

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



FEUP

Conversão assistida de código específico a um *browser* para outros *browsers*

Tiago Miguel dos Reis Órfão

Relatório de Projecto/Dissertação

Mestrado Integrado em Engenharia Informática e Computação

Orientador: João Manuel de Paiva Cardoso (Professor Associado da Faculdade de Engenharia da Universidade do Porto)

29 de Junho de 2009

Conversão assistida de código específico a um *browser* para outros *browsers*

Tiago Miguel dos Reis Órfão

Relatório de Projecto/Dissertação

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo júri:

Presidente: Pedro Alexandre Guimarães Lobo Ferreira do Souto (Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto)

Arguente: João Alexandre Baptista Vieira Saraiva (Professor Auxiliar da Universidade do Minho)

Vogal: João Manuel de Paiva Cardoso (Professor Associado da Faculdade de Engenharia da Universidade do Porto)

16 de Julho de 2009

Resumo

As diferenças entre os vários navegadores *web* disponíveis no mercado são um entrave para a exploração universal do conteúdo de diversas páginas. O Internet Explorer em particular, é um navegador que promove funcionalidades bastante diversas dos outros e, sendo o mais popular de todos, prejudica o funcionamento geral da World Wide Web. Nesta tese, procura-se encontrar uma forma de ultrapassar parte desse problema, focando as funcionalidades que fazem com que uma aplicação, a “Coolbiz - BackOffice”, seja incompatível com outros navegadores.

O trabalho desenvolvido nesta tese inclui um estudo sobre a conversão de código JavaScript específico do IE para outros navegadores, com foco especial no Firefox, e o desenvolvimento de uma estratégia para tornar HTML Components, tecnologia exclusiva do IE, compatíveis com outros navegadores.

Esse conhecimento adquirido foi utilizado na criação de dois protótipos de conversores automáticos, recorrendo a tecnologias como ANTLR, Majestic-12 HTML Parser e a linguagem TXL. Estes protótipos obtiveram resultados satisfatórios para algumas aplicações de teste.

Abstract

The differences between the multiple web browsers available on the market are a drawback to the universal exploration of the content of various pages. Internet Explorer in particular, is a browser that promotes functionalities far different from the others and, as the most popular browser in the world, it cripples the global operability of the World Wide Web. In this thesis, the aim is to find a way to overcome part of this problem, giving focus to the functionalities that make one application, “Coolbiz - BackOffice”, incompatible with other browsers.

The work developed in this thesis includes a study about the refactoring of IE-specific JavaScript code to other web browsers, with special focus on Firefox, and the development of a strategy to make HTML Components, a technology exclusive to IE, compatible with other browsers.

The acquired knowledge was used in the creation of two prototypes of automated translators, relying on technologies such as ANTLR, Majestic-12 HTML Parser and the TXL language. These prototypes obtained satisfying results for a number of test applications.

Agradecimentos

Gostaria de agradecer, antes de mais, a oportunidade proporcionada pela ParadigmaXis e Coollink para a realização desta dissertação, em particular aos engenheiros Alexandre Valente Sousa e Hugo Pacheco.

Agradeço também em especial ao Professor João Paiva Cardoso por ter aceite o meu convite para orientar esta tese, pelas ferramentas que me indicou, bem como pelas revisões que fez do meu trabalho. A todos os professores do MIEIC que, ao longo do curso, contribuíram para o crescimento das minhas competências na área da engenharia informática, muito obrigado.

Gostaria de fazer uma referência aos muitos colegas do Mestrado Integrado em Engenharia Informática e Computação que mantiveram o contacto entre si e partilharam experiências e ideias durante a realização dos seus projectos e dissertações de final de curso, contribuindo para a motivação uns dos outros.

À Coollink e ao Banco Carregosa, agradeço por terem fornecido, durante o período de realização desta tese, todas as condições de que precisei para desenvolver o trabalho, a nível de material, instalações e ferramentas de desenvolvimento.

Tiago Órfão

*“It is a mistake to think you can solve
any major problems just with potatoes.”*

Douglas Adams

Conteúdo

1	Introdução	1
1.1	Motivação e Objectivos	2
1.2	Contribuições da Tese	3
1.3	Estrutura da Dissertação	4
2	Estado da Arte	6
2.1	Introdução	6
2.2	Conversão Automática	6
2.2.1	HTML Tidy	6
2.2.2	PURE	8
2.2.3	ScriptConverter - VBScript to Javascript converter	9
2.3	Extensões e <i>wrapper</i> para Firefox	10
2.3.1	IE View	11
2.3.2	IE Tab	11
2.3.3	Moz Behaviors	11
2.4	Resumo	13
3	Revisão Tecnológica	14
3.1	<i>Browsers</i>	14
3.1.1	Internet Explorer	14
3.1.2	Mozilla Firefox	16
3.1.3	Safari	16
3.1.4	Outros	17
3.2	XSLT	17
3.3	Geradores de <i>parser</i>	18
3.3.1	GNU Bison	19
3.3.2	ANTLR	19
3.3.3	JavaCC	19
3.4	Majestic-12 HTML parser	20
3.5	Sistemas de Transformação de Programas	21
3.5.1	TXL	21
3.5.2	Stratego/XT	23
3.5.3	DMS Software Reengineering Toolkit	25
3.5.4	Tom	26
3.6	Conclusões	28

CONTEÚDO

4	Conversão de Código	29
4.1	Factores que geram incompatibilidade	29
4.1.1	Script	29
4.1.2	HTML <i>Components</i>	30
4.1.3	Cascading Style Sheets	33
4.2	Exemplos de conversão manual	34
4.2.1	Hilite	34
4.2.2	Calendar	36
4.3	Catálogo de Conversão	38
4.4	Estratégia de Conversão	41
4.5	Resumo e Conclusões	42
5	Implementação	46
5.1	Conversão de JavaScript	46
5.1.1	Fase A	47
5.1.2	Fase B	49
5.2	Tratamento de HTC	53
5.3	Primeira Versão	54
5.4	Segunda Versão	55
5.5	Resumo	57
6	Avaliação do Protótipo	59
6.1	Resultados	59
6.1.1	Hilite	59
6.1.2	Calendar	60
6.1.3	Coolbiz	63
6.2	Estatísticas	66
7	Conclusões e Trabalho Futuro	68
7.1	Satisfação dos Objectivos	68
7.2	Trabalho Futuro	69
	Referências	75
A	Hilite	76
A.1	hilite.htm	76
A.2	hilite.htc	76
A.3	Código após conversão automática	77
B	Calendar	78
B.1	calendar.html	78
B.2	calendar.htc	78
B.3	day.htc	82
B.4	today.htc	82
B.5	Código após conversão automática apenas	83
B.6	Código após conversão automática e intervenção manual	88

Lista de Figuras

1.1	Esquema de funcionamento previsto do protótipo	4
2.1	Esquema de funcionamento do sistema PURE	9
2.2	Demonstração do conversor de VBScript para JavaScript.	10
2.3	Firefox a correr Windows Update com IE Tab	12
3.1	Esquema duma transformação XSL	17
3.2	Janela de edição do ANTLRWorks	20
3.3	Estrutura duma transformação Stratego/XT.	24
4.1	Esquema da estratégia de conversão adoptada	45
5.1	Esquema de conversão de JavaScript.	47
5.2	Esquema das etapas ocorridas durante a Fase A.	49
5.3	Esquema das etapas ocorridas durante a Fase B.	50
5.4	Primeira versão da ferramenta aplicada ao exemplo Hilite.	54
6.1	Comparação da apresentação de Hilite antes e após a conversão em Firefox	60
6.2	Calendar convertido sem intervenção manual	62
6.3	Calendar convertido após intervenção manual	63
6.4	Menu do “Coolbiz - BackOffice” em IE	64
6.5	Menu do “Coolbiz - BackOffice” em Firefox sem conversão	65
6.6	Menu do “Coolbiz - BackOffice” em Firefox após conversão	66

Lista de Tabelas

3.1	Tabela de comparação de geradores de parser	18
4.1	Tabela de comparação da aplicação Calendar antes e depois de convertida	39
4.2	Conversão de propriedades JavaScript	40
4.3	Conversão de colecções JavaScript	41
4.4	Conversão de objectos JavaScript	42
4.5	Conversão de métodos JavaScript	43
4.6	Conversão de propriedades CSS	44
4.7	Conversão de eventos	44
4.8	Conversão de contexto	44
6.1	Tabela de características de HTC em “Coolbiz - Backoffice”	64
6.2	Tabela de estatísticas sobre as conversões	67

Abreviaturas

AJAX	Asynchronous JavaScript and XML
ANSI	American National Standards Institute
ANTLR	ANOther Tool for Language Recognition
API	Application Programming Interface
ASP	Active Server Pages
AST	Abstract Syntax Tree
CSS	Cascading Style Sheets
CTT	Code Transformation Tool
DHTML	Dynamic HTML
DOM	Document Object Model
DMS	Design Maintenance System
FEUP	Faculdade de Engenharia da Universidade do Porto
GLR	Generalized Left-to-right Rightmost derivation
GNU	GNU's Not Unix
HTC	HTML Components
HTML	HyperText Markup Language
IE	Internet Explorer
IIS	Internet Information Services
INESC	Instituto de Engenharia de Sistemas e Computadores
ISAPI	Internet Server Application Programming Interface
JavaCC	Java Compiler Compiler
LALR	LookAhead Left-to-right Rightmost derivation
MS	MicroSoft Corporation
MSFT	MicroSoFT Corporation
PURE	Page clean-Up through Reverse Engineering
SGML	Standard Generalized Markup Language
SLOC	Source Lines Of Code
W3C	World Wide Web Consortium
WWW	<i>World Wide Web</i>
XBL	XML Binding Language
XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language Transformations
XUL	XML User Interface Language
YACC	Yet Another Compiler Compiler

Capítulo 1

Introdução

Uma das principais dificuldades com que se deparam os programadores de aplicações *web* é a fraca interoperabilidade entre alguns dos navegadores mais populares a nível mundial. Este problema tem vindo a ser resolvido com a criação e adopção de normas abertas para o desenvolvimento e interpretação de conteúdos para a *web*. No entanto, continuam bem presentes os problemas resultantes da definição de dialectos não padronizados e tecnologias proprietárias exclusivas a determinadas aplicações.

O Windows Internet Explorer (IE) [1], desenvolvido pela Microsoft Corporation, é o *browser* mais popular em todo o mundo, gozando duma quota de mercado situada perto dos 70%, de acordo com os dados da Net Applications, relativos a Fevereiro de 2009 [2]. Sendo um dos navegadores que protagonizou a chamada “guerra dos *browsers*”, juntamente com o Netscape Navigator [3], o Internet Explorer não evoluiu como outros navegadores actuais, que têm no cumprimento das normas abertas estabelecidas por organizações como o W3C a seu principal fundação. Tendo aparecido numa fase anterior a esses *standards* e em que a definição do próprio HTML, bem como de tecnologias associadas ao desenvolvimento de páginas dinâmicas, estava no centro da competição entre *browsers*, o IE não só se tornou num *browser* realista, procurando adaptar-se ao que já existia, como acabou ele próprio por definir a norma do desenvolvimento *web* com tecnologias proprietárias da Microsoft, depois de ter vencido a sua batalha contra o Navigator da Netscape.

O domínio do IE acabou por levar muitos programadores a desenvolver as suas páginas sem a preocupação de as tornar compatíveis com múltiplos navegadores, mesmo depois de já estarem definidas algumas normas abertas sobre esse mesmo desenvolvimento. A pouco e pouco, no entanto, essas normas foram sendo adoptadas por muitos, e a crescente popularidade de navegadores como Mozilla Firefox [4] e de sistemas operativos alternativos que não incluem o IE, como Mac OS X e distribuições GNU/Linux, bem

como o crescimento de popularidade de vários dispositivos com acesso à Internet, como telemóveis e PDAs, que não são baseados em sistemas Microsoft, tornam a procura de interoperabilidade entre browsers cada vez mais relevante.

O objectivo deste trabalho é estudar a possibilidade de converter de forma assistida páginas desenvolvidas especificamente para o IE, tornando-as compatíveis com outros *browsers*, tendo em especial atenção o Mozilla Firefox, principal concorrente do IE na actualidade.

Esta dissertação foi proposta pela ParadigmaXis - Arquitectura e Engenharia de Software, S.A. [5], empresa criada no ano 2000 por ex-colaboradores do INESC Porto [6] e que desenvolve vários projectos na área da informática.

1.1 Motivação e Objectivos

Muitas organizações desenvolveram aplicações essenciais ao seu modelo de negócio numa fase em que o Internet Explorer tinha uma preponderância maior no mercado, e tiraram partido daquilo que a Microsoft tinha para oferecer sem terem preocupações de compatibilidade. Adicionalmente, muitas dessas aplicações eram para uso interno, onde o acesso a tecnologias Microsoft era uma certeza e simplesmente não valia a pena ter essa preocupação.

No cenário actual, no entanto, muitas empresas podem questionar as suas parcerias com a Microsoft ou, simplesmente, pretender expandir o número de potenciais clientes chegando ao maior número de utilizadores possível. Inclusivamente, alguns sítios de concursos públicos só são compatíveis com Internet Explorer, o que pode significar uma desvantagem competitiva para muitos que utilizem outro tipo de tecnologias. Este tipo de situações acontecia até há bem pouco tempo em Portugal [7].

Uma empresa ligada à ParadigmaXis, a CoolLink Serviços Informáticos e de Consultadoria Lda. [8], responsável pelo desenvolvimento de soluções informáticas para o Banco L. J. Carregosa, S. A. [9], criou uma aplicação designada “Coolbiz - Backoffice”, que serve de suporte ao trabalho de empresas que comercializem produtos financeiros ou que funcionem como bancos. Esta aplicação foi desenvolvida com recurso a tecnologias proprietárias da Microsoft e foi projectada para funcionar apenas no Internet Explorer.

A necessidade de expandir o negócio e a crescente utilização de navegadores e sistemas operativos alternativos faz com que a adopção de tecnologias *standard* se torne uma necessidade para a Coollink. Infelizmente, fazer a conversão manual duma aplicação tão complexa e tão dependente de tecnologias proprietárias pode ser um processo bastante moroso e dispendioso de recursos, pois, dependendo da complexidade do problema, pode quase implicar reescrever praticamente toda a aplicação.

Nesta dissertação pretende-se desenvolver uma solução que permita não só ajudar a Coollink a tornar a sua aplicação compatível com outros navegadores, mas também

fornecer uma solução mais global para todas as aplicações que sofram de problemas semelhantes de interoperabilidade entre *browsers*, causados por um desenvolvimento específico para o Internet Explorer. De notar que o Mozilla Firefox é o alvo principal dessa conversão, por ser o navegador mais popular a seguir ao Internet Explorer e estar disponível em várias plataformas. No entanto, idealmente, a solução deverá ser aplicável a qualquer programa ou, pelo menos, a outros navegadores populares como o Safari da Apple.

Não é obrigatório que a conversão seja completa, pois muitas funcionalidades podem nem ser exequíveis noutros *browsers*, deve no entanto ser completa o suficiente para facilitar em muito o trabalho de conversão manual, poupando cerca de 90% do esforço que essa tarefa teria originalmente.

Para atingir esse objectivo, procurou-se identificar o tipo de tecnologias utilizadas vulgarmente que geram dificuldades na compreensão do código HTML com CSS e JavaScript por parte de outras aplicações. Nesse sentido, procedeu-se à análise de códigos fonte, ou excertos de códigos, que apresentam as incompatibilidades que se pretende resolver.

De seguida, tratou-se de encontrar ou criar as respectivas alternativas que são compreendidas por outros browsers para os problemas identificados anteriormente. Nessa fase, foram convertidas manualmente algumas aplicações simples ou excertos de páginas mais complexas que só funcionavam correctamente no popular navegador da Microsoft.

Terminada essa fase, ganha relevância o estudo sobre a melhor maneira de fazer a conversão, as ferramentas que devem ser utilizadas para tal, se é necessário intervir noutras fases do desenvolvimento, e como aplicar a conversão.

1.2 Contribuições da Tese

O conhecimento adquirido nesta tese servirá para utilização na aplicação “Coolbiz - BackOffice” da forma que parecer mais indicada. Para efeitos de teste e prova de conceito, o protótipo do conversor automático será aplicado durante a execução da aplicação no servidor. Assim, a conversão ocorre aquando do envio do código por parte do servidor para o cliente. A figura 1.1 mostra os passos que ocorrem durante essa conversão em *runtime*.

Uma vez que parte da solução passará por criar um conversor de JavaScript específico do Internet Explorer para se tornar funcional noutros *browsers*, deverá ser possível também, utilizar parte do conteúdo desta tese para converter esses *scripts* em qualquer página que perca funcionalidade noutros browsers. Para que isto possa ser possível de forma clara, a implementação desta tarefa deverá estar bem identificada ou até separada da solução mais global.

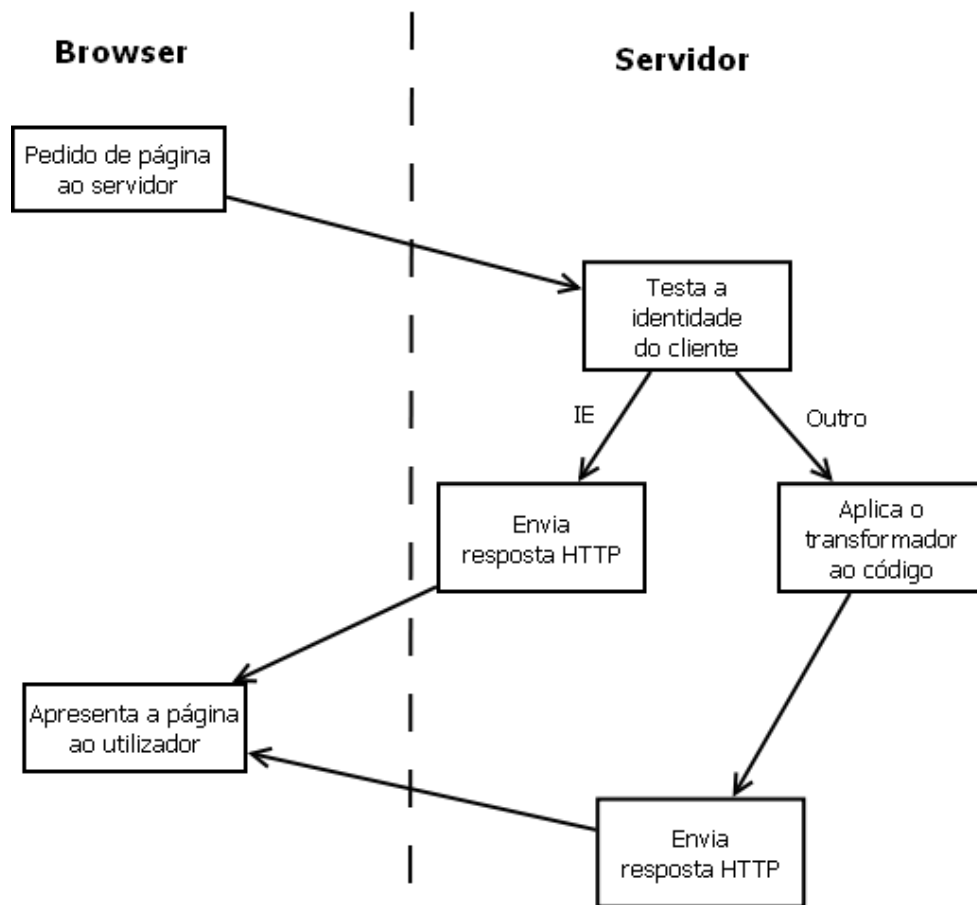


Figura 1.1: Esquema de funcionamento previsto do protótipo

Espera-se também que esta tese contribua para a criação duma solução global que auxilie o processo de conversão de um sítio web que recorra a componentes HTC e a JScript, como é o caso da aplicação Coolbiz.

1.3 Estrutura da Dissertação

Para além desta mesma introdução, a dissertação consiste em cinco capítulos. No capítulo 2, é descrito o estado da arte e são apresentadas soluções relacionadas com a temática deste trabalho. No capítulo 3 são apresentadas tecnologias que poderão ser utilizadas ou auxiliar na criação do conversor de código. No capítulo 4, é apresentado o estudo efectuado, sendo indicados alguns conselhos em relação ao desenvolvimento de páginas compatíveis com múltiplos *browsers* e alternativas a implementações proprietárias da Microsoft. No capítulo 5, é descrita a implementação do conversor automático, incluindo algumas das decisões tomadas nessa conversão, as gramáticas e ferramentas utilizadas, e os resultados obtidos por esse conversor. No capítulo 6, é testada

Introdução

a qualidade da solução e apresentadas estatísticas sobre as conversões. No capítulo [7](#), enumeram-se as conclusões do trabalho efectuado e descreve-se os futuros desenvolvimentos que este poderá ter.

Capítulo 2

Estado da Arte

Neste capítulo descreve-se o estado da arte e apresentam-se alguns projectos que estão relacionados com o assunto a desenvolver nesta dissertação.

2.1 Introdução

A pobre compatibilidade entre vários *browsers* afecta muitas páginas *Web*, em especial páginas que promovem uma interacção dinâmica, mas não só. Como tal, já existem algumas maneiras de contornar o problema. Estas soluções, no entanto, são pouco completas em termos de conversão, normalmente aplicando-se apenas a uma porção específica do que constitui uma aplicação *Web*.

Por outro lado, existem soluções que pecam por não serem aplicáveis a vários *browsers*, existindo mesmo soluções que não eliminam a dependência das tecnologias Microsoft.

2.2 Conversão Automática

Não foram encontradas soluções que tenham o mesmo objectivo da que se pretende desenvolver ao longo deste trabalho. Existem, no entanto, algumas soluções de conversão que seriam parcialmente úteis em tarefas relacionadas com a interoperabilidade entre navegadores. De seguida apresentam-se tais ferramentas.

2.2.1 HTML Tidy

Existe uma aplicação, denominada HTML Tidy [10], que foi desenvolvida no seio do W3C, que pretende converter automaticamente páginas legadas de forma a que se tornem válidas de acordo com os *standards* definidos. Para isso, verifica o código HTML e tenta corrigi-lo de forma a estar correctamente formatado.

Algumas das correcções que o Tidy implementa em HTML são:

- Etiquetas que faltam ou sem correspondência são detectadas e corrigidas.

```
<h1>heading  
<h2>subheading</h3>
```

torna-se em:

```
<h1>heading</h1>  
<h2>subheading</h2>
```

- Fecho de etiquetas pela ordem errada são corrigidos.

```
<p>here is a para <b>bold <i>bold italic</b> bold?</i> normal?
```

torna-se em:

```
<p>here is a para <b>bold <i>bold italic</i> bold?</b> normal?
```

- Corrige problemas com o uso de itálico em títulos.

```
<h1><i>italic heading</h1>  
<p>new paragraph
```

torna-se em:

```
<h1><i>italic heading</i></h1>  
<p>new paragraph
```

- Recupera de etiquetas misturadas.

```
<i><h1>heading</h1></i>  
<p>new paragraph <b>bold text  
<p>some more bold text
```

torna-se em:

```
<h1><i>heading</i></h1>  
<p>new paragraph <b>bold text</b>  
<p><b>some more bold text</b>
```

- Adiciona o carácter ‘/’ no fecho de etiquetas mal formatadas.
- Adiciona aspas à volta dos valores dos atributos.
- Alerta para o uso de elementos não incluídos na definição de HTML.
- Adiciona automaticamente o DOCTYPE, detectando automaticamente a versão de HTML que está a ser usada.

Aconselha-se o leitor a consultar [10, Examples of TIDY at work] para uma descrição mais aprofundada acerca do Tidy.

Este tipo de alterações permitem eliminar algumas ambiguidades que fazem com que certos browsers interpretem uma página de maneira diferente de outros, transformando o código de forma a que fique mais próximo de ser um ficheiro XHTML com uma estrutura XML válida.

2.2.2 PURE

A aproximação feita pelo HTML Tidy não é perfeita, uma vez que ao transformar o código para se tornar mais facilmente interpretável por múltiplos *browsers*, acaba por provocar o efeito oposto, muitas vezes alterando a forma como a página é apresentada completamente [11, chap. Introduction].

Em 2006, foi apresentado o protótipo duma aplicação, a PURE (Page clean-Up through Reverse Engineering) [11], que com um algoritmo baseado em engenharia inversa, tenta gerar código bem formatado, ao mesmo tempo garantindo que a página permanece com o estilo que o autor da mesma pretendia.

Para conseguir este objectivo, o programa utiliza um navegador popular para fazer a renderização da página a processar. Este passo chama-se de pré-processamento. Depois, examina o layout da página apresentada e divide a estrutura da página em múltiplos rectângulos, guardando os dados sobre a posição e tamanho dessas “caixas” que retira da árvore DOM gerada pelo browser. De seguida, reconstrói o layout da página utilizando o modelo de caixas de CSS, efectivamente ficando com uma página sem conteúdo mas dividida em vários elementos “div” posicionados de forma absoluta. O código fica então com a seguinte estrutura:

```
<BODY>
<DIV id="FirstBox"> ... </DIV>
<DIV id="SecondBox"> ... </DIV>
.....
<DIV id="LastBox"> ... </DIV>
</BODY>
```

e o CSS criado indica as posições e tamanhos respectivos:

```
#FirstBox {position:absolute; top: 10px;
left: 10px; width: 800px; height:
100px;}
#SecondBox {position:absolute; top:
110px; left: 10px; width: 200px; height:
400px;}
....
#LastBox {position:absolute; top: 660px;
```

```
left: 10px; width: 800px; height:
200px; }
```

Finalmente, quando está concluída esta reestruturação, o programa efectua uma conversão do código, colocando-o no elemento respectivo [11].

A figura 2.1, retirada de [11], apresenta os passos deste processo de transformação na forma de esquema.

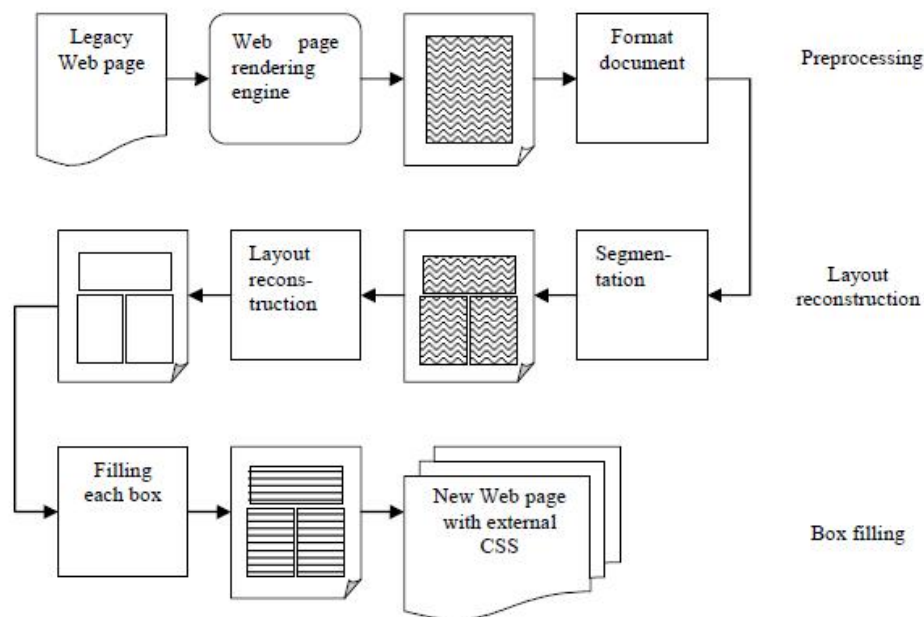


Figura 2.1: Esquema de funcionamento do sistema PURE

Este tipo de abordagem, tal como o HTML Tidy, apenas aborda uma parte do que deveria ser a conversão duma página dinâmica, sendo mais adequada para reestruturar *sites* menos interactivos que recorram sobretudo a HTML, com uma componente de *script* mínima ou inexistente. Por esse motivo, não seria de prever que tivesse quaisquer consequências positivas na conversão de páginas em que a pouca compatibilidade é motivada por um código JavaScript específico, CSS utilizando propriedades apenas reconhecidas por um *browser* e tecnologias também específicas como os HTML Components.

2.2.3 ScriptConverter - VBScript to Javascript converter

Indiscutivelmente, um dos problemas maiores da conversão de páginas de IE para outros *browsers* é a linguagem de *scripting*, a executar do lado do cliente, associada a essa mesma página.

Rob Eberhardt, da Slingshot Solutions [12], disponibiliza um conversor automático de VBScript para JavaScript [13], uma solução que pode ser bastante útil para quem tem um

site que recorra a VBScript e pretenda garantir compatibilidade com outros navegadores que não o IE. A figura 2.2 demonstra o conversor em funcionamento.

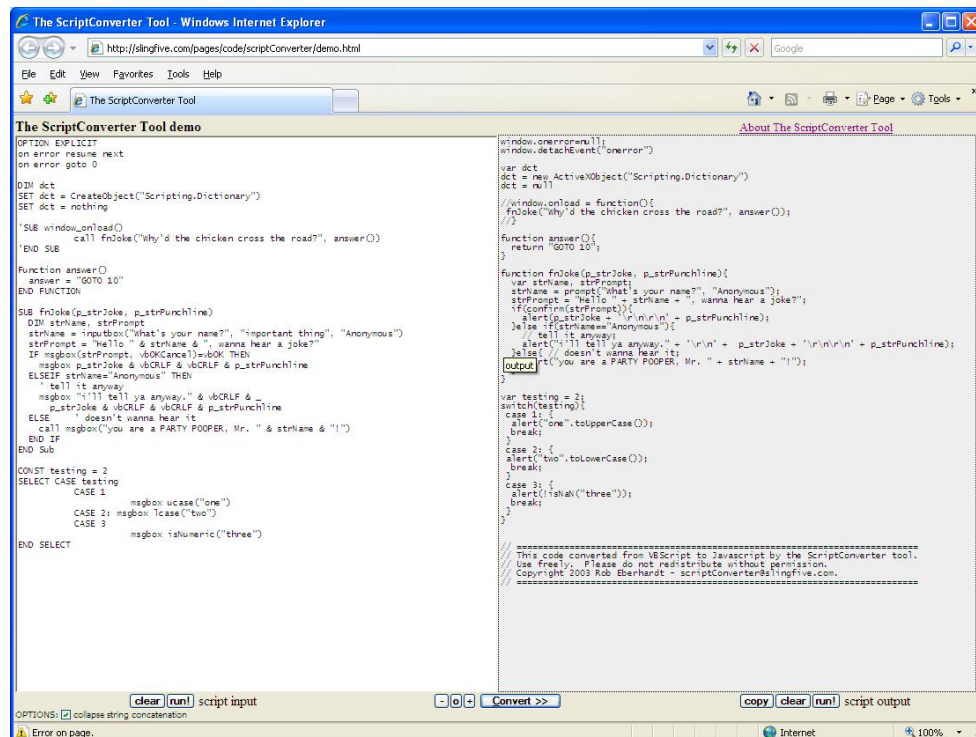


Figura 2.2: Demonstração do conversor de VBScript para JavaScript.

Recorrendo a este conversor é possível passar os *scripts* que são executados na página para JScript/JavaScript. E embora este código não seja garantidamente compatível com todos os *browsers* nem esteja garantidamente convertido a 100%, pois só converte as estruturas mais básicas e mais usadas da linguagem [13, chap. limitations], é pelo menos eliminado um grande passo da tarefa de conversão.

Este conversor no entanto não tem qualquer influência numa conversão de JScript para JavaScript, tarefa necessária para converter uma grande variedade de aplicações dinâmicas que funcionam exclusivamente em IE, seria no entanto uma hipótese a explorar a inclusão desta ferramenta numa solução mais global de conversão de páginas específicas para IE, ficando assim coberta a possibilidade de converter tipos de *script* diferentes com o mesmo conversor.

2.3 Extensões e *wrapper* para Firefox

Apesar das soluções seguintes não estarem relacionadas com tarefas de conversão de código, procuram oferecer soluções para o mesmo problema base que é apresentado nesta tese, sendo por isso relevante a sua apresentação.

Uma vez que o Mozilla Firefox é um *browser* que permite a criação de extensões para a aplicação de forma fácil, essa capacidade tem sido utilizada para eliminar a necessidade de utilizar o Internet Explorer para navegar na *Web*. Exemplos disso são extensões como o IE View e o IE Tab.

A juntar a estas extensões, existe também um *wrapper* que permite alguma compatibilidade entre HTML Components e Firefox.

Um problema claro destas soluções é que só se aplicam ao Firefox e não a qualquer outro *browser*.

2.3.1 IE View

O IE View [14] é uma extensão que permite ao utilizador abrir uma instância do IE dentro do Firefox, utilizando a página actual ou uma ligação para outra. É possível também definir uma lista de páginas que devem ser abertas com o IE por omissão.

Esta solução tem vários inconvenientes. Não nos livra da dependência do IE nem do sistema Microsoft Windows, uma vez que ambos necessitam de estar instalados para a extensão funcionar. Nem sequer permite ao utilizador ver a página dentro do Firefox *per se*, apenas elimina o incómodo de ter de abrir o programa manualmente para ver páginas específicas.

2.3.2 IE Tab

O IE Tab [15] é outra extensão que foi inspirada pela anterior e oferece uma funcionalidade um pouco mais interessante. Para além de permitir também definir as páginas que necessitam do IE para serem visualizadas correctamente, ela permite que as páginas sejam visualizadas dentro do próprio Firefox, mas recorrendo ao motor de *layout* do IE.

Assim, permite uma eliminação completa da interacção do utilizador com o IE. No entanto, não elimina a necessidade de o ter instalado, sofrendo das mesmas fraquezas a esse nível que a extensão anterior. Existem maneiras de utilizar a extensão noutros sistemas operativos, mas a funcionalidade fica seriamente comprometida.

A figura 2.3 demonstra as capacidades desta extensão, conseguindo correr páginas exclusivas da Microsoft em Mozilla Firefox.

2.3.3 Moz Behaviors

O Moz Behaviors [16] é um *wrapper*, ou seja, um adaptador, que permite adicionar a funcionalidade dos HTC ao Firefox, recorrendo para isso à linguagem XBL, referida anteriormente.

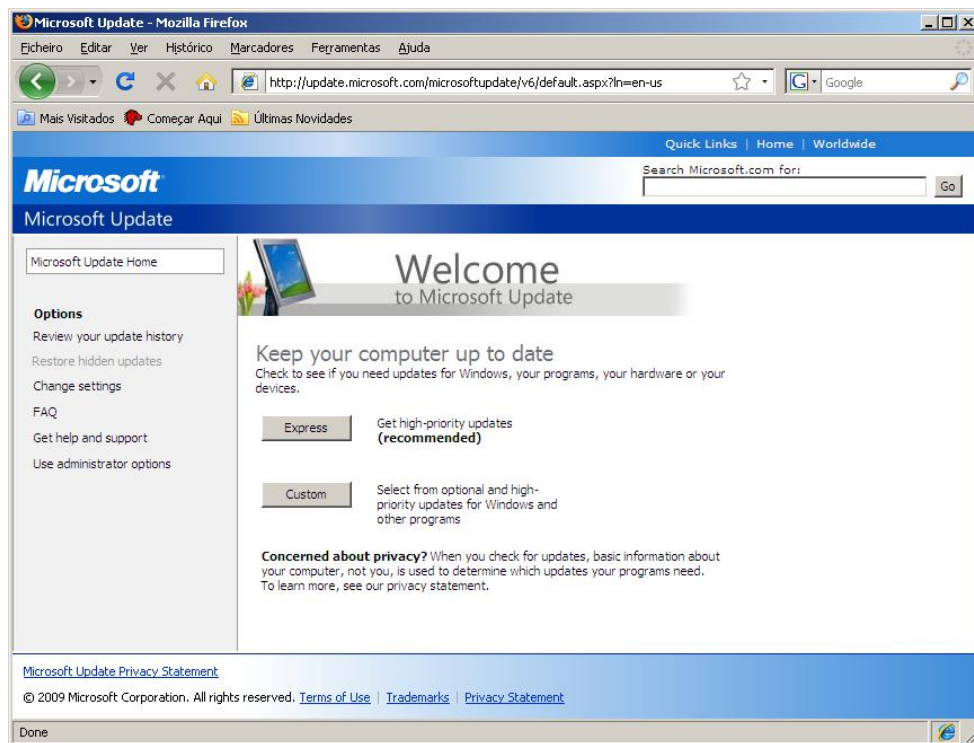


Figura 2.3: Firefox a correr Windows Update com IE Tab

Dean Edwards é o responsável por este projecto que consiste num ficheiro, `moz-behaviors.xml`, que age como “ponte” entre as funcionalidades que o XBL permite e as que são indicadas num ficheiro HTC.

O `moz-behaviors` oferece outro ficheiro, `bindings.xml`, no qual devem ser feitas as referências para os ficheiros HTC que se pretende utilizar de forma compatível com Firefox. A invocação desses comportamentos é feita utilizando a propriedade `-moz-binding` em CSS, muito à semelhança do que é feito com o IE para os invocar, embora nesse caso seja utilizada a propriedade *behavior*.

De seguida, mostram-se exemplos dessa invocação feita em CSS. No primeiro caso é apenas referido um ficheiro HTC.

```
div.drag-box {
    /* reference behaviors in the usual way for explorer */
    behavior: url(drag-box.htc);
    /* reference behaviors like this for mozilla */
    -moz-binding: url(bindings.xml#drag-box.htc);
}
```

No caso seguinte são invocados dois componentes HTC distintos.

```
pre.html {
    /* for explorer */
```

```

behavior: url(/my/star-html.htc) url(/my/star-light.htc);
/* and for mozilla */
-moz-binding: url(/my/bindings.xml#star-html.htc|star-light.htc);
}

```

Esta tecnologia apenas funciona para HTC que sejam aplicados a elementos existentes na definição do HTML, quer por via de folha de estilos, ou por via de script. Não é capaz de interpretar definições de elementos personalizados.

Outra limitação reside no facto de precisar que o script no ficheiro HTC seja interpretado como XML válido. Obrigando para isso a incluir as tags `<![CDATA[ e ]]>` comentadas nos extremos do script. É ainda incapaz de utilizar scripts que não sejam compatíveis com Firefox por si próprios e devido ao facto do projecto estar parado desde 2005 [16, secção Update], não existe garantia da sua compatibilidade com versões mais recentes das tecnologias que lhe estão associadas.

2.4 Resumo

Apesar do problema ser ainda bastante comum, não existe, aparentemente, nenhuma solução de conversão automática de páginas IE para outros navegadores. A maior parte das vezes esta conversão é feita manualmente, com um técnico dedicado a essa tarefa. A vantagem desse tipo de solução é que permite um controlo muito maior sobre a funcionalidade da nossa aplicação do que uma solução automática proporcionaria, bem como uma legibilidade e “limpeza” bem maior do código. Por outro lado, uma conversão manual pode ser particularmente morosa, quando feita sobre uma aplicação muito extensa e com grande complexidade como se vai verificar ser o caso da “Coolbiz - BackOffice”, estando também sujeita à criação de algumas inconsistências a nível de código por acção humana diferente em casos semelhantes.

Mesmo não existindo conversores gerais dedicados à eliminação da dependência do IE, existem trabalhos relacionados com a conversão de código HTML com o intuito de garantir maior compatibilidade entre browsers. Exemplos disso são o HTML Tidy, o PURE e o ScriptConverter. Estas ferramentas oferecem soluções bastante limitadas e que acabam por executar uma tarefa demasiado particular para serem úteis como solução global para o problema focado nesta tese.

Ainda no que diz respeito ao *web developer*, é possível criar alguma compatibilidade entre os recursos HTC que ele utiliza e o Firefox através do Moz Behaviors. Esta solução, no entanto, parece pouco viável para a maior parte dos casos.

Do ponto de vista do cliente, é possível instalar extensões para o Firefox, IE View e IE Tab que permitem uma melhor integração entre as capacidades do Internet Explorer em termos de compatibilidade com determinados sítios Web e o Mozilla Firefox.

Capítulo 3

Revisão Tecnológica

Neste capítulo são apresentadas várias ferramentas que, potencialmente, podem ser utilizadas neste trabalho e serão justificadas as opções tomadas no desenvolvimento futuro do tradutor de código HTML e JavaScript.

O trabalho a realizar consiste numa tradução do tipo *source-to-source*, uma vez que se trata de transformar código em linguagem de alto nível para outro código em linguagens de alto nível. Este código resultante da conversão deverá ser interpretado pelos *browsers*, nem sequer passando por uma fase de compilação previamente.

Existem uma série de tecnologias que podem ter aplicação nesta tarefa, sendo que nesta fase da tese serão descritas algumas dessas ferramentas, depois de feita uma revisão dos navegadores de Internet que dizem respeito a este trabalho.

3.1 *Browsers*

Nesta secção é feita uma pequena descrição dos navegadores de Internet que vão ser relevantes neste projecto, e faz-se uma apresentação de tecnologias específicas que são parcialmente responsáveis pelas incompatibilidades existentes entre eles.

3.1.1 Internet Explorer

O Windows Internet Explorer [1], anteriormente chamado Microsoft Internet Explorer, é o navegador *web* da Microsoft que está na origem do problema apresentado nesta tese. Foi introduzido em 1995 e, ao ser incluído com o sistema operativo Microsoft Windows, ganhou popularidade rapidamente.

Esse aumento de popularidade, que terá atingido o seu pico em 2002 com mais de 95% da quota de mercado dos *web browsers* [17], fez com que o desenvolvimento de páginas e aplicações *web* se centrasse na procura de compatibilidade apenas com esta

aplicação, depois de um período em que os programadores se dividiam entre IE e Netscape Navigator.

Ao longo das várias versões da aplicação, a Microsoft foi introduzindo uma série de tecnologias e funcionalidades exclusivas que, foram aproveitadas por muitos *web developers* no sentido de enriquecer o conteúdo das suas páginas, muitas vezes descurando a procura da compatibilidade com outros *browsers* que tinham uma quota de mercado desprezável.

Alguns exemplos dessas tecnologias são:

- ActiveX Controls [18]

Componentes de software que podem ser integrados em páginas *web*. Apenas funcionam caso o *browser* seja o Internet Explorer e o sistema operativo seja o Windows. Ainda são bastante comuns em páginas que tenham funcionalidades mais complexas como a navegação de mapas. Embora possam ser consideradas comparáveis a Java Applets, não são compatíveis com tantas plataformas e permitem um controlo muito superior sobre o sistema operativo, podendo ser, por isso, a sua utilização bastante perigosa para o cliente [19, chap. ActiveX Controls].

- HTML Components [20]

Usados no contexto do DHTML, os HTML Components fornecem um meio de definir um comportamento que pode ser associado a determinado elemento de uma página HTML. Este elemento pode ser um tipo definido do HTML, mas também pode ser um elemento criado pelo programador, invocado no HTML com o recurso a uma etiqueta personalizada e respectivo *namespace*.

- JScript [21]

Linguagem de *scripting* criada pela Microsoft para copiar as funcionalidades da linguagem JavaScript da Netscape. Por ser semelhante ao JavaScript, e actualmente respeitar o standard ECMAScript pode ser interpretada por outros browsers. No entanto, é comum os seus aspectos específicos tornarem o código exclusivo ao IE.

- VBScript [22]

Linguagem de *scripting* baseada em Visual Basic. É apenas interpretada pelo IE e a sua popularidade no contexto das aplicações *Web* parece ter vindo a diminuir nos últimos anos. Ainda assim, é um problema de compatibilidade para os *sites* que recorrem a esta linguagem.

3.1.2 Mozilla Firefox

O Mozilla Firefox [4] é o segundo browser mais popular no mundo [2] e é a aplicação sobre a qual se vai centrar a conversão de código a desenvolver neste trabalho. Trata-se duma aplicação de código aberto, multi-plataforma e que tem no respeito das normas abertas para o desenvolvimento *web* um dos seus principais fundamentos. Assim, é uma excelente plataforma de testes e também uma boa aplicação para servir como “meta” deste trabalho, pois uma página que funcione no Firefox pode ser acedida por qualquer computador pessoal, sem custos adicionais, independentemente da plataforma que esteja a correr.

Apesar da sua filosofia mais aberta, a Mozilla também adopta no Firefox algumas tecnologias exclusivas, entre as quais merece destaque, pela relevância que tem no contexto do trabalho, a XBL [23]. A XBL (XML Binding Language) oferece uma maneira de descrever comportamentos que podem ser associados a determinado elemento HTML numa página *web*. Assim, tem um objectivo similar a uma das funcionalidade dos HTML Components em IE, mas funciona de maneira bem distinta, pelo que não é trivial fornecer compatibilidade entre uma funcionalidade e outra, embora exista um projecto que consegue isso parcialmente (ver secção 2.3.3).

A versão 2.0 de XBL já é candidata a recomendação W3C para ser uma norma aberta, estando à espera que sejam feitas outras implementações dessa especificação para poder ser considerada para recomendação definitiva [23].

3.1.3 Safari

O Safari [24] é um navegador disponibilizado pela Apple desde 2003. Tornou-se no navegador padrão do sistema operativo Mac OS X, também da Apple, substituindo o Internet Explorer For Mac da Microsoft e o Netscape Navigator, até então incluídos com os sistemas Mac OS [25, chap. Safari].

A recente subida de popularidade dos computadores Macintosh e a chegada do *browser* ao sistema operativo Windows em 2007, tornou o Safari mais relevante neste contexto da procura de compatibilidade, uma vez que se destacou como terceiro browser mais utilizado [2].

Por isso, garantir a compatibilidade com o Safari será um objectivo secundário deste trabalho. E para isso acontecer, será mais interessante optar por uma abordagem que se sirva das normas abertas, ou de funcionalidades comuns aos dois *browsers* em vez de fabricar duas soluções separadas, que certamente iriam ter muito em comum entre si, pois tanto o Safari como o Firefox respeitam em grande parte as normas abertas definidas pelo W3C e outras organizações.

Desta forma, o conversor poderá ser aplicado a qualquer página desenvolvida apenas para o IE, resultando apenas em uma página com código compatível com mais plataformas.

3.1.4 Outros

Existem outros navegadores que gozam de alguma popularidade no mercado dos computadores pessoais e também de dispositivos móveis. Uma vez que se irá optar por criar uma conversão global em vez de específica para Firefox e/ou Safari, espera-se obter alguns resultados positivos também para essas aplicações. Exemplos de outros *browsers* que são utilizados com frequência são: Google Chrome, Opera, Netscape Navigator, Mozilla Web Suite e Opera Mini.

3.2 XSLT

O XSLT (Extensible Stylesheet Language Transformations) [26] é uma linguagem incluída na tecnologia XML para transformar documentos XML. Um cenário típico da sua utilização é receber como *input* um documento XML, aplicar a esse documento as transformações definidas na folha de estilos XSLT e gerar um novo documento, que tanto pode ser outro XML, até num dialecto diferente do original, como pode ser um qualquer documento legível por humanos, como HTML ou texto simples.

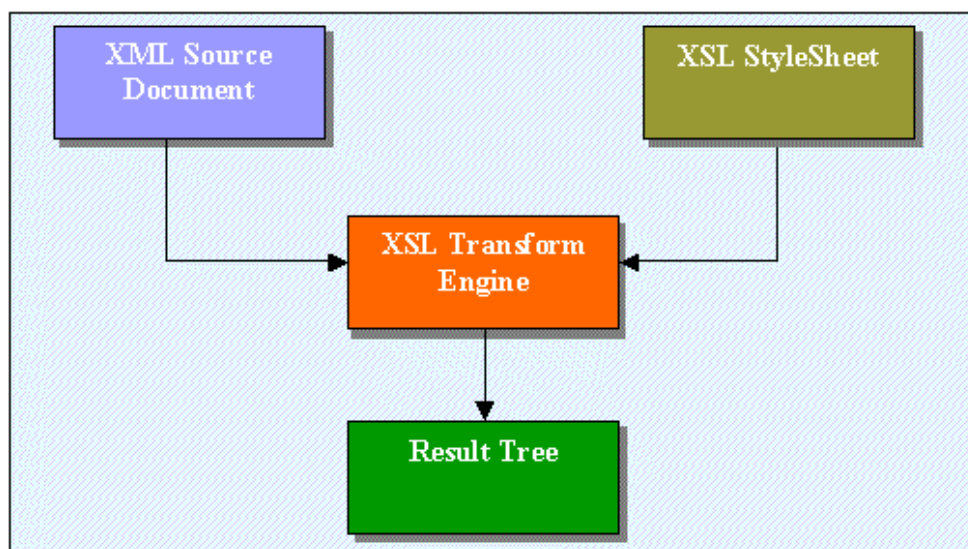


Figura 3.1: Esquema duma transformação XSL

Esta tecnologia poderia ser uma alternativa para transformar páginas *Web*, no entanto o XSLT é apenas aplicável a documentos XML, algo que o HTML não é. O HTML tem origem na metalinguagem SGML, que precede e também está na origem do XML [27].

Para ser possível a utilização do XSLT, a entrada teria de ser uma página web em XHTML, esse sim, um dialecto XML relacionado com o HTML. Mas mesmo nesse caso, são inúmeros os exemplos de páginas XHTML com XML mal estruturado que existem na *Web*. Assim não temos garantia que o XSLT consiga ler correctamente o documento de origem. Adicionalmente, tal como aconteceria com outra ferramenta qualquer para reconhecer o HTML, não era possível processar as partes do documento com código JavaScript e/ou CSS com XSLT. Existiria a necessidade de recorrer a um gerador de analisadores sintácticos de qualquer maneira, pois estas linguagens nada têm em comum com HTML ou XML.

3.3 Geradores de *parser*

Os geradores de *parser* são ferramentas utilizadas na análise sintáctica de linguagens. Através da definição de tokens num analisador léxico próprio ou externo, e da definição da estrutura semântica da linguagem em gramáticas próprias dos geradores, eles são capazes de gerar código passível de ser utilizado para manipular código fonte com diversos objectivos, como a criação de compiladores ou transformação de programas.

Foram considerados para a realização deste trabalho alguns dos geradores de *parser* mais famosos que estão disponíveis livremente.

- GNU Bison [28] - ferramenta clássica de geração de código, é alternativa de código aberto à ferramenta Yacc.
- ANTLR [29, 30] - gerador de parser escrito em Java, capaz de gerar código para várias linguagens.
- JavaCC [31] - gerador de parser escrito em Java e para Java.

A tabela 3.1, adaptada de [32], resume algumas das características dos geradores de *parser* considerados.

Tabela 3.1: Tabela de comparação de geradores de parser

Características das ferramentas				
<i>nome</i>	Algoritmo de <i>parsing</i>	Linguagens de saída	Analisadores léxicos	IDE
GNU Bison	LALR, GLR	C, C++	externo	não
ANTLR	LL(*)	C, C#, Java, Python	gerado	sim
JavaCC	LL(k)	Java	gerado	sim

3.3.1 GNU Bison

O GNU Bison [28] é uma ferramenta que, normalmente, é utilizada em conjunto com o flex (fast lexical analyzer), alternativa ao proprietário Lex, para gerar código de *parser*. Utiliza um algoritmo LALR(1) ou (GLR), ou seja, ao contrário do ANTLR, o analisador produz uma derivação mais à direita. Foi desenvolvido pelo GNU Project como alternativa *open source* ao YACC, sendo que, posteriormente, se tornou completamente compatível com as gramáticas definidas para YACC. É amplamente usado no contexto académico, embora tenha sido gradualmente substituído por outras ferramentas que apresentam uma maior facilidade para trabalho com várias linguagens.

3.3.2 ANTLR

O ANTLR [29] é uma ferramenta muito versátil, com excelente documentação e com muitas gramáticas disponibilizadas na sua página oficial. Utiliza um algoritmo do tipo LL(*) [33]. Isto significa que o analisador sintáctico é descendente, tentando deduzir as produções gramaticais a partir da raiz e lê a entrada de texto da esquerda para a direita (*Left-to-right*) e produz uma derivação mais à esquerda (*Leftmost derivation*) [34, chap. Classes of languages]. O carácter * indica que o analisador é capaz de utilizar qualquer número de tokens ainda não processados para tomar decisões, sendo que no caso do ANTLR é também capaz de realizar *backtracking*.

Existe um IDE próprio, chamado ANTLRWorks (Figura 3.2 [29]) que facilita em muito o processo de desenvolvimento das gramáticas para o ANTLR.

O ANTLR disponibiliza ainda, no seu sítio oficial, um bom número de gramáticas criadas pela comunidade para diversas linguagens, incluindo CSS e JavaScript/ECMAScript.

Desenvolvido como um projecto paralelo ao ANTLR, o StringTemplate [35] é uma biblioteca que permite gerar código fonte de acordo com um template definido. Pode ser uma opção a explorar na geração de código a partir da árvore de sintaxe, embora seja necessário fazer uma especificação do modelo a partir da árvore, o que seria uma tarefa com uma carga de trabalho semelhante à de reescrever uma gramática completa. O retorno em termos de facilidade de manipulação do código nem sempre justifica essa opção.

3.3.3 JavaCC

O JavaCC [31] é um gerador de parser de código aberto muito popular. No entanto, tem a limitação de trabalhar exclusivamente para Java. Ele utiliza uma abordagem LL(k) na geração do seu parser. Fornecido junto com o JavaCC, vem outra ferramenta, a JJTree, que é utilizada para assistir na construção de árvores.

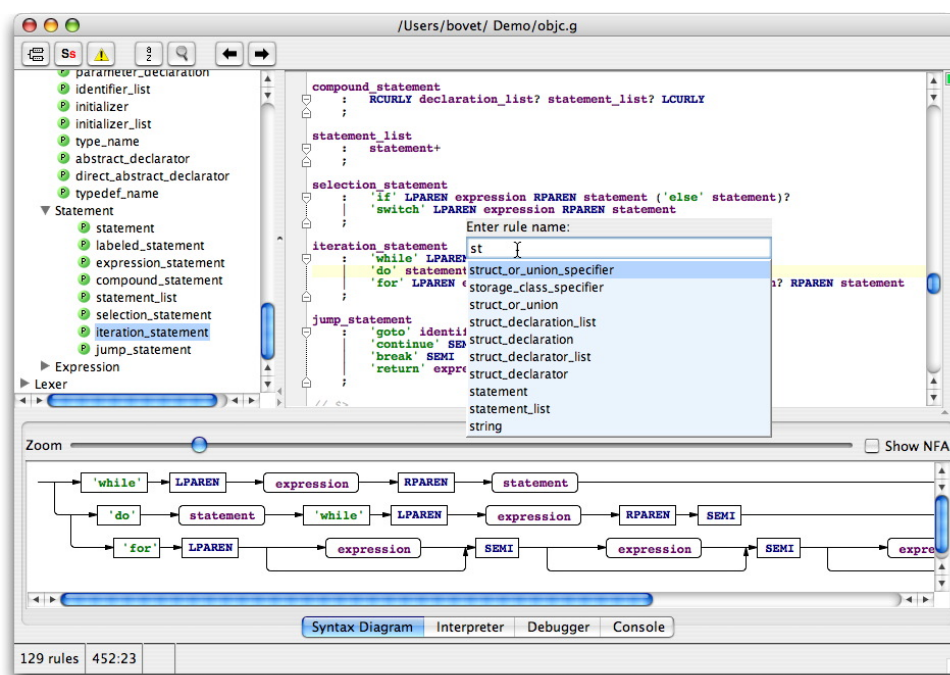


Figura 3.2: Janela de edição do ANTLRWorks

Sendo uma ferramenta escrita em Java, tem grandes vantagens para quem procura utilizar um analisador sintático em qualquer plataforma. E tal como o ANTLR, estão disponíveis facilmente um bom número de gramáticas para várias linguagens.

3.4 Majestic-12 HTML parser

O HTML parser da Majestic-12 [36] é um parser que pretende concentrar-se numa alta performance, sendo a ferramenta ideal para processar documentos grandes e complexos. Trata-se duma biblioteca desenvolvida para .NET 1.1 e 2.0 na linguagem C# e que oferece classes para o tratamento de HTML.

Este parser não transforma o código de entrada numa árvore de sintaxe, apenas percorre o documento, passo a passo, devolvendo informação sobre as tags de abertura e de fecho, os scripts que encontra, o texto entre tags e comentários.

A classe HTMLchunk é usada para devolver o texto dentro da tag ou as próprias tags, sendo capaz de gerar HTML com novos parâmetros e atributos a nível de programação. Pelas características das páginas Web, este tipo de analisador sintático deverá ser suficiente para ser utilizado na conversão automática do código.

Este parser também pode ser aplicado aos HTC, por estes terem uma estrutura comum ao HTML.

3.5 Sistemas de Transformação de Programas

Existem algumas ferramentas dedicadas à transformação de código que consistem na escrita de um conjunto de regras num formato específico das ferramentas, efectuando as transformações numa forma padronizada e independente do código desenvolvido por nós, aplicando um conjunto de regras em determinadas condições pré-definidas.

3.5.1 TXL

O TXL [37] é uma linguagem de programação dedicada à realização de transformações de código fonte. Essa linguagem pode ser utilizada com uma ferramenta livre, desenvolvida em conjunto com a linguagem, o FreeTXL, que oferece um conjunto de programas para múltiplas plataformas. São também disponibilizadas várias gramáticas para facilitar a conversão de linguagens existentes, inclusivamente de JavaScript, que irá ser relevante para o trabalho.

Para se utilizar deve ser definida uma gramática da linguagem a converter e um conjunto de regras a utilizar na conversão, esta gramática não recorre a qualquer tipo de ferramenta externa de geração de parser, sendo que a especificação da análise sintáctica é parte integrante do próprio TXL [38]. O programa deve ser invocado indicando o nome do ficheiro a traduzir. As regras serão encontradas automaticamente e aplicadas. É possível também o programa imprimir a árvore de sintaxe do programa antes de converter e depois de converter num formato tipo XML. Útil para tentar perceber que tipo de expressões se quer converter.

A gramática é definida por um conjunto de `define` que permitem associar uns símbolos não terminais a outras expressões não terminais bem como terminais [39].

```
define program
    [expression]
end define
```

Uma das definições tem de ser precisamente `program`, isto porque o TXL utiliza sempre a definição com este nome como a principal que define a estrutura da linguagem de input [39, 38].

Os símbolos terminais que possam entrar em conflito com palavras especiais do próprio TXL devem ser precedidos dum apóstrofo para os distinguir (`function` deve ser definido como `'function`, + como `'+`, # como `'#`, etc.). Os símbolos não terminais são apresentados entre parêntesis rectos (`[expression]`, `[if_statement]`). E para separar várias hipóteses de construção de um não terminal, utiliza-se o `|`.

Símbolos terminais que tenham mais do que um carácter, têm de ser definidos numa secção chamada `compounds`. Enquanto que palavras reservadas da linguagem de input devem ser definidas numa secção `keys` para as distinguir de identificadores.

Existem alguns símbolos não terminais pré-definidos pela linguagem, evitando assim precisar de tratar símbolos comuns em quase todas as linguagens como literais, identificadores e números.

- [number]
Qualquer constante numérica sem sinal (ex. 123, 1.23, 1.2e-3)
- [id]
Qualquer identificador (ex. abc, ABC, aBc, AbC, A_bc, abc_)
- [stringlit]
Qualquer *string* delimitada por aspas duplas (ex. “hi there”)
- [charlit]
Qualquer *string* delimitada por aspas simples (ex. ‘hello yourself’)

O seguinte excerto de código TXL, mostra como aplicar uma transformação nesta linguagem. Neste caso, uma qualquer expressão, ao qual é atribuída a variável E, é substituída por outra expressão que consiste nessa mesma expressão E, à qual são aplicadas as regras `resolveAddition`, `resolveSubtraction`, `resolveMultiplication`, etc. Devido à condição imposta depois, esta substituição de expressões só ocorre se a nova expressão for realmente diferente da primeira.

```
rule main
  replace [expression]
    E [expression]
  construct NewE [expression]
    E [resolveAddition] [resolveSubtraction]
      [resolveMultiplication] [resolveDivision]
        [resolveParentheses]
  where not
    NewE [= E]
  by
    NewE
end rule
```

A regra ou função `main` é a principal função ou regra a aplicar pelo TXL. A partir dela podem ser invocadas outras regras ou funções que vão sendo aplicadas tratando a transformação dos vários segmentos do código.

Todas estas características fazem do TXL uma ferramenta simples de utilizar, facilmente expansível e uma boa opção para a conversão dos scripts.

3.5.2 Stratego/XT

O Stratego e o XT [40, 41] são uma linguagem e um conjunto de ferramentas, respectivamente, utilizados no contexto da transformação de programas. A linguagem Stratego fornece um conjunto de regras para reescrever transformações básicas, estratégias de controlo da aplicação dessas mesmas regras, uma sintaxe concreta para exprimir os padrões de regras na sintaxe da linguagem objectivo e regras dinâmicas para transformações dependentes do contexto. O XT oferece uma colecção de componentes para transformação flexíveis e reutilizáveis bem como ferramentas para gerar tais componentes a partir de especificações declarativas [42].

A base das transformações suportadas pela linguagem Stratego está na definição de termos e em regras para reescrever esses termos. Os termos são semelhantes a árvores de sintaxe, servindo para representar documentos estruturados. Um exemplo dum termo representando uma estrutura em Stratego é `Call("square", [Plus(Var("x"), Int(3))])`, que será uma representação típica de `square(x + 3)`.

Os constructores de termos são declarados na forma de assinaturas, nestas assinaturas é indicado o tipo de resultado dos termos e os tipos dos seus argumentos:

```
signature
  sorts Id Exp
  constructors
    Var : Id -> Exp
    Plus : Exp * Exp -> Exp
    If : Exp * Exp * Exp -> Exp
```

Uma regra para transformação em Stratego é definida na forma $R : p_1 \rightarrow p_2$, em que R representa o nome da regra, p_1 representa o padrão a transformar, e p_2 o padrão que será o resultado da transformação. Estas transformações também podem ser condicionais, sendo que a regra só é aplicada perante determinadas condições. Um exemplo duma regra com condição é:

```
EvalBinOp : Plus(Int(i), Int(j)) -> Int(k) where k := <add>(i, j)
```

Neste caso podemos ver que algo que tipicamente representa uma soma entre duas variáveis será transformado na soma dessas mesmas variáveis.

Uma vez que a representação por termos pode não ser o mais adequada ou inteligível para aplicação de determinadas regras, o Stratego também suporta uma sintaxe concreta da definição das suas transformações. O exemplo apresentado anteriormente seria descrito assim (assumindo que a linguagem utiliza somas no tipo “ $a + b$ ”) nesta notação:

```
EvalBinOp : [| i + j |] -> [| k |] where k := <add>(i, j)
```

O Stratego permite também a definição a nível de programação da estratégia de aplicação das transformações. Assim, cabe ao utilizador decidir que tipo de estratégia aplicar a determinados termos, de acordo com o que melhor se aplicará ao seu projecto de transformação, pois é possível que a aplicação de regras sucessivamente até não existir nenhuma porção de código transformável seja adequada. Desta forma pode-se evitar, por exemplo, a aplicação de regras infinitamente recursivas.

Outra característica do Stratego, relativamente às regras, é permitir regras dinâmicas que podem ser criadas durante a execução, permitindo tornar algumas regras sensíveis ao contexto em que estão inseridas. É possível alterar, apagar ou criar novas regras e estratégias herdando informação do contexto onde estão a ser aplicadas.

Relativamente ao XT, a utilidade que ele tem ao ser utilizado junto com o Stratego é a de definir a sintaxe das linguagens a transformar e fornecer capacidades para impressão das mesmas. O conjunto de ferramentas é capaz de gerar uma AST a partir de código fonte, gerar as assinaturas para o Stratego baseado na mesma árvore (AST), gerar um validador para transformações que sejam eventualmente realizadas e ainda um *pretty-printer* para transformar as AST modificadas de volta a texto.

A figura 3.3, retirada de [42], mostra a estrutura duma transformação que aplica estas ferramentas. Na parte de transformação recorre à linguagem Stratego, enquanto que as outras tarefas são geradas pelas ferramentas XT.

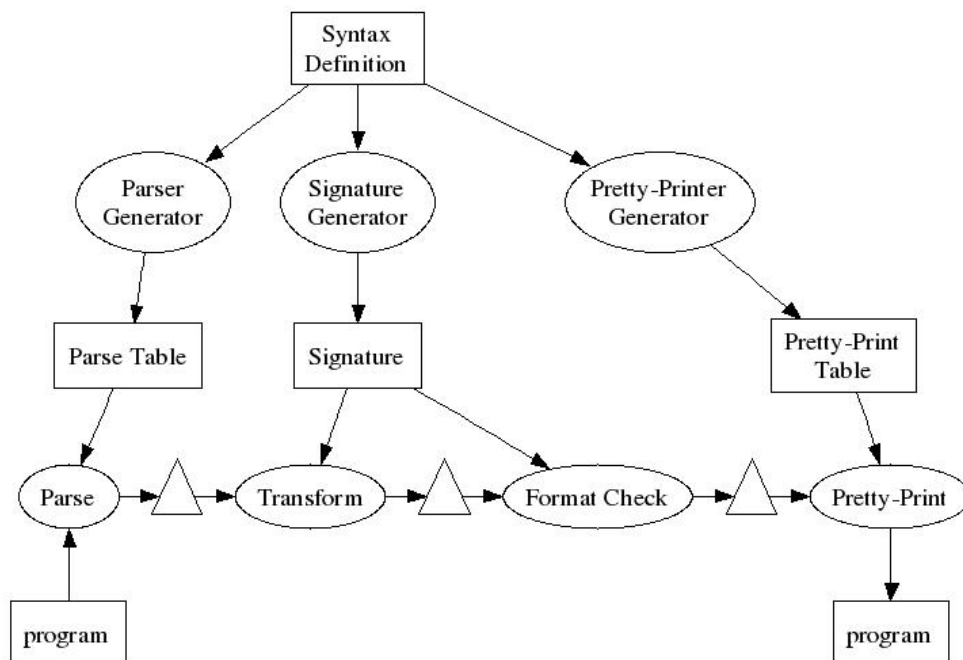


Figura 3.3: Estrutura duma transformação Stratego/XT.

3.5.3 DMS Software Reengineering Toolkit

O DMS [43] é um sistema de análise e transformação de programas que pode ser aplicado a tarefas que vão desde renovar documentação até migrar sistemas de larga escala [44]. Este sistema é distribuído comercialmente com preços diversos, dependendo das pretensões do cliente.

É uma solução desenvolvida pela Semantic Designs, Inc., empresa sediada em Austin, Texas. Esta empresa foi criada em 1995 depois dos fundadores terem recebido fundos no valor de 1,9 milhões de dólares para conduzir investigação sobre a ideia DMS. Desta investigação veio a resultar um conjunto de soluções que se mostram eficientes na automação de análise, melhoria e transformação de sistemas de software complexos [45].

Devido ao pouco sucesso inicial do projecto, que era incapaz de processar programas com 10000 SLOC enquanto inserido no contexto de investigação, os responsáveis chegaram à conclusão que a escalabilidade é o factor mais importante neste tipo de sistemas para que estes possam ter sucesso a nível comercial. Isto porque aplicações que precisem deste tipo de ferramentas, num cenário real, podem ser constituídas por centenas de milhares de linhas de código, podem recorrer a várias linguagens de diferentes tipos, ter configurações e fluxos de informação completamente diferentes, entre outras barreiras à aplicação destas ferramentas [44, chap. 2 - Application Scale as a barrier to tools]. Assim, conseguir lidar com esses desafios foi o principal aspecto considerado no desenvolvimento do DMS.

Algumas das funcionalidades que o DMS Software Reengineering Toolkit fornece incluem:

- Construção automática de AST, incluindo comentários do código.
- Geração de *pretty-printer* para transformar AST em código com boa formatação.
- Representação de múltiplos domínios ao mesmo tempo, ou seja, permite o processamento de sistemas com mais do que uma linguagem e a transformação de uma linguagem para outra.
- Transformações à árvore através da aplicação de regras ou de procedimentos.
- Construção duma tabela de símbolos sofisticada para nomes globais, locais, herdados e regras para gestão de *namespaces*.

Uma típica regra de transformação em DMS encontra-se na forma $LHS \rightarrow RHS$ **if** *condition*, em que LHS e RHS representam padrões reconhecidos da linguagem fonte, suportando a inclusão de variáveis arbitrárias que representam subcadeias reconhecidas da linguagem. A utilização do **if**, bem como da respectiva condição é opcional.

```

default domain C.
rule auto_inc(v:lvalue):
  statement->statement =
    "\\v = \\v+1;" rewrites to "\\v++;";
  if no_side_effects(v).

```

Este excerto de código representa uma regra para aplicar a linguagem C que transforma uma atribuição a uma variável, do valor dela própria somado a um numa incrementação.

```

antes: (*Z) [a>>2]=(*Z) [a>>2]+1;
depois: (*Z) [a>>2]++;

```

Este seria o efeito prático da transformação definida antes.

3.5.4 Tom

O Tom [46, 47] é uma ferramenta desenhada para manipular estruturas de árvore e documentos XML. Para isso, oferece funcionalidades de “casamento” de padrões para inspeccionar objectos e recolher valores. É um projecto que tem origem no seio do Institut National de Recherche en Informatique et Automatique e que está em desenvolvimento activo desde 2001.

Esta ferramenta pode ser utilizada em C, Java, OCaml, entre outras. No entanto, é quando utilizada em Java que as capacidades do Tom podem ser exploradas ao máximo, estando incluídos nessa versão um gerador de estruturas de dados baseadas em árvores orientadas a objectos e uma linguagem de estratégia para controlar transformações.

A principal constructor da linguagem é *%match*, este método é similar a um *switch/case*, uma vez que dado um determinado objecto e uma lista de padrões e respectivas acções, ele trata de encontrar o primeiro padrão que está de acordo com o objecto e executa a respectiva acção. A diferença em relação à expressão *switch/case*, comum a muitas linguagens, é que a operação é feita sobre termos e não sobre valores atómicos como caracteres ou números inteiros.

Outro constructor, essencial para a realização de transformações de linguagem, é o *%gom*. Este serve para definir a estrutura de dados, ou assinatura e prevê ainda a criação duma secção para inserir regras para reescrever os termos.

```

%gom {
  module Cars
    abstract syntax

    Vehicle = car(interior:Colors, exterior:Colors, type:Type)
              | bike()

```

```

    Type = ecological()
        | polluting()

    Colors = red()
        | green()
}

```

Esta é a estrutura duma secção *%gom* em que é definida uma estrutura de dados representando carros (Cars). Dessa estrutura fazem parte três tipos: *Vehicle*, *Type* e *Colors*, que por sua vez têm dois operadores cada.

O *Vehicle* tem um operador *car* que tem como parâmetros dois tipos *Colors*, para definir a cor interior e externa do carro, e de tipo *Type*, para definir se o carro é ecológico ou poluidor.

A definição de regras para reescrever termos é feita na mesma secção, com a seguinte estrutura:

```

%gom {
    module Logic
        imports int
        abstract syntax
        Bool = Input(n:int)
            | True()
            | False()
            | Not(b:Bool)
            | Or(b1:Bool, b2:Bool)
            | And(b1:Bool, b2:Bool)
            | Nand(b1:Bool, b2:Bool)
            | Xor(b1:Bool, b2:Bool)

        module Logic:rules() {
            Not(a)    -> Nand(a,a)
            Or(a,b)   -> Nand(Not(a),Not(b))
            And(a,b)  -> Not(Nand(a,b))
            Xor(a,b)  -> Or(And(a,Not(b)),And(Not(a),b))
            Nand(False(),_)    -> True()
            Nand(_,False())    -> True()
            Nand(True(),True()) -> False()
        }
    }
}

```

Este é um exemplo no qual são aplicadas regras para simplificar expressões lógicas, para isso é criada uma secção *rules()* associada à estrutura de dados respectiva, e as transformações são definidas com a sintaxe *LHS* $\rightarrow$ *RHS*, sendo que *LHS* representa o termo a transformar e o *RHS* o resultado da transformação [48].

3.6 Conclusões

O código HTML tem uma estrutura simples, relativamente fácil de tratar. Não existia a necessidade de criar uma AST complexa para tratar do mesmo, tendo sido tomada a opção de adoptar um parser já existente para lidar com as transformações a nível de HTML.

Relativamente às transformações a realizar a nível de *script* e estilo, pelo facto do trabalho se centrar em tecnologias Microsoft e, por isso, páginas que normalmente são servidas pelo IIS, servidor proprietário da Microsoft e com uma forte integração com tecnologias .NET, há todo o interesse em utilizar código gerado para .NET, pois assim o conversor poderá ser facilmente aplicado a nível do servidor da Microsoft, se tal for necessário.

A possibilidade de usar as bibliotecas ANTLR em .NET fazem dele o melhor candidato para a criação do *parser* para JavaScript e até de outros aspectos incluídos nos documentos HTML que possam necessitar de conversão, como CSS. A disponibilidade de muitas gramáticas para as mais variadas linguagens torna o ANTLR numa boa solução, caso seja necessário converter outras linguagens que possam surgir no futuro.

A linguagem TXL também é uma boa solução pois é dedicada a transformações de código fonte e é simples de interpretar e usar. No entanto, o facto de ser necessário ter o programa instalado pode causar inconveniências no conversor. O conjunto Stratego/XT e as ferramentas DMS parecem ser mais poderosas e versáteis, mas não são evidentes as vantagens que podem trazer em relação ao TXL no contexto deste projecto. A simplicidade de utilização do TXL e a maior facilidade com que se arranjam exemplos e tutoriais para FreeTXL, fazem com que valha a pena explorar a sua utilização para alcançar o objectivo proposto nesta tese. O Tom só pode ser devidamente explorado recorrendo à linguagem Java e, por isso, não é a ferramenta ideal no contexto deste projecto.

Capítulo 4

Conversão de Código

Neste capítulo da tese são abordados alguns factores que geram incompatibilidade entre *browsers* e que serão mais relevantes para o trabalho. São dados alguns exemplos de conversões de *sites* feitas de forma manual, tirando daí as respectivas ilações sobre os aspectos em que se deve intervir de forma a conseguir efectuar uma conversão bem sucedida do HTML com JavaScript.

4.1 Factores que geram incompatibilidade

No capítulo anterior, na secção 3.1.1, foram enunciados alguns dos factores que são específicos do Internet Explorer e que colocam dificuldades à interpretação do conteúdo das páginas por outras aplicações de navegação *Web*.

Desta feita, e fazendo uma pequena análise à estrutura da aplicação “Coolbiz - Back-Office” vão ser seleccionados os aspectos nos quais se vai centrar o protótipo desta conversão automática.

4.1.1 Script

No contexto das páginas *Web* dinâmicas é normal recorrer a funcionalidades de *scripting*. Estas funcionalidades são executadas do lado do cliente, para conseguir uma interacção mais rica com o utilizador. Como já vimos anteriormente existem duas alternativas em termos de linguagens neste contexto muito utilizadas no IE, a VBScript [22] e o JScript [21]. A última é uma linguagem que nasceu como a implementação do JavaScript, na altura parte do Netscape Navigator, para o *browser* da Microsoft. Precisamente por isso, estas linguagens têm muitos aspectos em comum.

Entretanto, com o surgimento das normas abertas para o desenvolvimento *Web*, foi criada a definição do ECMAScript [49], que é uma linguagem standard e cujos métodos e

propriedades estão implementados na sua totalidade tanto no JScript como no JavaScript, pelo que estas se tornaram implementações de ECMAScript com extensões específicas aos respectivos *browsers* [49]. Todas estas linguagens são normalmente referidas como JavaScript apenas.

A aplicação da Coollink a converter utiliza este tipo de linguagem de *scripting*, mas específico para IE. Assim, um dos aspectos sobre o qual esta tese se vai debruçar é na conversão de JavaScript específico para IE de forma a que se torne compatível com o Firefox e também com o Safari.

4.1.2 HTML Components

Uma tecnologia que provoca incompatibilidades graves entre *browsers* são os HTML Components (HTC) [20], sendo que, normalmente, o conteúdo definido por eles não é sequer apresentado em outras aplicações que não o IE. A aplicação “Coolbiz - BackOffice” depende em grande parte destes HTC no seu funcionamento e, por isso, é essencial saber como ultrapassar as dificuldades introduzidas por eles à interoperabilidade entre navegadores.

Os HTML Components [20] são uma tecnologia introduzida pela MicroSoft em 1998 e que permitem associar um determinado comportamento, através da definição de estilos e de *scripts* específicos, a um elemento HTML. Os ficheiros HTC são, essencialmente, páginas HTML contidas dentro de elementos especiais que servem para descrever algumas opções desse mesmo HTC. Esses elementos são [50]:

- COMPONENT

Este é o elemento principal de um HTC que serve para encapsular todos os outros elementos específicos de HTC, sendo que é nele que pode ser feita a identificação do HTC, por exemplo, por meio da *tag* HTML customizada a que está associado. Apenas deve aparecer uma vez por cada HTC.

Os atributos associados a este elemento são:

- **ID** → Opcional. Identificador do elemento COMPONENT. Análogo a atributo ID em DHTML.
- **lightWeight** → Opcional. Indica se o ficheiro HTC contém código HTML. Se não, o valor deve ser true para melhorar a performance.
- **literalContent** → Opcional. Indica se o conteúdo dentro da *tag* customizada deve ser renderizado normalmente ou se deve ser tratado apenas como um conjunto de dados.
- **NAME** → Opcional. Nome pelo qual o comportamento DHTML irá ser referido no documento que o contém.

- ***supportsEditMode*** → Opcional. Indica se o HTML contido neste componente é editável.
- ***tagName*** → Opcional. Indica qual a *tag* customizada a que este HTC corresponde no documento principal.
- ***URN*** → Opcional. String no formato *Uniform Resource Name* que identifica de forma única o componente.

- **PROPERTY**

Este elemento, que deve estar colocado dentro de `COMPONENT` e que pode aparecer múltiplas vezes, serve para indicar uma propriedade do HTC que irá ser exposta ao documento principal.

Os atributos associados a este elemento são:

- ***GET*** → Opcional. Indica a função a ser executada quando o valor da propriedade é lido.
- ***ID*** → Opcional. Identificador do elemento `PROPERTY`. Análogo a atributo `ID` em `DHTML`.
- ***INTERNALNAME*** → Opcional. Nome interno ao componente da propriedade.
- ***NAME*** → Necessário. Nome da propriedade a expor ao documento principal.
- ***PERSIST*** → Opcional. Indica se a propriedade persiste como parte da página.
- ***PUT*** → Opcional. Indica a função a executar quando o valor da propriedade é alterado.
- ***VALUE*** → Opcional. Valor por defeito da propriedade.

- **METHOD**

O elemento `METHOD`, também filho de `COMPONENT`, deve aparecer para indicar um método do HTC que deve ser exposto ao documento principal também. Permitindo assim, invocar esse método a partir do HTML que define a página.

Os atributos associados a este elemento são:

- ***ID*** → Opcional. Identificador do elemento `METHOD`. Análogo a atributo `ID` em `DHTML`.
- ***INTERNALNAME*** → Opcional. Nome pelo qual o método é conhecido dentro do componente.
- ***NAME*** → Necessário. Nome do método a expor ao documento principal.

- **EVENT**

`EVENT` é o elemento usado para definir eventos customizados associados ao HTC e que devem ser expostos ao documento principal, de forma a que esse evento possa ocorrer no contexto principal.

Os atributos associados a este elemento são:

- ***ID*** → Opcional. Identificador do elemento `EVENT`. Análogo ao atributo `ID` em `DHTML`.
- ***NAME*** → Necessário. Nome do evento a expor ao documento principal.

- `ATTACH`

O elemento `ATTACH` serve para associar funções do HTC a qualquer evento, de forma que essa função seja executada quando o evento ocorre. Tal como os anteriores elementos, este deve estar inserido dentro de `COMPONENT` e pode aparecer várias vezes.

Os atributos associados a este elemento são:

- ***EVENT*** → Necessário. Nome do evento `DHTML` associado.
- ***FOR*** → Opcional. Origem do evento: `window`, `document` ou `element`.
- ***ID*** → Opcional. Identificador do elemento `ATTACH`. Análogo ao atributo `ID` em `DHTML`.
- ***ONEVENT*** → Necessário. Função a executar quando o evento definido ocorre.

- `DEFAULTS`

Este elemento apenas deve aparecer uma vez por componente e serve para definir algumas propriedades por defeito do HTC.

Os atributos associados a este elemento são:

- ***canHaveHTML*** → Opcional. Booleano que indica se o ficheiro HTC pode ter `HTML`.
- ***contentEditable*** → Opcional. Booleano que indica se o conteúdo da *tag* do HTC é editável.
- ***style*** → Opcional. Especifica o estilo da *tag* definida no HTC.
- ***tabStop*** → Opcional. Booleano que indica se a *tag* definida no HTC se torna activa através da navegação por tabulador.
- ***viewInheritStyle*** → Opcional. Indica se o *viewlink* herda o estilo do documento `HTML` principal.
- ***viewLinkContent*** → Opcional. Indica se o `HTML` do ficheiro HTC é usado como *viewlink*.

- ***viewMasterTab*** → Opcional. Indica se o elemento associado ao *viewlink* é incluído na sequência de tabulação do documento principal.

Os HTML Components podem ser utilizados de duas maneiras [20, chap. Overview]. Num primeiro caso eles são usados para associar comportamentos a elementos HTML através da propriedade `behavior` (-ms-behavior no IE8) de CSS, existindo também as respectivas funções JavaScript: `addBehavior` e `removeBehavior`. Nesta hipótese não faz sentido incluir HTML no HTC, uma vez que apenas a folha de estilos e um *script* podem ser utilizados. Outra maneira de utilizar os HTC é para definir um elemento HTML personalizado que pode ter qualquer tipo de estilo, *script* e HTML associado.

Assim, um ficheiro HTC típico terá a seguinte estrutura:

```
<HTML>
<HEAD>
<PUBLIC:COMPONENT>
    <PUBLIC:DEFAULTS/>
    <PUBLIC:PROPERTY/>
    :
    <PUBLIC:ATTACH/>
    :
    <PUBLIC:METHOD/>
    :
    <PUBLIC:EVENT/>
    :
</PUBLIC:COMPONENT>
<STYLE>
</STYLE>
<SCRIPT>
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Todas estas *tags* são opcionais, inclusive o IE pode interpretar bem elementos como ATTACH fora de um COMPONENT, no caso do HTC não definir um elemento personalizado.

4.1.3 Cascading Style Sheets

Cascading Style Sheets (CSS) [51] é uma tecnologia que permite a definição de estilos de forma não dependente do código HTML da página, o que fornece ao programador a hipótese de mudar radicalmente o estilo da página de forma fácil, dependendo da boa

estruturação do documento. Estas folhas de estilo CSS estão presentes em grande escala nas páginas *web* e a aplicação financeira em causa no âmbito deste projecto não é excepção.

Embora sejam menos problemáticas as incompatibilidades a nível de CSS, afectando essencialmente a apresentação visual da página, existem propriedades CSS específicas de cada browser que são incompatíveis noutros. Normalmente, estas propriedades devem ter no seu nome um prefixo indicando a aplicação que a reconhece como válida, e isto acontece com as propriedades específicas do IE na nova versão 8. No entanto, em versões anteriores a Microsoft não utilizava estes prefixos, pelo que pode ser difícil de detectar numa folha de estilos CSS que propriedades estão a ser correctamente interpretadas ou não.

4.2 Exemplos de conversão manual

Nesta secção são apresentados exemplos de páginas que apenas funcionam em IE, devido a recorrerem a HTML Components na sua implementação. Estes exemplos representam as duas formas de se utilizar HTC no desenvolvimento de uma página *Web* e foram alvo de conversão manual para funcionarem total ou parcialmente noutros browsers. Os passos tomados nessas conversões vão ser descritos e explicados.

4.2.1 Hilite

O Hilite é um exemplo muito simples retirado de [52]. Esta aplicação apresenta uma lista sem ordenação em que cada elemento é iluminado ao ser sobreposto pelo rato. Simultaneamente, o cursor altera o seu estilo, transformando-se na representação de uma mão humana. Estes efeitos são definidos num ficheiro HTC que é invocado através de CSS. O código pode ser consultado no anexo A.

Uma vez que o Firefox não consegue perceber a instrução `behavior: url('hilite.htc');`, e como tal não carrega o conteúdo do ficheiro *hilite.htc*, os pedaços desse HTC relevantes para a execução correcta da página foram copiados para o documento HTML principal. Neste caso, esse conteúdo é apenas um *script*.

Este *script*, no entanto, tem de sofrer ainda dois tipos de alterações. A primeira tarefa diz respeito à mudança de contexto do código, isto porque num HTC ele insere-se no contexto do elemento apenas, e agora faz parte do contexto do documento. A segunda parte da conversão refere-se à conversão de métodos, propriedades e variáveis globais JavaScript que apenas funcionam em IE para alternativas que sejam compatíveis com Firefox e Safari.

A mudança de contexto é um problema devido ao uso no *script* da variável `element`. No HTC, esta variável refere-se ao elemento associado ao HTC, neste caso é um item de

uma lista, com *tag* `li` em HTML. Fora do HTC, no entanto, este `element` não existe, pura e simplesmente. A solução encontrada para lidar com este problema é passar um ID para as funções que utilizem a variável `element` e no início dessa função criar essa variável, recorrendo à função `document.getElementById(id)` ; , para atribuir o elemento correcto a essa variável. Desta forma, `element` já pode ser utilizado tal como era no componente e irá servir até para resolver outro problema da mudança de contexto.

```
function Restore(id)
{
    var element = document.getElementById(id);
    if (event.srcElement == element)
    {
        ...
    }
}
```

Originalmente, são feitas referências directas a propriedades do elemento (`style` e `runtimeStyle`), no script colocado no documento principal estas referências não estão associadas ao elemento `li`, assim, para que a página funcione como pretendido, coloca-se `element` antes de `style` e `runtimeStyle`, de forma a que estes últimos sejam invocados como um campo do elemento.

```
...
if (event.srcElement == element)
{
    normalColor = element.style.color;
    element.runtimeStyle.color = "red";
    element.runtimeStyle.cursor = "hand";
}
...
```

Efectuadas estas alterações, está na altura de passar pela fase de conversão do *script* para outros *browsers*. Um dos primeiros problemas é a variável `event`. Enquanto que o IE reconhece `event` como uma variável global pré-definida, o Firefox necessita que esta variável seja criada antes de poder ser utilizada. Para resolver esta situação basta passar como um novo argumento das funções essa mesma variável. O Safari reconhece a propriedade `srcElement` dos eventos, o Firefox, no entanto, utiliza `target` para se referir ao elemento onde o evento aconteceu. A condição das expressões *if* deve ser alterada para reflectir isso. Adicionalmente, é necessario substituir `runtimeStyle` (apenas reconhecido pelo IE) por `style` e `"hand"` por `"pointer"` na definição do estilo do cursor do rato.

```
function Hilite(id, event)
{
```

```
...
    if (event.srcElement == element || event.target == element)
    {
        ...
        element.style.color = "red";
        element.style.cursor = "pointer";
    }
}
```

Em termos de *script* a conversão está concluída, mas a página não está funcional nesta fase, uma vez que os eventos não estão associados às respectivas funções, enquanto que anteriormente estavam associados por meio dos dois elementos `PUBLIC:ATTACH` no HTC. A alteração para tornar a aplicação funcional vai ocorrer no HTML, dentro das *tags* dos itens da lista. Estas *tags* vão passar a ter atributos dos eventos definidos no HTML Component indicando as respectivas funções a ser executadas quando o evento ocorre, já com os novos argumentos incluídos.

```
<LI ID=L1 onmouseover="Hilite('L1',event)" onmouseout="Restore('L1',event)">
<LI ID=L2 onmouseover="Hilite('L2',event)" onmouseout="Restore('L2',event)">
```

Depois de concluídos todos estes passos, a página está finalmente compatível com os navegadores Internet Explorer, Firefox e Safari.

4.2.2 Calendar

Esta aplicação, retirada de [53], apresenta um calendário do mês actual ao utilizador e permite que o utilizador interaja com este, podendo colocar qualquer apontamento em dado dia, bastando para isso carregar sobre o respectivo dia e escrever o apontamento na caixa de diálogo que aparece. A forma como esta aplicação utiliza os HTC é substancialmente diferente do exemplo anterior. Neste caso, eles servem para definir elementos personalizados, no caso o próprio calendário, o dia actual e os restantes dias. O código pode ser consultado no anexo B. Os ficheiros HTC são referenciados, desta vez, através do uso de uma *tag* `?IMPORT` em HTML, contendo informação sobre a localização do ficheiro onde é implementado o HTC e o respectivo *namespace*.

O calendário é o elemento principal da página e uma vez que ele é definido num HTC, a página praticamente não apresenta conteúdo quando vista através do Safari ou do Firefox, como se pode ver na tabela 4.1. O HTC do calendário inclui ainda referências para os outros elementos definidos em HTC (o dia de hoje e um dia genérico), que ele usa na criação do calendário para aplicar o estilo a cada dia, diferenciando o dia actual.

O aspecto mais importante a incluir no documento principal é o *script* que é executado no corpo do componente calendário e que cria o mesmo. Este *script* não contém incompatibilidades com Firefox, como foi possível verificar por execução da aplicação. As

alterações que ele vai sofrer prendem-se apenas com o uso dos elementos `ANYDAY:DAY` e `TODAY:DAY`, definidos nos restantes HTC, isto porque o Firefox não processa sequer etiquetas que não compreende. A maneira de ultrapassar esta questão é substituir estas *tags* customizadas por elementos `div` contendo os mesmos atributos de `ANYDAY:DAY` e `TODAY:DAY`. A estes atributos vão ser acrescentados os eventos e respectivas funções a executar contidas em *day.htc* e *today.htc*.

```
<DIV onclick=fnShowAppts() oncontentready=fnInit() value=' + dayOfMonth + '>
</DIV>
```

As funções dos eventos vão sofrer alterações na sua passagem para o ficheiro HTML semelhantes àquelas que ocorreram no exemplo anterior, ou seja, inclusão de um argumento `ID` na função e criação da variável `element` a partir desse mesmo identificador. No entanto, no código original, estes elementos não têm um identificador definido e uma vez que eles são escritos no documento num ciclo dentro do *script*, o `ID` não pode ser fixo, pois é necessário identificar cada elemento criado dinamicamente. Uma estratégia possível é utilizar a variável `dayOfMonth`, que já está a ser utilizada para atribuir um valor diferente a cada dia na criação do `ID`. No entanto também podia ser criada uma função para gerar um `ID` aleatório, por exemplo, desde que esse identificador fosse usado na chamada das funções `fnShowAppts` e `fnInit`. Adicionalmente, e devido aos dois ficheiros HTC para qualquer dia e para o dia actual usarem o mesmo nome para as funções, o nome destas vai ser alterado ao serem transferidas para o documento principal, de forma a distingui-las.

```
<DIV onclick=todayfnShowAppts('\a' + dayOfMonth + '\')
oncontentready=todayfnInit('\a' + dayOfMonth + '\')
id=a' + dayOfMonth + ' value=' + dayOfMonth + '></DIV>
```

Para além das alterações já referidas, são efectuadas ainda algumas mudanças nas funções dos HTC. Uma mudança obrigatória é tirar a atribuição de um valor a *defaults.view-link*. Isto é uma propriedade exclusiva dos HTC e não só não faz sentido neste contexto, como quebra a execução do *script* nos outros *browsers*. Outra alteração deve-se ao facto do IE permitir utilizar atributos dos elementos HTML em JavaScript como se se tratassem de propriedades de um objecto. Não existe distinção, por exemplo, entre `element.value` e `element.date` na forma como são utilizados no *script*. O Firefox e outros navegadores não toleram esta ambiguidade, obrigando o programador a utilizar a função `getAttribute('value')` para conseguir utilizar o valor incluído no HTML. Uma última alteração tem a ver com a alteração do nome dado às classes, isto porque as folhas de estilo incluídas nos HTC também utilizam o mesmo nome para definir a classe, à semelhança das funções JavaScript.

```
...
function todayfnInit (id)
```

```
{
    var element = document.getElementById(id);
    element.innerHTML = '<FONT COLOR="red">'
        + element.getAttribute('value') + '</FONT>';
    element.className = "todayclsDay";
    //defaults.viewLink = document;
    ...
}
function anydayfnShowAppts (id)
{
    var element = document.getElementById(id);
    ...
}
...
<STYLE>
...
.anydayclsDay {
    ...
}
...
```

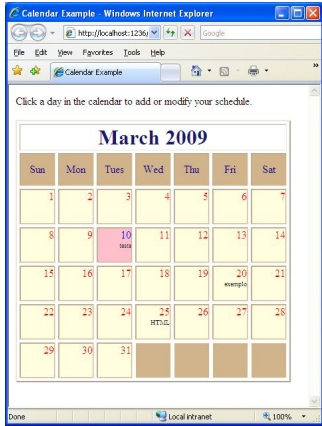
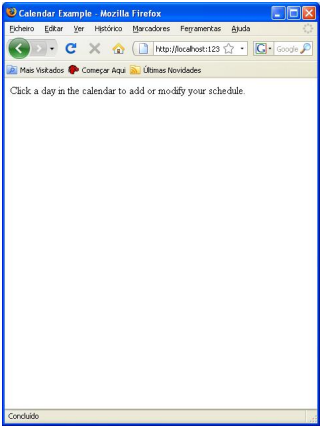
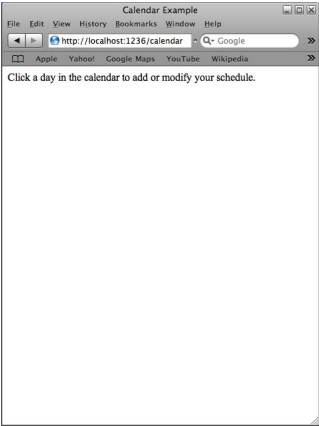
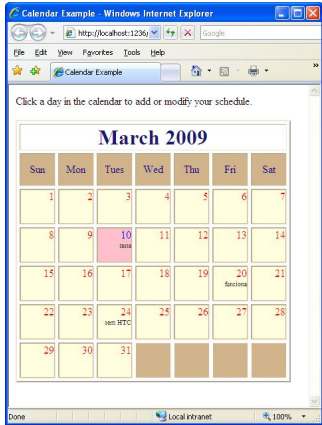
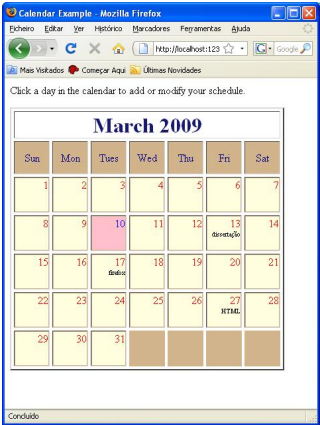
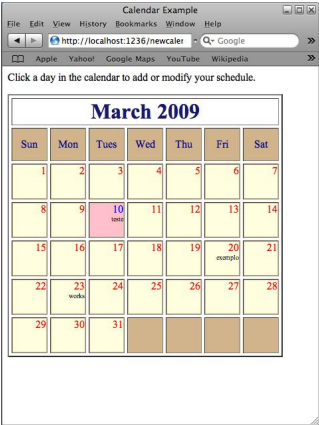
A tabela 4.1 mostra os resultados obtidos por esta conversão.

4.3 Catálogo de Conversão

Através da consulta de várias documentações [52, 54, 55, 56] afectas ao tema, foi reunida uma lista de termos apenas compatíveis com IE e respectivas alternativas a aplicar noutros *browsers*. Esta lista é composta por vários tipos de termos, tanto relacionados com Javascript, como CSS. Estas tabelas não são fruto de um estudo exaustivo de componentes incompatíveis e respectivas alternativas, estando limitada a casos mais comuns. A descoberta de novas incompatibilidade entre JScript e JavaScript é um processo de aprendizagem constante, baseado na prática de programação nestas linguagens. As implementações dos *browsers* mudam constantemente, tanto a nível de suporte de novos métodos, como a nível de correcção da implementação desses mesmos métodos. É comum em todos os *browsers* muito métodos *standard* não estarem implementados de acordo com a respectiva especificação, como se pode ver em [54], uma página *web* que se baseia precisamente na aplicação prática dos métodos para testar a compatibilidade de vários navegadores.

A tabela 4.2 apresenta um conjunto de propriedades comuns dos objectos em JavaScript. Estas propriedades são acedidas na forma *variável.propriedade* e servem para definir ou retirar dados sobre os respectivos elementos. Estes dados podem ser o código HTML

Tabela 4.1: Tabela de comparação da aplicação Calendar antes e depois de convertida

Aspecto da aplicação antes da conversão		
Internet Explorer 7	Firefox	Safari
		
Aspecto da aplicação após a conversão		
Internet Explorer 7	Firefox	Safari
		

contido nesses elementos, os nós de texto contidos nesses elementos, o nó pai desses elementos, a janela onde ele está, etc.

A tabela 4.3 mostra algumas colecções comumente utilizadas em JavaScript para fazer operações sobre um conjunto de elementos.

A tabela 4.4 mostra alguns objectos específicos que não se incluem noutras categorias. Estes objectos estão normalmente associados ao documento, mas também podem ser parte de elementos HTML.

A tabela 4.5, aquela que será mais extensa, demonstra alguns métodos reconhecidos pelo IE e a sua respectiva alternativa, se esta existir. Estes métodos servem para executar operações sobre elementos HTML, a janela ou o documento em si.

A tabela 4.6 mostra algumas propriedades CSS exclusivas da Microsoft e respectivos equivalentes em Firefox, se existirem, a maior parte destas propriedades são invocadas com o prefixo `-ms-` a partir do lançamento do Internet Explorer 8. Os campos sem

Tabela 4.2: Conversão de propriedades JavaScript

Internet Explorer	Firefox	Notas
.innerText	.textContent	Safari utiliza innerText tal como IE. Existe a possibilidade de usar .innerHTML que tendo como objectivo lidar com HTML dentro de uma tag e não só com texto, também suporta o texto e, como tal, funciona perfeitamente em operações efectuadas só sobre texto.
.outerText	nodeValue	necessário remover o nó se função for usada para definir um novo valor
.outerHTML	nodeValue	necessário remover o nó se função for usada para definir um novo valor
.parentWindow	.defaultView	
.sourceIndex	<pre>var e=document.getElementsByTagName('*'); for (var i=0,l=e.length;i<l;i++) e[i].sourceIndex=i;</pre>	Executar este código antes da execução do <i>script</i> para atribuir a propriedade .sourceIndex a todos os elementos.
.srcElement	.target	.target é standard, mas IE não reconhece. Se utilizados em conjunto deve-se verificar qual deles existe.

correspondência em Firefox não têm alternativa conhecida de momento, pelo que a sua utilização na criação de uma página web é desaconselhada. De uma maneira geral, a conversão de CSS não daria grandes resultados práticos, pois poucas propriedades têm alternativa.

A tabela 4.7 mostra alguns eventos exclusivos da Microsoft e respectivo equivalente standard ou próprio do Mozilla Firefox.

A tabela 4.8 mostra algumas das alterações relativas à mudança de contexto de HTML Components para o documento principal de uma página HTML.

Ainda relativamente à conversão de código, existem outros aspectos que são importantes referir:

- O Mozilla Firefox não aceita o tratamento de propriedades de elementos da mesma forma como são tratados os atributos, sendo por isso necessário utilizar a função `getAttribute()` ou a colecção `attributes` para aceder a atributos customizados em Firefox. Este comportamento também se verifica no Safari.
- Apesar de poder funcionar, é recomendado não referenciar directamente os elementos pelo seu id definido em HTML. Ou seja, no caso de existir um elemento `<li id="listitem">`, esse elemento não deve ser utilizado em *script* na forma `listitem.operação`, sendo preferível utilizar a função `document.getElementById("listitem")` para criar uma variável que sirva para fazer operações sobre o elemento.

Tabela 4.3: Conversão de colecções JavaScript

Internet Explorer	Firefox	Notas
.all	.getElementsByTagName("body")[0].getElementsByTagName("*");	
.children	.childNodes	Não são exactamente equivalentes. <code>children</code> lista os objectos filhos, enquanto que <code>childNodes</code> lista os elementos filhos, bem como o texto entre eles. Firefox 3.5 já suporta <code>children</code> , tal como os outros <i>browsers</i> mais populares, incluindo Safari.
.behaviorUrns		Relacionados com HTML Components, não existem em Firefox.
.namespaces		Relacionados com HTML Components, não existem em Firefox.

- A variável `event`, muito utilizada a nível de *JavaScript* para lidar com eventos, não é reconhecida em Firefox sem ser definida antes de ser referenciada. O que isto significa é que é necessário passar a variável global `event` a todas as funções que utilizem essa variável.

4.4 Estratégia de Conversão

Deste estudo sobre conversão de aplicações específicas do Internet Explorer para outros *browsers*, como Firefox e Safari, foi possível extrair uma estratégia que deverá ser aplicada na conversão automática de aplicações.

Essa estratégia passará por detectar a existência de referências a HTML Components no conteúdo inteiro do documento, quer seja feita através de CSS, quer seja feita no HTML propriamente dito, utilizando a *tag* `?IMPORT`. Efectuada essa detecção deve ser lido o conteúdo dos respectivos ficheiros HTC e guardada essa informação de forma a que possa ser aplicada no futuro. Todo este processo deve decorrer antes de se promover qualquer alteração na página.

O documento deve depois voltar a ser percorrido, mas desta vez efectuando as conversões à medida que ele é percorrido. As conversões devem incluir:

- Traduzir *scripts* incluídos no documento principal.
- Traduzir folhas de estilo incluídas no documento principal.
- Incluir e traduzir *scripts* em ficheiros externos.
- Incluir e traduzir *folhas de estilo* em ficheiros externos.
- Adaptar *tags* associadas a HTC.
 - Adicionar atributos de eventos e respectivos *handlers*.

Tabela 4.4: Conversão de objectos JavaScript

Internet Explorer	Firefox
.currentStyle	.style
obj.currentstyle[prop];	window.getComputedStyle(obj, null).getPropertyValue(prop);
.runtimeStyle	.style
.selection	window.getSelection()

- Incluir estilos, *scripts* e corpos HTML definidos em HTC dentro ou após a *custom tag*.
- Traduzir esses mesmos estilos e *scripts*.

A figura 4.1 mostra os passos a tomar para efectuar uma conversão de HTML específico de IE para Firefox.

Dependendo da complexidade da aplicação, estas regras podem não ser suficientes para uma conversão completa da funcionalidade e aspecto da página, sendo necessário, através de testes práticos, verificar se a página funciona e agir onde necessário para terminar a conversão com sucesso.

4.5 Resumo e Conclusões

A conversão de aplicações *Web* específicas do Internet Explorer para outros navegadores populares como Firefox ou Safari implica agir sobre vários aspectos da aplicação. Esses aspectos podem incluir controlos ActiveX, *scripts* VBScript e JScript, CSS, HTC e outras tecnologias menos relevantes não abordadas no relatório.

As características da aplicação “Coolbiz - BackOffice” tornam a conversão mais relevante a nível de JScript e HTC, e por essa razão foram usados exemplos que reflectiam a orientação do trabalho por essas tecnologias. A conversão desses exemplos foi concluída com sucesso, tendo sido definida uma estratégia de conversão para aplicar a páginas HTML que contenham tais características.

A conversão de HTCs não é linear, dependendo do conteúdo de cada componente e do tipo de aplicação que têm nas páginas *web*. Isso exige uma estratégia de detecção das necessidades da aplicação e uma acção de acordo com essas mesmas necessidades. A própria conversão de JavaScript vai sofrer destas particularidades, uma vez que é necessário adaptar o código durante a mudança de contexto em que ele é executado.

Conversão de Código

Tabela 4.5: Conversão de métodos JavaScript

Internet Explorer	Firefox	Notas
add(opção, índice)	add(opção, seleção.options[índice])	
attachEvent(sEvent, fpNotify)	addEventListener(sEvent, fpNotify, useCapture)	IE não suporta o segundo método, o <i>standard</i> , para efeitos de conversão pode-se assumir o último parâmetro como false. Para ter compatibilidade com IE ao mesmo tempo que outros <i>browsers</i> é necessário criar uma condição para verificar a disponibilidade dos métodos
createStyleSheet(ficheiroCSS)	<pre>var styles = ficheiroCSS; var newSS=document.createElement('link'); newSS.rel='stylesheet'; newSS.type='text/css'; newSS.href=escape(styles); document.getElementsByTagName("head")[0].appendChild(newSS);</pre>	
createPopup()	createElement("div")	Não existe o conceito de Popup noutros browsers, alternativa seria criar uma div com posição absoluta com o estilo ou funções associadas ao Popup, não é trivial fazer isto pois Popup pode servir para qualquer fim.
createRange()	getRangeAt(0)	métodos usados para trabalhar com selection
getAdjacentText(sWhere)	<pre>'beforeBegin': previousSibling.textContent; 'afterBegin': childNodes[0].textContent; 'beforeEnd': childNodes[element.childNodes.length - 1].textContent; 'afterEnd': nextSibling.textContent;</pre>	código a executar depende do valor de sWhere
insertAdjacentElement(position, newElem);	<pre>'beforeBegin': parentNode.insertBefore(newElem, this); 'afterBegin': insertBefore(parsedNode, this.firstChild); 'beforeEnd': appendChild(parsedNode); 'afterEnd': parentNode.insertBefore(parsedNode, this.nextSibling);</pre>	código a executar depende do valor de position. This representa o elemento em que o método foi invocado.
insertAdjacentHTML	”	Aplicam-se as mesmas regras que insert Adjacent Element.
insertAdjacentText	”	Aplicam-se as mesmas regras que insert Adjacent Element.
moveRow(i,j)	<pre>table.insertRow(j+1); table.rows[j+1].innerHTML = table.rows[i].innerHTML; table.deleteRow(i);</pre>	
pasteHTML	replace	
showModelessDialog	open	é possível utilizar open para substituir showModelessDialog, com funcionalidade dependent activada
addRule	insertRule	
removeRule	deleteRule	
ActiveXObject("Mi-crosoft.XMLHTTP")	XMLHttpRequest()	

Conversão de Código

Tabela 4.6: Conversão de propriedades CSS

Internet Explorer	Firefox
behavior	-moz-binding
background-position-x: xpos	background-position (xpos, ypos)
background-position-y: ypos	background-position (xpos, ypos)
box-sizing	-moz-box-sizing
layout-flow: "vertical-ideographic"	writing-mode: "tb-rl"(também incompatível, mas irá ser implementado em Firefox num futuro próximo)
layout-grid	
layout-grid-char	
layout-grid-line	
layout-grid-mode	
layout-grid-type	
line-break	
text-align-last	
text-autospace	
text-justify	
text-kashida-space	
text-overflow	
text-underline-position	
word-break	
word-wrap	
writing-mode	
block-progression	
overflow-x (já suportado nas últimas versões de Firefox)	-moz-scrollbar-horizontal
overflow-y (já suportado nas últimas versões de Firefox)	-moz-scrollbar-vertical
accelerator	
interpolation-mode	
scrollbar-3dlight-color	
scrollbar-arrow-color	
scrollbar-base-color	
scrollbar-darkshadow-color	
scrollbar-face-color	
scrollbar-highlight-color	
scrollbar-shadow-color	
scrollbar-track-color	
zoom	

Tabela 4.7: Conversão de eventos

Internet Explorer	Firefox
activate	DomFocusIn
beforecopy	
beforecut	
beforepaste	
deactivate	DomFocusOut
focusin	DomFocusIn
focusout	DomFocusOut
hashchange	
mouseenter	mouseover
mouseleave	mouseout
mousewheel	DOMMouseScroll
propertychange	DomAttrModified

Tabela 4.8: Conversão de contexto

HTML Components	Documento Original
document.body	element
element.document	document
this (depende do contexto original de this)	element
(em algumas ocasiões podem ser referidas propriedades directamente)	element

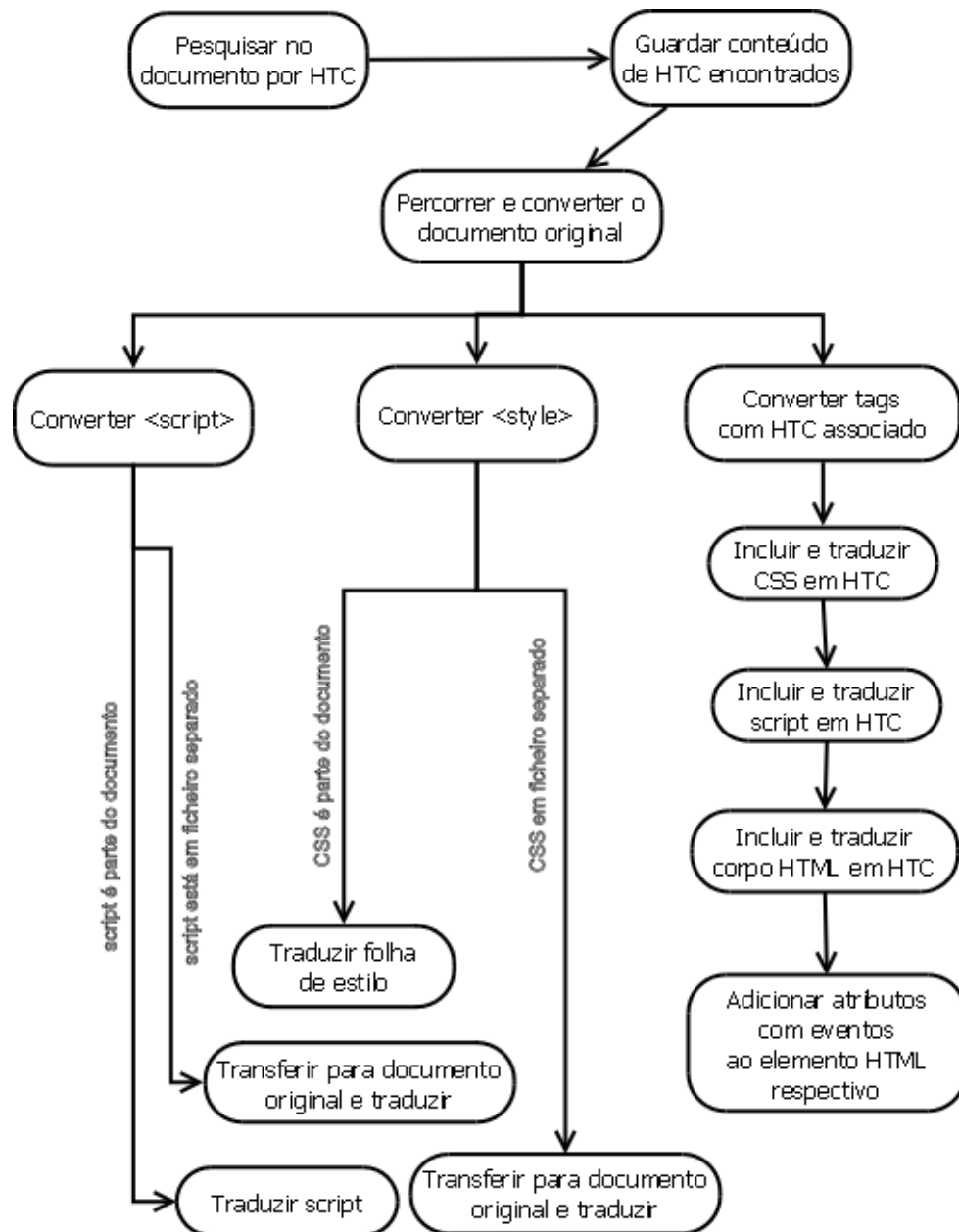


Figura 4.1: Esquema da estratégia de conversão adoptada

Capítulo 5

Implementação

Neste capítulo é descrita a implementação dos protótipos do conversor automático. O conversor foi implementado em .NET, com recurso à linguagem C# e utilizando o ambiente de desenvolvimento integrado Microsoft Visual Studio 2005. São também utilizadas as ferramentas FreeTXL e ANTLRWorks, e duas bibliotecas .NET desenvolvidas por terceiros: Majestic-12 HTML Parser e ANTLR.

Durante o período de estudo desta tese foram desenvolvidas duas ferramentas de conversão automática distintas. Estas duas ferramentas diferem, acima de tudo, na forma como aplicam a transformação, sendo que as classes que são responsáveis por essa mesma conversão são idênticas, sofrendo apenas pequenas alterações que serão explicadas à frente. Os dois processos essenciais da transformação utilizada em cada versão são a conversão de JavaScript e o tratamento de ficheiro HTC de forma a tornar o seu conteúdo compreensível para o Firefox.

A primeira das versões do conversor automático consiste num pequeno programa que transforma código HTML contido em qualquer ficheiro desse tipo, e que permite guardar, após a conversão, o código alterado num novo ficheiro. A segunda versão é específica para aplicação à “Coolbiz - BackOffice” e trata-se de um filtro aplicado no fluxo de dados entre servidor e *web browser* que efectua as alterações ao código em tempo real, caso esse navegador seja outro que não o IE.

5.1 Conversão de JavaScript

Nesta secção é explicada a implementação do processo de transformação de JavaScript em particular.

A conversão de JavaScript decorre em duas fases. A primeira recorre à biblioteca ANTLR [29] para C# e irá ser utilizada para modificar o script de forma a poder ser

utilizado no contexto do documento HTML e não no de um HTC, pelo que isto só é necessário quando o *script* está incluído num componente. A segunda fase trata de converter o script para Firefox/Safari, sendo aplicável a qualquer *script*. Nesta fase é aplicada a ferramenta FreeTXL [37].

Esta separação de processos permite uma utilização muito mais clara das ferramentas e de quando elas devem ser aplicadas, facilitando a tarefa do programador. No entanto, ela não acontece apenas por essa razão, mas sim porque o ANTLR permite uma utilização mais fácil de dados que são acessíveis no código C#, que são utilizados na aplicação das respectivas regras de mudança de contexto e não na fase de ganhar compatibilidade com outros navegadores.

A figura 5.1 mostra as ferramentas utilizadas durante a conversão de JavaScript contido em HTML Components.

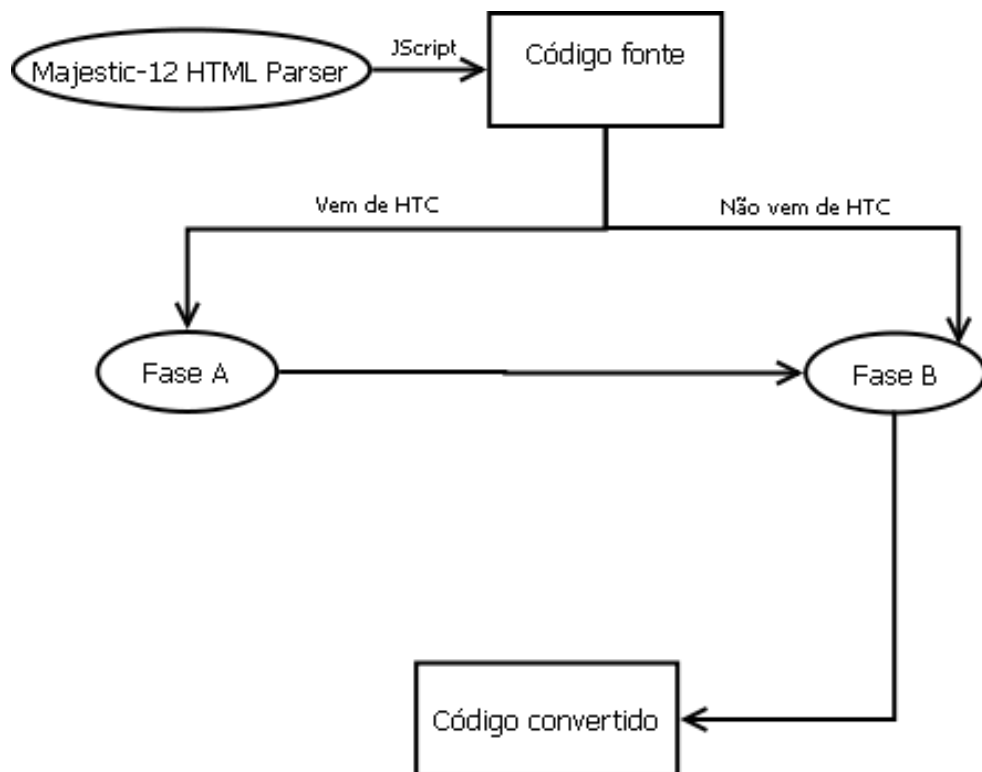


Figura 5.1: Esquema de conversão de JavaScript.

5.1.1 Fase A

No contexto da conversão em JavaScript, esta primeira fase, que recorre a ANTLR, é utilizada para alterar alguns parâmetros específicos dos *scripts* em HTC.

A gramática utilizada foi uma das gramáticas de ECMAScript/JavaScript disponibilizadas pela comunidade ANTLR no sítio oficial da ferramenta. No entanto, foi preciso

proceder a algumas alterações para poder ser utilizada em C#, uma vez que ela tinha sido construída para ser utilizada em Java. Como o ANTLR permite definir funções na linguagem de destino dentro da definição da gramática, foi necessário alterar não só o parâmetro que indica a linguagem de destino para usar C# de .NET 2.0, mas também foi preciso proceder a várias alterações na declaração dessas funções, de métodos ANTLR que são invocados de forma diferente em Java e em C#, bem como palavras exclusivas do Java que não existem em C#. Devido à similaridade entre as duas linguagens foi, no entanto, possível concretizar estas alterações sem um esforço dramático. Estas alterações à gramática foram feitas na ferramenta ANTLRWorks.

As regras aplicadas nesta fase resumem-se, essencialmente, à declaração de variáveis `element` nas funções que utilizam essa mesma variável, transformação de referências a características dos elementos que eram feitas sem indicar esse contexto no HTC, eliminação de algumas características relacionadas com a configuração dos HTC [20] e transformação de referências ao contexto do documento original, feitas de forma diferente no HTC.

Para além destas alterações, é necessário modificar os nomes das funções e variáveis utilizadas, porque cada página do Coolbiz utiliza múltiplos componentes que têm funções com designações em comum. É necessário garantir que o *browser* consegue distinguir essas funções, o mesmo se passando com as variáveis utilizadas por esses *scripts*. Para isso, passa a ser utilizado como prefixo dessas funções e variáveis, as designações que funcionam como *namespaces* das etiquetas personalizadas de cada HTC. Devido à criação de árvore de código JavaScript pelo ANTLR, é possível verificar que identificadores são funções e variáveis e existem nesse mesmo *script*. Aqueles que satisfizerem estas condições são alterados com a adição do prefixo.

Ainda sobre a utilização do ANTLR, ela é feita da seguinte forma:

- O ANTLR recebe o script na forma de cadeia de caracteres.
- Utilizando a gramática escolhida e respectivos analisador léxico e sintáctico gerados, é criada uma árvore de sintaxe que representa esse *script*.
- São efectuadas as alterações sobre a árvore.
- Depois de efectuadas todas as alterações, a árvore é processada por uma classe desenvolvida para o efeito que trata de transformar a árvore de volta a código fonte, já com as alterações efectuadas.
- É devolvida uma cadeia de caracteres com o novo código ao conversor principal.

A classe que trata de transformar a árvore em código fonte (*printer*) tem uma estrutura muito simples, mas é bastante extensa, uma vez que é necessário saber como imprimir cada tipo de *token* individualmente.

A figura 5.2 ilustra as transformações descritas anteriormente.

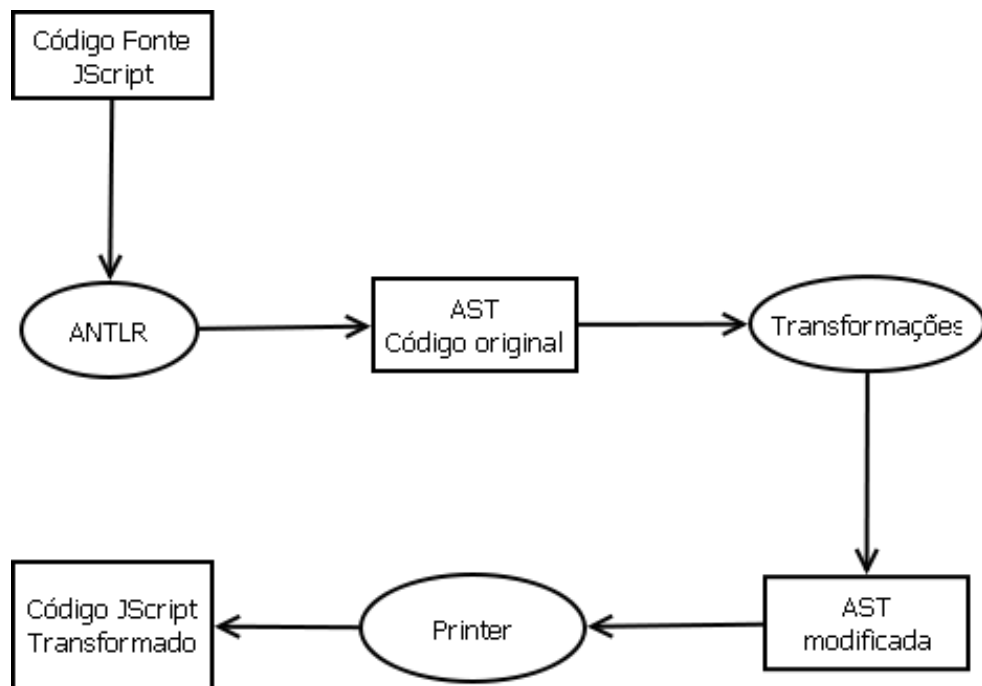


Figura 5.2: Esquema das etapas ocorridas durante a Fase A.

5.1.2 Fase B

Nesta fase B de conversão, que será a única fase a executar no caso da conversão ser feita sobre um *script* não incluído num HTC, deverão ser feitas as alterações que correspondem a uma tradução de JScript para JavaScript, tornando assim o código compatível com Firefox. Para isso, estando já definidas as gramáticas e regras respectivas, o programa FreeTXL é iniciado em segundo plano para aplicar as transformações ao código.

Preparado para este tipo de funções, o FreeTXL imprime para o *standard output* do sistema operativo apenas o código traduzido, deixando as restantes mensagens de informação sobre a conversão e execução do programa para o *standard error*. Isto acaba por simplificar a tarefa de implementação do conversor TXL a nível de .NET, sendo apenas necessário invocar o processo e ler a saída de texto gerada por esse mesmo processo para obter o código transformado.

A presença do FreeTXL no computador que executa o conversor é, logicamente, essencial para que este protótipo possa funcionar.

A sequência de execução neste caso é a seguinte:

- O script a transformar é guardado num ficheiro temporário.
- O executável principal do FreeTXL é invocado passando o ficheiro temporário como argumento.

- Recorrendo à gramática JavaScript respectiva e ao ficheiro onde estão definidas as regras, o FreeTXL transforma o código.
- O novo código fica disponível a nível de .NET através da leitura do *standard output* do processo respectivo.
- É devolvida uma cadeia de caracteres com o novo código ao conversor principal.

A figura 5.3 ilustra as transformações descritas anteriormente.

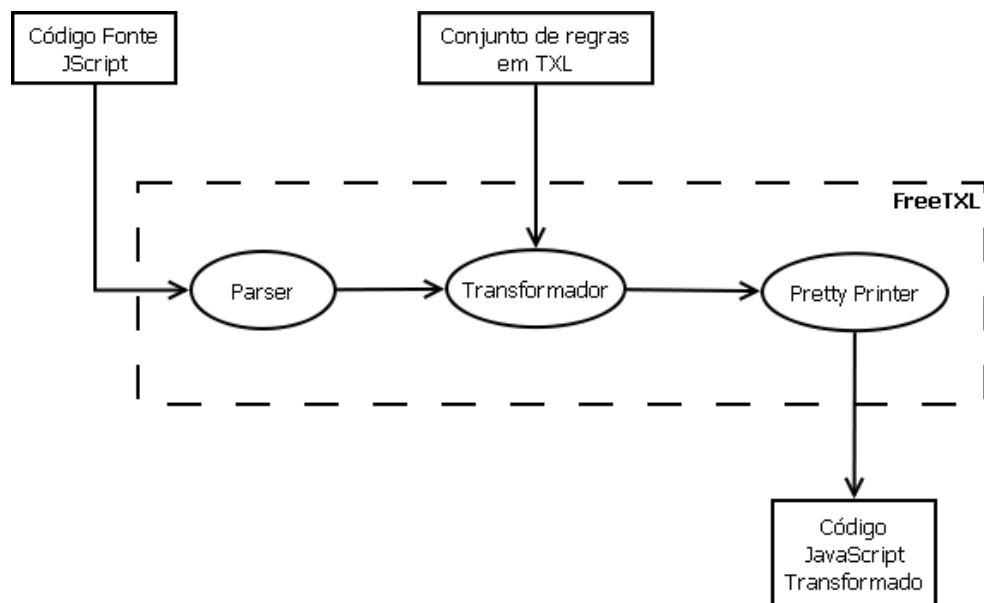


Figura 5.3: Esquema das etapas ocorridas durante a Fase B.

Neste momento, estão já definidas 35 regras de conversão em linguagem TXL neste conversor, sendo que 20 dessas regras são para aplicação geral a qualquer código JScript incompatível e, por isso, de extrema relevância na conversão de uma página específica do IE para outros navegadores de Internet. De seguida são apresentados alguns exemplos de regras implementadas em linguagem TXL.

• replaceSrcElement

Esta regra é um exemplo muito simples de uma conversão automática de código específico de IE para Firefox. O código respectivo é o seguinte:

```

rule replaceSrcElement
  replace [id]
    srcElement
  by
    target
end rule
  
```

O objectivo desta porção de código é substituir cada identificador do código que tenha o nome `srcElement`, pelo identificador `target`. Para atingir esse objectivo, é definido na linguagem que para uma expressão do tipo `id` a que corresponda o texto `srcElement` ele passará a ser `target`.

- **addEvent2Function**

Esta regra está relacionada com a necessidade de passar a variável `event` a funções que necessitam da mesma em Firefox. O código que a implementa é:

```
rule addEvent2Function
  replace $ [js_function_declaration]
    'function I [id] '( args [list id] ') { B [js_subscope] }'
  by
    'function I '( args [, 'event] ') { B }'
end rule
```

Esta transformação já exige alguma explicação. Para conseguir o objectivo pretendido é definido em TXL uma substituição em expressões do tipo *js_function_declaration*. A estrutura destas expressões é a seguinte: palavra `function` no início, seguida de um identificador, a que é atribuído a variável `I`, seguida duma lista de argumentos separados por vírgulas, lista à qual é atribuída o nome `args` e que, por sua vez, é seguida do bloco da função, entre chavetas. Estes blocos são do tipo `js_subscope` na gramática JavaScript utilizada.

A expressões que tenham esta estrutura, que no caso são todas as funções JavaScript são substituídas por uma função com estrutura em tudo idêntica, mas adicionando à variável `args` a palavra `event`, utilizando o operador de TXL `, .`

O operador `$`, colocado após `replace`, indica que esta transformação só deve ser aplicada uma vez a cada função. Sem este operador as funções iriam ser transformadas infinitamente.

- **replaceAttachEvent**

Esta é uma regra muito importante, pois é a regra que substitui a função específica de IE utilizada para adicionar eventos a determinados elementos. O código utilizado é o seguinte:

```
rule replaceAttachEvent
  replace [repeat js_component_selector]
    . attachEvent (args [list js_expn])
  construct False [js_expn]
    false
  construct newArgs [list js_expn]
    args [, False]
```

Implementação

```
by
  . addEventListener (newArgs [removeOnChar] [removeOnString])
end rule
```

Esta regra é aplicada a uma repetição de 1 ou mais expressões do tipo `js_component_selector`, que são expressões em que é usado o ponto em JavaScript para aceder a um método, ou propriedade de um determinado objecto. Uma das particularidades desta transformação é recorrer ao operador `construct` de TXL. Este operador permite criar expressões de tipos diferentes àquele que está a ser transformado que podem ser usadas na conversão. Neste caso, é criada uma expressão com a palavra `false` que é usada noutra `construct` na criação de uma nova lista de argumentos para a invocação do método `addEventListener`. Esta transformação é necessária para que possam ser aplicados as regras `removeOnChar` e `removeOnString` à lista de argumentos a utilizar já completa. Estas regras promovem a transformação de literais que se referem a eventos para que percam os caracteres “on”. Por exemplo, `addEventListener` reconhece “click”, `attachEvent` reconhece “onclick”.

• makeMenusWork

Esta regra já se trata de um exemplo de algo que é criado especificamente para aplicação a um *script*. No caso, relacionado com o menu principal do Coolbiz, do qual serão vistos resultados de conversão mais à frente. O código desta transformação é o seguinte:

```
rule makeMenusWork
  replace [repeat js_declaration_or_statement]
    childMenu.style.left=clickedMenuItem.getBoundingClientRect().left-1+shiftLeft;
    childMenu.style.top=clickedMenuItem.getBoundingClientRect().bottom+1+shiftTop;
    Outros [repeat js_declaration_or_statement]
  by
    childMenu.style.left=clickedMenuItem.getBoundingClientRect().left;
    childMenu.style.top=clickedMenuItem.getBoundingClientRect().bottom;
    Outros
end rule
```

Como é possível verificar, em relação aos exemplos anteriores, o conteúdo desta regra é muito mais preciso, não existindo quase a definição de variáveis no código a converter, pois ele deve ser reconhecido tal como está. A única definição de variável é utilizada para referir o que vem a seguir às duas linhas que se pretende transformar. A precisão de conversão que o TXL permite manter estas transformações no mesmo nível que as mais gerais, sem prejudicar a ferramenta, pois seria necessário que outra página a passar pelo conversor utilizasse um trecho de código exactamente igual para que esta alteração fosse aplicada a ela. E se tal acontecesse, a probabilidade

de precisar mesmo dessa conversão seria grande, pois a coincidência deveria ser motivada por código criado com o mesmo tipo de intenção.

5.2 Tratamento de HTC

A conversão tendo em conta as especificidades dum HTC não se limita à tradução de código JavaScript. É necessário também saber lidar com os parâmetros que um HTC define e atribui a determinado elemento. Estes parâmetros incluem eventos, métodos, propriedades a passar para o documento original. É preciso saber conciliar estes aspectos e é nesse particular que o analisador gramatical de HTML da Majestic-12 pode ser útil.

O Majestic-12 HTML Parser, como ferramenta capaz de analisar HTML, também pode ser utilizado com sucesso para o tratamento de HTC. Ele é utilizado primeiro a nível do documento HTML original para encontrar referências a HTC dentro do documento em *tags* `?IMPORT` e depois, quando percorre o conteúdo dos HTC, ele é capaz de retornar os valores definidos neste ficheiro relativamente a métodos, eventos, propriedades, componente, etc.

Assim, a cada HTC novo que encontra, o Parser é aplicado ao ficheiro respectivo e guarda tudo o que encontrar, distinguindo devidamente o que são propriedades, métodos, eventos, etc., guardando também todos os seus atributos em estruturas do tipo lista de *strings*. Para além de guardar dados sobre esses elementos específicos dos HTC, ele guarda também todo o conteúdo em termos de CSS, JScript e corpo do HTML no componente HTC.

Apenas alguns destes aspectos estão implementados, de facto, no conversor automático, porque ainda não se tornou relevante tratar de outras características dos HTC de forma assistida, como por exemplo eventos personalizados em HTC. Assim, aquilo que, sobre os HTC, vai ser guardado e também utilizado é a *custom tag* respectiva, que inclui também o *namespace* utilizado, o *script* desse HTC, a folha de estilo do HTC, o HTML do HTC, o nome dos eventos que devem ser disparados, o nome das funções associadas a esses mesmos eventos e, não relacionado com os HTC em si, mas com a sua implementação na conversão, é guardada um valor do índice que indica a ordem pela qual o HTC foi processado.

Este identificador é usado, por exemplo, para poder saber a que elemento numa lista ir buscar os dados para fazer uma conversão, quantas vezes numa página esse HTC já foi convertido, onde guardar dados retirados da *tag* de abertura para que se possa aplicar correctamente após a *tag* de fecho, entre outras funcionalidades. Resumindo, é utilizado para fazer vários tipos de controlo sobre a inserção de código novo no documento.

5.3 Primeira Versão

Esta primeira versão foi motivada pela necessidade de testar e desenvolver as técnicas e regras pensadas para a conversão automática das páginas *Web* utilizadas na fase de estudo anterior, a *Hilite* e a *Calendar*. Assim, o objectivo é aplicar a conversão sobre um ficheiro HTML, que pode conter, ou não, referências a ficheiros HTC e permitir guardar o resultado dessa conversão noutro ficheiro distinto.

A interface da aplicação é intuitiva e simples, consistindo de duas *RichTextBox* em que uma representa o código fonte original e a outra irá conter o código fonte convertido. O ficheiro de origem é carregado através de um *OpenDialog* comum, incluído no menu principal do programa, e por ser representado numa caixa de textos, o utilizador pode efectuar qualquer alteração ao código antes de fazer a conversão. Depois de feita a conversão existe também a possibilidade de alterar o código final, introduzindo alguma alteração que seja necessária. Para gravar este novo ficheiro o método é semelhante a qualquer outra aplicação Microsoft Windows, utilizando um *SaveDialog*.

A figura 5.4 mostra o resultado desta primeira implementação de conversão assistida, desenvolvida para aplicar a código fonte HTML contido em qualquer ficheiro de texto.

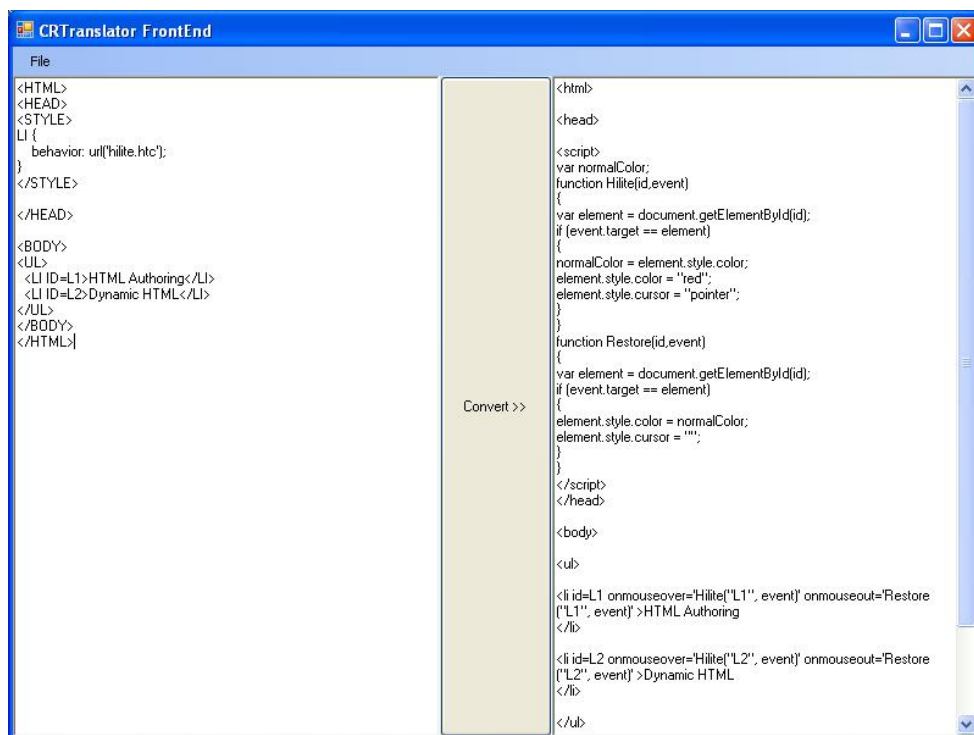


Figura 5.4: Primeira versão da ferramenta aplicada ao exemplo Hilite.

Esta versão prevê a possibilidade do código fonte invocar um número qualquer de ficheiros HTC seja no próprio documento, ou sejam invocados dentro de outros componentes. Por isso, verifica que ficheiros HTC são referenciados recursivamente e guarda em

memória todos os dados relevantes relativos a esses *HTML Components*. Esta verificação só é possível em CSS graças a uma gramática CSS para o ANTLR utilizada para o efeito e à leitura das *tags* `?IMPORT` feita pelo Majestic-12 HTML Parser.

Tendo este conhecimento acerca dos HTC, o ficheiro é percorrido pelo *parser* de HTML, que vai reproduzindo o código original no código final, promovendo as seguintes alterações:

- Alterações dentro do cabeçalho do HTML (etiqueta `HEAD`):
 - Inclui todo o conteúdo das folhas de estilo de todos os HTC na etiqueta `STYLE`. Se esta não existir, cria-a.
 - Inclui todo o conteúdo dos *scripts* de todos os HTC na etiqueta `SCRIPT`, procedendo às devidas alterações a esses *scripts*. Se a etiqueta não existir, cria-a.
- No corpo do HTML, para cada etiqueta que esteja associada a um HTC:
 - Adiciona atributos definidos em HTC, para associação de eventos a funções, alterando os nomes dessas funções se necessário e incluindo os argumentos respectivos.
 - Inclui todo o conteúdo do elemento body do HTC respectivo e altera a etiqueta para ser uma `div`, se este elemento for associado a uma *tag* customizada, ou seja, quando o HTC não serve apenas para aplicar um estilo ou um comportamento a um elemento padrão do HTML.

5.4 Segunda Versão

A segunda versão é mais recente que a primeira, e surgiu numa fase em que interessava aplicar os conhecimentos adquiridos com a experiência anterior à aplicação “Coolbiz - BackOffice”.

Esta versão é por um lado mais simples que a primeira, mas por outro lado mais avançada e melhor estruturada. Isto porque uma vez que a aplicação Coolbiz é desenhada com alguma coerência entre as várias páginas, não existe a necessidade de prever diferentes situações que motivam transformações de código fonte. O que isto significa é que, por exemplo, não existe necessidade de verificar se o HTC é invocado através de CSS porque isso nunca acontece nesta aplicação. Pela mesma razão, todas as transformações irão ser sobre elementos personalizados, não existindo necessidade de efectuar uma distinção entre o tipo de alterações que é feito num elemento ou noutro. Esta eliminação de complexidade, apesar de não ser necessária, torna o código do conversor mais legível e cingido ao que interessa, sendo de mais fácil compreensão para quem queira desenvolver e adicionar novas transformações.

Esta versão é aplicada ao Coolbiz em tempo de execução como um filtro à resposta do servidor para o *web browser*, que transforma cada pedaço de código enviado de forma a que a página gerada seja compatível com outros navegadores que não o IE. De notar que este processo não ocorre quando o navegador é identificado como sendo o IE, na versão actual do protótipo.

Uma vez que a aplicação Coolbiz é desenvolvida com tecnologia ASP.NET, em que tipicamente é utilizado um ficheiro com extensão .aspx para definir a estrutura da página e um ficheiro de código .NET (C# ou Visual Basic) associado a cada página para ser executado a nível do servidor, o código HTML de cada página nunca é transparente sem o servidor a correr. Uma das vantagens desta abordagem para aplicar o conversor é que se garante que o HTML sobre o qual é feita a conversão é mesmo o HTML que o servidor envia para o cliente *web*. Mas esta abordagem também apresenta desvantagens evidentes. Uma delas é a performance, o facto de efectuar uma conversão cada vez que é feito um pedido pelo cliente consome algum processamento que seria desnecessário se as páginas já estivessem convertidas previamente. Outra desvantagem é que a totalidade do código a enviar para o cliente não está, garantidamente, disponível no mesmo tempo de execução. Dependendo do tamanho do *buffer* de envio da resposta a quantidade de código disponível a cada passagem.

Este facto obriga a alterações na forma como o conteúdo dos HTC é inserido do documento principal, pois já não existe garantia de que se conheça todos os componentes HTC utilizados por uma página específica antes desta ser totalmente processada, inserir as funções JavaScript e as definições de CSS no cabeçalho do HTML torna-se impossível. Assim, o que esta versão faz é tirar partido da capacidade que os navegadores têm de ler *scripts* e pedaços de folhas de estilo em qualquer posição do HTML.

Para garantir que o código final está bem estruturado, este conversor verifica em todos os pedaços de código se existem referências a ficheiros HTC, guardando o conteúdo dos mesmos em caso afirmativo, se já não guardou em memória os dados desses mesmos HTC numa fase anterior. Para além desta estratégia de verificação de HTC, o conversor só actua junto das etiquetas dos dos componentes personalizados da seguinte forma:

- Etiqueta de abertura
 - Guarda os atributos definidos dentro dessa *tag*. Estes atributos são usados no JavaScript desse HTC com diversas aplicações e são definidos nos HTC como elementos `PROPERTY`.
 - Inclui eventos e respectivas funções definidos no HTC dentro da etiqueta de abertura, como atributos.
- Etiqueta de fecho
 - Incluído o corpo HTML definido no documento HTC.

- Incluída a folha de estilo definida no documento HTC.
- Incluído o *script* vindo do HTC com as devidas transformações aplicadas. A este JavaScript é adicionado no início os atributos do elemento guardados anteriormente e no final do *script* é adicionada uma invocação à função que estava associada ao evento *oncontentready* no HTML Component.

Para além destas transformações, o protótipo também prevê a aplicação do conversor de JavaScript a *scripts* incluídos no documento HTML principal ou a *scripts* externos referenciados nessa mesma página. Para isso, é aplicado o conversor de JavaScript quando o HTML Parser detecta a existência de uma tag *script* na página. Quando essa etiqueta, em vez de conter uma porção de código, contém atributos que indicam a referência a um *script* externo, normalmente guardado em ficheiros com extensão .js, o protótipo trata de ler o conteúdo desse ficheiro, aplicar-lhe o transformador, e inseri-lo no documento principal, eliminando a referência a esse ficheiro.

A limitação de ser aplicado a código parcial apenas, impossibilita também a correcta implementação de regras para referenciar atributos com a função `getAttribute` ou a colecção `attributes`. Isto acontece porque sem acesso à totalidade do código, não existe a possibilidade de verificar com garantia absoluta se determinada referência em JavaScript diz respeito a um atributo HTML ou não.

5.5 Resumo

Foram implementados dois protótipos de ferramentas ao longo da duração da tese. Estes protótipos serviram não só para auxiliar o estudo efectuado sobre o tema das conversões automáticas de páginas *Web*, mas também para mostrar que tipo de aplicações um estudo destes pode ter.

A implementação destes protótipos divergiu ligeiramente a nível de estratégia, mas aplicam essencialmente o mesmo nível de conhecimento, sendo perfeitamente válida a sua definição como implementações objectivas retiradas deste estudo.

As ferramentas utilizadas para levar a cabo esta implementação revelaram-se bastante competentes nesta tarefa específica, sendo que a limitação das suas funcionalidades, prende-se apenas com o período de tempo disponível para esta mesma implementação.

A quantidade de código desenvolvido não foi exagerada, limitando-se a alguns milhares de linhas de código fonte, o que faz crer que não existirá dificuldade de expansão das ferramentas já implementadas. Estes protótipos deverão permitir uma inclusão fácil de cada nova regra de transformação que seja descoberta para a conversão de JavaScript, bastando implementar em TXL uma nova regra. Para cada *token* que ainda não esteja previsto no *printer* do ANTLR, basta adicionar um novo *case* que implemente essa impressão. Se for necessário passar a utilizar o ANTLR para fazer transformações que não

Implementação

se limitam à mudança de contexto de HTC para o documento HTML, tal também é facilmente implementado sem prejudicar o restante trabalho desenvolvido.

Capítulo 6

Avaliação do Protótipo

Neste capítulo são apresentados os resultados obtidos e é feita uma apreciação da qualidade dos mesmos através do uso de vários dados estatísticos, aplicados a alguns exemplos de conversão.

6.1 Resultados

Nesta secção pretende-se mostrar os resultados da conversão assistida para as aplicações de teste utilizadas anteriormente, bem como os resultados obtidos na aplicação Coolbiz, indicando que aspectos da estratégia utilizada foram mais relevantes em cada caso, e que tipo de intervenções manuais foram necessárias para tornar a página completamente funcional.

6.1.1 Hilite

Devido à sua simplicidade, os resultados obtidos com este pequeno exemplo de incompatibilidade foram satisfatórios a 100%.

A particularidade deste caso é o facto de utilizar CSS para invocar o ficheiro HTC utilizado, tendo sido por isso utilizada a gramática de CSS em ANTLR para detectar esse ficheiro e respectivo elemento HTML associado (LI - item de lista).

Depois da cópia do conteúdo do HTC para o documento original, todas as tecnologias têm alguma intervenção na conversão do documento. No que toca ao JavaScript, tal como idealizado, o ANTLR efectua transformações a nível de árvore sintáctica para modificar o *script* de forma a adaptar-se ao novo contexto em que é inserido. Estas transformações incluem a inserção da declaração da variável `element` e outras transformações relacionadas com essa variável.

As transformações que o TXL aplica, neste caso, são essencialmente de mudança de nome de identificadores e *strings* literais, como a passagem de `srcElement` para `target` ou de `"hand"` para `"pointer"`. A isto adiciona-se a inserção do argumento `event` nas funções utilizadas.

Para completar a transformação é preciso, no entanto, também efectuar alterações a nível do HTML propriamente dito para adicionar as funções a executar quando os eventos são disparados, esta transformação é feita com recurso ao Majestic-12 HTML Parser utilizando os dados do HTC previamente guardados em memória.

As figuras 6.1(a) e 6.1(b) demonstram o que era obtido desta página em Firefox numa fase anterior à conversão e numa fase posterior. A conversão é completa em termos de funcionalidade.

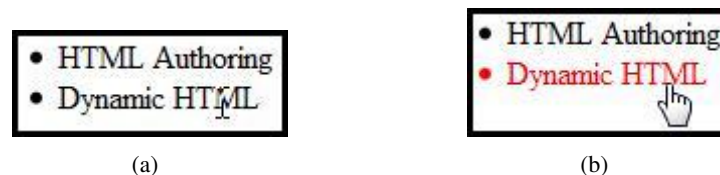


Figura 6.1: Comparação da apresentação de Hilite antes (a) e após (b) a conversão em Firefox

Para consultar o código resultante da conversão automática, consultar o anexo A.3.

6.1.2 Calendar

A aplicação Calendar já tem alguma complexidade e a intervenção manual irá ser necessária.

Esta aplicação recorre não a um, como no exemplo anterior, mas a três ficheiros de componentes HTML diferentes para se tornar funcional. Assim, os dados de todos esses ficheiros são guardados previamente e respectivos *scripts* e folhas de estilo transferidos para o ficheiro original. Estes mesmos *scripts* e folhas de estilo apresentam também a característica de terem funções JavaScript e nomes de classes para aplicação desses mesmos *scripts* com designações em comum. Embora o *namespace* utilizado pelos HTC seja utilizado para fazer essa distinção, o processo que promove a inclusão desse *namespace* no nome das funções e classes não está implementado a nível de CSS, embora tal pudesse ser feito com recurso às bibliotecas ANTLR. Por não estar implementado é necessário modificar o CSS manualmente.

Os *scripts* dos ficheiros HTC referentes a qualquer dia e ao dia actual vão sofrer alterações semelhantes a Hilite a nível de mudança de contexto, como a declaração da variável `element` e a mudança da referência ao corpo do documento para utilizar essa mesma variável. Estes scripts vão necessitar duma pequena alteração manual que se prende com a utilização dum atributo HTML desse elemento como uma propriedade do

objecto em JavaScript. Como foi visto anteriormente, o Firefox não permite isto, mas para fazer a conversão seria necessário pesquisar no documento por cada propriedade do elemento utilizada e verificar se ela é de facto um atributo HTML. Embora possível, esta implementação não foi julgada relevante. Note-se ainda que no conversor a aplicar ao “Coolbiz - BackOffice” não existe garantia que esta técnica seja fiável, como se vai ilustrar na próxima secção. Assim, é necessário modificar as funções `fnInit` para utilizar `element.getAttribute('value')` em vez de `element.value`.

Um aspecto particular desta aplicação que também impede o seu funcionamento normal ao aplicar a conversão assistida, é o facto de no script do componente `CALENDAR` ser criado o código HTML, que tratará de preencher o calendário com os dias do mês respectivo, durante a execução através do uso de cadeias de caracteres. Este código contém as *tags* que fazem referência aos outros HTC, e é portanto nesta porção que devem ser inseridos os eventos a disparar e respectivas funções. No entanto, o HTML Parser não é capaz de detectar sequer este tipo de HTML dentro dum script, e para a gramática JavaScript em ANTLR trata-se, claro, apenas de mais um literal. Esta porção de código, mostrada de seguida, precisa portanto duma intervenção manual extensa.

```
if (dayOfMonth == date) { // current cell represents today's date
    text += '<TD><TODAY:DAY value=' + dayOfMonth + '></TODAY:DAY></TD>';
} else {
    text += '<TD><ANYDAY:DAY value=' + dayOfMonth + '></ANYDAY:DAY></TD>';
}
```

Deve ser convertido, à semelhança do que já tinha acontecido na conversão manual da aplicação, em:

```
if (dayOfMonth == date) { // current cell represents today's date
    text += '<TD><DIV onclick=todayfnShowAppts(\'a\' + dayOfMonth
    + \'\'') oncontentready=todayfnInit(\'a\' + dayOfMonth
    + \'\'') id=a\' + dayOfMonth + \' value=' + dayOfMonth + '></DIV></TD>';
} else {
    text += '<TD><DIV onclick=anydayfnShowAppts(\'a\' + dayOfMonth
    + \'\'') oncontentready=anydayfnInit(\'a\' + dayOfMonth
    + \'\'') id=a\' + dayOfMonth + \' value=' + dayOfMonth + '></DIV></TD>';
}
```

Pela forma particular como este código é criado, no final do *script* é necessário também um código muito particular para que as funções do evento `oncontentready`, não reconhecido em Firefox, sejam executadas correctamente. Esse código é o seguinte:

```
// workaround para oncontentready
var divs = document.getElementsByTagName('div');
for(i=0;i<divs.length;i++){
```

Avaliação do Protótipo

```
if(divs[i].getAttribute("oncontentready"))
{
    ContentReadyString = divs[i].getAttribute("oncontentready");
    argsIndex = ContentReadyString.indexOf("(");
    functionString = ContentReadyString.slice(0,argsIndex);
    argsString = ContentReadyString.slice(argsIndex, ContentReadyString.length);
    argsString = argsString.replace("' ", "");
    argsString = argsString.replace("'", "");
    window[functionString](argsString);
}
}
```

Este excerto de código executa as funções `anydayfnInit` ou `todayfnInit`, respectivamente, depois de preparar os argumentos de forma a serem compreendidos pela invocação do tipo `window[função](argumentos)`.

A figura 6.2 mostra o resultado da aplicação da ferramenta à página, a funcionalidade não é completa uma vez que a porção de código em que é feita a associação entre dias e secções do calendário não está convertida, entre outros aspectos mencionados anteriormente. Para comparação com aspecto original da página, consultar a tabela 4.1. Para verificar o código fonte resultante da conversão automática, ver o anexo B.5.

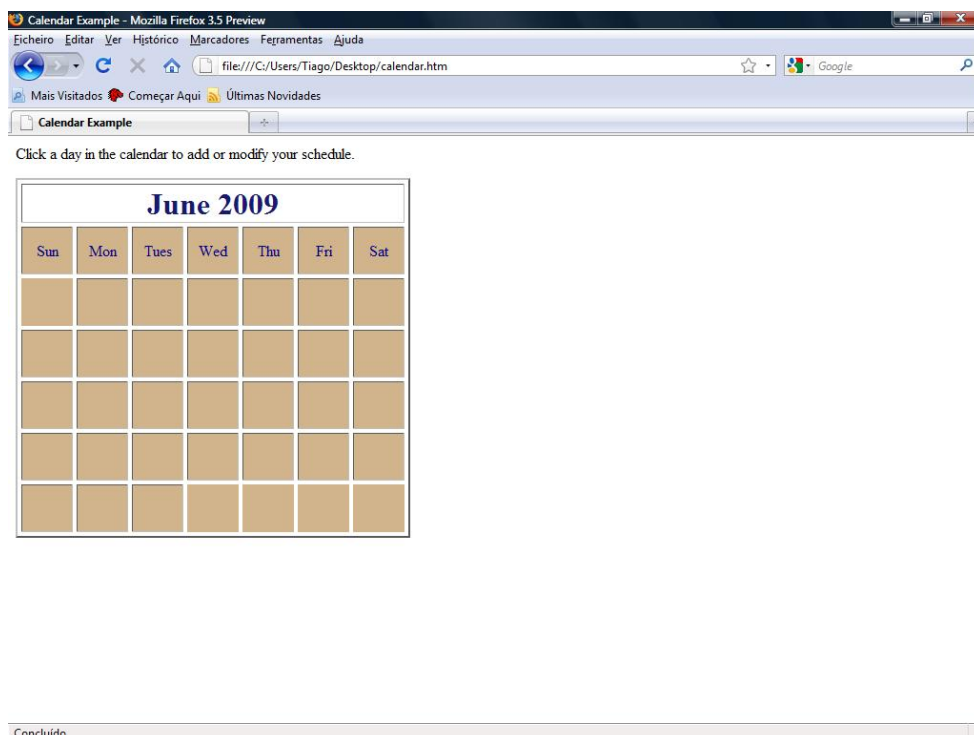


Figura 6.2: Calendar convertido sem intervenção manual

A figura 6.3 mostra a aplicação a funcionar completamente depois da intervenção manual. No anexo B.6 é apresentada esta versão final do código convertido.

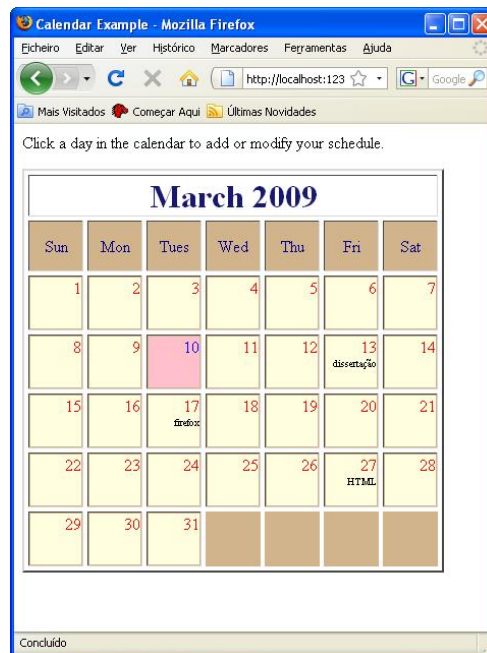


Figura 6.3: Calendar convertido após intervenção manual

6.1.3 Coolbiz

A aplicação Coolbiz consiste em 353 páginas desenvolvidas com tecnologia ASP.NET, cada uma com o seu código C# associado. No seu conjunto estas páginas invocam ficheiros HTC, pelo menos uma vez, em 1388 situações.

Uma vez que o Coolbiz tem características maiores de aplicação que os exemplos anteriores, faz mais sentido avaliar a conversão realizada a nível de componentes que são utilizados nas várias fases da aplicação, do que avaliar o funcionamento de cada página em particular. Assim, apesar do conversor actuar noutros níveis da aplicação, importa mais perceber de que maneira a funcionalidade oferecida pelos HTML Components é convertida com sucesso para a aplicação no Firefox, uma vez que são estas funcionalidades que apresentam os maiores problemas em *browsers* diferentes do IE na aplicação Coolbiz.

A tabela 6.1 apresenta os HTML Components utilizados na aplicação “Coolbiz - Back-office”, indicando o seu tamanho e a sua relevância no contexto da aplicação.

O componente mais utilizado é aquele que está definido no ficheiro `MainMenu.htc` e que serve para representar o menu principal da aplicação. Este HTC foi o componente focado durante o desenvolvimento da ferramenta de conversão e, como tal, encontra-se convertido de maneira já satisfatória, tendo sido necessária, no entanto, alguma intervenção

Avaliação do Protótipo

Tabela 6.1: Tabela de características de HTC em “Coolbiz - Backoffice”

Características de HTML Components			
<i>nome</i>	nº de linhas	nº de linhas de JScript	nº de páginas em que é utilizado
Calendar	1297	1020	96
CalendarPopUp	454	382	96
ComboBox	769	686	170
ContextSearch	553	520	48
ContextView	167	132	5
List	157	126	0
MainMenu	1077	949	282
MultiPage	199	167	168
MultiPageVertical	173	145	23
Numeric	475	412	86
QuickSearch	471	431	39
ScrollableTable	327	300	184
TabStrip	1518	1456	168
TabStripVertical	187	155	23
toolbar	2740	2692	0
treeview	4061	4005	0

manual para concluir essa conversão.

As figuras 6.4 e 6.5 apresentam o aspecto original deste componente em IE e Firefox, respectivamente. Como é possível verificar, a perda da funcionalidade de menu em Firefox é dramática, sendo apenas apresentada uma lista com o texto de cada ligação do menu, e este texto nem sequer é apresentado como uma ligação para outra página. Ou seja, a navegação em Firefox é completamente impossível.

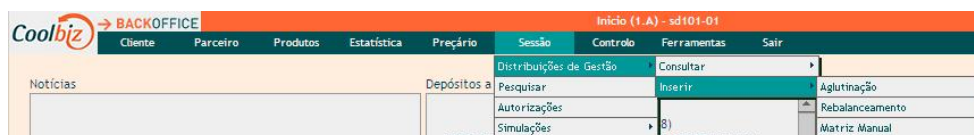


Figura 6.4: Menu do “Coolbiz - BackOffice” em IE

A conversão dos HTC na aplicação Coolbiz introduz uma primeira novidade em relação aos outros exemplos: a necessidade de passar argumentos utilizados nos HTC para o respectivo *script*. Estes argumentos, definidos no componente nos elementos com etiqueta *PROPERTY* são passados como atributos da *tag* customizada do HTC ao cliente Internet Explorer que utiliza os valores desses atributos na execução do *script*. Assim, durante a conversão, estes atributos são colocados como variáveis do *script* e o seu valor atribuído tal e qual está definido no HTML. Esta conversão é feita com intervenção do HTML Parser da Majestic-12, que é necessário para a ferramenta ter acesso a estes atributos e respectivos valores.

Avaliação do Protótipo

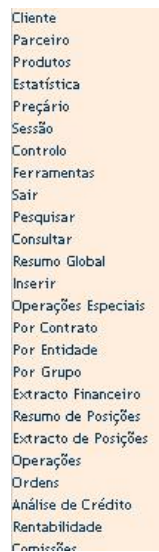


Figura 6.5: Menu do “Coolbiz - BackOffice” em Firefox sem conversão

Outra conversão feita com o auxílio do HTML Parser é a inclusão da função associada ao evento *oncontentready*, específico de HTC e Microsoft, para ser executada no final do *script*, quando todo o conteúdo do HTC estiver no documento original.

Isto já foi uma necessidade no exemplo do Calendar, mas neste caso esta inclusão ocorre automaticamente, uma vez que o componente responsável pelo menu principal apenas precisa que a função seja executada, sem as considerações específicas que foram necessárias para o Calendar.

A mudança de contexto dos *scripts* feita em ANTLR, ocorre tal como nos exemplos anteriores. Entre declarações de variáveis `element` nas funções que necessitam dessa variável e mudança de referências a `element.document` para `document`, a conversão é bem sucedida.

No que toca à conversão para Firefox feita em TXL, ocorrem 125 transformações. Entre essas, a substituição da função `attachEvent` por `addEventListener` é das que mais ocorre, e de especial relevância, uma vez que esta associação de eventos como `"click"` e `"mousedown"` a elementos HTML é o que confere uma grande dinâmica a páginas *web* como o Coolbiz.

Estas alterações não são ainda suficientes, no entanto, para que a funcionalidade esteja completamente implementada em Firefox. São necessárias ainda algumas alterações específicas, sendo que dois terços destas alterações são motivadas por uma razão apenas, o tratamento de atributos não válidos em HTML como propriedades dos elementos HTML em JavaScript, tal como tinha acontecido numa ocasião na conversão do Calendar. Os vários elementos do menu utilizam atributos como `ClickScript`, `URL` ou `TargetMenu` para definir o que esse elemento representa num menu (se é uma ligação para outra página, se é um caminho para um submenu, etc) e cada vez que os valores

desses atributos são necessários na execução do script, ocorre uma transformação para utilizar ou a função `getAttribute` ou a colecção `attributes`, em que `element.URL` passa a `element.attributes['URL']`.

O restante terço de alterações específicas necessárias estão relacionadas com operações inválidas em Firefox, como uma subtracção de determinado número de *pixels* por um número real, o facto do IE reconhecer e como tal não apresentar a tal lista por defeito, enquanto que o Firefox apesar de já construir o menu, continuava a apresentar a lista que aparecia originalmente, e ainda a invocação duma função que necessitava de passar `event` como argumento.

Aplicadas estas alterações o menu fica finalmente funcional em Firefox, embora sem ter o exacto aspecto que o caracterizava no IE. A figura 6.6 apresenta o resultado obtido para o menu principal em Firefox após utilização da ferramenta e algumas intervenções específicas.

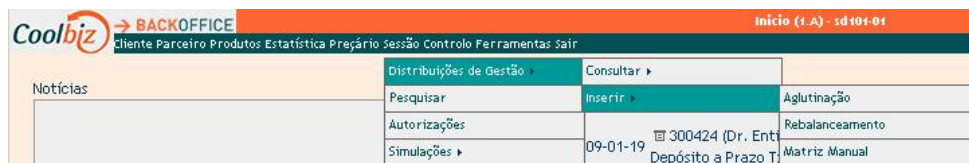


Figura 6.6: Menu do “Coolbiz - BackOffice” em Firefox após conversão

O protótipo foi também aplicado já na conversão de outro componente da aplicação Coolbiz, o ComboBox, com sucesso.

6.2 Estatísticas

Nesta secção são apresentados alguns dados sobre as conversões realizadas. Tal como a avaliação feita anteriormente, os dados sobre o Coolbiz referem-se apenas à conversão do componente personalizado em si e não a uma página completa que sofre ainda mais alterações a nível de TXL, devido à utilização de *scripts* próprios.

A tabela 6.2 apresenta as estatísticas retiradas das conversões realizadas sobre as páginas que serviram de exemplo, bem como da conversão do Coolbiz. O campo HTC indica o número de HTC que foram transferidos para o documento principal durante a conversão. O campo FA indica o número de transformações efectuadas ao JavaScript na fase de mudança de contexto HTC para documento principal, descrita no capítulo anterior como Fase A. O campo FB indica o mesmo relativamente à fase baseada em TXL, para aplicar transformações que promovem a compatibilidade com Firefox, enquanto que o campo FB (Esp.) indica as transformações feitas pelo TXL, nessa mesma fase, que são específicas daquele componente e portanto não devem ser vistas como conversões completamente automáticas. O campo HTML indica o número de transformações promovidas

pelo HTML Parser a nível do documento. O campo Manual indica intervenções no código feitas manualmente para conseguir obter a funcionalidade original da página.

Tabela 6.2: Tabela de estatísticas sobre as conversões

<i>nome</i>	Transformações ocorridas						
	HTC	FA	FB	FB (Esp.)	HTML	Manual	% Auto
Hilite	1	9	7	0	2	0	100%
Calendar	3	22	0	0	0	6	79%
CoolBiz (MainMenu)	1	17	110	15	2	0	90%
CoolBiz (ComboBox)	1	19	58	9	2	0	86%

O campo % Auto representa a percentagem de conversão que é feita automaticamente, através do quociente entre transformações manuais ou específicas e o total de transformações ocorridas, excluindo a transferência dos dados de ficheiros HTC que não são significativos para perceber qual a *performance* do conversor. É, por isso aplicada a seguinte fórmula, utilizando os campos da tabela:

$$\frac{Manual + FB(Esp.)}{FA + FB + FB(Esp.) + HTML + Manual}$$

O número de vezes que cada tipo de transformação é aplicada pode variar muito, dependendo da implementação do HTC respectivo. Em todos os casos em que a intervenção manual foi necessária, ela foi significativamente menor do que seria sem a utilização dos protótipos, sendo que a ferramenta desenvolvida assiste, de facto, o processo de conversão em grande parte. Inclusivamente, no caso do menu principal da aplicação Coolbiz foram atingidos os 90% pretendidos na especificação do objectivo da dissertação.

Capítulo 7

Conclusões e Trabalho Futuro

Este trabalho teve como objectivo a criação de um protótipo de conversão assistida de código específico do IE para se tornar compatível com outros *browsers*. Este protótipo deveria contribuir para alterar a funcionalidade de uma aplicação, chamada “Coolbiz - BackOffice”, que não suporta Firefox. Para atingir este fim, e ao mesmo tempo testar a utilização do conversor noutra tipo de páginas, foram desenvolvidos dois protótipos: um para aplicação directa no Coolbiz a funcionar durante a execução da aplicação e outro que traduz a partir de ficheiros HTML.

A seguir são apresentadas as conclusões sobre os resultados obtidos durante a elaboração desta tese, indicando o nível de satisfação dos resultados obtidos pela conversão proposta.

7.1 Satisfação dos Objectivos

A conversão automática de HTML Components seria a fase em que a intervenção automática era a mais desejada, uma vez que no que diz respeito à conversão do JScript propriamente dito, a estratégia seria à partida mais imediata, estando a dificuldade apenas em encontrar alternativas *standard* para o que era específico do Internet Explorer.

Neste sentido, tanto o Majestic-12 HTML Parser como o ANTLR permitiram lidar com o HTML e JavaScript de forma adequada. Apesar do HTML Parser não gerar uma árvore sintáctica do HTML, limitando-se a percorrer um documento de forma linear, tal não impediu, como se esperava, a realização das tarefas de conversão necessárias a atingir o objectivo do trabalho com sucesso.

A utilização da ferramenta TXL foi importante pois para além de ter permitido efectuar um conjunto significativo de alterações comuns em todos os *scripts*, também permitiu

actuar sobre transformações específicas a um *script* sem ter de modificar o conteúdo original do ficheiro. Esta característica pode evitar problema em termos de desenvolvimento da aplicação que pode ser feita independentemente do processo de conversão que é também desenvolvido.

Um objectivo que não foi tão bem sucedido foi o de conversão para o Safari da Apple. Se nos casos Hilite e Calendar, os resultados foram iguais para Safari e Firefox, bem como Opera e Google Chrome, no caso do menu principal do Coolbiz veio-se a perceber que a colocação dos submenus dependia de uma função implementada tanto em IE como em Firefox, mas sem implementação, nem implementação equivalente, no Safari.

Como foi visto no capítulo anterior, a aplicação das ferramentas desenvolvidas aos casos avaliados foi realizada com sucesso. Foram obtidos os 90% de transformações automáticas pretendidos inicialmente no caso da conversão do menu principal da Coolbiz. O pior caso até agora testado, converteu ainda assim perto de quatro quintos da aplicação. Em termos de funcionalidade, todos os objectos testados ficaram completamente funcionais, sendo que apenas são evidentes algumas diferenças de aspecto visual no caso do menu do Coolbiz.

7.2 Trabalho Futuro

Em termos de perspectivas futuras para esta solução, deverá ser aumentado o número de transformações automáticas com um trabalho de investigação ainda mais completo no que toca a funções específicas de JScript e respectivos equivalentes JavaScript. Outra possibilidade é implementar, para as funções que não tenham alternativas directas, uma biblioteca de funções para Firefox e Safari que sirvam o mesmo propósito dessas funções em Internet Explorer. Tornar esta biblioteca uma realidade eliminaria a necessidade de converter funções sem equivalente directo, ao mesmo tempo que permitiria utilizar nos outros *browsers* funcionalidades que normalmente não estariam disponíveis.

Em termos da aplicação Coolbiz, o desenvolvimento futuro passará por testar mais componentes HTC e colocar todos funcionais em Firefox e depois Safari. Também devem ser convertidos os *scripts* colocados em ficheiros externos às páginas da aplicação, bem como o JavaScript inserido em cada página. Embora estes scripts sejam convertidos com o TXL, têm características semelhantes a qualquer outro pedaço de código JavaScript e, como tal, podem necessitar de intervenção manual em vários pontos.

A implementação do protótipo de forma a poder ter acesso a todo o conteúdo da resposta HTML seria vantajosa porque permitiria testar que propriedades de elementos referidas são atributos HTML. Fazer esta alteração ao protótipo permitiria tornar a conversão para utilizar a função `getAttribute` ou a colecção `attributes` automática, e

sendo que este é um caso bastante comum em todas as conversões, aumentaria significativamente a percentagem de conversão efectuada automaticamente. Assim, é algo que deve ser realizado no futuro.

Parte do desenvolvimento futuro também passará por adicionar novas regras à classe que trata da passagem duma AST gerada pelo ANTLR para código JavaScript, para que todos os *tokens* possíveis estejam devidamente previstos e sejam imprimidos correctamente.

Referências

- [1] Microsoft. Internet Explorer. Disponível em <http://www.microsoft.com/windows/internet-explorer/>, acedido pela última vez em 19 de Maio de 2009.
- [2] Net Applications. Browser Market Share, 2009. Disponível em <http://marketshare.hitslink.com/browser-market-share.aspx?qprid=0>, acedido pela última vez em 11 de Março de 2009.
- [3] Netscape. Netscape Navigator. Disponível em <http://browser.netscape.com/>, acedido pela última vez em 18 de Junho de 2009.
- [4] Mozilla. Firefox Web Browser. Disponível em <http://www.mozilla.com/firefox/>, acedido pela última vez em 19 de Maio de 2009.
- [5] ParadigmaXis - Arquitectura e Engenharia de Software, S.A. Disponível em <http://www.paradigmaxis.pt/>.
- [6] Boletim INESC Porto. Crescer na incubadora - Empresas formadas com investigadores do INESC Porto, Novembro 2005. Disponível em <http://bipz.inescporto.pt/arquivo/56/paginas/destaque.html>, acedido pela última vez em 27 de Março de 2009.
- [7] Susana Almeida Ribeiro. Associação de Empresas de Software Open Source denuncia irregularidades na plataforma vortalGOV. Disponível em <http://ultimahora.publico.clix.pt/noticia.aspx?id=1367541&idCanal=61>, acedido pela última vez a 27 de Abril de 2009.
- [8] CoolLink Serviços Informáticos e de Consultadoria Lda. Disponível em <http://www.coollink.pt>.
- [9] Banco L. J. Carregosa, S. A. Disponível em <http://www.bancocarregosa.com/>.
- [10] Dave Raggett. Clean up your Web pages with HTML TIDY, Agosto 2000. Disponível em <http://tidy.sourceforge.net/docs/Overview.html>, acedido pela última vez em 15 de Abril de 2009.
- [11] Benfeng Chen e Vincent Y. Shen. Transforming web pages to become standard-compliant through reverse engineering. Em *W4A: Proceedings of the 2006 international cross-disciplinary workshop on Web accessibility (W4A)*, páginas 14–22, New York, NY, USA, 2006. ACM.

REFERÊNCIAS

- [12] Slingshot Solutions. Disponível em <http://slingfive.com/>.
- [13] Rob Eberhardt. ScriptConverter - Convert VBScript to Javascript, in a fraction of the time!, Julho 2008. <http://slingfive.com/pages/code/scriptConverter/>.
- [14] Paul Roub. IE View - Launch pages in IE from Firefox. <http://ieview.mozdev.org/>.
- [15] PCMan(Hong Jen Yee) e yuoo2k. mozdev.org - ietab. <http://ietab.mozdev.org/>.
- [16] Dean Edwards. moz-behaviors.xml. Disponível em <http://dean.edwards.name/moz-behaviors/>, acessado pela última vez a 27 de Abril de 2009.
- [17] Robert McMillan. Mozilla Gains on IE, 2004. Disponível em http://www.pcworld.com/article/116848/mozilla_gains_on_ie.html, acessado pela última vez em 29 de Maio de 2009.
- [18] Microsoft. ActiveX Controls. Disponível em <http://msdn.microsoft.com/en-us/library/aa751968.aspx>, acessado pela última vez em 22 de Junho de 2009.
- [19] Mozilla. ActiveX, 2009. Disponível em <http://support.mozilla.com/en-US/kb/Activex>, acessado pela última vez em 29 de Maio de 2009.
- [20] Chris Wilson. HTML Components, Componentizing Web Applications. submission to the W3C, Microsoft, Outubro 1998. <http://www.w3.org/TR/1998/NOTE-HTMLComponents-19981023>.
- [21] Microsoft. JScript (Windows Script Technologies). Disponível em <http://msdn.microsoft.com/en-us/library/hbxc2t98.aspx>, acessado pela última vez em 22 de Junho de 2009.
- [22] Microsoft. VBScript. Disponível em <http://msdn.microsoft.com/en-us/library/t0aew7h6%28VS.85%29.aspx>, acessado pela última vez em 22 de Junho de 2009.
- [23] Ian Hickson. XML Binding Language (XBL) 2.0 - W3C Candidate Recommendation 16 March 2007. Disponível em <http://www.w3.org/TR/2007/CR-xbl-20070316/>, acessado pela última vez a 27 de Abril de 2009.
- [24] Apple. Apple - Safari. Disponível em <http://www.apple.com/safari/>, acessado pela última vez em 19 de Maio de 2009.
- [25] John Siracusa. Mac OS X 10.3 Panther, 2003. Disponível em <http://arstechnica.com/apple/reviews/2003/11/macosex-10-3.ars>, acessado pela última vez em 29 de Maio de 2009.
- [26] James Clark. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, W3C, Novembro 1999. <http://www.w3.org/TR/1999/REC-xslt-19991116>.

REFERÊNCIAS

- [27] Cook, Craig and Schultz, David. *Beginning HTML with CSS and XHTML: Modern Guide and Reference (Beginning: from Novice to Professional)*. Apress, Berkely, CA, USA, 2007.
- [28] GNU Project. Bison - GNU parser generator. <http://www.gnu.org/software/bison/>.
- [29] Terence Parr. ANTLR Parser Generator. <http://www.antlr.org/>.
- [30] T. J. Parr e R. W. Quong. ANTLR: a predicated-LL(k) parser generator. *Softw. Pract. Exper.*, 25(7):789–810, 1995.
- [31] Javacc home. <https://javacc.dev.java.net/>.
- [32] Wikipedia. Comparison of parser generators — Wikipedia, The Free Encyclopedia, 2009. Disponível em http://en.wikipedia.org/w/index.php?title=Comparison_of_parser_generators&oldid=279404812, acessado pela última vez em 7 de Abril de 2009.
- [33] Terence Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Bookshelf, 2007.
- [34] Peter Sestoft. Grammars and parsing with ML, Janeiro 1994. Lecture notes for 4312, Department of Computer Science, Technical University of Denmark.
- [35] Terence Parr. StringTemplate Template Engine. <http://www.stringtemplate.org/>.
- [36] Majestic-12. C# HTML parser (.NET). http://www.majestic12.co.uk/projects/html_parser.php.
- [37] Software Technology Laboratory. The TXL Programming Language. Disponível em <http://www.txl.ca/>, acessado pela última vez em 19 de Maio de 2009.
- [38] James R. Cordy. Source transformation, analysis and generation in TXL. Em *PEPM '06: Proceedings of the 2006 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, páginas 1–11, New York, NY, USA, 2006. ACM.
- [39] Software Technology Laboratory. Guided Tour of TXL. Disponível em <http://www.txl.ca/tour/tour1.html>, acessado pela última vez a 16 de Abril de 2009.
- [40] Karl Trygve Kalleberg. Stratego: a programming language for program manipulation. *Crossroads*, 12(3):4–4, 2006.
- [41] Stratego/XT. Disponível em <http://strategoxt.org/>, acessado pela última vez em 19 de Maio de 2009.
- [42] Martin Bravenboer, Karl Trygve Kalleberg, Rob Vermaas, e Eelco Visser. Stratego/XT 0.17. A language and toolset for program transformation. *Sci. Comput. Program.*, 72(1-2):52–70, 2008.

REFERÊNCIAS

- [43] Semantic Designs, Inc. DMS Software Reengineering Toolkit. Disponível em <http://www.semdesigns.com/Products/DMS/DMSToolkit.html>, acessado pela última vez em 19 de Maio de 2009.
- [44] Ira D. Baxter, Christopher Pidgeon, e Michael Mehlich. Dms®: Program transformations for practical scalable software evolution. Em *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, páginas 625–634, Washington, DC, USA, 2004. IEEE Computer Society.
- [45] Semantic Designs, Inc. Semantic Designs Company Background. Disponível em <http://www.semdesigns.com/Company/>, acessado pela última vez em 29 de Abril de 2009.
- [46] Institut National de Recherche en Informatique et Automatique. Tom, 2009. Disponível em <http://tom.loria.fr/>, acessado pela última vez em 18 de Maio de 2009.
- [47] Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau, e Antoine Reilles. Tom: Piggybacking rewriting on java. Em *Conference on Rewriting Techniques and Applications - RTA'07 Proceedings of the 18th Conference on Rewriting Techniques and Applications LNCS*, volume 4533 de *LNCS*, páginas 36–47, Paris/France France, 06 2007. Springer-Verlag.
- [48] Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau, e Antoine Reilles. Tom Manual. LORIA, Nancy (France), version 2.6 edição, Abril 2008.
- [49] ECMA. ECMAScript Language Specification. Standard ECMA-262 3rd edition, Dezembro 1999.
- [50] Microsoft. Internet Explorer Developer Center, HTC Reference. Disponível em [http://msdn.microsoft.com/en-us/library/ms531018\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms531018(VS.85).aspx), acessado pela última vez em 19 de Maio de 2009.
- [51] World Wide Web Consortium. Cascading Style Sheets. <http://www.w3c.org/Style/CSS/>.
- [52] Microsoft. Internet Explorer Developer Center, HTML and CSS. Disponível em <http://msdn.microsoft.com/en-us/library/aa155133.aspx>, acessado pela última vez em 25 de Maio de 2009.
- [53] WebReference.com. HTML Components. Disponível em <http://www.webreference.com/js/column64/>, acessado pela última vez em 14 de Abril de 2009.
- [54] Peter-Paul Koch. Compatibility Master Table, 2009. Disponível em <http://www.quirksmode.org/compatibility.html>, acessado pela última vez em 25 de Maio de 2009.
- [55] Mozilla Foundation. Web Development, MDC, 2009. Disponível em https://developer.mozilla.org/en/Web_Development, acessado pela última vez em 25 de Maio de 2009.

REFERÊNCIAS

- [56] Refsnes Data. W3Schools - JavaScript Reference, 2009. Disponível em <http://www.w3schools.com/jsref/default.asp>, acedido pela última vez em 25 de Maio de 2009.

Anexo A

Hilite

Neste anexo apresenta-se o código original da aplicação de teste Hilite, bem como o resultado da conversão automática realizada.

A.1 hilite.htm

```
<HTML>
<HEAD>
<STYLE>
LI {
    behavior: url('hilite.htc');
}
</STYLE>
</HEAD>

<BODY>
<UL>
    <LI ID=L1>HTML Authoring</LI>
    <LI ID=L2>Dynamic HTML</LI>
</UL>
</BODY>
</HTML>
```

A.2 hilite.htc

```
<PUBLIC:ATTACH EVENT="onmouseover" ONEVENT="Hilite()" />
<PUBLIC:ATTACH EVENT="onmouseout" ONEVENT="Restore()" />
<SCRIPT LANGUAGE="JScript">
var normalColor;

function Hilite()
{
    if (event.srcElement == element)
    {
        normalColor = style.color;
        runtimeStyle.color = "red";
        runtimeStyle.cursor = "hand";
    }
}
```

```

}

function Restore()
{
    if (event.srcElement == element)
    {
        runtimeStyle.color = normalColor;
        runtimeStyle.cursor = "";
    }
}
</SCRIPT>

```

A.3 Código após conversão automática

```

<HTML>
<HEAD>
<SCRIPT>
var normalColor;

function Hilite(id, event)
{
    var element = document.getElementById(id);
    if (event.target == element)
    {
        normalColor = element.style.color;
        element.style.color = "red";
        element.style.cursor = "pointer";
    }
}

function Restore(id, event)
{
    var element = document.getElementById(id);
    if (event.target == element)
    {
        element.style.color = normalColor;
        element.style.cursor = "";
    }
}

</SCRIPT>
</HEAD>

<BODY>
<UL>
    <LI ID=L1 onmouseover="Hilite('L1', event)"
    onmouseout="Restore('L1', event)">HTML Authoring</LI>
    <LI ID=L2 onmouseover="Hilite('L2', event)"
    onmouseout="Restore('L2', event)">Dynamic HTML</LI>
</UL>
</BODY>
</HTML>

```

Anexo B

Calendar

Neste anexo apresenta-se o código original da aplicação de teste Calendar.

B.1 calendar.html

```
<HTML XMLNS:MYCAL>
<HEAD>
  <TITLE>Calendar Example</TITLE>
  <?IMPORT NAMESPACE="MYCAL" IMPLEMENTATION="calendar.htc"/>
</HEAD>

<BODY>
<P>Click a day in the calendar to add or modify your schedule.</P>

<MYCAL:CALENDAR></MYCAL:CALENDAR>

</BODY>
</HTML>
```

B.2 calendar.htc

```
<HTML XMLNS:MYCAL XMLNS:TODAY XMLNS:ANYDAY>
<HEAD>
  <?IMPORT NAMESPACE="ANYDAY" IMPLEMENTATION="day.htc"/>
  <?IMPORT NAMESPACE="TODAY" IMPLEMENTATION="today.htc"/>

<PUBLIC:COMPONENT tagName="CALENDAR">
  <ATTACH EVENT="oncontentready" ONEVENT="fnInit()" />
</PUBLIC:COMPONENT>

<SCRIPT LANGUAGE="JavaScript">
<!--
function fnInit() {
  defaults.viewLink = document;
}
// -->
</SCRIPT>

<STYLE>
```

Calendar

```
TD {
    background-color:tan;
    width:50;
    height:50;
}
</STYLE>
</HEAD>

<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--

// Copyright 1997 -- Tomer Shiran

setCal();

function leapYear(year) {
    if (year % 4 == 0) { // basic rule
        return true; // is leap year
    }
    /* else */ // else not needed when statement is "return"
    return false; // is not leap year
}

function getDays(month, year) {
    // create array to hold number of days in each month
    var ar = new Array(12);
    ar[0] = 31; // January
    ar[1] = (leapYear(year)) ? 29 : 28; // February
    ar[2] = 31; // March
    ar[3] = 30; // April
    ar[4] = 31; // May
    ar[5] = 30; // June
    ar[6] = 31; // July
    ar[7] = 31; // August
    ar[8] = 30; // September
    ar[9] = 31; // October
    ar[10] = 30; // November
    ar[11] = 31; // December

    // return number of days in the specified month (parameter)
    return ar[month];
}

function getMonthName(month) {
    // create array to hold name of each month
    var ar = new Array(12);
    ar[0] = "January";
    ar[1] = "February";
    ar[2] = "March";
    ar[3] = "April";
    ar[4] = "May";
    ar[5] = "June";
    ar[6] = "July";
    ar[7] = "August";
```

Calendar

```
ar[8] = "September";
ar[9] = "October";
ar[10] = "November";
ar[11] = "December";

// return name of specified month (parameter)
return ar[month];
}

function setCal() {
    // standard time attributes
    var now = new Date();
    var year = now.getFullYear();
    var month = now.getMonth();
    var monthName = getMonthName(month);
    var date = now.getDate();
    now = null;

    // create instance of first day of month, and extract the day on which it occurs
    var firstDayInstance = new Date(year, month, 1);
    var firstDay = firstDayInstance.getDay();
    firstDayInstance = null;

    // number of days in current month
    var days = getDays(month, year);

    // call function to draw calendar
    drawCal(firstDay + 1, days, date, monthName, year);
}

function drawCal(firstDay, lastDate, date, monthName, year) {
    // constant table settings
    //var headerHeight = 50 // height of the table's header cell
    var border = 2; // 3D height of table's border
    var cellspacing = 4; // width of table's border
    var headerColor = "midnightblue"; // color of table's header
    var headerSize = "+3"; // size of tables header font
    var colWidth = 60; // width of columns in table
    var dayCellHeight = 25; // height of cells containing days of the week
    var dayColor = "darkblue"; // color of font representing week days
    var cellHeight = 40; // height of cells representing dates in the calendar
    var todayColor = "red"; // color specifying today's date in the calendar
    var timeColor = "purple"; // color of font representing current time

    // create basic table structure
    var text = ""; // initialize accumulative variable to empty string
    // table settings
    text += '<TABLE BORDER=' + border + ' CELLSPACING=' + cellspacing + '>';
    text += '<TH COLSPAN=7 HEIGHT=' + 10 + '>'; // create table header cell
    // set font for table header
    text += '<FONT COLOR="' + headerColor + '" SIZE=' + headerSize + '>';
    text += monthName + ' ' + year;
    text += '</FONT>'; // close table header's font settings
    text += '</TH>'; // close header cell
```

Calendar

```
// variables to hold constant settings
var openCol = '<TD WIDTH=' + colWidth + ' HEIGHT=' + dayCellHeight + '>';
openCol += '<FONT COLOR="' + dayColor + '">';
var closeCol = '</FONT></TD>';

// create array of abbreviated day names
var weekDay = new Array(7);
weekDay[0] = "Sun";
weekDay[1] = "Mon";
weekDay[2] = "Tues";
weekDay[3] = "Wed";
weekDay[4] = "Thu";
weekDay[5] = "Fri";
weekDay[6] = "Sat";

// create first row of table to set column width and specify week day
text += '<TR ALIGN="center" VALIGN="center">';
for (var dayNum = 0; dayNum < 7; ++dayNum) {
    text += openCol + weekDay[dayNum] + closeCol;
}
text += '</TR>';

// declaration and initialization of two variables to help with tables
var dayOfMonth = 1;
var curCell = 1;

for (var row = 1; row <= Math.ceil((lastDate + firstDay - 1) / 7); ++row) {
    text += '<TR ALIGN="right" VALIGN="top">';
    for (var col = 1; col <= 7; ++col) {
        if ((curCell < firstDay) || (dayOfMonth > lastDate)) {
            text += '<TD></TD>';
            curCell++;
        } else {
            if (dayOfMonth == date) { // current cell represents today's date
                text += '<TD><TODAY:DAY value=' + dayOfMonth + '></TODAY:DAY></TD>';
            } else {
                text += '<TD><ANYDAY:DAY value=' + dayOfMonth + '></ANYDAY:DAY></TD>';
            }
            dayOfMonth++;
        }
    }
    text += '</TR>';
}

// close all basic table tags
text += '</TABLE>';
text += '</CENTER>';

// print accumulative HTML string
document.write(text);
}

// -->
</SCRIPT>
```

Calendar

```
</BODY>
</HTML>
```

B.3 day.htc

```
<HTML XMLNS:ANYDAY>
<HEAD>
<PUBLIC:COMPONENT tagName="DAY">
  <PROPERTY NAME="value"></PROPERTY>
  <ATTACH EVENT="oncontentready" ONEVENT="fnInit()"></ATTACH>
</PUBLIC:COMPONENT>

<STYLE>
  .clsDay {
    width:50;
    height:50;
    background-color:lightyellow;
    align:center;
    text-align:right;
  }
</STYLE>

<SCRIPT LANGUAGE="JavaScript">
<!--
function fnInit() {
  document.body.innerHTML = '<FONT COLOR="red">' + element.value + '</FONT>';
  document.body.className = "clsDay";
  defaults.viewLink = document;
  element.appointments = "";
  element.date = element.value;
}

function fnShowAppts() {
  newAppointments = prompt("Add your appointment:", element.appointments);
  if (newAppointments != null) element.appointments = newAppointments;
  document.body.innerHTML = '<FONT COLOR="red">' + element.date + '</FONT>'
  + "<BR>" + '<FONT SIZE="1">' + element.appointments + '</FONT>';
}
// -->
</SCRIPT>
</HEAD>

<BODY onclick="fnShowAppts()">
</BODY>
</HTML>
```

B.4 today.htc

```
<HTML XMLNS:TODAY>
<HEAD>
<PUBLIC:COMPONENT tagName="DAY">
  <PROPERTY NAME="value"></PROPERTY>
  <ATTACH EVENT="oncontentready" ONEVENT="fnInit()"></ATTACH>
```

Calendar

```
</PUBLIC:COMPONENT>

<STYLE>
  .clsDay {
    width:50;
    height:50;
    background-color: pink;
    align:center;
    text-align:right;
  }
</STYLE>

<SCRIPT LANGUAGE="JavaScript">
<!--
function fnInit()
{
  document.body.innerHTML = '<FONT COLOR="blue">' + element.value + '</FONT>';
  document.body.className = "clsDay";
  defaults.viewLink = document;
  element.appointments = "";
  element.date = element.value;
}

function fnShowAppts()
{
  newAppointments = prompt("Add your appointment:", element.appointments);
  if (newAppointments != null) element.appointments = newAppointments;
  document.body.innerHTML = '<FONT COLOR="blue">' + element.date + '</FONT>'
    + "<BR>" + '<FONT SIZE="1">' + element.appointments + '</FONT>';
}
// -->
</SCRIPT>
</HEAD>

<BODY onclick="fnShowAppts()">
</BODY>
</HTML>
```

B.5 Código após conversão automática apenas

```
<HTML xmlns:mycal>

<HEAD>

<TITLE>
Calendar Example
</TITLE>

<SCRIPT>
function fnInit ()
{

}
```

Calendar

```
function todayfnInit (id)
{
    var element = document.getElementById(id);

    element.innerHTML = '<FONT COLOR="red">' + element.value + '</FONT>';
    element.className = "todayclsDay";
    element.appointments = "";
    element.date = element.value;
}

function todayfnShowAppts (id)
{
    var element = document.getElementById(id);

    newAppointments = prompt("Add your appointment:",element.appointments);
    if (newAppointments != null)
        element.appointments = newAppointments;
    element.innerHTML = '<FONT COLOR="red">' + element.date + '</FONT>'
        + "<BR>" + '<FONT SIZE="1">' + element.appointments + '</FONT>';
}

function anydayfnInit (id)
{
    var element = document.getElementById(id);

    element.innerHTML = '<FONT COLOR="blue">' + element.value + '</FONT>';
    element.className = "anydayclsDay";
    element.appointments = "";
    element.date = element.value;
}

function anydayfnShowAppts (id)
{
    var element = document.getElementById(id);

    newAppointments = prompt("Add your appointment:",element.appointments);
    if (newAppointments != null)
        element.appointments = newAppointments;
    element.innerHTML = '<FONT COLOR="blue">' + element.date + '</FONT>'
        + "<BR>" + '<FONT SIZE="1">' + element.appointments + '</FONT>';
}
</SCRIPT>

<STYLE>

    TD {
        background-color:tan;
        width:50;
        height:50;
    }

    .clsDay {
```

Calendar

```
        width:50;
        height:50;
        background-color:lightyellow;
        align:center;
        text-align:right;
    }

    .clsDay {
        width:50;
        height:50;
        background-color: pink;
        align:center;
        text-align:right;
    }

</STYLE>

</HEAD>

<BODY>

<P>
Click a day in the calendar to add or modify your schedule.
</P>

<DIV>
<SCRIPT>

setCal();

function leapYear(year) {
    if (year % 4 == 0) { // basic rule
        return true; // is leap year
    }
    /* else */ // else not needed when statement is "return"
    return false; // is not leap year
}

function getDays(month, year) {
    // create array to hold number of days in each month
    var ar = new Array(12);
    ar[0] = 31; // January
    ar[1] = (leapYear(year)) ? 29 : 28; // February
    ar[2] = 31; // March
    ar[3] = 30; // April
    ar[4] = 31; // May
    ar[5] = 30; // June
    ar[6] = 31; // July
    ar[7] = 31; // August
    ar[8] = 30; // September
    ar[9] = 31; // October
    ar[10] = 30; // November
    ar[11] = 31; // December
}
```

Calendar

```
// return number of days in the specified month (parameter)
return ar[month];
}

function getMonthName(month) {
    // create array to hold name of each month
    var ar = new Array(12);
    ar[0] = "January";
    ar[1] = "February";
    ar[2] = "March";
    ar[3] = "April";
    ar[4] = "May";
    ar[5] = "June";
    ar[6] = "July";
    ar[7] = "August";
    ar[8] = "September";
    ar[9] = "October";
    ar[10] = "November";
    ar[11] = "December";

    // return name of specified month (parameter)
    return ar[month];
}

function setCal() {
    // standard time attributes
    var now = new Date();
    var year = now.getFullYear();
    var month = now.getMonth();
    var monthName = getMonthName(month);
    var date = now.getDate();
    now = null;

    // create instance of first day of month, and extract the day on which it occurs
    var firstDayInstance = new Date(year, month, 1);
    var firstDay = firstDayInstance.getDay();
    firstDayInstance = null;

    // number of days in current month
    var days = getDays(month, year);

    // call function to draw calendar
    drawCal(firstDay + 1, days, date, monthName, year);
}

function drawCal(firstDay, lastDate, date, monthName, year) {
    // constant table settings
    //var headerHeight = 50 // height of the table's header cell
    var border = 2; // 3D height of table's border
    var cellspacing = 4; // width of table's border
    var headerColor = "midnightblue"; // color of table's header
    var headerSize = "+3"; // size of tables header font
    var colWidth = 60; // width of columns in table
    var dayCellHeight = 25; // height of cells containing days of the week
    var dayColor = "darkblue"; // color of font representing week days
```

Calendar

```
var cellHeight = 40; // height of cells representing dates in the calendar
var todayColor = "red"; // color specifying today's date in the calendar
var timeColor = "purple"; // color of font representing current time

// create basic table structure
var text = ""; // initialize accumulative variable to empty string
// table settings
text += '<TABLE BORDER=' + border + ' CELLSPACING=' + cellspacing + '>';
text += '<TH COLSPAN=7 HEIGHT=' + 10 + '>'; // create table header cell
// set font for table header
text += '<FONT COLOR="' + headerColor + '" SIZE=' + headerSize + '>';
text += monthName + ' ' + year;
text += '</FONT>'; // close table header's font settings
text += '</TH>'; // close header cell

// variables to hold constant settings
var openCol = '<TD WIDTH=' + colWidth + ' HEIGHT=' + dayCellHeight + '>';
openCol += '<FONT COLOR="' + dayColor + '">';
var closeCol = '</FONT></TD>';

// create array of abbreviated day names
var weekDay = new Array(7);
weekDay[0] = "Sun";
weekDay[1] = "Mon";
weekDay[2] = "Tues";
weekDay[3] = "Wed";
weekDay[4] = "Thu";
weekDay[5] = "Fri";
weekDay[6] = "Sat";

// create first row of table to set column width and specify week day
text += '<TR ALIGN="center" VALIGN="center">';
for (var dayNum = 0; dayNum < 7; ++dayNum) {
    text += openCol + weekDay[dayNum] + closeCol;
}
text += '</TR>';

// declaration and initialization of two variables to help with tables
var dayOfMonth = 1;
var curCell = 1;

for (var row = 1; row <= Math.ceil((lastDate + firstDay - 1) / 7); ++row) {
    text += '<TR ALIGN="right" VALIGN="top">';
    for (var col = 1; col <= 7; ++col) {
        if ((curCell < firstDay) || (dayOfMonth > lastDate)) {
            text += '<TD></TD>';
            curCell++;
        } else {
            if (dayOfMonth == date) { // current cell represents today's date
                text += '<TD><TODAY:DAY value=' + dayOfMonth + '></TODAY:DAY></TD>';
            } else {
                text += '<TD><ANYDAY:DAY value=' + dayOfMonth + '></ANYDAY:DAY></TD>';
            }
            dayOfMonth++;
        }
    }
}
```

Calendar

```
    }
    text += '</TR>';
}

// close all basic table tags
text += '</TABLE>';
text += '</CENTER>';

// print accumulative HTML string
document.write(text);
}

</SCRIPT>
</DIV>

</BODY>

</HTML>
```

B.6 Código após conversão automática e intervenção manual

```
<HTML xmlns:mycal>

<HEAD>

<TITLE>
Calendar Example
</TITLE>

<SCRIPT>
function fnInit ()
{

}

function todayfnInit (id)
{
    var element = document.getElementById(id);

    element.innerHTML = '<FONT COLOR="red">' + element.getAttribute('value')
    + '</FONT>';
    element.className = "todayclsDay";
    element.appointments = "";
    element.date = element.getAttribute('value');
}

function todayfnShowAppts (id)
{
    var element = document.getElementById(id);

    newAppointments = prompt("Add your appointment:", element.appointments);
    if (newAppointments != null)
        element.appointments = newAppointments;
```

Calendar

```
        element.innerHTML = '<FONT COLOR="red">' + element.date + '</FONT>'
        + "<BR>" + '<FONT SIZE="1">' + element.appointments + '</FONT>';
    }

function anydayfnInit (id)
{
    var element = document.getElementById(id);

    element.innerHTML = '<FONT COLOR="blue">' + element.getAttribute('value')
    + '</FONT>';
    element.className = "anydayclsDay";
    element.appointments = "";
    element.date = element.getAttribute('value');
}

function anydayfnShowAppts (id)
{
    var element = document.getElementById(id);

    newAppointments = prompt("Add your appointment:",element.appointments);
    if (newAppointments != null)
        element.appointments = newAppointments;
    element.innerHTML = '<FONT COLOR="blue">' + element.date + '</FONT>'
    + "<BR>" + '<FONT SIZE="1">' + element.appointments + '</FONT>';
}
</SCRIPT>

<STYLE>

    TD {
        background-color:tan;
        width:50;
        height:50;
    }

    .anydayclsDay {
        width:50;
        height:50;
        background-color:lightyellow;
        align:center;
        text-align:right;
    }

    .todayclsDay {
        width:50;
        height:50;
        background-color: pink;
        align:center;
        text-align:right;
    }

</STYLE>
```

Calendar

</HEAD>

<BODY>

<P>

Click a day in the calendar to add or modify your schedule.

</P>

<DIV>

<SCRIPT>

```
setCal();
```

```
function leapYear(year) {  
    if (year % 4 == 0) { // basic rule  
        return true; // is leap year  
    }  
    /* else */ // else not needed when statement is "return"  
    return false; // is not leap year  
}
```

```
function getDays(month, year) {  
    // create array to hold number of days in each month  
    var ar = new Array(12);  
    ar[0] = 31; // January  
    ar[1] = (leapYear(year)) ? 29 : 28; // February  
    ar[2] = 31; // March  
    ar[3] = 30; // April  
    ar[4] = 31; // May  
    ar[5] = 30; // June  
    ar[6] = 31; // July  
    ar[7] = 31; // August  
    ar[8] = 30; // September  
    ar[9] = 31; // October  
    ar[10] = 30; // November  
    ar[11] = 31; // December  
  
    // return number of days in the specified month (parameter)  
    return ar[month];  
}
```

```
function getMonthName(month) {  
    // create array to hold name of each month  
    var ar = new Array(12);  
    ar[0] = "January";  
    ar[1] = "February";  
    ar[2] = "March";  
    ar[3] = "April";  
    ar[4] = "May";  
    ar[5] = "June";  
    ar[6] = "July";  
    ar[7] = "August";  
    ar[8] = "September";  
    ar[9] = "October";
```

Calendar

```
ar[10] = "November";
ar[11] = "December";

// return name of specified month (parameter)
return ar[month];
}

function setCal() {
    // standard time attributes
    var now = new Date();
    var year = now.getFullYear();
    var month = now.getMonth();
    var monthName = getMonthName(month);
    var date = now.getDate();
    now = null;

    // create instance of first day of month, and extract the day on which it occurs
    var firstDayInstance = new Date(year, month, 1);
    var firstDay = firstDayInstance.getDay();
    firstDayInstance = null;

    // number of days in current month
    var days = getDays(month, year);

    // call function to draw calendar
    drawCal(firstDay + 1, days, date, monthName, year);
}

function drawCal(firstDay, lastDate, date, monthName, year) {
    // constant table settings
    //var headerHeight = 50 // height of the table's header cell
    var border = 2; // 3D height of table's border
    var cellspacing = 4; // width of table's border
    var headerColor = "midnightblue"; // color of table's header
    var headerSize = "+3"; // size of tables header font
    var colWidth = 60; // width of columns in table
    var dayCellHeight = 25; // height of cells containing days of the week
    var dayColor = "darkblue"; // color of font representing week days
    var cellHeight = 40; // height of cells representing dates in the calendar
    var todayColor = "red"; // color specifying today's date in the calendar
    var timeColor = "purple"; // color of font representing current time

    // create basic table structure
    var text = ""; // initialize accumulative variable to empty string
    // table settings
    text += '<TABLE BORDER=' + border + ' CELLSPACING=' + cellspacing + '>';
    text += '<TH COLSPAN=7 HEIGHT=' + 10 + '>'; // create table header cell
    // set font for table header
    text += '<FONT COLOR="' + headerColor + '" SIZE=' + headerSize + '>';
    text += monthName + ' ' + year;
    text += '</FONT>'; // close table header's font settings
    text += '</TH>'; // close header cell

    // variables to hold constant settings
    var openCol = '<TD WIDTH=' + colWidth + ' HEIGHT=' + dayCellHeight + '>';
```

Calendar

```
openCol += '<FONT COLOR="' + dayColor + '">';
var closeCol = '</FONT></TD>';

// create array of abbreviated day names
var weekDay = new Array(7);
weekDay[0] = "Sun";
weekDay[1] = "Mon";
weekDay[2] = "Tues";
weekDay[3] = "Wed";
weekDay[4] = "Thu";
weekDay[5] = "Fri";
weekDay[6] = "Sat";

// create first row of table to set column width and specify week day
text += '<TR ALIGN="center" VALIGN="center">';
for (var dayNum = 0; dayNum < 7; ++dayNum) {
    text += openCol + weekDay[dayNum] + closeCol;
}
text += '</TR>';

// declaration and initialization of two variables to help with tables
var dayOfMonth = 1;
var curCell = 1;

for (var row = 1; row <= Math.ceil((lastDate + firstDay - 1) / 7); ++row) {
    text += '<TR ALIGN="right" VALIGN="top">';
    for (var col = 1; col <= 7; ++col) {
        if ((curCell < firstDay) || (dayOfMonth > lastDate)) {
            text += '<TD></TD>';
            curCell++;
        } else {
            if (dayOfMonth == date) { // current cell represents today's date
                text += '<TD><DIV onclick=todayfnShowAppts(\'a\' + dayOfMonth + \' \')
oncontentready=todayfnInit(\'a\' + dayOfMonth + \' \') id=a\'
+ dayOfMonth + \' value=\' + dayOfMonth + \'></DIV></TD>';
            } else {
                text += '<TD><DIV onclick=anydayfnShowAppts(\'a\' + dayOfMonth + \' \')
oncontentready=anydayfnInit(\'a\' + dayOfMonth + \' \') id=a\'
+ dayOfMonth + \' value=\' + dayOfMonth + \'></DIV></TD>';
            }
            dayOfMonth++;
        }
    }
    text += '</TR>';
}

// close all basic table tags
text += '</TABLE>';
text += '</CENTER>';

// print accumulative HTML string
document.write(text);

// workaround para oncontentready
var divs = document.getElementsByTagName('div');
```

Calendar

```
for(i=0;i<divs.length;i++){
    if(divs[i].getAttribute("oncontentready"))
    {
        ContentReadyString = divs[i].getAttribute("oncontentready");
        argsIndex = ContentReadyString.indexOf("(");
        functionString = ContentReadyString.slice(0,argsIndex);
        argsString = ContentReadyString.slice(argsIndex,
        ContentReadyString.length);
        argsString = argsString.replace("'", "");
        argsString = argsString.replace('"', "");
        window[functionString](argsString);
    }
}

</SCRIPT>
</DIV>

</BODY>

</HTML>
```