Criação Interactiva de Banda Desenhada

Ricardo Abreu Lopes Manuel J. Fonseca
Dep. Eng. Informática, IST / INESC-ID
R. Alves Redol, 9, 1000-029 Lisboa
rval@ist.utl.pt, mjf@inesc-id.pt

Tiago D. Cardoso Nelson F. Silva
inEvo, I & D
Lisboa
tiago.cardoso@inevo.pt, nelson.silva@inevo.pt

Resumo

A criação interactiva de Banda Desenhada (ou comics), via plataformas na Internet, tem vindo a destacar-se nos últimos anos. Contudo, a variabilidade visual e os mecanismos de reutilização de símbolos que estas plataformas oferecem, revelam-se desajustados da criação tradicional de Banda Desenhada, em formato físico. A solução que propomos tem como objectivo aproximar estes dois universos, permitindo que uma interacção mais rica resulte numa melhor expressividade gráfica. Com a aplicação que aqui apresentamos, o utilizador pode desenhar livremente e aproveitar essa interacção caligráfica para recuperar e reutilizar desenhos criados anteriormente.

Palavras-Chave

Criação de Banda Desenhada, Comics, Desenho Livre, Recuperação Caligráfica

1. INTRODUÇÃO

As ferramentas que existem actualmente para a criação de Banda Desenhada (doravante BD) em formato digital dividem-se entre o *software* de suporte local e as aplicações *web*. Esta última alternativa tem-se destacado como a mais aceite para a criação de BD, num contexto amador.

Verifica-se este comportamento porque a crescente popularidade da Internet tem permitido a ascensão de um novo tipo de criadores anónimos de BD. Estes criadores utilizam esta tecnologia como o seu único meio de produção e distribuição, imediata e gratuita perante uma audiência global de leitores. O acesso a plataformas para a criação de BD que são livres, comuns e ubíquas, tem acentuado o carácter interactivo deste processo e da própria relação entre este tipo de autores e os leitores.

Apesar destas vantagens, as aplicações *web* para a criação de BD ainda apresentam algumas limitações no suporte que oferecem para a criação completamente livre e para a reutilização de símbolos visuais.

Para identificar o problema e estudar possíveis abordagens de solução, realizámos uma análise ao trabalho relacionado nos campos de estudo relevantes: técnicas de suporte à criação de *comics*, quer em formato físico, quer em formato digital. Com a pesquisa realizada constatámos que existem convenções tácitas que são utilizadas globalmente por diferentes artistas de BD e que nos permitem especificar um vocabulário próprio da linguagem visual de *comics* [McCloud93]. Esse vocabulário demonstra o uso recorrente de uma simbologia comum. Concluímos ainda que a recombinação de imagens predefinidas, retiradas de um conjunto fixo e diminuto, é a técnica mais utilizada para a criação de *comics online*[Williams05].

Deste modo, podemos formalizar o problema observado: os sistemas para a criação *online* de BD falham na reprodução da variabilidade gráfica inerente à criatividade humana e provocam uma sobrecarga de informação na tarefa de selecção de elementos visuais a reutilizar.

Podemos inferir que uma aplicação que garanta a liberdade do desenho livre, associada à reutilização de elementos visuais, é uma inovação na criação interactiva de BD. Neste documento, propomos e descrevemos a arquitectura de uma aplicação *web* que permite o desenho livre de BD e suporta a recuperação dos elementos visuais criados anteriormente.

2. ARQUITECTURA

A Figura 1 representa a arquitectura geral do sistema proposto. Como se pode observar, optámos por estruturar o sistema em dois módulos: o editor e o recuperador. O primeiro é responsável pela interacção com o utilizador e permite que este crie BD utilizando um conjunto de operações oferecidas pela interface. O segundo módulo suporta a recuperação de elementos visuais guardados num repositório de desenhos anteriores. Estes dois componentes respeitam uma arquitectura cliente-servidor, em que o módulo editor age como o cliente que requer um serviço ao módulo recuperador, que assume o papel de servidor.

O editor é uma aplicação *web* desenvolvida na plataforma Adobe Flex. Como se pode constatar pela Figura 1, a arquitectura proposta permite que, no contexto do módulo editor, exista uma independência entre a interface de utilizador e a lógica das operações de edição. Tomámos a opção anterior tendo como motivação a vontade de manter o editor como um módulo extensível e reutilizável. Deste modo pensamos estar a potenciar o sucesso da sua

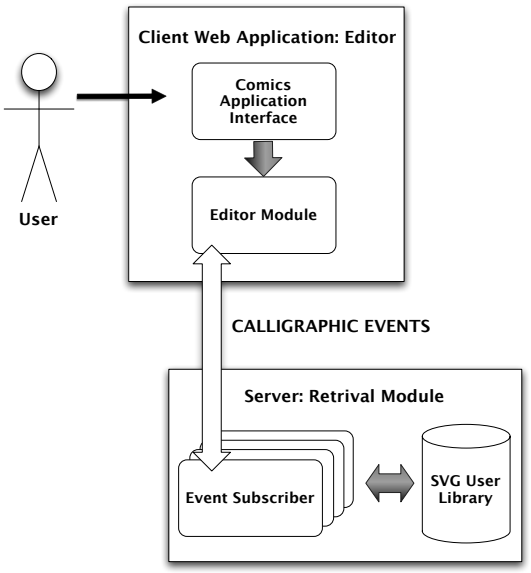

Figura 1. Arquitectura Geral

utilização.

O módulo editor consiste num conjunto de componentes individuais, que podem ser instanciados e configurados de modo a criar uma aplicação e respectiva interface. Criámos componentes que suportam o conjunto de operações necessárias à criação de BD: quadros de desenho livre e componentes de edição.

O módulo recuperador suporta, como o nome indica, a recuperação de elementos visuais já criados, garantindo-se a reutilização de desenhos anteriores. Para se atingir este objectivo, torna-se necessária a comunicação entre o módulo editor (aplicação cliente) e o módulo recuperador (aplicação servidor). Esta comunicação vai consistir em pedidos e respostas de recuperação de desenhos, a serem utilizados no editor.

O facto do editor ter como funcionalidade central o desenho livre foi um factor determinante para a escolha do tipo de recuperação a ser suportada pelo módulo servidor. A interacção utilizador-sistema numa aplicação de desenho tem como ponto de partida o esboço de traços. Considerámos que a melhor opção seria não introduzir um novo tipo de interacção que perturbasse o utilizador. Deste modo, a recuperação caligráfica foi a técnica eleita para permitir a reutilização de desenhos. Isto significa que os pedidos e respostas de recuperação vão ter a mesma estrutura: traços que compõem formas geométricas.

Conforme ilustrado na Figura 1, o módulo recuperador alimenta-se de acontecimentos caligráficos com origem no módulo editor. Estes acontecimentos não são mais que pedidos do utilizador que representam potenciais conteúdos a recuperar. Existem diferentes tipos de acontecimentos caligráficos, classificados consoante os objectivos do utilizador para os traços esboçados. Estes acontecimentos são observados por subscritores do módulo recuperador que actuam em conformidade. Para reconhecer os acontecimentos caligráficos, utilizamos o trabalho proposto por Fonseca *et al.*[Fonseca02].

Decidimos estender o modelo de recuperação proposto em [Fonseca02] com outras opções que julgamos serem mais flexíveis. A nossa contribuição será a definição do conceito de reconhecimento por atalhos caligráficos. Estes atalhos são escolhidos e definidos pelo próprio utilizador e apontam para os desenhos criados pelo mesmo. O utilizador é livre de criar abstracções mais simples dos desenhos originais ou outras associações que considere mais lógicas. Em relação à apresentação dos resultados da recuperação iremos usar uma fila de expectativas, de modo a prevenir situações de semelhança entre atalhos e a resolver ambiguidades.

3. INTERFACE DO EDITOR

Para atingir os objectivos propostos, decidimos que o desenho livre, a manipulação de desenhos e a edição em camadas visuais sobrepostas seriam os principais requisitos a suportar, do ponto de vista dos processos de criação de BD.

Estes requisitos resultaram num conjunto de operações que estão disponíveis através da interface da aplicação editor, como representada na Figura 2.

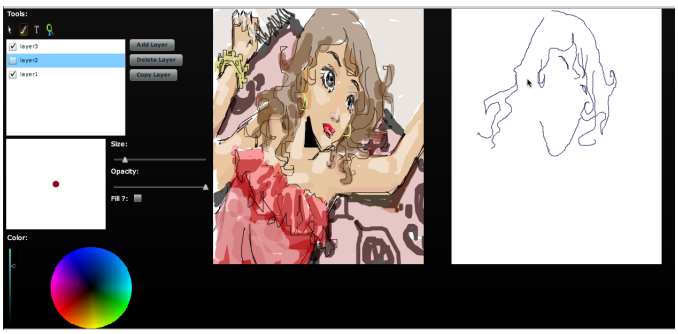

Figura 2. Exemplo de vinheta criada com a versão actual do nosso editor

Os desenhos são criados em telas individuais, via gestos com o rato ou dispositivo semelhante. Cada tela individual representa a vinheta de uma tira de BD. Estas telas são os componentes responsáveis pela visualização de desenhos criados ou recuperados.

Existe um conjunto adicional de componentes de interface que são responsáveis por distintas operações de edição. Denominámos estes componentes de ferramentas de edição. Estes componentes estão organizados num menu que permite alterar entre contextos de edição.

Na Figura 2 temos representado o contexto de desenho que disponibiliza os componentes que permitem alterar características do traço, como a cor, opacidade, espessura ou preenchimento. Neste contexto é ainda possível distribuir os desenhos por diferentes camadas visuais sobrepostas,

que podem ser removidas ou copiadas. É também possível alterar a visibilidade ou a ordem de sobreposição de cada camada.

No contexto de selecção, é possível seleccionar vários traços na tela de desenho activa e aplicar-lhe operações de transformação. Estas operações permitem, através de gestos com o rato, remover o traço seleccionado ou mesmo alterar a sua posição, orientação e dimensão. Neste contexto, existem ainda componentes para agrupar ou separar os traços individuais em conjuntos seleccionáveis.

No contexto de texto é possível inserir texto na tela activa, utilizando o teclado para tal e o rato para seleccionar a posição do mesmo. Existem componentes adicionais para seleccionar características do texto, como o tamanho ou o tipo de letra.

No contexto de zoom, é possível mudar a escala global da tela, utilizando o rato para este efeito ou mesmo para navegar na tela com a nova escala. Neste contexto, existe ainda um componente adicional que permite visualizar a tela na escala original.

4. ARQUITECTURA DO EDITOR

O módulo editor foi implementado seguindo o padrão de desenho *Model - View - Controller*, que separa a lógica da aplicação da sua interface. Este padrão de desenho é a melhor opção para conseguirmos gerir e criar operações independentes que têm origem num conjunto de acontecimentos sobre a mesma interface. A Figura 3 representa a arquitectura geral do módulo editor da solução que propomos neste documento.

Na Figura 3 podemos observar que a interacção com o utilizador tem como ponto de partida a tela de desenho (*Canvas*) e um conjunto de ferramentas de edição (*Tools*). Definimos estes objectos como componentes individuais que podem ser instanciados e configurados numa aplicação Flex, via ficheiro MXML.

A camada *View* da arquitectura ilustrada pela Figura 3 agrega estes componentes. Deste modo, esta camada funciona simultaneamente como ponto de entrada na interacção com o utilizador e ponto de saída na apresentação de resultados dessa interacção. Em relação à interacção com o utilizador, esta camada é responsável pela criação de objectos a serem interpretados na camada *Controller*. Denominámos estes objectos de acontecimentos de edição.

O processamento destes acontecimentos de edição, nas camadas *Controller* e *Model*, acaba por traduzir-se em objectos adicionais da camada *View*, que denominámos como vistas de traços. Cada gesto detectado na tela de desenho acaba eventualmente por resultar num objecto gráfico individual adicionado à tela de desenho. Estes objectos, além de serem a representação visual dos traços desenhados, são responsáveis por criar e despachar acontecimentos de edição para a manipulação das suas próprias características.

Como se pode constatar na Figura 3, os acontecimentos de

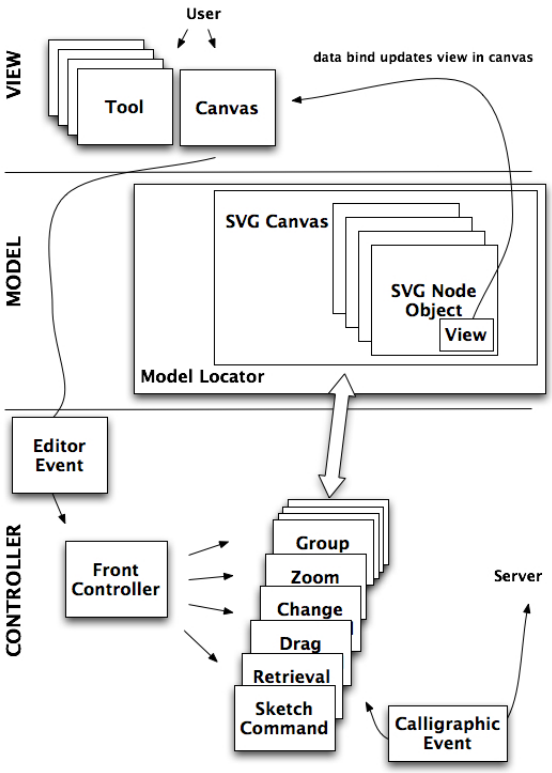

Figura 3. Arquitectura do módulo Editor

edição são transmitidos à camada *Controller* do módulo editor. O controlador é o objecto responsável por escutar os acontecimentos de edição com origem na camada *View*, elegendo e executando um comando adequado para tratar o respectivo acontecimento. Os comandos invocados pelo controlador são responsáveis por executar a lógica da respectiva operação. A lógica suportada pelo módulo editor pode dividir-se em dois tipos: a gestão interna do desenho e a recuperação de desenhos anteriores. Na recuperação, o comando irá comunicar um acontecimento caligráfico ao servidor. Na gestão interna do desenho, os respectivos comandos irão criar ou invocar alterações no modelo de dados que representa os desenhos criados / manipulados pelo utilizador.

O gestor do modelo de dados da camada *Model* guarda a representação interna do desenho actual, mantendo referências individuais para objectos que representam cada traço desenhado. Estas referências são organizadas para representar a totalidade do desenho. Optámos por manter sempre referências para cada traço para que estes possam ser rapidamente identificados. Deste modo, a manipulação de um traço individual é facilitada: as operações de criação, edição ou mesmo recuperação de cada traço conseguem ser feitas sem interferir no resto da globalidade do desenho.

Cada objecto do modelo de dados guarda a informação relativa à forma e características do traço desenhado e possui ainda a referência para a já abordada vista de traço. Esta vista é um objecto gráfico distinto que resulta da tradução

directa do que foi desenhado na tela de desenho. Conforme já explicámos, este objecto permanece visível quando é acrescentado à tela de desenho, na camada *View*.

Esta estrutura de dados permite afirmar que uma actualização ao modelo de dados da camada *Model* consiste na criação / alteração de um objecto individual e sua respectiva vista. O objecto individual é acrescentado / actualizado no modelo de dados (camada *Model*). A respectiva vista é acrescentada ou actualizada na tela de desenho (camada *View*). Tomámos esta decisão porque é necessário garantir simultaneamente dois aspectos. Por um lado, é necessária uma representação interna do desenho que guarde informação para a recuperação de desenhos anteriores ou a persistência do desenho actual. Por outro lado, é necessária uma representação gráfica do desenho que ofereça *feedback* ao utilizador do que este está a criar e lhe permita a interacção com o que já está desenhado.

Concluímos que a melhor opção para o modelo de dados do editor seria sincronizar a representação interna do desenho com o formato SVG, uma norma W3C para desenhos vectoriais que é utilizada pelo repositório de desenhos no servidor. Ao tratar desta sincronização no modelo de dados do editor, estamos a garantir que a comunicação entre este último e o módulo recuperador possa ser feita através de mensagens SVG. As vantagens ocorrem em ambos os lados. Do lado do recuperador, os acontecimentos caligráficos que são recebidos como pedidos de recuperação podem ser comparados com o repositório de desenhos e atalhos, sem necessidade de conversão de dados adicional. Do lado do editor, as respostas com o conteúdo recuperado consistem em dados SVG enviados pelo servidor. O modelo de dados do editor está sincronizado com o formato SVG e, como tal, sabe interpretar e desenhar imediatamente essas respostas, criando as respectivas vistas individuais necessárias.

5. CONCLUSÕES E TRABALHO FUTURO

A solução que propomos neste documento baseia-se em dois módulos distintos: o editor e o recuperador. Neste momento, o módulo editor encontra-se terminado a nível de implementação. O recuperador já tem a sua arquitectura definida, devidamente explicada na secção 2 deste documento.

Neste momento, apenas conseguimos avaliar os resultados associados ao editor. Podemos desde já concluir que o editor tem sucesso no objectivo de não limitar a criatividade visual normalmente associada à criação *online* de BD. Ao optar por uma interacção baseada em desenho livre, estamos a colocar o ónus da riqueza visual nas mãos do autor (ver Figura 2). As opções estudadas, como trabalho relacionado, normalmente colocam esse ónus do lado do programador da aplicação, que tem de disponibilizar um repositório de desenhos, passíveis de serem recombinados pelos autores (ver Figura 4). Concluímos que com uma interacção sem restrições, é possível chegar a resultados que emulam a expressividade natural do vocabulário BD.

Acerca do trabalho futuro, este pode ser analisado sob dois

Figura 4. sistema ComicKit: exemplo de criação de comics online

pontos de vista. Em primeiro lugar, ainda temos que concretizar a arquitectura proposta para o módulo recuperador. A implementação deste permitirá uma avaliação total à solução que aqui apresentamos.

O segundo ponto de vista do trabalho futuro está relacionado com mudanças arquitecturais que poderiam ser introduzidas nesta solução, de modo a melhorar os nossos resultados. Pensando numa melhoria da variabilidade visual que desejamos suportar e incentivar, poderíamos acrescentar ao módulo editor capacidades de processamento de imagem. Como exemplo, referimos a possibilidade de manipular características gerais de cor e iluminação, reproduzindo efeitos normalmente associados à edição de gráficos *bitmap*.

6. REFERÊNCIAS

[McCloud93] Scott McCloud, Understanding Comics. New York, NY: Kitchen Sink Press / Harper Perennial.

[Williams05] Williams, R., Barry, B., Singh, P., ComicKit: Acquiring Story Scripts Using Common Sense Feedback. *International Conference on Intelligent User Interfaces archive. Proceedings of the 10th international conference on Intelligent user interfaces*. pp. 302-304. San Diego, California, USA, 2005.

[Fonseca02] Manuel J. Fonseca, César Pimentel e Joaquim A. Jorge, CALI: An Online Scribbler Recognizer for Calligraphic Interfaces, *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, pp. 51-58, Palo Alto, USA, Mar 2002.