

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CENTRO TECNOLÓGICO
UNIVERSIDADE FEDERAL DE SANTA CATARINA

**PORTE DE UMA LINGUAGEM DE PROGRAMAÇÃO PARA EXECUÇÃO
NATIVA EM NAVEGADORES WEB**

**PORTE DO AMBIENTE DE COMPILAÇÃO E EXECUÇÃO
DE UMA LINGUAGEM DE PROGRAMAÇÃO
INTRINSECAMENTE PARALELA
PARA EXECUÇÃO NATIVA EM NAVEGADORES WEB**

MARCELO SAVISKI

MONOGRAFIA DE CONCLUSÃO DO CURSO DE CIÊNCIAS DA COMPUTAÇÃO

Orientador: Luiz Fernando Bier Melgarejo

Florianópolis, 15 de julho de 2010.

MARCELO SAVISKI

**PORTE DE UMA LINGUAGEM DE PROGRAMAÇÃO
PARA EXECUÇÃO NATIVA EM NAVEGADORES WEB**

Trabalho de conclusão de curso apresentado
como parte das atividades para obtenção do
título de Bacharel, do curso de Ciências da
Computação

Orientador: Luiz Fernando Bier Melgarejo

Florianópolis, 2010

Autoria: Marcelo Saviski

Título: Porte de uma linguagem de programação para execução nativa em navegadores Web

Trabalho de conclusão de curso apresentado
como parte das atividades para obtenção do
título de Bacharel, do curso de Ciências da
Computação

Prof. Luiz Fernando Bier Melgarejo

Orientador

Giovani Pieri

Co-orientador

Banca Examinadora

Prof. Dr. Rosvelter Coelho da Costa

Ricardo Ghisi Tobaldini

*Dedico este trabalho aos programadores,
que não serão lembrados por todos os
problemas que ajudaram a solucionar,
mas sim por aqueles que ajudaram a criar.*

AGRADECIMENTOS

*Agradeço a todos com os quais já tive
a felicidade de compartilhar o tempo
desenvolvendo projetos, com os quais aprendi
grande parte dos conhecimentos aqui
aplicados.*

*“O que é mais difícil não é escrever muito,
é dizer tudo, escrevendo pouco”.*

Júlio Dantas

RESUMO

O desenvolvimento de aplicações nos navegadores Web sempre dependeu de extensões de terceiros para realizar tarefas mais sofisticadas como multitarefa, manipulação gráfica e sonorização. Este trabalho visa explorar os novos recursos disponibilizados pelos navegadores modernos. Recentemente surgiu um novo padrão chamado de HTML5 que disponibiliza as páginas da internet muitas funcionalidades antes apenas disponíveis nas aplicações *desktop*. Aplicações que residem na Web podem ser acessadas através de um endereço virtual, elas apresentam vantagens sobre os aplicativos instalados localmente nas máquinas de seus usuários no que diz respeito a velocidade com que atualizações são propagadas, diminuindo o tempo levado para correções de falhas encontradas no aplicativo por seus usuários. O ciclo de liberação de novas versões de softwares é cada vez mais constante, o que implica na inclusão de suporte a atualizações automatizadas, por outro lado, em aplicações Web todos os seus usuários sempre executam sua versão mais recente.

A infra-estrutura para o desenvolvimento de aplicativos Web evoluiu, permitindo que cada vez mais aplicações exclusivas do *desktop* sejam substituídas por versões *on-line*, podendo ser acessadas de qualquer computador sem a instalação do aplicativo localmente.

O amadurecimento da infra-estrutura disponível nos navegadores de internet recentemente, permitiu que um ambiente de compilação e execução de uma linguagem de programação intrinsecamente paralela fosse portado do ambiente Java a qual foi inicialmente projetado para executar nativamente em um navegador.

Palavras-chave: Telis, JavaScript, HTML5, GWT, Web-Workers.

LISTA DE ABREVIATURAS E SIGLAS

ECMA	European Computer Manufacturers Association
GWT	Google Web ToolKit
HTML	Hypertext Markup Language
WHATWG	Web Hypertext Application Technology Working Group
W3C	World Wide Web Consortium

SUMÁRIO

Introdução.....	10
1 Telis.....	12
2 Navegadores e javascript como linguagem de montagem.....	14
2.1 Principais Navegadores.....	14
2.2 Motores de Renderização.....	17
2.3 JavaScript.....	18
2.3.1 Interpretadores de JavaScript.....	18
3 HTML5.....	21
3.1 Canvas.....	22
3.2 Web Workers.....	25
4 Porte.....	30
4.1 GWT.....	30
4.2 Estrutura da máquina Telis.....	31
4.3 Problemas encontrados durante a conversão do projeto.....	31
4.3.1 Classes não implementadas pelo GWT.....	32
4.3.2 Reflexão.....	32
4.3.3 Internacionalização.....	34
4.3.4 Serialização.....	35
4.3.5 Concorrência.....	36
4.3.6 Primitivas Gráficas:.....	37
4.4 Abordagem adotada para comunicação.....	38
4.4.1 Comunicação bidirecional entre a Página e o Worker.....	38
4.4.2 Comunicação assíncrona.....	43
4.4.3 Comunicação síncrona.....	44
Considerações finais.....	45
Anexos.....	48

INTRODUÇÃO

Telis é uma ferramenta de aprendizado de programação desenvolvido pelo EDUGRAF. Composto por uma linguagem de programação e um ambiente de desenvolvimento especialmente projetado para estudantes que não tiveram contato com linguagens de programação. Telis diminui a curva de aprendizado de conceitos avançados como concorrência, computação distribuída e orientação a objetos reduzindo a carga conceitual requerida para colocá-los em prática, mesclando a assimilação destes conceitos com a etapa de aprendizagem do uso da ferramenta.

Os navegadores Web atualmente são sofisticados softwares que não apenas visualizam *sites* e páginas da internet, são plataformas abertas ao desenvolvimento de aplicativos que residem naturalmente na *Web* e manipulam recursos *on-line*.

Esta trabalho pretende mostrar que é possível executar aplicaques escritos na linguagem Telis utilizando unicamente a infra-estrutura provida pelos navegadores de internet que adotam as novas tecnologias propostas pelo padrão HTML5, sem o uso de *applets*. Para isso, portou-se a máquina Telis da linguagem Java para JavaScript. Foram feitas pesquisas para encontrar meios de suportar concorrência e manipulação gráfica em uma página HTML. A aplicação foi testada nos principais navegadores em produção que estão em conformidade com os padrões vigentes na Web.

Objetivos Gerais

Portar uma implementação existente de um ambiente de compilação e execução de uma linguagem de programação escrito em Java para JavaScript, com o uso do Google Web Toolkit e implementar um ambiente de execução Web para a máquina Telis, com primitivas gráficas baseadas em *turtle-graphics*.

Objetivos Específicos

- Porta a máquina Telis para compilação com o GWT e gerar código na linguagem JavaScript.
- Pesquisar o suporte a concorrência em navegadores Web e adequar a implementação da máquina a este modelo.
- Estender o ambiente de execução da máquina Telis fornecendo primitivas gráficas baseada em *turtle-graphics*.

1 TELIS

Telis é uma linguagem interpretada com tipagem dinâmica, implementada em Java e desenvolvida pelo laboratório Edugraf. A linguagem expõe de maneira natural paralelismo, princípios de orientação a objetos e conceitos de sistemas distribuídos permitindo que programadores inexperientes desenvolvam programas que manipulam objetos gráficos na tela, colhem entradas do usuário e se comunicam em rede.

A simplicidade de Telis a torna adequada para o aprendizado dos iniciantes. Todos os nomes de comandos são em português, todas as operações realizadas obtêm seus operandos de uma pilha, o mesmo local aonde os resultados das operações são armazenados. Para isso é utilizada a notação pós-fixa de operadores. Listas e operações sobre listas são partes da linguagem, assim como criação de variáveis, as quais os valores da pilha podem ser associados.

Concorrência também é tratada diretamente pela linguagem sem a necessidade de outros mecanismos de sincronização, encapsulada pelos atores que são autônomos e executam comandos em pilhas distintas de forma independente. A comunicação entre os atores pode ser síncrona ou assíncrona. Se for assíncrona, um ator repassa uma mensagem a todos os outros atores e continua sua execução normalmente. Caso os outros atores desejem tratar a mensagem recebida, interrompem sua tarefa atual, executam uma tarefa relacionada com a mensagem recebida e depois continuam com a execução da tarefa anterior do ponto em que foi interrompida. Se for síncrona, um ator requisita a outro ator específico que interrompa sua execução e trate uma requisição, ambos só continuam seu ciclo de execução normal quando essa requisição for tratada completamente.

Os comandos, em Telis conhecidos como primitivas, são agrupados em agendas similares aos métodos, procedimentos ou funções em outras linguagens de programação. Por sua vez, as

agendas são nomeadas e agrupadas em Moldes e Modelos. Um Molde é semelhante a uma classe abstrata, que não pode ser instanciada. Modelos são subdivididos em Modelos de Objeto e Modelos de Ator. Modelos de Objetos são equivalentes a classes no paradigma de orientação a objetos e Modelos de ator são os equivalentes as classes executáveis em *Threads* ou processos separados. Ambos Moldes e Modelos podem herdar outros Moldes, em Telis é dito que um Molde molda o outro

Por fim, um Modelo de ator pode ser executado por vários atores. Atores são entidades ativas por onde a execução flui.

2 NAVEGADORES E JAVASCRIPT COMO LINGUAGEM DE MONTAGEM

Um navegador é um aplicativo que recupera e apresenta recursos disponíveis na internet. Um dos primeiros navegadores gráficos foi o Mosaic, que influenciou seus sucessores Netscape Navigator e Internet Explorer (Grosskurth *et al.*, 2006). Com o tempo, surgiram outros navegadores, como o Opera e o Mozilla, este último desenvolvido a partir do Netscape. Em 2003 o Safari foi criado, em 2004 o Mozilla deu origem ao Firefox. E recentemente o Google disponibilizou o seu navegador o Google Chrome.

As seções seguintes apresentam características da infra-estrutura dos navegadores, traçando um histórico de sua evolução e pretendem demonstrar que estes são plataformas robustas para o desenvolvimento de softwares.

2.1 Principais Navegadores

As seções a seguir, apresentam o histórico e características dos navegadores mais populares.

2.1.1 Netscape

Como dito por Cusumano(1999, tradução nossa)

A Netscape Communications Corp. foi estabelecida em Abril de 1994 por Jim Clark, que também fundou a *Silicon Graphics, Inc.*, e Marc Andreessen, um recém graduado da *University of Illinois*, local onde liderou o time de programadores que criou o *Mosaic*, o primeiro navegador www a tornar a Web popular ao público geral. Juntos eles fundaram a Netscape para criarem uma interface universal simples que deveria permitir aos usuários com praticamente qualquer dispositivo de comunicação acessar a *Web*. O foco inicial foi em dois produtos: um navegador comercial que deveria fun-

cionar corretamente nos pontos em que o *Mosaic* falhava, e um servidor Web, o software que permitia que indivíduos e companhias criassem *Web sites*.

A primeira versão no Navigator foi disponibilizada em 1994, se tornando rapidamente um espetacular sucesso. A empresa Netscape se distinguia por sua habilidade em escrever seu *software* de Internet para todas as principais plataformas de computadores, incluindo o Unix. O Netscape usava técnicas de programação multi-plataforma e por isso conseguia publicar seu produto para todas as plataformas rapidamente.

O Navigator chegou a possuir 90% do mercado de navegadores, mas quando a Microsoft começou a empacotar o seu Internet Explorer no Windows 95, a popularidade do Navigator despencou até que no final de 1997 não possuía mais do que 50% do mercado, despencando de forma gradual até ser adquirida pela *America Online* em 1998.

Por volta de 1997, engenheiros na Netscape estavam entusiasmados com Java, que prometia aos desenvolvedores que escrevendo um único código eles o conseguiriam executar em vários sistemas operacionais. Esta característica do Java eliminaria a necessidade de adaptações para cada plataforma. O time de desenvolvedores do Netscape desenvolveu uma máquina virtual Java própria e planejou re-escrever o código de seu navegador e de outros componentes nesta linguagem. Mas houveram muitas frustrações com o Java, principalmente relacionadas com o desenvolvimento de interfaces. Como o Java não funcionou como esperado a versão Java do Navigator foi cancelada em 1998.

Outra grande contribuição do Netscape foi a criação da linguagem JavaScript para permitir a inclusão de blocos de código que executassem no cliente, aumentando o dinamismo das páginas da internet.

2.1.2 Internet Explorer

O Microsoft Internet Explorer teve como base o Mosaic, ele ganhou popularidade por vir incluído no Sistema Operacional Windows 95 e por ser permitido baixá-lo gratuitamente, mesmo para uso comercial. Rapidamente novas funcionalidades foram incluídas, como suporte a *JavaScript* e marcações em HTML exclusivas. Mas ele só passou a dominar o mercado a partir da versão número quatro. Em 2002 já na sexta versão, alcançou o pico na quantidade de

usuários, dominando 96% do mercado, desde então se mantém como o navegador mais utilizado, porém sua popularidade vem caindo. Este período de monopólio de um único navegador estagnou a inovação nos navegadores por um longo tempo.

2.1.3 Firefox

Depois que o Navigator perdeu espaço para o Internet Explorer, a Netscape disponibilizou o seu código fonte e o confiou a recém-criada Fundação Mozilla, um projeto dirigido pela comunidade para criar um sucessor do Navigator. Depois de anos de desenvolvimento a primeira versão foi liberada no final de 2004 com o nome de Phoenix que depois foi alterado para Firefox. Ainda no início de 2004 a Fundação Mozilla e a Opera reuniram esforços para desenvolverem novos padrões abertos, com isso criaram um grupo voltado para a rápida criação de novos padrões para serem submetidos ao W3C¹ para aprovação, chamado de WHATWG².

2.1.4 Chrome

O Google Chrome é um navegador desenvolvido pelo Google, montado através de 25 bibliotecas de terceiros e da própria empresa. É voltado para aplicações Web onde a performance e a iteratividade são importantes, como quando as pessoas assistem vídeos, conversam e jogam jogos online baseados na Web. Também é reconhecido pelo rápido desenvolvimento e pelo crescimento que teve em um curto período de tempo, ultrapassando em número de usuários outros navegadores já consolidados. A empresa também possui planos de um sistema operacional baseado no Google Chrome.

Conforme expresso por McCloud (2008) em sua tirinha apresentando os conceitos do novo navegador, o problema com os navegadores atuais, segundo os desenvolvedores do Google Gears³, é que eles são todos *single-threaded*⁴. Enquanto um código em JavaScript estiver

1 World Wide Web Consortium. <http://www.w3.org/>

2 Web Hypertext Application Technology Working Group. <http://www.whatwg.org/>

3 <http://gears.google.com/>

4 Termo em língua inglesa que se refere a softwares que executam sequencialmente em um único processo, sem suporte a execução de tarefas em paralelo.

executando o navegador não pode realizar nenhuma outra tarefa. O Gears é um software que adiciona funcionalidades aos navegadores existentes.

O Chrome se destaca dos outros navegadores pela velocidade com que seu interpretador executa códigos JavaScript. Chamada de V8, ele compila JavaScript em código nativo de máquina antes de executá-lo além de implementar otimizações relacionadas à classificação de objetos em classes.

2.2 Motores de Renderização

O motor de renderização é um dos componentes de software do navegador, renderiza e posiciona elementos na página, convertendo o conteúdo lido da Web (arquivos HTML, CSS e JavaScript) em conteúdo formatado pronto para ser exibido em uma tela. Existe uma grande variedade de motores de renderização e geralmente cada navegador possui o seu próprio. Mas como o modelo de software livre está consolidado, muitos softwares de renderização são baseados parcialmente no código de outros navegadores.

Entre os principais motores de renderização, estão:

- **Gecko:** É o motor de renderização gráfica do Firefox, distribuído como um software livre. Teve como base o trabalho iniciado no Navigator e inicialmente era chamado de NGLayout. Foi projetada para seguir os padrões internacionais estabelecidos pelo W3C. No dado momento suporta HTML4, CSS 2.1, JavaScript 1.8, DOM níveis 1 e 2 entre outras tecnologias e fornece suporte parcial a HTML5, CSS3, ECMAScript5 e DOM 3.
- **Trident:** Usado no Internet Explorer e também compartilhado com o software de leitura de e-mails e com algumas ferramentas do sistema operacional da mesma empresa proprietária deste navegador. É o mais criticado pelo suporte limitado aos padrões abertos da Web e pela inclusão de uma grande quantidade de extensões proprietárias.
- **Presto:** É usado exclusivamente no navegador Opera, atualmente é o motor de renderização que oferece suporte mais completo ao HTML5.
- **KHTML:** Desenvolvido pelo projeto KDE, também é software livre e usado no navegador Konqueror.

- **WebKit:** Alguns navegadores compartilham um mesmo motor de renderização, como é o caso do Safari e do Google Chrome, ambos utilizam o WebKit que foi desenvolvido a partir do KHTML.

2.3 JavaScript

JavaScript é basicamente a linguagem para *scripting da Web* (Seeley, 2008). Permitindo a manipulação dinâmica dos elementos que compõem as páginas. A linguagem é um dialeto do padrão ECMAScript, pois inclui extensões na linguagem padrão. Surgiu primeiramente no Netscape para permitir a validação de dados de formulários. A linguagem JavaScript é oficialmente gerenciada pela Fundação Mozilla.

JavaScript é uma linguagem orientada a objetos desenvolvida em 1995 por Brendan Eich na Netscape para permitir as pessoas sem profundos conhecimentos de programação estenderem sites com código que executasse no cliente. (Richards *et al.*, 2010) Diferente de outras linguagens, como Java ou C#, ela não possui classes. No lugar disso JavaScript prioriza a flexibilidade, atributos e métodos podem ser incluídos em um objeto a qualquer momento.

A linguagem tornou-se um inegável sucesso e está se tornando uma plataforma para computação de propósito geral, sendo considerada a “linguagem de montagem” da Internet. Devido ao seu sucesso, JavaScript começou a ganhar atenção e respeito do meio acadêmico. As pesquisas se concentram principalmente em segurança, estabilidade e performance. Depois de muitos anos estagnados as modernas implementações de interpretadores JavaScript começaram a usar o estado da arte das técnicas de compilação *just-in-time*⁵. (Gal *et al.* in Richards *et al.*, 2010)

2.3.1 Interpretadores de JavaScript

Os navegadores modernos separaram o motor de renderização do interpretador de JavaScript e implementam variações do conceito de compilação *just-in-time*. Convertendo em tempo de execução o código JavaScript em linguagem de máquina.

⁵ Termo em língua inglesa que se refere ao processo de gerar código de máquina dinamicamente durante a execução do programa.

Um interpretador de JavaScript é um software especializado que processa código na linguagem JavaScript. O primeiro interpretador foi o SpiderMonkey desenvolvido na Netscape, e herdado pela Fundação Mozilla.

A seguir, são apresentados os principais interpretadores de JavaScript.

- **SquirrelFish:** Desenvolvida pela mesma equipe do WebKit, o SquirrelFish foi re-escrito a partir do JavaScriptCore que era baseada no KJS, o interpretador JavaScript do navegador Konqueror. O SquirrelFish é um interpretador de *bytecode* e por isso proporciona performance superior ao interpretador JavaScriptCore. Em 2008 o projeto se converteu no SquirrelFish Extreme também conhecido como Nitro, que compila os *scripts* em código de máquina nativo.
- **V8:** O projeto do V8 começou em 2006 com o objetivo de incrementar a performance dos interpretadores de JavaScript para que complexos programas orientados a objeto executem rápido no Google Chrome. Foi o pioneiro em desenvolver uma máquina virtual que compila o código desta linguagem em código de máquina. O V8 associa classes (no estilo Java) a objetos assumindo que eles não mudarão muito.
- **SpiderMonkey:** É o nome do primeiro interpretador de JavaScript escrita por Brendan Eich na Netscape⁶, atualmente mantida pela Fundação Mozilla. É composto por um compilador, interpretador, descompilador e coletor de lixo.

O TraceMonkey adiciona otimizações com “Trace Trees” no Spider Monkey, otimizando *loops* que são executados frequentemente. “Trace Trees” se refere a técnica de compilar parte do código durante a execução depois de detectado que ela é executada constantemente, expandindo árvores de execução separadas em cada ponto em que uma decisão do tipo então-senão é tomada.

Uma nova versão está em desenvolvimento chamada de JägerMonkey. Ela pretende incrementar a performance do SpiderMonkey nos pontos em que o TraceMonkey falha usando o montador do Nitro (o interpretador de JavaScript do WebKit) para gerar código de máquina.

⁶ SPIDERMONKEY. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2010. Disponível em: <http://en.wikipedia.org/w/index.php?title=Spider_monkey&oldid=362745132>. Acesso em: 20 maio 2010.

- **Chakra:**

É um interpretador da linguagem JScript⁷ criado pela Microsoft que está sendo desenvolvida para o Internet Explorer 9. Seu diferencial é compilar os *scripts* em um núcleo separado do processador e prover acesso ao processador gráfico (GPU)⁸.

- **Carakan:**

Carakan é o interpretador JavaScript usado pelo Opera. Ele também é baseado na classificação automática de objetos em classes, na geração dinâmica de *bytecode*⁹ e código nativo.

7 O nome JScript é marca registrada da Microsoft e se refere a um clone da linguagem JavaScript.

8 "Internet Explorer 9 showcases 'GPU-powered' browsing" disponível em: <<http://www.ibtimes.co.uk/articles/20100316/internet-explorershowcases-039-gpu-powered-039-browsing.htm>>. IBTimes. 2010

9 *Bytecode* é um estágio intermediário entre o código fonte e o código compilado que não é imediatamente executável e será executado por uma máquina virtual.

3 HTML5

Esta seção pretende apresentar alguns dos conceitos introduzidos pelo HTML5 que foram utilizados na implementação deste trabalho.

Fraternali (2010) diz que em sua origem a rede mundial de computadores era uma plataforma para acesso de conteúdo codificado em HTML, limitando a interação do usuários a navegar por ligações a outras páginas e preencher dados em formulários. Uma arquitetura simples e universal que não requeria quase nenhuma instalação de local de software, mas que severamente limitava a qualidade das aplicações que poderiam ser disponibilizadas pela internet.

Devido as limitações desta linguagem, extensões de terceiros foram incorporadas aos navegadores. Estas extensões introduziram funcionalidades extras que logo passaram a ser utilizadas por grande parte dos *sites*. Muitos destes novos recursos estão sendo padronizados e incluídos nativamente nos navegadores atuais.

HTML5 é uma proposta de padrão para substituir o HTML 4.01 bem como o XHTML 1.0¹⁰ e DOM¹¹ nível 2. Pretende reduzir a necessidade de *plug-ins* proprietários como Flash e Java por parte das aplicações Web. Começou a ser especificada em 2004, em 2007 o W3C adotou esta especificação como base dos trabalhos de padronização do novo HTML.

Mansfield-Devine (2010) diz que o HTML5 não é apenas mais uma coleção de novos marcadores e sim que ele provê novos mecanismos para transferência e armazenamento de dados, apresentação de vídeo, som, animações, novas fontes tipográficas de texto e novos mode-

¹⁰ O XHTML é uma reformulação do HTML que segue as regras do XML rigorosamente. O HTML5 não requer que seus documentos sejam XML.

¹¹ Modelo de objetos exposto por um documento que permite sua manipulação.

los de disposição da estrutura gráfica das páginas (*layout*). A preocupação principal não é com a tecnologia mas em como as pessoas a utilizam. A tecnologia responsável por estas funcionalidades já existe. Há uma grande variedade de extensões que implementam as mesmas funcionalidades, porém sem uma padronização.

O W3C estima que apenas por volta de 2012 a especificação final do HTML5 será retificada. Mas muitos elementos do HTML5 já são suportados pela maioria dos navegadores, entre as funcionalidades adicionadas estão:

- Elemento **canvas** para renderização de gráficos bidimensionais. Inicialmente introduzido no WebKit pela Apple para potencializar mini-aplicações conhecidas como *widgets* fornecidas pelo seu sistema operacional. Logo foi adotado pelo Gecko, pelo Opera e padronizado pela WHATWG.
- Elementos **video** e **audio** permitem a reprodução de vídeos e áudio respectivamente, incorporando no navegador a infra-estrutura requerida para execução destes recursos.

3.1 Canvas

O canvas é um elemento gráfico que pode ser embutido na página, ocupa uma região retangular e provê métodos que operam sobre uma imagem bidimensional. É dependente da resolução sendo utilizado para renderizar gráficos e imagens, podendo modifica-las durante a execução. Cada elemento canvas disponibiliza um contexto de renderização que representa uma superfície de desenho plana acessível por coordenadas cartesianas, cuja origem (0, 0) corresponde ao canto superior esquerdo.

Todos os principais navegadores implementam o elemento canvas, exceto o Internet Explorer. Entretanto, um projeto nomeado ExplorerCanvas¹² o implementa usando outros recursos gráficos disponibilizados pelo navegador.

Dentre as funcionalidades disponibilizadas pelo contexto de renderização do canvas, as principais são:

¹² <http://code.google.com/p/explorercanvas/>

- Desenho de figuras geométricas. As figuras representáveis são: Retângulo - definido pela coordenada do canto superior direito, largura e altura; Elipse - definida pelo retângulo na qual está contida; Polígono - definido por uma coleção de coordenadas; Outras formas podem ser desenhadas combinando curvas diversas e preenchendo a área interna delimitada por estas linhas.



Figura 1: Preenchimento degradê aplicado a um retângulo



Figura 2: Preenchimento degradê aplicado a um círculo

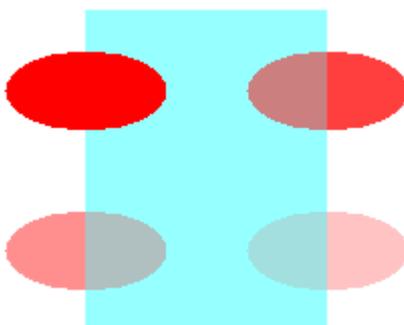


Figura 3: Figuras desenhadas com diferentes níveis de opacidade

- Preenchimento de figuras geométricas em degradê de cores como visto na Figura 1 e na Figura 2.
- Suporte a operações para composição de imagens, como transparência. Mostrada na Figura 3.

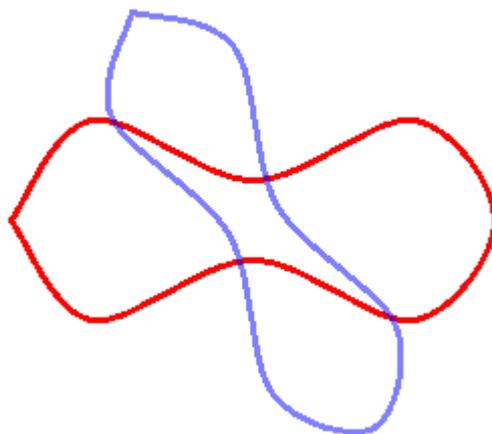


Figura 4: Várias curvas de Bézier.

- Desenho de segmentos de reta, arco, curvas quadráticas e curvas de Bézier como visto na Figura 4.
- Elementos tipográficos (texto) com alinhamento horizontal, vertical e definição do tipo de letra a ser utilizado.
- Configuração de cor, espessura e estilos do tracejado para o desenho de linhas.

Exemplos de estilos de traço:

Sólido	
Tracejado	
Pontilhado	
Traço-Ponto	
Traço-Ponto-Ponto	

Exemplos de estilo da intersecção e delimitadores de pontas nas linhas, mostrados nas figuras Figura 5 e Figura 6.



Figura 5:
Sequência de retas unidas com um canto arredondado



Figura 6: Segmento de reta com estilos distintos aplicados a cada ponta

- Desenho de linha com ou sem suavização de serrilhado, mostrado nas figuras Figura 8 e Figura 9.
- Redimensionamento e desenho de imagens ou partes dela, como exemplificado na Figura 7;
- Aplicação de transformações nas coordenadas de desenho: escala, rotação e deslocamento. As transformações são refletidas em todas as operações gráficas subsequentes.
- Manipulação de *pixels*, funcionalidade introduzida e implementada recentemente pelos navegadores. Permite obter e fixar a cor de um ponto específico na área de desenho do canvas.



Figura 7: Seleção de uma área de uma imagem

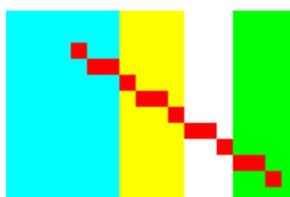


Figura 8: Linha desenhada sem suavização de serrilhado

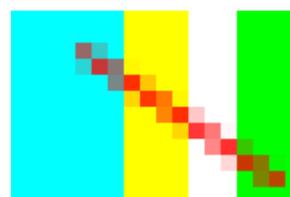


Figura 9: Linha desenhada com suavização de serrilhado

- Salvar e restaurar o contexto gráfico. Inclui configurações atuais do tipo de preenchimento, largura e estilo de tracejado da linha, fonte de texto, deslocamento, escala e rotação aplicadas as coordenadas.

Outra opção existente para a renderização de gráficos nos navegadores é o SVG, o padrão recomendado pela W3C para gráficos vetoriais. Porém toda figura desenhada precisa ser lembrada como um objeto na hierarquia de objetos da página, além de requerer que todas as figuras desenhadas previamente sejam redesenhadas quando há mudanças em alguma propriedade de algum elemento. O canvas provê um protocolo de desenho mais baixo nível, otimizado para a manipulação de imagens sendo superior ao SVG neste aspecto. Entretanto, é recomendável se trabalhar com SVG quando os elementos que precisam ser animados são essencialmente figuras geométricas.

3.2 Web Workers

Os navegadores sempre executaram o código das páginas em um processo único, sem paralelismo. Não forneciam mecanismos para permitir que um código embutido em uma página HTML pudesse separar a execução de tarefas mais complexas em Threads separadas da Thread que executa o ciclo de tratamento de eventos da interface. Logo, como o *script* executa na página concorrendo com a resposta a eventos, toda vez que uma operação ocupar muito tempo de processamento irá bloquear a interação do usuário com o ambiente gráfico do navegador.

Como alternativa pode-se temporizar a execução de tarefas para que elas sejam executadas em determinado intervalo de tempo repetidamente. Isso permite que no intervalo entre uma execução da tarefa e o próximo ciclo os eventos da interface sejam tratados. Porém, caso em algum destes períodos de execução um processamento pesado esteja executando, a navegação será bloqueada da mesma forma.

A alternativa disponível para execução de tarefas sem o bloqueio do ciclo de verificação de eventos de interação do usuário com a interface do navegador é separar fragmentos do código original que estava sendo executado pela página e o isolar em uma Thread ou processo separado do processo responsável por tratar eventos de interface. Para poder executar blocos de código isolados do ciclo de execução principal, era necessário possuir instaladas extensões no navegador. A primeira extensão que incluiu suporte a execução de fragmentos de código JavaScript concorrentes foi o Google Gears, introduzindo o conceito de WorkerPools.

Porém o Google Gears está sendo descontinuado em benefício dos novos conceitos apresentados pelo HTML5. Inspirado no Google Gears a WHATWG está trabalhando em uma especificação de uma API que provê suporte a concorrência nativa no interpretador JavaScript do navegador, permitindo que blocos de código sejam executados paralelamente por uma aplicação Web. A especificação ainda não faz parte do padrão HTML5 e poderá sofrer mudanças em sua estrutura atual, entretando já é implementada por navegadores em produção e está disponível e funcional. É implementado no Firefox a partir da versão 3.5 e no Google Chrome. Por ser compatível com o WorkerPool criado pelo Gears, os navegadores que ainda não intro-

duziram suporte a concorrência nas suas implementações de interpretadores de JavaScript, podem agregar esta funcionalidade com a instalação deste *plug-in*.

A especificação do WHATWG define uma API de Workers para executar *scripts* independentes da interface em plano de fundo. O navegador cria Threads reais a nível do Sistema Operacional para cada Worker que então executa um script independente do escopo da página por quem foi criado.

O Worker define um modelo de ambiente de execução totalmente isolado diferente do modelo de Threads disponibilizado por outras linguagens de programação populares. Um Worker não pode compartilhar nenhum objeto com outros Workers nem com a entidade que o criou. Isso elimina os problemas de concorrência e necessidade de sincronismo ao acesso de atributos de objetos. Um Worker não tem acesso ao documento e não pode manipular a interface gráfica. É limitado e executar um *script* e fornecer retorno enviando mensagens assíncronas que são enfileiradas e tratadas quando o receptor não estiver ocupado.

A API de comunicação define um modelo de troca de mensagens de texto. Um Worker pode disparar uma mensagem enviando um texto para ser tratado pela entidade que o criou. A entidade que o criou também pode se comunicar com o Worker pelo mesmo método, enviando uma mensagem textual. O Worker fica aguardando o recebimento de uma mensagem. Caso esteja ocupado no seu script de inicialização ou tratando outra mensagem, as novas mensagens são enfileiradas e aguardam até que o Worker possa processá-las.

Esse modelo se assemelha muito com o modelo de Atores proposto pelo Telis, onde as entidades ativas são independentes e se comunicam por troca de mensagens.

Um exemplo pode ser visto na listagem 1. Este fragmento de código cria um worker, associa uma função a um tratador de mensagens e envia uma mensagem para o worker.

```
var worker = new Worker("worker.js");  
  
worker.onmessage = function (evento){}  
  
worker.postMessage("mensagem");
```

Listagem 1: exemplo na de criação e comunicação com um Worker.

Este modelo de concorrência é limitado visto que não provê um mecanismo para checagem de novas mensagens enquanto uma mensagem esta sendo tratada. A presença de um mecanismo destes permitiria a reentrância no ciclo de checagem de mensagens. No modelo atual, novas mensagens só são tratadas quando o Worker está ocioso.

Para solucionar este impecílio, pode ser utilizado um misto de funções temporizadas dentro do Worker. Assim o Worker não bloqueia a recepção de mensagens por estar ocupado tratando apenas uma mensagem, mas executa pequenos fragmentos em intervalos de tempo pequenos. No intervalo que o worker permanece ocioso, entre uma execução da função temporizada e outra, pode tratar as mensagens enfileiradas permitindo a comunicação com o worker mesmo quando ele ainda não completou o tratamento completo de uma mensagem anteriormente recebida, que está aguardando um novo sinal para prosseguir.

Está solução não é a ideal mas enquanto não é especificado um modelo substituto para tratar mensagens pendentes em momento oportuno durante o tratamento de mensagens ou em um ciclo inserido na inicialização, ela fornece uma alternativa.

4 PORTE

A máquina que roda os apiques do Telis foi reescrita em JavaScript para que execute nativamente no navegador sem requerer *plug-ins* adicionais. A linguagem foi escolhida pois oferece todos os recursos necessários para a execução da máquina Telis inclusive suporte a paralelismo, de forma que o único requisito para a execução de apiques em Telis é um navegador de internet em sua instalação padrão. O porte foi efetuado com o auxílio do Google Web Toolkit (GWT), que faz a compilação cruzada de código Java para o JavaScript. Foi implementado um ambiente de execução Web para a máquina Telis baseado na estrutura abstrata de ambientes definida pela máquina, no qual também foi incluído mecanismos gráficos baseado em *turtle-graphics*¹³.

As seções a seguir descrevem as ferramentas utilizadas para realizar este porte, discursando sobre a estrutura da máquina Telis e como ela foi adaptada para o modelo de concorrência disponibilizado nos navegadores.

4.1 GWT

O Google Web Toolkit é uma nova tecnologia que automaticamente traduz código na linguagem Java em código JavaScript. O GWT resolve automaticamente inconsistências entre os navegadores gerando versões do código específicas para cada plataforma.

O cerne do GWT é um compilador que converte uma aplicação Java em uma equivalente em JavaScript com suporte automático para os navegadores: Internet Explorer, Firefox, Mozilla, Safari e Opera. A aplicação final gerada é totalmente independente da aplicação Java que a deu origem.

¹³ Método de programação de gráficos vetoriais usando um cursor que é movimentado sobre um plano cartesiano.

Suporta grande parte da sintaxe e da semântica da linguagem Java, com algumas exceções:

- Sem suporte a reflexão: porém bibliotecas de terceiros implementam esta funcionalidade, neste trabalho foi utilizada a biblioteca `gwt-ent`¹⁴.
- Sem paralelismo, não há implementação de Threads.
- Expressões regulares podem não ter o mesmo significado em Java e JavaScript.

4.2 Estrutura da máquina Telis

A Máquina Telis é dividida em três grandes partes:

- **Wrappers:** biblioteca hierárquica de tipos primitivos. Forcena os tipo básicos para se operar na linguagem Telis: Número, Texto, Símbolo, Listas, Listas de comandos, etc.
- **Coração da máquina:** *framework* que oferece a estrutura da linguagem. Controla o fluxo de execução realizando o *parsing* da entrada oferecendo a infraestrutura para a interpretação de primitivas, herança, escopo de variáveis e comunicação de atores. Oferece mecanismos para inclusão de novas primitivas e criação de novos ambientes de execução. O parser é gerado com o JavaCC e tem a vantagem de não depender de nenhuma outra biblioteca.
- **Ambientes de execução:** especializam o coração da máquina o estendendo. Exemplos de ambientes já implementados são o ambiente console e o ambiente gráfico. Neste trabalho implementou-se um ambiente de execução em um navegador de internet.

4.3 Problemas encontrados durante a conversão do projeto

As seções subsequentes descrevem os problemas encontrados durante a tradução do código Java em JavaScript citando as soluções encontradas.

¹⁴ Disponível em: <http://code.google.com/p/gwt-ent/>

4.3.1 Classes não implementadas pelo GWT.

Algumas classes não são implementadas pelo GWT, mas existem outras com funcionalidade equivalente. A Tabela 1 mostra as classes que foram substituídas.

Classes não implementadas pelo GWT e soluções alternativas.	
<code>java.text.NumberFormat</code>	Substituído por <code>com.google.gwt.i18n.client.NumberFormat</code> .
<code>Double.doubleToLongBits</code>	Usado no método <code>hashCode</code> . A implementação do método foi alterada para não depender desta função.
<code>java.security.InvalidParameterException</code>	Substituída por <code>IllegalArgumentException</code>
<code>java.util.regex.Pattern</code>	Substituído por <code>String.matches</code> , a expressão regular também precisou ser alterada.
<code>java.io.StringReader</code>	Implementado um novo <code>StringReader</code> , com a mínima funcionalidade exigida para seu funcionamento
<code>System.getProperty("line.separator", "\n")</code>	Substituir por <code>"\n"</code>
<code>java.util.ResourceBundle</code>	Substituir pela abordagem <code>i18n</code> equivalente do GWT
Testes Unitários	Re-escritos conforme o modelo de testes do GWT
<code>java.util.concurrent</code>	Foi necessário remodelar a estrutura de execução para que a dependência deste pacote pudesse ser eliminada.
<code>java.util.WeakHashMap</code>	Substituído por <code>java.util.HashMap</code>
<code>java.lang.Thread</code>	Foi necessário remodelar a estrutura de execução para que os atores também pudessem executar em <code>Workers</code> , não dependendo exclusivamente de <code>Threads</code> .
<code>java.lang.reflect</code>	Substituído pela API de reflexão disponibilizada pela biblioteca <code>gwt-ent</code> . Visibilidade de classes que precisam fornecer reflexão alterada para pública e marcadas com <code>@Reflectable</code> .
<code>Collections.synchronizedMap()</code>	suprimido
<code>classe.isInstance</code>	Substituído por um método equivalente da biblioteca <code>gwt-ent</code>
<code>Object.wait()</code>	Foi necessário remodelar a estrutura de execução para que a dependência deste método pudesse ser eliminada.
<code>Object.notify()</code>	Foi necessário remodelar a estrutura de execução para que a dependência deste método pudesse ser eliminada.

Tabela 1: Classes e métodos não implementados pelo GWT que foram substituídos.

4.3.2 Reflexão

A classe `Executor de Operação` definida na Máquina `Telis` usa reflexão para procurar um método com o nome `executar` que tenha um par de parâmetros de tipos específicos, como exemplificado no Listagem 2.

A biblioteca gwt-ent pode localizar o método corretamente se a classe contiver mais de um método executar com parâmetros diferentes, porém devido a uma deficiência no conhecimento do código compilado sempre um mesmo método é executado.

```
public void executar( IPilha pilha, Numero numero1, Numero numero2)
```

Listagem 2: método executar de um executor de Operações

A solução encontrada foi substituir o nome executar por uma anotação. Cada método de operação tem um nome distinto. Foi criada a anotação `@Operacao` que informa quando um método pertencente a um executor de operação implementa alguma operação. Como pode ser visto na Listagem 3.

```
public class Soma extends ExecutorDeOperacao {

    @Operacao
    public void concatenar(IPilha pilha, Texto texto1,
        Texto texto2) {}

    @Operacao
    public void somar(IPilha pilha, Numero numero1, Numero
        numero2) {}

    @Operacao
    public void concatenarTextoComNúmero(IPilha pilha,
        Texto texto, Numero numero) {}

    @Operacao
    public void concatenarNúmeroComTexto(IPilha pilha,
        Numero numero, Texto texto) {}

    @Operacao
    public void ou(IPilha pilha, Booleano booleano1,
        Booleano booleano2) {}

}
```

Listagem 3: Exemplo da classe Soma com métodos anotados como executores de operação

4.3.3 Internacionalização

O modelo de internacionalização adotado pelo GWT é diferente da abordagem comumente usada em Java. Em Java costuma-se usar um `ResourceBundle` para recuperar versões internacionalizadas de textos. O modelo usado pela máquina Telis que pode ser visto na Listagem 4 é baseado no modelo tradicional. O GWT implementa outro estilo para fornecer informações de internacionalização que se assemelha ao fragmento de código exibido na Listagem 5. A solução encontrada, foi mesclar os dois modelos em um modelo híbrido, que pode ser visto na Listagem 7. Neste modelo o método `obterString` original chama um método `getString` fornecido pela implementação do GWT.

```
public class Constantes {

    public static String obterString(String key) { (...) }

}

Constantes.obterString("nome")
```

Listagem 4: Classe da implementação original do Telis que retorna um texto internacionalizado.

```
public interface Modelo extends Constants {

    String nome();

}

GWT.create(Modelo.class).nome()
```

Listagem 5: Interface que segue o modelo de internacionalização do GWT e define um método que retorna um texto internacionalizado.

```

public class Linguagem

extends ConstantsWithLookup

implements Modelo {...}

public interface Modelo {

    @Key("Modelo.nomeAgendaIniciar")

    String nomeAgendaIniciar();

}

```

Listagem 6: Modelo híbrido que incorpora o modelo original da máquina Telis no modelo adotado pelo GWT

4.3.4 Serialização

A comunicação entre atores que executam em Workers, descrita com mais detalhes na página 34, requer que estruturas da linguagem sejam serializadas em texto para que possam ser enviadas para os atores.

Os módulos de serialização disponíveis para o GWT requerem que as classes implementem uma interface `Serializable`, tenham um construtor vazio e métodos para obter (*getters*) e fixar (*setters*) os atributos da classe. A fatoração de um número considerável de classes necessária para a adequação a este modelo não era viável e faria o código dessoar do padrão do restante do projeto. Preferiu-se implementar classes auxiliares que implementam a serialização dos contêineres de estruturas da linguagem Telis, como mostrado na Listagem 7.

```

Palavra texto = new Texto("texto");
JSONValue palavraSerializada =
    SerializadorDePalavras.serializarPalavra(texto);

Palavra mesmoTexto = SerializadorDePalavras.
    construirPalavra(palavraSerializada);

```

Listagem 7: Exemplo de serialização de um Texto

4.3.5 Concorrência

A Máquina Telis possui grande dependência de Threads para implementar a concorrência nos atores. Para que a compilação com o GWT fosse possível, foi necessário separar o modelo atual e isolar as partes dependentes de Threads para que uma implementação baseada em Workers pudesse ser desenvolvida sem interferir no restante do projeto.

A classe ator que representar um ator do Telis e o executa foi fatorada. Dela foi extraído um ator executável em Thread e outro ator executável em Worker que estendem um Ator abstrato em comum. A implementação do ator executável em Worker é separada em duas partes, uma que efetivamente executa o ator dentro do Worker enquanto outra classe apenas representa o ator no escopo da página repassando os comando para o executor por mensagens. O Worker também possui referência aos outros atores criados por ele através da classe RepresentacaoDoAtorCriadoNaPagina. A página é o escopo global responsável pela criação dos atores.

```
Ator ator = new AtorExecutavelEmWorker(modelo, identifi-
cador, parâmetro);
```

```
ator.iniciarExecução();
```

Listagem 8: instanciação de ator executável em Worker

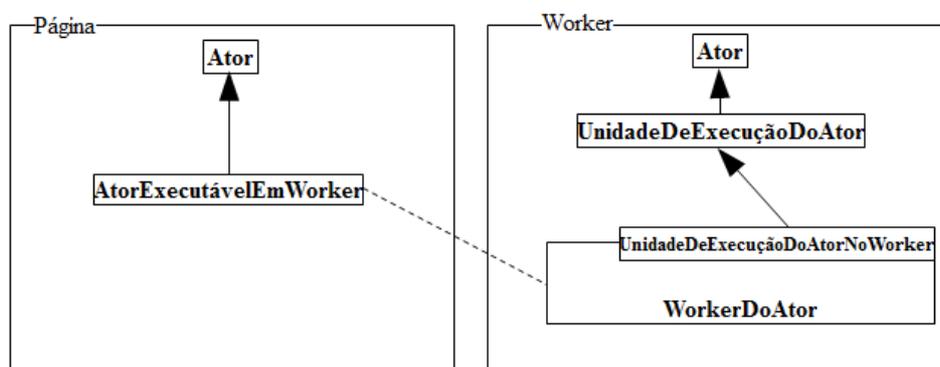


Diagrama 1: Separação dos módulos de execução do ator em Worker

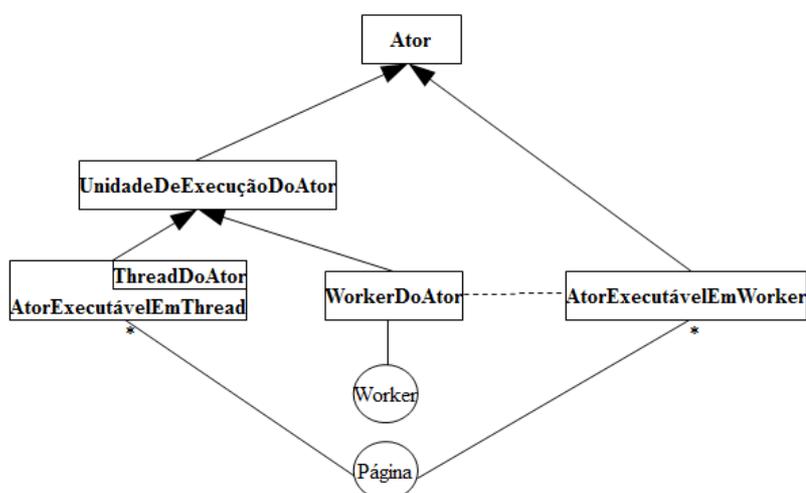


Diagrama 2: Separação de classes do Ator

4.3.6 Primitivas Gráficas:

Muitas primitivas não podem ser executadas pelo ator dentro do Worker devido a imposição de limitações no acesso ao escopo global do documento. Como não se possui acesso a página não é possível manipular o canvas. Por isso a execução de primitivas gráficas que manipulam a interface gráfica visível pelo usuário teve que ser separada em dois estágios.

No primeiro estágio a primitiva é executada internamente no Worker. Nesta etapa da execução, a primitiva apenas serializa seus parâmetros que são repassados a página pelo protocolo de comunicação do Worker.

No segundo estágio a página recebe um comando de execução de primitiva. Na página são implementadas versões executáveis das primitivas gráficas serializadas.

4.4 Abordagem adotada para comunicação

As próximas seções descrevem o modelo de comunicação entre o Worker e a Página, quais as limitações impostas por esta estrutura e os caminhos encontrados para desviá-las.

4.4.1 Comunicação bidirecional entre a Página e o Worker

Para que a página e o Worker possam tratar mensagens eles devem estar ociosos. Isso impede que o ator executando dentro do Worker possa executar todo o código sem interrupções dentro de uma estrutura de repetição. Na implementação existente da máquina Telis, ao ser criado um ator é executado a sua agenda iniciar. Este método que executa a agenda iniciar só retorna depois que todo o código for executado, inibindo assim o Worker de tratar as mensagens enviadas pela página.

Uma abordagem que pode ser considerada para se contornar essa limitação é executar o código do ator em blocos, um de cada vez. Uma função em JavaScript pode ser agendada para executar com um atraso de tempo simulando um temporizador. Para isto é necessário alterar a máquina de pilha, que é a classe responsável pelo controle de execução do código para executar iterativamente com um método executarPróximaInstrução.

Porém uma das complicações na implementação desta máquina iterativa advém do fato de algumas primitivas serem reentrantes. Elas executam blocos de instruções chamando o método executar comandos da máquina de pilha repetidas vezes. Por isso é necessário armazenar a posição que se estava executando para poder voltar para este ponto depois que a lista de comandos nova terminar de executar, além de armazenar o escopo e outras informações do estado atual da máquina naquele momento da execução.

Algumas instruções só podem ser chamadas depois que o código foi realmente executado. Quando é executada uma agenda é criado um novo escopo que deve ser restaurado depois que a agenda terminar de executar. Métodos de rastreamento, como avisar um gerente de retorno que um método terminou de executar ou tratar uma exceção registrando um erro e encerrando a execução só devem ser executados depois que a função executou realmente e não depois que ela foi agendada. A execução passo a passo precisa tratar corretamente estes pon-

tos executando trechos de código de inicialização e finalização para cada agenda, estrutura de repetição e lista de comandos.

Algumas primitivas precisam ser reimplementadas para se adequarem a execução passo a passo, na qual o código não é executado no momento em que é chamado o método executar da maquina de pilha, mas sim só quando o executarPróximaInstrução é chamado. As primitivas que associam valores temporários só podem fazer essa associação quando é requisitada a execução da próxima instrução. Um trecho de código do agendador de tarefas implementado pode ser visto na Listagem 9.

```
agendar(númeroDeRepetições, new ComandoDeferido() {
    public void executar(int repetição) {}
    public void finalizar() {}
    public void tratarErro(ExceçãoTelis erro) {}
});
```

Listagem 9: Agendamento de um comando para posterior execução.

Outro método que precisou ser modificado foi o método avaliarExpressão usado por primitivas para obter o resultado de uma expressão. O resultado da expressão sendo avaliada não é conhecido no momento que se solicita a execução pois os comando serão empilhados para execução futura. Foi adicionado um método avaliar que não retorna o valor imediatamente mas recebe como parametro adicional uma função de resposta que será executada quando o resultado estiver disponível, como pode ser visto na Listagem 10.

```
controlador.avaliarExpressao(closure, new Resposta() {
    public void receberResposta(Palavra resultado) {}
}, parametros);
```

Listagem 10: Função avalia o resultado de uma expressão assíncronamente.

Outra dificuldade encontrada, foi em relação as primitivas que precisam conhecer o valor de alguma propriedade que não está disponível dentro Worker. Elas precisam ser executadas pela página, que possui acesso ao valor da propriedade, exigindo que se aguarde até a página retornar um resultado para poder prosseguir com a execução. Neste intervalo o Worker precisa ficar ocioso, para que possa tratar a mensagem enviada pela página contendo a resposta.

Toda vez que for necessário fazer alguma pergunta para a página, descobrir se existe um molde ou modelo com um dado nome por exemplo, o ator irá pedir para a página esta informação. A página é a entidade que centraliza todos os recursos e contém todos os Modelos. Outro exemplo pode ser exemplificado por um ator que precisa criar outro ator, os atores devem ser criados pela página para que esta possa se comunicar com eles posteriormente. Então é preciso aguardar a página criar o ator e responder com a identidade. Assim a chamada para criação de um novo ator só pode retornar depois que a página responder com a identidade do ator recém criado.

Para se contornar este problema foi elaborado uma abordagem. Esta abordagem consiste em gerar uma exceção de um tipo Pergunta quando é preciso interromper a execução e requisitar a página a execução de determinada tarefa. Isso fará com que o fluxo de execução retorne até chegar na função que controla a execução passo a passo, o método que chama o executarPróximaInstrução. Neste momento trata-se esta exceção de pergunta enviando para a página um pedido contendo o identificador de qual tarefa deverá ser executada pela página e os parâmetros encapsulados pela pergunta, conforme exemplificado na Listagem 12, congelando a execução passo a passo e colocando o Worker em estado de espera.

Quando a página estiver ociosa, pois não está tratando um pedido de nenhum outro ator, ela irá responder. O ator receberá a resposta e a armazenará associando àquela última pergunta. A próxima etapa é reabilitar o temporizador reiniciando a execução passo a passo. Como uma pergunta foi feita durante a execução da última primitiva, esta instrução não foi descartada e será re-executada. Quando a pergunta for feita novamente a resposta será encontrada já armazenada, nenhuma exceção será lançada e o método de pergunta retornará com a resposta. Um fragmento do código do projeto que implementa este modelo pode ser visto na Listagem 11.

Um dos problemas decorrente desta abordagem é que algumas primitivas tentarão consumir os parâmetros da pilha novamente devido a re-execução. Então é preciso isolar o trecho da primitiva que não remove elementos da pilha, o envolvendo em um bloco de comandos que terá sua execução deferida para que apenas esta região lógica interna da primitiva seja re-executada. Fazendo assim que a execução repetida aconteça apenas a partir do ponto em que os parâmetros já são conhecidos.

```
public Palavra perguntar(String identificador,
    Palavra parametro) {

    Pergunta pergunta = new Pergunta(
        identificador, parametro);

    if (sabeAResposta(pergunta))

        return obterResposta(pergunta);

    else

        throw pergunta;

}
```

Listagem 11: Método Perguntar que lança uma exceção Pergunta caso não saiba a resposta.

```

private void criarTimerDeExecução() {
    executor = new Temporizador() {
        public void run() {
            try {
                executarPróximaInstrução();
                agendarPróximaExecução(1);
            } catch (ExcecaoTelis e) {
            } catch (Pergunta pergunta) {
                enviarParaAPágina(
                    pergunta.obterIdentificador(),
                    pergunta.obterParametros());
            } catch (ExecutouAUltimaInstrucao e) {
            }
        }
    };
}

```

Listagem 12: Função temporizada da execução passo a passo.

Como complemento a este modelo, foi incluída a possibilidade de agendar comandos na máquina de execução iterativa que possuem um par pergunta e resposta. No primeiro momento é executado o método relativo a pergunta. O método que recebe a resposta só é executado quando a página devolver o valor requisitado. Neste intervalo de tempo a execução fica congelada. A abordagem inicial proposta, com o lançamento de exceções foi mantida pois interfere de maneira menos intrusiva na estrutura do programa, sendo aplicada nos métodos em que não há problemas em cancelar sua execução abruptamente a retomando do início posteriormente.

4.4.2 Comunicação assíncrona

A linguagem Telis define um modelo de comunicação assíncrona com as primitivas `dizer` e `seDito`. Um ator pode se cadastrar para receber estímulos externos, através da primitiva `seDito` que recebe uma tupla que representa um filtro e uma lista de comandos para tratar este estímulo. Todos os estímulos que passarem por esse filtro serão enviados para este ator. Se o ator não estiver tratando nenhum outro estímulo uma interrupção é gerada e o fluxo de execução é desviado para o tratador. Caso o ator esteja ocupado tratando um estímulo anterior, este novo estímulo gerado é perdido. Os estímulos são gerados pelos outros atores com a primitiva `dizer`, que recebe uma tupla com os parâmetros do estímulo a ser propagado. A Figura 10 exibe o fluxo de execução da propagação de um estímulo entre dois atores executando em Workers distintos com a comunicação sendo centralizada pela página.

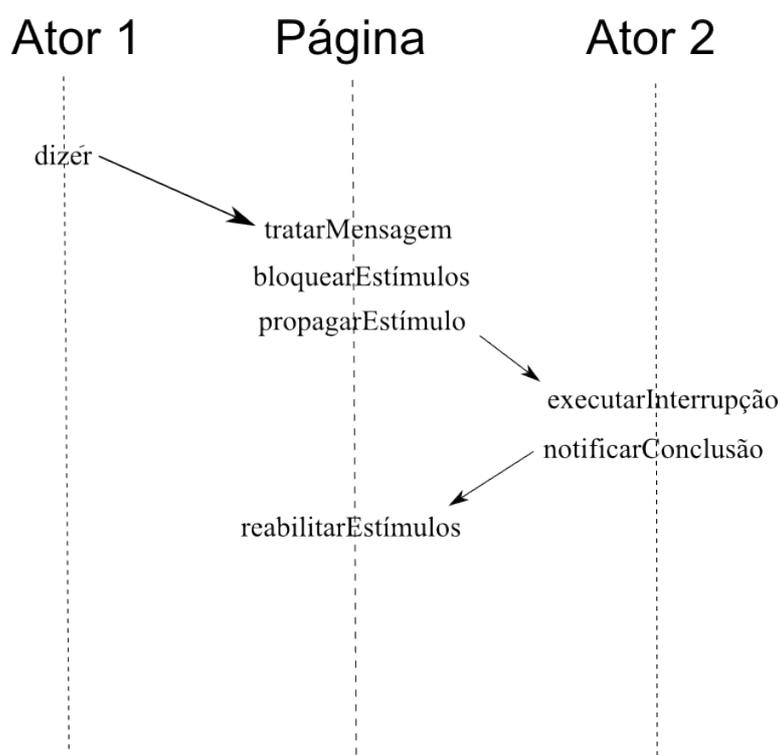


Figura 10: Diagrama de sequência da primitiva `dizer`.

4.4.3 Comunicação síncrona

A linguagem Telis define um modelo de comunicação síncrona baseado no operador ponto. Um ator pode requisitar a execução de uma agenda de outro ator caso conheça a identidade deste outro ator. A agenda que será chamada é precedida de um ponto, denotando que se deseja enviar uma mensagem para outro ator e aguardar até que este complete a execução e opcionalmente retorne valores como resposta. A Figura 11 descreve o fluxo de execução gerado pela comunicação direta entre dois atores executando em Workers distintos intermediada pela página.

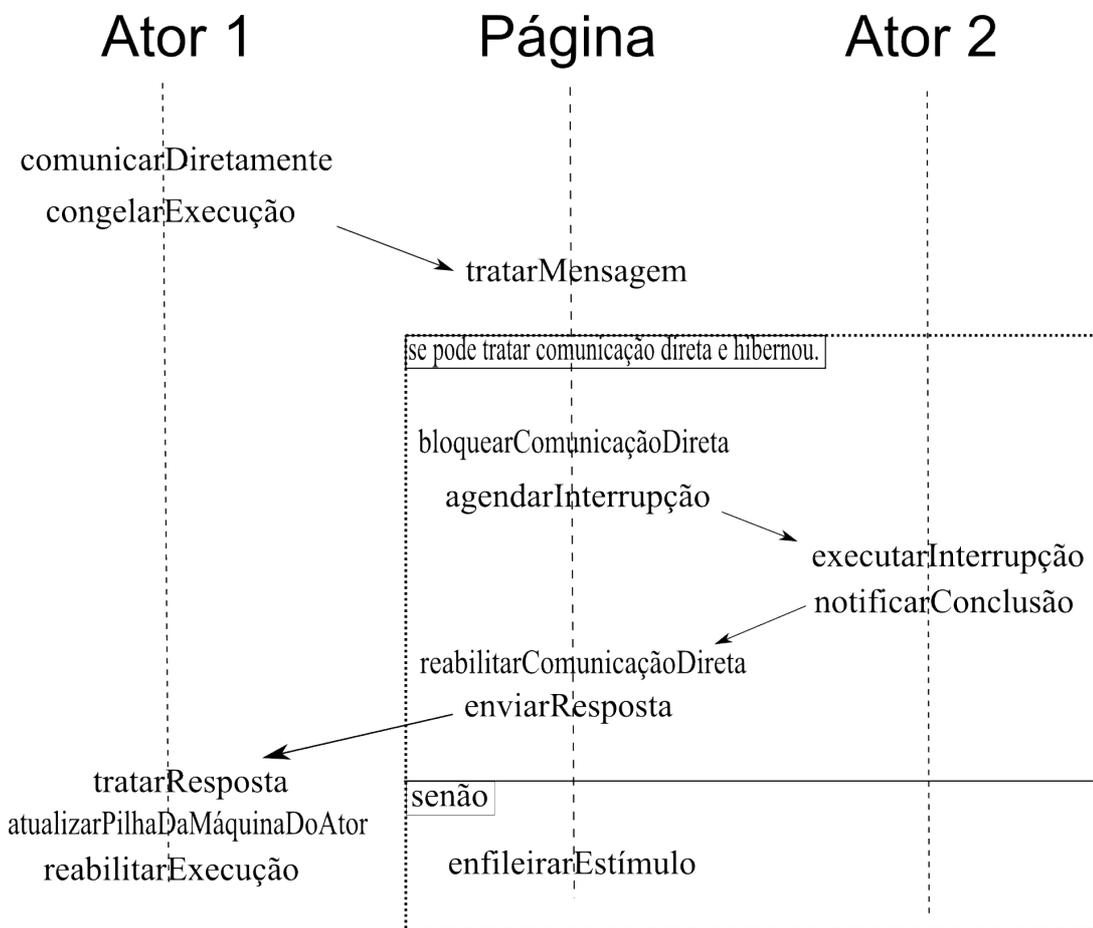


Figura 11: Diagrama de sequência do operador de comunicação direta, o ponto.

CONSIDERAÇÕES FINAIS

A experiência pessoal com a linguagem de programação Telis contribuiu com a minha formação sendo responsável por despertar uma maneira nova de pensar e criar programas de computador. Priorizando a legibilidade do código fonte, realizando a distribuição de responsabilidades as classes corretas, fatorando o código depois de escrito e principalmente a simplicidade.

Por Telis ser um ambiente nativamente Web sempre se desejou que ele estivesse integrado diretamente nos navegadores. No primeiro momento *applets* em Java cumpriram o seu papel mas adicionaram uma dependência que não é necessária, já que os navegadores podem proporcionar a Telis toda a infra-estrutura requerida.

O esforço de escrever uma máquina Telis nova, na linguagem nativa dos navegadores seria um trabalho árduo. Graças a todo o trabalho em equipe desenvolvido pela EDUGRAF o Telis atingiu um alto nível de estabilidade. Porém o tempo também o estagnou devido ao código legado que contém. A implementação de uma máquina Telis nova foi proposta de TCC anterior a este, trabalho esse de grande qualidade que forneceu uma coleção de projetos muito bem estruturados usado nesse trabalho como base para o porte realizado.

A conversão de código Java para JavaScript feita pelo GWT se mostrou de alta qualidade aliada a uma boa performance junto aos navegadores modernos. Um conjunto muito grande de classes Java pode ser convertido automaticamente com esta ferramenta.

As dificuldades enfrentadas com a paralelização da execução dos atores foi inferior a esperada, visto que o suporte a concorrência nos navegadores nem mesmo era possível a pouco tempo atrás. Quase todas as deficiências puderam ser contornadas. O principal desafio foi a implementação de sincronia com uso de assincronia.

Nem toda a gama de primitivas disponibilizadas pela máquina Telis foram convertidas neste porte, porém a infra-estrutura básica de todas elas está presente requerido apenas um esforço de implementação que não era possível realizar em tempo hábil até a conclusão deste trabalho.

Espera-se que esta versão de Telis com execução nativa nos navegadores possa representar um novo salto neste ambiente de programação o tornando mais popular e difundido, que ele realmente seja aplicado, os erros corrigidos, novas funcionalidades sejam adicionadas e tenha seu desenvolvimento mantido.

REFERÊNCIAS

FRATERNALI, Piero; ROSSI, Gustavo; SANCHEZ-FIGUEROA, Fernando. **Rich Internet Applications**. Ieee Internet Computing, [s. L.], p.9, maio 2010.

GROSSKURTH, Alan; GODFREY, Michael W.. **Architecture and evolution of the modern Web browser**. Waterloo, p.3, jun. 2006.

MCCLLOUD, Scott (Comp.). **Google on Google Chrome - comic book**. Disponível em: <http://www.google.com/googlebooks/chrome/big_02.html>. Acesso em: 1 out. 2008.

RICHARDS, Gregor; LEBRESNE, Sylvain; VITEK, Brian; JAN, **Burg**. **An Analysis of the Dynamic Behavior of JavaScript Programs**. p.1, 2010.

WHATWG (Org.). **The canvas element - HTML5**. Disponível em: <<http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html>>. Acesso em: 20 maio 2010.

WHATWG (Org.). **Web Workers**. Disponível em: <<http://www.whatwg.org/specs/web-workers/current-work/>>. Acesso em: 20 maio 2010.

ANEXO – ARTIGO

***Abstract.** The Web applications development usually require external browser plugins to support multithreading and graphical manipulation. This work intends to analyze the new features available in modern browsers and port the execution environment of Telis programming language, originally designed in Java, to run natively on browsers infrastructure.*

***Resumo.** O desenvolvimento de aplicações nos navegadores Web sempre dependeu de extensões de terceiros para realizar tarefas mais sofisticadas como multitarefa, manipulação gráfica e sonorização. Este trabalho visa explorar os novos recursos disponibilizados pelos navegadores modernos. O amadurecimento da infra-estrutura disponível nos navegadores de internet recentemente, permitiu que um ambiente de compilação e execução de uma linguagem de programação intrinsecamente paralela fosse portado do ambiente Java a qual foi inicialmente projetado para executar nativamente em um navegador.*

1. Introdução

Telis é uma ferramenta de aprendizado de programação desenvolvido pelo EDUGRAF. Composto por uma linguagem de programação e um ambiente de desenvolvimento especialmente projetado para estudantes que não tiveram contato com linguagens de programação. Telis diminui a curva de aprendizado de conceitos avançados como concorrência, computação distribuída e orientação a objetos reduzindo a carga conceitual requerida para colocá-los em prática, mesclando a assimilação destes conceitos com a etapa de aprendizagem do uso da ferramenta.

Esta trabalho pretende mostrar que é possível executar aplicaes escritos na linguagem Telis utilizando unicamente a infra-estrutura provida pelos navegadores de internet que adotam as novas tecnologias propostas pelo padrão HTML5, sem o uso de applets. Para isso, portou-se a máquina Telis da linguagem Java para JavaScript. Foram feitas pesquisas para encontrar meios de suportar concorrência e manipulação gráfica em uma página HTML. A aplicação foi testada nos principais navegadores em produção que estão em conformidade com os padrões vigentes na Web.

2. Telis

Telis é uma linguagem interpretada com tipagem dinâmica, implementada em Java e desenvolvida pelo laboratório EDUGRAF. A linguagem expõem de maneira natural paralelismo, princípios de orientação a objetos e conceitos de sistemas distribuídos permitindo que programadores inexperientes desenvolvam programas que manipulam objetos gráficos na tela, colhem entradas do usuário e se comunicam em rede.

A simplicidade de Telis a torna adequada para o aprendizado dos iniciantes. Todos os nomes de comandos são em português, todas as operações realizadas obtêm seus operandos de uma pilha, o mesmo local aonde os resultados das operações são armazenados. Para isso é utilizada a notação pós-fixa de operadores. Listas e operações sobre listas são partes da linguagem, assim como criação de variáveis, as quais os valores da pilha podem ser associados.

Concorrência também é tratada diretamente pela linguagem sem a necessidade de outros mecanismos de sincronização, encapsulada pelos atores que são autônomos e executam comandos em pilhas distintas de forma independente. A comunicação entre os atores pode ser síncrona ou assíncrona. Se for assíncrona, um ator repassa uma mensagem a todos os outros atores e continua sua execução normalmente. Caso os outros atores desejem tratar a mensagem recebida, interrompem sua tarefa atual, executam uma tarefa relacionada com a mensagem recebida e depois continuam com a execução da tarefa anterior do ponto em que foi interrompida. Se for síncrona, um ator requisita a outro ator específico que interrompa sua execução e trate uma requisição, ambos só continuam seu ciclo de execução normal quando essa requisição for tratada completamente.

Os comandos, em Telis conhecidos como primitivas, são agrupados em agendas similares aos métodos, procedimentos ou funções em outras linguagens de programação. Por sua vez, as agendas são nomeadas e agrupadas em Moldes e Modelos. Um Molde é semelhante a uma classe abstrata, que não pode ser instanciada. Modelos são subdivididos em Modelos de Objeto e Modelos de Ator. Modelos de Objetos são equivalentes a classes no paradigma de orientação a objetos e Modelos de ator são os equivalentes as classes executáveis em *Threads* ou processos separados. Ambos Moldes e Modelos podem herdar outros Moldes, em Telis é dito que um Molde molda o outro

Por fim, um Modelo de ator pode ser executado por vários atores. Atores são entidades ativas por onde a execução flui.

3. Navegadores e javascript como linguagem de montagem

Um navegador é um aplicativo que recupera e apresenta recursos disponíveis na internet. Um dos primeiros navegadores gráficos foi o Mosaic, que influenciou seus sucessores Netscape Navigator e Internet Explorer (Grosskurth *et al.*, 2006). Com o tempo, surgiram outros navegadores, como o Opera e o Mozilla, este último desenvolvido a partir do Netscape. Em 2003 o Safari foi criado, em 2004 o Mozilla deu origem ao Firefox. E recentemente o Google disponibilizou o seu navegador o Google Chrome.

Um dos componentes de software do navegador é o motor de renderização. Ele renderiza e posiciona elementos na página, convertendo o conteúdo lido da Web (arquivos HTML, CSS e JavaScript) em conteúdo formatado pronto para ser exibido em uma tela. Existe uma grande variedade de motores de renderização e geralmente cada navegador possui o seu próprio. Mas como o modelo de software livre está consolidado, muitos softwares de renderização são baseados parcialmente no código de outros navegadores.

A linguagem para scripting da Web é basicamente o JavaScript (Seeley, 2008). A linguagem é um dialeto do padrão ECMAScript, pois inclui extensões na linguagem padrão. Surgiu primeiramente no Netscape para permitir a validação de dados de formulários. A linguagem JavaScript é oficialmente gerenciada pela Fundação Mozilla.

JavaScript é uma linguagem orientada a objetos desenvolvida em 1995 por Brendan Eich na Netscape para permitir as pessoas sem profundos conhecimentos de programação estenderem sites com código que executasse no cliente. (Richards et al., 2010) Diferente de outras linguagens, como Java ou C#, ela não possui classes. No lugar disso JavaScript prioriza a flexibilidade, atributos e métodos podem ser incluídos em um objeto a qualquer momento.

A linguagem tornou-se um inegável sucesso e está se tornando uma plataforma para computação de propósito geral, sendo considerada a “linguagem de montagem” da Internet. Devido ao seu sucesso, JavaScript começou a ganhar atenção e respeito do meio acadêmico. As pesquisas se concentram principalmente em segurança, estabilidade e performance. Depois de muitos anos estagnados as modernas implementações de interpretadores JavaScript começaram a usar o estado da arte das técnicas de compilação just-in-time¹. (Gal et al. in Richards et al., 2010)

Os navegadores modernos separaram o motor de renderização do interpretador de JavaScript e implementam variações do conceito de compilação just-in-time. Convertendo em tempo de execução o código JavaScript em linguagem de máquina.

4. HTML5

Fraternali (2010) diz que em sua origem a rede mundial de computadores era uma plataforma para acesso de conteúdo codificado em HTML, limitando a interação do usuários a navegar por ligações a outras páginas e preencher dados em formulários. Uma arquitetura simples e universal que não requeria quase nenhuma instalação de local de software, mas que severamente limitava a qualidade das aplicações que poderiam ser disponibilizadas pela internet.

Devido as limitações desta linguagem, extensões de terceiros foram incorporadas aos navegadores. Estas extensões introduziram funcionalidades extras que logo passaram a ser utilizadas por grande parte dos *sites*. Muitos destes novos recursos estão sendo padronizados e incluídos nativamente nos navegadores atuais.

HTML5 é uma proposta de padrão para substituir o HTML 4.01 bem como o XHTML 1.0 e DOM nível 2. Pretende reduzir a necessidade de *plug-ins* proprietários como Flash e

Java por parte das aplicações Web. Começou a ser especificada em 2004, em 2007 o W3C adotou esta especificação como base dos trabalhos de padronização do novo HTML.

4.1. Canvas

O canvas é um elemento gráfico que pode ser embutido na página, ocupa uma região retangular e provê métodos que operam sobre uma imagem bidimensional. É dependente da resolução sendo utilizado para renderizar gráficos e imagens, podendo modifica-las durante a execução. Cada elemento canvas disponibiliza um contexto de renderização que representa uma superfície de desenho plana acessível por coordenadas cartesianas, cuja origem (0, 0) corresponde ao canto superior esquerdo.

Todos os principais navegadores implementam o elemento canvas, exceto o Internet Explorer. Entretanto, um projeto nomeado ExplorerCanvas o implementa usando outros recursos gráficos disponibilizados pelo navegador.

4.2. Web Workers

Os navegadores sempre executaram o código das páginas em um processo único, sem paralelismo. Não forneciam mecanismos para permitir que um código embutido em uma página HTML pudesse separar a execução de tarefas mais complexas em Threads separadas da Thread que executa o ciclo de tratamento de eventos da interface. Logo, como o script executa na página concorrendo com a resposta a eventos, toda vez que uma operação ocupar muito tempo de processamento irá bloquear a interação do usuário com o ambiente gráfico do navegador.

A alternativa disponível para execução de tarefas sem o bloqueio do ciclo de verificação de eventos de interação do usuário com a interface do navegador é separar fragmentos do código original que estava sendo executado pela página e o isolar em uma Thread ou processo separado do processo responsável por tratar eventos de interface. Para poder executar blocos de código isolados do ciclo de execução principal, era necessário possuir instaladas extensões no navegador.

A WHATWG está trabalhando em uma especificação de uma API que provê suporte a concorrência nativa no interpretador JavaScript do navegador, permitindo que blocos de

código sejam executados paralelamente por uma aplicação Web. A especificação ainda não faz parte do padrão HTML5 e poderá sofrer mudanças em sua estrutura atual, entretanto já é implementada por navegadores em produção e está disponível e funcional. É implementado no Firefox a partir da versão 3.5 e no Google Chrome.

O Worker define um modelo de ambiente de execução totalmente isolado diferente do modelo de Threads disponibilizado por outras linguagens de programação populares. Um Worker não pode compartilhar nenhum objeto com outros Workers nem com a entidade que o criou. Isso elimina os problemas de concorrência e necessidade de sincronismo ao acesso de atributos de objetos. Um Worker não tem acesso ao documento e não pode manipular a interface gráfica. É limitado a executar um *script* e fornecer retorno enviando mensagens assíncronas que são enfileiradas e tratadas quando o receptor não estiver ocupado.

5. Porte

A máquina que roda os aplicativos do Telis foi reescrita em JavaScript para que execute nativamente no navegador sem requerer *plug-ins* adicionais. A linguagem foi escolhida pois oferece todos os recursos necessários para a execução da máquina Telis inclusive suporte a paralelismo, de forma que o único requisito para a execução de aplicativos em Telis é um navegador de internet em sua instalação padrão. O porte foi efetuado com o auxílio do Google Web Toolkit (GWT), que faz a compilação cruzada de código Java para o JavaScript. Foi implementado um ambiente de execução Web para a máquina Telis baseado na estrutura abstrata de ambientes definida pela máquina, no qual também foi incluído mecanismos gráficos baseado em *turtle-graphics*.

5.1 Google web Toolkit

O Google Web Toolkit é uma nova tecnologia que automaticamente traduz código na linguagem Java em código JavaScript. O GWT resolve automaticamente inconsistências entre os navegadores gerando versões do código específicas para cada plataforma. O Google Web Toolkit é uma nova tecnologia que automaticamente traduz código na linguagem Java em código JavaScript. O GWT resolve automaticamente inconsistências entre os navegadores gerando versões do código específicas para cada plataforma. O cerne do GWT é um compil-

ador que converte uma aplicação Java em uma equivalente em JavaScript. Suporta grande parte da sintaxe e da semântica da linguagem Java, com algumas exceções:

- Sem suporte a reflexão: porém bibliotecas de terceiros implementam esta funcionalidade, neste trabalho foi utilizada a biblioteca gwt-ent.
- Sem paralelismo, não há implementação de Threads.
- Expressões regulares podem não ter o mesmo significado em Java e JavaScript.

5.2 Estrutura da máquina Telis

A Máquina Telis é dividida em três grandes partes:

- Wrappers: biblioteca hierárquica de tipos primitivos. Forcena os tipo básicos para se operar na linguagem Telis: Número, Texto, Símbolo, Listas, Listas de comandos, etc.
- Coração da máquina: framework que oferece a estrutura da linguagem. Controla o fluxo de execução realizando o parsing da entrada oferecendo a infraestrutura para a interpretação de primitivas, herança, escopo de variáveis e comunicação de atores. Oferece mecanismos para inclusão de novas primitivas e criação de novos ambientes de execução. O parser é gerado com o JavaCC e tem a vantagem de não depender de nenhuma outra biblioteca.
- Ambientes de execução: especializam o coração da máquina o estendendo. Exemplos de ambientes já implementados são o ambiente console e o ambiente gráfico. Neste trabalho implementou-se um ambiente de execução em um navegador de internet.

6. Considerações finais

Por Telis ser um ambiente nativamente Web sempre se desejou que ele estivesse integrado diretamente nos navegadores. No primeiro momento applets em Java cumpriram o seu papel mas adicionaram uma dependência que não é necessária, já que os navegadores podem proporcionar a Telis toda a infra-estrutura requerida.

A conversão de código Java para JavaScript feita pelo GWT se mostrou de alta qualidade aliada a uma boa performance junto aos navegadores modernos. Um conjunto muito grande de classes Java pode ser convertido automaticamente com esta ferramenta.

Referências

FRATERNALI, Piero; ROSSI, Gustavo; SANCHEZ-FIGUEROA, Fernando. **Rich Internet Applications**. Ieee Internet Computing, [s. L.], p.9, maio 2010.

GROSSKURTH, Alan; GODFREY, Michael W.. **Architecture and evolution of the modern Web browser**. Waterloo, p.3, jun. 2006.

MCCLLOUD, Scott (Comp.). **Google on Google Chrome - comic book**. Disponível em: <http://www.google.com/googlebooks/chrome/big_02.html>. Acesso em: 1 out. 2008.

RICHARDS, Gregor; LEBRESNE, Sylvain; VITEK, Brian; JAN, Burg. **An Analysis of the Dynamic Behavior of JavaScript Programs**. p.1, 2010.

WHATWG (Org.). **The canvas element - HTML5**. Disponível em: <<http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html>>. Acesso em: 20 maio 2010.

WHATWG (Org.). **Web Workers**. Disponível em: <<http://www.whatwg.org/specs/web-workers/current-work/>>. Acesso em: 20 maio 2010.