3
A Biblioteca Piccolo

O *Piccolo* é uma feramenta para o desenvolvimento de programas gráficos 2D em general, e interfaces de usuário com zoom - *Zoomable User Interfaces* (ZUIs), em particular. Uma ZUI é um tipo novo de interface que apresenta uma tela enorme de informação (chamada canvas) em uma visão tradicional do computador, possibilitando ao usuário mudar gradualmente o zoom de visão, aproximando para obter informação mais detalhada ou afastando para uma visão geral.

O *Piccolo* utiliza um modelo de "cena-gráfica" que é comum em ambientes 3D. Basicamente, isto significa que o *Piccolo* mantém uma estrutura hierárquica dos objetos e das câmeras, permitindo que o desenvolvedor de aplicação oriente, agrupe e manipule objetos de maneira significativa.

O *Piccolo* é uma camada construída sobre uma camada mais baixa de API gráfica. Existem três versões da ferramenta: Piccolo.Java, Piccolo.NET e PocketPiccolo.NET (para a estrutura compacta de .NET). A versão de Java é construída em Java 2 e se baseia no Java2D API para renderizar os gráficos. Já a versão .NET é construída na estrutura .NET e se baseia no GDI+API. Isto facilita para programadores de Java e de C#, mesmo aqueles que alvejam PDAs, para construir suas próprias aplicações gráficas animadas.

3.1
Visão geral da arquitetura Piccolo

Os objetos na tela (*canvas*) do piccolo são chamados de nós (*nodes*). Os nós são usados para representar os componentes discretos de uma interface. Alguns nós já são implementados pela ferramenta (formas geográficas, imagens, texto) e o desenvolvedor tem a oportunidade de definir novos nós para suas interfaces.

Todas as interfaces do *Piccolo* necessitam ser colocadas na *PCanvas* de modo que possam ser vistas e interagidas com pelo usuário. *PCanvas* estende a classe de javax.swing.JComponent em Piccolo.Java e a classe equivalente de

System.Windows.Forms.Control em Piccolo.NET. Em conseqüência, o *Piccolo* pode facilmente ser integrado com aplicações tanto do Windows quanto de Java. Além de hospedar o *Piccolo* na interface da aplicação, a tela (*canvas*) mantém também um nó da câmera e da camada (*layer*).

3.2 Nós

Há uma grande variedade dos nós que podem ser incluídos em uma interface de usuário com zoom. E, há diversas maneiras fazer novos nós para cada aplicação.

3.2.1 Usando classes existentes

O *Piccolo* contém algumas implementações padrões para três tipos de nós visuais: *PText*, *PPath*, e *PImage*. Estes podem ser usados diretamente em uma nova aplicação, ganhando tempo para desenvolvedor dos esforços requeridos para criar tais elementos, mas reduzem o conhecimento que têm sobre como esse elemento é implementado.

3.2.2 Subclasses de classes existentes

Freqüentemente as implementações padrões dos *PNodes* são "quase mas não completamente" o que é requerido para uma interface. Em tais circunstâncias, faz sentido customizar o nó existente para atender as necessidades da interface. Isto ganha tempo comparado a escrever uma nova classe e fornece a introspecção na organização do nó. Esta metodologia aumenta o número das classes no sistema e assegura que a intenção original dessa hierarquia de herança é mantida.

3.2.3 Composição

Uma alternativa a subclasses, composição pode ser usada nas situações onde a funcionalidade nova requerida pode ser criada arranjando diversos nós pré-existentes. Uma etiqueta de texto poderia ser definida como um retângulo (para

definir a borda) e com um objeto do texto dentro. Para facilitar este tipo da criação qualquer nó pode ter outros nós filhos adicionados a eles. Adicionando um nó de texto como um filho do nó do retângulo cria-se a etiqueta do texto mencionada acima. A composição livra o desenvolvedor dos detalhes da classe, permite reusar ferramentas existentes e permite a configuração dinâmica (os novos arranjos das figuras podem ser construídos durante o funcionamento do programa). Mas a composição não fornece tanta liberdade quanto a subclasses (somente se pode compor o que está disponível). E, por causa de sua natureza dinâmica, pode ser difícil localizar problemas.

3.2.4
Nó Customizado

A máxima liberdade criativa vem de criar subclasses de *PNode* diretamente. *PNode* fornece já o comportamento padrão para todas as operações (é uma classe concreta). Assim, é razoavelmente fácil gerar subclasses. O código a seguir define um nó de forma elíptica e sem borda (Quadro 1).

```
public class SimpleEllipseNode : PNode {
  private GraphicsPath ellipse;

  // Este nó se utiliza de GraphicsPath para definir seu formato.
  public GraphicsPath Ellipse {
        get {
              if (ellipse == null) ellipse = new GraphicsPath();
              return ellipse;
        }
  }

  // Este método mantem a elipse consistente com os limites.
  public override bool SetBounds(float x, float y, float width, float height) {
        if(base.SetBounds (x, y, width, height)) {
              Ellipse.Reset();
              Ellipse.AddEllipse(x, y, width, height);
              return true;
        }
        return false;
  }
```

```
        // Subclasses não retangulares devem sobrescrever este método
        // para receber corretamente eventos do mouse.
        public override bool Intersects(RectangleF bounds) {
                Region ellipseRegion = new Region(Ellipse);
                return ellipseRegion.IsVisible(bounds);
        }

        // Nós que sobrescrevem a representação visual devem
        // sobrescrever o método Paint (renderização)
        protected override void Paint(PPaintContext paintContext) {
                Graphics g = paintContext.Graphics;
                if (Brush != null) {
                        g.FillPath(Brush, Ellipse);
                }
        }
    }
```

Quadro 1 – Exemplo de código para criar um nó customizado.

3.3 Interações do usuário

Event listeners (ouvintes de eventos) representam os modos da interação entre o usuário e a interface. O *Piccolo* possui *event listeners* que deixam o usuário alterar o zoom e reposicionar o seu ponto de visão, e arrastar nós na interface. Um ponto importante no projeto de uma interface que se utiliza do *Piccolo* é projetar um conjunto de *event listeners* que definirão a experiência do usuário.

Uma vez criado um *event listeners* deve ser registrado com um nó de modo que possa receber eventos. Muitos *event handlers* (processador de eventos) são registrados com o nó da câmera de modo que captem todos os eventos que vêm da tela associada a esta câmera.

3.4 Restrições de Layout

Geralmente uma interface possui restrições que devem ser mantidas entre os nós e seus filhos. Por exemplo, um nó pode sempre manter seus filhos alinhados

ou um nó pode querer expandir sua base para sempre abrigar seus filhos dentro de seus limites.

3.5 Ações

Event handlers permitem a interface reagir ao usuário. Ações são usadas para dar vida a interface através de animação e outros comportamentos.

Ações controlam aspectos temporais do sistema do *Piccolo*, normalmente parte de um nó. Este comportamento pode não ter uma duração predeterminada ou pode continuar até uma condição de parada é atendida. Ações de duração determinada podem ser definidas pela quantidade de tempo, independente do frame rate.

3.6 Modelo de Implementação

Piccolo é um framework de manipulação direta que permite zoom indefinido nas interfaces [26].

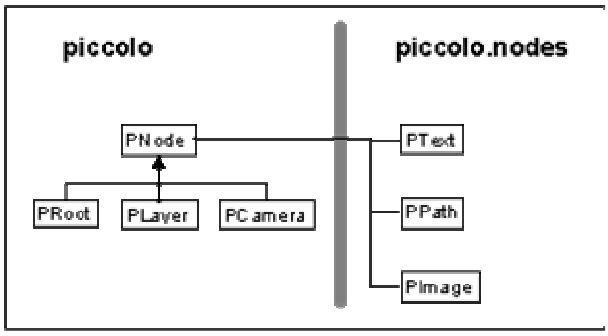

Figura 5 – Hierarquia de classes do Piccolo

Existem cinco classes principais que definem o núcleo da estrutura:

- PNode (qualquer coisa que é visível e inicia eventos)
- PCamera (um nó que olha outros nós da camada, e aplica uma transformação visual)
- PLLayer (um nó que pode ser visto por uma câmera)
- PRoot (a raiz da árvore de exibição do *Piccolo*)

- PCanvas (fica fora da visão hierárquica das classes pois é o componente que abriga os *PNodes* em Java Swing ou em aplicações Windows .NET. Cada PCanvas é associado com uma *PCamera*. Mas todas as câmeras não são necessariamente associadas diretamente com um *PCanvas.* Câmeras internas, por exemplo, não o são.)

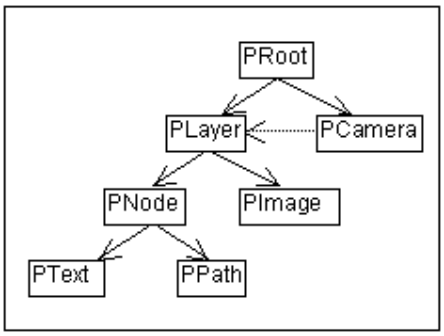

Figura 6 – Estrutura de Execução do Piccolo

Em tempo de execução estas classes formam uma estrutura de árvore com a *PRoot* situada no alto. Cada *PCamera* é normalmente ligada com pelo menos uma *PLayer* que enxerga as transformações. Se uma câmera for associada com uma tela então a visão da câmera é exibida na tela, e os eventos de entrada vindos da tela entram na cena gráfica do *Piccolo* no ponto hierárquico da câmera.

3.6.1 A Classe PNode

Os nós representam o principal conceito no design do *Piccolo*. Qualquer objeto que aparecer na tela deve herdar da classe nó. Mais do que isso, todos nós desenhados na tela podem ter outros nós filhos associados a eles. Estruturas visuais são formadas do agrupamento de coleções de nós.

Cada nó tem também sua própria transformação que é aplicada antes que o nó seja desenhado na tela. Esta transformação pode ser modificada para escalar e transladar o nó. A transformação existe diretamente acima do nó, mas abaixo do pai do nó. Assim, transladando ele, será transladado o nó (e todos os seus descendentes), mas não afetará o nó pai.

3.6.2
A Classe PCamera

As câmeras são os nós que têm uma visão adicional de transformação e uma coleção de camadas além da coleção de filhos que herdam de *PNode*. A visão de transformação é aplicada antes de desenhar ou de escolher as camadas, mas não ao arrastar ou ao escolher os filhos da câmera.

As câmeras podem também referenciar uma *PCanvas*, e passar para esta tela eventos de renderização. A tela então pediria mais tarde à câmera para desenhar a região danificada em sua superfície.

3.6.3
A Classe PLayer

Nós de camadas são nós que podem ser vistos por uma ou mais câmeras. Cada nó de camada mantém uma lista das câmeras que a estão vendo, e notificam estas câmeras quando são repintados.

3.6.4
A Classe PRoot

O *PRoot* serve como o nó mais alto na estrutura de execução do *Piccolo*; todos nós restantes são filhos ou descendentes diretos de seus filhos. O *PCanvas* comunica-se com o nó da raiz para controlar atualizações da tela e para despachar eventos aos descendentes.

3.6.5
A Classe PCanvas

O *PCanvas* é um JComponent em Piccolo.Java e em um controle em Piccolo.NET. Assim, é usado para ver a cena gráfica do *Piccolo* no Java.Swing e nas aplicações de Windows .NET respectivamente. O *PCanvas* vê a cena gráfica com uma *PCamera*. Ela Envia eventos da entrada a essa câmera, e usa essa câmera para desenhar a si mesma. Transladar e escalar a visão dessa câmera é a forma que o zoom e o reposicionamento funcionam.