# Editor para Textos em Língua de Sinais Escritos em SignWriting[1]

**Rafael P. Torchelsen, Antônio Carlos R. Costa, Graçaliz P. Dimuro**
Universidade Católica de Pelotas, Escola de Informática,
Félix da Cunha 412, Pelotas, Brasil, 96010-000
{toto,rocha,liz}@ucpel.tche.br

## Abstract

This work concerns the development of a set of programs to help deaf users in the creation of texts in sign language, based on the system of sign representation called *SignWriting*. The main program is an editor – called *SWEdit* -, to be used to edit texts in sign languages, and an auxiliary program - called *AlfaEdit* -, that updates the sets of symbols used by *SWEdit*. Both have been developed especially for deaf people, with interfaces that explore the ability of visual interpretation of deaf people, through the use of images instead of ordinary texts in the menus.

**Keywords:** *SignWriting*, C++, Multi-Plataform Programs, *wxWindows*, Deaf, Sign Language, Deaf Education, Deaf-adapted interfaces.

## Resumo

Este trabalho consiste no desenvolvimento de um sistema para auxiliar o usuário surdo na criação de textos em linguagem de sinais, baseados no sistema de representação de sinais *SignWriting*. O sistema consiste de um editor, chamado *SWEdit*, para criação dos textos propriamente ditos, e da ferramenta *AlfaEdit*, que auxilia na atualização dos conjuntos de símbolos utilizados no editor. Ambos foram desenvolvidos especialmente para os surdos, com interfaces que exploram a capacidade de interpretação visual dos surdos, através da utilização de figuras onde normalmente seriam utilizados textos.

**Palavras-chaves:** *SignWriting*, C++, Programas Multi-Plataforma, *wxWindows*, Surdos, Língua de Sinais, Educação de Surdos, Interfaces voltadas para Surdos.

[1] Este trabalho contou com financiamento do CNPq e da FAPERGS.

# 1. Introdução

A história da educação dos surdos é rica em concepções, experiências, alternativas pedagógicas e metodológicas, desde as primeiras iniciativas sistemáticas realizadas no século XVIII [3]. Neste trabalho, parte-se do reconhecimento de que as línguas de sinais (gestuais) são as verdadeiras línguas maternas dos surdos - e não as línguas faladas (orais) nas comunidades ouvintes em que eles vivem. Como conseqüência das particularidades das línguas de sinais, os surdos constituem comunidades culturalmente diferenciadas dentro daquelas comunidades ouvintes em que estão inseridas, o que sempre deveria ser levado em conta na condução da sua educação.

Esta mesma dificuldade pode ser observada no desenvolvimento de software em geral. As características especiais do usuário surdo não são consideradas na concepção de um novo software [4,5]. Isso gera uma escassez de aplicativos para o uso desse público especial. Com isso, pessoas surdas são forçadas a se adaptar a uma linguagem que não é a sua linguagem natural, e que possuem muita dificuldade para entender e utilizar.

São raros os programas que utilizam a linguagem da comunidade surda. O *SignWriter* [7, 8] é um software para edição de textos em língua de sinais, baseado no sistema de escrita de línguas de sinais *SignWriting* [9]. Entretanto, esse software é muito antigo, pode ser executado somente em MS-DOS, não possui suporte a mouse, somente apresenta textos em preto e branco e possui poucos recursos visuais.

Este artigo descreve o desenvolvimento de um conjunto de ferramentas (S*WEdit*, *AlfaEdit*), concebidos especialmente para o usuário surdo, que podem ser executados nos sistemas Windows95/98/ME/XP/NT/2000, UNIX e MAC OS. Esses sistemas utilizam a biblioteca gráfica wxWindows [10] e não fazem uso de chamadas de sistema (*System Calls*) e de recursos dependentes de sistema. Com relação à interface, procuram fazer o melhor uso de formas auto-explicativas e de figuras onde normalmente seriam usados textos.

Este artigo está organizado da seguinte forma. Na seção 2, apresenta-se o sistema *SignWriting*. Na seção 3, descrevem-se as características especiais apresentadas pelo o sistema *SWEdit*, como um sistema voltado para utilização do usuário surdo. O editor de símbolos *AlfaEdit* é introduzido na seção 4. O editor de sinais *SWEdit* é descrito na seção 5. A seção 6 apresenta o dicionário de sinais. A interface é apresentada na seção 7. A seção 8 traz as conclusões do trabalho e as perspectivas de trabalhos futuros.

## *2.* O Sistema de Escrita de Línguas de Sinais *SignWriting*

O *SignWriting* [9] é um sistema de escrita de línguas de sinais desenvolvido por Valerie Sutton em 1975, que consiste em uma representação gráfica da forma gestual da linguagem de sinais[2]. O sistema oferece uma notação prática para a escrita de línguas de sinais. Ele é um sistema notacional de características gráficas e esquemáticas, constituído de um rico repertório de elementos para a representação dos principais aspectos gestuais das línguas de sinais (configuração de mãos, pontos de articulação, movimentos, expressões faciais, etc.) e de fácil aprendizagem.

O sistema de escrita *SignWriting* se organiza de forma similar à forma de escrita oral. Ao invés de possuir letras, existem símbolos (veja a Figura 1), os quais individualmente representam um gesto ou movimento na língua de sinais dos surdos. Um conjunto desses símbolos forma um sinal (veja a Figura 2), significando uma palavra ou conjunto de palavras.

Figura 1 - Exemplos de símbolos do sistema *SignWriting*

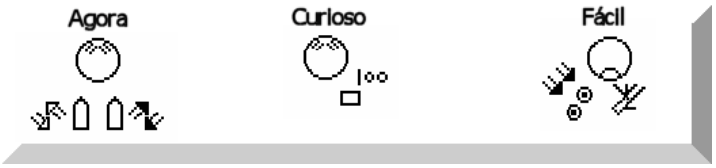


Figura 2 - Exemplos de sinais escritos no sistema *SignWriting*

## 3. *SWEdit* – um Sistema Desenvolvido Especialmente para Usuários Surdos

O sistema desenvolvido *SWEdit* tem, como principal funcionalidade, a edição de textos em línguas de sinais, baseado no sistema de escrita *SignWriting*. Permite também a inclusão de textos em linguagem oral, figuras e imagens, *drag & drop* entre diferentes programas, salvar e carregar arquivos no formato SWML[3] (SignWriting Markup Language [1,2]). Apresenta uma base de dados expansível e inclui dicionários.

A interface do *SWEdit* é projetada especialmente para a utilização por pessoas surdas, explorando a capacidade de interpretação visual que essas pessoas possuem, evitando que as mesas tenham que interpretar textos escritos em linguagem oral [4,5]. Em muitos casos, textos foram substituídos por figuras ou imagens, para representar uma determinada ação ou ferramenta no editor. Com isso, explorou-se uma qualidade sobre uma dificuldade. Essa qualidade foi explorada utilizando-se figuras auto-explicativas, que aparecem em botões e menus, para representar ações e ferramentas disponíveis no programa.

Além disso, a interface e as ferramentas são similares a dos editores de texto comumente utilizados. Isto torna a interface bem mais amigável, pois mesmo tendo sido projetada para pessoas surdas, um ouvinte pode aprender a utilizá-la apenas interpretando as funcionalidades similares a outros editores, o que se aplica também aos surdos.

Para implementação do *SWEdit*, foi escolhida a linguagem C++ por se tratar de uma linguagem orientada a objetos, o que se assemelha com a forma que a linguagem de sinais é formada. Isso possibilita que o programa tenha um código fonte reutilizável, característica essa que foi utilizada na criação do *AlfaEdit* (veja seção 4), que utiliza muito do código fonte proveniente do *SWEdit*. O compilador escolhido foi o Microsoft Visual C++ 6.0 por se tratar de uma plataforma estável e possuir muitas ferramentas para o auxilio na programação.

Utilizou-se também a biblioteca gráfica *wxWindows* [10], que fornece métodos para criação de interfaces, utilização de protocolos de rede, manipulação de imagens, protocolos de impressão, manipulação de arquivos, etc., de forma portável. Essa biblioteca possui código fonte aberto e programas gerados com ela podem ser compilados em diversos sistemas, dentre eles estão Windows95/98/ME/NT/2000/XP, UNIX e MAC OS.

Neste trabalho, foi empregada a aplicação XML chamada SWML (SignWriting Markup Language), que foi criada por Costa [1,2] com o intuito de ser um formato a ser utilizado por sistemas que utilizam linguagem de sinais. Com a SWML é possível o intercambio de documentos entre diferentes programas e a análise de textos independentemente do editor. Também pode servir como um formato de armazenamento de textos. A biblioteca Xerces C++[4] foi escolhida para manipulação dos dados em XML, por ser portável, ser de código fonte aberto e ser amplamente utilizada, o que facilita o seu aprendizado.

## *4.* O Editor de Símbolos *AlfaEdit*

O *AlfaEdit* consiste de um editor de símbolos - a forma básica dos sinais em língua de sinais escrita. Um símbolo pode ser obtido pela inclusão de figuras nos formatos GIF, PNG, JPG, BMP ou TIFF, ou pode ser criado utilizando-se diretamente o *AlfaEdit*. Ao incluir uma figura, também serão fornecidas as informações relativas às características que o símbolo terá, tais como: seu código (para a sua identificação dentro das tabelas de símbolos e no arquivo SWN gerado pelo *SWEdit*), rotação (informa se pode ser rotacionado), etc. Essas características, juntamente com a figura, formam um símbolo, que será gravado juntamente com outros em um arquivo do formato SSS (SignSymSequence), que será utilizado pelo editor para fornecer os símbolos empregados para a edição dos textos.

A Figura 3 representa a interface principal do *AlfaEdit.* Os números incluídos na figura indicam os seguintes itens da interface:

1. Área de visualização e seleção de símbolos;
2. Área de navegação pela base de dados;
3. Área de edição de imagem;
4. Ferramentas de edição de imagem;
5. Área de edição de propriedade do símbolo;
6. Barra de ferramentas relevantes ao símbolo que está sendo editado, apresentando os seguintes métodos: “Novo Símbolo”, “Salvar Símbolo”, “Salvar Novo Símbolo”, “Procurar Símbolo” e “Apagar Símbolo”;

---

[2] Veja outras informações em http://www.sigwriting.org.

[3] Mais informações sobre SWML em http://swml.ucpel.tche.br

[4] Mais informações sobre a biblioteca Xerces C++ em http://xml.apache.org

7. A barra de ferramentas relevante a todo o conjunto de símbolos;

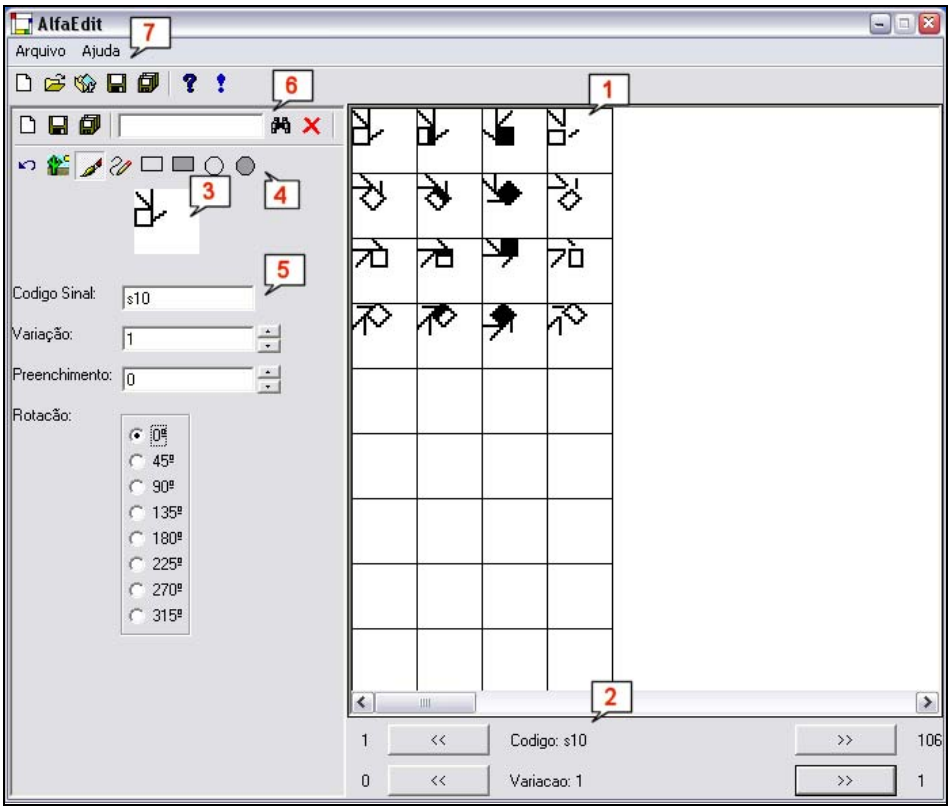


Figura 3 - Interface *AlfaEdit*

Um conjunto de símbolo criado no *AlfaEdit* é denominado de "alfabeto". Após a sua criação, o alfabeto é salvo em um arquivo do formato SSS. Para utilizá-lo no editor de sinais *SWEdit*, basta que esse arquivo seja copiado para o diretório "/SSS". Quando o *SWEdit* for iniciado esse novo conjunto de símbolos ficará disponibilizado ao usuário.

## 5. O Editor de Sinais *SWEdit*

### 5.1. Visão geral do Editor do Ponto de Vista do Usuário

Pela visão do usuário, o *SWEdit* se comporta como um editor de textos comum, com a diferença que está centrado na escrita da linguagem de sinais e não na escrita da linguagem oral. É possível trabalhar com texto da linguagem oral da mesma forma que se utilizam caixas de textos, como no MS Word™, por exemplo. É possível escolher a fonte, a cor, o tamanho e o estilo, permitindo a inclusão de textos em alfabeto normal, usando as fontes True Type[5] *SignWriting*. Também é possível a inserção de figuras, em diversos formatos, tais como JPGE, BMP, PNG e outros.

O foco principal do *SWEdit* são os símbolos. Estes estão dispostos em grupos criados pelo *AlfaEdit*. Os símbolos possuem funcionamento similar às figuras, sendo movidos em uma tabela. Possuem propriedades tais como cor, *flop* (espelhamento no eixo x), *flip* (espelhamento no eixo y), etc. Outras funcionalidades normalmente observadas em editores de textos também estão disponíveis, tais como: "Copiar", "Recortar", "Colar", "Visualização de Impressão e Impressão", "Paginação", "Dicionário", "Menus sensíveis ao contexto", etc.

Os textos gerados podem ser salvos em diversos formatos, tais como SWN, SWML e vários formatos de imagem.

Talvez a maior diferença para um editor comum seja a ausência do ponteiro de posição de texto, pois já que o símbolo se comporta como uma figura sendo movida em uma tabela, a interação principal do editor é similar a um editor de imagens, com o foco principal na interação com o mouse.

[5] Mais informações sobre fontes *SignWriting* consulte http://www.signwriting.org/catalog/sw214.html

## 5.2. Visão geral do Editor do Ponto de Vista do Programador

Do ponto de vista do programador, o sistema funciona como uma base de dados, onde objetos são movidos de uma base para outra e/ou suas propriedades são alteradas. Por exemplo, na área de edição de textos, as operações são referentes a uma lista com células, as quais possuem as suas próprias bases.

Uma operação de mudança de atributo geralmente passa por mais de um contexto até chegar ao seu destino. Por exemplo, para uma mudança de cor, uma área deve estar selecionada na área de edição. Os índices dos objetos que se encontrarem nessa área serão então passados para as suas células pais, que, juntamente com o índice, recebem a cor que deve ser aplicada. Esta será passada para os objetos referentes aos índices e somente então os próprios objetos selecionados recebem a cor. Essa ordem de eventos é a ordem básica utilizada pelo programa para a grande maioria das operações.

Outra diferença do ponto de vista do programador para o ponto de vista do usuário é que os objetos quando movidos são passados para um buffer, deletados e a partir desse buffer um novo objeto é criado. Do ponto de vista do usuário, esse objeto movido é o mesmo, apenas com sua posição modificada, mas na realidade ele é um novo objeto. Essa operação foi feita dessa forma para a reutilização do código das operações de copiar e colar, pois o funcionamento é o mesmo do ponto de vista do usuário.

## 5.3. Elementos do Documento

### *5.3.1. Célula*

A célula é uma das classes mais importantes do sistema, por ser onde símbolos, textos e figuras são inseridos e manipulados. Um documento é a união de uma ou varias células. Cada célula é responsável por uma área específica do documento. Essa área pode variar conforme a vontade do usuário, através de interação com o mouse.

A responsabilidade principal da célula é gerenciar os objetos do documento para a sua área de atuação e também responder a eventos vindos do usuário ou do sistema, os quais podem ser em relação a si mesmo ou a um dos componentes do documento pelas quais ela é responsável. Cada célula é um dos itens da tabela que compõem o documento, como mostra a Figura 4.

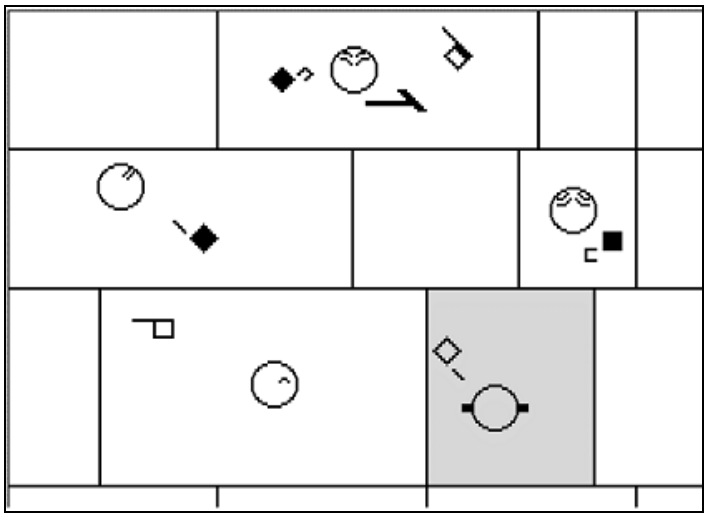

Figura 4 – Exemplo de células

A célula funciona basicamente como um repositório de componentes do documento para uma área específica. Para isso ela possui três listas, uma lista para objetos da classe `Símbolo`, uma para os objetos provenientes da classe `Texto` e outra para a classe `Figura`, que são os três tipos de componentes suportados pelo editor para um documento. Os objetos da classe `Célula` são armazenados na classe `CanvasEdit`, que é responsável pela área visível do documento.

Quando o usuário clica no documento, um método de `CanvasEdit` procura na lista de células uma que esteja naquela área específica. Se for encontrada, a posição do mouse é passada para um método da célula, que procurará em suas lista se existe um símbolo, um texto ou uma figura sobre a posição do mouse e tomará as ações necessárias baseadas no estado do sistema. Ela também é responsável por identificar objetos que estejam dentro de uma área de seleção e inseri-los em um buffer, para serem passados para a área de transferência, ou mudar suas propriedades, tais como a cor, por exemplo.

Em relação ao tamanho, uma célula só pode ser dimensionada através dos seus limite direito e limite inferior, pois os outros dois limites restantes podem ser dimensionados através das laterais das células vizinhas. Isso por que não pode haver espaços não ocupados entre as células. Se não existir células vizinhas, a célula em questão não pode ser re-dimensionada, por estar no limite esquerdo do documento.

Outra razão para que a célula só possa ser dimensionada por apenas dois dos seus vetores é a célula vizinha estar no seu tamanho mínimo permitido. No momento em que uma célula é redimensionada, a responsabilidade de informar a essa célula que seu tamanho não pode mais ser alterado é de `CanvasEdit`, pois cada célula é responsável somente por si e seus elementos, desconhecendo a existência de outras células, e até mesmo do documento.

Dessa mesma forma, a classe `CanvasEdit` só é responsável por objetos da classe célula, não sendo responsável por objetos da classe `Símbolo`, `Texto` e `Figura`, que são da responsabilidade da classe `Célula`.

### *5.3.2. Símbolo*

Essa classe representa o que é um símbolo, como explicado nas seções anteriores. As informações mantidas na classe `Símbolo` são:

- Código: indica qual a bitmap é utilizada para mostrar o símbolo; essa bitmap é retirada da classe `Alfabeto` abordada a seguir.
- Cor: armazena a cor do símbolo.
- Flip: informa se o símbolo foi espelhado no eixo x.
- Flop: informa se o símbolo foi espelhado no eixo y.
- Retângulo: tem como coordenadas X e Y superiores à posição do símbolo na célula; a largura e altura são a área ocupada pelo bitmap do símbolo; o retângulo é usado para saber se um click do mouse foi sobre o símbolo.

O método mais importante dessa classe é o método para transferir todos os dados para um buffer. Através desse método o símbolo é copiado para a área de transferência bem como movido pelo documento e também utilizado quando o documento é salvo em arquivo.

### *5.3.3. Texto*

Essa classe representa uma caixa de texto em linguagem oral. O funcionamento dos objetos dessa classe é muito similar ao de uma caixa de texto de um editor de texto comum, podendo-se escolher a fonte, cor, tamanho da fonte, negrito, etc. Essas operações são possíveis através de menus sensíveis ao contexto bem como a click duplo sobre o texto, que abre uma janela padrão do sistema operacional para seleção de fonte, cor e outras propriedades. Essa classe possui os seguintes atributos: texto (uma string de tamanho fixo), fonte, cor e retângulo (posição do texto e área que ele ocupa).

Como em `Símbolo`, o método mais relevante é o que transforma os dados da classe em buffer para movimentação através da área de transferência, pois através desse método a classe gerenciadora pode movimentar o objeto dessa classe através do documento, bem como enviá-lo para outro programa ou salvar em arquivo.

Outros métodos disponíveis são os de mudança de atributos, de acesso aos atributos da classe, de desenho e os utilizados para testar posição relativa ao retângulo usado na verificação do click do mouse.

### *5.3.4. Figura*

Representa uma figura inserida no documento, que pode estar em um dos seguintes formatos: BMP, JPG, TIFF, PNG, PCX e GIF. Essa classe comporta-se de modo similar a uma figura que é inserida num editor de texto comum, com a diferença de estar ancorada a uma célula, da mesma forma que um símbolo e um texto. Possui implementação similar às duas classes anteriores. Os atributos dessa classe são apenas dois, Bitmap que representa a figura em si, e sua posição. Como nas duas classes anteriores o método mais relevante é o método de inserção das informações da bitmap em um buffer.

## 5.4. CanvasSign

O objetivo dessa classe é agrupar um conjunto de Símbolos, como mostrado na Figura 5, para facilitar a busca ou procura de símbolos. A área relevante a `CanvasSign` se restringe à área em branco na Figura 5. Os outros itens não são parte de `CanvasSign`, são componentes da interface.

Essa classe tem a responsabilidade de manter um conjunto de objetos da classe `Símbolo`, exibí-los e disponibilizá-los para o usuário.

Quando um objeto é inicializado, recebe um identificador de grupo, o qual é passado para o objeto global de uma

classe denominada `Alfabeto`, que procurará uma lista de `Signs` que possuam o mesmo identificador de grupo passado. Após, a lista de `CanvasSign` é "populada" através da criação de objetos da classe `Símbolo`, baseados nos objetos da classe `Sign` que se encontram na lista de `Alfabeto`. Concluído esse processo, `CanvasSign` irá exibir os símbolos na sua área visível e disponibilizá-los para o usuário através de dois métodos: "click duplo" sobre um símbolo ou "arrastar" e "colar".

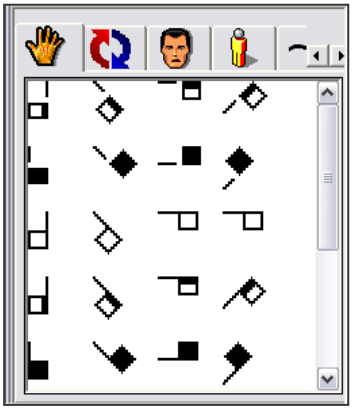

Figura 5 - `CanvasSign`

### 5.5. CanvasEdit

A classe `CanvasEdit` é a classe mais importante dentre os três "canvas", pois essa representa a área de edição do documento. É essa classe que tem a responsabilidade de manter o documento e processar as interações do usuário com os seus vários componentes. Ela também é responsável pela organização dos objetos da classe célula. `CanvasEdit` deve manter as células organizadas de tal forma que todo o documento pareça uma grande tabela, sem espaços não ocupados entre as células. Outra responsabilidade organizacional referente às células é mantê-las dentro da área da página.

`CanvasEdit` também funciona como editor de sinais quando utilizada para edição de sinais que serão enviados para o dicionário. Isso é possível apenas mudando uma variável que indica em que modo ela está atualmente. Nesse modo, ela apenas aceita objetos da classe `Símbolo`, mantendo todas as outras funcionalidades.

Os documentos podem ter um número indeterminado de páginas. O limite é principalmente determinado pelo tamanho da memória da máquina onde o editor esteja sendo executado. Várias são as razões para a utilização de páginas, razões que vão desde facilitar a compreensão do texto até a velocidade na exibição do conteúdo na tela.

O sistema de paginação é relativamente simples. `CanvasEdit` possui uma lista onde todas as células do documento são armazenadas. Cada célula possui um atributo que indica a sua página e a linha que se encontra. A lista é organizada por esses fatores, primeiro por página, seguido por linha e por último por coluna. Com isso se tem a vantagem de se processar apenas as células que se encontram na página atual, que se aplica em muitos casos.

## 6. O Dicionário

O dicionário é umas das ferramentas mais importantes do sistema, pois acelera a criação de textos, facilita a leitura do documento, gerando sinais com melhor formatação no documento. Também possibilitam que um ouvinte possa identificar um sinal e saber o seu significado bem como uma pessoa surda que esteja aprendendo *SignWriting*.

Um dicionário possui as seguintes propriedades: Nome (esse atributo também é o nome do arquivo no formato SWD que foi criado), Autor, Descrição, Língua Oral, Língua de Sinais. Esses atributos podem ser vistos na Figura 6, que mostra a janela de edição de dicionário.

O usuário tem a possibilidade de trabalhar com vários dicionários, o que possibilita a utilização de dicionários em diferentes línguas ao mesmo tempo. Todos os dicionário criados são armazenados no diretório "SWEdit/DIC", são carregados na inicialização do sistema e salvos no encerramento do programa.

Como os dicionários são salvos em arquivos separados do sistema, isso possibilita que eles sejam disponibilizados na Web, por exemplo, e possam ser compartilhados com outros usuários.

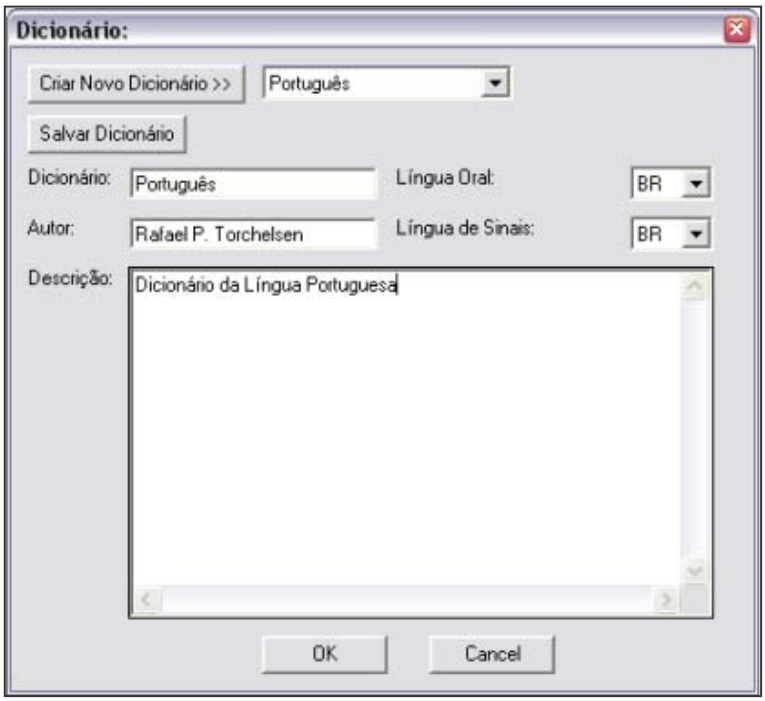


Figura 6 – Janela de edição de dicionário

### 6.1. CanvasDic

A classe `CanvasDic` tem a responsabilidade de gerenciar e exibir os dicionários que estejam sendo utilizados no momento, como é visto na Figura 7.


Figura 7 – Área de visualização de sinais

Os sinais do dicionário que está sendo utilizado no momento serão exibidos em ordem alfabética. Exibindo o sinal e seu significado em língua oral, essas palavras podem ser inseridas no documento atual através de click duplo ou por menu, como mostrado na Figura 7.

Os sinais também podem ser editados numa área especial para edição de sinais, mostrada na Figura 8. Essa área é um objeto da classe `CanvasEdit` em modo palavra. Temos também opções para enviar o sinal para o documento ou para o dicionário atual.

Devido aos dicionários possuírem geralmente um número muito grande de sinais, foi necessário incluir ferramentas de procura. Para isso temos a opção de procura de sinal por língua oral e por língua de sinais.

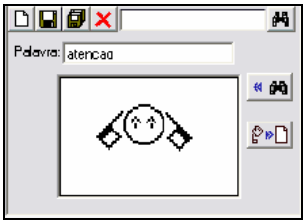


Figura 8 – Área de edição de sinal

## 7. A Interface

A Figura 9 mostra a interface do editor, onde as seguintes características podem ser observadas:

1. Tabs contendo os conjuntos de Símbolos;
2. O conjunto de Símbolos da tab atual;
3. Área de edição de Sinal;
4. Área de edição do documento;
5. Uma Célula, contendo exemplos de Símbolos;
6. Exemplo de inserção de texto na forma oral;
7. Exemplo de inserção de uma figura;
8. Menu sensível ao contexto;
9. ComboBox contendo os dicionários disponíveis;
10. Botões para navegação nas paginas.

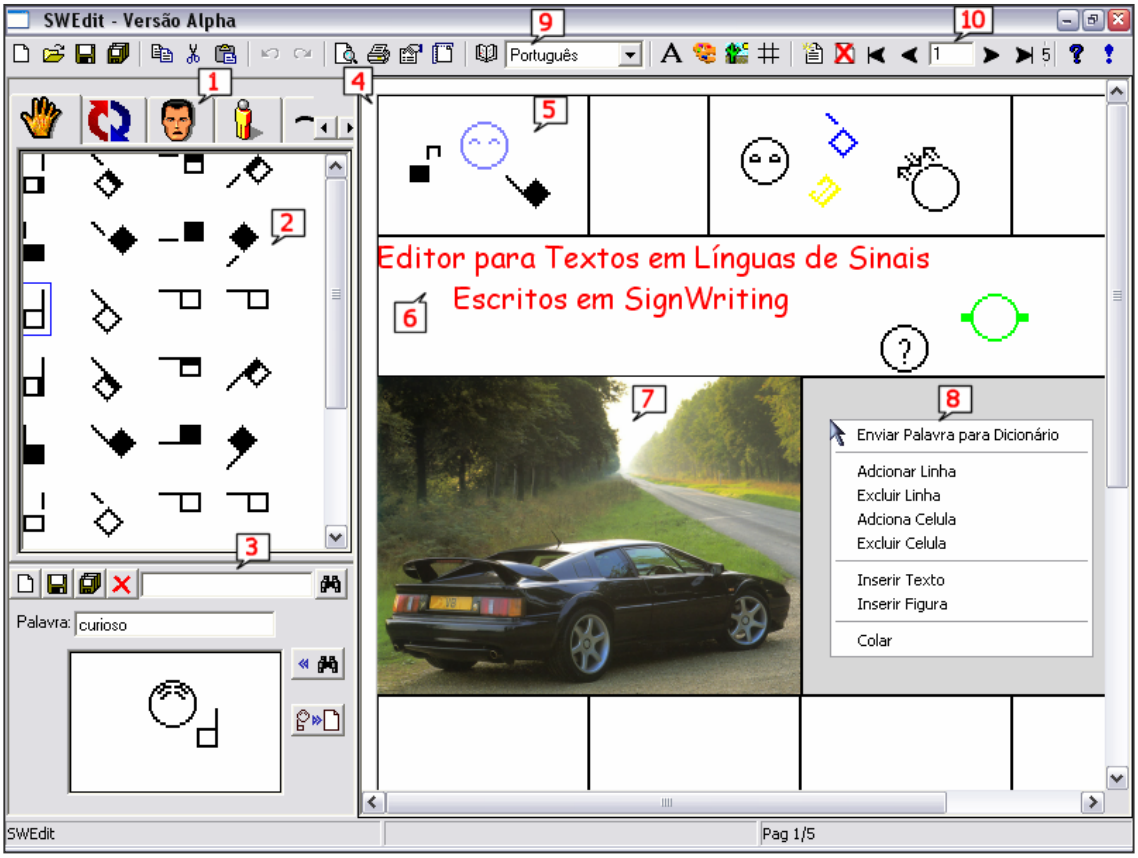


Figura 9 – Interface do SWEdit

## 8. Conclusões

Este artigo descreveu o desenvolvimento de ferramentas planejadas e implementadas especialmente para usuários surdos, com o objetivo de auxiliar esses usuários na confecção de textos na sua língua natural, que é a língua de sinais. Foram apresentados o editor de sinais *SWEdit*, baseado no sistema de escrita de sinais *SingWriting*, e o editor de símbolos que compõem os sinais – o *AlfaEdit.*

A escolha da biblioteca gráfica wxWindows para o desenvolvimentos dessas ferramentas trouxe muitas facilidades, principalmente por disponibilizar muitas funcionalidades, que foram amplamente utilizadas, embora seja necessário ressaltar a carência de bibliografia e documentação sobre essa biblioteca.

Um desafio deste trabalho foi o gerenciamento dos objetos criados no decorrer da execução do sistema, pois já na inicialização são criados aproximadamente quatro mil objetos, na sua maioria objetos da classe `Símbolo`.

Para possibilitar expansões, os módulos suportam a inclusão de novas funcionalidades. Como exemplo disso, cita-se o Dicionário, o qual não havia sido modelado inicialmente.

A interface constitui um dos fatores diferenciadores destas ferramentas. Foi projetada especificamente para o público alvo, levando-se em consideração as suas dificuldades e qualidades. Como o objetivo de limitar o uso da língua oral, explorou-se a facilidade visual dos usuários surdos, expressando as funcionalidades por figuras. Para evitar que a interface ficasse saturada de informações, as funcionalidades exibidas ao usuário foram limitadas, e funcionalidades similares agrupadas. A interface também utiliza técnicas de redimensionamento automático, o que possibilita que usuários que utilizem resoluções superiores a 800 x 600 pixels disponham de mais área útil.

Esta é a primeira versão do software. Algumas funcionalidades ainda deverão ser desenvolvidas, como as que serão explicadas a seguir.

### 8.1. Algoritmo de Transformação de Símbolos

Na maioria dos casos os símbolos possuem transformações idênticas, mas em muitos casos existem transformações específicas para um determinado símbolo. Essas transformações nem sempre se encontram em seu código que informa sua variação, rotação e preenchimento. Para que essa tarefa seja realizada é necessária a criação de um algoritmo que identifique o símbolo e informe as suas transformações possíveis, o que possibilitaria a implementação de uma função para passar todas as funções de um determinado símbolo.

No sistema atual, muitos símbolos podem ser rotacionados em direções sem significado. Entretanto, essa característica não chega a ser uma limitação, pois o usuário conhecedor do sistema *SignWriting* sabe quais as transformações um determinado símbolo possui, e quais não podem ser realizadas.

### 8.2. Adição de novos algoritmos de procura

Foi implementada uma funcionalidade de procura por sinais que apenas procura por sinais que possuem o mesmo número de símbolos informados. Isso gera um problema, pois se o usuário não possuir o conhecimento de todos os símbolos que o sinal possui ele não terá sucesso na sua procura. A implementação de novos algoritmos de procura seria muito útil, tais como a procura por semelhança, por exemplo. Uma proposta para essa relação de semelhança entre sinais pode ser encontrada em http://sw-journal.ucpel.tche.br/number0/article1/index.htm.

### 8.3. Adaptação para o Sistema DancingWriting e SportWriting

O sistema *DancingWriting* e *SportWriting* também foram desenvolvidos por Valerie Sutton os quais descrevem passos de dança e movimentos realizados em esportes, respectivamente, através de símbolos gráficos [6].

Uma adaptação para esses sistemas envolveria a inclusão de uma nova classe, talvez derivada da classe `Símbolo`. As alterações necessárias não são muito significativas, pois esses sistemas se assemelham ao *SignWriting,* pois todos esses sistemas fazem parte do *MovementWriting* (veja http://www.movementwriting.org). Essa adaptação seria útil para professores e estudantes na especificação de movimentos na dança ou esporte.

## Agradecimentos

Os autores agradecem a participação e colaboração dos vários bolsistas surdos vinculados ao Projeto SignNet – Adaptando as Tecnologias da Internet às Línguas de Sinais e à Educação dos Surdos, em desenvolvimento na Escola de Informática da UCPel. Em especial, agradecemos a colaboração e valiosas sugestões de Valerie Sutton, coordenadora da Deaf Action Committee for SingWriting. Agradecemos também ao CNPq e à FAPERGS pelo apoio financeiro recebido.

## Referências


[1] Costa, A. C. R. and Dimuro, G. P. SignWriting-Based Sign Language Processing. In: International Gesture Workshop, 2001, London. *Gesture and Sign Language in Human-Computer Interaction, Lecture Notes in Artificial Intelligence*, Berlin, Springer-Verlag, 2002, n. 2298, pp. 202-205

[2] Costa, A. C. R. and Dimuro, G. P. Supporting Deaf Sign Languages in Written Form on the Web. In: 10th World Wide Web Conference, 2001, Hong Kong. *Home Page of Web and Society Track*. Hong Kong: CD&E Dept, The Chinese University of Hong Kong, 2001. (Available in http://www10.org/cdrom/posters/frame.html).

[3] Cuxac, C. Le Pouvoir des Signes. In: *Sourds et Citoyens*. Institut National de Jeunes Sourds de Paris, Paris, 1990. pp. 99-110.

[4] Pontes, A. M. Estudo da Percepção de Signos por Sujeitos Inseridos em Diferentes Meios Culturais: um passo inicial para o desenvolvimento de uma interface voltada a usuários surdos. Porto Alegre: PPGCC/PUCRS, 2000. (dissertação de mestrado)

[5] Pontes, A. M. e Orth, A. I. Proposta de Linguagem de Interação para Interfaces Voltadas a Usuários Surdos. In: Seminário Integrado de Software e Hardware, XXI Congresso da Sociedade Brasileira de Computação, Fortaleza, 2001. *Anais* ... Porto Alegre: SBC, 2001.

[6] Sutton, V. Sutton *Movement Shorthand: a Quick Visual Easy-to-Learn Method of Recording Dance Movement* - Book One: The Classical Ballet Key. The Movement Shorthand Society, Irvine. 1973. (http://www.dancewriting.org/)

[7] Sutton, V. *SignWriter Manual*. La Jolla: Deaf Action Committee for SingWriting. 1995.

[8] Sutton, V. and Gleaves, R. *SignWriter – The World's First Sing Language Processor*. La Jolla: Deaf Action Committee for SingWriting. 1995

[9] Sutton, V. *Lessons in SignWriting – Textbook and Workbook.* Second Edition, La Jolla: Deaf Action Committee for SingWriting. 1999.

[10] wxWindows. *Cross-Platform GUI Library*. (available in http://www.wxwindows.org, accessed in July, 3rd)