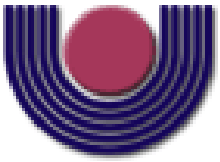


UNIOESTE – Universidade Estadual do Oeste do Paraná

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

Colegiado de Ciência da Computação

Curso de Bacharelado em Ciência da Computação

Inclusão de recursos de acessibilidade no Firefox para a plataforma multissensorial PlatMult

Mauricio Begnini

CASCAVEL

2012

MAURICIO BEGNINI

INCLUSÃO DE RECURSOS DE ACESSIBILIDADE NO FIREFOX PARA A PLATAFORMA MULTISSENSORIAL PLATMULT

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Dr. Marcio Seiji Oyamada

CASCAVEL

2012

MAURICIO BEGNINI

INCLUSÃO DE RECURSOS DE ACESSIBILIDADE NO FIREFOX PARA A PLATAFORMA MULTISSENSORIAL PLATMULT

Monografia apresentada como requisito parcial para obtenção do Título de *Bacharel em Ciência da Computação*, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Dr. Marcio Seiji Oyamada (Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Dr. Jorge Bidarra
Colegiado de Ciência da Computação,
UNIOESTE

Claudir Galesky Junior
Bolsista DTI do projeto PlatMult,
UNIOESTE

Cascavel, 30 de outubro de 2012.

DEDICATÓRIA

Para Cecilia e Renato Begnini.

AGRADECIMENTOS

Agradeço a minha família que com todo apoio tornaram possível que eu estivesse realizando este trabalho. Ao meu orientador que me convidou para participar do projeto PlatMult. Aos membros da equipe do PlatMult, pelo desenvolvimento pelo dos módulos que utilizei no projeto. Aos professores que me tornaram apto a estar concluindo o curso. Aos meus amigos, por todos os momentos que passamos juntos.

EPÍGRAFE

"Eu não falhei, encontrei 10 mil soluções que não davam certo."

Thomas A. Edison

Lista de Figuras

Lista de Abreviaturas e Siglas

API	Application Programming Interface
AT-SPI	Assistive Technology Service Provider Interface
ATK	Accessibility Toolkit
BV	Baixa Visão
DV	Deficiente Visual
FireDEMO	Firefox Detector de Eventos de Mouse Over
GAP	GNOME Accessibility Project
GNOME	GNU Network Object Model Environment
GNU	GNU is Not Unix
Mouse AFM	Mouse de Acessibilidade por Feedback Motor
OMS	Organização Mundial de Saúde
TA	Tecnologia Assistiva

Sumário

Resumo

Tecnologias assistivas são recursos para fornecer acessibilidade aos usuários, permitindo ou facilitando que uma pessoa com algum tipo de deficiência realize alguma tarefa que sua deficiência a impediria ou atrapalharia. Hoje elas podem ser consideradas como recursos essenciais para a sociedade, já que estamos visando a igualdade de oportunidades a todos as pessoas. Entre as deficiências, a visual é a que atinge a maior parcela da população. Como visão é o sentido encarregado de fornecer a maior parte da percepção do ambiente para as pessoas a deficiência visual causa prejuízos sérios para os que possuem graus elevados.

Visando fornecer acessibilidade para estes usuários, o *feedback* motor é capaz de apresentar um novo campo de percepção para usuários deficientes visuais. O *feedback* motor pode ser fornecido de varias maneiras, como a vibração de um componente, via temperatura, ou mesmo um componente aplicar uma resistência na direção contraria à que o usuário está aplicando força. Se tratando de computadores e terminais de auto atendimento, a visão é responsável por praticamente toda a percepção do usuário deficiente visual, mas é possível que tecnologias assistivas forneçam percepção auditiva e motora.

A tecnologia assistiva PlatMult, é uma plataforma multissensorial para prover acessibilidade na utilização de totens de auto atendimento, desenvolvida na UNIOESTE. É multissensorial, pois não limita o usuário com apenas a percepção visual da tela, o usuário também tem percepção auditiva e táctil. O recurso visual é provido por um ampliador de tela, e um monitor. O recurso auditivo é provido por um leitor de tela, e caixas de som. O recurso táctil é provido pelo *driver* Mouse Feedback através do Mouse AFM (Mouse de Acessibilidade por *Feedback* Motor).

Este trabalho estuda e apresenta os métodos utilizados para que o *feedback* motor seja fornecido ao usuário, tanto na utilização do sistema operacional Linux, quanto na navegação na Internet utilizando o Firefox. Também apresenta o desenvolvimento do módulo para que o leitor de tela forneça o *feedback* auditivo durante a utilização do Firefox.

Palavras-chave: Baixa Visão, Tecnologia Assistiva, *Feedback*.

Capítulo 1

Introdução

De acordo com o CENSO do IBGE divulgado em 2010, 24% da população brasileira, 45,6 milhões de pessoas, possui algum tipo de deficiência motora, mental ou visual (IBGE, 2012). Desta parcela estima-se que 60% dos indivíduos são deficientes visuais (DV), distribuídos em cegos e baixa visão (BV). A OMS (Organização Mundial de Saude) (WHO, 2012) classifica cegos como pessoas que com a visão corrigida o melhor dos seus olhos possui acuidade visual inferior a 20/400, ou seja, se ela pode ver a 20 pés (aproximadamente 6 metros) o que uma pessoa de visão normal pode ver a 400 pés (aproximadamente 122 metros). Pessoas com BV são indivíduos que possuem acuidade visual entre 20/60 e 20/400.

Segundo (Masini, 1994) a visão é um dos principais sentidos do homem, responsável por grande parte da integração das atividades motoras, perceptivas e mentais, mas crianças com DV se bem integrados com seus familiares e o ambiente, podem realizar atividades normais com outras crianças e são bem aceitas no meio, sendo tratadas primeiro como crianças e depois como DV. Ainda segundo (Masini, 1994), a DV não causa problemas a criança se seus familiares e educadores forem bem preparados e tiverem os recursos para auxiliar na educação da criança, mas isso geralmente não ocorre.

Tecnologias assistivas são um recurso poderoso no processo de integração dos deficientes. (Bersch, 2008) define tecnologia assistiva (TA) como qualquer recursos que proporcione ou amplie as habilidades funcionais das pessoas, em especial as com deficiência de qualquer tipo, possibilitando que estas realizem atividades desejadas, e por consequência, dando ao usuário maior independência, qualidade de vida e inclusão social. (Bersch, 2008) introduz o conceito de TA com a citação:

> *"Para as pessoas sem deficiência, a tecnologia torna as coisas mais fáceis.*
>
> *Para as pessoas com deficiência, a tecnologia torna as coisas possíveis."* (Radabaugh, 1993).

Motivado pela importância de tecnologias assistivas para alcançar a inclusão social e digital, na UNIOESTE, o projeto PlatMult vem desenvolvendo uma plataforma multissensorial, chamada também de PlatMult, que proporciona ao usuário três formas de interação: visual, via o ampliador de tela xLupa, auditivo, via o leitor de tela (Balansin, 2011) e táctil via o driver Mouse Feedback. Com isso é possível que o usuário com baixa visão ou cegos, mesmo com sua deficiência, realize tarefas em computadores e terminais de autoatendimento com maior facilidade.

1.1 Objetivo

Este trabalho tem como objetivo geral utilizar os eventos que ocorrem durante a utilização do navegador Firefox para fornecer acessibilidade na plataforma PlatMult. Para alcançar o objetivo geral, os objetivos específicos são:

- Estudar o funcionamento biblioteca de acessibilidade AT-SPI, e sua relação com o fornecimento de *feedback* motor e auditivo.
- Estudar a integração da biblioteca de acessibilidade AT-SPI com o Firefox, e o desenvolvimento do complemento para Firefox FireDEMO, utilizado para captação de eventos de *mouse over* durante a utilização do navegador.
- Estudar o método utilizado para o FireDEMO notificar o Mouse Feedback dos eventos captados no Firefox, e sua eficiência em termos de utilização de CPU.
- Desenvolver um módulo em que os eventos captados pelo FireDEMO sejam utilizados para fornecimento de *feedback* motor e de *feedback* auditivo.

1.2 Organização do texto

O trabalho esta organizado nos seguintes capítulos:

- Capítulo 2. Apresenta o projeto da plataforma PlatMult e as ferramentas utilizadas para fornecer acessibilidade a usuários de baixa visão.
- Capítulo 3. Introduz a biblioteca AT-SPI e sua utilização para prover acessibilidade. Descreve os eventos captados na AT-SPI e sua utilidade no fornecimento de *feedback* motor e auditivo. Apresenta como o *feedback motor e auditivo* são fornecidos.

- Capítulo 4. Apresenta o complemento FireDEMO, seu objetivo e funcionamento, sua integração com o Mouse Feedback, e os métodos de notificação dos eventos captados.
- Capítulo 5. Apresenta a modularização do FireDEMO, para injetar os eventos na AT-SPI, e assim possam ser utilizados por qualquer aplicação que forneça acessibilidade.
- Capítulo 6. Apresenta uma analise dos resultados obtidos, uma discussão sobre trabalhos futuros.

Capítulo 2

PlatMult

Este capítulo tem o objetivo de apresentar a plataforma PlatMult, descrevendo em suas seções o funcionamento das aplicações de acessibilidade utilizadas pela plataforma. As informações utilizadas para descrever as ferramentas são referentes ao artigo "*Development of an interactive kiosk with screen amplifier targeting low vision and old-aged people*" (Bidarra, Oyamada, 2011).

2.1 Introdução

A plataforma PlatMult é um totem de auto atendimento, desenvolvendo soluções tanto em software como em hardware, com propósito de permitir que pessoas com baixa visão possam realizar tarefas que necessitam utilizar computadores.

O PlatMult presta auxílio visual, auditivo, e tátil em um único aparelho, a fim de possibilitar que esses usuários possam utilizar recursos de um computador. Um dos objetivos dessa solução é realizar um projeto de baixo custo, na qual permita o acesso a grande número de pessoas. A fim de alcançar o propósito, um estudo foi realizado sobre os componentes de arquitetura, tanto de hardware e software. A respeito de hardware é feito o uso da plataforma x86. Enquanto para os software é utilizado componentes de softwares livres, Ubuntu 10.10 (Ubuntu, 2012) é o sistema operacional utilizado neste projeto, o gerenciador de janelas é o GNOME e o servidor gráfico Xorg (2012). Para os recursos de acessibilidade, é usada a biblioteca AT-SPI (2012).

2.2 Componentes do PlatMult

A arquitetura do PlatMult é baseada em três interações de servidores: ampliador de tela xLupa, leitor de tela e mouse feedback. Cada um deles é responsável por processamento de sentidos como a visão, audição e tato, respectivamente (Bidarra, Oyamada, 2011). A Figura 2.1 apresenta a arquitetura geral do PlatMult.

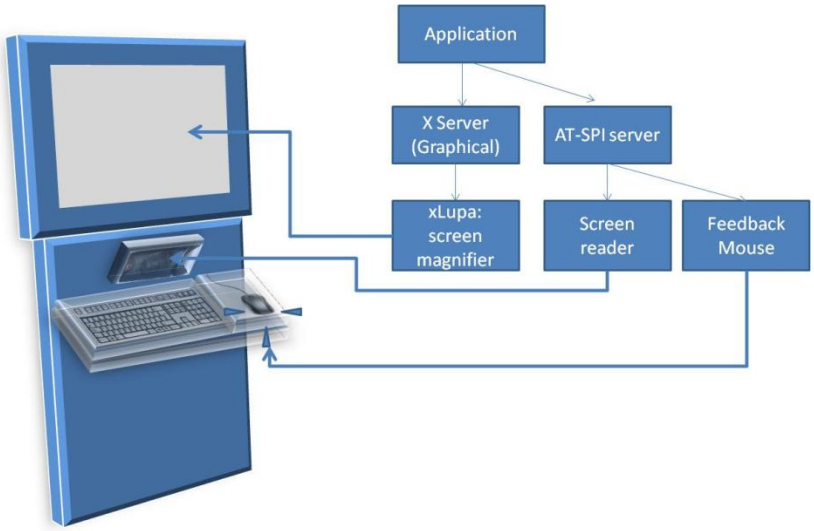


Figura 2.1: Arquitetura geral do PlatMult

2.2.1 Ampliador de Tela

A fim de desenvolver uma solução para facilitar usabilidade de computadores para pessoas com baixa visão, em 2003 foi iniciado, na UNIOESTE, o projeto do ampliador de tela Xlupa (Bidarra, Boscarioli, Rizzi, 2009), um software livre para plataforma Linux. O xLupa oferece recurso de ampliação de textos e imagens, com recursos que permitem aos usuários calibrar a exibição de acordo com suas necessidades específicas. Estes recursos incluem a seleção de cor para o fundo e primeiro plano, a aplicação de algoritmos de computação gráfica para processamento de imagens e linhas de contorno, bem como definição de contraste e brilho. Os fatores de ampliação e cor correspondem ao tamanho do tipo de letra da e cor de fundo que melhor se ajuste as necessidades do usuário.

2.2.2 Leitor de Tela

Para que o usuário do PlatMult tenha auxílio auditivo enquanto utiliza o computador um leitor de tela foi incluído. O leitor de tela utilizado foi desenvolvido como trabalho de conclusão de curso por Balansin (2011) para o curso de ciência da computação na UNIOESTE campus cascavel. As vantagens da utilização deste leitor vem do fato de ter sido desenvolvido na

O leitor fornece ao usuário informação auditiva durante a utilização de aplicativos, fazendo a leitura de textos, leitura do nome de menus, botões, janelas que receberem foco, e auxiliando durante a edição de textos.

2.2.3 Mouse Feedback

As ferramentas de auxilio visual e auditivo são soluções em software que foram incluídas na utilização do PlatMult, enquanto o auxilio táctil vem de um desenvolvimento de *hardware* e *software*, feito diretamente para a plataforma.

A solução em hardware é o Mouse AFM (Mouse de Acessibilidade por *Feedback* Motor) um dispositivo de *feedback* motor de baixo custo montado pela equipe do PlatMult. Para montá-lo foi utilizado um *mouse* normal ao qual foi adicionado um motor vibratório similar aos utilizados para que celulares vibrem. A alimentação do motor vibratório é feita via porta paralela no computador. Um protótipo do Mouse AFM pode ser visto na figura 2.2 abaixo. O motor vibratório é indicado na figura.

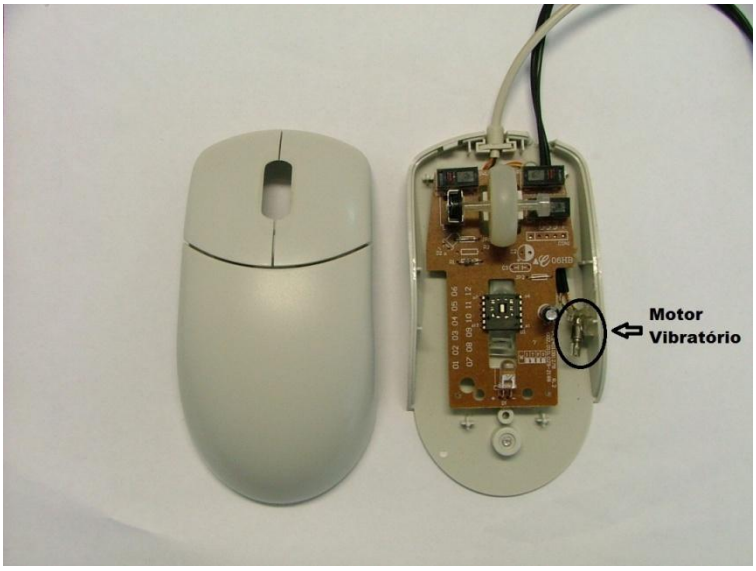


Figura 2.2: Mouse AFM

A solução em software consiste na implementação de um *driver* chamado de Mouse Feedback, que é responsável pela ativação do recurso vibratório do Mouse AFM. O objetivo do auxilio táctil é ajudar o usuário na localização de campos de interesse, como menus, botões e ícones na interface do sistema, fornecendo um *feedback* motor via vibração do dispositivo, quando o ponteiro do *mouse* passar por algum destes campos de interesse.

Capítulo 3

Fornecimento de Feedback motor e auditivo

No capítulo anterior foi apresentada a plataforma de tecnologia assistiva PlatMult e a participação de seus componentes para prover acessibilidade aos usuários. O objetivo deste trabalho é incluir acessibilidade nos componentes Mouse Feedback e Leitor de Tela. Neste capítulo são discutidos os aspectos técnicos relacionados ao fornecimento do *feedback* por esses dois componentes.

Para tanto introduz-se na seção 3.1 do capítulo a biblioteca de acessibilidade AT-SPI e como ela se relaciona com os aplicativos de sistema e com as tecnologias assistivas. Na seção seguinte, é abordado o estudo dos eventos que ocorrem durante a utilização do sistema e a descrição dos eventos que ativam os recursos de *feedback*. Na última seção, são apresentados os passos feitos pelo Mouse Feedback e pelo Leitor para ativação do *feedback*.

3.1 AT-SPI

A AT-SPI é uma API de acessibilidade desenvolvida pela GAP (GNOME *Accessibility Project*) para que os desenvolvedores pudessem prover recursos de acessibilidades em suas aplicações, tornando o GNOME um ambiente acessível para pessoas com deficiências (Frauenberger, Höldrich, 2004). Programas compatíveis com a AT-SPI fornecem informações de sua interface com o usuário para a API. No PlatMult o Leitor de Tela e o Mouse Feedback utilizam a AT-SPI para fornecer acessibilidade, e por isso o funcionamento da biblioteca será explicado.

A figura 3.1 abaixo exibe o esquema global da AT-SPI (adaptada de Lee, 2008) junto com vários recursos. A AT-SPI é um meio de ligação entre os aplicativos do sistema e os aplicativos de acessibilidade (Balansin, 2011).

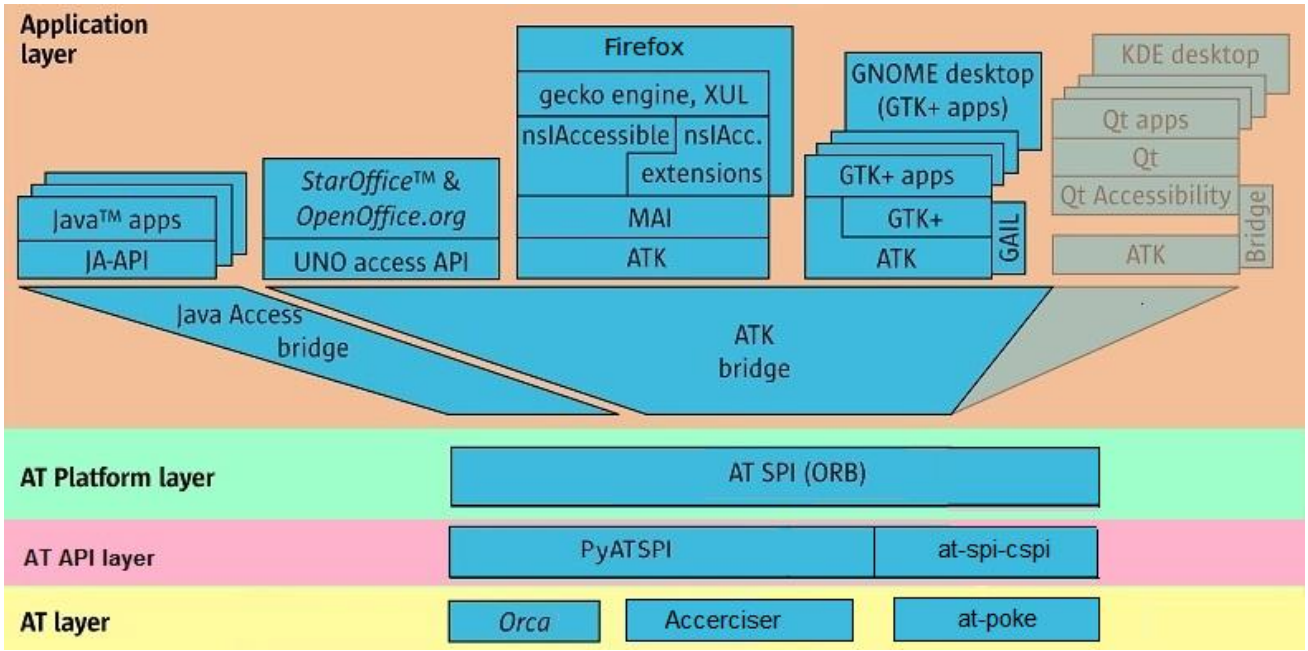


Figura 3.1: Esquema Global do AT-SPI

A figura 3.1 é composta por quatro camadas. Na primeira camada, "*Application layer*", estão os aplicativos do sistema com os quais o usuário interage diretamente, como aplicações Java, OpenOffice, Firefox e o GNOME desktop. Dependendo das aplicações, informações sobre sua interface com o usuário são disponibilizadas para a AT-SPI. Essas informações fornecidas por uma aplicação para a AT-SPI são utilizadas por desenvolvedores de tecnologias assistivas para que durante a utilização desta aplicação a tecnologia assistiva forneça recursos de acessibilidade ao usuário.

Para que um desenvolvedor possa fazer com que a sua aplicação forneça informações de acessibilidade para a AT-SPI, ele pode usar ferramentas como o ATK (*Accessibility Toolkit*), um pacote de desenvolvimento do GNOME que fornece aos programadores a compatibilidade entre as informações da interface da sua aplicação e a AT-SPI e outros recursos de acessibilidade do GNOME. Como pode ser visto na figura 3.1, outras interfaces além do ATK como "JA-API", "UNO access API", "nsIAccessible" e "GAIL" implementam as interfaces de acessibilidade para aplicações.

Na segunda camada, "*AT Platform layer*", é onde se encontra a AT-SPI. Cabe a AT-SPI gerenciar as informações que recebe das aplicações sobre a interface com os usuários e repassa elas para aplicativos de acessibilidades registrados.

Na terceira camada, "*AT API layer*", estão as bibliotecas utilizadas para registro na AT-SPI. Essa camada é utilizada pelos aplicativos fornecedores de acessibilidade, da quarta camada, a "*AT layer*". Aplicações fornecedoras de acessibilidade se registram à AT-SPI usando uma biblioteca, como no caso do Accerciser, que utiliza a PyATSPI, e passa a receber informações da interface das aplicações. Em posse das informações da interface das aplicações, cabe ao desenvolvedor utilizá-las para prover acessibilidade.

Cabe mencionar que não é papel da AT-SPI fornecer os eventos diretamente aos aplicativos de acessibilidade. É responsabilidade do desenvolvedor acessar a AT-SPI para consultar, filtrar e recuperar os eventos captados por ela, e então, utilizar destes eventos no fornecimento de acessibilidade por suas aplicações. Na Figura 3.1, podemos verificar que as bibliotecas PyATSPY e at-spi-cspi estão entre a AT-SPI e os fornecedores de acessibilidade, justamente porque elas estão realizando o processo de seleção das informações, para que então, forneçam os eventos aos aplicativos de acessibilidade ligados a elas.

3.2 Identificação dos Eventos

Para identificar os eventos de acessibilidade utilizamos o Accerciser, um explorador acessibilidade interativo feito em Python para a área de trabalho GNOME . Ele utiliza a AT-SPI para verificar se um aplicativo está fornecendo a informações de sua interface com o usuário (Project 2012).

O Accerciser foi utilizado no estudo da AT-SPI, especificamente a função "Monitor de Eventos", que permite selecionar os tipos de eventos que precisam ser notificados, e utilizar o sistema normalmente, enquanto a tela apresenta os eventos registrados na API durante a execução do seu programa. Na figura 3.2 abaixo é possível ver a função "Monitor de Eventos" do Accerciser em execução, junto com a aplicação Firefox. A Marcação 1 mostra o aplicativo que está sendo monitorado, no caso o Firefox, na marcação 2 os tipos de eventos que estão sendo monitorados, no caso os do tipo *focus* (os outros eventos disponíveis são: *document*, *mouse*, *object*, *terminal* e *window*), e na marcação 3 uma lista com a descrição dos eventos que foram captados.

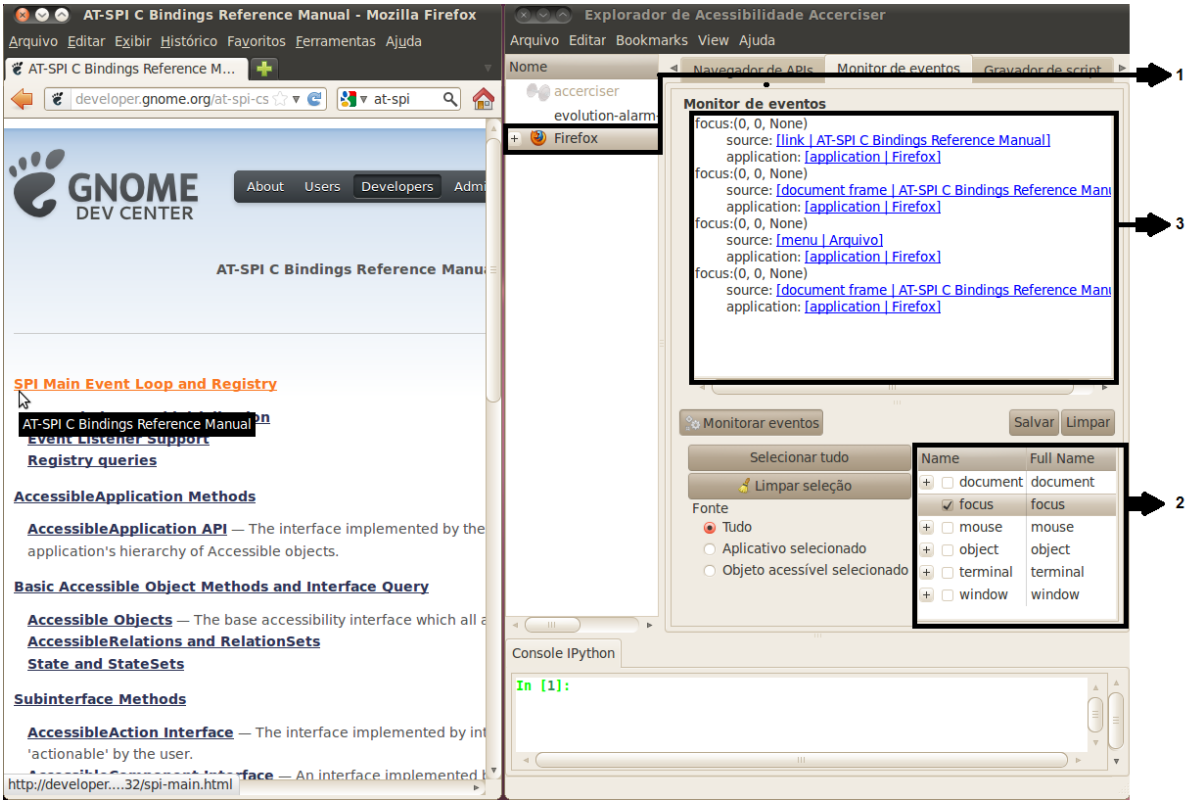


Figura 3.2: Accerciser captando eventos de “Focus” no Firefox

Utilizando a função “Monitor de Eventos” do Accerciser foi possível identificar os eventos utilizados no fornecimento de *feedback* motor e auditivo. Para o fornecimento de *feedback* motor, os eventos de *Focus* são utilizados, enquanto que para o fornecimento do *feedback* auditivo são utilizados os eventos de *Focus*, *Object:text-changed:insert*, *Object:text-changed:delete*, *Object:text-caret-moved*, *Window:activate* e *Window:create*. Abaixo, uma breve descrição dos tipos de eventos utilizados para o fornecimento de *feedback*, e qual é a sua contribuição no fornecimento do *feedback*.

Focus: Evento responsável por monitorar todos os elementos gráficos da tela que podem receber o foco. Em uma utilização normal do sistema, este evento é o que mais será gerado, pois itens como menus, botões, ícones e abas que receberem foco gerará tal evento. Os eventos são utilizados para ativar o *feedback* motor, e sua utilização se deve ao fato do ponteiro do *mouse* trazer foco aos itens pelos quais passa. Os eventos são utilizados para ativar o *feedback* auditivo porque quando um item recebe foco é possível obter o texto contido nele, e esse texto pode ser lido.

Object:text-changed: Esse evento ocorre na edição de texto. Diferente dos eventos de Focus, em que qualquer evento pertencente a classe é utilizada, neste caso a classe de eventos

Object tem os eventos do tipo *text-changed* filtrados para serem usados. Dentro dos eventos *text-changed* são utilizados eventos ocorridos em dois casos: *insert*, onde é monitorada a inserção de texto via digitação, utilização do comando colar, ou auto completar dos aplicativos; *delete* para quando ocorre a exclusão do texto. Os evento são utilizados para ativar o *feedback* auditivo, para que seja lida a ação que está acontecendo com o texto.

Object:text-caret-moved: Similar ao evento anterior, também os eventos do tipo "text-caret-moved" também pertencem a classe de eventos "Object" e também são utilizados durante a edição de texto. O evento ocorre quando o cursor do texto é movimentado, diferente do anterior não é ligado a edição do texto em si, mas acontece quando o usuário movimenta o cursor de digitação sobre o texto (usando as setas, por exemplo). Os evento são utilizados para ativar o *feedback* auditivo, para informar o usuário, através da leitura, a posição do cursor no texto, quando este sofre uma movimentação.

Window:activate e **Window:create:** Eventos *activate* e *create* da classe de eventos *Window*. Estes dois eventos ocorrem durante a manipulação de janelas, sendo semelhantes entre si, porém são gerados em momentos diferentes. O *Window:activate* é acontece quando em que uma janela é ativada, ou seja, no momento em que ela é colocada em destaque estando escondida ou minimizada anteriormente. Já o evento *Window:create* acontece apenas no momento em que uma janela é criada na tela. Os evento são utilizados para ativar o *feedback* auditivo, para que seja lido o nome da janela em que a ação está acontecendo.

3.3 Fornecimento de Feedback

Nas seções acima foi explicado como tecnologias assistivas podem utilizar a AT-SPI para fornecer acessibilidade, e foi feito um levantamento dos eventos utilizados para o *feedback* motor e auditivo. Nos sub tópicos desta seção temos a apresentação individual do esquema para fornecimento do *feedback* motor via Mouse Feedback, e do *feedback* auditivo via o Leitor (Balansin, 2011).

3.3.1 Feedback Motor

O Mouse Feedback é o driver responsável pela ativação do recurso de *feedback* motor no Mouse AFM. O diagrama de sequencia apresentado na figura 3.3 pode ser visto como uma forma simplificada da figura 3.1 para apresentar o funcionamento do driver Mouse Feedback.

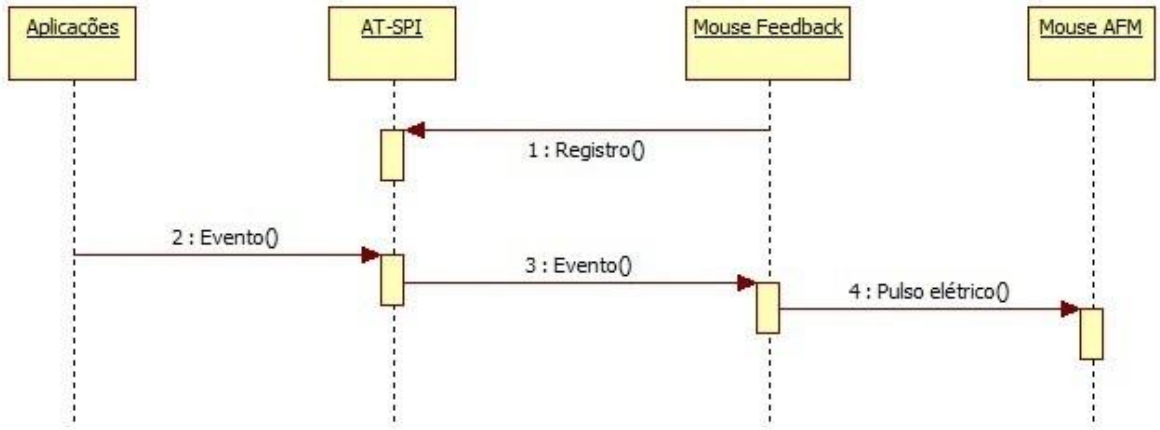


Figura 3.3: Diagrama de sequencia para fornecimento de *feedback* motor

Os passos do diagrama de sequencia apresentado na figura 3.3 são descritos abaixo:

1. Registro: Quando entra em execução, o Mouse Feedback faz o registro na AT-SPI para ser notificado quando ocorrer eventos de *Focus*.

2. Evento: Quando um evento ocorre em uma aplicação, o evento é reportado para a AT-SPI.

3. Evento: Quando a AT-SPI recebe a informação de um evento ocorrido em um aplicativo o Mouse Feedback é notificado do evento.

4. Pulso elétrico: Quando o Mouse Feedback é notificado sobre a ocorrência de um evento, este envia um pulso elétrico via porta paralela, para que o motor vibratório do Mouse AFM seja ativado.

O registro feito pelo Mouse Feedback na AT-SPI utiliza a biblioteca at-spi-cspi, mas esse passo ficou ocultado no diagrama, porque acontece internamente no código do driver. A biblioteca utilizada é a at-spi-cspi porque o driver foi desenvolvido em linguagem C.

O diagrama de sequencia apresentado na figura 3.3 serve apenas para apresentar o comportamento do Mouse Feedback, pois ignora qualquer comportamento dos dispositivos que não sejam utilizados pelo driver. Nesse esquema são considerados apenas eventos do tipo

focus, pois são os eventos para os quais o Mouse Feedback se registra para receber, e são considerados apenas os eventos que ocorrem depois do registro, pois eventos anteriores a este passo não tem nenhuma influencia no fornecimento de *feedback* motor.

Como resultado, depois do registro do driver na AT-SPI, sempre que um evento do tipo *Focus* ocorrer em uma aplicação, o Mouse AFM vai ter seu recurso de *feedback* motor ativado.

3.3.2 Feedback auditivo

O leitor (Balansin, 2011) é o *software* responsável pela ativação do recurso de *feedback* auditivo. O diagrama de sequencia apresentado na figura 3.4 pode ser visto como uma forma simplificada da figura 3.1 para apresentar o funcionamento do leitor (Balansin, 2011).

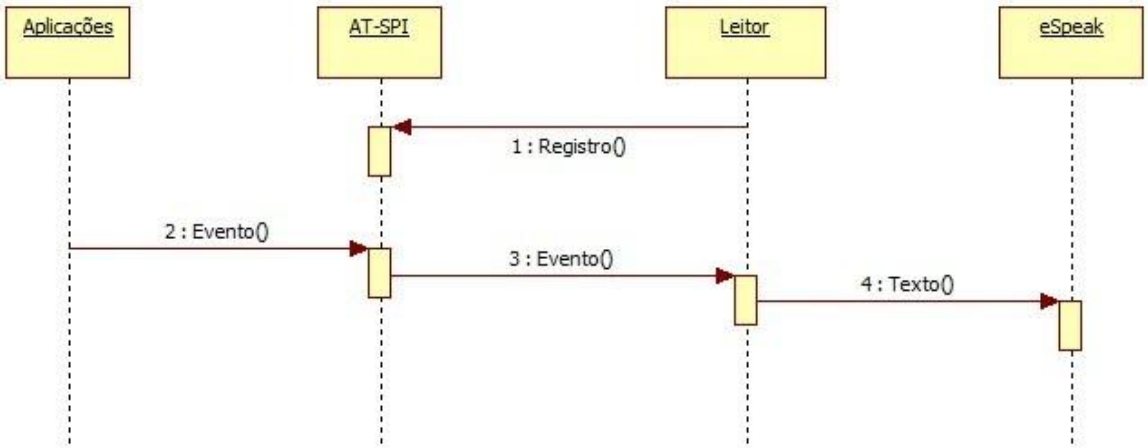


Figura 3.4: Diagrama de sequencia para fornecimento de *feedback* auditivo

Os passos do diagrama de sequencia apresentado na figura 3.4 são descritos abaixo:

1. Registro: Quando entra em execução, o leitor faz o registro na AT-SPI para ser notificado quando ocorrer eventos de *Focus*, *Object:text-changed:insert*, *Object:text-changed:delete*, *Object:text-caret-moved*, *Window:activate* e *Window:create*.

2. Evento: Quando um evento ocorre em uma aplicação, o evento é reportado para a AT-SPI.

3. Evento: Quando a AT-SPI recebe a informação de um evento ocorrido em um aplicativo o leitor é notificado do evento.

4. Texto: Quando leitor é notificado sobre a ocorrência de um evento, este envia um texto referente ao evento para o sintetizador de voz eSpeak, para que este leia o texto para o usuário.

O registro feito pelo leitor na AT-SPI utiliza a biblioteca PyATSPI, mas esse passo ficou ocultado no diagrama, porque acontece internamente no código do driver. A biblioteca utilizada é a PyATSPI porque o driver foi desenvolvido em linguagem Python.

O diagrama de sequencia apresentado na figura 3.4 serve apenas para apresentar o comportamento do leitor de tela (Balansin, 2011), pois ignora qualquer comportamento dos dispositivos que não sejam utilizados pelo leitor. Nesse esquema são considerados apenas eventos dos tipos *Focus*, *Object:text-changed:insert*, *Object:text-changed:delete*, *Object:text-caret-moved*, *Window:activate* e *Window:create* pois são os eventos para os quais o leitor se registra para receber, e são considerados apenas os eventos que ocorrem depois do registro, pois eventos anteriores a este passo não tem nenhuma influencia no fornecimento de *feedback* auditivo.

Como resultado, depois do registro do leitor na AT-SPI, sempre que um evento dos tipos *Focus*, *Object:text-changed:insert*, *Object:text-changed:delete*, *Object:text-caret-moved*, *Window:activate* e *Window:create* ocorrer em uma aplicação, o eSpeak vai ler, trazendo ao usuário o *feedback* auditivo.

Capítulo 4

Firefox no PlatMult

No capítulo anterior foi visto como eventos que ocorrem enquanto o usuário interage com interfaces de aplicações são repassados para ele em forma de *feedback* motor e auditivo.

Este capítulo mostra o estudo da integração de um aplicativo em particular, o navegador Firefox, com o PlatMult. Inicialmente é discutida a falta de integração do navegador com a AT-SPI. Na sequencia é apresentado o FireDEMO (Firefox Detector de Eventos de Mouse Over), um complemento para Firefox, desenvolvido para captação de eventos. No final do capítulo dois métodos de comunicação entre o FireDEMO e o Mouse Feedback são apresentados e comparados.

4.1 Integração do Firefox com a AT-SPI

A AT-SPI é utilizada para captação de eventos que ocorrem em um aplicativo, para assim fornecer o *feedback* ao usuário, então para estudar a integração do Firefox com o PlatMult é necessário estudar a integração do navegador com a AT-SPI.

Utilizando o “Monitor de Eventos” do Accerciser, além do levantamento dos tipos de eventos, também foi possível descobrir quais as informações de sua interface uma determinada aplicação informa para a AT-SPI. Com isso, foi feito o estudo dos eventos que o navegador Firefox fornece para à API.

Com o estudo foi percebido que eventos do tipo *Focus* ocorrem apenas quando o foco é sobre menus do navegador, mas nada é notificado quando o ponteiro do *mouse* passa sobre o conteúdo da página *web*, como como *links*, botões e áreas de texto.

Essa insuficiência de notificação de eventos durante a utilização do Firefox é um problema que precisou ser resolvido, já que apenas com os eventos notificados pela API para o Mouse

Feedback não é possível auxiliar o usuário na localização de conteúdo enquanto navega na *internet,* porque páginas *web* possuem muitos campos que recebem foco do ponteiro do *mouse*.

Para localizar o usuário durante a utilização do Firefox, foram definidos campos de interesse, que quando o ponteiro do *mouse* passar sobre, o Mouse Feedback deve ser notificado de um evento e ativar o *feedback* motor do Mouse AFM. Os campos de interesse são *links*, botões, campos de entrada e áreas de texto.

4.2 FireDEMO

Para solucionar o problema da insuficiência de eventos durante a utilização do Firefox, um módulo para a captação de eventos de *mouse over* foi desenvolvido em forma de um complemento para o navegador Firefox. Este complemento se chama FireDEMO (Firefox Detector de Eventos de *Mouse Over*), e foi desenvolvido em linguagem de programação JavaScript.

Para gerar os eventos de *mouse over* sobre os campos de interesse o FireDEMO é ativado quando o Firefox é aberto, e monitora a execução do navegador. Quando uma pagina é carregada o complemento faz uma busca dos itens que são considerados campos de interesse, e coloca um monitor (*Listener*) de eventos de *mouse over* em cada um deles. A figura 4.1 apresenta um diagrama de sequencia do funcionamento do FireDEMO.

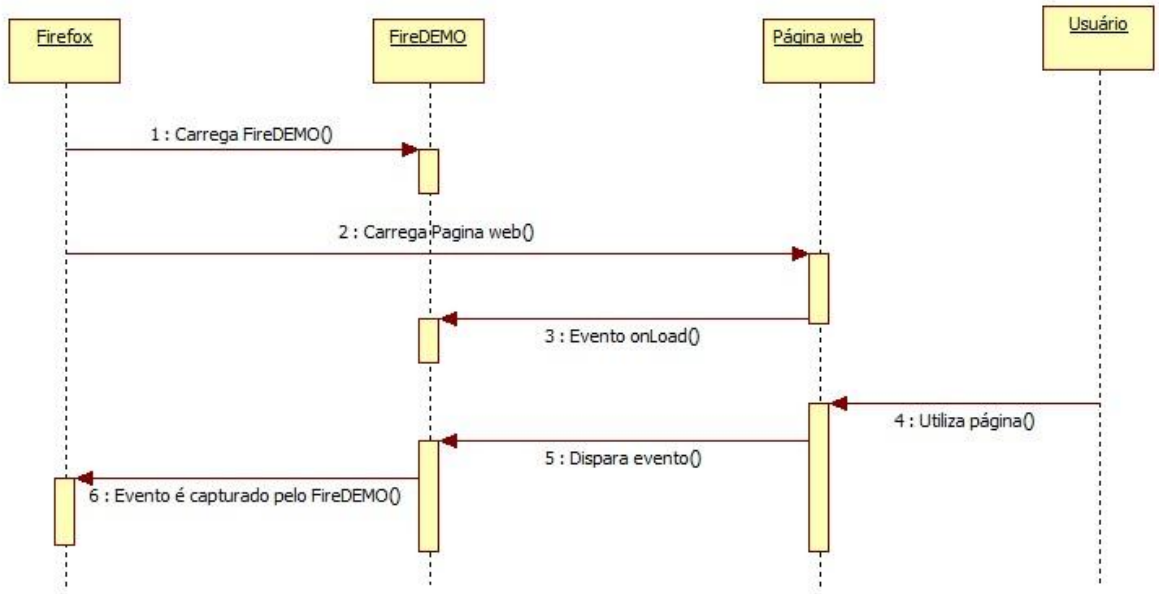


Figura 4.1: Diagrama de sequencia para funcionamento do FireDEMO

Os passos do diagrama de sequencia apresentado na figura 4.1 são descritos abaixo:

1. Carrega FireDEMO: Quando o navegador é iniciado ele carrega o complemento FireDEMO.

2. Carrega Pagina web: O navegador carrega uma nova pagina web.

3. Evento onLoad: Logo após a pagina web ser carregada, o evento onLoad é ativado pelo FireDEMO, para que ele adicione monitores de eventos (*Event Listeners*) aos campos de interesse da pagina.

4. Utiliza página: O usuário navega na página web.

5. Dispara evento: Quando o ponteiro do mouse passa sobre um campo de interesse monitorado, um evento é disparado.

6. Evento é capturado pelo FireDEMO: O complemento obtem as informações sobre o evento no campo.

O diagrama de sequencia apresentado na figura 4.1 apresenta os passos da inicialização do FireDEMO, até o momento em que um evento é capturado. Este diagrama não apresenta o destino dos eventos, que será discutido na próxima seção.

4.3 Notificação do evento

O FireDEMO monitora o Firefox para saber quando um evento do tipo *mouse over* ocorre sobre um campo de interesse, mas para o recurso de *feedback* do Mouse AFM ser ativado ainda é necessário que o driver Mouse Feedback seja notificado deste evento. Para fazer o envio do evento do FireDEMO para o Mouse Feedback fez-se necessário o desenvolvimento de um módulo de comunicação. Como o Mouse Feedback precisa continuar atento aos eventos que são notificados pela AT-SPI, uma *thread* é criada no Mouse Feedback, e essa *thread* executa o módulo responsável por receber as notificações de eventos captados pelo FireDEMO.

Foram desenvolvidos dois métodos, um utilizando arquivo, e outro utilizando envio de mensagem HTTP. Os métodos são apresentados nas subseções seguintes, e depois comparados.

4.3.1 Notificação via arquivo

A primeira solução desenvolvida para que o FireDEMO reporte os eventos ao Mouse Feedback foi a utilização de um arquivo. Sempre que um evento é detectado o complemento escreve as informações sobre este evento em um arquivo. O Mouse Feedback, não sabe quando um evento ocorreu, nem o momento em que o FireDEMO escreveu no arquivo, então ele faz acessos constantes a este arquivo em busca de uma modificação no conteúdo dele desde o ultimo acesso. Se quando acessado o arquivo, o conteúdo for diferente do encontrado no ultimo acesso, isso significa que o FireDEMO detectou um novo evento e escreveu no arquivo. Quando isso ocorre, o driver chama a função para ativação do recurso *feedback* do Mouse AFM. A figura 4.2 apresenta um diagrama de sequencia para a notificação de evento entre o FireDEMO e o Mouse Feedback via arquivo.

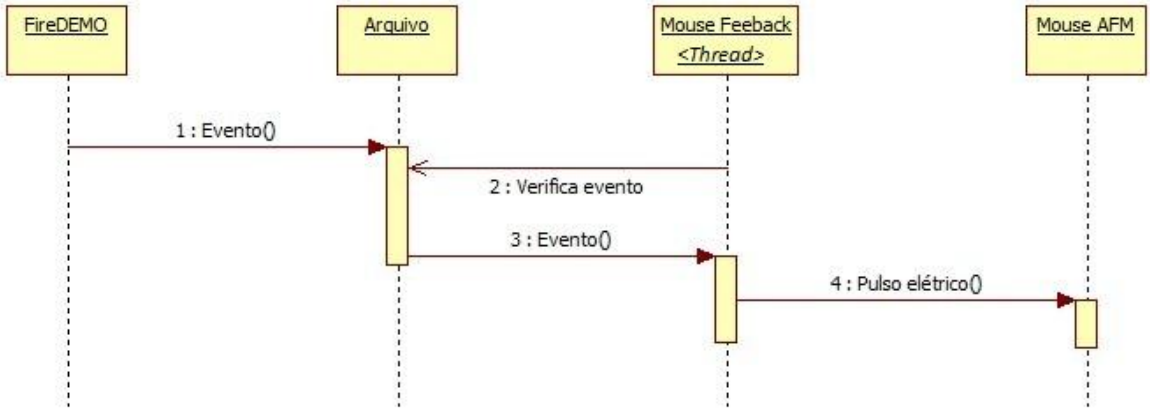


Figura 4.2: Diagrama de sequencia para notificação de evento via arquivo

Os passos do diagrama de sequencia apresentado na figura 4.2 são descritos abaixo:

1. Evento: O FireDEMO capta um evento, então escreve suas informações no arquivo.

2. Verifica evento: O Mouse Feedback (a *thread* do Mouse Feedback responsavel pela comunicação) acessa o arquivo em busca de conteudo novo (faz um acesso ao arquivo a cada 500 milissegundos).

3. Evento: O conteudo do novo evento é recuperado pelo Mouse Feedback.

4. Pulso elétrico: Quando o Mouse Feedback é notificado sobre a ocorrência de um evento, este envia um pulso elétrico via porta paralela, para que o motor vibratório do Mouse AFM seja ativado.

O diagrama de sequencia apresentado na figura 4.2 serve apenas para apresentar o comunicação entre o FireDEMO e o Mouse Feedback via arquivo, não apresentando a relação do Mouse Feedback com os eventos recebidos da AT-SPI. As notificações de eventos da AT-SPI não sofrem nenhuma alteração.

Como resultado, sempre que o FireDEMO captar um evento de *mouse over* sobre um campo de interesse em uma pagina web, o Mouse AFM vai ter seu recurso de *feedback* motor ativado.

4.3.2 Notificação de evento via mensagem HTTP

Na arquitetura apresentada na seção anterior, para ser notificado de um evento de *mouse over* a *thread* do Mouse Feedback precisa fazer acessos constantes ao arquivo, mesmo que

nenhum evento tenha ocorrido, tendo assim muitos acessos desnecessários. Para evitar os acessos desnecessários, e assim tentar diminuir a utilização do processador durante a utilização do driver, foi desenvolvido novo método de comunicação, com envio de mensagens via HTTP, eliminando assim a necessidade do arquivo.

Para enviar a mensagem diretamente, o FireDEMO utilizou a biblioteca jQuery (Foundation 2012). Utilizando a função "post" da jQuery, é possível enviar uma mensagem para um endereço de *internet* qualquer. Como o destino da mensagem, o Mouse Feedback, executa no mesmo computador em que o FireDEMO o endereço de entrega é o local, em uma porta que em que o driver esteja esperando. A porta 5444 foi escolhida por não ter nenhuma utilização no computador.

No driver, para receber a mensagem enviada pelo complemento, a solução foi a utilização de *Socket*. *Socket* é um ponto final de fluxo de comunicação entre dois aplicativos através da rede. Na figura 4.3 temos um diagrama de sequencia para notificação de evento via mensagem HTTP.

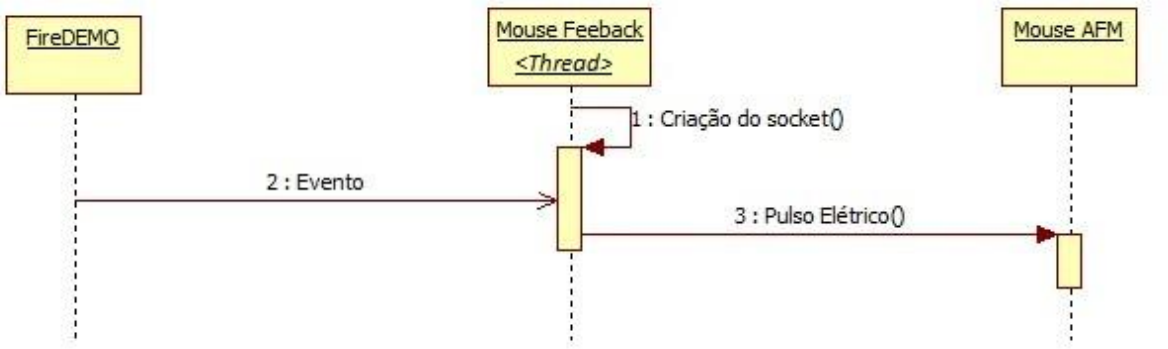


Figura 4.3: Diagrama de sequencia para notificação de evento via mensagem HTTP

Os passos do diagrama de sequencia apresentado na figura 4.3 são descritos abaixo:

1. Criação do socket: O Mouse Feedback cria um socket para ouvir mensagens na porta 5444.

2. Evento: O FireDEMO envia uma mensagem HTTP para o endereço local na porta 5444.

3. Pulso Elétrico: Quando o Mouse Feedback é notificado sobre a ocorrência de um evento, este envia um pulso elétrico via porta paralela, para que o motor vibratório do Mouse AFM seja ativado.

O diagrama de sequencia apresentado na figura 4.3 serve apenas para apresentar a comunicação entre o FireDEMO e o Mouse Feedback via mensagem HTTP, não apresentando a relação do Mouse Feedback com os eventos recebidos da AT-SPI. As notificações de eventos da AT-SPI não sofrem nenhuma alteração.

Como resultado, sempre que o FireDEMO captar um evento de *mouse over* sobre um campo de interesse em uma pagina web, o Mouse AFM vai ter seu recurso de *feedback* motor ativado.

4.3.3 Comparação entre os Métodos

A diminuição do uso de CPU durante a utilização do PlatMult é interessante para reduzir os requisitos de hardware da plataforma e consequentemente seu custo. O método de notificação de eventos via arquivo, utilizado inicialmente, exige constante uso do processador para verificar o conteúdo do arquivo. O desenvolvimento de um novo método de notificação de eventos via HTTP teve como objetivo eliminar a necessidade do arquivo, esperando assim diminuir a utilização de processador.

Após o desenvolvimento do método de notificação de evento via mensagem HTTP, foram feitos testes comparativos entre a segunda versão do Mouse Feedback e FireDEMO, que utiliza o envio de mensagens HTTP, e a primeira versão, que fazia a notificação dos eventos utilizando arquivo.

Os testes foram feitos em um protótipo do terminal de autoatendimento, com as seguintes configurações: processador Pentium 4 2.8ghz; Memória 1gb RAM, HD 80gb; sistema operacional Ubuntu 10.10. As medições de processamento foram feitas utilizando o comando “top” em terminal, para acompanhar as informações sobre o sistema em tempo real.

As comparações entre os métodos foram feitas executando o ampliador de tela xLupa, o driver Mouse Feedback e o navegador Firefox, com o complemento FireDEMO, simultaneamente.

Nos testes utilizando a primeira versão do FireDEMO e do Mouse Feedback, o uso de CPU é de aproximadamente 80% enquanto eventos não estão sendo gerados, e quando eventos ocorrem a utilização de CPU sobe para aproximadamente 85%. Desta porcentagem o xLupa é responsável por aproximadamente 65% do uso da CPU, independente dos eventos estarem sendo gerados ou não.

Nos testes utilizando a segunda versão do FireDEMO e do Mouse Feedback, era esperado que o uso de CPU fosse menor que utilizando a versão anterior, mas o alto custo causado para abrir e manter um socket aberto fez com que o uso de CPU fosse o mesmo que utilizando a versão anterior.

Além de não reduzir a utilização de CPU, como era esperado, a notificação de eventos via mensagem HTTP gerou um problema no fornecimento de *feedback* que não ocorria quando era utilizada a notificação de eventos via arquivo. Quando vários eventos são gerados em sequencia, com um intervalo de tempo muito baixo, o FireDEMO envia uma mensagem para cada um dos eventos, e estas mensagens entram em fila para aceitação do Mouse Feedback. O Mouse Feedback recebe uma mensagem por vez, e gera o *feedback* para cada uma das mensagens recebidas. Como existe um enfileiramento das mensagens recebidas, haverá um atraso no *feedback* do Mouse AFM em relação ao evento correspondente.

Esse problema não ocorre no método em que a notificação de evento é feita via Arquivo, pois não existe uma fila de eventos. Se vários eventos ocorrem em sequencia, com intervalo de tempo muito baixo, O FireDEMO vai escrever cada um deles no arquivo, mas a informação de um evento mais recente sobrescreve um escrito anteriormente. Sempre que o Mouse Feedback verificar o arquivo ele encontrará informações apenas sobre um evento, independente de quantos ocorreram desde a ultima verificação. Então o Mouse Feedback ativa o recurso do Mouse AFM apenas uma vez, impedindo assim o *feedback* atrasado.

O método que faz a notificação de eventos via mensagem HTTP poderia ser modificado para solucionar o problema de atraso no fornecimento do *feedback*, e tentar diminuir o uso de CPU. Para solucionar o problema de atraso no fornecimento do *feedback* poderia ser desenvolvido um método que validasse o evento antes de ativar o *feedback* do Mouse AFM, verificando o intervalo de tempo entre sua captação pelo FireDEMO e seu processamento pelo Mouse Feedback. O evento seria descartado se esse intervalo de tempo fosse considerado elevado.

Para tentar diminuir o uso de CPU, o FireDEMO, quando inicializado, poderia abrir uma única conexão com o Mouse Feedback, e manter esta conexão aberta, notificando os eventos via esta conexão, ao invés de criar uma nova conexão com o Mouse Feedback para cada evento a ser notificado.

Entretanto a implementação dessas soluções exigiria uma quantia de tempo maior que a disponível para finalizar o trabalho, e não existiria a garantia de que a utilização de CPU

realmente seria diminuída, então foi optado pela utilização do método que faz notificação de eventos via arquivo, que já estava implantado e funcionando de forma adequada.

Capítulo 5

Injeção de eventos na AT-SPI

O capítulo anterior se focou no FireDEMO, um complemento para o navegador Firefox, que capta eventos de *mouse over* sobre campos de interesse em uma página web, e seu relacionamento com o Mouse Feedback. Entretanto, o objetivo deste trabalho é a inclusão de recursos de acessibilidade durante a utilização do Firefox tanto para o Mouse Feedback quanto para o Leitor de Tela.

Este capítulo apresenta as alterações feitas, para tornar o FireDEMO um módulo separado do Mouse Feedback, capaz de fornecer eventos para qualquer tecnologia assistiva que se registre na AT-SPI. Na primeira seção do capítulo a modularização do FireDEMO é descrita. Na seção seguinte, o a injeção de eventos na AT-SPI é discutida. A terceira seção apresenta o método de notificações dos eventos a serem injetados. A seção final apresenta o modelo dos eventos, para que o texto que identifique o componente que gerou o evento seja lido.

5.1 Modularização do FireDEMO

O FireDEMO foi desenvolvido com proposito de permitir que um usuário se localizasse em uma página do Firefox utilizando o Mouse AFM, então, quando um evento é captado, até o momento, apenas o Mouse Feedback é notificado. No entanto, estes eventos podem vir a ser uteis para outras aplicações provedoras de acessibilidade.

Componente do PlatMult, o leitor de tela (Balansin, 2011) também utiliza eventos da AT-SPI para ler as informações da interface de aplicativos, e como o Mouse Feedback enquanto não possuía o auxilio do FireDEMO, não é notificado de eventos de *mouse over* sobre campos de interesse da tela, sendo incapaz, por exemplo, de ler o nome de um *link* em uma pagina do web no Firefox quando o ponteiro do *mouse* está sobre ele.

Fazer com que as notificações do FireDEMO estejam disponíveis também para o Leitor de Tela é um objetivo deste trabalho, mas a abordagem atual, em que o complemento faz a notificação diretamente para o provedor de acessibilidade não é viável se quando queremos tornar os eventos disponíveis para mais componentes provedores de acessibilidade. Essa método não é viável pois exige a implementação de um módulo receptor de notificações dos eventos em cada um dos componentes provedores de acessibilidade, criando uma dependência de compatibilidade de versões entre o componente e o FireDEMO, como ocorre com o Mouse Feedback.

A criação do FireDEMO decorreu da AT-SPI não receber eventos do *mouse over* sobre paginas web no Firefox, mas se esses eventos forem fornecidos para a biblioteca, não existe a necessidade de uma conexão direta entre provedores de acessibilidade e o FireDEMO.

A solução para disponibilizar os eventos captados pelo FireDEMO para o Leitor de Tela, Mouse Feedback, e qualquer tecnologia assistiva que tenha interesse, foi separar ele do Mouse Feedback, tornando um módulo separado, e fornecer os eventos captados diretamente para a AT-SPI.

Com a modularização do FireDEMO, o Mouse Feedback não recebe mais notificações diretas do complemento, ficando atendendo apenas a eventos da AT-SPI, removendo a necessidade da existência de uma *thread* para receber as notificações de eventos seja via arquivo ou via mensagem HTTP. O Mouse Feedback mantem apenas a estrutura apresentada na Figura 3.3.

5.2 Injeção de eventos na AT-SPI

Descobrir um modo de injetar eventos na AT-SPI foi o maior desafio encontrado neste trabalho, e o que consumiu mais tempo. Essa dificuldade ocorreu pela pouca documentação existente sobre a AT-SPI, e durante a pesquisa, nenhum trabalho relacionado a injeção de eventos foi encontrado.

Para encontrar um meio de injetar os eventos na AT-SPI foi necessário estudar os códigos da biblioteca. Na parte de testes, existe um código, chamado de *stress-test.c*, que realiza teste de estresse na AT-SPI, criando vários eventos, sem conteúdo, e verificando o tempo levado para que a biblioteca fosse notificada dos eventos.

O estudo dos códigos deixou claro que não é viável fazer a injeção de eventos na AT-SPI diretamente do FireDEMO, pois a criação de eventos, conforme ocorre no exemplo stress-test.c, exige a utilização de muitos códigos da AT-SPI, que estão escritos em linguagem C utilizando CORBA para comunicação, e que precisariam ser reescritos em JavaScript.

A solução encontrada, foi desenvolver um módulo, em linguagem C, e fazer com que esse módulo injete os eventos na AT-SPI. Este módulo segue o modelo do *stress-test.c* e é chamado de Injetor de Eventos. Para explicar a parte técnica de como um evento é injetado o Algoritmo 5.1 apresenta uma versão reduzida da função responsável por injetar o evento na AT-SPI. O código completo do Injetor de Evento é apresentado no Apêndice C.

```
01  void injetaEvento(){
02   atk_object_set_name (atko,("[%s|%s]",tipo, texto));
03   source = spi_accessible_new (atko);
04   e.type = "focus:";
05   e.source = BONOBO_OBJREF (source);
06   spi_init_any_nil (&e.any_data,
07        e.source,
08        Accessibility_ROLE_DESKTOP_FRAME,"");
09   Accessibility_Accessible_ref (e.source, &ev);
10   Accessibility_Registry_notifyEvent (registry, &e, &ev);
11   SPI_exit ();
12   return 0;
13 }
```

Algoritmo 5.1: Injeção de Evento na AT-SPI

Para gerar um evento de acessibilidade é necessário um componente gráfico acessível. No Algoritmo 5.1, a variável "*atko*" faz esse papel, sendo declarado como uma *window* (janela) do tipo *AtkObject*. A função na linha 02 nomeia esse componente grafico da seguinte forma: "[*tipo*|*texto*]", onde o tipo é um tipo representa a fonte do evento, podendo ser por exemplo terminal, *link* e *input*, enquanto o texto identifica o evento, podendo ser, por exemplo, um texto selecionado, ou o texto sobre um link.

O componente grafico "*atko*" é a fonte do evento de acessibilidade, e na linha 03 a variável "*source*", do tipo "*SpiAccessible*", é inicializada utilizando a função "*spi_accessible_new(AtkObject)*", que torna as informações de um objeto *ATK* uma fonte para um evento de acessibilidade.

A variável "*e"* do tipo "*Accessibility_Event*", é o evento acessível que será injetado. Na linha 04 o tipo do evento acessível "*e*" é declarado como "*focus:*" para indicar que o evento que está sendo injetado é de foco. Na linha 05 o a fonte do evento acessível "*e*" recebe uma referencia a variável "*source*", utilizando a função "*BONOBO_OBJREF(SpiAccessible)*".

O comando da linha 06, que se estende pela linha 07 e 08 faz uma inicialização do evento acessível "*e*" junto a AT-SPI, para que ele possa ser injetado. A comunicação com a AT-SPI é feita via CORBA, e por isso a variável "*ev*" do tipo "*CORBA_Environment*" é necessária para se comunicar com a biblioteca. Na linha 09 a fonte do evento é registrada como fonte de eventos acessíveis, utilizando a função "*Accessibility_Accessible_ref(SpiAccessible, CORBA_Environment)*", assim eventos causados por esta fonte são aceitos pela AT-SPI.

A variável "*registry*" do tipo "*Accessibility_Registry*", junto com a variável "*ev*" fazem uma linha de comunicação com a AT-SPI, que permite fazer registros na biblioteca. Na linha 10 o evento acessível "*e*" é injetado na AT-SPI utilizando a função "*Accessibility_Registry_notifyEvent(Accessibility_Registry, Accessibility_Event, CORBA_Environment)*". Na linha 10 a conexão com a AT-SPI é encerrada, e na linha 11 a função é encerrada. Desta forma, para que o Injetor de Eventos possa enviar diferentes textos para o Leitor de Tela, basta apenas modificar o campo nome do objeto "*atko*".

5.3 Notificação de Eventos

Como a injeção de eventos não ocorre no FireDEMO, e sim no módulo Injetor de Eventos, é necessário que ocorra a notificação dos eventos. Como o Injetor de Eventos é único e seu desenvolvimento é totalmente dependente do FireDEMO, não há problemas em existir uma conexão direta entre estes dos componentes.

Para que o Injetor de Eventos seja notificado dos eventos que o FireDEMO capta, foi utilizada a notificação de eventos via arquivo. A notificação de eventos ocorre da mesma forma com que ocorria para o Mouse Feedback. Sempre que um evento é detectado o complemento escreve as informações sobre este evento em um arquivo. O Injetor de Eventos, não sabe quando um evento ocorreu, nem o momento em que o FireDEMO escreveu no arquivo, então ele faz acessos constantes a este arquivo em busca de uma modificação no conteúdo dele desde o ultimo acesso. Se quando acessado o arquivo, o conteúdo for diferente do encontrado no ultimo acesso, isso significa que o FireDEMO detectou um novo evento e

escreveu no arquivo. Quando isso ocorre, o Injetor de Eventos chama a função para injetar o evento na AT-SPI.

A figura 5.1 apresenta um diagrama de sequencia para injeção de eventos na AT-SPI.

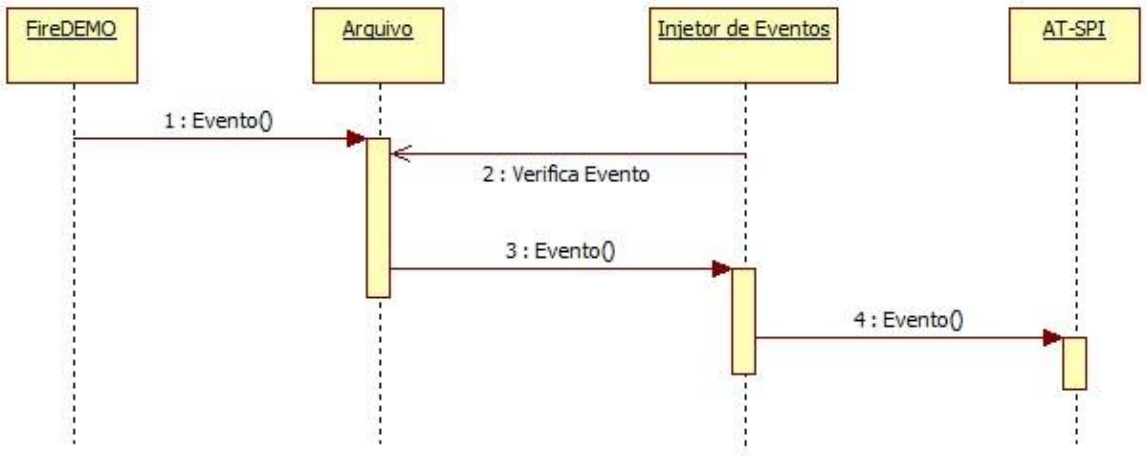


Figura 5.1: Diagrama de sequencia para injeção de eventos na AT-SPI

Os passos do diagrama de sequencia apresentado na figura 5.1 são descritos abaixo:

1. Evento: O FireDEMO capta um evento, então escreve suas informações no arquivo.

2. Verifica evento: O Injetor de Eventos acessa o arquivo em busca de conteúdo novo.

3. Evento: O conteúdo do novo evento é recuperado pelo Injetor de Evento.

4. Evento: O Injetor de Eventos passa o evento para a AT-SPI como um evento do tipo *Focus*.

Os eventos são injetados como eventos do tipo *Focus* pois o ponteiro do *mouse* traz foco ao campo de interesse pelo qual ele passa. O FireDEMO recupera as informações sobre componente que gerou o evento e escreve estas informações no arquivo.

5.4 Eventos injetados

Quando detecta um evento o FireDEMO escreve as seguintes informações no arquivo:

- **title:** Texto com o titulo do objeto. O texto depende do desenvolvedor da pagina *web*.

- **class:** A classe que identifica qual o tipo do objeto. Existe uma variedade muito grande de classes para definir o tipo. Alguns exemplos encontrados na página do Google (2012) são: “l” para *link* em sobre um texto, “rg_hi uh_hi” para *link* sobre uma imagem, “gbqfif” para o *input* de texto para busca.
- **innerHTML:** O que aparece na pagina *web*. Pode ser um texto, um endereço para uma imagem ou não possuir nada.

Quando o injetor de eventos recupera as informações do arquivo, ele monta um evento no padrão da AT-SPI. A figura 5.2 apresenta o exemplo de um evento da AT-SPI identificado pelo *Accerciser*.

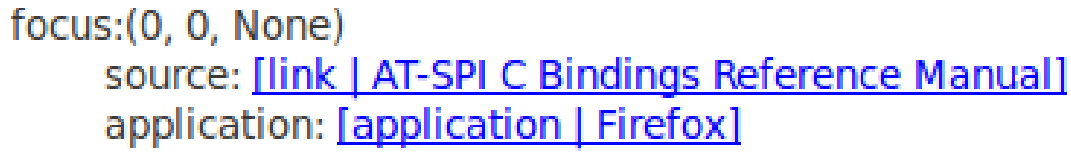


Figura 5.2: Evento padrão da AT-SPI.

A estrutura de um evento e a seguinte:

- **Tipo:** O tipo do evento, ele pode ser: *focus*, *document*, *mouse*, *object*, *terminal* e *window*. Na Figura 5.2, e em todos os casos do Injetor de Eventos, o evento é do tipo de *focus*.
- **Source:** A fonte do evento. Esse atributo é dividida em duas partes, o objeto e o texto.
  - **Objeto:** O objeto que causou o evento. Existem vários tipos de objeto, como por exemplo *link, button, window,* etc. Na Figura 5.2 temos um *link*.
  - **Texto:** O texto referente ao objeto. Na Figura 5.2 o texto referente ao objeto é “AT-SPI C Bindings Reference Manual”.
- **Aplicação:** A aplicação em que ocorreu o evento. No caso da aplicação desenvolvida será o injetor de eventos. Como nenhuma janela principal é criada, todos os eventos chegam com esse parâmetro nulo.

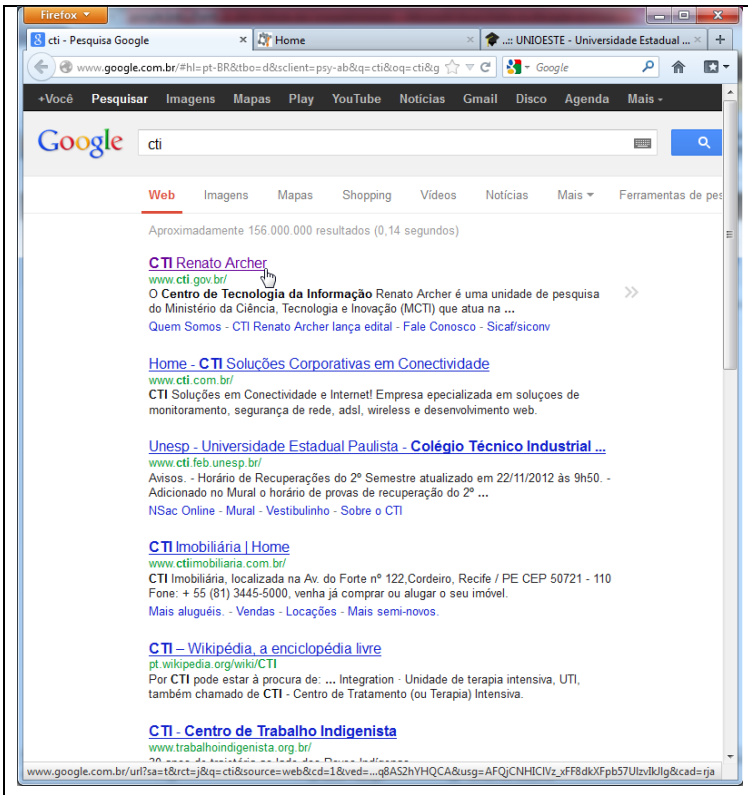
O tipo do evento é a única informação relevante para o Mouse Feedback. Se o tipo for *focus* então o *feedback* do Mouse AFM deve ser ativado, das outras informações do evento.

Para o Leitor além do tipo, o texto contido na fonte do objeto é importante, pois é esse texto que deve ser lido.

Um problema encontrado, que impediu que na primeira versão fosse possível ler uma identificação para qualquer componente, foi que componentes em diferentes páginas *web* não mantem um padrão em suas informações. Muitas páginas *web* não apresentam os atributos *class* e *title* em seus componentes, e o innerHTML nem sempre é um texto que possa ser lido.

Por exemplo, páginas como a do CTI (2012) e da UNIOESTE (2012) não apresentam o atributo *class* em nenhum componente, apresentam *title* apenas quando se tem um *link* para uma imagem. Diferente destes exemplos, página do Google (2012) apresenta *class* para todos seus componentes, mas nunca apresenta o *title*. Diferentes páginas ainda seguem outros padrões.

As figuras abaixo apresentam exemplos das informações *title*, *class* e *innerHTML* nos componentes das páginas do CTI e Google.

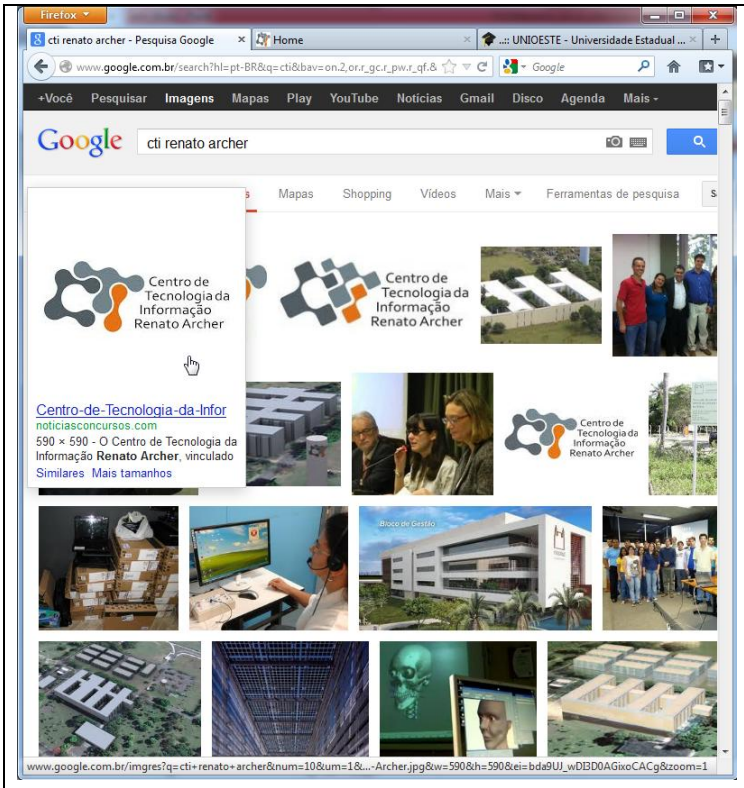


Title: sem informação

Class: "l"

innerHtml: "CTI Renato Archer"

Figura 5.3: *Mouse over* sobre *link* (texto) na página do Google

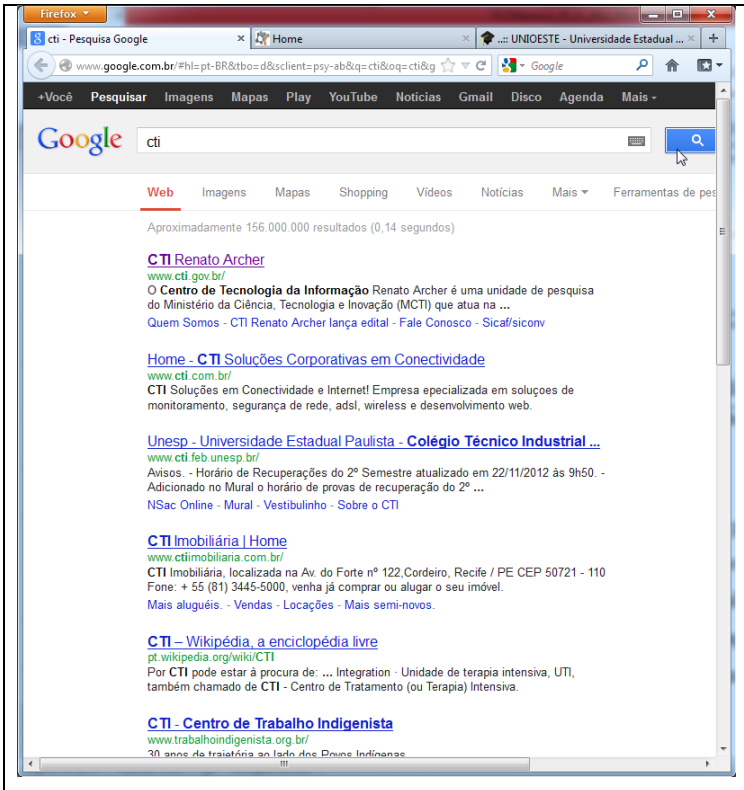


Title: sem informação

Class: “rg_hi uh__hi”

innerHtml: “<img class="rg_hi uh_hi" id="rg_hi" alt="" width="225" height="225" data-width="225" data-height="225" style="width:225px;height:225px" src=".../Z">

Figura 5.4: *Mouse over* sobre *link* (imagem) na página do Google

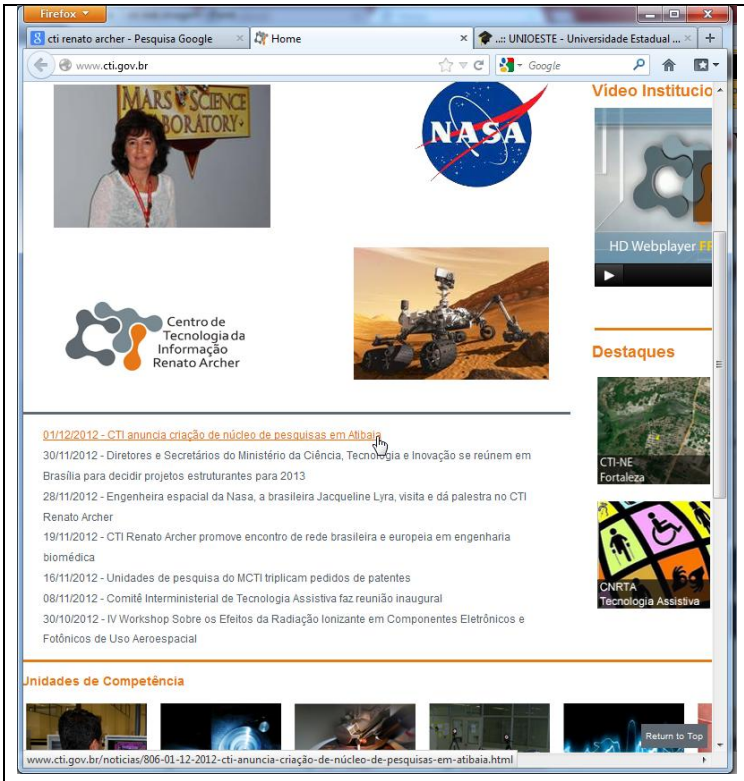


Title: sem informação

Class: “gbqfb”

innerHtml: sem informação

Figura 5.5: *Mouse over* sobre um botão na página do Google

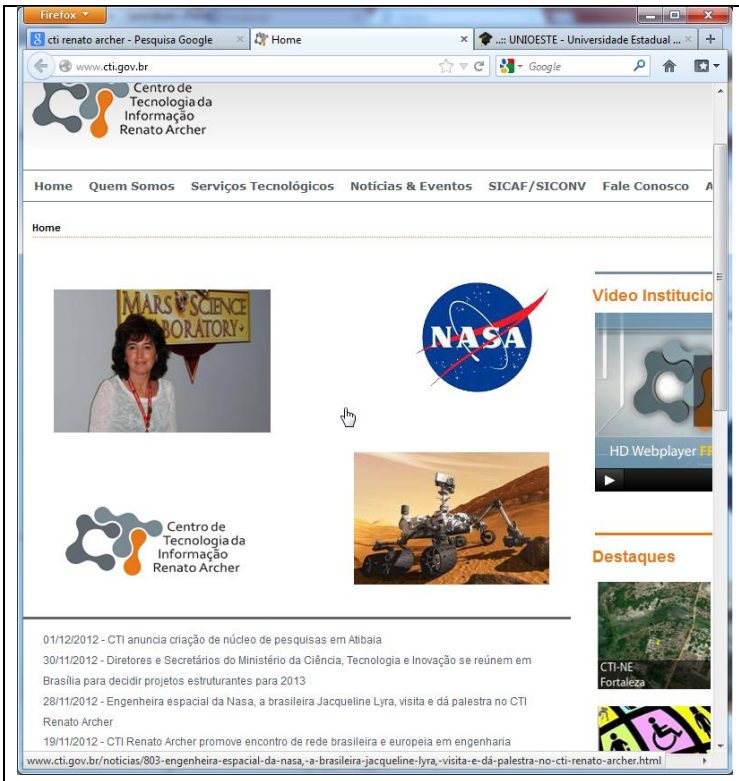


Title: sem informação

Class: sem informação

innerHtml: “01/12/2012 - CTI anuncia criação de núcleo de pesquisas em Atibaia”

Figura 5.6: *Mouse over* sobre *link* (texto) na página do CTI

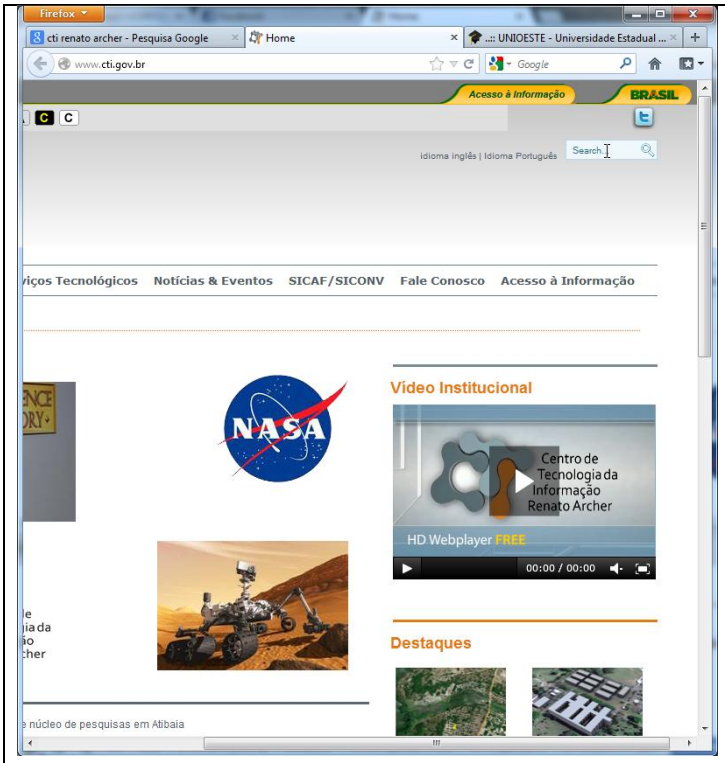


Title: “Engenheira espacial da Nasa, a brasileira Jacqueline Lyra, visita e dá palestra no CTI Renato Archer”

Class: sem informação

innerHtml: “<img src="http://www.cti.gov.br/media/mod_ninja_simple_icon_menu/images/engenheira_nasa2.png" alt="Engenheira espacial da Nasa, a brasileira Jacqueline Lyra, visita e dá palestra no CTI Renato Archer " height="370" width="600">”

Figura 5.7: *Mouse over* sobre um *link* (imagem) na página do CTI



Title: sem informação

Class: sem informação

innerHtml: sem informação

Figura 5.8: *Mouse over* sobre um campo de entrada na página do CTI

As figuras 5.3, 5.4, 5.5, 5.6, 5.7 e 5.8 apresentam exemplos de *mouse over* sobre determinados campos das páginas do Google e CTI, e as informações que o FireDEMO escreve no arquivo quando os eventos ocorrem. As figuras mostram a variação de informações fornecidas por diferentes componentes em diferentes sites, o que dificultou a passagem de informações para leitura.

Na versão atual do Injetor de Eventos, por falta de tempo para concluir o projeto, apenas eventos com o atributo *class* sendo "l", ou seja, um *link* em forma de texto são injetados na AT-SPI com informações suficientes para serem lidos pelo Leitor de Tela. Essa escolha foi feita por permitir a leitura de links nos resultados de pesquisa no Google.

Para escrever o atributo *source* (fonte do evento) é necessário nomear o objeto ATK que foi criado para gerar o evento. Quando o Injetor de Eventos recupera as informações sobre o evento escritas no arquivo ele verifica se o conteúdo encontrado na *class* é "l". Se for, então o nome atribuído ao objeto ATK é "[link|" mais o conteúdo encontrado no *innerHTML* mais "]". Se o conteúdo de *class* não for "l" então o nome atribuído ao objeto ATK é "[|]". Depois disso o objeto é declarado como fonte do evento acessível, e o tipo do evento é declarado

como “focus:”, e o evento acessível está pronto para ser injetado utilizando uma função da AT-SPI.

Para que o Leitor de Tela identifique outros componentes o FireDEMO poderia escrever no arquivo além do *title, class* e *innerHTLM* também a *tagName* do campo de interesse por qual o mouse passou. A *tagName* identifica qual o campo causou o evento, podendo ser INPUT, para campos de entrada, BUTTON, para botões , A para *links*, e TEXTAREA, para áreas de texto.

Com isso deveria ser implementado um filtro no Injetor de Eventos, para avaliar as informações lidas no arquivo e assim montar o evento de forma que o texto identifique o componente quando lido pelo Leitor de Tela. A Figura 5.8 apresenta um diagrama de estado para esse filtro.

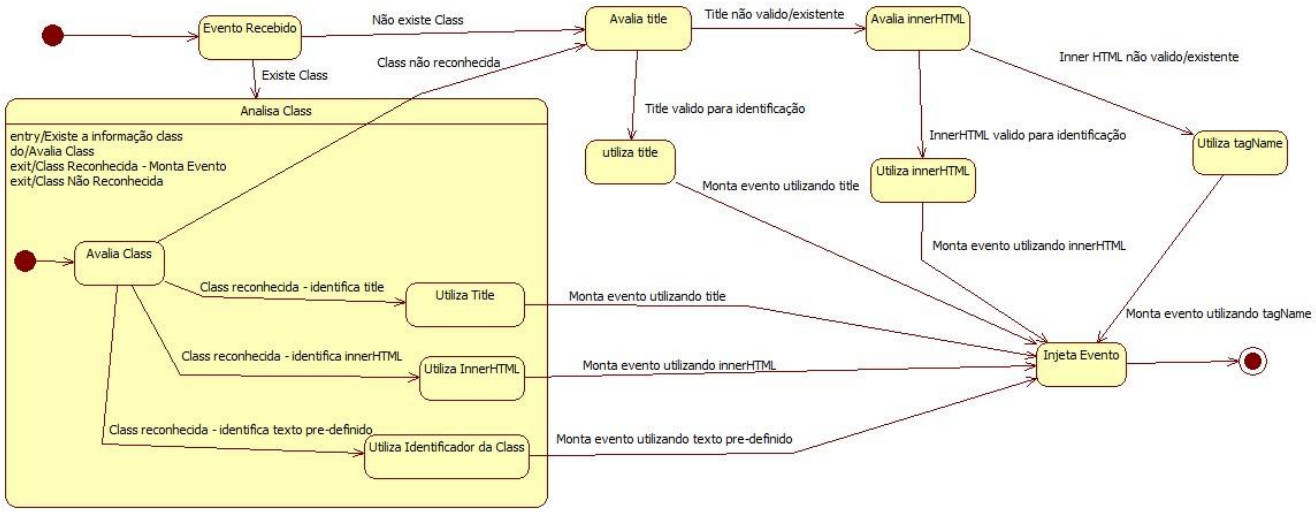


Figura 5.8: Diagrama de estado para o filtro do Injetor de Eventos

Conforme pode ser visto no diagrama apresentado na Figura 5.8, o Injetor de Eventos, após receber as informações de um evento, verifica a existência do atributo *class* nessas informações. Caso esse atributo exista, o filtro tenta reconhecer o conteúdo deste atributo, para saber se o *title, innerHTML,* ou mesmo um identificador já cadastrado para esse atributo pode ser utilizado para reconhecer o evento. Caso a *class* não exista, ou não seja reconhecida, verifica-se a existência do atributo *title*, e se ele pode identificar o componente que gerou o evento. Caso o *title* não exista, ou não identifique o componente, verifica-se a existência do

atributo *innerHTML*, e se ele pode identificar o componente que gerou o evento. Em ultimo caso, se o *innerHTML*, não existir, ou não identifique o componente, utiliza-se a tagName para identificar o evento. Nesse caso, o texto será referente apenas ao tipo de item que gerou o evento, podendo ser *link*, botão, campo de entrada ou área de texto.

Capítulo 6

Considerações Finais

Nesse trabalho foi feito um estudo da biblioteca AT-SPI e como ela permite que desenvolvedores forneçam acessibilidade em suas aplicações. Estudando a API foi possível levantar os tipos de eventos gerados por ela e que eventos as aplicações, o Firefox em especifico, fornecia para a biblioteca.

Também foi estudado o PlatMult, e como o Mouse Feedback e o leitor (Balansin, 2011) tem utilizado a AT-SPI para prover acessibilidade. Inicialmente o Mouse Feedback e o FireDEMO foram os principais alvos do trabalho, tendo sido feito um estudo mais aprofundado no seu desenvolvimento e métodos de comunicação baseado em comunicação via *socket*.

Um novo método foi desenvolvido como parte do trabalho, para que o FireDEMO notificasse o Mouse Feedback dos eventos de uma forma mais eficiente, mas os testes não mostraram vantagem no novo método desenvolvido, além de apresentar um comportamento que inviabilizou seu uso.

A arquitetura que ligava o FireDEMO e o Mouse Feedback foi alterada, para que o FireDEMO ficasse em um módulo separado, e então foi desenvolvida uma maneira que permitisse que os eventos captados fossem injetados diretamente na AT-SPI, para que qualquer aplicação que forneça acessibilidade possa utilizar dos eventos. Isso permite que o Mouse Feedback funcione de forma equivalente a versão em que o FireDEMO estava diretamente ligado a ele, e permite que outro recurso do PlatMult, o leitor (Balansin, 2011) utilize os eventos para leitura do conteúdo de uma pagina web. O código do Mouse Feedback, do FireDEMO e do Injetor de Eventos são apresentados, respectivamente, no Apêndice A, Apêndice B e Apêndice C.

6.1 Trabalhos Futuros

Uma versão do Mouse AFM com dois motores vibratórios em lados opostos do dispositivo está sendo planejada. O desenvolvimento de um método que identifique a localização de um item que disparou o evento em relação ao ponteiro do *mouse*, para ativar o *feedback* em um dos motores, de forma que o usuário perceba a direção do item que gerou o *feedback* pode facilitar ainda mais para que o usuário se localize enquanto navega pelo sistema. Para fazer isso seria necessário obter também a localização do componente que gerou o evento e a localização do ponteiro do *mouse* segundos após o evento, para escolher o motor a ser ativado de forma que induza o usuário a seguir até o componente.

No complemento FireDEMO, seria interessante o desenvolvimento de um menu de configuração, onde o usuário pode ver os componentes disponíveis para ativação do feedback, e ativar e desativar conforme sua preferencia.

A proposta mais interessante como trabalho futuro é trabalhar no desenvolvimento de um filtro no Injetor de Eventos, como discutido no final da seção 5.4. O estudo dos componentes de uma página *web*, junto com o desenvolvimento de um filtro no Injetor de Eventos, com objetivo de que os eventos injetados tenham informações o suficiente para que o Leitor de Tela possa ler um texto que identifique qualquer componente em uma página web seria uma grande inclusão de acessibilidade no PlatMult.

Apêndice A

Algoritmo Mouse Feedback

```
/*
 * AT-SPI - Assistive Technology Service Provider Interface
 * (Gnome Accessibility Project; http://developer.gnome.org/projects/gap)
 *
 * Copyright 2001, 2002 Sun Microsystems Inc.,
 * Copyright 2001, 2002 Ximian, Inc.
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library; if not, write to the
 * Free Software Foundation, Inc., 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 */

#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/ppdev.h>
#include <linux/parport.h>
#include <stdio.h>
#include <strings.h>
```

```
#include <stdlib.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/un.h>
#include <sys/io.h>
#undef MAGNIFIER_ENABLED
#include <cspi/spi.h> /* A hack for now */
#include <pthread.h>

//#define PRINT_TREE

static void report_focus_event (const AccessibleEvent *event, void
*user_data);
static void get_environment_vars (void);
//MSO
void init_parport();
void rumble();


#ifdef PRINT_TREE
static void print_accessible_tree (Accessible *accessible, char *prefix);
#endif

#ifdef MAGNIFIER_ENABLED
static SPIBoolean use_magnifier = FALSE;
#endif

static SPIBoolean use_festival = FALSE;
static SPIBoolean festival_chatty = FALSE;
static SPIBoolean name_changed = FALSE;

static AccessibleEventListener *focus_listener;

//MSO

int fd, mode;
int flag_rumble;
```

```
int
main (int argc, char **argv)
{
  int i, j;
  int n_desktops;
  int n_apps;
  char *s;
  Accessible *desktop;
  Accessible *application;
  const char *modules;

  if ((argc > 1) && (!strncmp (argv[1], "-h", 2)))
    {
      printf ("Usage: simple-at\n");
      printf                    ("\tEnvironment                    variables
used:\n\t\tFESTIVAL\n\t\tMAGNIFIER\n\t\tFESTIVAL_CHATTY\n");
      exit (0);
    }

  modules = g_getenv ("GTK_MODULES");
  if (!modules || modules [0] == '\0')
    {
      putenv ("GTK_MODULES=");
    }
  modules = NULL;
  init_parport();
  SPI_init ();

  focus_listener  =  SPI_createAccessibleEventListener  (report_focus_event,
NULL);
  SPI_registerGlobalEventListener (focus_listener, "focus:");
  n_desktops = SPI_getDesktopCount ();

  for (i=0; i<n_desktops; ++i)
    {
      desktop = SPI_getDesktop (i);
      s = Accessible_getName (desktop);
      fprintf (stderr, "desktop %d name: %s\n", i, s);
```

```
      SPI_freeString (s);
      n_apps = Accessible_getChildCount (desktop);
      for (j=0; j<n_apps; ++j)
        {
          application = Accessible_getChildAtIndex (desktop, j);
        s = Accessible_getName (application);
          fprintf (stderr, "app %d name: %s\n", j, s ? s : "(nil)");
#ifdef PRINT_TREE
        print_accessible_tree (application, "*");
#endif
          SPI_freeString (s);
          Accessible_unref (application);
        }
      Accessible_unref (desktop);
    }



  get_environment_vars ();

  SPI_event_main ();

  putenv ("AT_BRIDGE_SHUTDOWN=1");

  return SPI_exit ();
}



void init_parport(){

      ioperm(0x378,3,1); //inicializa a porta paralela
      outb(0x02, 0x378);
        sleep(1) ;
      outb(0x00, 0x378);

   if ((fd=open("/dev/parport0", O_RDWR))<0){
      printf("\nErro ao abrir parport0: porta paralela nao encontrada\n");
      exit(0);
```

```
   }
   if (ioctl(fd, PPCLAIM)<0){
      printf("\nErro ao obter acesso a porta PPCLAIM: execute como
root\n");
      perror("PPCLAIM");
      close(fd);
      exit(0);
   }


   mode = IEEE1284_MODE_COMPAT;
   if (ioctl(fd, PPSETMODE, &mode)<0){
      printf("\nErro ao setar o modo da porta PPSETMODE: execute como
root\n");
      perror("PPSETMODE");
      close(fd);
      exit(0);
   }

}

void rumble() {
        char buff[1];
      buff[0]= 1;
      outb(0x02, 0x378);
      usleep(600000);
      outb(0x00, 0x378);
      buff[0]= 0;
       fflush(stdout);
}

static void
get_environment_vars (void)
{
  if (g_getenv ("FESTIVAL"))
    {
      fprintf (stderr, "Using festival\n");
      use_festival = TRUE;
      if (g_getenv ("FESTIVAL_CHATTY"))
```

```
        {
          festival_chatty = TRUE;
      }
    }
#ifdef MAGNIFIER_ENABLED
  if (g_getenv ("MAGNIFIER"))
    {
      fprintf (stderr, "Using magnifier\n");
      use_magnifier = TRUE;
    }
  else
    {
      fprintf (stderr, "Not using magnifier\n");
    }
#endif

  if (!use_festival)
    {
      fprintf (stderr, "No speech output\n");
    }
}

#ifdef PRINT_TREE
static void
print_accessible_tree (Accessible *accessible, char *prefix)
{
      int n_children;
      int i;
      char *name;
      char *role_name;
      char *parent_name = NULL;
      char *parent_role = NULL;
      char child_prefix[100];
      Accessible *child;
      Accessible *parent;

      strncpy (child_prefix, prefix, 98);
      strcat (child_prefix, "*");
      parent = Accessible_getParent (accessible);
```

```
      if (parent)
        {
            parent_name = Accessible_getName (parent);
            parent_role = Accessible_getRoleName (parent);
              Accessible_unref (parent);
        }
      name = Accessible_getName (accessible);
      role_name = Accessible_getRoleName (accessible);
      fprintf (stdout, "%sAccessible [%s] \"%s\"; parent [%s] %s.\n",
             prefix, role_name, name ? name : "(nil)",
             parent_role ? parent_role : "(nil)",
             parent_name ? parent_name : "(nil)");
      SPI_freeString (name);
      SPI_freeString (role_name);
      SPI_freeString (parent_name);
      SPI_freeString (parent_role);
      n_children = Accessible_getChildCount (accessible);
      for (i = 0; i < n_children; ++i)
              {
                  child = Accessible_getChildAtIndex (accessible, i);
                  print_accessible_tree (child, child_prefix);
                  Accessible_unref (child);
              }
}
#endif

void
report_focus_event (const AccessibleEvent *event, void *user_data)
{
  char *s;

  //g_return_if_fail (event->source != NULL);
  s = Accessible_getName (event->source);
  if (s)
    {
      //fprintf (stderr, "%s \t %s\n", event->type, s);
      //printf("%s \n" , s);
      rumble();
        flag_rumble=1;
```

```
      SPI_freeString (s);
      //report_focussed_accessible (event->source, TRUE);

    }
  Accessible_getParent (event->source);
  name_changed = FALSE;
}
```

Apêndice B

Algoritmo FireDEMO

```
var platmult = {
    onLoad: function() {
        // initialization code
        this.initialized = true;
        this.strings = document.getElementById("platmult-strings");
        platmult.debug('platmult.onLoad()');
    },

    onPageLoad: function(obj, evt) {
        try{

//            platmult.debug('nomeName: '+evt.originalTarget.nodeName);
//                                                platmult.debug('onPageLoad:
'+evt.originalTarget.defaultView.location.href);

            if (evt.originalTarget.nodeName == '#document')  {

                var                          pageDoc                          =
document.commandDispatcher.focusedWindow.document;


                pageDoc.addEventListener ('DOMNodeInserted', function(e){
                    platmult.OnDOMNodeInserted(this,e);
                }, false);

                var inputList = pageDoc.getElementsByTagName('input');
                for (var i=0; i<inputList.length; i++) {
                    //platmult.debug('Registrou         listener         em:
'+inputList[i]);
```

```
                    inputList[i].addEventListener("mouseover",   function(e)
{
                        platmult.onMouseOver(this, e);
                    }, false);
                }
                var buttonlist = pageDoc.getElementsByTagName('button');
                for (var i=0; i<buttonlist.length; i++) {
                    //platmult.debug('Registrou         listener         em:
'+buttonlist[i]);
                    buttonlist[i].addEventListener("mouseover",  function(e)
{
                        platmult.onMouseOver(this, e);
                    }, false);
                }
                var alist = pageDoc.getElementsByTagName('a');
                for (var i=0; i<alist.length; i++) {
                    //platmult.debug('Registrou listener em: '+alist[i]);
                    alist[i].addEventListener("mouseover", function(e) {
                        platmult.onMouseOver(this, e);
                    }, false);
                }
                var talist = pageDoc.getElementsByTagName('textarea');
                for (var i=0; i<talist.length; i++) {
                    //platmult.debug('Registrou listener em: '+talist[i]);
                    talist[i].addEventListener("mouseover", function(e) {
                        platmult.onMouseOver(this, e);
                    }, false);
                }
                obj.platMultIsLoaded = true;
            }


        }catch(err){
            Components.utils.reportError(err);
        }
    },
    OnDOMNodeInserted: function(obj, evt) {
        try{
```

```
            //platmult.debug('OnDOMNodeInserted    obj:'+obj+'    inseriu:
'+evt.target);
            if((evt.target.tagName == 'INPUT') ||
                (evt.target.tagName == 'BUTTON') ||
                (evt.target.tagName == 'A') ||
                (evt.target.tagName == 'TEXTAREA') ){
                evt.target.addEventListener("mouseover", function(e) {
                    platmult.onMouseOver(this, e);
                }, false);
            }
        }catch(err){
            Components.utils.reportError(err);
        }
    },
    onMouseOver: function(obj, mouseOverEvent) {
        try{
            platmult.debug('mouseOverEvent   obj:'+obj+'   id:   '+obj.id+'
className: '+obj.className+' innerHtml: '+obj.innerHTML);
          platmult.writeEvent(obj);
        }catch(err){
            Components.utils.reportError(err);
        }
    },

    debug: function(txt){
        try{
            var                        consoleService                        =
Components.classes["@mozilla.org/consoleservice;1"].getService(Components.i
nterfaces.nsIConsoleService);
            consoleService.logStringMessage('PlatMult debug: '+txt);
        }catch(err){
            Components.utils.reportError(err);
        }
    },

    writeEvent: function(obj){
      url = "http://127.0.0.1:80/";
      data = {msg: 'id: '+obj.id+' className: '+obj.className+' innerHtml:
'+obj.innerHTML};
```

```
      $.post(url, data);
      /*
      try{
            var data = new Date().toString()+'\n';

            var                              file                              =
Components.classes["@mozilla.org/file/local;1"].createInstance(Components.i
nterfaces.nsILocalFile);
            file.initWithPath("/home/cgalesky/platmult");
            platmult.debug('file.isWritable(): '+file.isWritable());
            var   stream   =   Components.classes["@mozilla.org/network/safe-
file-output-
stream;1"].createInstance(Components.interfaces.nsIFileOutputStream);
            stream.init(file,   0x10 , -1, null); // open file
//          stream.init(file, 0x04 | 0x10 | 0x20, 0600, 0); // open file
for appending
            stream.write(data, data.length);
            if                        (stream                        instanceof
Components.interfaces.nsISafeOutputStream) {
                stream.finish();
            } else {
                stream.close();
            }

      }catch(err){
            Components.utils.reportError(err);
      }
      */
    }
};

window.addEventListener("load", function () {
    platmult.onLoad();
}, false);
gBrowser.addEventListener("load", function (e) {
    platmult.onPageLoad(this,e);
} , true);
```

Apêndice C

Algoritmo Injetor de Eventos

```
/*
 * AT-SPI - Assistive Technology Service Provider Interface
 * (Gnome Accessibility Project; http://developer.gnome.org/projects/gap)
 *
 * Copyright 2001, 2002 Sun Microsystems Inc.,
 * Copyright 2001, 2002 Ximian, Inc.
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library; if not, write to the
 * Free Software Foundation, Inc., 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 */

#include <unistd.h>
#include <stdlib.h>
#include "../cspi/spi-private.h" /* A hack for now */
#include <glib-object.h>
#include <gtk/gtk.h>
#include <atk/atk.h>
```

```
#include <atk/atknoopobject.h>
#include <bonobo-activation/bonobo-activation-register.h>
#include <bonobo/bonobo-main.h>
#include <libspi/libspi.h>

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#include <pthread.h>

      Accessibility_Event e;
      Accessibility_Registry registry;
      CORBA_Environment ev;
      SpiAccessible *source;
      GObject *object;
      AtkObject *atko;


void injetaEvento(char *s){
      char *class;
      char *title;
      char *innerHtml;

      class = strtok(s, "|");
      title = strtok(NULL, "|");
      innerHtml = strtok(NULL, "|");

      /*printf("class: %s\t title: %s\t innerHTML: %s\n", class, title,
innerHtml);
      fflush(stdout);*/

      if(strcmp(class, "l")==0){
            if(innerHtml!=NULL){
                  /*printf("[link|%s]\n",innerHtml);
                  fflush(stdout);*/
                  atk_object_set_name (atko, ("[link|%s]",innerHtml));
```

```
            }else if(title !=NULL){
                  /*printf("[link|%s]\n",title);
                  fflush(stdout);*/
                  atk_object_set_name (atko, ("[link|%s]",title));
            }
      }else{
            atk_object_set_name (atko, "[ | ]");
      }

      source = spi_accessible_new (atko);
      e.type = "focus:";
      e.source = BONOBO_OBJREF (source);
      e.detail1 = 0;
      e.detail2 = 0;

      spi_init_any_nil (&e.any_data,
            e.source,
            Accessibility_ROLE_DESKTOP_FRAME,
            "");
      Accessibility_Accessible_ref (e.source, &ev);
      Accessibility_Registry_notifyEvent (registry, &e, &ev);

      SPI_exit ();
      return 0;
}

int
main (int argc, char **argv)
{
      gtk_init (NULL, NULL);
      SPI_init ();
      object = g_object_new (GTK_TYPE_WINDOW, NULL);
      gtk_window_set_title (object, "teste");
      atko = atk_no_op_object_new (object);

      sleep (1);

      CORBA_exception_init (&ev);
```

```
  registry = bonobo_activation_activate_from_id (
        "OAFIID:Accessibility_Registry:1.0", 0, NULL, &ev);

  if (ev._major != CORBA_NO_EXCEPTION){
        g_error ("Accessibility app error: exception during "
         "registry activation from id: %s\n",
         CORBA_exception_id (&ev));
        CORBA_exception_free (&ev);
  }

  if (registry == CORBA_OBJECT_NIL) {
        g_error ("Could not locate registry");
  }

  bonobo_activate ();

  FILE *fp;
  char s[255];
  char s1[255];
  s[254]= '\0';
  s1[0]= '\0' ;
  int i=0;
  while (1) {
        for(i=0; i<255; i++)
              s[i]='\0';
        fp= fopen("/home/lsc/platmult" ,"r" );
        fread(s, 1, 254, fp);
        if (strcmp(s, s1) != 0 ) {
              for(i=0; i<255; i++)
                    s1[i]='\0';
              strcpy(s1, s);
              injetaEvento(s);
        }
        fclose(fp);
        usleep(500);
  }

}
```

Referências Bibliográficas

[Balansin, 2011] Balansin, C. Especificação e Implementação de um Leitor de Tela. Trabalho de Conclusão de Curso: Ciência da Computação. Unioeste, 2011.

[Bersch, 2008] Bersch, R. Introdução à Tecnologia Assistiva. CEDI • Centro Especializado em Desenvolvimento Infantil. Porto Alegre. 2008. Consultado na internet: http://200.145.183.230/TA/4ed/material_apoio/módulo2/M2S1A5_Introducao_TA_Rita_Bersch.pdf, em 09/07/2012.

[Bidarra, Boscarioli, Rizzi, 2009] Bidarra, J.; Boscarioli, C.; Rizzi, C. B. xLupa – um ampliador de tela com interface adaptativa para pessoas com baixa visão. In: Melo, A. M.; Piccolo, L. S. G.; Ávila, I. M. A; Tambascia, C. A. (Org.). Usabilidade, Acessibilidade e Inteligibilidade Aplicadas em Interfaces para Analfabetos, Idosos e Pessoas com Deficiência: Resultados do Workshop. Campinas: CPqD, 2009. p. 23-30.

[Bidarra, Oyamada, 2011] Bidarra, J. Oyamada, M. S. Development of an interactive kiosk with screen amplifier targeting low vision and old-aged people. In: AIRTECH - Accessibility, Inclusion and Rehabilitation using Information Technologies. Havana- Cuba, 2011.

[Canonical, 2012] Canonical. ubuntu-br. 2011. Consultado na internet: http://www.ubu ntu-br.org/ubuntu, em 09/07/2012.

[CTI, 2012] CTI – Renato Archer, 2012, Consultado na internet: http://www.unioeste.br/, em 11/11/2012.

[Foundation, 1995] Foundation, M. Mozilla Developer Network. 1995. Consultado na internet: https://developer.mozilla.org/en/JavaScript , em 09/07/2012.

[Foundation, 2004] Foundation, M. Mozilla Firefox. 2004. Consultado na internet: http://www.mozilla.org/en-US/firefox/new/ , em 09/07/2012.

[Foundation, 2008] Foundation, F. S. at-spi - Free Software Directory. 2008. Consultado na internet: http://directory.fsf.org/project/at-spi/, em 09/07/2012.

[Foundation, 2009] Foundation, L. IAccessible2. 2009. Consultado na internet: http://www.linuxfoundation.org/collaborate/workgroups/accessibility/iaccessible2, em 09/07/2012.

[Foundation, 2012] Foundation, T. j. jQuery write less, do more. 2012. Consultado na internet: http://jquery.com/ em 09/07/2012.

[Frauenberger, Höldrich, 2004] Frauenberger, C.; Höldrich, R. - A Generic, Semantically Based Design Approach For Spatial Auditory Computer Displays. In Proceedings of ICAD 04-Tenth Meeting of the International Conference on Auditory Display, Sydney, Australia, July 6-9, 2004.

[Google, 2012] Google, 2012, Consultado na Internet: https://www.google.com.br/, em 11/11/2012.

[IBGE, 2012] IBGE 2012- Censo Demográfico 2000. Consultado na internet: http://www.ibge.gov.br/ , em 09/07/2012.

home/presidencia/noticias/27062003censo.shtm, em 28/03/2012.

[Isaacson, 2011] ISAACSON, E. Accerciser Manual v0.2.0. 2011. Consultado na internet: http://librarygnome.org/devel/accerciser/1.12/accerciser.html , em 09/07/2012.

[Lee, 2008] LEE, S. Python Powered Accessibility. 2008. Consultado na internet: http://live.gnome.org/Accessibility/PythonPoweredAccessibility, em 09/07/2012.

[Kim, Park, Park, 2011] Kim, L. Park, W. Park, S. Haptic Mouse Interface Actuated by an Electromagnet. International Conference on Complex, Intelligent, and Software Intensive Systems. 2011.

[Kumazawa, 2010] Kumazawa, I. Haptic Mouse with Quick and Flexible Tactile Feedback Generated by Double Control Loop. 19th IEEE International Symposium on Robot and Human Interactive Communication. Principe di Piemonte - Viareggio, Italy. 2010.

[Masini, 1994] Masini, E F S. A educação do portador de deficiência visual: as perspectivas do vidente e do não vidente- 1 In: Alencar, EML. Tendência e desafios da deficiência visual. Brasília: MEC/SEESP;1994. 193p. Consultado na internet: http://emaberto.inep.gov.br/index. php/emaberto/issue/view/67, em 09/07/2012.

[Project, 2005] Project, T. G. Orca. 2005. Consultado na internet: https://live.gnome.org/Or ca, em 09/07/2012.

[Project, 2010] Project, T. G. Java Access Bridge. 2010. Consultado na internet: https://live.gnome.org/Java%20Access%20Bridge/ , em 09/07/2012.

[Project, 2012] Project, T. G. Accercise. 2012. Consultado na internet: https://live.gnome.org/Accerciser/ , em 09/07/2012.

[Radabaugh, 1993] Radabaugh, M. P. NIDRR's Long Range Plan - Technology for Access and Function Research Section Two: NIDDR Research Agenda Chapter 5: TECHNOLOGY FOR ACCESS AND FUNCTION – Consultado na internet: http://www.ncddr.org/new/announceme nts/lrp/fy1999-2003/lrp_techaf.html, em 09/07/2012.

[UNIOESTE, 2012] UNIOESTE – Universidade Estadual do Oeste do Paraná, 2012, Consultado na internet: http://www.unioeste.br/, em 11/11/2012.

[WHO, 2012] WHO - World Health Organization, 2012, Consultado na internet: http://www.who.int/en/, em 11/11/2012.

[Xorg, 2012] Xorg. xorg(1) - Linux man page. Consultado na internet: http://linux.die.net/man/1/xorg. em 09/07/2012.