoio que serve para encapsular recortes de informações.
Device.java	**NÃO**	**Responsável por toda a comunicação da aplicação com o dispositivo de Hardware.**
DeviceInfo.java	SIM	Classe de apoio que guarda atributos de determinado dispositivo.
ImageRegistrationMode.java	SIM	Classe de apoio que fornece nomes de sequencia para os valores dos códigos dos tipos de registros de Imagem.
NativeMethods.java	SIM	Classe que encapsula toda a API em C/C++ através do JNI para que se possa utilizá-la desde o JAVA.
OpenNI.java	**NÃO**	**Classe principal que faz a configuração do SDK e abri/fecha a comunicação com os Drivers.**
OutArg.java	SIM	Classe de apoio que define tipos de dados para a classe NativeMethods.
PixelFormat.java	SIM	Classe de apoio que define os tipos de pixels a ser utilizados.
PlaybackControl.java	NÃO	**Classe que gerencia reprodução de arquivos .ONI.**
Point2D.java	SIM	Classe de apoio para manipulação de pontos em 2D.
Point3D.java	SIM	Classe de apoio para manipulação de pontos em 3D.
Recorder.java	**NÃO**	**Classe responsável pela gravação em disco das capturas de uma aplicação (grava arquivos .ONI).**
SensorInfo.java	SIM	Classe de apoio que guarda informações de in determinado tipo de sensor. Informações como o tipo do sensor e o modo de vídeo utilizado.
SensorType.java	SIM	classe de apoio que armazena os três tipos de sensores ser utilizados (RGB, Infravermelho e de Profundidade).
Version.java	SIM	Classe de apoio que retorna qual a versão do OpenNI está sendo utilizada.
VideoFrameRef.java	**SIM**	**Classe que armazena os dados e atributos de cada frame que é capturado pelo dispositivo.**
VideoMode.java	**NÃO**	**Classe que configura o modo de vídeo a ser utilizado**
VideoStream.java	**NÃO**	**Classe responsável pela transmissão das capturas de frames feita pelo dispositivo.**

O *wrapper Java* do *OpenNI* 2.2 é formado de 19 classes das quais 7 são, consideradas como principais pelos autores deste texto. Este agrupamento se deve ao fato destas classes serem as principais responsáveis pelo gerenciamento dos dispositivos e pela captura e gravação dos dados das cenas. As demais classes são consideradas como de apoio, uma vez que são voltadas, de alguma forma, ao auxílio das chamadas classes principais. A Tabela 3.1 faz uma breve descrição do objetivo de cada classe Java e destaca em negrito as classes que foram categorizadas como sendo mais relevantes para os desenvolvedores que utilizam o *wrapper Java OpenNI 2.2*.

3.4.2 Detalhando as Classes do Wrapper Java OpenNI 2.2

Nesta seção serão explicadas todas as classes citadas na Tabela 3.1, mostrados os principais métodos associados, além de alguns códigos de exemplo de uso. É importante ressaltar que todas as documentações disponíveis no site oficial da *OpenNI* [OpenNI 2011] estão voltadas para a API original em C++. A documentação aqui apresentada é fruto de um trabalho árduo de engenharia reversa e análise dos códigos disponibilizados no site.

a) A classe Device

A classe Device é a responsável por estabelecer a conexão com um único dispositivo físico. Ela possibilita também simular um dispositivo de hardware, caso ele não esteja conectado ao computador, através da leitura das informações armazenadas em arquivo com extensão ".oni" (seção k)). Este último arquivo deve ser previamente criado e possuir informações das capturas de um dispositivo físico para que se consiga fornecer os dados necessários à simulação do *Kinect*, por exemplo.

Tabela 3.1.Métodos da Classe Device

Métodos	Descrição
open() e open(String uri)	Para abrir conexão com algum dispositivo é utilizado o método, open(). A depender dos parâmetros passados é possível comunicar com um dispositivo físico (hardware) ou simular um através de conexão com arquivo em disco na extensão .oni. Para conectar-se a um dispositivo físico podemos utilizar o construtor default ou passar o caminho completo do dispositivo, que poderia ser a URI da porta USB onde ele está.
close()	É responsável por fechar a conexão com o dispositivo. Esse método deve ser chamado sempre que se haja terminada a aplicação ou quando o dispositivo não vá mais ser utilizado.
getDeviceInfo()	Retorna as informações do dispositivo que se esta utilizando. Esse retorno vem na forma de um objeto do tipo DeviceInfo.
getHandle()	Utilizado pelo JNI para identificar o endereço de memória C/C++ onde se encontra este objeto.
getImageRegistrationMode()	Retorna um objeto do tipo ImageRegistrationMode.
isFile()	Retorna um boolean indicando true se o Device foi criado através de um arquivo ou false se é uma comunicação direta com o hardware.
getPlaycackControl()	Esse método permite controlar a execução da simulação de um dispositivo físico. Caso o dispositivo físico seja realmente um hardware ao invés de simulação através de arquivo .oni, este método irá retornar sempre um objeto nulo.
getSensorInfo(SensorType type)	Retorna o objeto SensorInfo que contem informações associadas ao tipo do sensor passado como parâmetro (SensorType).
hasSensor(SensorType type)	Esse método permite saber se o dispositivo conectado possui um determinado tipo de sensor passado como parâmetro, por exemplo RGB,IR ou DEPHT.
isImageRegistrationModeSupported (ImageRegistrationMode mode)	Indica se o dispositivo suporta ou não o registro de um determinado tipo de imagem o qual está sendo passado como parâmetro.
setDepthColorSyncEnabled (boolean bln)	Permite capturar,de forma síncrona, os sensores DEPHT e RGB.
setImageRegistrationMode (ImageRegistrationMode mode)	Configura o Device para operar com um determinado tipo de imagem ImageRegistrationMode.

Um dispositivo físico tem como objetivo capturar *streams* a todo o momento, sendo que as capturas podem ser feitas por tipos distintos de sensores. Assim, o objetivo da classe *Device* é comunicar-se com esse dispositivo físico e capturar a informação "crua" de cada um de seus sensores. Após capturar essas informações, outra classe poderá armazená-las num arquivo para que a aplicação possa utilizá-las posteriormente.

Para que a classe *Device* possa se comunicar com um dispositivo físico, é necessário que este esteja conectado ao computador e que todos os seus *drivers* de comunicação sejam corretamente instalados. Caso algo não esteja sendo feito de maneira correta, pode ser que a classe *Device* não encontre o dispositivo e então será lançada a mensagem de exceção "*No Device is Connected*".

A Tabela 3.1 ilustra todos os métodos da classe Device, bem como as explicações dos conceitos de cada um deles. A seguir, demonstraremos como criar e utilizar um objeto do tipo Device para conectar com os dispositivos.

O Método *open*() – construindo um *Device*

O método *open*() é permite iniciar uma conexão com algum dispositivo. A depender dos parâmetros passados é possível se comunicar com um dispositivo físico (*hardware*) ou simular um através da conexão com arquivo de extensão ".oni" em disco. Para conectar-se a um dispositivo físico, pode-se utilizar o construtor *default* ou passar, como parâmetro, o caminho completo do dispositivo, que poderia ser a URI da porta USB onde ele está conectado. Os exemplos abaixo ilustram o uso do método:

Conecta-se com o primeiro dispositivo físico encontrado.
Device.open();

Conecta-se com o dispositivo que estiver conectado através da URI passada por parâmetro.
Device.open(String URI);

Caso existam mais de um dispositivo conectado ao computador e o desenvolvedor não saiba qual dos dispositivos é o correto, pode-se utilizar o método *enumeratDevices*(), da classe *OpenNI* (seção f)), para se obter a lista de dispositivos conectados no momento. Essa lista contém objetos do tipo *DeviceInfo* (Tabela 3.1), que são responsáveis por guardar informações dos dispositivos. Cada objeto *DeviceInfo* possui um método chamado *getURI*() que retorna o caminho completo do dispositivo em questão, facilitando assim o trabalho do desenvolvedor.

O trecho de código da Figura 3.7 ilustra a utilização de uma conexão com o primeiro dispositivo físico, utilizando os métodos citados até o momento.

```
17          //Obtendo uma lista com os dispositivos conectados ao sistema
18          List<DeviceInfo> devicesInfo= OpenNI.enumerateDevices();
19          //Verificando se existe dispositivo conecta ao sistema
20          if (devicesInfo.isEmpty()) {
21              JOptionPane.showMessageDialog(null, "No device is connected",
22                                            "Error", JOptionPane.ERROR_MESSAGE);
23              return;
24          }
25          //Referenciando o DeviceInfo da posição "0" da lista "devicesInfo".
26          DeviceInfo deviceInfo=devicesInfo.get(0);
27          //Criando um Device a partir da URI obtida de "deviceInfo"
28          Device device=Device.open(deviceInfo.getUri());
```

Figura 3.7. Trecho de código com a utilização de um objeto do tipo Device

b) A classe PlaybackControl

A facilidade de se reproduzir dados previamente capturados permite que os desenvolvedores possam trabalhar simulando o ambiente desejado, o que é muito interessante para reprodução e depuração durante o desenvolvimento. Isto possibilita ainda reduzir custos, pois não é necessário adquirir um sensor 3D para cada desenvolvedor construindo um ambiente interativo. Essa característica de simulação também permite que se realize algum tipo de tratamento dos dados capturados antes de apresentá-los, ou mesmo, a visualização destas informações em um momento posterior, como no caso de sistemas de câmeras de segurança.

A classe *PlaybackControl* controla a reprodução de um arquivo ".oni" (seção k)) e serve para simular, a partir deste arquivo, um dispositivo físico. Com ela é possível controlar a velocidade de reprodução, posicionar a reprodução em um determinado

frame, saber qual a duração da reprodução, dentre outras funcionalidades. A Tabela 3.2 ilustra detalha todos os métodos da classe *PlaybackControl*.

A sequência do texto demonstrará como é possível criar e utilizar um objeto do tipo *PlaybackControl* para reprodução das *streams* gravadas em um arquivo .oni.

Tabela 3.2. Métodos da Classe PlaybackControl

Método	Descrição
seek(VideoStream stream, int i)	Esse método retorna uma determinada imagem de índice ***i***. Ou seja, como o arquivo utilizado para a simulação do dispositivo físico é formado por um conjunto de imagens capturadas e indexadas de maneira sequencial. Pode-se, com esse método, posicionar a execução em um ponto especifico desse conjunto. Para isso, se faz necessário indicar onde estão as imagens e qual o seu índice através dos parâmetros ***stream*** e ***i***, respectivamente.
getNumberOfFrames(VideoStream stream)	Permite saber quantos *frames* tem a gravação ao total. Assim é possível estimar o tempo que vai levar a execução ou mesmo delimitar o numero máximo do índice que pode ser passado no parâmetro ***i***.
getSpeed()	Esse método retorna um *float* que diz qual a velocidade atual da execução.
setSpeed(float speed)	Permite configurar a velocidade de execução.
getRepeatEnable()	Esse método retorna um boolean que indica se a reprodução irá se repetir ou não.
setRepeatEnable(boolean bool)	Configura o modo de reprodução em *loop* conforme o valor passado como parâmetro.

O Método Device.get*PlaybackControl* () - construindo um *PlaybackControl* :

Não é possível criar um objeto *PlaybackControl* diretamente. Esse tipo de objeto só pode ser construído através da classe *Device*, pois ela é a responsável por fazer uma conexão com os dispositivos físicos ou virtuais. O trecho de código da Figura 3.8 ilustra como criar um objeto *PlaybackControl* de forma correta.

```
31          PlaybackControl pbControl;
32          //Verificando se o Device foi criado a partir de um arquivo
33          if(device.isFile()){
34              pbControl=device.getPlaybackControl();
35          }
36          else{
37              JOptionPane.showMessageDialog(null, "Device is not a file.",
38                                            "Error", JOptionPane.ERROR_MESSAGE);
39              return;
40          }
```

Figura 3.8. Criando um objeto *PlaybackControl* a partir da instância de uma classe *Device*

Na linha 33 do código da Figura 3.8 é feita a verificação se o *Device* está conectado a um dispositivo virtual, isto é, se os dados das *streams* são provenientes de um arquivo .oni. Se esta condição for satisfeita, na linha 34, o objeto *PlaybackControl* será obtido através do método get*PlaybackControl*() descrito na Tabela 3.1.

c) A classe VideoStream

A classe *VideoStream* é responsável por encapsular todos os fluxos de dados criados pela classe *Device*. Permite configurar o modo em que o vídeo será executado, além de diversas outras configurações relacionadas ao vídeo. Esta é uma das classes do *OpenNI* que é indispensável para construir qualquer aplicação, uma vez que, para exibir ou tratar um conjunto de imagens, se faz necessário que todo o fluxo das capturas de informações passe por esse objeto. É a única classe que pode se comunicar com a classe *Device* para obter os *frames* capturados pelo dispositivo. Por ser tão importante, ela é a classe mais extensa, em quantidade de métodos, da API.

O Método *create*() – construindo um *VideoStream*

A classe *VideoStream* é responsável por se conectar em um determinado sensor de um dispositivo. Para isso, se faz necessário passar como parâmetros um dispositivo e o tipo de sensor que será utilizado para fornecer os dados da *stream*.

Para inicializar um objeto do tipo *VideoStream* se faz necessário chamar o método abstrato desta classe, intitulado de *create*(), passando como parâmetros um objeto do tipo *Device* e um *SensorType*.

O trecho de código da Figura 3.9 (linha 34) demonstra como instanciar um objeto do tipo *VideoStream* que está associado a um sensor RGB. Pode-se observar que, somente será criado um *VideoStream*, após verificado se o dispositivo suporta sensores do tipo RGB (linhas 26, 27 e 31).

```
25          //Obtendo e referenciando um SensorType do tipo COLOR
26          SensorType type=SensorType.COLOR;
27          boolean isSupported=device.hasSensor(type);
28          //Criando uma variável VideoStream
29          VideoStream videoStream;
30         //Verificando se o dispositivo suporta sensor do tipo COLOR (RGB)
31          if(isSupported){
32          //Criando um objeto VideoStream, responsável pela devolução
33          //dos dados capturados, e referenciando-o com a variável videoStream.
34          videoStream=VideoStream.create(device, type);
35          }else{
36            JOptionPane.showMessageDialog(null, "Sensor not supported.",
37                                          "Error", JOptionPane.ERROR_MESSAGE);
38              return;
39          }
```

Figura 3.9. Criando um *VideoStream*

A Tabela 3.3 faz um resumo de todos os outros métodos da classe *VideoStream*, com o objetivo de fornecer os conceitos de utilização de cada um deles.

Tabela 3.3.Métodos da Classe VideoStream.

Método	Descrição
create(Device device, SensorType type)	É um método abstrato da classe VideoStream que é responsável por criar um objeto deste tipo, já que essa classe não possua construtor.
addNewFrameListener(VideoStream.NewFrameListener nl)	Quando um frame é capturado o dispositivo gera um evento para avisar ao VideoStream que tem um novo frame a ser buscado. Este método é utilizado para adicionar um listener, chamado de NewframeListener, ao VideoStream.
removeNewFrameListener(VideoStream.NewFrameListener nl)	Remove um NewFrameListener antes adicionado. Desta forma, o VideoStream ficará sem receber eventos desse listener.
destroy()	Serve para limpar toda a memória utilizada pelo VideoStream.
getCameraSettings()	Retorna um objeto do tipo CameraSettings.
isCroppingSupported()	Indica se o VideoStream suporta recortes.
getCropping()	Retorna um objeto do tipo CropArea.
setCropping(CropArea ca)	Passa um objeto do tipo CropArea para o VideoStream.
getHandle()	Retorna o numero do endereço de memória onde está o objeto VideoStream que foi criado pelo código nativo em C/C++.
getHorizontalFieldOfView()	Obtém o campo de visão horizontal dos frames recebidos a partir desse stream.
getVerticalFieldOfView()	Obtém o campo de visão vertical dos frames recebidos a partir desse stream.
getMaxPixelValue()	Fornece o valor máximo para pixels obtidos pelo VideoStream.
getMinPixelValue()	Fornece o valor mínimo para pixels obtidos pelo VideoStream.
getMirroringEnabled()	O VideoStream tem a capacidade de fazer refletir os pixels da imagem através de um eixo vertical, fazendo com que a imagem fique como se estivesse sendo refletida por um espelho. Ou seja, esse método é o responsável por indicar se a reflexão esta ativada ou não.
setMirroringEnabled(boolean bln)	É responsável por ativar ou desativar a reflexão dos pixels de um VideoStream.
getSensorInfo()	Como o VideoStream foi criado a partir do objeto Device, é possível saber as informações de seus sensores através desse método.
getSensorType()	Fornece qual o tipo de sensor utilizado para adquirir as imagens, retornado assim um objeto do tipo SensorType.
getVideoMode()	Retorna um objeto do tipo VideoMode que configura os parâmetros do modo de vídeo utilizados pelo VideoStream.
setVideoMode(VideoMode mode)	Permite a configuração do objeto VideoStream através das informações encapsuladas em um VideoMode.
readFrame()	É utilizado para ter acesso ao novo frame capturado através do objeto de retorno do tipo VideoFremeRef.
start()	Inicia a busca pelas imagens capturadas disponíveis no objeto Device.
stop()	Esse método interrompe a busca de imagens no Device. Ele pode ser chamado quando se deseja pausar a exibição do um vídeo, por exemplo.

d) A classe VideoStream.CameraSettings

CameraSettings é uma classe interna da classe *VideoStream* que fornece possibilidades de configurações de uma câmara RGB. Ela permite ativar ou desativar o balanço automático do branco e também a exposição automática.

A Tabela 3.4 ilustra todos os métodos da classe *CameraSettings*, bem como as explicações dos conceitos associados a cada um deles. Na sequência do texto será explicado o funcionamento do método utilizado para ter acesso ao objeto *CameraSettings* (Figura 3.10).

Tabela 3.4.Métodos da Classe VideoStream.CameraSettings

Método	Descrição
getAutoExposureEnabled()	Verifica se a opção de exposição automática está habilitada.
getAutoWhiteBalanceEnabled()	Verifica se o balanço automático de branco está habilitado.
setAutoExposureEnabled (boolean enabled)	Habilita ou desabilita a exposição automática da câmara.
setAutoWhiteBalanceEnabled (boolean enabled)	Habilita ou desabilita o balanceamento automático de branco da câmara.

Acessando o objeto *CameraSettings*:

Não existe construtor para esta classe. Um objeto desse tipo só poderá ser obtido através da chamada do método *getCameraSettings*() do *VideoStream*.

```
31        SensorType type=SensorType.COLOR;
32        //Criando um Device
33        Device device=Device.open();
34        //Criando um objeto VideoStream, responsável pela devolução
35        //dos dados capturados, e referenciando-o com a variável stream.
36        VideoStream stream=VideoStream.create(device, type);
37        //Criando um CameraSettings
38        VideoStream.CameraSettings cs =stream.getCameraSettings();
39
```

Figura 3.10. Criando um objeto do tipo *CameraSettings*

e) A classe VideoMode

Um objeto da classe *VideoMode* é utilizado para configurar o comportamento da captura dos frames do sensor de vídeo. Os atributos disponíveis para essas configurações incluem: (i) o formato de pixel das capturas; (ii) a resolução da imagem e (iii) a taxa de frames por segundo (fps). Assim, é possível determinar, por exemplo, que cada imagem tenha resolução de 640x480 pixels ou que a taxa de captura de imagens seja 30 fps.

A Tabela 3.5 traz todos os principais métodos da classe *VideoMode* e fornece os conceitos básicos para utilização de cada um deles. Em seguida, um trecho de código ilustrando a criação de um objeto do tipo *VideoMode* é apresentado.

Tabela 3.5. Métodos da Classe VideoMode

Método	Descrição
setResolution(int resX, int resY)	Permite passar a resolução em que as imagens devem ser capturadas.
setPixelFormat(PixelFormat pf)	Configura o formato de *pixel* desejado para as imagens.
setFps(int fps)	Possibilita determinar a quantidade de *frames* que devem ser capturados a cada segundo.

Construindo um *VideoMode*:

Para se criar um objeto do tipo *VideoMode*, se faz necessário configurar pelo menos os atributos relacionados à taxa fps, ao formato do pixel e às resoluções no eixo X e no eixo Y. O trecho de código da Figura 3.11 ilustra como instanciar o objeto da classe *VideoMode* com o construtor padrão (linha 45), como configurar esses parâmetros utilizando os métodos *setters* (linhas 47, 49 e 51) e como criar um outro objeto da mesma classe com a passagem de todos os parâmetros no construtor (linha 54).

```
44          //Criando um modo de video (VideoMOde)  vazio
45          VideoMode mode=new VideoMode();
46          //Passando a resolução dascapturas
47          mode.setResolution(640,480);
48          //Passando a quantidade de FLOPS
49          mode.setFps(30);
50          //Passando o tipo de Pixel
51          mode.setPixelFormat(PixelFormat.RGB888);
52
53          //Criando um VideoMode já com todas as características
54          VideoMode mode1=new VideoMode(640,480,30,PixelFormat.RGB888.toNative());
55
```

Figura 3.11. Exemplos de como criar um *VideoMode*

f) A classe OpenNI

A classe *OpenNI* é responsável pela inicialização da aplicação, permitindo o acesso às bibliotecas ou SDK. É por meio desta classe que é estabelecida a conexão com o dispositivo físico, através da verificação de seus respectivos *drivers* instalados.

Tabela 3.6. Métodos da Classe OpenNI

Método	Descrição
initialize()	Responsável por toda a configuração do ambiente, desde a checagem dos drives até a carga das bibliotecas. Importante ressaltar que deve ser executado antes do método de criação de um *Device*. Do contrário, o dispositivo físico não vai poder ser localizado pelo *Device*.
enumerateDevices()	Retorna uma lista de objetos do tipo *DeviceInfo*, que nada mais é que uma lista com informações de todos os dispositivos físicos conectados ao sistema.
getExtendedError()	Retorna a descrição de um erro. Esse método é comumente utilizado quando se deseja saber qual foi o erro que gerou uma determinada exceção.
waitForAnyStream(List<VideoStream> list, int i)	Aguarda até que um novo *frame*, de qualquer um dos *VideoStreams* pertencentes à lista, seja identificado ou que o seu tempo de espera tenha sido ultrapassado.
shutdown()	Cancela a comunicação da aplicação com o dispositivo. Deve ser chamado no final de cada aplicação para que o dispositivo físico seja liberado.
addDeviceConnectedListener(OpenNI.DeviceConnectedListener dl)	Adiciona um novo *listener* de conexão de dispositivo à aplicação.
addDeviceDisConnectedListener(OpenNI.DeviceDisConnectedListener dl)	Adiciona um novo *listener* de desconexão de dispositivo à aplicação.
addDeviceStateChangedListener(OpenNI.addDeviceStateChangedListener dl)	Adiciona um novo *listener* de verificação de estado de um dispositivo à aplicação.
removeDeviceConnectedListener(OpenNI.DeviceConnectedListener dl)	Remove um determinado *listener* de conexão de dispositivo.
removeDeviceDisConnectedListener(OpenNI.DeviceDisConnectedListener dl)	Remove um determinado *listener* de desconexão de dispositivo.
removeDeviceStateChangedListener(OpenNI.addDeviceStateChangedListener dl)	Remove um *listener* de verificação de estado de um dispositivo.

A classe *OpenNI* possui três *listeners* (*DeviceConnectedListener*, *DeviceDisconectedListener* e *DeviceStateChangedListener*) que escutam o estado do dispositivo físico e disparam eventos sempre que um dispositivo físico seja conectado, desconectado ou seu estado tenha sido alterado sistema.

Essa classe não tem construtor, pois todos os seus métodos são abstratos. Contudo, seu método *initialize*() deverá ser o primeiro a ser chamado em uma aplicação, pois, sem essa chamada, não será possível acessar os dispositivos e seus sensores disponíveis. De toda forma, é importante conhecer todos os métodos disponíveis nessa classe, bem como seus métodos. A Tabela 3.6 ilustra todo o conjunto de métodos disponíveis na classe e uma breve explicação de cada um destes métodos.

g) A classe SensorType

A classe *SensorType* utilizada apenas para armazenar definições de tipos de sensores suportados[1]. Basicamente, ela não possui métodos disponíveis e dentre as definições armazenadas, estão:

- **IR**: para imagens geradas a partir do sensor de infravermelho;
- **COLOR**: para imagens geradas a partir do sensor RGB e;
- **DEPTH**: para imagens geradas a partir do sensor de profundidade.

h) A classe SensorInfo

Um objeto desse tipo armazena informações do sensor que são utilizados pela classe *Device* e *VideoStream*. Dentre as informações presentes nesse objeto, podemos encontrar o tipo de sensor (*SensorType*), além de uma lista de modos de vídeo (*VideoMode*) suportados pelo sensor.

[1] Na versão atual do *OpenNI*, não existe suporte aos sensores de áudio e motor.

Conforme mostrado na Tabela 3.7, essa classe possui apenas dois métodos. Um exemplo de como construir um *SensorInfo*, bem como a utilização de um de seus métodos comumente utilizado, são descritos a seguir.

Tabela 3.7. Métodos da Classe SensorInfo

Métodos	Descrição
getSensorType()	Retorna o tipo de sensor *(IR, RGB ou DEPTH)*.
getSupportedVideoModes()	Retorna uma lista de modos de vídeo suportados pelo sensor.

Construindo um objeto do tipo *SensorInfo*:

Esse objeto não possui construtor. Sua instância só poderá ser obtida através de um objeto do tipo *Device* ou *VideoStream*. Os trechos de código abaixo ilustram capturas das instâncias de *SensorInfo* obtidas através do *Device* (a) e do objeto *SensorInfo* (b).

```
SensorType type = SensorType.DEPTH;
Device device= Device.open();
VideoStream stream=VideoStream.create(device, type);

//Obtendo um SensorInfo
SensorInfo info;
//Através de um Device
info =device.getSensorInfo(type);
```

```
SensorType type = SensorType.DEPTH;
Device device= Device.open();
VideoStream stream=VideoStream.create(device, type);

//Obtendo um SensorInfo
SensorInfo info;
// ou Através de um VideoStream
info=stream.getSensorInfo();
```

Figura 3.12. Obtendo SensorInfo através do *Device* (a) e (b) do *VideoStream*

O Método *getSupportedVideoModes*()

O trecho de código ilustrado a seguir na Figura 3.13 ilustra como utilizar o objeto *SensorInfo* para verificar os modos de operação suportados pelo sensor. O exemplo mostra como a lista de configurações suportadas por um sensor RGB pode ser capturada. Esse tipo de abordagem é a mais recomendada, uma vez que evita que os desenvolvedores emitam parâmetros de configurações inválidos ou não suportados pelo sensor em que está tendo acesso no momento.

```
30          SensorType type=SensorType.COLOR;
31          boolean isSupported=device.hasSensor(type);
32          //Criando uma variável VideoStream
33          VideoStream videoStream;
34         //Verificando se o dispositivo suporta sensor do tipo COLOR (RGB)
35          if(isSupported){
36          //Criando um objeto VideoStream, responsável pela devolução
37          //dos dados capturados, e referenciando-o com a variável stream.
38          videoStream=VideoStream.create(device, type);
39          }else{
40            JOptionPane.showMessageDialog(null, "Sensor not supported.",
41                                          "Error", JOptionPane.ERROR_MESSAGE);
42              return;
43          }
44          //Obtendo uma lista com os modos de vídeo suportados pelo sensor COLOR
45          List<VideoMode> supportedVideoModes=videoStream.getSensorInfo().
46                                              getSupportedVideoModes();
47
```

Figura 3.13. Exemplo de uso do método *getSupportedVideoModes*()

i) A classe DeviceInfo

A classe *DeviceInfo* encapsula todas as informações referentes a um dispositivo. Algumas dessas informações comumente utilizadas pelos desenvolvedores são o nome do dispositivo, nome do seu fabricante e a URI.

A Tabela 3.8 lista todos os métodos disponíveis nessa classe e faz uma breve explicação dos conceitos de cada um desses métodos. Note que os objetos do tipo *DeviceInfo* devem ser obtidos através do método *getDeviceInfo*() do classe *Device* (seção a)).

Tabela 3.8. Métodos da Classe DeviceInfo

Métodos	Descrição
getName()	Retorna uma *String* com o nome do dispositivo.
getUri()	Retorna a rota de conexão do dispositivo no sistema: *URI*.
getUsbProductId()	Retorna o identificador do dispositivo USB (*PID*).
getVendor()	Retorna o nome do fabricante do dispositivo.
getUsbVendorId()	Retorna o identificador do fornecedor/fabricante do dispositivo (*VID*).

j) A classe VideoFrameRef

A classe *VideoFrameRef* encapsula todos os dados relacionados a um determinado *frame* obtido por um *VideoStream*. É a principal classe utilizada pelo *VideoStream* para retornar cada novo *frame* capturado. Um objeto *VideoFrameRef* permite acessar um *buffer* que contém os dados do *frame*, bem como quaisquer outros metadados que se façam necessários para trabalhar com o *frame* atual.

Construindo um *VideoFrameRef*

Dado que a classe *VideoFrameRef* não possui construtor, um objeto desta classe exige um objeto *VideoStream* e uma chamada ao método *readFrame* para sua instanciação. O exemplo da Figura 3.14 ilustra uma criação de um *VideoFrameRef*. Porém, o mais natural seria que o *VideoFrameRef* fosse obtido dentro de um método que captura o evento produzido pelo *NewFrameListener*. Isto porque, o *VideoFrameRef* guarda informações de apenas um *frame* e assim, se for necessário capturar todos os *frames* de uma sequência, deve ser criado um *VideoFrameRef* para cada nova captura.

```
30  SensorType type=SensorType.COLOR;
31  Device device=Device.open();
32  VideoStream stream= VideoStream.create(device, type);
33  //Criando um FrameRef
34  VideoFrameRef ref =stream.readFrame();
35
```

Figura 3.14. Criando um VideoFrameRef

A Tabela 3.9 lista todos os métodos disponíveis na classe *VideoFrameRef*, bem como uma breve descrição dos conceitos de cada um desses métodos.

Tabela 3.9. Métodos da Classe VideoFrameRef

Método	Descrição
getCropOriginX()	Indica a coordenada ***X*** do canto superior esquerdo da janela onde será obtida a informação. Retorna a distância de origem do recorte a partir do lado esquerdo da imagem, em *pixels*.
getCropOriginY()	Indica a coordenada ***Y*** do canto superior esquerdo da janela onde será obtida a informação. Retorna a distância de origem do recorte da parte superior da imagem, em *pixels*.
getCroppingEnabled()	Indica se a propriedade de recorte está ativa no momento em que o frame tenha sido capturado.
getData()	Retorna um buffer com todos os bytes/pixels do frame capturado.
getFrameIndex()	Este método retorna o número de identificação do frame atual, que é atribuído ao frame no momento em que é capturado.
getHeight()	Fornece a altura do *frame* atual, medida em *pixels*. Este método depende da ativação ou não da opção de recorte. Caso a opção de recorte estiver ativada, o método retornará a altura da janela de recorte. Do contrário, irá retornar a resolução ***Y*** do *VideoMode*, utilizado para reproduzir o *frame*.
getSensorType()	Retorna o tipo de sensor utilizado para a captura do *frame*.
getStrideInBytes()	Fornece o comprimento de uma linha de *pixels*, medido em *bytes*.
getTimestamp()	Retorna o tempo da captura do *frame* atual, a partir de um tempo zero arbitrado pelo dispositivo.
getWidth()	Fornece a largura do frame atual, medida em pixels. Este método depende da ativação ou não da opção de recorte. Caso a opção de recorte estiver ativada, o método retornará a largura da janela de recorte. Do contrário, irá retornar a resolução **X** do *VideoMode*, utilizado para reproduzir o *frame*.
getVideoMode()	Retorna uma referência ao objeto *VideoMode* que associado ao *frame* atual.
release()	Libera a referência ao *frame*.

k) A classe Recorder

A classe *Recoder* é a responsável por criar e gravar em disco um arquivo ".oni", que contenha os dados capturados através de um sensor vinculado ao *VideoStream*. Isso permite que, posteriormente, esse arquivo seja utilizado como provedor de informações de um dispositivo virtual. Assim, a reprodução dos dados nele armazenados, simula um dispositivo físico em funcionamento (seção b)).

Todos os métodos desta classe estão na Tabela 3.10. Na sequência, um exemplo que ilustra a criação e inicialização do objeto *Recorder* para executar uma gravação dos *frames* em um arquivo em disco na Figura 3.15.

Tabela 3.10. Métodos da Classe Recorder

Método	Descrição
create (String fileName)	Método estático que cria um arquivo com o nome especificado em *fileName*. É indispensável que no nome do arquivo contenha também uma extensão, que aqui deve ser ***.ONI***, pois caso contrário o arquivo não poderá ser reproduzido.
addStream(videoStream stream, boolean bln)	Adiciona uma ou mais *stream* para que seus *frames* seja guardados no arquivo ***.ONI*** . O parâmetro ***boolean bln*** diz respeito à compressão do arquivo. Aconselha-se que seja falso. Pois caso seja verdadeiro, o arquivo será salvo de maneira comprimida com perda de qualidade de informações. O que as vezes não é recomendável, pois a qualidade da reprodução poderá ser prejudicada.
start()	Esse método inicia a gravação dos dados do *stream* no arquivo.
stop()	Esse método da uma pausa na gravação dos dados.
destroy()	Finaliza a gravação dos dados junto com os objetos criados pela gravadora.

No trecho de código ilustrado na Figura 3.15, para que a classe *Recorder* funcione a contento, é necessário que tenha sido instanciada e inicializada passando como parâmetro o caminho onde será armazenado o arquivo ".oni" (linha 55). Em seguida (linha 59), o *Recorder* é vinculado a um ou mais *streams* e após isso, deve-se solicitar o início da gravação para que sejam armazenados cada um dos *frames* de cada *stream* no arquivo especificado como parâmetro.

```
53          //Criando uma gravadora e especificando o nome e extensão
54          //do arquivo onde serão gravado os dados
55          Recorder recorder= Recorder.create("Exemplo.ONI");
56          //Adicinando uma stream a qual fornecerá os dados a serem gravados
57          //e fornecendo a informação que diz se o arquivo deverá ser gravado de
58          //maneira compactada.
59          recorder.addStream(stream, false);
60          //Iniciando o processo de gravação dos dados
61          recorder.start();
62
```

Figura 3.15. Exemplo de como instanciar uma classe do tipo *Recorder*

l) A classe CordinateConverter

A classe *CordinateConverter* permite converter pontos entre os diferentes sistemas de coordenadas. Neste ponto, cabe uma breve explicação para facilitar o entendimento do leitor no que se refere aos conceitos e à importância desta classe. As aplicações *OpenNI* geralmente usam dois sistemas de coordenadas distintos para representar a profundidade. Estes dois sistemas de coordenadas são comumente relatados como sendo coordenadas de profundidade e coordenadas de representação no mundo real.

O sistema de coordenadas de profundidade trabalha sobre as representações de dados nativos extraídos diretamente dos sensores. Neste sistema, a estrutura é um mapa (matriz bidimensional) da imagem e um valor de profundidade é atribuído a cada pixel da imagem. Este valor de profundidade representa a distância entre o plano da câmara e qualquer objeto. As coordenadas X e Y são simplesmente o local no mapa, onde a origem é o canto superior esquerdo do campo de visão.

Já no sistema de coordenada do mundo real, cada coordenada é especificada pelos eixos X, Y e Z. Como exemplo, do ponto de vista da câmera, um objeto que se move da esquerda para a direita está se movendo ao longo do eixo X. Um objeto que se move para cima e para baixo está se movendo ao longo do eixo Y, e um objeto se afastando da câmera está se movendo ao longo do eixo Z. A Figura 3.16 ilustra a disposição dos eixos desse sistema de coordenadas.

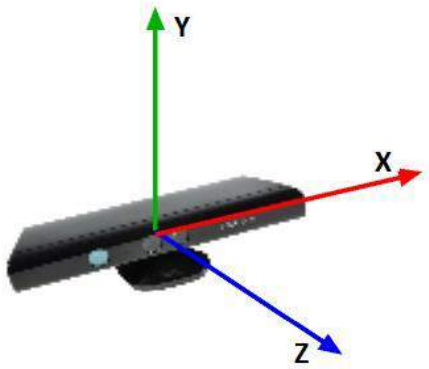


Figura 3.16. Sistema de Coordenadas do Mundo Real

Importante ressaltar que não se deve fazer conversão dos sistemas de coordenadas em tempo real devido ao alto custo de processamento. É aconselhável que o desenvolvedor trabalhe todo o software em cima do sistema de coordenadas nativo (sistema de coordenadas de profundidade) e faça a conversão somente no momento anterior que necessitar produzir a saída dos dados.

Todos os métodos disponíveis para a classe *CordinateConverter*, assim como uma breve descrição de cada um deles, estão descritos na Tabela 3.11.

Tabela 3.11. Métodos da Classe CordinateConverter

Métodos	Descrição
convertDepthToColor(VideoStream depthStream, VideoStream colorStream, int depthX, int depthY, DepthPixel depthZ, int pColorX, int pColorY)	Para um determinado ponto de profundidade, fornece as coordenadas do valor da cor correspondente.
convertDepthToWorld(VideoStream depthStream, int depthX, int depthY, DepthPixel depthZ, float pWorldX, float pWorldY, float pWorldZ)	Converte um único ponto do sistema de coordenadas de profundidade para o sistema de coordenadas de mundo real.
convertDepthToWorld (VideoStream depthStream, float depthX, float depthY, float depthZ, float pWorldX, float pWorldY, float pWorldZ)	Converte um único ponto, a partir de uma representação de ponto flutuante, de um sistema de coordenadas de profundidade em um sistema de coordenadas do mundo real.
onvertWorldToDepth (VideoStream depthStream, float worldX, float worldY, float worldZ, int pDepthX, int pDepthY, DepthPixel pDepthZ)	Converte um único ponto do sistema de coordenadas mundo real para o sistema de coordenadas de profundidade.
convertWorldToDepth (VideoStream depthStream, float worldX, float worldY, float worldZ, float pDepthX, float pDepthY, float pDepthZ)	Converte um único ponto, a partir de uma representação de ponto flutuante, de um sistema de coordenadas de mundo real, em um sistema de coordenadas de profundidade.

m) As demais classes

As demais classes serão apenas listadas e descritas sem maiores detalhes por se tratarem apenas de classes de apoio e por, em sua maioria, possuem apenas métodos *Getters* e *Setters* de fácil entendimento.

- **Classe *NativeMethods***: Responsável por todo o mapeamento da API *OpenNI* 2.x em C/C++ utilizado pelo *wrapper Java* da API 2.x para a comunicação com o código nativo através do JNI.

- **Classe *OutArg***: Serve de apoio à classe *NativeMethods* para definir tipos de variáveis utilizadas pelo código nativo.
- **Classe *CropArea***: Essa classe encapsula informações de recortes de dados.
- **Classe *PixelFormat***: Define os diferentes tipos de pixels utilizados pela API.
- **Classe *Version***: Identifica qual a versão atual do *OpenNI* que a aplicação está utilizando.
- **Classe *ImageRegistrationMode***: Dois ou mais dispositivos podem trabalhar em conjunto para a formação de uma só imagem com o *framework OpenNI*. Os dados dos dispositivos são sobrepostos fazendo com que, por exemplo, uma imagem RGB possa se sobrepor a uma imagem de profundidade, produzindo como resultado final uma única imagem. Alguns dispositivos possuem a capacidade de fazer cálculos matemáticos necessários para essas sobreposições em seu hardware. Esta classe permite saber se o dispositivo utilizado tem ou não essa capacidade para que possa gerenciá-la.
- **Classes *Point3D* e *Point2D***: Encapsulam, respectivamente, pontos de 3 dimensões e pontos de 2 dimensões.

3.5. Programando com o *OpenNI*

As seções seguintes deste capítulo terão foco nos aspectos práticos da programação utilizando o *framework OpenNI*, fornecendo o conhecimento básico e essencial para qualquer desenvolvedor iniciar suas aplicações relacionadas à interação natural dentro desta plataforma. O texto está estruturado de forma a apresentar: (i) a montagem e a configuração de um ambiente de desenvolvimento; (ii) a integração entre o *wrapper Java* do *OpenNI* [OpenNI 2011] e o *NetBeansIDE* [Netbeans 2012]; (iii) a criação de uma aplicação básica com Java e *OpenNI*; (iv) a criação de uma aplicação simulando um dispositivo físico e (iv) de uma aplicação para seguir os movimentos das mãos (*Hand Tracking*).

3.5.1 Montagem e configuração de um ambiente para o desenvolvimento

O ambiente de exemplo para criar aplicações integrando o *OpenNI* e a linguagem Java utiliza um *Kinect* como hardware de obtenção de dados, o *framework OpenNI*2.2 e o *middleware* NITE. O texto explica como obter todos os arquivos utilizados, como instalá-los e configurá-los e como testar todo o ambiente antes que o desenvolvedor inicie a programação. Em seguida, será mostrado exemplo de uma construção de aplicação própria no ambiente montado.

Este trabalho não abordará as instalações do sistema operacional Windows 7 Home basic (64 Bits), jdk 1.7.0_21 (64 Bits), jdk1.7.0_21 (32 Bits) e a IDE NetBeans IDE 7.3, utilizados como base para o ambiente demonstrado.

a) Instalação do Kinect SDK

Kinect SDK é o ambiente de programação criado pela Microsoft [Microsoft 2010], contendo os drivers e bibliotecas necessários para programação com *Kinect*. Para sua instalação e configuração, se faz necessário a execução dos passos que se seguem:

1. *Download* e instalação do KinectSDK através do link:

http://www.microsoft.com/en-us/kinectforwindows/develop/developer-downloads.aspx;

2. Após instalação concluída, conectar um *Kinect* na porta USB 2.0[2] e verificar se os *drivers* do sensor foram instalados com sucesso, acessando "Panel de controle» Sistema e segurança»Sistema» Gerenciador de dispositivo", conforme ilustra a Figura 3.17.

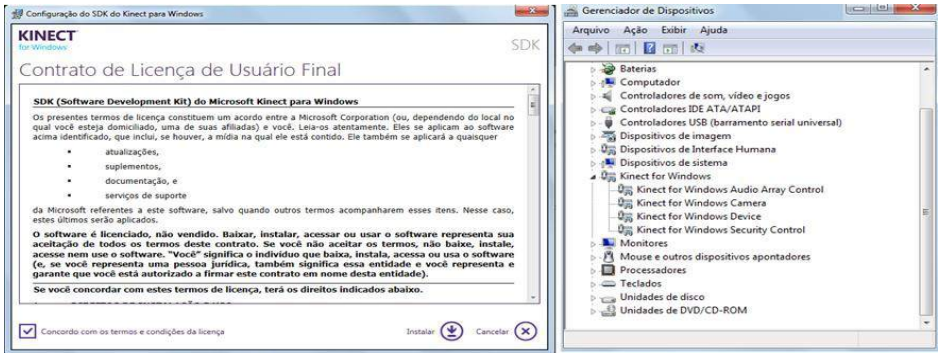


Figura 3.17. a) Instalando o Kinect SDK e b) Verificando se foi instalado corretamente no gerenciador de dispositivos

b) Instalação do OpenNI

O *OpenNI* é um SDK aberto, que é utilizado para desenvolvimento de bibliotecas de *middleware* e aplicações que utilizam sensores 3D. Esta plataforma fornece suporte a diversos sistemas operacionais e a sensores 3D distintos como o *Kinect* [Microsoft Kinect 2010], Carmine ou Xition [PrimeSense 2011]. A instalação e configuração do *OpenNI* será descrita e ilustrada a seguir:

1. É necessário obter o *OpenNI* através do *download* do SDK acessando a URL http://www.OpenNI.org/OpenNI-sdk/
2. Após a conclusão do *download*, executar o arquivo para instalá-lo mantendo suas configurações padrões, conforme ilustra (Figura 3.18).

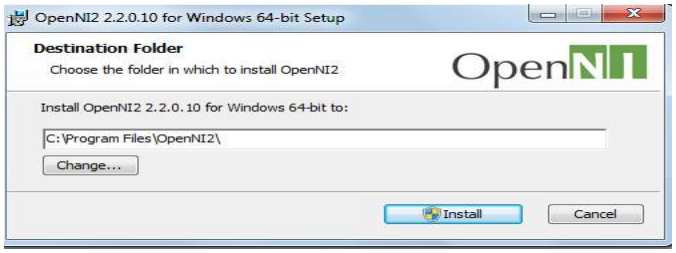


Figura 3.18. Instalação do *OpenNI*

c) Instalação do Middleware NITE

O *NITE* é um *middleware* de Visão Computacional 3D que possibilita criar aplicações que controlem movimentos da mão ou de todo o corpo. O presente trabalho não abordará a construção de aplicações com este *middleware*. Porém, no ambiente de desenvolvimento serão previstas a instalação e configuração do *NITE* para uso futuro. Os passos para instalação deste *middleware* se resumem a: (i) obter o *NITE* mais atual

[2] É recomendável o uso da porta USB 2.0, pois a conexão com uma porta 3.0 pode causar travamentos indesejáveis na obtenção de dados do Sensor.

(*NITE* 2.x) através da URL: http://www.OpenNI.org/files/nite/ e (ii) executar o arquivo para instalá-lo.

d) Testando a Instalação

Para vericar se a instalação está funcionando corretamente, basta acessar a pasta onde estão os exemplos do *OpenNI* ("C:\Program Files\OpenNI2\Samples\Bin", em geral) e executar um dos arquivos disponíveis. Se a execução mostrar uma imagem de algum sensor, significa que tudo foi instalado com sucesso. Na Figura 3.19 é ilustrado a execução, com sucesso, do exemplo *SimpleViewer*.

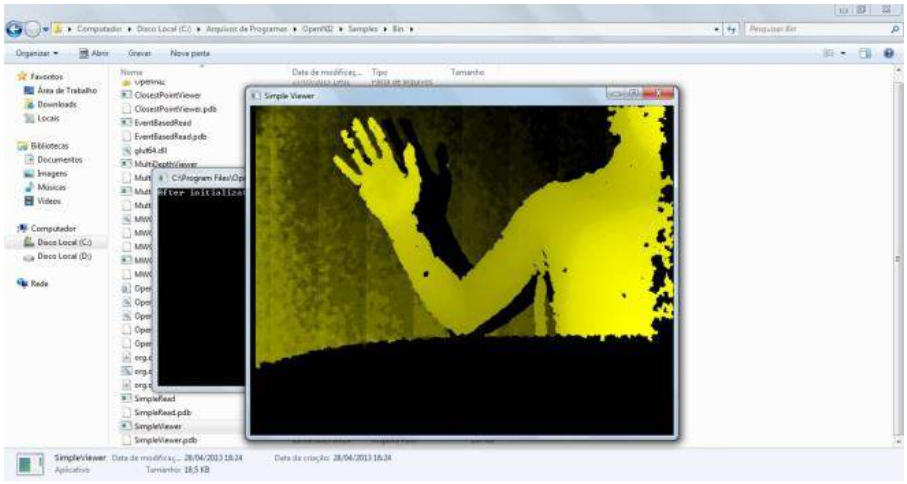


Figura 3.19. Executando o exemplo *SimpleViewer* para testar a instalação

e) Outras Considerações

O diagrama da Figura 3.20 ilustra como deveria ficar o ambiente de desenvolvimento após todas as instalações sugeridas nos tópicos anteriores. Estaria disponível para o desenvolvedor a IDE *NetBeans* 7.3, que poderá fazer compilações para computadores 64/32 bits, através dos dois tipos de JDK disponíveis na IDE. Ainda na IDE, será possível desenvolver códigos utilizando o *NITE*, o *OpenNI* ou ambos.

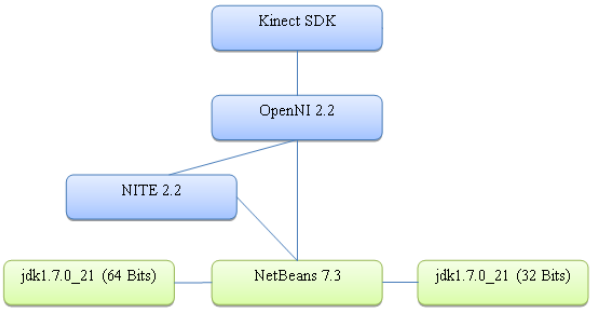


Figura 3.20. Diagrama de Componentes do Ambiente de Desenvolvimento

Importante notar que o *OpenNI*, no Sistema Operacional *Windows*, está diretamente ligado ao SDK da *Microsoft*. Pois, é com este SDK que a *Microsoft* disponibiliza os *drivers* necessários para acessar os sensores do dispositivo físico (Kinect). Logo, é por meio desses *drivers* que o *OpenNI* consegue capturar os dados de todos os sensores.

3.5.2 Integrando o *wrapper* Java do *OpenNI* com o *NetBeansIDE*

Nesta seção será mostrado como localizar o *wrapper* Java após a instalação dos pacotes necessários. Além disso, será mostrada toda a configuração necessária para que o usuário deixe a sua IDE *NetBeans* funcional e isenta de erros.

a) Wrappers Java

A localização dos *wrapper*s do *OpenNI* e do *NITE* ficam na pasta "Redist" de cada instalação. E, por padrão, se encontram nos respectivos endereços:

"C:\Program Files\OpenNI2\Redist\org.OpenNI.jar"

"C:\Program Files\PrimeSense\NiTE2\Redist\com.primesense.nite.jar"

```
Exception in thread "main" java.lang.UnsatisfiedLinkError: no OpenNI2.jni in java.library.path
  at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1860)
  at java.lang.Runtime.loadLibrary0(Runtime.java:845)
  at java.lang.System.loadLibrary(System.java:1084)
  at org.openni.NativeMethods.<clinit>(NativeMethods.java:44)
  at org.openni.OpenNI.initialize(OpenNI.java:113)
  at org.openni.Samples.SimpleViewer.SimpleViewerApplication.main(SimpleViewerApplication.java:193)

Exception in thread "main" java.lang.UnsatisfiedLinkError: no NiTE2.jni in java.library.path
  at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1860)
  at java.lang.Runtime.loadLibrary0(Runtime.java:845)
  at java.lang.System.loadLibrary(System.java:1084)
  at com.primesense.nite.NativeMethods.<clinit>(NativeMethods.java:31)
  at com.primesense.nite.NiTE.initialize(NiTE.java:15)
  at com.primesense.nite.Samples.UserViewer.UserViewerApplication.main(UserViewerApplication.java:71)
```

Figura 3.21. Exceção lançada quando não é encontrado o pacote java que possue a implementação JNI (o *wrapper* Java)

Para utilizar o *NetBeansIDE*, é importante colocar o endereço de ambos *wrapper*s dentro do path do *Windows*. Desta maneira, os programas em desenvolvimento escritos com o auxílio desta IDE serão executados com êxito. Caso contrário erros, conforme ilustra a Figura 3.21 , serão bastante comuns no momento da execução da aplicação.

b) Testando o NetBeans - Executando uma aplicação exemplo em Java

Para verificar se o ambiente foi montado e configurado corretamente, basta executar os exemplos disponibilizados pelo *framework*. Para isso, o texto discute os exemplos "SimpleViewer.java" e "UserView.java", que por padrão encontram-se na pasta Redist[3] da intalação *OpenNI* e do NITE , respectivamente.

No caso do exemplo "UserView.java", ou qualquer outro exemplo relacionado ao *middleware* NITE, se faz necessário copiar toda a pasta NITE2, que se encontra na pasta Redist, para o diretório raiz do projeto do *NetBeans*. Do contrário, ocorrerão "*exceptions*" pela falta do arquivo "s.dat", conforme ilustrado na Figura 3.22.

```
Could not find data file .\NiTE2/s.dat
current working directory = C:\Users\Lucas\Documents\NetBeansProjects\UserViewer
Exception in thread "main" java.lang.RuntimeException
  at com.primesense.nite.NativeMethods.checkReturnStatus(NativeMethods.java:40)
  at com.primesense.nite.UserTracker.create(UserTracker.java:90)
  at com.primesense.nite.Samples.UserViewer.UserViewerApplication.main(UserViewerApplication.java:80)
```

Figura 3.22. Exceção lançada ao não encontrar o arquivo de banco de dados utilizado pelo *NITE*

Para se executar os exemplos no *NetBeans*, deve-se criar um projeto, copiar os fontes disponíveis e fazer a importação das bibliotecas (org.OpenNI.jar), do *wrapper*

[3] Diretório padrão onde se encontram os exemplo disponibilizados do OpenNI e do NiTE são, respectivamente: C:\Program Files\OpenNI2\Redist. e C:\Program Files\Primesense\Nite2\Redist

Java do *OpenNI* e, (com.primesense.nite.jar) do *wrapper java* do NITE . Em seguida, deve-se compilar o projeto criado e executar separadamente cada um dos exemplos. O resultado final das execuções das aplicações exemplo, deve ser similar à Figura 3.23.

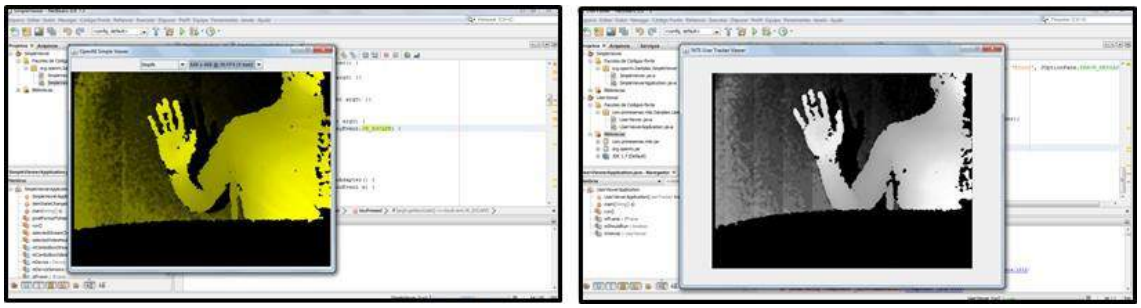


Figura 3.23. a) Resultado da execução do *SampleViewer.java* e b) Resultado da execução do *UserViewer.java*

3.5.3 Criando uma aplicação básica em Java com o *OpenNI*

Concluída a configuração do ambiente conforme os tópicos anteriores, esta parte do trabalho traz exemplos de como se construir uma aplicação básica para obter os dados do sensor RGB do *Kinect*. O exemplo visa colocar em prática os conhecimentos do *framework* e conceitos vistos até aqui.

3.5.4 Preparando o projeto Java, na IDE NetBeans, para fazer o primeiro exemplo com o *OpenNI*

Para que seja iniciado o desenvolvimento de uma aplicação Java com o *OpenNI*, é necessário que o usuário crie um novo projeto Java no *NetBeans*. E, após um novo projeto Java ter sido criado, é de fundamental importância importar as bibliotecas necessárias para programação com o *OpenNI* e *NITE*, localizadas em suas respectivas pastas *Redist* e cujos os nomes são *org.OpenNI.jar* e *com.primesense.nite.jar*[4].

a) Criando a primeira aplicação OpenNI: Iniciando e obtendo conexão com o sensor RGB

Basicamente, para se construir uma aplicação com o *OpenNI*, é necessário conhecer pelo menos quatro classes principais, que estão presentes na maioria das aplicações que utilizam esse *framework*. As principais classes são *OpenNI*, *Device*, *VideoStream* e *VideoMode*, ilustradas no diagrama de classes da Figura 3.24 e comentadas mais adiante. Embora estas classes já tenham sido citadas anteriormente (seção 3.4.2), é necessário compreender a relação entre elas.

A Classe *OpenNI* é responsável por inicializar todo o *framework* (seção f)). É através do método estático *OpenNI.initialize*() que é verificado quais os dispositivos são compatíveis e disponíveis. E, em seguida, carregados os drivers necessários para o acesso aos dispositivos no momento da execução da aplicação. Caso a aplicação não utilize esse método irá apresentar um erro comum no momento de sua execução. Será lançado uma exceção com o nome de *RuntimeException*.

4 Por ser trivial, assume-se que o leitor já possua os conhecimentos necessários para se criar um novo projeto java no NetBeans, bem como adcicionar bibliotecas .jar.

A Classe *Device* é a responsável por fazer uma conexão com o dispositivo físico ou virtual (seção a)). É a partir da sua instância que os dados necessários para o acesso aos sensores DEPTH, RGB e IR são obtidos.

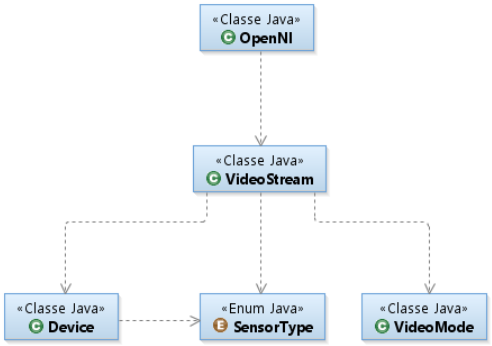


Figura 3.24. Diagrama das principais classes, comum na maioria das aplicações qualquer aplicação *OpenNI*

O trecho de código ilustrado na Figura 3.25 mostra como inicializar a biblioteca e criar um objeto do tipo Device

```
14  public class ExampleNI {
15
16      public static void main(String[] args) {
17          OpenNI.initialize();
18          Device device = Device.open();
19      }
20  }
```

Figura 3.25. Inicializando o *OpenNI* e criando um *Device*

A Classe *VideoStream* é a classe central do *OpenNI*. A partir dela é possível ler os dados provenientes do dispositivo(s) conectado(s) e encapsulá-los em um único fluxo, que além de fornecer os dados de vídeo, também fornece informações como campo de visão, modos compatíveis de vídeo, valores máximos e mínimos de pixels válidos, entre outras (seção c).

```
16  public class ExampleNI {
17
18      private static VideoStream videoStream;
19
20      public static void main(String[] args) {
21          OpenNI.initialize();
22          Device device = Device.open();
23          videoStream = VideoStream.create(device, SensorType.COLOR);
24      }
25  }
```

Figura 3.26. Criando um VideoStream

Para se capturar dados de um sensor, é necessário se criar de um fluxo de dados vinculado a algum tipo de sensor existente no dispositivo físico. O trecho de código que se segue (Figura 3.26) ilustra como criar um objeto do tipo *VideoStream* que encapsula o fluxo de dados proveniente de um sensor RGB.

Na linha 23, pode-se observar que no parâmetro do método *VideoStrem.create* (*device*, *SensorType.COLOR*), existe uma referência ao dispositivo físico (*device*) e a um tipo de sensor (*SensorType.COLOR*) ao qual o *VideoStream* deverá ser associado (ver seção g)). Neste caso, o *SensorType*.COLOR está se referindo ao sensor RGB, mas poderia ter sido associado a outros sensores, como os de profundidade e o de infravermelho.

A Classe *VideoMode* encapsula um grupo de informações do *VideoStream*. Como por exemplo, a resolução da imagem capturada, a taxa de quadros por segundo (fps) e o formato de pixel (seção e)).

O trecho de código da Figura 3.27 configura o sensor RGB no modo de funcionamento padrão (resolução de 640x480 pixels, taxa de 30 fps e formato dos pixels de 24 bits RGB). Na linha 26, através do método *getSupportedVideoModes*().*get*(1), é capturado o modo de funcionamento padrão do sensor. Em seguida, na linha 27, o *VideoMode* desejado é passado como parâmetro para o sensor RGB.

```
18  public class ExampleNI {
19
20      private static VideoStream videoStream;
21
22      public static void main(String[] args) {
23          OpenNI.initialize();
24          Device device = Device.open();
25          videoStream = VideoStream.create(device, SensorType.COLOR);
26          VideoMode mode = videoStream.getSensorInfo().getSupportedVideoModes().get(1);
27          videoStream.setVideoMode(mode);
28          videoStream.start();
29      }
30  }
31
```

Figura 3.27. Criando o VideoMode e iniciando a captura de dados do sensor

Como boa prática de programação, recomenda-se que se obtenha uma lista de configurações suportadas para um determinado sensor (utilizando o método *SensorInfo.getSupportedVideoModes*()). Em seguida, o desenvolvedor deve utilizar a lista obtida para configurar o sensor, reduzindo a possibilidade de que um modo inválido seja escolhido para o mesmo.

A partir deste ponto, o código está pronto para iniciar a captura dos dados do sensor RGB em tempo real e, para isso, basta que o método *videoStream.start*() seja chamado (Linha 28 da Figura 3.27). Porém, para que o usuário visualize os dados capturados em tela, alguns componentes gráficos do Java devem ser utilizados.

b) Visualizando os dados do sensor RGB

Embora não faça parte do *framework*, uma classe de exemplo (*SimpleViewer.java*) é disponibilizada junto com o pacote do *OpenNI*. Ela foi utilizada como modelo para criação do componente visual, denominado de *ComponentViewer*. A Classe *ComponentViewer* estende a classe *java.awt.Component* e implementa a interface *NewframeListener* da classe *VideoStream*. Ela serve basicamente para receber dados de um *VideoStream* e exibi-los na tela.

```
18  public class ExampleNI {
19
20      private static VideoStream videoStream;
21      private static ComponentViewer viewer;
22
23      public static void main(String[] args) {
24          OpenNI.initialize();
25          Device device = Device.open();
26          videoStream = VideoStream.create(device, SensorType.COLOR);
27          VideoMode mode = videoStream.getSensorInfo().getSupportedVideoModes().get(1);
28          videoStream.setVideoMode(mode);
29          viewer = new ComponentViewer();
30          viewer.setStream(videoStream);
31          viewer.setSize(mode.getResolutionX(), mode.getResolutionY());
32      }
33  }
```

Figura 3.28. Criação e configuração do component visual responsável por exibir os dados do sensor em tela

O trecho de código da Figura 3.28, nas linhas 29,30 e 31, ilustra a criação do componente visual, a passagem da referência do sensor utilizado e a configuração do tamanho em tela. Após o componente visual criado e configurado, se faz necessário

adicioná-lo a um *JFrame* e informar as propriedades necessárias para sua correta exibição em tela (Figura 3.29).

```
18  public class ExampleNI {
19
20      private static VideoStream videoStream;
21      private static ComponentViewer viewer;
22      private static JFrame frame;
23
24      public static void main(String[] args) {
25          OpenNI.initialize();
26          Device device = Device.open();
27          videoStream = VideoStream.create(device, SensorType.COLOR);
28          VideoMode mode = videoStream.getSensorInfo().getSupportedVideoModes().get(1);
29          videoStream.setVideoMode(mode);
30          viewer = new ComponentViewer();
31          viewer.setStream(videoStream);
32          viewer.setSize(mode.getResolutionX(), mode.getResolutionY());
33          frame = new JFrame("Example NI");
34          frame.setSize(viewer.getWidth() + 20, viewer.getHeight() + 80);
35          frame.add("Center", viewer);
36          frame.setVisible(true);
37          frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
38      }
39  }
```

Figura 3.29. Iniciando a captura do fluxo de dados do sensor RGB e exibindo o *ComponentViewer* em um *JFrame*

3.5.5 Simulando um *Kinect* fisicamente conectado ao computador

As seções anteriores mostraram como acessar um determinado sensor e iniciar a captura de seu fluxo de dados (seção a)). Nesta parte do texto será demonstrado como capturar o fluxo de dados de um sensor e armazená-lo em disco para futura utilização em uma simulação de dispositivo físico.

a) Gravando o fluxo de dados de um sensor em arquivo

Após a criação e conexão com um sensor físico, demonstradas na seção a), é possível armazenar o fluxo de dados obtido em um arquivo do disco rígido através da classe *Recorder*. Antes de descrever o funcionamento desta classe, cabe ressaltar que ela não faz conexão com um sensor físico. Esta função fica a cargo da classe *VideoStream* descrita anteriormente. A classe *Recorder* se limita a receber uma ou mais *streams* e um caminho de arquivo e gravar o fluxo de dados dessa(s) *stream*(s) no arquivo passado como parâmetro.

Para fazer a gravação em arquivo se faz necessário utilizar um objeto do tipo *Recorder* (já descrita na seção k)). Como parâmetro deve-se passar o caminho completo onde se deseja que o arquivo[5] seja salvo.

```
52          //Criando uma gravadora e especificando o nome e extensão
53          //do arquivo onde serão gravado os dados
54          Recorder recorder= Recorder.create("Exemplo.ONI");
```

Figura 3.30. Criando um Recorder e um arquivo .ONI

```
55          //Adicinando uma stream a qual fornecerá os dados a serem gravados
56          //e fornecendo a informação que diz se o arquivo deverá ser gravado de
57          //maneira compactada ou não.
58          recorder.addStream(videoStream, false);
59          //Iniciando o processo de obtenção dos dados
60          videoStream.start();
61          //Iniciando o processo de gravação dos dados
62          recorder.start();
```

Figura 3.31.Iniciando a classe recorder

Os trechos de códigos das FigurasFigura 3.30 e Figura 3.31 ilustram, respectivamente: (i) como criar e configurar um *Recorder* para que as informações

[5] Os arquivos utilizados pela classe *Recorder* obrigatoriamente deverão ter extensão ".oni".

capturadas sejam salvas no arquivo “./Exemplo.oni” e (ii) como iniciar a gravação do fluxo de dados capturados pelo *videoStream* (Linha 62).

b) Reproduzindo o arquivo com a gravação dos dados

Após o arquivo “.oni” ter sido gravado (seção a)), já é possível utilizá-lo como fonte de dados para simular um dispositivo físico. Nas etapas seguintes serão descritos os passos necessários para fazer essa simulação. Todas as etapas aqui descritas são bem semelhantes ao processo de obtenção de dados da seção a). Por esse motivo, em alguns casos, ao invés de descrevê-las novamente, apenas serão citadas as seções referentes às respectivas explicações.

Primeiro, deve-se criar um dispositivo virtual que segue os mesmos fundamentos do método *Open*(*String URI*) da classe *Device* (seção a)). A diferença é que, ao invés de se indicar no parâmetro URI o caminho de um dispositivo físico, será indicado o caminho e nome do arquivo “.oni” gravado em disco (seção a)). A Figura 3.32 mostra como criar um dispositivo através de um arquivo em disco.

```
12          //Inicializando o OpenNI
13          OpenNI.initialize();
14          //Criando um Device a partir de um arquivo .oni criado por um Recorder
15          Device device=Device.open("Exemplo.ONI");
```

Figura 3.32.Figure 1.32. Criando um Device a partir de um arquivo ".oni"

Depois de configurar o *Device* como dispositivo virtual, apontando para o arquivo “.oni”, deve ser inicializada a captura de um fluxo de dados de um dos sensores disponíveis, através do *VideoStream*, conforme ilustrado na linha 23 do trecho de código da Figura 3.26 (seção a)).

A classe que se utilizará para gerenciar toda a reprodução do arquivo é a *PlaybackControl* descrita na seção b). Para que fique mais claro, a Figura 3.33 mostra uma maneira correta de se obter um *PlayBackControl*, fazendo a verificação se o *Device* foi realmente configurado para ser um dispositivo virtual.

```
14          //Criando um Device a partir de um arquivo .oni criado por um Recorder
15          Device device=Device.open("Exemplo.ONI");
16          //Criando um PlaybackControl
17          PlaybackControl pbControl;
18          if(device.isFile()){
19              pbControl=device.getPlaybackControl();
20          }else{
21           JOptionPane.showMessageDialog(null, "Device isn't  a file.",
22                                         "Error", JOptionPane.ERROR_MESSAGE);
23              return;
24          }
```

Figura 3.33. Criando um PlaybackControl

Com a *PlaybackControl* é possível controlar toda a reprodução do fluxo de dados obtido de qualquer sensor virtual. O trecho de código da Figura 3.34 a seguir apresenta exemplos práticos, além de algumas funcionalidades da classe.

```
40          //Configurando tempo de execução
41          pbControl.setSpeed(1);
42          //Habilitando repetição automátic
43          pbControl.setRepeatEnabled(true);
44          //Obtendo o número total de frames da gravação
45          int numOfFrames=pbControl.getNumberOfFrames(videoStream);
46          //Iniciando obtenção dos dados
47          videoStream.start();
48          //Posicionando a execução no meio da gravação
49          pbControl.seek(videoStream, numOfFrames/2);
```

Figura 3.34. Exemplos das funcionaliddes - seek, repeat,

As funcionalidades destacadas no exemplo da Figura 3.34 permitem:

1. Saber qual a velocidade atual de reprodução, através do método *getSpeed*() ou indicar qual a velocidade desejada, através do método *setSpeed*(*float speed*) (linha 41).
2. Fazer com que a gravação possa se repetir a cada vez que chegar ao fim, através do método *setRepeatEnabled*(*boolean repeat*) (linha 43). Isso é muito útil nos casos em que a gravação não tem dados suficientes para suprir o tempo requerido pela simulação.
3. Identificar qual a quantidade total de frames da gravação para depois calcular a duração[6] da reprodução, através do método *getNumberOfFrames*(*VideoStream stream*) (linha 45).
4. Posicionar a reprodução em partes distintas da gravação, apenas fazendo uso do método *seek*(*int frame*)[7] (linha 49).

A partir deste ponto, é necessário iniciar a obtenção do fluxo, com o método *start*() do objeto *VideoStream* e exibir as imagens como mostrado na seção c) para simular a obtenção de dados de um dispositivo físico.

Para auxiliar o desenvolvedor, a Figura 3.35 mostra o código completo, necessário para se efetuar a simulação com todas as funcionalidades citadas até o momento.

```
11      public static void main(String[] args) {
12         //Inicializando o OpenNI
13         OpenNI.initialize();
14         //Criando um Device a partir de um arquivo .oni criado por um Recorder
15         Device device=Device.open("Exemplo.ONI");
16         //Criando um PlaybackControl
17         PlaybackControl pbControl;
18         if(device.isFile()){
19             pbControl=device.getPlaybackControl();
20         }else{
21          JOptionPane.showMessageDialog(null, "Device isn't  a file.",
22                                        "Error", JOptionPane.ERROR_MESSAGE);
23             return;
24         }
25         //Obtendo e referenciando um SensorType do tipo COLOR
26         SensorType type=SensorType.COLOR;
27         boolean isSupported=device.hasSensor(type);
28         //Criando uma variável VideoStream
29         VideoStream videoStream;
30        //Verificando se o dispositivo suporta sensor do tipo COLOR (RGB)
31         if(isSupported){
32         //Criando um objeto VideoStream, responsável pela devolução
33         //dos dados capturados, e referenciando-o com a variável videoStream.
34         videoStream=VideoStream.create(device, type);
35         }else{
36           JOptionPane.showMessageDialog(null, "Sensor not supported.",
37                                         "Error", JOptionPane.ERROR_MESSAGE);
38             return;
39         }
40         //Configurando tempo de execução
41         pbControl.setSpeed(1);
42         //Habilitando repetição automática
43         pbControl.setRepeatEnabled(true);
44         //Obtendo o número total de frames da gravação
45         int numOfFrames=pbControl.getNumberOfFrames(videoStream);
46         //Iniciando obtenção dos dados
47         videoStream.start();
48         //Posicionando a execução no meio da gravação
49         pbControl.seek(videoStream, numOfFrames/2);
50     }
```

Figura 3.35. Exemplo de código completo para a simulação de um dispositivo físico

[6] Para calcular o tempo total da reprodução apenas se faz necessário obter a quantidade total de frames, dividir pelo produto da velocidade de reprodução e a velocidade de captura (FLOPS).

[7] O *seek* só funciona caso o fluxo esteja em execução, caso contrário não será possível posicionar em um determinado *frame* do *stream*.

3.5.6 Criando aplicações para seguir os movimentos das mãos (Hand Tracking).

Antes de dar continuidade às explicações sobre como construir o exemplo para seguir os movimentos das mãos será feita uma breve explanação do *middleware* NITE, principal responsável pelos recursos utilizados no exemplo que se segue.

O *NiTE* é um *middleware* desenvolvido pela *PrimeSense* que utiliza o *framework OpenNI* para prover funções interativas aos aplicativos que utilizam sensores de profundidade, como por exemplo, o *Kinect* ou o *Carmine* 1.08 [OpenNI 2011],[PrimeSense 2011].

Enquanto o *OpenNI* é responsável pela comunicação e obtenção de dados oriundos dos dispositivos compatíveis e conectados ao sistema, o *NiTE* fica responsável pelo processamento e reconhecimento dos dados obtidos. Com este processamento, é possível: (i) obter quais são os usuários presentes na cena; (ii) identificar os pixels pertencentes a cada usuário e ao *background*; (iii) criar um mapeamento das principais articulações dos usuários e (iv) detectar a posição das mãos no espaço (*hand tracking*).

Dentre as diversas possibilidades oferecidas do *middleware* NITE, os exemplos que se seguem irão se restringir às características necessárias para a construção de uma aplicação com *hand tracking*. Para isso, a seção seguinte fará uma explanação apenas de algumas das características e métodos do *NITE* necessárias para a execução e o entendimento dos exemplos mostrados.

a) Criando um HandTracker utilizando o NITE.

No NITE, as classes disponíveis para a busca e identificação das mãos dos usuários presentes em uma cena são: *HandTracker*, *HandData* e *HandTrackerFrameRef*. A Tabela 3.12 descreve estas classes e sua utilização numa aplicação de *HandTracker*.

Tabela 3.12. Classes utilizadas para o HandTracker

Classes	Descrição
HandTracker	É a principal classe do algoritmo rastreador de mãos. Ela, junto com *UserTracker*, é uma das duas principais classes do *NITE*.
HandData	Esta classe encapsula os dados de uma mão durante um *frame* de detecção de mãos. Ela pode ser usada para descobrir: onde está a mão no espaço; o ID da mão e o status de rastreamento.
HandTrackerFrameRef	Contém todas as saídas, a partir de um único quadro de profundidade, do algoritmo de rastreamento da mão. Ela contém todas as mãos e gestos detectados, no momento atual.
GestureData	Esta classe armazena dados sobre um gesto que está a ser detectado. "Gestos", neste contexto, se referem aos movimentos das mãos detectados diretamente do DepthMap. Objetos desta classe armazenam os dados para uma instância específica de um determinado gesto.

O exemplo *HandTracker* exige a captura dos eventos de detecção de mão para cada *frame* enviado pelo dispositivo. Para isso o método *onNewFrame*(*HandTracker ht*) da interface *NewFrameListener*, explicado na Tabela 3.13 deve ser implementado.

Tabela 3.13. Objeto Listener responsável pela detecção da mão

Interfaces	Descrição
HandTracker.NewFrameListener	Esta é uma interface usada para reagir aos acontecimentos gerados pela classe HandTracker. Para usar essa interface, se faz necessário: • derivar uma classe a partir dela e implementar a função *onNewFrame()*. Esta é a função de retorno que será chamada quando um evento for gerado. • Criar uma nova instância de sua classe derivada. Em seguida, usar o *HandTracker.create()* • *E* adicionar seu escutador criado ao HandTracker através do método addNewFrameListener () Assim, o HandTracker lançará um evento sempre que um novo frame de detecção de mão for encontrado (*onNewFrame*). A função *de* retorno especificada será então chamada.

A linha 22 do trecho de código da Figura 3.36 mostra a inicialização do *middleware* NITE com o método *NITE.initialize*(). A linha 23 cria um rastreador de mãos e a 24, a detecção do gesto "mãos levantadas" é iniciada. Cabe destacar ainda, na linha 24, o método *startGestureDetection*(*GestureType.HAND_RAISE*). Ele é responsável por inicializar o algoritmo de detecção de mãos, para qualquer mão que aparecer realizando o gesto especificado pelo *Enum GestureType*. A Tabela 3.14 descreve os demais tipos de gestos disponíveis no *GestureType*.

```
16  public class ExampleNITE_HandTracker implements HandTracker.NewFrameListener {
17
18      private static HandTracker handTracker;
19      private static HandTrackerFrameRef handFrameRef;
20
21      public static void main(String[] args) {
22          NiTE.initialize();
23          handTracker = HandTracker.create();
24          handTracker.startGestureDetection(GestureType.HAND_RAISE);
25      }
26
27
28      @Override
29      public void onNewFrame(HandTracker ht) {
30          try {
31              handFrameRef = handTracker.readFrame();
32              for (GestureData gesture : handFrameRef.getGestures()) {
33                  if (gesture.isComplete()) {
34                      handTracker.startHandTracking(gesture.getCurrentPosition());
35                  }
36              }
37          } catch (Exception ex) {
38          }
39      }
```

Figura 3.36. Criação de uma aplicação NiTE e indicação do gesto a ser rastreado

Tabela 3.14. Tipos de gestos disponíveis no Enum GestureType

Gesto	Descrição
GestureType.CLICK	Gesto de click com a mão
GestureType.WAVE	Gesto de tchau
GestureType.HAND_RAISE	Mãos levantadas

Após terem sido iniciados o *middleware*, a detecção de gesto e o rastreamento das mãos, falta verificar se o gesto realmente ocorreu por completo e capturar a posição da mão que o fez. Para isso, é necessário que sejam capturadas todas as referências dos *frames* relacionados com a detecção de mão e obtidos pelo algoritmo de *HandTracker*, através do código da linha 31 da Figura 3.36.

Ainda na mesma figura, as linhas 32, 33 e 34, estão associadas, respectivamente, às seguintes ações: (i) busca dos gestos detectados pelo *handFrameRef* ; (ii) verificação se o gesto foi totalmente finalizado, através do método *isComplete*(); e, (iii) o rastreamento da posição da mão que executou tal gesto, caso tenha sido finalizado.

```
42      public void printCoordinates() {
43          try {
44              for (HandData hand : handFrameRef.getHands()) {
45                  if (hand.isTracking()) {
46                      com.primesense.nite.Point2D<Float> pos =
47                              handTracker.convertHandCoordinatesToDepth(hand.getPosition());
48                      System.out.println(pos.getX().intValue() + "  " + pos.getY().intValue());
49                  }
50              }
51          } catch (Exception ex) {
52          }
53      }
```

Figura 3.37. Código para imprimir as coordenadas da mão em um plano 2D

Até este ponto, já é possível fazer o rastreamento das mãos. Porém, para que seja possível acompanhar sua posição no espaço, foi criado um método *printCoordinates*(), que imprime as coordenadas cartesianas 2D da mão, no console, caso esta mão esteja sendo monitorada (Figura 3.37). Para que o método seja chamado a cada 200 ms, a *thread* atual é alterada conforme ilustra o código da Figura 3.38.

```
55          void run() {
56            synchronized (this) {
57                while (true) {
58                    try {
59                        Thread.sleep(200);
60                        printCoordinates();
61                    } catch (InterruptedException e) {
62                        e.printStackTrace();
63                    }
64                }
65            }
66      }
```

Figura 3.38. Criando a thread para chamar o método *printCordinates*() de tempos em tempos

Para finalizar o exemplo, no método *main* da aplicação, é criada uma instância da classe atual (linha 27), passando sua referência ao *Listener HandTracker* do *middleware* NiTE (linha 28). Em seguida, é iniciada a captura dos eventos gerados (linha 29), conforme ilustra o trecho de código da Figura 3.39.

```
23      public static void main(String[] args) {
24          NiTE.initialize();
25          handTracker = HandTracker.create();
26          handTracker.startGestureDetection(GestureType.HAND_RAISE);
27          final ExampleNITE_HandTracker app = new ExampleNITE_HandTracker();
28          handTracker.addNewFrameListener(app);
29          app.run();
30      }
```

Figura 3.39. Adicionando a classe *ExampleNite_HandTracker* ao *Listener* e iniciando a thread

b) Modificando o HandTracker para exibir imagens nas mãos dos usuários em cena

Com o intuito de chamar a atenção do leitor para algumas possibilidades de interação com o ambiente virtual, serão fornecidos conceitos básicos de exibição e interação com um ícone ou uma imagem. Através desses conceitos e com poucas modificações, o desenvolvedor poderá interagir com esses objetos virtuais através de movimentos e gestos com as mãos.

Para ilustrar uma aplicação mais visual e interativa, será aproveitado o exemplo que mostra os dados do sensor RGB da seção b), adicionando as funcionalidades de

rastreamento de mãos dos usuários presentes em cena (seção 3.5.6) e sobreposição dessas mãos com uma determinada imagem.

As partes destacadas no código ilustrado pela Figura 3.40 foram as modificações realizadas sobre o exemplo *ExampleNI* (seção b)). Porém, a classe auxiliar *ComponentViewer* também recebeu modificações com a adição de mais dois métodos, *setHandTracker*(*HandTracker tracker*) e *setImage*(*String image*), explicados adiante.

```
20  public class ExampleNI2 {
21
22      private static VideoStream videoStream;
23      private static ComponentViewer viewer;
24      private static JFrame frame;
25      private static HandTracker handTracker;
26
27      public static void main(String[] args) {
28          OpenNI.initialize();
29          Device device = Device.open();
30          videoStream = VideoStream.create(device, SensorType.COLOR);
31          VideoMode mode = videoStream.getSensorInfo().getSupportedVideoModes().get(1);
32          videoStream.setVideoMode(mode);
33          viewer = new ComponentViewer();
34          viewer.setStream(videoStream);
35          viewer.setSize(mode.getResolutionX(), mode.getResolutionY());
36          frame = new JFrame("Example NI2");
37          frame.setSize(viewer.getWidth() + 20, viewer.getHeight() + 80);
38          frame.add("Center", viewer);
39          frame.setVisible(true);
40          frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
41          videoStream.start();
42
43          handTracker = HandTracker.create();
44          handTracker.startGestureDetection(GestureType.HAND_RAISE);
45          viewer.setHandTracker(handTracker);
46          viewer.setImage("./TTAir.PNG");
47      }
48  }
```

Figura 3.40. Classe executável do exemplo modificado para exibir imagens sobrepondo as mãos dos usuários

O método *setHandTracker*(*HandTracker tracker*) é responsável, basicamente, por passar a referência do rastreador de mãos para o *ComponentViewer* e adicionar o *listener* de *frames* relacionados com detecção de mãos. Assim, o *ComponentViewer* poderá ter o controle do rastreamento das mãos, capturando suas coordenadas no espaço. A Figura 3.41 ilustra o trecho de código com as modificações necessárias para acrescentar essa funcionalidade.

```
17  public class ComponentViewer extends Component
18          implements VideoStream.NewFrameListener, HandTracker.NewFrameListener {
19
20      int[] mImagePixels;
21      VideoStream mVideoStream;
22      VideoFrameRef mLastFrame;
23      BufferedImage mBufferedImage;
24      HandTracker mTracker;
25      HandTrackerFrameRef handFrameRef;
26      String fileName;
27      BufferedImage image;
28
29      public void setHandTracker(HandTracker tracker) {
30          mTracker = tracker;
31          mTracker.addNewFrameListener(this);
32      }
```

Figura 3.41.Trecho de código ilustrando a adição do método *setHandTracker*()

Para dar continuidade ao exemplo, se faz necessário carregar a imagem passada como parâmetro, do disco para a memória. Assim, com a referência da imagem em memória, será possível implementar o método para desenhá-la em tela. A Figura 3.42 exibe o trecho de código necessário para executar tal funcionalidade. Neste trecho, o método *setImage*(*String image*) passa a referência do caminho da imagem no disco e chama o método *loadImage*() que faz a carga da imagem para memória.

Para exibição das imagens, nas posições onde estão sendo rastreadas as mãos dos usuários em cena, se faz necessário modificar o método *paint*(*graphics g*) do componente visual responsável pela exibição em tela. O trecho de código da Figura 3.43 exemplifica uma maneira de desenhar uma determinada imagem sobrepondo a mão detectada e rastreada pelo middleware NITE.

```
84      public void setImage(String image) {
85          this.fileName = image;
86          this.image = loadImage();
87      }
88
89      private BufferedImage loadImage() {
            BufferedImage image = null;
91          try {
92              image = ImageIO.read(new File(fileName));
93          } catch (Exception erro) {
94              System.out.println("Arquivo não encontrado");
95          }
96          return image;
97      }
```

Figura 3.42. Trecho de código que ilustra os métodos necessários para carregar uma imagem em memória (*setImage(String image) e loadImage()*)

```
71          try {
72              for (HandData hand : handFrameRef.getHands()) {
73                  if (hand.isTracking()) {
74                      com.primesense.nite.Point2D<Float> pos =
75                              mTracker.convertHandCoordinatesToDepth(hand.getPosition());
76                      g.drawImage(this.image, framePosX + pos.getX().intValue() - 20,
77                              framePosY + pos.getY().intValue() - 20, this);
78                  }
79              }
80          } catch (Exception ex) {
81          }
```

Figura 3.43. Bloco de código pertencente ao método *paint(Graphics g)* da classe *ComponentViewer*

Por fim, conforme já explicado no exemplo da Figura 3.36 da seção a), se faz necessário a implementação do método *onNewFrame*(*HandTracker tracker*) para fazer o rastreamento da mão que executou um determinado gesto.

```
137     @Override
        public void onNewFrame(HandTracker tracker) {
139         try {
140             handFrameRef = mTracker.readFrame();
141             for (GestureData gesture : handFrameRef.getGestures()) {
142                 if (gesture.isComplete()) {
143                     mTracker.startHandTracking(gesture.getCurrentPosition());
144                 }
145             }
146         } catch (Exception ex) {
147         }
148     }
```

Figura 3.44. Trecho de código que implementa do método da interface *HandTracker.NewFrameListener*

Após todas as modificações até aqui citadas, já é possível visualizar o exemplo em execução conforme ilustra a Figura 3.45. No exemplo, a logomarca do grupo de pesquisa *Touching The Air* é exibida nas mãos dos usuários em cena, após os mesmos executarem com êxito um determinado gesto monitorado pelo *middleware*.

Figura 3.45.Tela de uma aplicação que reconhece um gesto, e inicia o rastreamento da mão exibindo uma imagem na posição da mão rastreada.

3.6. Conclusão

Este capítulo tratou do desenvolvimento de aplicações voltadas para interação natural utilizando o *framework OpenNI*. A intenção não foi esgotar o assunto, mas levantar questões importantes e divulgar algumas das tecnologias mais recentes ligadas à Interação Natural. Foram apresentados alguns conceitos sobre a Interação Natural, dispositivos envolvidos e sua evolução, assim como uma explicação mais detalhada sobre o *Kinect* e toda a motivação para as pesquisas relacionadas a esta plataforma. As seções finais do capítulo foram dedicadas ao *framework OpenNI* e à sua aplicação em um estudo de caso que mostrou o desenvolvimento, passo a passo, de aplicações em Java para ambientes interativos.

Como foi observado no texto, as interfaces voltadas à interação natural trazem consigo vários desafios e oportunidades para os desenvolvedores de software e produtores de dispositivos. Atualmente, nota-se uma tendência de utilização de plataformas similares ao *Kinect*, não apenas na sua função original de controle de jogos, mas também como um novo periférico de entrada para a comunicação usuário computador.

Outro ponto importante é que o *framework OpenNI* está se tornando a principal aposta dos desenvolvedores de aplicações e comunidades científicas de interação natural. Não só por ser um excelente *framework*, mas por ter se tornado uma plataforma padrão (de facto) de desenvolvimento para estas aplicações. O projeto *OpenNI* abordado no capítulo oferece uma padronização de desenvolvimento, uma abstração maior dos elementos de hardware e uma facilidade de integração de outros componentes de softwares para interpretação dessas informações. Isto que permite tanto reuso das implementações feitas, quanto a leitura conjunta de sensores de mesma natureza aumentando a acuidade da informação.

3.7. Agradecimentos

Os autores agradecem à CAPES pelo suporte financeiro ao primeiro autor através da bolsa do Programa de Demanda Social para o Programa Multiinstitucional de Pós-graduação em Ciência da Computação UFBA/UNIFACS/UEFS (28001010061P1), à PROPEX do Instituto Federal de Sergipe e à *PrimeSense Corporation* pelos equipamentos e apoio.

Referencias

Cordeiro Jr. A. A. A. Modelos e Métodos para Interação Homem-Computador Usando Gestos Manuais. Doutorado, LNCC, 2009.

Valli A. Natural interaction white paper, 2007.

Gregory D. Abowd and Elizabeth D. Mynatt. Charting past, present, and future research in ubiquitous computing. ACM Trans. Comput.-Hum. Interact., 7(1):29– 58, Mar 2000.

BlaGames. Games: Manual de instruções power glove, 2008.

Wiiremote Commander. Wiiremote commander, 2010.

Adafruit Corporation. Open kinect challenge, 2011.

Crawford. How microsoft kinect works, 2010.

Gallud, J.A. and Villanueva, P.G. and Tesoriero, R. and Sebastian, G.and Molina S., and Navarrete A. Gesture-based interaction: Concept map and application scenarios. In Advances in Human-Oriented and Personalized Mechanisms, Technologies and Services (CENTRIC), 2010 Third International Conference on, pages 28–33, 2010.

Heckel, P. Software amigável. Técnicas de projeto de software para uma melhor interface com o usuário. Ed. Campus Ltda, Rio de Janeiro, 1993.

Hewett, Baecker, Card, Carey, Gasen, Mantei, Perlman, and V. Strong. Curricula for Human Computer Interaction. ACM SIGCHI, 1996.

Kinecthacks. Hacks para o kinect, 2010.

Microsoft. Kinect web site, 2010.

Netbeans. Netbeans ide - the smarter and faster way to code. 2012

Nintendo Organization. Wii console controllers, 2009.

OpenKinect Organization. Open kinect roadmap, 2011.

OpenNI Organization. OpenNI corporation web site, 2011.

PrimeSense Organization. Primesense natural interaction, 2011.

Shape Quest. Point cloud data from structured light scanning,2010.

A.N. Rehem Neto, C.A.S. Santos, and M. V. R. Andrade. Interfaces para aplicações de interação natural baseadas na api OpenNI e na plataforma kinect. Tópicos em Banco de Dados, Multimídia e Web / XVII Webmedia / XXVI SBBD. Florianópolis, Brazil: Sociedade Brasileira de Computação, 2011, v. 1, p. 35-60.

A.N. Rehem Neto, C.A.S. Santos, and L. A. Carvalho. Touch the air: Um *framework* orientado a eventos para ambientes interativos. *In* Webmedia, 2013 (*to appear*).

Microsoft Research. Kinect for windows sdk, 2010.

Saffer. Designing Gestual Interfaces. Ot'Reilly, 2009.

Shiratori, T. and Hodgins ,J. K.. Accelerometer-based user interfaces for the control of a physically simulated character. In ACM SIGGRAPH Asia 2008 papers, SIGGRAPH Asia '08, p.123:1–123:9, New York, NY, USA, 2008. ACM.

Shiratori, T and Hodgins, J. K.. Accelerometer-based user interfaces for the control of a physically simulated character. ACM Trans. Graph., 27(5):123:1– 123:9, December 2008.

Nakra T., Ivanov Y., Smaragdis P., and Ault C. The usb virtual maestro: an interactive conducting system. NIME2009, pages 250–255, 2009.

Toyama, K. Look, ma - no hands! - hands-free cursor control with real-time 3D face tracking, 1998.

Twiky, K.. Using Nintendo wiimote with kyma, 2010.

Weiser, M, and Brown, J. S.. Designing calm technology. Powergrid Journal, 1, 1996.

Capítulo

4

Desenvolvimento para Dispositivos Móveis usando Tecnologias Web com Ênfase em Jogos

André Santanchè, Renoir Boulanger, Gabriela Viana, Ricardo Panaggio, Bruno Melo, Hugo Aboud

Abstract

Games are typically challenging applications to develop and they are omnipresent in the mobile platforms. Besides the hardware limitations and heterogeneity of mobiles, we are experiencing a "software platform epoch". Platforms like iOS, Android and Windows Phone offer to developers complete hermetic environments, which come with compatibility side effects, constraining applications to be reimplemented in order to address each platform. One of the most promising approaches to face this platform battle is based on the combination of Web technologies – HTML5, CSS3 and JavaScript – to produce platform-independent applications for mobiles. The question here is how to achieve the balance between independence and performance, required by games. This mini-course gives an overview of this scenario focusing in aspects addressed to game development.

Resumo

Jogos são tipicamente aplicações de desenvolvimento desafiador e são onipresentes nas plataformas de dispositivos móveis. Além das limitações e heterogeneidade do hardware em dispositivos móveis, nós estamos vivenciando uma "época das plataformas de software". Plataformas como iOS, Android e Windows Phone oferecem a seus desenvolvedores ambientes completos e herméticos, que trazem efeitos colaterais de compatibilidade, obrigando que aplicativos sejam reimplementados para atender a cada plataforma específica. Frente a este cenário, este texto apresenta uma abordagem promissora para produzir aplicativos independentes de plataforma para dispositivos móveis, a partir da combinação de tecnologias Web – HTML5, CSS3 e JavaScript. A questão aqui é como alcançar o equilíbrio entre independência e desempenho, exigido pelos jogos. Este minicurso oferece uma panorâmica deste cenário tendo como foco aspectos voltados ao desenvolvimento de jogos.

4.1. Introdução

O tema de desenvolvimento baseado em tecnologias Web sempre levanta alguma suspeita. Há dúvida sobre o desempenho dos aplicativos resultantes e da real flexibilidade na construção de interfaces exigida pelos jogos.

Porém este cenário tem mudado muito. Diversas organizações e pessoas têm se empenhado em transformar a Web em uma plataforma universal para o desenvolvimento de aplicativos, e isto inclui o universo dos dispositivos móveis. Nos últimos anos não apenas as tecnologias Web evoluíram para tornar possível o desenvolvimento de aplicativos completos, como também têm alcançado desempenho para dar suporte ao desenvolvimento de jogos populares.

Em 2013, o número de usuários acessando o Facebook por dispositivos móveis ultrapassou aqueles que acessam pela Web (Kelly, 2013). Este crescente mercado tem se segmentado em diferentes plataformas de software, definidas seja por fabricantes de hardware – e.g., iOS da Apple, BlackBerry OS – seja por companhias que desenvolvem o sistema operacional – e.g., Android da Google, Windows Phone da Microsoft. Cada plataforma define um conjunto especializado de linguagens e APIs para acesso a seus dispositivos, exigindo que desenvolvedores multipliquem seus esforços para atender a mais de uma plataforma.

O desenvolvimento de aplicativos independente de plataforma para dispositivos móveis se apresenta como uma alternativa interessante, que reduz esforços e custos. O princípio básico é desenvolver uma única vez um aplicativo capaz de se adaptar às diferentes plataformas (multiplataforma). Além da heterogeneidade destas plataformas, há desafios na crescente diversidade de dispositivos, com diferenças de resolução, capacidade e funcionalidades.

Segundo Martin Fowler (2012), quando um aplicativo multiplataforma tenta mimetizar o comportamento nativo das aplicações em cada uma das plataformas alvo, usualmente o resultado não agrada ao usuário. O problema é que uma aplicação multiplataforma nunca será capaz de reproduzir com precisão o comportamento de aplicações nativas. Deste modo, uma mimetização aproximada causa uma sensação de que algo não está funcionando tão bem. A solução apresentada por (Fowler, 2012) é portar a plataforma completa. A Web pode ser tratada como uma plataforma a ser portada, sobre a qual se desenvolvem os aplicativos. O usuário já está habituado com o modo de interagir com aplicativos Web e encontrará a mesma abordagem nos diferentes dispositivos.

A combinação entre HTML, CSS e JavaScript atingiu um grande grau de maturidade no desenvolvimento de aplicativos independentes de plataforma na Web. Enquanto a Hypertext Markup Language (HTML) é a linguagem de publicação da Web (http://www.w3.org/html/), o Cascading Style Sheets (CSS) é um mecanismo complementar para adicionar estilo a documentos Web (http://www.w3.org/Style/CSS/). Finalmente, a linguagem de programação JavaScript opera de forma integrada com o HTML e CSS.

Dentre os progressos alcançados no desenvolvimento de aplicativos, está a possibilidade de se usar estas três tecnologias para o desenvolvimento de aplicativos locais autônomos. O JavaScript originalmente desenvolvido para operar como acessório

sobre páginas passa a ser uma linguagem completa, com funcionalidades de acesso a APIs e armazenamento locais. Outro avanço, que é de especial interesse no contexto de jogos, é a possibilidade de rompimento com a interface convencional baseada em documentos, através do uso de linguagens de marcação como o SVG, que introduz um novo horizonte para a concepção de jogos Web.

Esta Web como plataforma de desenvolvimento local é o objeto central de estudo deste texto, que tem enfoque em sua aplicação no desenvolvimento para dispositivos móveis.

A estrutura do texto está organizada da seguinte maneira: na Seção 4.2 são revisados conceitos chave das tecnologias Web em questão, com enfoque no desenvolvimento de aplicativos para dispositivos móveis, bem como são apresentados avanços importantes destas tecnologias que contribuem para este desenvolvimento; na Seção 4.3 são descritas tecnologias para a construção de gráficos e animação que são chave no desenvolvimento de jogos; na Seção 4.4 são tratados alguns fundamentos importantes de JavaScript para o desenvolvimento de aplicativos; na Seção 4.5 são apresentadas estratégias e boas práticas para o uso de marcadores de forma eficiente e reusável; na Seção 4.6 são sistematizados alguns conceitos sobre os Web Components; na Seção 4.7 são descritas as diferentes arquiteturas que dão suporte ao desenvolvimento com tecnologias Web em dispositivos móveis; na Seção 4.8 é dada uma visão geral de alguns fundamentos para o projeto de aplicativos com foco no usuário; na Seção 4.9 serão apresentadas as considerações finais.

4.2. Visão geral de HTML5 e CSS3 para Aplicativos

Esta seção dá uma visão geral de alguns conceitos das tecnologias HTML5 e CSS3 que são importantes para este texto. Por serem temas amplamente conhecidos, com uma vasta documentação disponível, esta seção parte do pressuposto de que o leitor já conhece fundamentos de HTML e CSS. Uma versão mais aprofundada desta seção está disponível em nosso texto anterior (Santanchè, 2013).

4.2.1. HTML5 e CSS3

A especificação da linguagem HTML é mantida pelo World Wide Web Consortium (W3C). Desde a sua última versão 4, o W3C vem se empenhando na especificação da sua versão sucessora, conhecida como HTML5. Entretanto, tal especificação assumiu um novo caráter de "*living standard*" (Berjon et al., 2012) pela participação da organização WHATWG (http://www.whatwg.org). Living se refere ao fato de que uma especificação HTML constantemente atualizada é mantida pelo WHATWG, de modo que inovações possam ser adotadas e experimentadas dinamicamente, na medida em que o padrão evolui, sem compartimentar saltos de evolução em versões. O W3C continua responsável pelo padrão, lançando especificações versionadas.

HTML5, portanto, tornou-se uma "*buzzword*" que incorpora os mais recentes avanços do HTML após a versão 4, com funcionalidades de semântica, acessibilidade, recursos multimídia e de interação, que só eram possíveis com o uso de tecnologias de terceiros (*Flash like*). O HTML5 inaugurou uma nova relação com aplicativos, oferecendo-lhes acesso a funcionalidades que antes exigiam estratégias indiretas, complexas e eventualmente extensões não padronizadas à linguagem. Para dar suporte

as novas funcionalidades como persistência, desenhos 2D, áudio e vídeo, estão sendo desenvolvidas novas APIs.

O CSS permite separar a forma com que os elementos HTML deveriam ser exibidos do seu conteúdo. Esta separação conteúdo-estilo é fundamental no desenvolvimento de aplicações que se adaptam a vários dispositivos. As especificações do CSS são definidas em níveis ao invés de versões, em que cada nível acrescenta funcionalidades ao anterior. O CSS nível 3, CSS3, dividiu o processo de especificação em pequenos módulos que evoluem de forma independente, com base a especificação CSS 2.1. Além disto, adicionou funcionalidades para efeitos de texto, transformações 2D/3D e animações.

4.2.2. Separação Conteúdo – Estilo

Ao se tratar de layout no contexto de dispositivos móveis, uma questão chave é adaptabilidade, dada a diversidade de formatos e resoluções. A Figura 4.1 ilustra o aspecto chave para tratar a questão: a separação de conteúdo e estilo, usando a semântica como ponte.

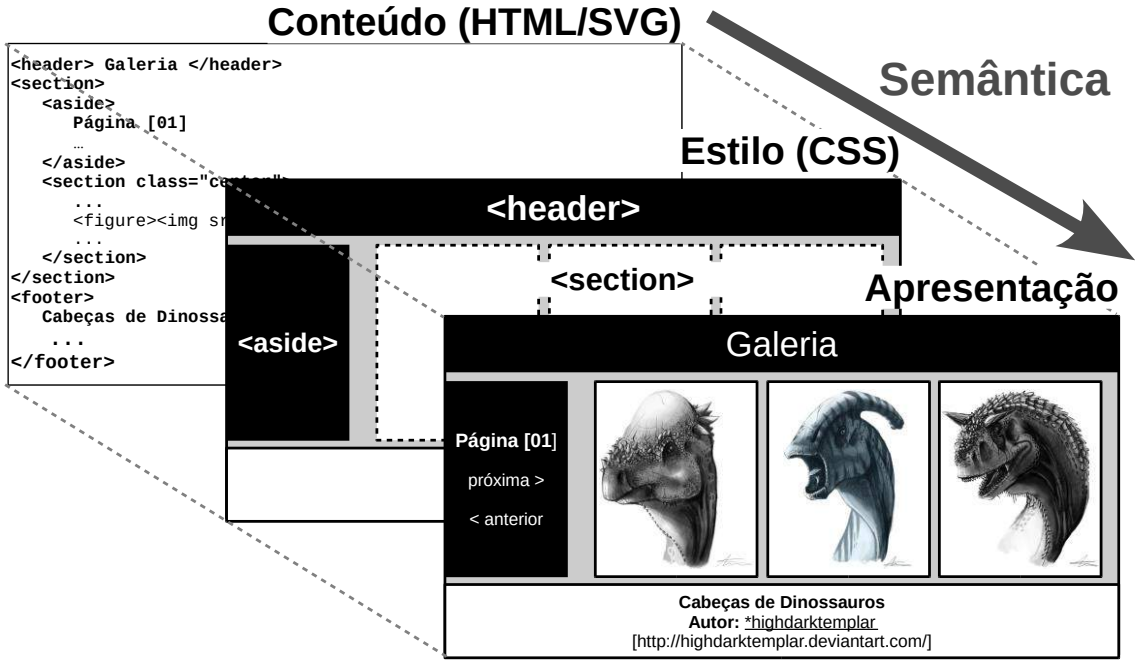


Figura 4.1. Separação Conteúdo-Estilo em uma tela de aplicativo.

Na figura, o Conteúdo representa os elementos da tela inicial de um aplicativo. Eles estão dispostos em categorias, conforme seu papel, que são anotações semânticas sobre os dados. Uma segunda camada de Estilo associa elementos anotados semanticamente a seu layout visual. O Conteúdo associado às diretrizes de apresentação do Estilo produz a Apresentação, ilustrada na terceira camada. Deste modo, o mesmo conteúdo fonte pode ser associado a diferentes diretrizes de apresentação para se adaptar a diferentes contextos.

Em nosso texto anterior (Santanchè, 2013) tratamos com detalhes esta questão, apresentando técnicas e melhores práticas de realizar tal separação e como o HTML5 tem progredido na direção de uma marcação mais semântica.

4.2.3. Responsividade e Adaptabilidade

Em 2010, Ethan Marcotte (2010) chama atenção para uma disciplina emergente na arquitetura chamada *Responsive Architecture*. A questão central é projetar ambientes que respondem às mudanças e à presença ou ação de pessoas, explorando sensores, robótica e novos materiais para criar paredes flexíveis que se ajustam; divisórias de smart glass que se tornam opacas quando pessoas iniciam uma reunião; ajustes de luz e temperatura que se adequam à presença e volume de pessoas (Marcotte, 2010). O termo Responsive Design deriva desta inspiração, em que Marcotte propõe que "Ao invés de customizar projetos desconexos para cada e sempre crescente número de dispositivos web, nós podemos tratá-los como facetas da mesma experiência"[1] (Marcotte, 2010). Algumas novidades introduzidas pelo HTML5 e CSS3 – a exemplo da media query e dos layouts flexíveis – contribuem na construção de interfaces Web "responsivas". Mas antes disto, deve-se considerar quais as características que se esperam neste tipo de projeto adaptável/responsivo.

Um exemplo bastante característico desta estratégia está ilustrado na Figura 4.2. Trata-se do Table:Reflow do framework jQuery (http://view.jquerymobile.com/1.3.2/dist/demos/widgets/table-reflow/). Este tipo de componente é capaz de ir adaptando o formato da tabela de acordo com o layout visual e proporções do dispositivo em que ela é apresentada. No limite, em telas muito estreitas (Figura 4.2 direita), o table reflow transforma a tabela em uma lista de "cartões". Com isto o usuário restringe o seu "scroll" a uma única dimensão, facilitando a acesso a dados em dispositivos com capacidade de apresentação limitada.

Rank	Movie Title	Year	Rating	Reviews
1	Citizen Kane	1941	100%	74
2	Casablanca	1942	97%	64
3	The Godfather	1972	97%	87
4	Gone with the Wind	1939	96%	87
5	Lawrence of Arabia	1962	94%	87
6	Dr. Strangelove Or How I Learned to Stop Worrying and Love the Bomb	1964	92%	74
7	The Graduate	1967	91%	122
8	The Wizard of Oz	1939	90%	72
9	Singin' in the Rain	1952	89%	85
10	Inception	2010	84%	78

Rank	1
Movie Title	Citizen Kane
Year	1941
Rating	100%
Reviews	74
Rank	2
Movie Title	Casablanca
Year	1942
Rating	97%
Reviews	64

Figura 4.2. Exemplo de Responsive Design no Table Reflow.

Em nosso texto anterior (Santanchè, 2013) tratamos em detalhes algumas técnicas que permitem a criação de design adaptável/responsivo, inclusive revisando

[1] Tradução livre (Marcotte, 2010)

alguns conceitos básicos de CSS no projeto de layout. Dentro as inovações do CSS3, são apresentados os conceitos de *flexible box* ou flexbox (Atkins Jr. et al., 2012) – que consiste em uma técnica baseada em um gerenciamento automatizado de layout – e *media query* (Lie et al., 2012) que permite inspecionar o contexto e estabelecer escolhas condicionais de estilo.

4.2.4. Semântica nos documentos Web

Desde o começo da Web, páginas apresentadas ao usuário mantiveram suas raízes no HTML, muito embora a Web tenha evoluído para padrões mais genéricos, como a linguagem XML. A Extensible Markup Language – XML (Bray et al., 2008) foi criada com o intuito de permitir a uma comunidade definir seus próprios marcadores, as regras de composição de documentos e, consequentemente, o modo como os segmentos marcados serão interpretados. Isto a classifica como uma metalinguagem (linguagem para a definição de linguagens), ao contrário do HTML que possui um conjunto predefinido de marcadores.

Iniciativas como Microformats (Khare, 2006), RDFa (Adida & Birbeck, 2008) (Adida et al., 2011) e Microdata (Hickson, 2011) perceberam a importância de permitir que usuários definam seus próprios vocabulários (como acontece no XML), de modo que possam criar anotações em páginas HTML, sem romper com a abordagem nativa de representação do conteúdo. As anotações cumprem o papel de metadados e permitem uma interpretação mais rica do conteúdo por máquinas e humanos. Neste sentido, elas enriquecem semanticamente este conteúdo.

Em nosso texto anterior (Santanchè, 2013) apresentamos a iniciativa dos Microformats que foi pioneira neste contexto. Neste texto, apresentaremos o RDFa. Enquanto o Microformats define uma estratégia para incorporar vocabulários em documentos HTML, a interpretação da semântica dos mesmos é delegada para os programas, já que ela não pode ser descrita de forma explícita usando o próprio Microformats. Neste sentido, ele está bastante alinhado com a possibilidade de construção de vocabulários XML.

O RDFa, por outro lado, está alinhado com a camada de semântica explícita da Web Semântica, o *Resource Description Framework* – RDF. O RDF define um modelo e uma linguagem para a representação homogênea de descrições associadas a recursos, que podem ser identificados por meio da Web (Manola, 2004). Ele á capaz de tornar a semântica das descrições explícita através de uma rede de relações semânticas entre conceitos, identificados através das URIs. Por esta razão, o RDF é a base, por exemplo, para a construção de ontologias.

O RDFa torna possível mesclar descrições RDF em documentos HTML. A descrição detalhada da Web Semântica e do RDF vão além do escopo deste material. Sumarizaremos aqui alguns conceitos importantes para a autoria na Web, dando uma visão geral de como implementá-los.

Considere que queremos representar em RDF a seguinte sentença: “o céu tem cor azul”. O RDF representa tais sentenças em um modelo baseado em triplas, que explicita a semântica da descrição. Cada tripla define o recurso que está sendo descrito que está sendo descrito – o “céu” neste exemplo – uma propriedade – a “cor” neste exemplo – e o valor para a propriedade – “azul” neste exemplo. Isto forma a tripla

(recurso, propriedade, valor) com os valores (“céu”, “cor”, “azul”). Os elementos da tripla, aqui apresentados de forma abstrata entre aspas, serão na maioria dos casos entidades representadas de forma única por URIs e dentre elas conceitos em ontologias.

O exemplo a seguir mostra uma reinterpretação das receitas sob a ótica do RDFa. O atributo especial `property` serve para mediar a construção de triplas. Ele associa um recurso, definido pelo bloco em que está inserido, com uma propriedade declarada em `property` e um valor, que é o valor do elemento que tem o atributo `property`. Também é usado um atributo do tipo `typeof`, que define recursos que são instâncias de classes.

```
<div xmlns:v="http://rdf.data-vocabulary.org/#"
     typeof="v:Recipe">
...
<p property="v:yield">6 to 8 servings</p>

...
<ul rel="v:ingredient">
  <li typeof="v:Ingredient">
    <span property="v:amount">1/2 pound</span> of
    <span property="v:name">elbow macaroni</span></li>
...
</ul>
...
<p property="v:instructions">
<p>Preheat oven to 350 degrees F.</p>
...
</div>
...
</div>
```

Se por um lado um documento HTML pode ser visto como a base para construção de aplicativos, tal como estamos tratando neste documento, através do RDFa ele se torna base para a construção de descrições semânticas aptas a serem extraídas e recuperadas na forma de query.

O query sobre RDF pode ser feito através do SPARQL (W3C SPARQL Working Group, 2013), que pode ser definido de forma simplificada como uma linguagem de *query* no estilo SQL, mas voltado para o modelo RDF de triplas e rede de associações.

4.3. Gráficos vetoriais, SVG, Canvas e WebGL

Tecnologias Web têm permitido ir além do clássico modelo de construção visual baseada em documentos, para a exploração de um espaço vetorial, usado não apenas para a inserção de imagens dentro de documentos, mas para o projeto da interface completa. Isso é especialmente importante no desenvolvimento de jogos e, por esta razão, damos especial atenção ao tema nesta seção.

Nos primórdios do desenvolvimento Web, o layout das páginas era composto basicamente por textos e imagens dispostos linearmente. Eventualmente eram usados frames ou tabelas delimitando e organizando seções dentro da página. Tal paradigma foi se tornando obsoleto com o surgimento do CSS e a consequente possibilidade de se posicionar blocos de conteúdo associado a estilo, definidos pelas tags `<DIV>`.

Porém, o grande avanço no sentido de dinamização de conteúdo veio com a possibilidade de alterar propriedades de estilo (CSS) dos elementos após o carregamento da página, através de Javascript. Frameworks como o jQuery (http://jquery.com) tornaram a Web uma experiência mais interativa, apesar de ainda limitada pelo conceito de blocos de conteúdo associados aos DIVs, que exigiam estratégias indiretas para a construção de primitivas gráficas, não previstas originalmente no HTML e que afetavam o bom desempenho em navegadores.

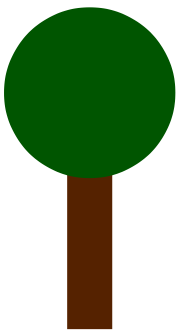
Para projetos de interfaces mais arrojadas, que façam uso de formas mais complexas, as DIVs podem dificultar o desenvolvimento, dada a sua natureza quadrada. Pode-se utilizar estilos em CSS3 e imagens transparentes para talhar forma e efeitos desejados, porém, muita memória e processamento são utilizados para a apresentação.

Além disso, se o projeto exigir muitos elementos em uma mesma cena, a utilização de DIVs poderá apresentar desempenho insatisfatório na experiência do usuário. Por este motivo, existe uma tendência de não se utilizar as DIVs como solução, em projetos preocupados com questões de desempenho e fluidez da experiência de usuário.

4.3.1. SVG

Um avanço neste sentido foi o suporte nativo dos navegadores ao formato SVG – Scalable Vector Graphics (Dahlström et al., 2011). O SVG nasceu como um formato XML para a representação de imagens vetoriais. Enquanto formatos raster – tais como GIF, JPG e PNG – representam a imagem como uma matriz de pixels, formatos vetoriais representam a imagem como uma coleção de objetos geométricos, em que são registradas propriedades de posicionamento, dimensões e características de apresentação, como cor e transparência. A imagem é reconstruída em cada apresentação. Imagens vetoriais se adaptam a diferentes configurações de apresentação, sendo capazes de explorar os limites de resolução da área de apresentação.

Ao contrário de outros padrões vetoriais, como o WMF e EPS, o SVG representa a imagem em formato texto baseado em XML. Por esta razão ele tem se tornado um padrão para apresentar conteúdo vetorial na Web. É possível inserir diretamente uma imagem SVG dentro de um documento HTML. A Figura 4.3 ilustra um segmento SVG tal como inserido no corpo (`<body>`) de uma documento HTML.



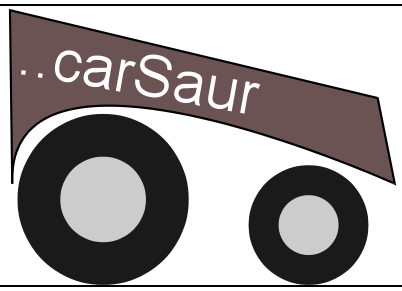
```
<svg id="desenho" version="1.1"
     xmlns="http://www.w3.org/2000/svg"
     width="205" height="370">

    <rect style="fill:#552200"
          x="77" y="179" width="50" height="190" />

    <circle style="fill:#005500"
            cx="102" cy="106" r="95" />

</svg>
```

Figura 4.3. Imagem de uma árvore descrita em SVG.

O tag `<svg>` delimita e dimensiona (através dos atributos `width` e `height`) a área de desenho. Dentro desta área foram usadas as seguintes primitivas de desenho:

Primitiva	Descrição	Atributos	
`<rect>`	Desenha um retângulo.	`style`	Estilo de apresentação. Neste caso define a cor de preenchimento.
		`x, y`	Coordenadas do canto esquerdo superior.
		`width, height`	Altura e largura do retângulo.
`<circle>`	Desenha um círculo.	`style`	Estilo de apresentação. Neste caso define a cor de preenchimento.
		`cx, cy`	Coordenadas do centro do círculo.
		`r`	Raio do círculo.

O SVG também pode ser usado para a criação de curvas mais complexas, como o carro ilustrado na Figura 4.4. Neste caso, foi usado um recurso de construção de polígonos baseado na descrição do contorno (`<path>`). Além disto, há um texto rotacionado.




```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg"
     width="181" height="130">

  <circle style="fill:#1a1a1a" cx="44" cy="90" r="40" />
  <circle style="fill:#cccccc" cx="44" cy="90" r="20" />

  <circle style="fill:#1a1a1a" cx="140" cy="102" r="28" />
  <circle style="fill:#cccccc" cx="140" cy="102" r="14" />

  <path style="fill:#6c5353; stroke:#000000; stroke-width:1px"
        d="M 180.322,82.637687 172.30769,42.566127 0.50088787,1.4927774
           1.5026779,82.637677 c -2.50447,-82.14667965
           178.8193221,1e-5 178.8193221,1e-5 z" />

  <text style="fill:white; font-size:28px; font-family:Arial"
        x="9" y="30"
        transform="rotate(10)">
    ..carSaur
  </text>


</svg>
```

Figura 4.4. Imagem de um carro descrito em SVG.

Para a construção deste carro foram usadas as seguintes primitivas adicionais:

Primitiva	Descrição	Atributos	
`<path>`	Descreve um trajeto que usualmente será usado para a definição de contornos de polígonos.	`style`	Estilo de apresentação. Neste caso define a cor de preenchimento e do contorno e a espessura do contorno.
		`d`	Sequência de contorno formada por letras que representam primitivas de descrição do contorno e coordenadas.
`<text>`	Insere um texto.	`style`	Estilo de apresentação. Neste caso define a cor e fonte da letra.
		`x, y`	Coordenadas do esquerdo inferior do texto.

O atributo `transform`, associado neste exemplo à primitiva `<text>` permite a aplicação de transformações geométricas sobre as primitivas. Tal atributo pode ser aplicado na maioria das primitivas e permite a realização de transformações de: rotação (`rotate`), translação (`translate`), mudança de escala (`scale`) etc.

O SVG não apenas pode ser inserido em páginas HTML, mas pode assumir o lugar do HTML na definição da estrutura do documento. Ou seja, é possível construir um aplicativo Web completamente a partir de uma descrição SVG. Isto permite romper o layout tipicamente horizontal e orientado a documentos do HTML, o que é essencial no desenvolvimento de jogos. O SVG interage com os demais atores de um documento Web do mesmo modo que o HTML. É possível se associar folhas de estilo ao SVG, bem como scripts em JavaScript.

O exemplo a seguir mostra uma redefinição do exemplo anterior, com o uso de uma folha de estilos associada ao SVG. Do lado esquerdo, está a descrição SVG e do lado direito a folha de estilo usada pela descrição.

```
<svg version="1.1"
xmlns="http://www.w3.org/2000/svg"
     width="181" height="130">
  <circle class="tyre" cx="44" cy="90" r="40" />
  <circle class="rim" cx="44" cy="90" r="20" />

  <circle class="tyre" cx="140" cy="102" r="28"/>
  <circle class="rim" cx="140" cy="102" r="14" />

  <path class="frame"
        d="M 180.322,82.637687 172.30769,42.566127
          0.50088787,1.4927774 1.5026779,82.637677
          c -2.50447,-82.14667965 178.8193221,1e-5
          178.8193221,1e-5 z" />

  <text class="nameStyle" x="9" y="30"
        transform="rotate(10)">
    ..carSaur
  </text>

</svg>
```

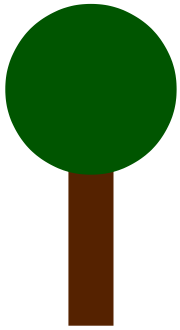
```
<style type="text/css">
  .tyre {
    fill: #1a1a1a;
  }
  .rim {
    fill: #cccccc;
  }
  .frame {
    fill: #6c5353;
    stroke: #000000;
    stroke-width: 1px;
  }
  .nameStyle {
    fill: white;
    font-size: 28px;
    font-family: Arial;
  }

</style>
```

4.3.2. Canvas

A presença de um renderizador vetorial nativo nos navegadores – trazido pelo SVG – permitiu a concepção de um mecanismo no HTML5 que o utilizasse de forma dinâmica para gerar conteúdo: o canvas. Esta nova abordagem substitui o mecanismo de blocos de conteúdo, introduzindo a possibilidade de construções mais complexas e dinâmicas.

O canvas surgiu como uma ferramenta da Apple em seu WebKit e passou a ser adotado pela W3C como uma forma padrão de criação de imagens diretamente em uma área do navegador, pelo uso de primitivas de desenho. O canvas se comporta como uma tela de pintura, em que programas em JavaScript podem plotar coisas. Ele é capaz de traçar desde simples quadrados a cenários tridimensionais complexos.

A Figura 4.5 apresenta a mesma árvore da figura anterior descrita através do uso do canvas e comandos JavaScript. No corpo do documento HTML a área do canvas é delimitada, dimensionada e nomeada pelo tag <canvas>.



```
function desenhar() {
   var desenho = document.querySelector("#desenho");

   if (desenho.getContext) {
      var contexto = desenho.getContext("2d");

      contexto.fillStyle = "#552200";
      contexto.fillRect(77, 179, 50, 190);

      contexto.fillStyle = "#005500";
      contexto.arc(102, 106, 95, 0, 2*Math.PI);
      contexto.fill();
   }
}
```

```
<canvas id="desenho" width="205" height="370">
  Área de desenho
</canvas>
```

Figura 4.5. Imagem de uma árvore construída em um canvas.

No módulo JavaScript que desenha a árvore, o objeto canvas é recuperado através de um querySelector e é recuperado um objeto de contexto bidimensional (“2d”) deste canvas. Tal objeto define um conjunto de métodos que são primitivas para desenho no canvas. Foram usados os seguintes métodos:

Atributo	Descrição		
`fillStyle`	Define o estilo de preenchimento das figuras que serão traçadas subsequentemente. No exemplo, a cor de preenchimento.		
Método	**Descrição**	**Parâmetros**	
`fillRect`	Desenha um retângulo preenchido.	`x, y`	Coordenadas do canto esquerdo superior.
		`width, height`	Altura e largura do retângulo.
`arc`	Define uma forma geométrica a ser usada posteriormente para desenho.	`cx, cy`	Coordenadas do centro do arco.
		`r`	Raio do arco.
		`começa, termina`	Indica o ângulo em que o desenho do arco começa e termina respectivamente.
`fill`	Preenche uma forma geométrica. No exemplo, o arco.		

Há diferenças fundamentais em traçar imagens em SVG ou desenhando no canvas. Uma imagem SVG é preservada vetorialmente na árvore DOM de objetos que compõem o documento. Ele será adaptado e redesenhado cada vez que a área de apresentação sofrer mudanças. O desenho no canvas funciona como riscar um quadro. Apesar de terem sido usados métodos que se comportam como as primitivas de desenho do SVG – ou seja, `fillRect()` corresponde ao `<rect>`; `arc()` mais `fill()` correspondem ao `<circle>` – uma vez que a forma é "riscada" no canvas, ela se torna uma conjunto de pixels e perde-se a sua origem de objeto vetorial.

Mesmo com a introdução do canvas, há aspectos práticos que ainda herdam características clássicas de apresentação Web, que em sua maioria não se modificam com grande dinâmica – e.g., animações. Muito se discute, no entanto, acerca do seu desempenho: trata-se de uma biblioteca gráfica "*immediate mode*", ou seja, o canvas é atualizado conforme as chamadas às funções de desenho ocorrem, não havendo um buffer que esconde do usuário as etapas do render (*double-buffering*). Devido a este problema, em imagens de alta complexidade o usuário pode ver o '*flick'* da tela sendo atualizada. Há várias formas de contornar este problema – como, por exemplo, o uso de display lists (canvas pré-renderizados que são inseridos na cena em runtime) e buffers de hardware – mas, em termos práticos, o método ainda se mostra ineficaz para aplicações mais complexas.

Visando melhorar desempenho de aplicações 3D na Web, o WebGL (http://webgl.org) surgiu como um aprimoramento do canvas. Para isto utilizou-se da recente comunicação com hardware que os navegadores passaram a oferecer. A comunicação mais direta com a GPU permitiu a renderização de buffers 3D em tempo satisfatório, em comparação com o canvas, utilizando-se do conceito de matrizes de

projeção e model-view, presentes no OpenGL. A Graphics Processing Unit (GPU) é uma unidade de processamento especializada no tratamento dos gráficos que são apresentados no display. Usualmente, ela fica situada na placa gráfica do computador.

A comunicação com a GPU permitiu, além do ganho de desempenho, a implementação de shaders (scripts que rodam na GPU) específicos para o aplicativo que está em desenvolvimento e a possibilidade de trabalhar com muito mais flexibilidade em cores e texturas, bem como na implementação mais eficaz de interfaces 3D. Isto representa um grande avanço no desenvolvimento de jogos Web/mobile, uma vez que apresenta uma forma robusta e flexível de renderização.

Já existem centenas de exemplos didáticos da capacidade das novas formas de renderização do HTML5 espalhados pela Web, utilizando-se de técnicas de canvas e WebGL, com e sem o uso de frameworks. Muitas vezes misturam-se técnicas no desenvolvimento, como por exemplo jogos cuja interface 3D é desenvolvida em WebGL e a HUD (Heads-up Display, informações que aparecem em posição estática sobre o jogo) através de DIVs. No entanto, por ser uma tecnologia nova, pouco se explorou comercialmente de suas possibilidades.

4.3.3. Animação em SVG

Há duas estratégias fundamentais para se produzir animações em SVG. A primeira envolve o uso de primitivas do próprio SVG e a segunda é feita através de programação e requer interação com o JavaScript.

Para a primeira estratégia o SVG define primitivas de animação (Dahlström, 2011), dentre as quais destacamos:

Primitiva	Descrição	Atributos	
`<animate>`	Varia um valor de atributo dentro de um período de tempo.	`attributeName`	Nome do atributo afetado.
		`from`	Valor de origem da variação.
		`to`	Valor de destino da variação.
`<animateTransform>`	Realiza uma transformação geométrica dentro de um período de tempo.	`type`	Tipo de transformação: e.g., `rotate`, `translate`.
		`from`	Valor inicial da transformação.
		`to`	Valor final da transformação.
`<animateMotion>`	Move-se por um trajeto em um período de tempo.	`path`	Trajeto a ser seguido. É descrito da mesma maneira que uma linha poligonal.

Para as três primitivas de animação valem os atributos:

Atributo	Descrição
`begin`	Instante no tempo em que a ação inicia.
`dur`	Duração da ação.
`fill`	Recebe o valor `freeze` para indicar que o objeto se manterá na posição final após a animação e recebe `remove` caso o objeto deva retornar à posição original depois da animação.

O seguinte trecho realiza uma animação aumentando o raio (atributo `r`) da jante do carro de `0` para `20` em seis segundos. Note que a primitiva de animação é colocada dentro do elemento animado:

```
<circle style="fill:#cccccc" cx="44" cy="90" r="0">
  <animate attributeName="r" attributeType="XML"
           begin="0s" dur="6s" fill="freeze" from="0" to="20" />

</circle>
```

O seguinte trecho realiza uma animação rotacionando o nome do carro de `-90` até `10` graus em seis segundos:

```
<text style="fill:white; font-size:28px; font-family:Arial"
      x="9" y="30">
  ..carSaur
  <animateTransform attributeName="transform" attributeType="XML"
       type="rotate" from="-90" to="10"
       begin="0s" dur="6s" fill="freeze" />

</text>
```

A animação por trajeto exige a definição de um caminho (`path`) na forma de linha poligonal. No exemplo a seguir a descrição geométrica do carro foi agregada dentro de um elemento de agrupamento `<g>`. O `<animatedMotion>` foi aplicado ao `<g>` de modo que todo o carro participe da animação.

```
<g>
  <circle … />
  <circle … />
  <circle … />
  <circle … />
  <path … />
  <text … />

  <animateMotion path="m 2.8571401,475.21933 c 162.8571499,-105.71428
282.8043099,34.17323 388.5714299,0 105.76713,-34.17323 127.12446,-162.77867
225.71429,-162.85714 98.58983,-0.0785 254.28571,182.85714
254.28571,182.85714"
                 begin="0s" dur="10s" fill="freeze" />
</g>
```

O resultado das animações acima está ilustrado na Figura 4.6, que além disto inclui a plotagem da linha poligonal equivalente ao trajeto a ser seguido pelo carro. É importante ressaltar que o navegador aqui carregou diretamente o arquivo SVG.

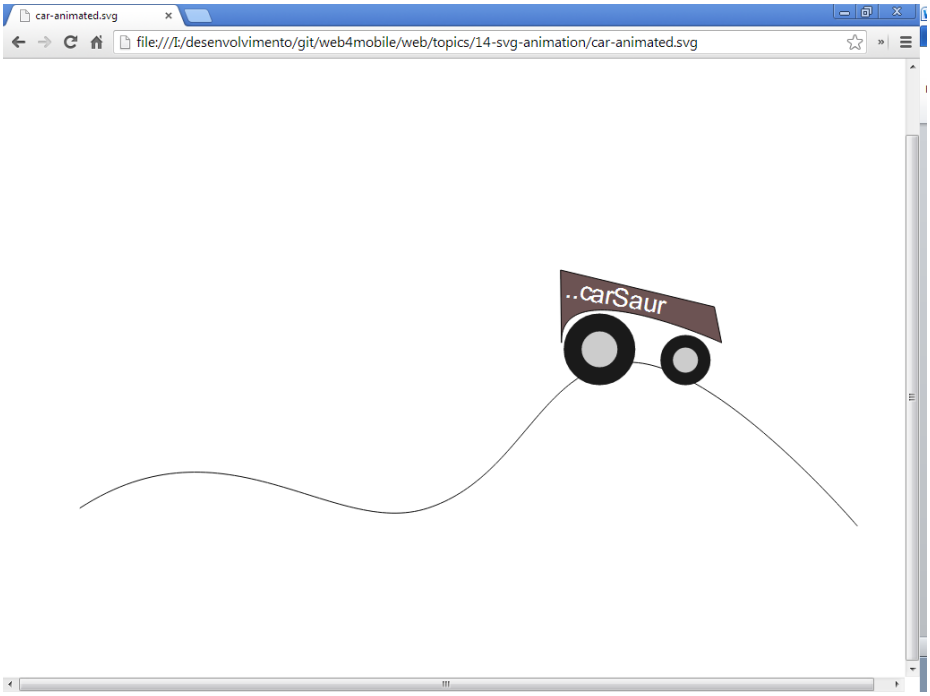


Figura 4.6. Carro executando uma animação SVG.

4.3.4. Ferramentas para SVG

Por ser o SVG um formato de gráficos vetoriais há ferramentas para a construção de imagens e paths, que podem ser exploradas no desenvolvimento de aplicativos. Dentre elas, o Inkscape (http://inkscape.org/) se destaca por editar nativamente SVG. Tais ferramentas se tornam essenciais, principalmente para a construção de curvas complexas usadas tanto nas ilustrações quanto nas animações. O ponto negativo é que nem sempre o Inkscape produzirá uma representação simples e limpa da imagem.

A Figura 4.7 ilustra um trajeto o trajeto aplicado na animação do carro sendo modelado no Inkscape pelo uso de curvas de Bezier. Após a gravação do arquivo SVG, o trajeto, representado no atributo path, foi copiado para o elemento de animação `<animateMotion>`, de modo que o carro se deslocasse por sobre o trajeto.

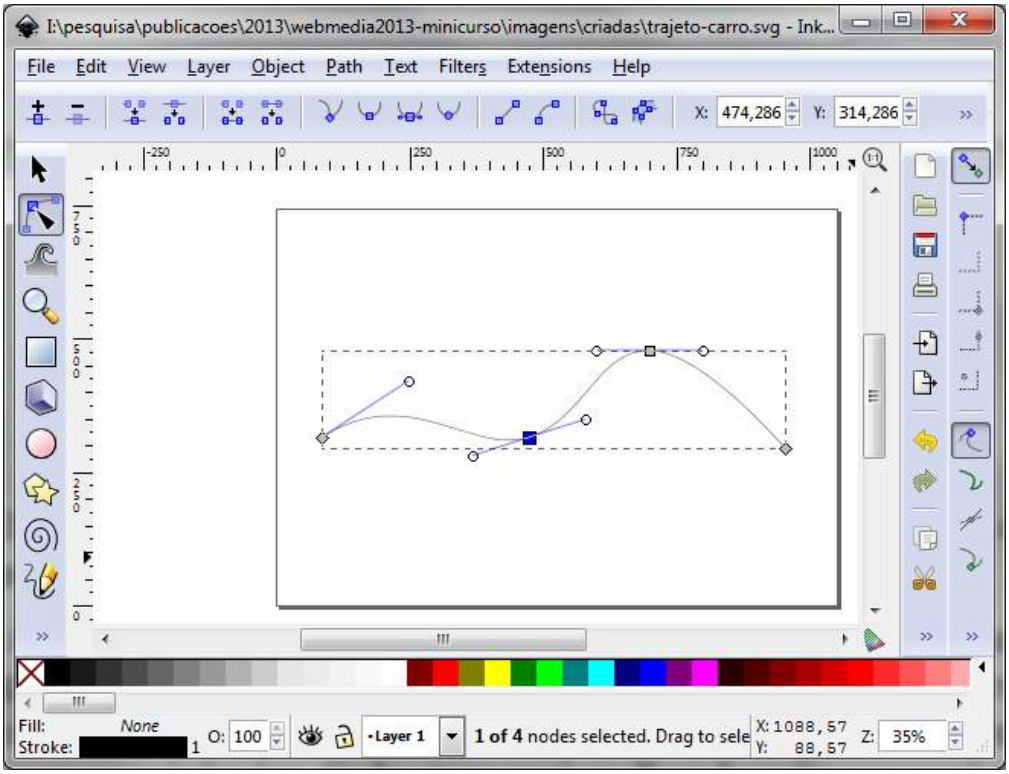


Figura 4.7. Trajeto (path) sendo modelado no Inkscape.

4.4. JavaScript para Aplicativos e DOM

JavaScript é a principal linguagem para programação em navegadores Web. Apesar de ter sido originalmente projetada para o desenvolvimento de pequenos scripts – que permitem definir o comportamento dos elementos do HTML, manipulando suas características, alterando as propriedades e valores definidos no CSS – a linguagem se tornou a melhor opção para o desenvolvimento de aplicações independentes de plataforma sobre navegadores Web. Com o HTML5 e as novas APIs, surgem mais possibilidades de controle dos elementos criados no código.

A linguagem foi inventada por Brendan Eich na Netscape e implementada no navegador Netscape 2.0. Em seguida, foi incorporada no navegador Internet Explorer 3.0 da Microsoft com o nome de JScript (Ecma, 2011). Com o intuito de padronizar a linguagem JavaScript, que passou a ser adotada e modificada nos mais diversos navegadores, a organização Ecma International lançou o padrão ECMAScript, baseado no JavaScript e JScript (Ecma, 2011). Os diversos navegadores têm progressivamente suportado o padrão ECMAScript.

O W3C constituiu o Web Applications Working Group – WebApps WG (http://www.w3.org/2008/webapps/) com o intuito de desenvolver especificações para dar suporte ao desenvolvimento de Aplicativos Web (WebApps). As especificações deste grupo serão de especial interesse nesta seção. Dentre elas, trataremos do DOM, Web Components, Web Storage e padronização das Widgets.

4.4.1. DOM - Document Object Model

O paradigma orientado a objetos tem se mostrado uma excelente ponte entre os universos de documentos (HTML/CSS/SVG) e programas de computador (JavaScript). Árvores de objetos usadas para modelar estruturas de dados neste paradigma se casam de forma elegante com árvores de conteúdo, que são representações típicas de documentos. O padrão de interfaces entre aplicativos e documentos chamado DOM – *Document Object Model* – proposto pelo W3C (Kesteren et al., 2012) tira vantagem desta similaridade.

A interface DOM é certamente um dos padrões mais importantes para se estabelecer a ponte entre o HTML/CSS/SVG e o JavaScript. No texto anterior (Santanchè, 2013) mostramos como o DOM interage com um documento HTML. O DOM não está restrito a documentos HTML/CSS, ele pode ser igualmente usado para documentos XML. Ele é fundamental na manipulação de documentos XML que o cliente intercambia com o servidor.

Mostraremos aqui como esta interação ocorre com o SVG do carro apresentado na Seção 4.1, cuja árvore DOM é ilustrada na Figura 4.8. A representação SVG fica completamente subordinada ao nó `<svg>`, situado na raiz da árvore. Como mostra a figura, os atributos `width` e `height` determinam a extensão horizontal e vertical da ilustração. Abaixo do nó raiz, aparecem: quatro nós `<circle>` que representam as duas rodas; um nó `<path>` traçando o polígono que constitui o chassi do carro; um no `<text>` contendo o nome do veículo.

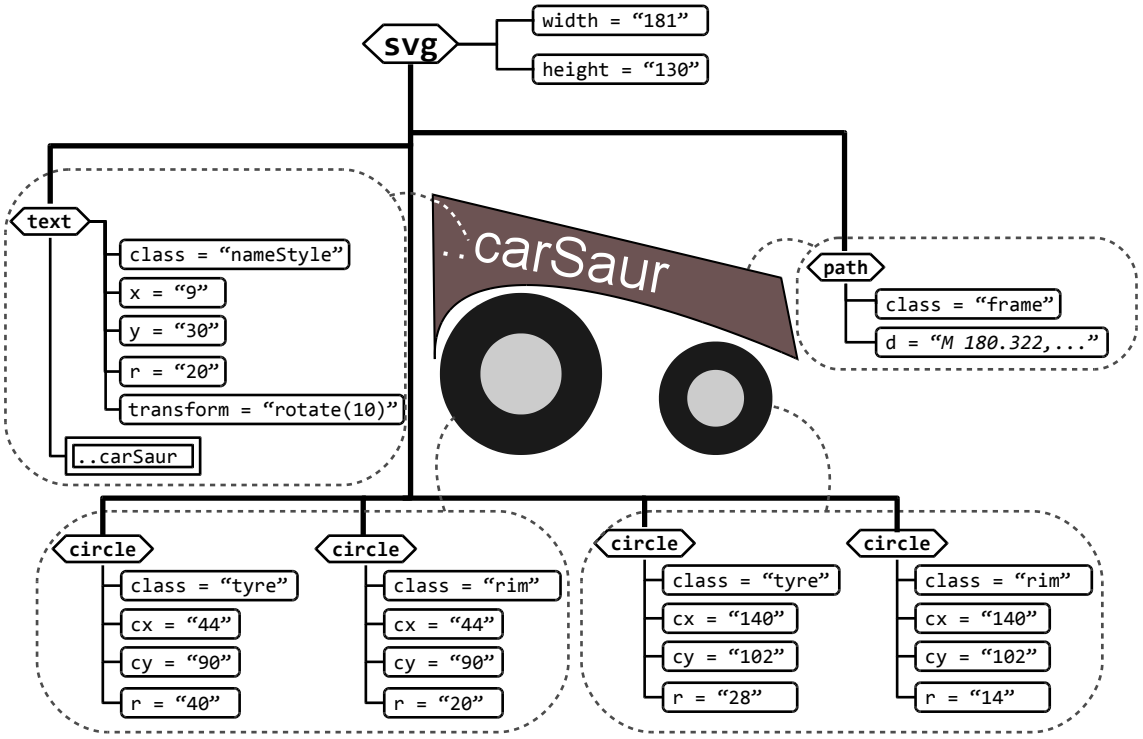


Figura 4.8. Árvore DOM do carro em SVG.

Como está ilustrado na Figura 4.9, o DOM pode ser entendido como uma lente entre o documento e o aplicativo JavaScript. Ele permite que este aplicativo visualize e

manipule tanto o conteúdo SVG quanto o estilo CSS como árvores de objetos em JavaScript.

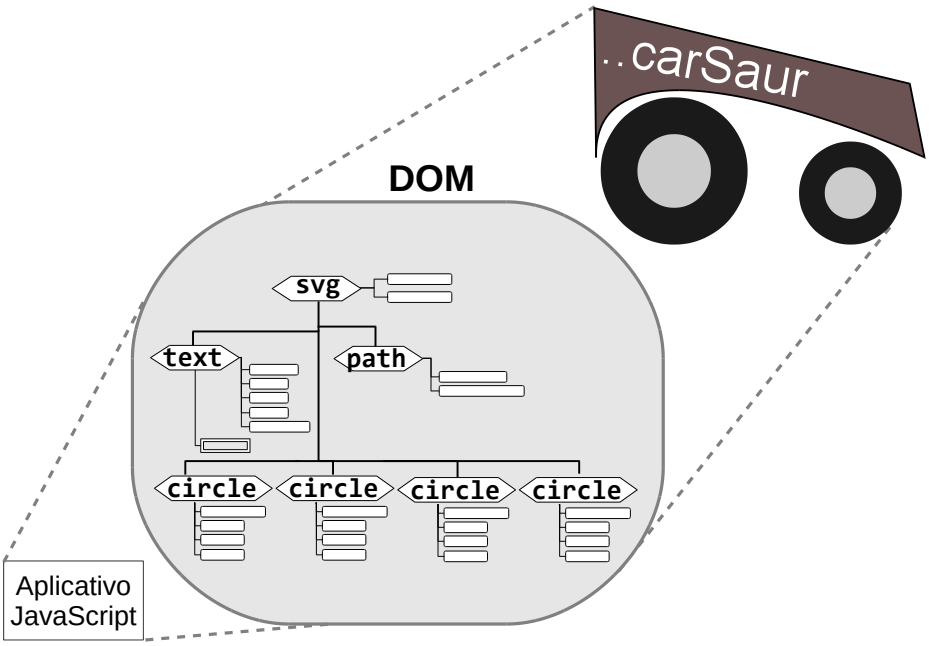


Figura 4.9. Perspectiva DOM sobre um desenho em SVG.

A especificação DOM é independente de plataforma. Por este motivo, ela não possui qualquer descrição de como seus métodos serão implementados, se limitando à interface. A implementação é delegada para soluções sobre plataformas e linguagens específicas.

Cada tipo de nó de um documento HTML ou XML se torna em uma instância de classe DOM em JavaScript. Alguns destaques:

- **Node** – Representa genericamente qualquer nó da árvore e encapsula o acesso a informações comuns a todos os nós: tipo, nome, valor, atributos etc. Também concentra os métodos fundamentais de acesso a dados, navegação e modificação da árvore. As classes a seguir são especializações de Node.
- **Element** – Elementos HTML/XML, que são representados por tags, e.g., `<text>`, `<path>`, `<circle>` etc.
- **Attr** – Atributos associados a elementos, e.g., `cx`, `cy` e `r` associados a `<circle>`.
- **Text** – Conteúdo texto livre dentro do documento.
- **Document** – Nó raiz da árvore que representa o documento completo.

Atributos disponíveis em cada nó (instâncias de Node) são usados para a navegação pelo documento. A Figura 4.10 apresenta a árvore DOM da Figura 4.8 e algumas operações de navegação. Através do método getElementById() do objeto document, é possível se alcançar diretamente um elemento a partir do valor de seu identificador. O atributo firstChild mantém um ponteiro para o seu primeiro nó filho, se houver. O atributo nextSibling permite alcançar seu nó irmão – e.g., navegar de um nó

`<circle>` para o seu vizinho. Através destes dois atributos, é possível se percorrer uma árvore inteira.

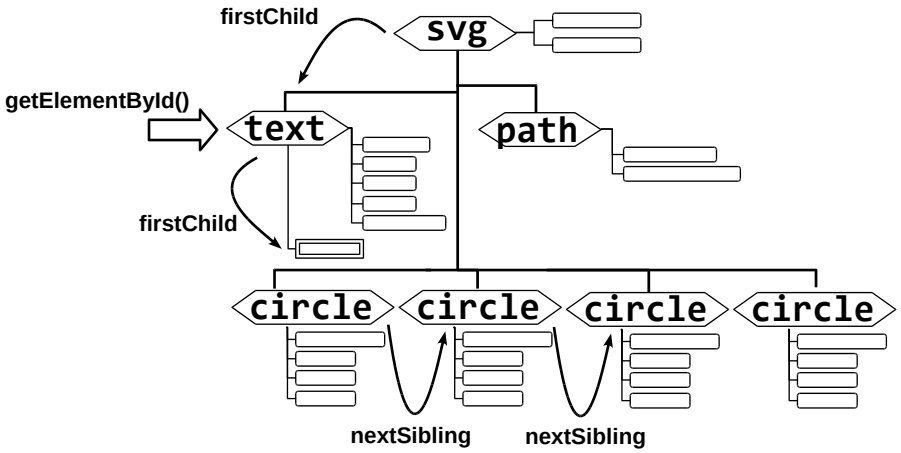


Figura 4.10. Navegação pela árvore DOM do SVG.

4.4.2. Selector

Embora o método `getElementById()` seja amplamente usado para o acesso de um elemento a partir do seu identificador, há uma nova maneira de se alcançar um elemento específico em uma árvore DOM. Trata-se da API Selector (Kesteren, 2013) que dispõe de uma sintaxe compacta e poderosa para a seleção de nós, alinhada com a sintaxe do selector do CSS e similar a usada pelo jQuery em sua biblioteca selector (Sizzle).

Considere o trecho HTML a seguir que será usado para a extração de dados:

```
...
<main>
...
<table id="score">
  <thead>
    <tr>
      <th>Test</th>
      <th>Result</th>
    </tr>
  <tfoot>
    <tr>
      <th>Average</th>
      <td>82%</td>
    </tr>
  <tbody>
    <tr>
      <td>A</td>
      <td>87%</td>
    </tr>
    <tr>
      <td>B</td>
      <td>78%</td>
    </tr>
    <tr>
      <td>C</td>
      <td>81%</td>
    </tr>
</table>
...
</main>
```

Podemos realizar a extração tanto alcançando os elementos pelo método `getElementById()` quanto pelo API Selector. Usando o `getElementById()` teríamos:

```
var table = document.getElementById("score"),
    groups = table.tBodies,
    rows = null,
    cells = [];

for (var i = 0; i < groups.length; i++) {
  rows = groups[i].rows;
  for (var j = 0; j < rows.length; j++) {
    cells.push(rows[j].cells[1]);
  }
}

console.log(cells[0].innerText); // 87%
```

Esta abordagem tem as seguintes limitações:

1. Ela exige um elemento com atributo `id` como ponto de partida.
2. Foi necessário um esforço de programação para navegar pelos nós.
3. A estratégia de acesso é baseada em funções.

Como será demonstrado a seguir, a extração fazendo-se uso da API Selector torna o processo mais conciso e simples de implementar e interpretar. Os nós alvo são expressos por uma sentença:

```
var qs = document.querySelectorAll("table:nth-of-
type(1)>tbody>tr>td:nth-of-type(2)");

console.log(qs[0].innerText); // 87%
```

Desde o CSS3, muitos seletores foram adicionados como Seletores de Nível 3 (Kesteren, 2013), que agora podem ser usados não apenas nas folhas de estilo, como também no JavaScript a partir da API DOM.

Neste exemplo foi usado o método `querySelectorAll()` para retornar todos os nós que atendem à expressão, porém a Figura 4.5 também mostra um exemplo do método `querySelector()` que retorna um elemento alvo.

Um objeto Document dispõe de métodos para a criação dinâmica de elementos na árvore. Por exemplo, o método `createElement` cria um novo elemento. Cada nó possui um método `appendChild` para a inserção de novos nós como filhos. Do mesmo modo, `removeChild` remove tal associação de filho.

Na medida em que um aplicativo JavaScript interage com uma interface DOM modificando um documento HTML, o navegador atualiza dinamicamente a apresentação da página para refletir cada mudança. Isto permite tornar os documentos Web espaços de apresentação e interação para aplicativos JavaScript.

4.4.3. Eventos

Os eventos direcionam tudo que está acontecendo dentro do documento Web e está no núcleo do JavaScript, uma vez que ele está na origem. Tratando a questão de forma simplificada, um evento é um *broadcast* no documento DOM de cada acontecimento:

`click`, `scrollup`, `scrolldown`, `focus`, `blur` etc. Embora haja elementos que tenham eventos em comum, o modo como eventos são manipulados e o seu impacto serão diferentes.

Para ter uma visão de todos os eventos suportados pelo seu navegador, você pode executar a seguinte rotina no console do seu navegador (e.g., Firefox Firebug ou Chrome developer tools):

```
// Source:

// http://coding.smashingmagazine.com/2012/08/17/javascript-events-responding-user/
var log = function(what){ console.log(what) },
    i = '',
    out = [];
for (i in window) {
  if ( /^on/.test(i)) { out[out.length] = i; }
}
log(out.join(', '));
```

Considere o evento `click` e o exemplo de um "`form`" e um link "`a`" (âncora).

Se nós clicarmos em um "`input[type=submit]`" dentro do `form`, um novo evento é emitido pelo `form` e este evento irá ler o tag que é pai do `form`, e usar o valor do atributo chamado `method` para fazer uma chamada HTTP (seja GET ou POST) e submeter os valores do `form` baseado em cada valor de `input` disponível.

Um outro manipulador para o evento click é tag "`a`", quando ele recebe um clique. O manipulador executará uma ação equivalente a chamar o método `window.location`; ele lê o valor do atributo `a[href]` do elemento clicado e faz o navegador ler a nova localização.

Um detalhe importante a ser lembrado sobre manipulação e acionamento de eventos é que há muitos exemplos desatualizados em que é sugerido o uso de atributos do *tag* para disparar funções de evento, tal como o trecho de código a seguir. Esta prática era utilizada há muito tempo, mas a documentação a respeito não se atualiza rápido o suficiente.

```
// If you do this, please stop.
// HTML
<button onclick="javascript:sayhello()" >Hello!</button>

// JavaScript
function sayhello() {
  alert ('Hello!');
}
```

O problema desta técnica é que não há um controle central de eventos, que ficam espalhados em diversos pontos do documento. O que a documentação do Mozilla Developer Network e o jQuery sugerem é que você associe um manipulador de eventos a um nó via JavaScript e atribua uma função ao evento, como o exemplo a seguir:

```
// HTML
<button id="hello">Hello!</button>


// JavaScript
document.getElementById("hello").addEventListener("click",  function()
{
      alert('Hello!');
}, false);
```

4.5. Engenharia de Marcação e Estilos

4.5.1. Markup Patterns

A forma como usamos HTML, JavaScript e CSS para construir código reusável e eficiente tem relação direta com a velocidade de *renderização* dos nossos documentos.

Como produzir documentos HTML é trivial, as pessoas que estão construindo Web sites pela primeira vez são capturados nas mesmas "armadilhas".

Muitas práticas adotadas nas últimas décadas para o desenvolvimento de Web sites pode torná-los difíceis de manter. De fato, quantas vezes ouvimos dizer que alguém está apenas mudando o visual (*'skin'*) de uma aplicação preexistente?

A questão central torna-se então: por que não se usa o *markup* existente, mudando-se apenas o tema em um novo projeto? Não estamos considerando aqui aqueles casos em que o projeto tenha novos requisitos de software, uma nova estrutura, e nós queremos reimplementá-lo para torná-lo melhor, redefinindo inclusive as definições CSS.

O conhecimento dos conceitos arquiteturais por detrás do HTML, CSS e JavaScript é fundamental para potencializar a ação do desenvolvedor do *frontend*. Os casos comuns que tornam um documento difícil de manter são:

1. Redundância nos estilos, mesmo com o uso de propriedades herdadas.
2. Falta de uso de estilos herdados (quando apropriado), causando redundância.
3. Estilos demasiadamente específicos, quando partes dele poderiam ser reusadas.
4. Interpretação incorreta do *box model* atual.
5. Criação de estilos dependentes de localização.

Do mesmo modo que o domínio da arquitetura de software define seus *design patterns* (Gamma et al., 1995), é possível defini-los também no domínio do desenvolvimento de *frontends*. A seguir apresentamos patterns para a construção de CSS.

Para ajudar a descrever como criar um documento usando componentes reusáveis, Nicole Sullivan introduziu uma técnica chamada *Object Oriented* CSS - "OOCSS" (Sullivan, 2011).

Os princípios são simples e podem ser sumarizados como:

> *"Um 'Objeto' CSS é a repetição de um padrão visual, que pode ser abstraído em um fragmento independente de HTML, CSS e possivelmente JavaScript. Uma vez criado, um objeto pode ser então reusado ao longo do site." (Sullivan, 2011)*

Tal como os objetos, os estilos e sub-estilos (estilos componentes do estilo mais abrangente) devem ser concebidos hierarquicamente. Há muitos autores que documentam e nomeiam suas estratégias de atuação, mas todas elas têm uma coisa em comum: pense em um elemento visual como um componente e garanta que ele é autocontido.

Para que isto aconteça é necessário aplicar duas regras:

1. Separar a estrutura do visual (*skin*);
2. Separar o recipiente (*container*) do conteúdo.

Por exemplo, um elemento visual comum consiste em um conjunto de poucas palavras ao lado de uma imagem e, algumas vezes, alguns metadados associados. Vamos denominar este elemento de 'media', conforme exemplo ilustrado na Figura 4.11. Este elemento é muito comum e pode assumir várias formas. A Figura 4.11 mostra a imagem do lado esquerdo do texto, mas é possível, por exemplo, haver variantes com a imagem do lado direito.

Figura 4.11. Exemplo de uso do elemento media (fonte: http://www.stubbornella.org/).

O trecho de código a seguir ilustra a representação HTML + CSS do exemplo da Figura 4.11. Note que há uma concepção hierárquica baseada em objetos, em que os estilos `media-img` e `media-body` foram concebidos para o conteúdo que estará subordinado ao container `media`. Para tratar as variantes do estilo, a metodologia recomenda que a classe pai determine como deve se comportar os subordinados.

```
<div class="media attribution">
  <a href="http://twitter.com/stubbornella"
     class="media-img img">
    <img src="http://stubbornella.com/profile_image.jpg"
         alt="me" />
  </a>
  <div class="media-body">
    @Stubbornella 14 minutes ago
  </div>
</div>
```

```
/* ====== media ====== */
.media {margin:10px;}
.media, .media-body {
  overflow:hidden; _overflow:visible; zoom:1;}
.media .media-img {float:left; margin-right: 10px;}
.media .media-img img{display:block;}
```

O trecho ilustra o elemento media introduzido por Nicole Sullivan, mas alterado para refletir a metodologia SMACSS de Jonathan Snook (Snook, 2013).

Embora as metodologias sejam similares, elas não têm suas especificidades de acordo com o foco. Por exemplo, o OOCSS tem foco em construir coisas tão pequenas e modulares quanto possível, enquanto o SMACSS recomenda tornar-se explícito ao custo de se adicionar mais nomes de classes.

4.5.2. Criando um Módulo de Marcação

Tudo é uma caixa (“box”).

Uma vez construído, torna-se simples compreender a aplicação do elemento `media`, mas como podemos criar um novo elemento é outra questão. Para atender a esta demanda, Yandex criou uma metodologia que orienta o modo de fazer componentes (Yandex, 2013).

Ao invés de se criar componentes altamente especializados, é recomendado adotar uma visão diferente do problema e abstraí-lo que nos permita reusá-lo. Os tópicos a seguir sintetiza alguns dos aspectos chave da metodologia BEM (Yandex, 2013):

1. cada elemento visual é uma caixa;
2. caixas têm partes separadas;
3. caixas podem ter variantes;
4. partes e caixas têm uma representação básica e comportamento;
5. partes e caixas podem ter uma camada temática aplicada a eles.

O projeto de uma caixa inclui seu comportamento em JavaScript. Neste sentido, deve-se considerar o que pode acontecer com ela – incluindo eventos e mudanças de estado – e quais os impactos na caixa e na estrutura em que está inserida.

4.5.3. Pré-processadores CSS

Se por um lado o JavaScript está preparado para o reúso de patterns, uma estratégia equivalente para reuso de CSS ainda é um problema em aberto.

Embora o W3C e empresas que desenvolvem navegadores estejam empenhados na definição de componentes reusáveis, há algumas especificações que irão potencializar o modo como criamos marcações, dentre os recentes: Web Components, CSS Variables e CSS Fragmentation. Enquanto isso, há a necessidade de se criar componentes modulares com o que temos hoje.

Na criação de componentes, torna-se necessária a possibilidade de se criar variáveis, de se referir a outros *patterns* ("*mixins*") ou de se usar funções, e.g., para calcular variantes de cores. Para atender a esta demanda, surgiram os pré-processadores CSS. Um exemplo é o trecho CSS a seguir que usa variáveis, mixins e funções do LESS (Sellier, 2013):

```
@base: #f938ab;

.box-shadow(@style, @c) when (iscolor(@c)) {
  box-shadow:         @style @c;
  -webkit-box-shadow: @style @c;
  -moz-box-shadow:    @style @c;
}

.box-shadow(@style, @alpha: 50%) when (isnumber(@alpha)) {
  .box-shadow(@style, rgba(0, 0, 0, @alpha));
}

.box {
  color: saturate(@base, 5%);
  border-color: lighten(@base, 30%);
  div { .box-shadow(0 0 5px, 30%) }
}
```

Em LESS, variáveis e parâmetros são prefixados por `@`. No exemplo, a variável `@base` define o valor `#f938ab` a ser usado em vários pontos do CSS. O `.box-shadow` é um estilo reusável na forma de *pattern*. Ele recebe dois parâmetros que definem valores de algumas das suas propriedades. No exemplo, há duas possibilidades de `.box-shadow` com uma condição de escolha definida pela cláusula `when`. A classe `.box` instancia o `.box-shadow` dando valores aos seus parâmetros. Algumas funções usadas neste exemplo são: `iscolor()`, `isnumber()` e `lighten()`.

O LESS pode ser pré-processado no cliente ou no servidor. No caso do cliente, uma rotina em JavaScript carregada em tempo de execução realiza a conversão (Sellier, 2013):

```
<link rel="stylesheet/less" type="text/css"
href="styles.less" />

<script type="text/javascript">
    less = { env: "development" };
</script
<script src="less.js" type="text/javascript"></script>
```

Embora esta abordagem seja conveniente para uso em tempo de desenvolvimento e não exija um sistema de conversão rodando no servidor, ela não é adequada para distribuição, já que isto exigiria uma reconversão para cada visitante.

O pré-processamento no servidor distingue a versão local de desenvolvimento daquela para distribuição. A execução de um pré-processador *standalone* converte o CSS em LESS para uma versão estática com CSS puro.

Na medida em que o desenvolvimento envolve uma crescente quantidade de ferramentas, tais como as apresentadas nesta seção, há uma tendência recente em automatizar a sua ação coordenada e gerenciar o ciclo de vida de desenvolvimento das aplicações através de uma ferramenta de workflow chamada Yeoman (http://yeoman.io/).

4.5.4. Desenvolvimento JavaScript centrado em marcação

Nas seções anteriores foi destacada a importância de se estruturar *patterns* reusáveis para marcação. Nesta seção, estendemos o conceito para a sua aplicação no contexto de JavaScript, mais especificamente para uma técnica que chamaremos aqui de desenvolvimento centrado na marcação.

Um bom exemplo desta técnica é mostrado na documentação do *toolkit* Bootstrap (http://getbootstrap.com/2.3.2/javascript.html):

```
<div class="dropdown">
  <a class="dropdown-toggle" data-toggle="dropdown" href="#">
    Dropdown
  </a>
  <ul class="dropdown-menu" role="menu">
    ...
  </ul>
</div>
```

Este exemplo do `dropdown` ilustra o que é denominado data-api no Bootstrap, na qual é possível especificar e parametrizar um componente *dropdown* sem que se escreva uma única linha de código. Os componentes são parametrizados por metadados introduzidos na marcação e alguns dos componentes providos têm um comportamento JavaScript associado.

Componentes usando esta estratégia propiciam reuso e aumento de produtividade, já que eles cuidam de detalhes como o registro e manipulação de eventos, poupando esforço de codificação. Muitas interações podem ser generalizadas e

manipuladas de maneira similar. Um dos usos notáveis desta técnica foi no Facebook, quando eles otimizaram seu código JavaScript usando um 'Primer' (Adeagbo, 2013).

Este concepção de componente centrada nos dados será retomada na seção de Web Components.

O uso de `ids` (referenciando um único elemento) ao invés de classes (referenciando grupos de elementos) em estilos potencialmente reusáveis vai no sentido contrário das tendências aqui apresentadas e é considerada uma prática ruim. Estratégias de *markup*, como o Object Oriented CSS e o SMACSS, recomendam maximizar o reuso generalizando todos os padrões CSS, JavaScript e HTML.

4.6. Web Components

Web Components é uma iniciativa do WebApps WG de criar um modelo de componentes para a Web (Cooney & Glazkov, 2013).

O desafio de produzir um modelo de componentes para a Web (Web Component model) vai além de produzir um modelo de componentes de software para JavaScript. O núcleo da Web é construído sobre uma abordagem orientada a dados (data-driven), i.e. documentos são hospedeiros e dão estrutura às apps. A despeito da crescente relevância do JavaScript, o modelo centrado em documentos (document-centric ou data-centric) ainda prevalece.

Para tornar claro o que consideramos aqui um modelo de componentes data-centric, vamos partir de uma definição clássica de um componente, ilustrada na Figura Figura 4.12. Mesmo que não haja um consenso sobre a definição de componente de software (Broy et al., 1998), há uma distinção comum entre a interface pública – que apresenta uma abstração parametrizável do componente – e uma implementação privada – que define o comportamento ou serviço do componente. Os parâmetros da interface direcionam o seu comportamento. A interface tem dois papéis aqui: (i) ela abstrai os aspectos principais do componente, escondendo detalhes de implementação de um observador externo; (ii) ela torna explicitas as dependências/interações do componente com o sistema externo.

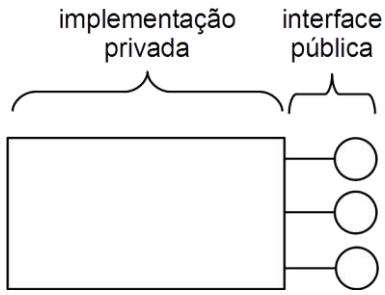


Figura 4.12. Diagrama de componente clássico.

4.6.1. *Templates*

Outro consenso relativo a componentes é que eles são blocos de conteúdo ou serviço reusáveis. Há diferentes perspectivas sobre o que pode ser este conteúdo, e.g., um código executável ou um binário. Em nosso contexto, o conteúdo será um segmento reusável de HTML/CSS/JavaScript.

Componentes precisam de um mecanismo independente para predefinir blocos de conteúdo antes do seu uso, de modo que fiquem disponíveis para serem usados tantas vezes quanto necessário. Este é o propósito dos *templates*.

O segmento a seguir ilustra como o carro SVG introduzido na Figura 4.4 pode ser transformado em um bloco reusável através de *templates*. Acima à esquerda aparece a descrição SVG do carro, apresentada anteriormente, na forma de *template*. Acima à direita é apresentada uma rotina em JavaScript responsável pela instanciação e uso *template*. Abaixo é apresentado o segmento no documento HTML referente ao ponto onde o template será inserido. A Figura 4.13 ilustra a estratégia de funcionamento dos templates em HTML.

```
<template
id="carComponent">

  <style scoped>
    …
  </style>

  <svg …>
    …
  </svg>


</template>
```

```
function applyTemplate()
{
  var carComponent =
                document.querySelector("#carComponent");

  var myCar = document.querySelector("#myCar");

  myCar.appendChild(carComponent.content.cloneNode());
}
```

```
<div id="myCar">
  <!-- empty -->
</div>
```

Conforme ilustrado no exemplo, cada template SVG, pode encapsular segmentos de um documento Web auto-contidos, ou seja, que representem uma sub-árvore DOM na íntegra. Isto inclui HTML, CSS, JavaScript e outras descrições XML aptas a serem incluídas em documentos HTML, tal como o SVG do exemplo. O CSS recebe um atributo “scoped” para restringir sua atuação no contexto do template.

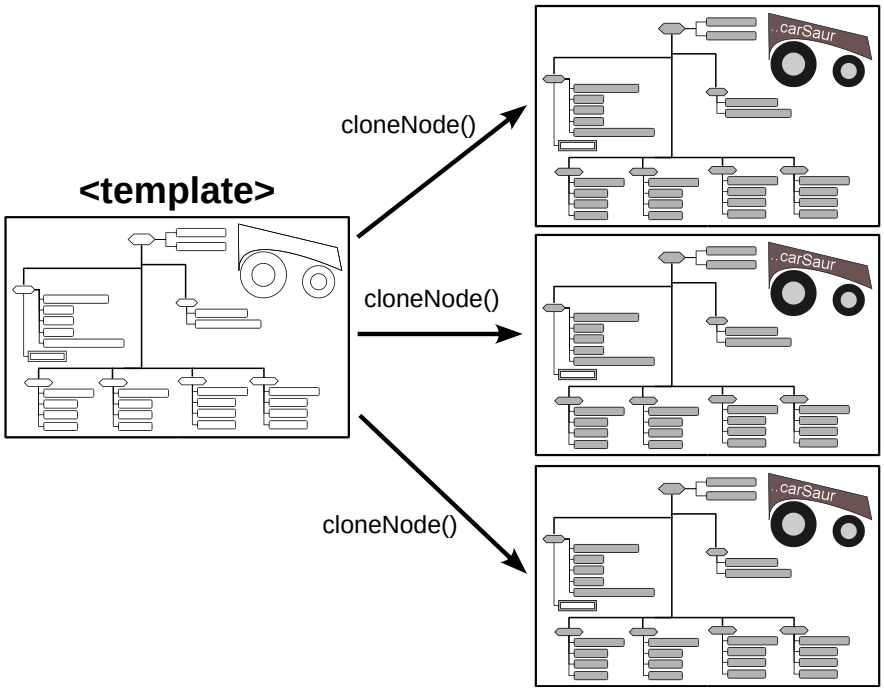


Figura 4.13. Instanciação de templates através da clonagem.

Como está ilustrado à esquerda da Figura 4.13, um *template* não é apresentado pelo navegador, mas seu conteúdo passa pelo parser e fica disponível para uso. O uso de um template é feito através de um método de clonagem do original, ativado pelo método `cloneNode()`.

Este processo está detalhado no método `applyTemplate()` apresentado anteriormente, que executa as seguintes tarefas:

1. Localiza o template dentro do documento a partir de seu identificador. Neste caso foi usada a função querySelector() que localiza um elemento a partir de uma query.
2. Localiza o alvo de inserção no documento Web. Trata-se da myCar, ilustrada abaixo do método.
3. Instancia o template a partir do processo de clonagem (função cloneNode()).
4. Adiciona a árvore instanciada no alvo.

Tão logo o template é clonado e inserido na árvore DOM ele passa a funcionar, e o carro aparece no documento HTML.

4.6.2. Shadow DOM

O tema encapsulamento em componentes de software pode se ampliar bastante quando aplicado ao contexto de Web. Uma das questões fundamentais diz respeito ao encapsulamento da árvore DOM. Ao se pensar em componentes Web que coexistem em um mesmo espaço de aplicação, o compartilhamento de uma grande árvore DOM pública e visível por todos pode se tornar um problema.

O shadow DOM (Cooney, 2013) (Glazkov, 2011) tem o propósito de encapsular árvores DOM na construção de componentes. Ele segue a lógica de encapsulamento em que parte da árvore DOM é pública (apenas a abstração do componente) e parte é privada (detalhes da sua implementação).

No texto anterior (Santanchè, 2013) apresentamos um exemplo do slider tratado por Glazkov (2011). Neste texto, apresentaremos a aplicação do conceito no componente carro. Considere que um desenvolvedor deseja criar um componente carro, de tal forma que ele possa configurá-lo facilmente através de marcação, como está ilustrado na Figura 4.14. No exemplo, a classe "`carComp`" associada ao marcador de id "`myCar`" indica que ele deseja instanciar um componente `car`. Para tornar o componente reusável, é importante que seja possível parametrizá-lo (e.g., cor, tamanho, tipo de roda, nome do carro etc.), de modo que o componente se adapte a diferentes contextos. Neste caso, foi escolhido um único parâmetro – o nome do carro – para fins de simplificação.

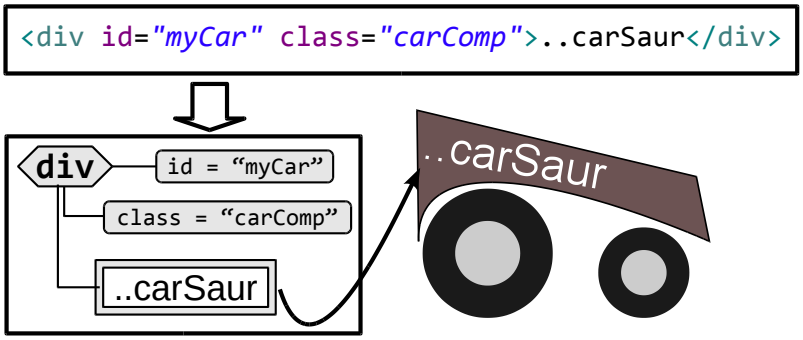


Figura 4.14. Interface de configuração de um componente carro.

Este parâmetro representa uma abstração externa simplificada. Ao alterar o nome do carro dentro da `<div>` de classe "`carComp`", o que de fato deve acontecer é que o nome do carro deve mudar no gráfico. Isto significa uma distinção entre a abstração pública e seu reflexo na implementação privada, como foi introduzido na Figura 4.12 e está ilustrado na Figura 4.15. A parametrização da abstração pública pode ser vista como uma árvore DOM simplificada, cujos valores afetarão uma segunda árvore DOM privada (shadow DOM), que esconde os detalhes SVG de como o carro é desenhado. Os valores modificados na interface devem refletir em pontos específicos da árvore privada que são chamados de *insertion points*.

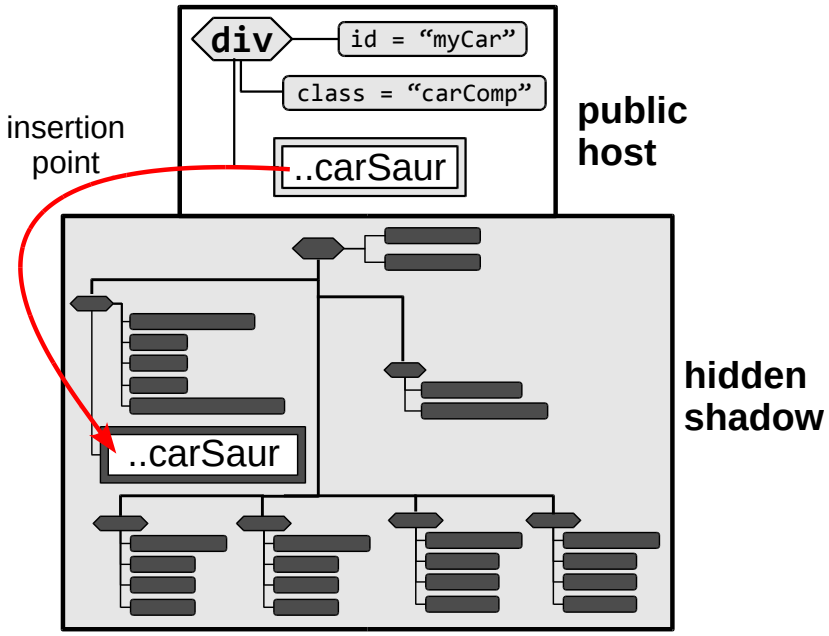


Figura 4.15. Divisão pública/privada do shadow DOM.

O shadow DOM possibilita que o aplicativo, ao acessar a árvore DOM, veja somente a parte pública ilustrada na Figura 4.14. A árvore que compõe esta abstração pública perceptível pelo JavaScript é chamada de host. Ela não é usada para renderizar a apresentação. Entretanto, é possível associar a esta árvore uma segunda responsável apenas pela apresentação. Esta árvore é conhecida como root. Ela não é visível pelo JavaScript mas será usada para montar a apresentação visual.

A árvore que ficará privada (shadow) pode ser definida na forma de template como ilustra a definição a seguir (esquerda). O ponto de inserção é definido através do tag `<content>`. Para fins de ilustração no exemplo, foi introduzido uma `<div>` externa à figura SVG onde será colocado o nome que está no parâmetro. Em testes realizados, o <contente> ainda não se comporta bem dentro do SVG, ao contrário do HTML. Uma estratégia para contornar este problema é fazer a substituição via JavaScript.

```
<template id="carComponent">

  <style scoped>
    …
  </style>

  <svg …>
    …
  </svg>

  <div id="textCarName"
       class="nameStyle"
       width="181px">
    <content></content>
  </div>
</template>
```

```
function applyTemplate()
{
  var myCar = document.querySelector("#myCar");

  var carComponent =
   document.querySelector("#carComponent").content;

  var myCarShadow = myCar.webkitCreateShadowRoot();
  // standard: createShadowRoot()

  myCarShadow.appendChild(carComponent);
}
```

```
<div id="myCar" class="carComp">..carSaur</div>
```

No código JavaScript são executadas as seguintes tarefas:

1. O host identificado por `myCar` é recuperado. Neste caso, foi usado o identificador para fins de simplificação mas, para fins de reúso, é mais adequada a recuperação a partir da classe.
2. É recuperado o root do *template* que será usado para a criação da parte escondida que descreve o carro em SVG.
3. É montada uma árvore shadow DOM do host através do método createShadowRoot() (este é o método padrão; no exemplo foi usada uma implementação provisória que roda no Google Chrome).
4. A árvore shadow é associada ao root do *template*. Neste momento os parâmetros do host irão automaticamente alimentar os pontos de inserção.

É possível haver mais de um parâmetro no host. Cada parâmetro estará em um atributo ou sub-elemento do host. O exemplo a seguir ilustra este caso, em que há um elemento identificado como "`carName`", dentro do elemento host, com o nome do carro. Para este caso, o ponto de inserção pode incluir um parâmetro `select` (vide exemplo a seguir) e selecionar o que irá recuperar. A expressão segue a abordagem da API Selector, descrita na Seção 4.4.

```
<div id="myCar" class="carComp">
  <div id="carName">..carSaur</div>
</div>
```

```
<content select="#carName"></content>
```

4.7. Evolução das arquiteturas de aplicativos móveis

4.7.1. Aplicativos usando tecnologias Web como tecnologia secundária

Em inúmeras plataformas, tecnologias Web não estavam na lista oficial de tecnologias suportadas para o desenvolvimento de aplicações, mas sim para papéis menores. Nessas plataformas, tecnologias Web só podem ser utilizadas pela inserção na aplicação de um componente nativo, o webview, que funciona como um motor Web para o qual são despachadas as partes Web. Dessa forma, essas plataformas suportam HTML, CSS e JavaScript como recursos complementares e via o componente webview. As aplicações não são desenvolvidas integralmente com as tecnologias Web por não serem suportadas direta/nativamente.

Por esta razão, para os casos em que se deseja criar um aplicativo usando integralmente tecnologias Web, é necessário utilizar um aplicativo-envoltório, conforme ilustrado na Figura 4.16, implementado em alguma das tecnologias nativas, que faz o bootstrap do motor e o aplicativo Web dentro dele. Para utilizar os recursos de hardware, são usadas bibliotecas híbridas, que de um lado se apresentam como uma API JavaScript para que o aplicativo Web tenha acesso ao hardware, e do outro são implementadas em código nativo para acessar o hardware usando as APIs nativas. Uma

das bibliotecas mais conhecidas que implementa essas duas partes é o Phonegap, da Adobe, e sua equivalente livre, Cordova, da Apache. A maior parte das plataformas utiliza essa abordagem, incluindo Android e iOS.

Figura 4.16. Web App com aplicativo-envoltório.

Outra alternativa está ilustrada na Figura 4.17. O código escrito em HTML, CSS e JavaScript é compilado para código nativo da plataforma específica em que ele será executado. Deste modo, o mesmo código pode ser compilado para diversas plataformas. Exemplos desta tecnologia são o Titanium Mobile Development Environment (http://www.appcelerator.com/platform/titanium-platform/) e o Marmelade (http://www.madewithmarmalade.com/).

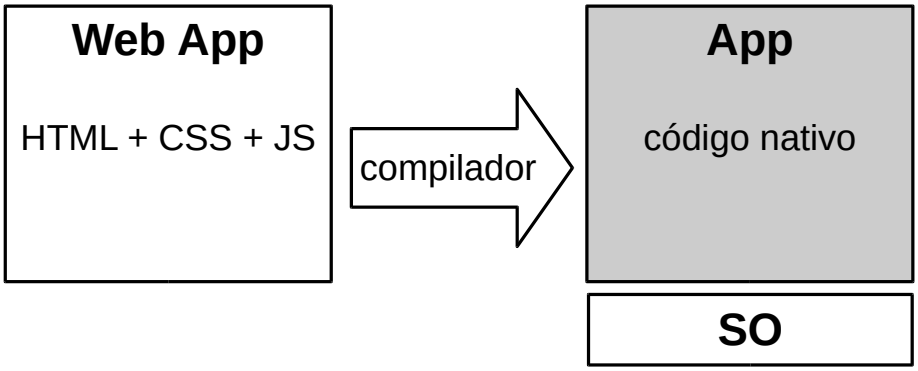


Figura 4.17. Web App compilada para a plataforma.

4.7.2. Aplicativos usando tecnologias Web como tecnologia primária

As plataformas que não oferecem tecnologias Web por padrão – apresentadas na seção anterior – possuem muitos problemas. Entre eles, os principais são o acesso ao hardware, que depende de bibliotecas de terceiros e não é uniforme, e o desempenho, que tende a ser muito menor que na plataforma nativa por conta da quantidade de indireções.

Esta seção analisará como algumas plataformas tentaram resolver isso em duas diferentes estratégias: (i) como não havia especificação padrão Web para acessar os componentes de hardware diretamente, cada plataforma criou uma API Web própria para acesso a tais componentes; (ii) a plataforma oferece uma solução híbrida combinando desenvolvimento nativo e Web, que dispõe de apenas parte das ferramentas Web para o desenvolvimento de aplicações. No caso (i), tem-se uma abordagem muito parecida com a anterior, mas nativa, e não dependente de uma biblioteca externa,

implementada por terceiros. No caso (ii), tem-se um desenvolvimento mais próximo do nativo, com uma curva de aprendizado menor, em que parte do conhecimento com tecnologias Web pode ser reaproveitado. Em ambas as soluções os problemas de desempenho também são diminuídos, já que o suporte é nativo, como ilustra a Figura 4.18.

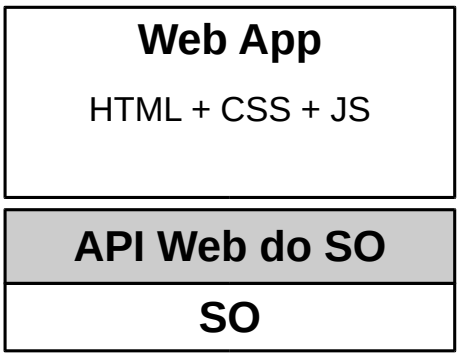


Figura 4.18. Web App com API suportada pelo SO.

Várias plataformas mais recentes utilizam essa abordagem. Alguns exemplos são BlabkBerry 10 – abordagem (i) – e Windows 8 – abordagem (ii).

4.7.3. Aplicativos Web como aplicativos nativos ou de primeira classe

Nesse caso, tecnologias Web padrão compõem o mecanismo nativo para construção de aplicações. Por nativo entende-se que todo o processo de interação com o SO e o hardware – incluindo-se dispositivos de apresentação – é feito por meio de protocolos e padrões Web. A Figura 4.19 ilustra esta abordagem de aplicativo, que pode ser confrontada com aquela ilustrada na Figura 4.16.

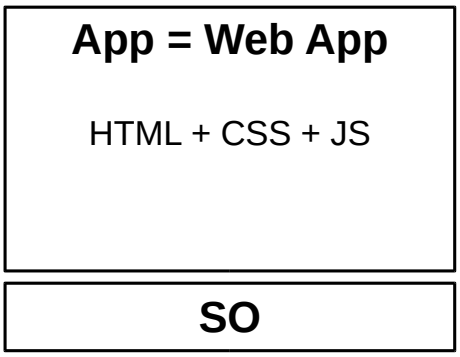


Figura 4.19. Web App nativa.

Fazendo-se um paralelo: as plataformas Android e iOS possuem uma API própria para interação com o display, que pode ser acessada por qualquer linguagem de programação considerada nativa para a plataforma; plataformas Web nativas, por outro lado, usam apenas as APIs Web padrão, definidas em conjunto com orgãos como o W3C e WHATWG. No segundo caso, a disposição visual dos elementos, por exemplo, fica a cargo do HTML+CSS, que é manipulado via JavaScript, da mesma forma que o seria em qualquer navegador Web. Adicionalmente, aplicações para plataformas desse tipo também funcionam (ou deveriam funcionar, dados padrões que ainda estão em

construção) em qualquer navegador, independendo de plataforma ou sistema operacional, dependendo exclusivamente do suporte dessa plataforma aos padrões Web.

Essa proposta é a mais alinhada com aquela aberta e livre da Web. Entretanto, ainda há limitações a serem superadas, já que não existem especificações padronizadas para as APIs que dão acesso ao componentes de hardware. Portanto, plataformas que querem seguir nesta linha devem colaborar para construir novos padrões. O Firefox OS é um exemplo de plataforma que utiliza essa abordagem, ainda muito pouco explorada.

4.8. Planejamento para criação de interfaces e considerações sobre usabilidade

Todo sistema homem-máquina tem como objetivo facilitar à execução de tarefas a serem realizadas por humanos. Por isso, considerar o usuário como foco principal ao desenvolver o sistema homem-máquina é fator determinante como critério de sucesso do sistema.

Existem diversas métodos que visam otimizar à concepção de um sistema. Por exemplo, há o ciclo de desenvolvimento em V e em cascata. Mas, entre estes métodos, a norma ISO 9241-210 (2010) é a que tem maior foco no usuário e deve ser aplicada ainda antes da fase de concepção.

A norma ISO 9241-210 é dividida em etapas: Planificar o processo de concepção, definir o uso de contexto e suas limitações, definir as especificações das exigências (*must*, *should*, *could*), conceber soluções e, por fim, avaliar o produto de acordo com as exigências. Cada etapa será detalhada nos parágrafos a seguir.

4.8.1. Planificar o processo de concepção

A primeira etapa da norma ISO 9241-210, planificar o processo de concepção, consiste em definir o tamanho da equipe que irá participar do projeto de desenvolvimento do sistema, bem como a definição do cronograma e do orçamento. A concepção deve ser baseada sobre uma compreensão clara de quem são os usuários, as tarefas e ambiente de uso do sistema. Os usuários devem participar durante todo o processo de desenvolvimento do sistema e as equipes de concepção devem pertencer a equipe multidisciplinares.

4.8.2. Especificar o contexto de uso

Para especificar o contexto de uso, deve-se primeiro, definir os seguintes aspectos:

- Quem são os usuários primários e secundários.
- Quais os objetivos e as tarefas a serem executadas.
- Qual o ambiente/contexto de uso.

4.8.3. Especificar as necessidades dos usuários

Nesta etapa, deve-se reunir todas as informações que foram levantadas e traduzi-las em requisitos e exigências funcionais, que devem refletir os objetivos do sistema e dos usuários para com o sistema.

Deve-se também, considerar todas as limitações do sistema. Desta forma, no momento da decomposição das tarefas e dos objetivos relacionados à eficácia, esses podem ser considerados de forma optimizada.

De acordo com a norma ISO 9241-11, a eficácia, eficiência e a satisfação são definidos como:

- Eficácia: precisão e completitude com que os usuários alcançam os objetivos.
- Eficiência: os recursos utilizados em relação à exatidão e exaustão que os usuários alcançam seus objetivos.
- Satisfação: quando não há desconforto e há atitudes positivas por parte dos usuários ao utilizar o sistema.

4.8.4. Produzir soluções de concepção que correspondem as necessidades dos usuários

Uma forma de verificar se a concepção corresponde às exigências dos usuários, é desenvolver protótipos com diferentes níveis de fidelidade. Inicialmente, a criação de uma versão de baixa fidelidade e, posteriormente, adicionar mais detalhes de forma que os membros da equipe de concepção possam validar os requisitos.

Os protótipos considerados de baixa fidelidade (no caso de interfaces são conhecidos como wireframes e sketches) são os que oferecem um mínimo para ilustrar um caso e é útil como apoio para explicar uma ideia. Eles podem ser fabricados de diversos tipos de materiais, por exemplo, em papel, em cartolina etc. Eles são geralmente simples, rápidos e baratos a serem produzidos. A principal vantagem deste tipo de protótipo, é que ele pode ser facilmente modificado para ajudar na exploração de ideias e concepções alternativas.

O último nível de prototipagem deve ser o mais próximo do produto final. É interessante desenvolver um protótipo do sistema o mais real possível para que usuários possam testá-lo. Esse tipo de experimentação tem um grande valor por validar os conceitos antes de colocá-los em produção final.

4.8.5. Verificação e validação da interface

Depois de conceber os diferentes tipos de soluções, é importante verificar o sistema em relação às exigências do projeto, bem como, entender como as limitações do sistema podem afetar os usuários. Desta forma, a concepção será guiada por avaliações detalhadas e centradas no usuário. Este processo interativo garantirá que cada fase ajude a melhorar o caso de uso e aumentará a qualidade do produto a cada protótipo criado.

4.8.6. Avaliar a concepção de acordo as exigências

Uma vez que o sistema é desenvolvido, existem dois métodos que podem ser considerados para verificar se todas as condições do sistema foram atendidas:

- O uso da lista de verificação em conformidade da ISO.
- Os testes de usabilidade para fim de validação.

4.8.7. Lista de verificação em conformidade da ISO 9241-210

Uma lista de elementos deve ser usada para validar e assegurar que nenhum ponto foi esquecido em relação à norma em vigor.

Uma lista permite, entre outros:

- determinar quais as recomendações são aplicáveis;
- determinar se as recomendações aplicáveis foram seguidas;
- fornecer uma lista de todas as recomendações aplicáveis e se elas foram seguidas.

Independente do método de verificação considerado o mais adequado, a lista proposta pela ISO 9241-151 oferece um espaço para indicar o nível de conformidade, bem como, os métodos utilizados que podem ser inscritos na coluna de comentários.

Os números e os títulos dos Artigos/Parágrafos estão presentes dentro das duas primeiras colunas da tabela. A terceira coluna é utilizada para indicar se a recomendação dentro de cada artigo ou parágrafo é aplicável ou não. Todos esses aspectos devem ser verificados em que se concerne ao quadro de concepção do sistema.

4.8.8. Analise das tarefas com os usuários - Validação

O teste de validação é efetuado em uma etapa avançada do processo para verificar a usabilidade do sistema. O objetivo é controlar a conformidade da interface do sistema. As avaliações de teste de usabilidade de sistemas interativos são geralmente constituídas de um conjunto de tarefas que os usuários devem executar a fim de resolver problemas.

Com as tarefas apropriadas para avaliação, pode-se mensurar a facilidade de utilização através de variáveis objetivas e subjetivas.

De acordo com a norma ISO 9241-11, depois da identificação dos objetivos, deve-se determinar os resultados desejáveis, as zonas aceitáveis para definir um nível de satisfação. Existem cinco parâmetros que devem ser considerados para avaliar a usabilidade:

- Fácil de aprender, o usuário pode interagir rapidamente com o sistema, o aprendizado das funcionalidades e dos botões de navegação.
- Eficaz de usar, uma vez que o usuário compreende como o sistema funciona, ele é capaz de encontrar a informação que ele necessita.
- Fácil de reter, mesmo por um usuário que não utiliza o sistema diariamente, ele é capaz de lembrar as principais funcionalidades do sistema.
- Poucas chances de erro, os usuários não comentem muitos erros e são igualmente capazes de corrigir um erro.
- Agradável de usar, os usuários se sentem satisfeitos com o sistema, da forma que eles interagem.

4.9. Considerações Finais

Este texto apresentou uma panorâmica de aspectos relevantes do desenvolvimento de aplicativos independentes de plataforma para dispositivos móveis, baseado em tecnologias Web. Ele tratou de diversos projetos e padrões que ainda estão em formação, tentando capturar os elementos fundamentais de cada um deles.

O Javacript já foi duramente criticado nos últimos anos em razão de problemas de desempenho, devido à sua natureza interpretada e dificuldades na fase de desenvolvimento para a equipe de programação. Atualmente, com a chegada dos navegadores modernos e o avanço das plataformas móveis, observou-se uma grande melhora no tempo de execução, em razão de otimizações, compilação e outras técnicas utilizadas.

Particularmente no desenvolvimento de aplicativos de alto desempenho como jogos, uma das mais importantes variáveis dependentes do desempenho é conhecida como FPS (*Frames per Second*). Como jogos exibem animações, o controle de tempo é necessário para suavizar as animações apresentadas, por isso o controle do tempo (ou de quantos quadros é possível desenhar por segundo) é muito importante para o sucesso de uma implementação. O HTML5 trouxe, portanto, um método – que ainda está em fase de uniformização entre os navegadores – que permite ao desenvolvedor determinar tarefas a serem executadas, toda vez que o navegador está pronto para atualizar seu conteúdo, ou seja, o navegador toma a responsabilidade pelo controle de FPS.

Ainda por se tratar de uma linguagem interpretada, o JavaScript apresenta alguns problemas de segurança, uma vez que permite ataques de *injection*, que podem alterar completamente o funcionamento do aplicativo, abrindo brechas para invasão em aplicações que estão sendo executadas na plataforma do usuário final, dificultando portanto a venda de soluções nesta plataforma.

Outro ponto importante acerca do JavaScript está no fato de que, apesar de ser uma especificação padronizada, a implementação desta é dependente de cada navegador, e os desenvolvedores vivenciam dificuldades diversas. Isso torna difícil a tarefa de criar um código único e 100% funcional independente de plataforma. Adicionalmente, observam-se inconsistências no resultado produzido entre diferentes navegadores, ou seja, dois navegadores diferentes executam o mesmo código, porém o efeito observado diverge de alguma forma.

Este problema motivou o desenvolvimento de diversos frameworks que visam homogeneizar variações entre navegadores e simplificar o trabalho de equipes de programação, uma vez que oferecem artefatos – por exemplo, classes e métodos – que resolvem problemas recorrentes para desenvolvedores com essas tecnologias.

Agradecimentos

Este trabalho foi parcialmente financiado pela FAPESP, o Microsoft Research FAPESP Virtual Institute (projeto NavScales), CNPq (projeto MuZOO) e PRONEX-FAPESP, INCT in Web Science (CNPq 557.128/2009-9) e CAPES, bem como financiamento individual do CNPq.

Ilustrações de cabeças de dinossauros, usadas nas figuras, cedidas sob licença Creative Commons Attribution-Noncommercial-No Derivative por *highdarktemplar (http://highdarktemplar.deviantart.com/).

Referências

Adeagbo, M. (2013). Primer. Retrieved from http://youtu.be/wHlyLEPtL9o.

Adida, B., & Birbeck, M. (2008). RDFa Primer. online: www.w3.org/TR/2008/NOTE-xhtml-rdfa-primer-20081014/

Adida, B., Birbeck, M., & Pemberton, S. (2011). HTML+RDFa 1.1 -- W3C Working Draft 13 January 2011. online: http://www.w3.org/TR/2011/WD-rdfa-in-html-20110113/

Atkins Jr., T., Etemad, E. J., Mogilevsky, A., Baron, L. D., Deakin, N., Hickson, I., & Hyatt, D. (2012). CSS Flexible Box Layout Module - W3C Candidate Recommendation, 18 September 2012. online: http://www.w3.org/TR/css3-flexbox/

Berjon, R., Leithead, T., Navara, E. D., O'Connor, E., Pfeiffer, S., & Hickson, I. (2012). A vocabulary and associated APIs for HTML and XHTML - W3C Candidate Recommendation 17 December 2012. online: http://www.w3.org/TR/2012/CR-html5-20121217/

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2008). Extensible Markup Language (XML) 1.0 (Fifth Edition) -- W3C Recommendation 26 November 2008.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M., Szyperski, C. (1998). What characterizes a (software) component? Software -- Concepts & Tools, 19(1), 49–56.

Cooney, D., & Glazkov, D. (2013). Introduction to Web Components - W3C Editor's Draft 13 April 2013. online: https://dvcs.w3.org/hg/webcomponents/raw-file/tip/explainer/index.html

Cooney, D. (2013). Shadow DOM 101. HTML5 ROCKS. online: http://www.html5rocks.com/en/tutorials/webcomponents/shadowdom/

Dahlström, E., Dengler, P., Grasso, A., Lilley, C., McCormack, C., Schepers, D., Watt, J., et al. (2011). Scalable Vector Graphics (SVG) 1.1 (Second Edition) - W3C Recommendation 16 August 2011. online: http://www.w3.org/TR/2011/REC-SVG11-20110816/

Ecma International (2011). ECMAScript Language Specification - Standard ECMA-262 (5.1 ed.).

Fowler, M. (2012). Developing Software for Multiple Mobile Devices. online: http://martinfowler.com/articles/multiMobile/

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design patterns: elements of reusable object-oriented software. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Glazkov, D. (2011). What the Heck is Shadow DOM? online: http://glazkov.com/2011/01/14/what-the-heck-is-shadow-dom/

Hickson, I. (2011). HTML Microdata -- W3C Working Draft 13 January 2011. W3C. Retrieved from http://www.w3.org/TR/2011/WD-microdata-20110113/

ISO 9241-210 (2010). Ergonomics of human-system interaction -- Part 210: Human-centered design for interactive systems. (Geneva: International Standards Organization).

ISO 9241-151 (2008). Ergonomics of human-system interaction -- Part 151: Guidance on World Wide Web user interfaces. (Geneva: International Standards Organization).

Kelly, H. (2013). "Facebook mobile users surpass desktop users for first time". CNN. Online: http://edition.cnn.com/2013/01/30/tech/social-media/facebook-mobile-users

Kesteren, A. van, Gregor, A., Hunt, L., & Ms2ger. (2012). DOM4 - W3C Working Draft 6 December 2012. online: http://www.w3.org/TR/2012/WD-dom-20121206/

Kesteren, A. van, & Hunt, L. (2013). Selectors API Level 1 - W3C Recommendation 21 February 2013. Retrieved from http://www.w3.org/TR/2013/REC-selectors-api-20130221/

Khare, R. (2006). Microformats: The Next (Small) Thing on the Semantic Web - IEEE Internet Computing, 10(1), 68–75.

Lie, H. W., Çelik, T., Glazman, D., & Kesteren, A. van. (2012). Media Queries - W3C Recommendation 19 June 2012.

Manola, F., & Miller, E. (2004). RDF Primer -- W3C Recommendation. Retrieved from http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

Marcotte, E. (2010, May 25). Responsive Web Design. online: http://alistapart.com/article/responsive-web-design.

Santanchè, A., & Panaggio, R. (2013). Desenvolvimento Independente de Plataforma para Dispositivos Móveis usando Tecnologias Web. In Atualizações em Informática 2013 (pp. 74–139). Maceió: Edufal.

Snook, Jonathan. (2013). SMACSS. . Retrieved August 27, 2013, from http://smacss.com/

Sellier, A. (2013). Leaner CSS ("LESS"). Retrieved August 27, 2013, from http://lesscss.org/

Sullivan, N. (2011). Object Oriented CSS. Retrieved August 26, 2013, from https://github.com/stubbornella/oocss/wiki

W3C SPARQL Working Group (2013). SPARQL 1.1 Overview - W3C Recommendation 21 March 2013. Retrieved from http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/

Yandex. (2013). BEM. Retrieved August 27, 2013, from http://bem.info/

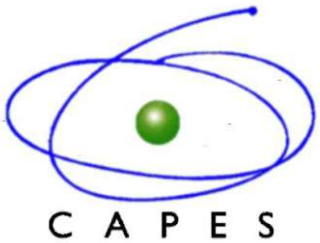
Promoção

Sociedade Brasileira de Computação

Organização

UFBA

Patrocínio



Fundação de Amparo
à Pesquisa do Estado da Bahia

SECRETARIA DE CIÊNCIA,
TECNOLOGIA E INOVAÇÃO

Núcleo de Informação
e Coordenação do
Ponto BR

Comitê Gestor da
Internet no Brasil