UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE COMPUTAÇÃO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

BRUNO DA COSTA MOREIRA
GIANCARLO VASCONCELOS TAVEIRA DO NASCIMENTO

APLICATIVO MULTIUSUÁRIO COM A UTILIZAÇÃO DA TECNOLOGIA MICROSOFT MULTIPOINT INTEGRADA A FERRAMENTA FLASH

NITERÓI
2009

BRUNO DA COSTA MOREIRA
GIANCARLO VASCONCELOS TAVEIRA DO NASCIMENTO

APLICATIVO MULTIUSUÁRIO COM A UTILIZAÇÃO DA TECNOLOGIA MICROSOFT MULTIPOINT INTEGRADA A FERRAMENTA FLASH

Dissertação apresentada ao Curso de Graduação em Ciência da Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Bacharel.

Orientador: Prof. Esteban Walter Gonzalez Clua

NITERÓI
2009

BRUNO DA COSTA MOREIRA
GIANCARLO VASCONCELOS TAVEIRA DO NASCIMENTO

APLICATIVO MULTIUSUÁRIO COM A UTILIZAÇÃO DA TECNOLOGIA MICROSOFT MULTIPOINT INTEGRADA A FERRAMENTA FLASH

Dissertação apresentada ao Curso de Graduação em Ciência da Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Bacharel.

Aprovada em dezembro de 2009.

BANCA EXAMINADORA

Prof. Dr. Esteban Walter Gonzalez Clua - Orientador
UFF

Prof. Dr. Anselmo Antunes Montenegro
UFF

Prof.ª Dr.ª Aura Conci
UFF

NITERÓI
2009

AGRADECIMENTOS

À Universidade Federal Fluminense,

Ao Professor Esteban Walter Gonzalez Clua,

À psicóloga Vanessa Karam de Lima Ferreira.

À Pedro Thiago de Souza Catunda Mourão, responsável pela concepção dos *layouts* utilizados nesta aplicação.

À todos os professores com quem convivemos e de quem tivemos o privilégio de sermos alunos no Curso de Graduação em Ciência da Computação.

RESUMO

A tecnologia Microsoft *MultiPoint* é comumente utilizada para softwares com o foco em e-learning, pois tem como propriedade principal a possibilidade do uso de vários mouses em um mesmo terminal, possibilitando que vários alunos interajam com o computador ao mesmo tempo. Este projeto visa utilizar essa tecnologia de propriedade muito interessante de outra forma. Nele foi criado um software multiusuário que auxilia na investigação comportamental de candidatos através da aplicação da técnica de dinâmica de grupo ecológica, que visa expor o candidato a um cenário diretamente ligado à prática do cargo para o qual ele está sendo selecionado.

Palavras-chave: Dinâmica ecológica, Microsoft MultiPoint, Investigação comportamental

ABSTRACT

The Microsoft MultiPoint technology is commonly used to develop software with focus on e-learning because its main feature is the possibility to use several mice on the same screen, allowing many students to interact with the computer at the same time. This project aims to use this interesting technology in a different way. The idea is to create a multi-user software that will help on the research of the behavior of candidates on an ecological group dynamics, links the candidate to a scenario very similar to the one he would find at work.

Keywords: Ecological Dynamics, Microsoft MultiPoint, Behavior Research.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
SDK	Software Development Kit
USB	Universal Serial Bus
WPF	Windows Presentation Foundation
XML	eXtensible Markup Language
DLL	Dynamic Link Library
RH	Recursos Humanos
PNG	Portable Network Graphics
BMP	Bitmap
JPG	Joint Photographic Experts Group
SWF	Shockwave Flash

SUMÁRIO

1. INTRODUÇÃO

1.1. VISÃO GERAL

O presente trabalho monográfico tem a finalidade de demonstrar como foi feita uma aplicação para o uso investigativo do comportamento de candidatos em um tipo de dinâmica de grupo diretamente ligada a prática do cargo selecionado, que será chamada de dinâmica de grupo ecológica.

Serão apresentadas as motivações existentes para o desenvolvimento do trabalho e os objetivos do mesmo. Diversos temas serão abordados, tais como o porquê do desenvolvimento da tecnologia *MultiPoint*, como ela é inserida e o seu propósito.

Será explicado o que é a tecnologia, acompanhado de uma explicação com detalhes de como se utilizar o *MultiPoint* para projetos de finalidades genéricas. Serão explicadas também, as bibliotecas e métodos utilizados para fazer a integração com a plataforma *Flash*. Esta plataforma proporciona diversas opções para o desenvolvimento mais amigável de uma boa interface gráfica além de ser multiplataforma.

Também será definido e apresentado efetivamente o projeto de aplicação, o que foi feito em termos de estudos da tecnologia e da área de psicologia, levantamento de requisitos, implementação do código e organização do projeto. Além disso, será visto a preparação do que é considerado o ambiente ideal para o uso do software com os propósitos abordados.

Neste projeto também serão expostas as dificuldades encontradas durante a etapa de implementação do *software*. Ademais, é feita também uma análise crítica da ferramenta e das tecnologias envolvidas, através do levantamento de vantagens e desvantagens encontradas nas mesmas, e então são elaboradas sugestões para trabalhos futuros assim como melhorias no projeto.

1.2. MOTIVAÇÃO

Pesquisadores da Microsoft criaram uma tecnologia conhecida como *MultiPoint* para ajudar estudantes de países em desenvolvimento a tirarem o máximo de proveito possível de computadores precários, permitindo que eles trabalhem juntos utilizando diversos mouses (ao contrário de apenas um mouse como dispositivo de entrada) em um único computador.

Os irmãos Jimmy, Mark e Luke Dickinson, que são moradores da cidade de Tigard no estado de Oregon nos Estados Unidos da América, são grandes fãs do *MultiPoint*, o que faz sentido, pois eles cresceram em uma família de treze irmãos. [2]

Os irmãos Dickinson usaram o *MultiPoint* como a tecnologia base para criar uma coletânea de mini-jogos educacionais [14] para a competição de tecnologia entre estudantes promovida pela Microsoft: A "Imagine Cup". O projeto deles ganhou o prêmio máximo na competição Imagine Cup 2009, realizada nos E.U.A., na categoria Software Design.

O tema da Imagine Cup era como usar a tecnologia para resolver alguns dos problemas mundiais mais difíceis, como descrito no documento das Nações Unidas que citam as Metas de Desenvolvimento do Milênio. [4]

Esses objetivos incluem Educação Global, que é onde se encaixa o *MultiPoint*. Esta tecnologia nasceu no laboratório de pesquisa da Microsoft na Índia, enquanto pesquisadores buscavam maneiras de utilizar o computador de forma mais eficaz nas escolas que possuem um número limitado de máquinas e com um número grande de estudantes.

Observando o trabalho realizado pelos irmãos Dickinson[2][14] percebe-se o potencial contido nas aplicações desenvolvidas com o SDK Microsoft MultiPoint. A motivação deste trabalho é desenvolver uma aplicação utilizando esta tecnologia inovadora e pouco difundida.

1.3. OBJETIVOS

Um dos objetivos deste projeto é desenvolver um software voltado para aplicação em dinâmicas de grupo que torne possível a aplicação da técnica de dinâmica de grupo ecológica. A dinâmica de grupo ecológica é aquela que aproxima o candidato do cenário encontrado no ambiente de trabalho da empresa. Desta forma, uma avaliação diferenciada do perfil do candidato pode ser feita, uma vez que são levadas em consideração tarefas que ele realizará no seu dia a dia de trabalho.

O software visa expor o candidato a cenários mais complexos, elaborados e realistas do que aqueles possíveis apenas com caneta e papel, tais como o uso de testes psicológicos, que vêm sendo aplicados em ambientes organizacionais desde a década de

70 (Alencar e Gomes, 2005). Os testes psicológicos, embora tenham sua determinada importância, são circunscritos a avaliação de determinado aspecto psicológico ou cognitivo. Além disso, a utilização do computador na dinâmica de grupo aprovisiona um ambiente mais confortável para o candidato, uma vez que diminui sua inibição, permitindo com isso que ele aja de forma mais natural. Este é um outro fator que contribui para uma avaliação diferenciada do candidato.

Para o desenvolvimento desta aplicação será utilizado a tecnologia Microsoft *MultiPoint* que provê todas as bibliotecas necessárias para a interação de diversos dispositivos de entrada (mouse) em um mesmo terminal. Esta tecnologia estimulou a criação de diversos aplicativos, em sua grande maioria, voltados para e-Learning. Este projeto visa aplicar esta tecnologia de uma forma diferenciada, uma vez que deixa de ser unicamente aplicada em ambientes escolares, para ser aplicada dentro da organização.

Com isso em mente, outro objetivo deste projeto é ampliar a aplicabilidade dessa tecnologia multiusuário. Dessa forma, o computador auxiliaria na avaliação dos comportamentos humanos, possibilitando um ambiente colaborativo, o que facilitaria ainda mais a investigação dos aspectos comportamentais.

Seguindo estas duas vertentes, o foco principal deste trabalho é mostrar que a tecnologia Microsoft *MultiPoint* pode ser aplicada com um propósito diferente do qual ela foi criada, e mostrar que a necessidade do uso da mesma na aplicação da dinâmica de grupo ecológica é altamente relevante.

1.4. ORGANIZAÇÃO DO TRABALHO

O capítulo 1 traz uma breve Introdução e uma visão geral sobre o documento. No capítulo 2 será mostrado como funciona a API Microsoft MultiPoint e como fazer sua integração com a plataforma *Flash*. O terceiro tópico apresenta o aplicativo proposto, que visa possibilitar o uso da técnica de dinâmicas de grupos ecológicas, utilizando a tecnologia Microsoft MultiPoint. Em seguida, no capítulo 4, haverá uma análise crítica da API, relatando as experiências obtidas com o desenvolvimento do projeto. Para finalizar serão apresentadas ideias para trabalhos futuros.

2. A API MULTIPOINT

O MultiPoint SDK é primariamente escrito para ser programado em Visual C#, linguagem de programação orientada a objetos desenvolvida como parte da plataforma .NET. Por este motivo, a maioria das aplicações é feita nessa linguagem. Porém, utilizando uma ferramenta Visual Studio, que é um pacote de programas da Microsoft para desenvolvimento de software, desde que este possua a extensão que dá suporte a Windows Presentation Foundation (WPF), é possível desenvolver aplicativos com MultiPoint também em outras linguagens como C++,com o Visual C++ , ou Basic, com o Visual Basic. Além da ferramenta de desenvolvimento é necessário ter instalado na máquina o .NET Framework versão 3.0 ou mais atual. A tecnologia Microsoft *MultiPoint* funciona com Microsoft Windows XP e Microsoft Windows Vista.

Para esta aplicação se fez o uso do .NET 3.5 com o Visual C# e o Microsoft Windows XP.

2.1. UM BREVE HISTÓRICO

Para diversas escolas, prover um computador por criança não é uma opção financeiramente viável. [1]

Como parte do comprometimento da Microsoft em expandir os benefícios da tecnologia para comunidades desprivilegiadas, pesquisadores desta empresa desenvolveram a plataforma Microsoft *MultiPoint*.

Microsoft *MultiPoint* permite que um computador seja utilizado por diversos estudantes de forma simultânea. Utilizando múltiplos dispositivos de mouse, cada um com seu cursor único na tela do computador, o Microsoft *MultiPoint* permite que até 30 estudantes utilizem ao mesmo tempo um único computador e aprendam através de softwares educacionais.

Esta abordagem não apenas oferece uma solução mais acessível, mas também cria um ambiente colaborativo que envolve cada um dos alunos permitindo que eles efetivamente aprendam. *MultiPoint* pode ser utilizado no computador em um laboratório composto por um pequeno grupo de alunos trabalhando em conjunto dentro de uma aplicação voltada para o ensino. Ele também poderá ser utilizado em salas de aula com um número grande de alunos participando da atividade, enquanto o professor supervisiona a execução da mesma.

MultiPoint coloca diversos cursores na tela do computador quando um usuário insere diversos dispositivos na portas USB da máquina. Além disto, um cursor é atribuído para cada mouse. Normalmente o que ocorre quando múltiplos dispositivos de mouse são inseridos em um computador é que os usuários ficariam se degladiando para controlar apenas um cursor. A proposta do *MultiPoint* é permitir um ambiente mais aberto à competição e colaboração.

Esta tecnologia ajuda os estudantes a mudar de uma aprendizagem passiva para uma aprendizagem ativa, pois cria ambientes colaborativos para que os estudantes interajam com eles.

De acordo com a Microsoft, dentro de uma classe com quarenta alunos e apenas quatro computadores para todos eles, colocar dez alunos ao redor de cada máquina enquanto apenas um estudante assume a posição central e controla o mouse é extremamente ruim. Os outros estudantes apontam, fazem gestos, anseiam pelo controle do mouse, mas eles acabam por não tê-lo e logo perdem o interesse e voltam a atenção para outras coisas.

Melhorar a proporção de PC-por-aluno através da compra de mais computadores não é uma solução viável, em diversas escolas nos países em desenvolvimento, pois os custos financeiros são muito altos, e mesmo com um número maior de máquinas os computadores tradicionais não permitem um ambiente de aprendizagem colaborativa e de trabalho em equipe.

Microsoft *MultiPoint* busca resolver este problema colocando estudantes para usar computadores e aprenderem juntos ao invés de uma experiência isolada onde cada aluno senta-se sozinho, cada um em seu computador. Além disso, *MultiPoint* oferece uma opção financeiramente viável para reduzir a proporção de estudantes-por-PC, e provê uma plataforma Windows para desenvolvedores de jogos educacionais criarem aplicações de aprendizagem colaborativa.

O SDK do Microsoft *MultiPoint* torna simples a captura de cliques e outras ações do mouse, além do salvamento das ações dos mesmos, possibilitando que o programador se concentre em desenvolver soluções para a sua aplicação.

MultiPoint não deve ser confundido com uma aplicação que permite que múltiplos usuários controlem múltiplos dispositivos de mouse para realizar operações básicas do sistema operacional ou sobre qualquer tipo de aplicação que nele esteja sendo executada. Nesses casos, o sistema normalmente não consegue identificar qual mouse realizou determinada ação e, em geral, não há opções de controlar as permissões de cada mouse. *MultiPoint* é um framework que permite que desenvolvedores criem aplicações específicas para extrair as vantagens de se usar múltiplos dispositivos, incluindo a capacidade de reconhecer e gerenciar os cliques de cada mouse, tratar cada usuário de forma distinta e ainda designar permissões diferentes para cada mouse. Por exemplo, o mouse da professora em uma classe necessita ter permissões diferenciadas das permissões dos alunos.

Como pode ser notado, os softwares desenvolvidos até então são de cunho exclusivamente educacional e se demonstraram aplicações bastante simples, que não exigem muito da tecnologia *MultiPoint*, como operações mais avançadas ou novos métodos. Isto demonstra um aspecto negativo de se ter uma tecnologia com tamanho potencial sendo utilizada em apenas um contexto.

Com a comprovação da utilidade da tecnologia para outros contextos tem-se um grande ganho, pois com a difusão da tecnologia para outras áreas, haveria um maior interesse em incluir novas funcionalidades e realizar melhorias no sistema atual. Assim, acredita-se que investimentos de diversas áreas surgiriam de forma a desenvolver ainda mais o potencial da tecnologia.

2.2. PRINCIPAIS COMPONENTES DO SDK

Abaixo estão listadas algumas classes, propriedades e métodos contidos no SDK MultiPoint:

Namespace: Microsoft.SDK

Contém classes que ajudam os usuários a inicializar as variáveis de controle, desenhar cursores para os diversos dispositivos de mouse e gerenciar o cursor do sistema.

CurrentWindow

Referente à janela corrente da aplicação. Os dispositivos de mouse somente funcionarão na janela declarada como **CurrentWindow**.

DeviceArrivalEvent

Evento que é disparado quando um mouse é conectado ao computador.

DeviceRemoveCompleteEvent

Evento que é disparado quando um mouse é desconectado do computador.

Instance

Retorna uma instância do SDK.

DrawDevices()

Usada para renderizar o atributo DeviceVisual para todos os dispositivos de mouse conectados ao computador – isto faz com que os cursores sejam desenhados na tela.

HideSystemCursor()

Torna o cursor do sistema invisível.

MouseDeviceList

Possui o atributo **DeviceInfo** do tipo MouseDevice. Esta lista é preenchida pelos métodos **RegisterMouseDevice**() e **RegisterInputDevices**().

Namespace: Microsoft.CommonTypes.

Inclui os tipos mais usados pelo *framework* MultiPoint.

DeviceInfo

Define diversas propriedades de um mouse.

DeviceID

String de valor único que identifica cada dispositivo de mouse. Usualmente é um número de 10 dígitos codificado como uma string.

DeviceName

O nome que o Sistema Operacional atribui ao dispositivo. Em geral, este atributo não é muito utilizado nas aplicações MultiPoint.

DeviceType

Tipo do dispositivo como, por exemplo, “mouse”.

DeviceVisual

O objeto que contém informações visuais do dispositivo. Alguns dos atributos e métodos usuais para o desenvolvimento de aplicações MultiPoint são as seguintes:

CursorColor

Retorna ou atribui uma cor para o cursor. Aceita cores que contenham valores do tipo **System.Windows.Media.Color.**

CursorImage

Retorna ou atribui uma imagem para o cursor. Aceita imagens do tipo **System.Windows.Media.Imaging.BitmapImage.**

DisableLeftMouseButton(), DisableRightMouseButton()

Possibilita que os botões sejam desabilitados.

DisableMouse

Possibilita que o mouse seja desabilitado.

DisableMovement

Possibilita "congelar" o mouse na posição corrente na tela.

Visible

Atributo que retorna/modifica a visibilidade do mouse.

ID

Identificador mais simples e amigável. Este valor é atribuído pelo SDK e não pelo Sistema Operacional. O ID é um valor inteiro que começa em zero e vai sendo incrementado quando novos dispositivos são inseridos.

2.3. INICIALIZAÇÃO DOS COMPONENTES

O primeiro passo a ser dado para que seja possível que uma aplicação receba eventos de múltiplos dispositivos de mouse é inicializar o SDK MultiPoint. Para isso, é preciso associar o objeto **MultiPointSDK** à janela WPF. Abaixo estão os passos que devem ser dados para a correta inicialização:

i. Declarar um *event handler* para o carregamento da janela (Window.Loaded). Como a inicialização dos dados deve ser feita depois que a janela foi carregada, coloca-se os passos de inicialização dentro deste *event handler*.

ii. Chamar o método **initialize** presente no objeto **MultiPointSDK**, passando como parâmetro a janela corrente.

```
private void Window1_Loaded(object sender, EventArgs e)
{
    MultiPointSDK.Instance.Initialize(this);
}
```

O que o SDK do MultiPoint faz em sua implementação interna é:

- Guarda a referência na janela corrente para que o SDK saiba qual janela deve ser monitorada para a captura eventos de MultiPoint. Como o SDK precisa saber qual janela é responsável pela interface gráfica, esta associação deve ser feita antes de desenhar qualquer elemento na tela.
- Guardar as referências para os dispositivos de mouse conectados ao computador. O SDK enumera todos os dispositivos de mouse e os coloca em uma lista disponível para a aplicação. Apenas dispositivos USB são reconhecidos pelo MultiPoint SDK, então dispositivos de mouse com entrada PS/2, por exemplo, não são reconhecidos.
- Cria um **DeviceVisual** para cada mouse e atribui um cursor padrão para cada um deles.
- Esconde o ponteiro do Sistema Operacional. Dentro de uma aplicação MultiPoint , o ponteiro do sistema não é utilizado então torná-lo invisível é uma boa prática para evitar confusão.

2.4. EXECUTANDO A APLICAÇÃO

Com apenas os passos de inicialização é possível ver algum resultado ao rodar a aplicação. Ao executá-la pode-se ver que, mesmo sem plano de fundo com o qual os mouses vão interagir, a aplicação exibe um cursor padrão para cada mouse conectado ao computador. Cada mouse possui todas as suas propriedades, incluindo **DeviceVisual**. Além disso, cada cursor já responde separadamente ao movimento de cada mouse.

Neste momento surge um problema que é a inexistência de um tratamento para fechar a aplicação. Os dispositivos de mouse controlados pelo MultiPoint não são capazes de clicar em botões padrão, como o botão de **Fechar** no topo de uma janela do sistema. Todavia, uma solução simples é associar um botão do teclado, por exemplo, o

“Esc”. Para que isto ocorra, basta adicionar um **KeyEventHandler** dentro do construtor da aplicação.

```
this.KeyDown += new KeyEventHandler(KeyDown_Event);
```

Após adicionar o KeyEventHandler, criar um método que trata o evento, como a seguir:

```
private void KeyDown_Event(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Escape)
    {
        App.Current.Shutdown();
    }
}
```

2.5. INSTANCIANDO O CONTROLADOR MULTIPOINT

A maneira que o MultiPoint trata as mensagens de interrupções de mouse de forma a permitir o uso de múltiplos dispositivos faz com que a maioria dos controles WPF não respondam aos eventos de clique de MultiPoint. MultiPoint possui o seu próprio controlador de interrupção de mouse, que se encarrega de fazer o tratamento dos cliques de mouse.

Para adicionar este controlador que recebe e trata os eventos de clique MultiPoint, é necessário adicionar um controlador que implementa a interface **IMultiPointMouseEvents**. O *namespace* **Microsoft.MultiPoint.Controls** provê os controladores **MultiPointButton** e **MultiPointTextBox.** A partir do momento que a interface IMultiPointMouseEvents é implementada, estes controladores ganham a capacidade de receber e tratar os eventos gerados pelos diversos dispositivos de entrada (mouse).

Primeiro, adiciona-se uma referência dentro do Projeto para os arquivos binários Microsoft.MultiPoint.Controls (que estão presentes dentra da pasta Bin da pasta de instalação do SDK), e importa-se o *namespace* dentro do código do arquivo **Window1** como é apresentado no exemplo abaixo:

```
using Microsoft.MultiPoint.Controls;
```

Depois, adiciona-se a referência para este *namespace* dentro do arquivo Window1.xaml, como demonstrado a seguir:

```
<Window x:Class="MultiPoint_Basic_App.Window1"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentatio
n"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mp="clr-
namespace:Microsoft.MultiPoint.Controls;assembly=Microsoft.Mult
iPoint.Controls"
    Title="Window1" WindowState="Maximized">
```

Como o cursor do sistema não é completamente desativado, e ao invés disso ele é apenas escondido, ou seja, não é visível, é recomendado que o valor do atributo **WindowState** seja igual a “Maximized” para evitar que o cursor do mouse seja capaz de clicar em regiões fora da janela da aplicação, ativando outras janelas, por exemplo.

Neste momento, deve-se adicionar um botão **MultiPointButton** à janela através do seguinte código, que deverá ser incluído dentro do arquivo Window1.xaml:

```
    <Grid>
        <mp:MultiPointButton Name="mpButton"
                             Content="Click Me"
                             Click="mpButton_Click"/>
    </Grid>
```

2.6. TRATANDO MÚLTIPLOS CLIQUES

Agora que já existe um controlador (botão) capaz de receber os cliques de múltiplos dispositivos, é necessário declarar um *event handler* para tratar o evento. No código seguinte, por exemplo, é declarado um *event handler* chamado **mpButton_Click** para responder aos cliques dentro do **mpButton.**

```
mpButton.MultiPointClick +=
      new RoutedEventHandler(mpButton_Click);
```

Cria-se então o método que é executado quando o evento ocorre:

```
private void mpButton_Click(object sender, RoutedEventArgs e)
{
    mpButton.Content = "Click";
}
```

2.7. DETERMINANDO QUAL MOUSE FOI CLICADO

Para se adicionar código que determina qual mouse clicou no botão e para exibir o ID deste mouse dentro do botão serão realizados os seguintes passos:

Primeiro, pega-se uma instância do botão que foi clicado. Faz-se a conversão do objeto “sender” para o tipo **MultiPointButton** e atribui este valor a um objeto local do tipo **MultiPointButton**:

```
MultiPointButton btn = (MultiPointButton)sender;
```

Para recuperar as propriedades do mouse que foi clicado deve-se fazer uma conversão do objeto **RoutedEventArgs**, recebido por parâmetro pelo *event handler*, para um tipo **MultiPointMouseEventsArgs**, como demonstrado no código a seguir:

```
MultiPointMouseEventArgs multipointargs = e as MultiPointMouseEventArgs;
```

O operador **as** é utilizado para evitar que **InvalidCastExceptions** ocorram, isto é, caso a conversão de tipo não seja possível. Por exemplo, se um botão for clicado por um evento de teclado então a conversão falhará e a variável **multipointargs** terá valor *null*. Deve-se então testar o valor de multipointargs antes de prosseguir com a execução:

```
if (multipointargs != null)
{
    // Não houve problema e o código pode ser executado
}
```

Para determinar qual mouse foi clicado, deve-se consultar a propriedade **ID** presente dentro do objeto **DeviceInfo**:

```
int current = multipointargs.DeviceInfo.ID;
```

Para fins de teste, pode-se utilizar o seguinte código, onde é exibido o ID do mouse dentro do botão no qual ele clicou:

```
mpButton.Content =
    String.Format("Mouse #{0} Clicou aqui", current.ToString());
```

2.8. CUSTOMIZANDO O MULTIPOINT

A partir do momento que se tem todos os componentes necessários para o tratamento dos eventos de múltiplos dispositivos de mouse surge um novo problema: cada um dos dispositivos recebe um mesmo cursor padrão. Isto pode gerar confusão, pois será difícil distinguir entre os dispositivos já que todos os cursores são idênticos. Com isto em mente, uma solução é customizar a aplicação através da atribuição de cursores diferentes para cada dispositivo.

A customização dos ponteiros pode ser feita de duas formas: pode-se atribuir uma imagem diferente para cada mouse ou é possível também alterar propriedades do mouse como, por exemplo, a cor.

2.8.1 CUSTOMIZANDO AS CORES

Para fazer a customização de cores deve-se criar um método que irá iterar por cada um dos dispositivos e alterar a propriedade de cor de cada um deles. Desta forma, cria-se um método, por exemplo, **CustomizeCursors()** que será chamado dentro do **Window1_Loaded** *handler*, logo após a inicilização do MultiPoint.

Dentro do método *CustomizeCursors()* será feita a iteração pela lista de dispositivos conectados e alteração das cores. A lista de dispositivos conectados está dentro de **MouseDeviceList** que é uma propriedade do objeto **MultiPointSDK.** O código abaixo mostra a iteração por todos os dispositivos contidos na lista **MouseDeviceList**:

```
for (int i = 0; i <
MultiPointSDK.Instance.MouseDeviceList.Count; i++)
{
    // Realizar alteração em cada dispositvo
}
```

Para recuperar um membro específico da lista de dispositivos é possível fazer referência a ele por seu índice. Pode-se pegar um objeto do **MouseDeviceList**, cujo tipo é **MouseDevice**, e fazer a conversão para **DeviceInfo**, como mostrado abaixo:

```
DeviceInfo mouseObject =
    (DeviceInfo)MultiPointObject.MouseDeviceList[i];
```

Após criar um **mouseObject,** tem-se acesso a diversas propriedades do dispositivo, como o **DeviceVisual**. Para se ter acesso às propriedades específicas de MultiPoint faz-se a conversão do DeviceVisual para MultiPointMouseDevice. O código abaixo ilustra esta conversão:

```
MultiPointMouseDevice mpMouseDevice =
    (MultiPointMouseDevice)mouseObject.DeviceVisual;
```

O objeto **MultiPointMouseDevice** contém múltiplas propriedades, incluíndo **CursorColor**. Para modificar este atributo, utiliza-se o método **GetCursorColor()** que recebe um tipo inteiro como parâmetro e retorna uma cor. A utilização deste método encontra-se no exemplo seguinte:

```
mpMouseDevice.CursorColor =
    GetCursorColor(i); // eg: Cursor.Green
```

A implementação do método **GetCursorColor** está localizada abaixo:

```
private System.Windows.Media.Color GetCursorColor(int id)
{
    switch (id)
    {
        case 0:
            return Colors.Blue;
        case 1:
            return Colors.Green;
        case 2:
            return Colors.Red;
        default:
            return Colors.HotPink;
    }
}
```

Se existem dois dispositivos de mouse conectados ao computador então existirão dois cursores visíveis na tela (um azul e outro verde) quando a aplicação for executada. Se existirem quatro dispositivos conectados então existirão quatro ponteiros das seguintes cores (azul, verde, vermelho, rosa). Caso existam mais de quatro dispositivos conectados então todos esses excedentes terão a cor rosa.

Existem duas formas de evitar que haja uma repetição de cores. Pode-se adicionar mais cores ao método **GetCursorColor()** criando uma função que gere cores aleatórias baseadas no número inteiro passado como parâmetro (para evitar que uma mesma cor seja usada duas vezes) ou limitar o número de dispositivos.

Para limitar o número de dispositivos, é preciso saber a priori quantos dispositivos estão efetivamente conectados. Para tal, basta consultar a propriedade **Count** da lista **MouseDeviceList**. Se o número de dispositivos de mouse for maior que o desejado então será necessário modificar a propriedade **Visible** para o valor *false*.

```
private void EnforceMouseLimit(int max)
{
    //Caso existão muitos dispositivos...
    if (MultiPointSDK.Instance.MouseDeviceList.Count > max)
    {
       for (int i = max;
            i < MultiPointSDK.Instance.MouseDeviceList.Count;
            i++)
        {
            DeviceInfo mouseObject =
       (DeviceInfo)MultiPointSDK.Instance.MouseDeviceList[i];

            MultiPointMouseDevice mpMouseDevice =
       (MultiPointMouseDevice)mouseObject.DeviceVisual;
            //esconda os excedentes.
            mpMouseDevice.Visible = false;
        }
    }
}
```

Este método deve ser chamado dentro do *event handler* **Window1_Loaded,** passando como parâmetro o número máximo de dispositivos necessários para a aplicação. Por exemplo, limitar o número de dispositivos a três vai garantir que nenhum mouse rosa seja exibido.

Da mesma forma, é possível exigir um número mínimo de dispositivos que deve estar conectados ao computador para que a aplicação funcione.

2.8.2. ATRIBUINDO IMAGENS CUSTOMIZADAS

É possível atribuir imagens como ponteiros ao invés de simples cores. O primeiro passo é escolher as imagens que serão utilizadas como cursores, que podem ser de diversos formatos (.jpg, .gif, .png). As imagens escolhidas devem ser adicionadas como **resources** ao projeto criado dentro do Visual C#.

É necessário importar o *namespace* **System.Drawing** para se ter acesso às imagens como objetos **Bitmap**.

```
using System.Drawing;
```

Além disso, é necessário criar uma função que retorne uma imagem baseada no identificador do dispositivo, semelhante ao que foi feito com a customização de cores:

```
private Bitmap GetCursorImage(int id)
{
    switch (id)
    {
        case 0:
            return Properties.Resources.image01;
        case 1:
            return Properties.Resources.image02;
        case 2:
            return Properties.Resources.image03;
        case 3:
            return Properties.Resources.image04;
        default:
            return Properties.Resources.image05;
    }
}
```

Por último, como a propriedade **CursorImage** contida no **MultiPointMouseDevice** espera um objeto do tipo **BitmapImage**, é necessário fazer a conversão. O seguinte método cuida desta etapa:

```
public static BitmapImage
ConvertBitmapToBitmapImage(System.Drawing.Bitmap b)
{
    BitmapImage bmpimg = new BitmapImage();

    System.IO.MemoryStream memStream =
            new System.IO.MemoryStream();

    bmpimg.BeginInit();
    b.MakeTransparent(System.Drawing.Color.White);
    b.Save(memStream, System.Drawing.Imaging.ImageFormat.Png);
    bmpimg.StreamSource = memStream;
    bmpimg.EndInit();
    return bmpimg;
}
```

A função de conversão recebe um objeto **System.Drawing.Bitmap** e retorna um objeto **System.Windows.Media.Imaging.BitmapImage**. É possível agora pegar um ponteiro **Bitmap**, converter para **BitmapImage** e atribuir esta imagem à propriedade **CursorImage**:

```
mpMouseDevice.CursorImage =
    ConvertBitmapToBitmapImage(GetCursorImage(i));
```

A partir deste momento, já é possível ver as imagens como ponteiros dos dispositivos conectados, quando a aplicação for executada.

2.8.3. BLOQUEANDO OS DISPOSITIVOS

Em determinadas situações pode ser necessário bloquear os dispositivos de mouse, exceto o da pessoa que inicia e controla a aplicação.

Para permitir esta funcionalidade, deve-se modificar a propriedade **DisableMovement** do objeto **MultiPointMouseDevice** para *True*. Para bloquear todos os dispositivos de mouse, basta iterar pela lista de dispositivos, bloqueando cada um deles. O método **ToggleMiceFrozen** referencia um valor Booleano (**areMiceFrozen**) que pode ser utilizado para bloquear/desbloquear os diversos dispositivos quando necessário. A variável areMiceFrozen é declarada dentro da janela e contém o estado dos dispositivos, então deve-se manter esta variável atualizada sempre que uma alteração foi feita.

```
private void ToggleMiceFrozen()
{
    for (int i = 0;
             i < MultiPointSDK.Instance.MouseDeviceList.Count;
             i++)
    {
        DeviceInfo mpDeviceInfo =
        (DeviceInfo)MultiPointSDK.Instance.MouseDeviceList[i];

        MultiPointMouseDevice mpMouseDevice =
        (MultiPointMouseDevice)mpDeviceInfo.DeviceVisual;

        mpMouseDevice.DisableMovement = !areMiceFrozen;
    }
    areMiceFrozen = !areMiceFrozen;
}
```

Como a pessoa que controla a atividade é provavelmente a única que tem acesso ao teclado, é conveniente atribuir este evento a uma tecla específica. Desta forma, a pessoa controladora pode bloquear a atividade de todos os dispositivos e conseguir voltar a atenção de todos os usuários para alguma outra coisa. O código abaixo cuida do trabalho:

```
if (e.Key == Key.Space){
    ToggleMiceFrozen();
}
```

Um último detalhe que não pode ser esquecido é evitar que a tecla acionada (neste exemplo o "espaço") também ative o evento **mpButton_Click** quando o botão tiver o foco (quando um mouse tiver clicado nele). Para evitar este incidente basta fazer com que o botão não ganhe foco, alterando sua propriedade **focusable** para *False.*

```
<mp:MultiPointButton Name="mpButton"
                     Content="Click Me"
                     Click="mpButton_Click"
                     Focusable="False"/>
```

2.9. INTEGRAÇÃO COM O FLASH

2.9.1. A FERRAMENTA ADOBE FLASH

O Adobe *Flash* é um software primariamente de criação interfaces de gráfico vetorial utilizado geralmente para animações interativas.Estas que funcionam embutidas num navegador web ou diretamente pelo *Flash Player*, *player* distribuído pela Adobe.O player executa o arquivo de extensão ".swf" que é a extensão gerada pela ferramenta.

2.9.2. A INTEGRAÇÃO

Esta seção descreve como pode ser utilizado o *MultiPoint* para construir aplicações Adobe *Flash* que façam uso de múltiplos dispositivos de mouse. Será mostrado como comunicar eventos de *MultiPoint* do Visual C# para *Flash* e como receber pedidos de *Flash* para Visual C#.

Quando se utiliza a integração do *Flash* com o SDK *MultiPoint*, é possível acoplar conteúdo *Flash* dentro de uma aplicação Visual C# que suporta múltiplos dispositivos de mouse.

O conteúdo *Flash* é acoplado dentro de uma variável UserControler e é guardado dentro de um *Windows forms*, que por sua vez fica residente dentro de um WPF *grid*. Todos os eventos *MultiPoint* são capturados por um botão transparente que é posicionado dentro do *Windows forms*. Este botão cobre toda a área ocupada pelo conteúdo *Flash*, provendo assim uma camada transparente, que suporta *MultiPoint* por cima do conteúdo *Flash*.

Todos os eventos são capturados usando a aplicação Visual C# e comunicados para a aplicação *Flash* utilizando-se a API ***CallFunction*** da classe ***ExternalInterface*** disponibilizado pela Adobe *Shockwave Flash* OCX. Pedidos da aplicação *Flash* para o Visual C# são gerenciados através do registro para o evento ***FlashCall*** do *Shockwave Flash* OCX.

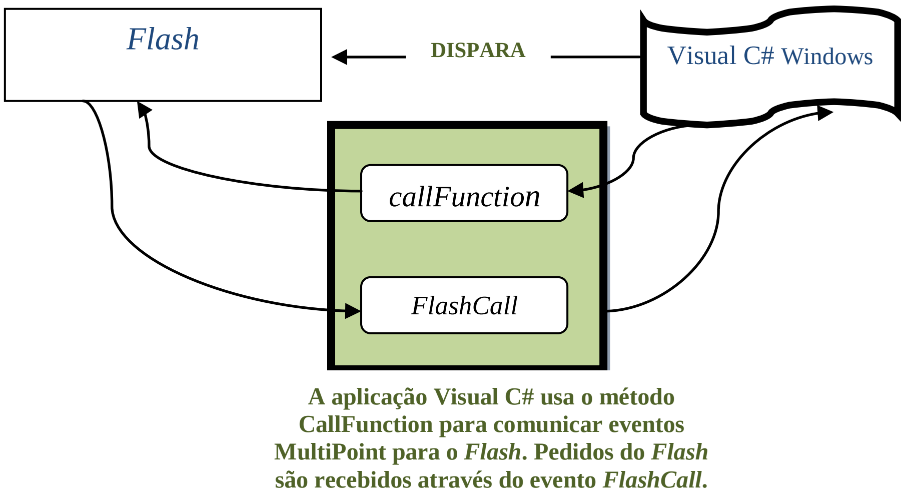

Fig.1 – Integração do Visual C# com o Flash

Através do uso do ***FlashUserControl***, incluso no SDK *MultiPoint*, é possível fazer o carregamento do *Flash Movie* e gerenciar toda a comunicação entre a aplicação Windows Visual C# e *Flash*. Este controlador pode ser utilizado para criar aplicações *MultiPoint* em *Flash*.

Sendo assim, mantem-se na aplicação Visual C# uma instância estática do ***FlashUserControl.***

```
private static FlashUserControl flashUC = new FlashUserControl();
```

Quando a janela principal é carregada, o ***FlashUserControl*** é adicionado à instância de ***WindowsFormsHost*** que é posicionado dentro do *grid* da aplicação principal. O nome do arquivo *Flash* é lido do arquivo de configuração e carregado através do método ***LoadMovie*** do ***FlashUserControl.*** Para passar os dados de inicialização, o ***FlashUserControl*** utiliza o método ***CallFunction*** do *Shockwave Flash OCX*, como pode ser visto no exemplo abaixo:

```
public void LoadMovie(int layer, string moviePath)
 {
      this.Flash.LoadMovie(layer, moviePath);
      this.Flash.CtlScale = "ExactFit";
      this.Flash.Top = 0;
      this.Flash.Left = 0;
 }

 public void InitializeFlash(string dataForInitialization)
 {
      this.Flash.CallFunction("<invoke name=\"InitializeFlash\"
returntype=\"xml\"><arguments><string>" + dataForInitialization +
"</string></arguments></invoke>");

 }
```

Na parte do código ActionScript dentro do *Flash*, deve-se importar a classe ***ExternalInterface*** para receber os dados da aplicação Windows Visual C#:

```
import flash.external.ExternalInterface;
```

A partir de agora será visto como é feita a comunicação de eventos da aplicação Windows Visual C# para o *Flash*, e como essas chamadas de eventos podem ser recebidas e processadas dentro da aplicação *Flash*.

O botão *MultiPoint* reconhece as coordenadas X e Y dos eventos de *MultiPoint*, bem como o atributo identificador (ID) do cursor (mouse) que efetuou o clique. Em seguida, essas informações são passadas para a aplicação *Flash* através da chamada do método ***CallFunction*** dentro da classe ***ExternalInterface.***

Pode-se imaginar o seguinte exemplo para ilustrar esta comunicação: têm-se o evento ***MultiPointMouseLeftButtonEvent*** que é capturado pelo ***MultiPointButton*** e comunicado para a aplicação *Flash* através do método ***HandleMouseLeftButtonUp*** do ***FlashUserControl***.

O ***FlashUserControl*** comunica esta informação para o *Flash* utilizando o método da ***CallFunction***, como mostrado abaixo:

```
public void HandleMouseLeftButtonUp(int coordinateX, int coordinateY,
string mouseDeviceInfo)
{
     this.Flash.CallFunction("<invoke name=\"HandleMouseLeftButtonUp\"
     returntype=\"xml\"><arguments><number>" + coordinateX +
     "</number><number>" + coordinateY + "</number><string>" +
     mouseDeviceInfo + "</string></arguments></invoke>");
 }
```

Na parte do *Flash*, a aplicação utiliza as coordenadas enviadas pelo Visual C# para determinar o que deve ser feito para responder àquele evento (realizar um *hitTest* no *stage* ou qualquer outro tratamento que for adequado). O *Flash* vai utilizar as coordenadas, bem como o MouseID para tratar o evento da forma mais apropriada.

```
ExternalInterface.addCallback ("HandleMouseLeftButtonUp", null,
HandleMouseLeftButtonUp);

function HandleMouseLeftButtonUp(x:String, y:String, m:String):Void
{
    // Tratar o evento aqui
}
```

A aplicação *Flash* também pode fazer pedidos para a parte do Visual C#. Tais pedidos podem ser realizados através do registro do evento *Flash*Call. Para exemplificar isto, será mostrado como é feita o tratamento do evento de clique no botão EXIT dentro da aplicação *Flash*.

```
ExternalInterface.call("FlashRequest", “exit”);
```

Dentro da aplicação Visual C#, isto é tratado pelo ***FlashCall***:

```
private void FlashPlayer_FlashCall(object sender,
_IShockwaveFlashEvents_FlashCallEvent e)
 {
      XmlDocument document = new XmlDocument();
      document.LoadXml(e.request);
      XmlNodeList list = document.GetElementsByTagName("arguments");
      if (list != null && list[0] != null)
      {
          this.HandleRequestFromFlash(list[0].FirstChild.InnerText);
      }
 }
```

O método **HandleRequestFrom*Flash*** recebe como parâmetro uma string e trata o pedido de forma adequada, neste exemplo finalizando a aplicação:

```
private void HandleRequestFromFlash(string request)
{
     switch (request)
     {
         case "exit":
         this.ExitApplication();
         break;

         default:
         break;
      }
 }
```

Dependendo do tipo de aplicação se faz necessária a passagem de mais de um parâmetro pela função **call** da **ExternalInterface**. Algo como:

```
ExternalInterface.call("FlashRequest", “mudacursor”, "1");
```

Para tratar mais de um parâmetro é preciso fazer umas pequenas mudanças nas duas funções **HandleRequestFromFlash** e **FlashCall**. Na função **FlashCall,** é necessário inserir um tratamento no arquivo XML que é recebido do evento, que contém os argumentos da função vinda do *Flash*, para pegar os próximos parâmetros. A seguir é mostrado como fica esse tratamento, por exemplo, para três parâmetros:

```
private void FlashPlayer_FlashCall(object sender,
_IShockwaveFlashEvents_FlashCallEvent e)
 {
      XmlDocument document = new XmlDocument();
      document.LoadXml(e.request);
      XmlNodeList list = document.GetElementsByTagName("arguments");
      if (list != null && list[0] != null &&
list[0].FirstChild.NextSibling != null &&
list[0].FirstChild.NextSibling.NextSibling!=null))
      {
          this.HandleRequestFromFlash(list[0].FirstChild.InnerText,
             list[0].FirstChild.NextSibling.InnerText,
       list[0].FirstChild.NextSibling.NextSibling.InnerText););
      }
      else if (list != null && list[0] != null &&
         list[0].FirstChild.NextSibling!= null)
           {
            this.HandleRequestFromFlash(list[0].FirstChild.InnerText,
             list[0].FirstChild.NextSibling.InnerText,"");
           }
           else if (list != null && list[0] != null)
                {
                  this.HandleRequestFromFlash(
                    list[0].FirstChild.InnerText, "","");
                }
 }
```

Nota-se que agora a função **HandleRequestFromFlash** recebe três parâmetros também e não apenas um, como ocorria anteriormente e por isso, ela também deve ser modificada.

```
private void HandleRequestFromFlash(string request, string request2, string request3)
{
     switch (request)
     {
         case "exit":
         this.ExitApplication();
         break;

         case "mudacursor":
         ChangeCursor(int.Parse(request2), int.Parse(request3));
         break;

         default:
         break;
      }
 }
```

Em suma, sabe-se que o SDK *MultiPoint* permite que desenvolvedores criem aplicativos inovadores que tiram proveito de diversos dispositivos de mouse conectados a um único computador. A capacidade de integração com o *Flash* oferece a possibilidade de se construir uma interface gráfica de forma rápida e simples, já que a ferramenta Adobe *Flash* é bastante amigável para desenvolvimento.

3. O APLICATIVO PROPOSTO

3.1. DESCRIÇÃO DO APLICATIVO

Para aplicar a dinâmica de grupo ecológica é necessário criar um contexto específico e, no aplicativo proposto neste trabalho, tem-se por base uma empresa de vendas. Portanto, torna-se essencial a produção de uma “simulação” de trabalho dentro deste tipo de indústria. Esta simulação se dará a partir de uma empresa que deverá ser gerenciada pelos usuários envolvidos na tarefa.

Em um primeiro momento o candidato deverá escolher um personagem que o representará. Esta parte é bastante interessante, pois tem diversas finalidades e dentre elas está a capacidade de aliviar a tensão no ambiente, ou coloquialmente falando de “quebrar o gelo”. Além disso, o psicólogo, condutor da dinâmica, poderá pedir neste momento que o candidato se apresente e também lhe perguntar qual foi o motivo que o levou a escolher determinado personagem, se aquela era a sua primeira opção ou, em caso de mudança de ideia, qual foi o motivo que o levou a tal escolha.

A escolha do personagem vai além de pequenas perguntas, pois a imagem escolhida servirá também na atividade seguinte como cursor do mouse para melhor identificação no decorrer das atividades. As telas da atividade serão detalhadas e apresentadas com figuras na seção 3.5.

Após cada candidato ter escolhido o cursor que o representará, o gerente, psicólogo ou outro membro dos Recursos Humanos (RH) que está no controle do computador, dará início efetivamente à atividade. A atividade deve ser realizada por dois grupos diferentes: Grupo 1 e Grupo 2. O gerente deve decidir previamente quais candidatos irão compor qual grupo.

Cada grupo ficará responsável por setores de uma empresa como Marketing, Vendas e Recursos Humanos e Produção. Os candidatos terão que, por exemplo, comprar insumos de produção, contratar mão-de-obra e tomar decisões sobre o preço de venda, entre outras tarefas, que se adéquem à empresa que está selecionando o candidato.

Será fornecido a cada candidato um *script*, montado pelo próprio psicólogo e impresso em uma folha de papel, que conterá as informações sobre a empresa. Essas informações são sobre o produto de venda, funcionários disponíveis para contratação acompanhados de seus currículos, em suma, todos os dados necessários sobre a empresa. Com base nelas, os candidatos deverão tomar suas decisões.

Esse *script* pode ser flexível, dependendo de quais características a empresa quer investigar no candidato e no trabalho que aquele candidato irá realizar, caso seja selecionado.

Após tempo definido pelo gerente, o mesmo encerrará a atividade. Depois do encerramento será exibida a tela de análise de dados. Esta consiste na exibição de todas as ações realizadas por cada mouse dentro da dinâmica.

3.2. A VISÃO DA PSICOLOGIA

Sabendo-se que a proposta apresentada trata-se de um projeto multidisciplinar, a coleta de informações oriundas do campo da Psicologia iniciou-se a partir de discussões com profissionais da área. Concluiu-se que um software com a ideia descrita anteriormente pode trazer inúmeros benefícios para a técnica de dinâmica de grupo ecológica, tendo em vista que um auxílio desse porte pode trazer para os avaliadores de dinâmicas um grande número de informações agregadas sobre o candidato. Inúmeras vezes, a quantidade de informações obtidas é tão grande que supera as expectativas dos avaliadores e pode ajudar muito para a seleção de um bom profissional para ocupar o cargo disponível.

Tais informações são importantes para avaliação técnica do candidato, uma vez que provoca dificuldades reais do cargo (Andrade, 1999). Para Ivancevich (1995), seleção é o processo pelo qual uma organização escolhe de uma lista de candidatos, a pessoa que melhor alcança os critérios de seleção para a posição disponível, considerando as atuais condições de mercado. Resumindo, como conceitua Chiavenato (1983), a seleção de Recursos Humanos poderia ser definida como “a escolha do homem certo para o cargo certo” para assim poder manter ou aumentar a eficiência e a produtividade. Para que isso seja realizado, o autor sugere ainda que a seleção seja baseada seguindo as características do cargo a ser ocupado para haver maior precisão e objetividade em relação a seleção. Por isso, nesse estudo é ressaltado essa ferramenta que pode ser desenvolvida para atender as características do cargo a ser selecionado.

O software proporciona uma melhor investigação comportamental dos candidatos, não só pelos dados de ações e resultados da tarefa colhidos pelo mesmo, mas também pelas vantagens oferecidas. Uma dessas vantagens é a de que o software permite que o candidato, por estar em um ambiente virtual simulado, possua uma maior imersão e concentração na tarefa que está realizando, diminuindo a tensão provocada pela presença dos profissionais aplicadores da dinâmica de grupo, que muitas vezes são vistos como avaliadores do comportamento.

Com o uso de múltiplos mouses, a investigação se torna ainda mais interessante, pois quando se é dada uma tarefa complexa e elaborada para uma equipe, podem ser notados diversos comportamentos, como, por exemplo, os candidatos se subdividem para atender a tarefa proposta (hierarquicamente ou divisão de tarefas), ou, até mesmo, observar a iniciativa dos candidatos. Ou seja, pensar em como uma equipe se estrutura a partir da possibilidade de todos terem o poder para a tomada de decisão.

Outra vantagem muito importante se deve ao fato de que durante a realização de toda a dinâmica e ao fim da mesma, um psicólogo tem como opções inúmeros tipos de abordagens para extrair mais informações sobre os candidatos. A atividade cria todo um contexto propício para que os psicólogos possam fazer a investigação também através de perguntas e discussões a serem levantadas. Como exemplo, pode-se citar que ao fim da atividade os psicólogos peçam a cada grupo que avaliem o porquê das suas tomadas de decisões e o que mudariam na decisão do grupo concorrente.

3.3. ANÁLISE DE REQUISITOS

Ao conversar com profissionais da área de psicologia, com experiência na análise comportamental e na extração de características do ser humano, foi feito um levantamento de requisitos necessários para a estruturação das características que o profissional de Recursos Humanos deseja que o software possua.

É importante que existam papéis distintos dentro da aplicação: o do gerente e do candidato. Cada um destes possui ações e permissões distintas durante a execução da atividade.

O gerente é aquele responsável por dar início à aplicação, por finalizar a aplicação e é um ator especial, pois ele é o único que possui o controle do teclado.

O candidato é o participante que seguirá as instruções do gerente e será convidado a resolver as atividades propostas. Ele será observado e avaliado a todo momento. Ele poderá decidir de que forma irá tomar as suas decisões como, por exemplo, se irá clicar e modificar os valores das atividades a bel prazer ou se irá discutir com seu grupo sobre as melhores abordagens para realizar a atividade proposta.

Durante qualquer momento da atividade, o gerente pode pausar a atividade, congelando os mouses dos candidatos e evitando que cada mouse tenha o poder de clicar. Esta funcionalidade pode ser útil caso seja interessante interromper a atividade para o gerente fazer observações ou para quando o tempo da atividade terminar.

A partir destas considerações iniciais e da descrição do aplicativo foi feito o diagrama de casos de uso (Figura 2), visto que o mesmo tem a principal finalidade de documentar as funcionalidades do sistema e seus relacionamentos com o ambiente externo composto pelos usuários. O diagrama ilustra de forma clara como cada ator, que neste cenário pode ser representado pelo candidato e pelo gerente, interage com o sistema. Cada ator possui um conjunto finito de ações e cada uma dessas ações deve ser implementada dentro do sistema. É importante haver uma lista de todas as possíveis interações do usuário com o sistema para nenhum cenário seja deixado de lado.

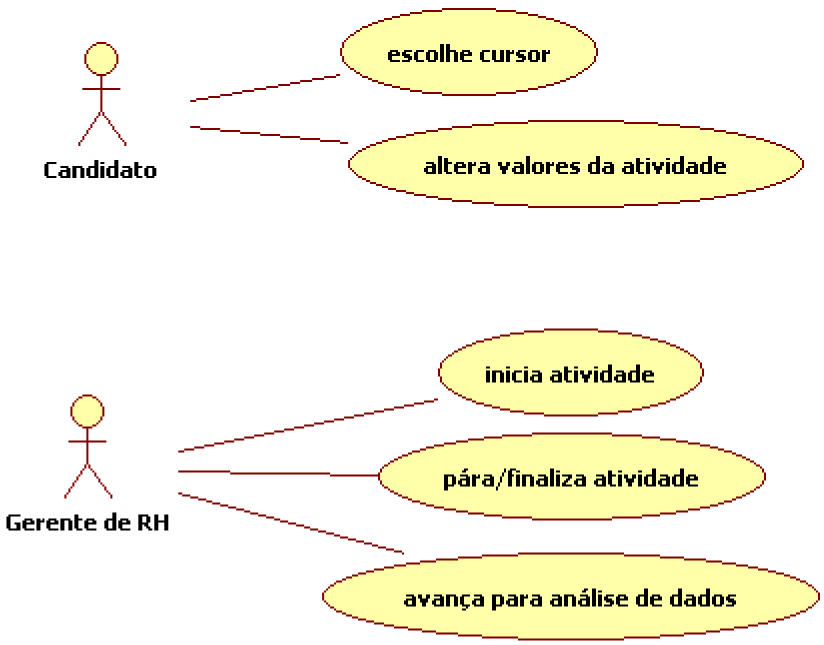

Fig.2: Diagrama de Casos de Uso

Como se pode observar, o diagrama de casos de uso do projeto em questão é um diagrama bem simples, visto que toda interação entre o usuário e a máquina se dá através dos diversos dispositivos de mouse, simplesmente alterando os valores das atividades. Outro fator que torna o diagrama simples é o software não possuir muitos tipos distintos de interações com os atores, mas sim de interações de mesmo tipo que insira o candidato em um contexto propício a questionamentos, discussões e outros tipos de atividades no ambiente exterior ao virtual. Atividades estas que serão propostas diretamente pelo gerente ou psicólogo do RH envolvido.

Após o término da atividade, a próxima tela exibida deve ser a que mostra as estatísticas de cliques de cada mouse. Através desta, será possível avaliar as ações de cada candidato e fazer questionamentos para ele e para o grupo sobre o motivo de seu comportamento.

A partir desta análise primária foi criada uma lista com os requisitos do sistema, para facilitar a concepção do projeto:

- Requisitos Funcionais:

  [RF001] – O gerente deve ser capaz de iniciar, pausar e finalizar a aplicação.

  [RF002] – Cada candidato deve ser capaz de modificar os valores das atividades.

  [RF003] – O sistema deve registrar a quantidade de cliques de cada mouse.

  [RF004] – O gerente deve ter o controle da passagem da tela da dinâmica para a tela de análise de dados.

  [RF005] – O sistema deve prover diferentes imagens para que o candidato escolha uma para representá-lo como um cursor.

- Requisitos Não Funcionais:

  [RNF001] – O sistema deve prover oito imagens de personagens caricaturados de diferentes etnias.

  [RNF002] – O sistema deve possibilitar a interação de até oito mouses na tela.

- Requisitos Inversos:

  [RI001] – O sistema não deve permitir que o candidato pause ou finalize a atividade.

3.4. PLANEJAMENTO E IMPLEMENTAÇÃO

A partir da análise realizada dos requisitos para o projeto, faz-se necessário detalhar todos os passos a serem realizados durante a implementação.

Sabe-se porém que não apenas a criação da atividade é necessária, mas também toda a parte de integração com a tecnologia *MultiPoint*. Tem-se como primeira dificuldade o problema da integração da parte do Visual C# com a plataforma *Flash*. É necessário então algum tipo de diagrama que dê a clara noção de como é feita a integração do projeto. Podemos então representar essa integração com o diagrama de pacotes a seguir.

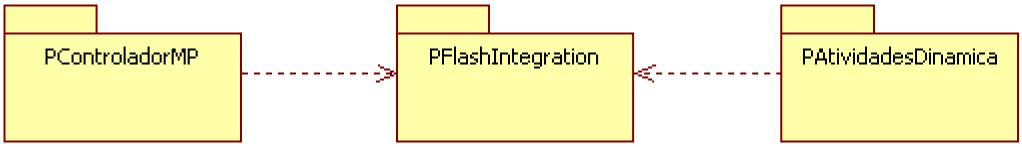

Fig.3: Diagrama de Pacotes

Com este diagrama fica claro a integração da tecnologia com a atividade em si. Exposta também pelo diagrama está a modularização da utilização desta tecnologia. Vê-se o quão separadas estão as classes relativas ao controlador da API MultiPoint, as classes responsáveis pelas funções que fazem a integração com o *Flash* e a atividade da dinâmica em si.

O pacote definido por **PControladorMP** representa as classes implementas em C# e as DLL's do SDK *MultiPoint*, responsáveis por gerenciar os eventos e a renderização dos múltiplos mouses na tela. O pacote **PAtividadesDinamica** contém as classes da parte em *Flash* da aplicação. Estas classes são responsáveis pela dinâmica em si, por armazenar os valores das decisões e por exibir as estatísticas de cliques de cada mouse. As classes deste pacote serão detalhadas a posteriori.

O pacote **PFlashIntegration** é o responsável por possibilitar a integração da API MultiPoint com o *Flash* através da comunicação com Visual C#. Este pacote contém DLL's para comunicação de eventos que serão recebidos, enviados e tratados pelo código C# e pelo *Flash*.

Para o início efetivo da atividade terão que ser levados em consideração os papéis de cada ator dentro do sistema. Sabe-se que o gerente deve ser o único capaz de controlar o fluxo da dinâmica, ou seja, avançar de uma tela para outra, congelar os mouses dos candidatos e finalizar a aplicação.

Tendo em vista esses requisitos, foi definido que todas as ações do gerente serão realizadas através do teclado. Os comandos do teclado estão listados a seguir:

- **"Esc"** - finaliza a aplicação.
- **"Espaço"** - congela/descongela a movimentação e inibe/libera o clique dos mouses.
- **"Enter"** – avança para a próxima tela.

Os códigos que tratam os eventos de teclado estão definidos na parte C#, como o exemplo a seguir:

```
private void KeyDown_Event(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Escape)
    {
        App.Current.Shutdown();
    }
}
```

Seguindo este exemplo é possível fazer o tratamento para quaisquer outras teclas, tratando o evento da forma que convir.

A proposta possui também uma etapa de interação entre a API *MultiPoint* e a atividade feita no *Flash* para a definição do "personagem" de cada usuário, que será representado pelo ponteiro de cada mouse. Esta etapa não é tão genérica como a maioria das aplicações, que simplesmente atribuem cores ou figuras para cada cursor de forma a diferenciá-los. A atividade proposta precisa tratar o cursor de cada usuário de acordo com a escolha do próprio candidato. Vista como mais uma funcionalidade pelo campo da psicologia, a escolha do personagem já é considerada parte da dinâmica para a investigação do candidato.

Como os cursores são renderizados pelo código do Visual C# e a interface gráfica da atividade é feita em *Flash*, precisa-se de uma correspondência das duas partes para o resultado esperado. Levando este fato em consideração foram feitas as seguintes implementações:

- Da parte *ActionScript* do projeto são apresentados os personagens disponíveis para a escolha. Os mesmos permitem que os usuários os selecionem através do clique do mouse. Este evento de clique envia para o código C# qual personagem foi selecionado por qual dispositivo.
- No Visual C# o evento de clique nos personagens é tratado para que se faça a substituição da figura que representa o ponteiro daquele usuário. De forma correspondente, o ponteiro do usuário passa a ser o rosto ou uma característica marcante do personagem escolhido.

A segunda etapa após a escolha de cursores é a atividade da dinâmica em si, demonstrada a seguir. O diagrama de classes apresentado a seguir (Figura 4) retrata de forma geral a organização do código.

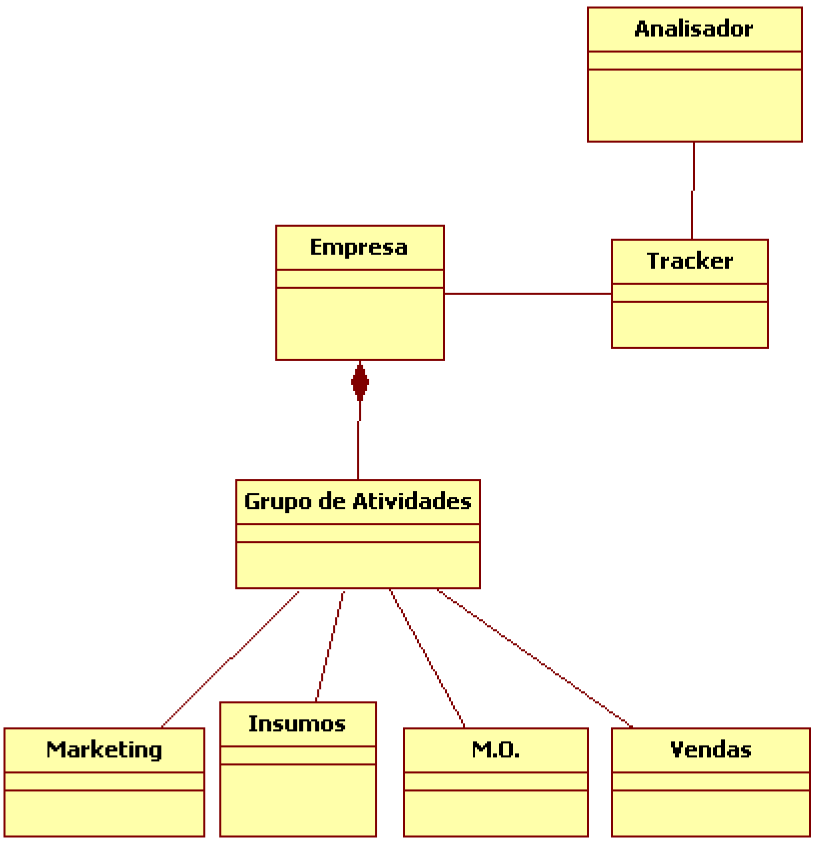

Fig.4: Diagrama de Classes

A partir do diagrama da figura 4 pode-se agora, para um melhor entendimento, apresentar as classes e suas funcionalidades da atividade, justificando a modularização e apresentando-as de forma detalhada:

- Classe **Empresa**: representação da empresa que precisa ser administrada durante a dinâmica. Ela é responsável pela atividade como um todo. Possuindo cada grupo de atividades que devem ser decididas pelos usuários.
- Classe **GrupoDeAtividades**: nesta classe encontramos todas as instâncias de atividades que um grupo deve realizar dentro da dinâmica
- Classe **Marketing**: responsável pelos valores da atividade de marketing e os métodos que permitem a mudança dos mesmos.
- Classe **Insumos**: armazena o tipo de insumos de produção que os candidatos decidirem usar, assim como a quantidade que será comprada dos mesmos.
- Classe **MaoDeObra**: responsável pela atividade que trata da contratação de mão-de-obra. São armazenados aqui valores como a quantidade de funcionários a serem contratados para cada área.
- Classe **Vendas**: representa o setor de vendas da empresa. Nele são tomadas as decisões como preço do produto.
- Classe **Tracker**: armazena os dados coletados dos cliques durante a dinâmica para que possam ser consultados após a sua realização.
- Classe **Analisador**: usada para apresentar de forma organizada na tela os dados coletados que estão armazenados na classe **Tracker.**

É necessário que haja uma tela ao final da dinâmica que exiba o histórico de cliques de cada mouse. O armazenamento destes dados é de extrema importância como funcionalidade do sistema, pois provê um leque maior de questionamentos ao candidato sobre seu comportamento.

Tendo em vista esta necessidade, foi criada uma classe **Tracker**, que funciona como um rastreador, cuja finalidade é armazenar, na sua estrutura de dados, as informações de ID de cada mouse e a quantidade de cliques que o candidato realizou ao longo da dinâmica. Esta classe fornece métodos que retornam os dados detalhados de acordo com uma entrada específica. Isto permite um melhor encapsulamento dos dados para que a modificação da estrutura interna da classe **Tracker** não incorra em efeitos colaterais por parte das classes que forem utilizar esses dados.

Além da gravação do histórico de cliques de cada mouse, outra preocupação que surge é a disposição destas informações na tela. Para gerenciar de forma organizada a exibição dos dados na tela foi criada a classe **Analisador**, que utiliza componentes do *Flash* para exibir as informações de ID do mouse e o número de cliques do mesmo.

As classes supracitadas foram implementadas em *ActionScript* 3.0 na parte *Flash* do aplicativo. É importante notar que a ferramenta *Flash* não é usada somente para interface. Neste projeto, o pacote de classes do *Flash* também possui os *scripts* de armazenamento de informações. Com a atividade definida acima e a integração com a API MultiPoint funcionando já é possível utilizar o *software* para testes ou demonstrações para psicólogos interessados na tecnologia.

3.5. DO PRODUTO FINALIZADO

O produto finalizado apresenta uma divisão em três telas. A primeira tela, e primeiro contato da dinâmica é a escolha dos personagens que representarão os cursores, (figuras 5 e 6). Observa-se que de uma forma bem amigável o candidato pode tomar a sua decisão de qual personagem o representará dentro da atividade.

Fig.5 - durante a escolha de cursores

Fig.6 - após escolhidos os cursores

A segunda tela (figura 7) é o palco onde acontecem as interações da dinâmica, tomadas de decisões de n tipos diferentes dentro da empresa fictícia. Nesta tela, as telas com as decisões dos dois grupos são exibidas simultaneamente.

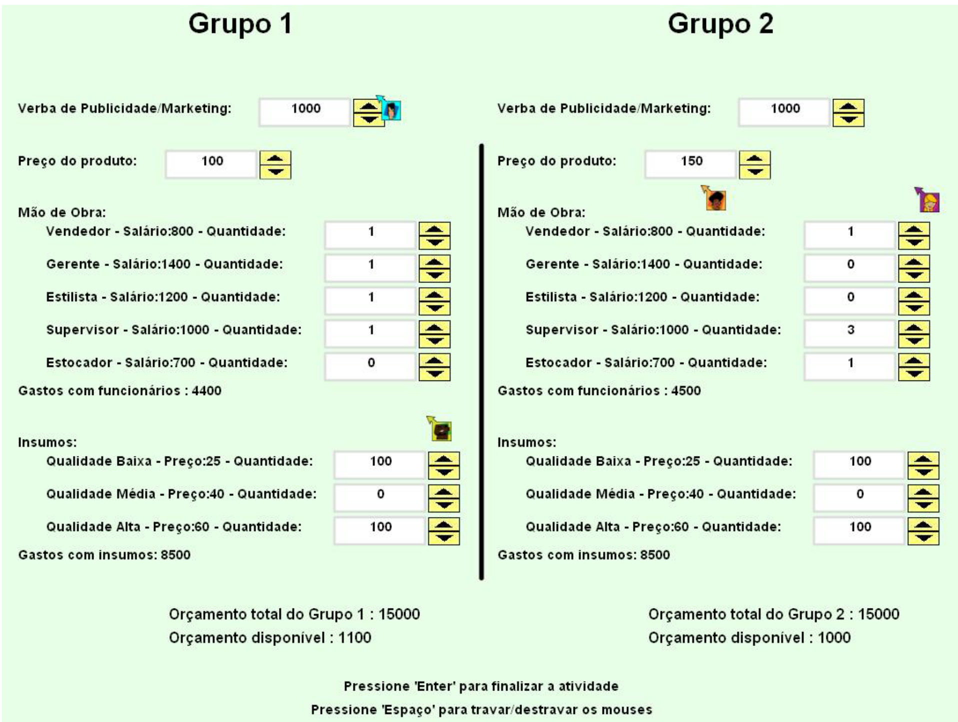

Fig.7 - Durante a atividade

Na terceira tela tem-se a exibição dos dados coletados durante a atividade (figura 8). Essa tela é muito importante e de grande uso para o psicólogo ou gerente responsável pela aplicação da técnica de dinâmica nos candidatos, porque é nela que ele pode observar comportamentos como o de candidatos tomando decisões em um grupo de atividades que não é o seu, ou candidatos que simplesmente não tomaram nenhum tipo de decisão, deixando para os demais do grupo a realização da tarefa.

Fig.8 - Exibição das estatísticas de cliques de cada mouse

3.5.1. RESULTADOS

Após o término da parte de implementação do aplicativo proposto, foi possível disponibilizar o *software* para que profissionais da área de Psicologia pudessem avaliar o resultado final.

A principal observação feita sobre o *software* é que ele proporciona a extração de uma grande variedade de informações sobre o candidato de forma mais detalhada acerca da tarefa propriamente dita realizada pelo candidato. Através dessas observações, os avaliadores podem tirar inúmeras conclusões do comportamento dos candidatos dependendo do seu foco. Desta forma, a primeira impressão desses profissionais em relação ao aplicativo foi de boa aceitação.

Ainda se faz necessário a construção de um cenário específico de uma empresa para que se possa aplicar a dinâmica de grupo ecológica em uma equipe de candidatos. Mesmo assim, os diferentes questionamentos que podem ser feitos ao candidato já se mostraram bastantes satisfatórios para uma boa avaliação comportamental.

Em suma, a aplicação desenvolvida demonstrou atender às expectativas dos profissionais de Recursos Humanos e da área de Psicologia, cumprindo com o propósito para o qual foi criada.

3.5.2. REQUISITOS DE SOFTWARE PARA EXECUTAR A APLICAÇÃO

Para rodar a aplicação é necessário ter instalado o Microsoft *Framework .Net* versão 3.0 ou superior e o *Flash Player* 9 ou mais recente.

3.6. DIFICULDADES ENCONTRADAS

Dada a proposta de utilizar uma tecnologia nova com um propósito diferente do qual ela é comumente utilizada, é natural o surgimento de efeitos inesperados e problemas durante a implementação da aplicação. Deste modo, foi necessário listar as principais dificuldades encontradas ao trabalhar com a tecnologia, como pode ser visto a seguir:

- Problemas ao lidar com os métodos que dependem de eventos nativos do mouse do sistema operacional dentro da linguagem *ActionScript* como, por exemplo, o clique em botões padrão do *Adobe Flash*. Os eventos nativos não são reconhecidos já que o cursor de cada mouse vindo do código C# não é considerado como um ponteiro padrão do sistema operacional, mas apenas uma figura emulando um, e o que seriam os eventos básicos do mesmo, porém estes tratados e sendo enviados internamente ao código C# e não transparente para o usuário sendo enviado para qualquer tipo de aplicação em execução como ocorre com o mouse do sistema. Para a solução desse problema foram utilizadas formas de interação básicas tratadas uma a uma, estimuladas através desses eventos básicos internos ao C# fazendo com que os mesmos sejam enviados para o arquivo *Flash*. Um exemplo disso seria um simples botão que ao invés de seu clique ser tratado automaticamente pelos eventos nativos do mouse do Windows, o mesmo é tratado através de uma mensagem enviada pelo código C# para o *Flash*, que então testa o clique no botão e realiza a função esperada pelo clique.

- Dificuldade gerada pela restrição de cada usuário possuir apenas um mouse mas não possuir teclado. Para algumas das atividades existentes na dinâmica é preciso a inserção de valores e a falta do teclado impossibilita a escolha de um valor qualquer pelo usuário. Como uma primeira abordagem para essa impossibilidade foi adotada a seguinte solução: as atividades possuem opções de valores restritas. Será dado para os usuários dependendo do tipo de atividade cerca de 3 a 10 opções para escolha. Para esse trabalho, essa é uma opção ainda viável pelo seu cunho demonstrativo da ideia principal.

- A integração do *MultiPoint* com o *Flash* depende da classe **ExternalInterface** usada no código *Flash*. Uma limitação que complica bastante a programação dessa interação é a que quando se tem classes instanciadas umas dentro das outras e essas possuem métodos que respondem a uma mesma chamada de função do código externo ao *Flash*, apenas o método da última classe instanciada é executado, sendo desconsiderados os demais métodos das classes instanciadas anteriormente. Para solucionar isso foi necessário tratar a chamada externa apenas em uma classe e através do método que vai ser executado repassar a chamada com todos os parâmetros novamente para as funções internas das outras classes instanciadas. Faz-se assim um efeito cascata de forma a chamar os métodos em todas as classes necessárias.

3.7. PREPARAÇÃO DO AMBIENTE

A disposição dos múltiplos dispositivos de mouse pode ser feita de duas formas: em um posicionamento semelhante a de um laboratório ou utilizando-se um projetor. Na configuração de um laboratório, uma instância da aplicação roda em cada máquina que será utilizada por 4 ou 6 candidatos. Na configuração onde se utiliza um projetor, o computador principal é conectado a um projetor que irá projetar em grande tela visível a todos. Todos os dispositivos de mouse utilizados são conectados ao computador principal através de *USB hubs.* Deve haver uma pessoa responsável pelo computador principal, que controla o computador e inicializa a aplicação.

Em ambientes onde não há o projetor, recomenda-se um máximo de seis pessoas por computador, pois aqueles mais afastados do monitor não tiram máximo proveito da aplicação, pois têm dificuldade em visualizar o monitor. Quando é utilizado o ambiente com a configuração de um projetor, pode-se ter até trinta pessoas simultaneamente interagindo no mesmo terminal. Apesar de o sistema, teoricamente, suportar até 250 pessoas nota-se que um número muito elevado de cursores na tela pode prejudicar o entendimento e a execução do objetivo da aplicação.

O ambiente mais adequado para a aplicação do projeto de uma dinâmica de grupo ecológica é o mostrado na Fig.9. Essa configuração permite que os candidatos fiquem sentados em lados opostos, formando dois grupos e os candidatos estarão próximos fisicamente o suficiente para poderem discutir as decisões a serem tomadas para a realização da atividade.

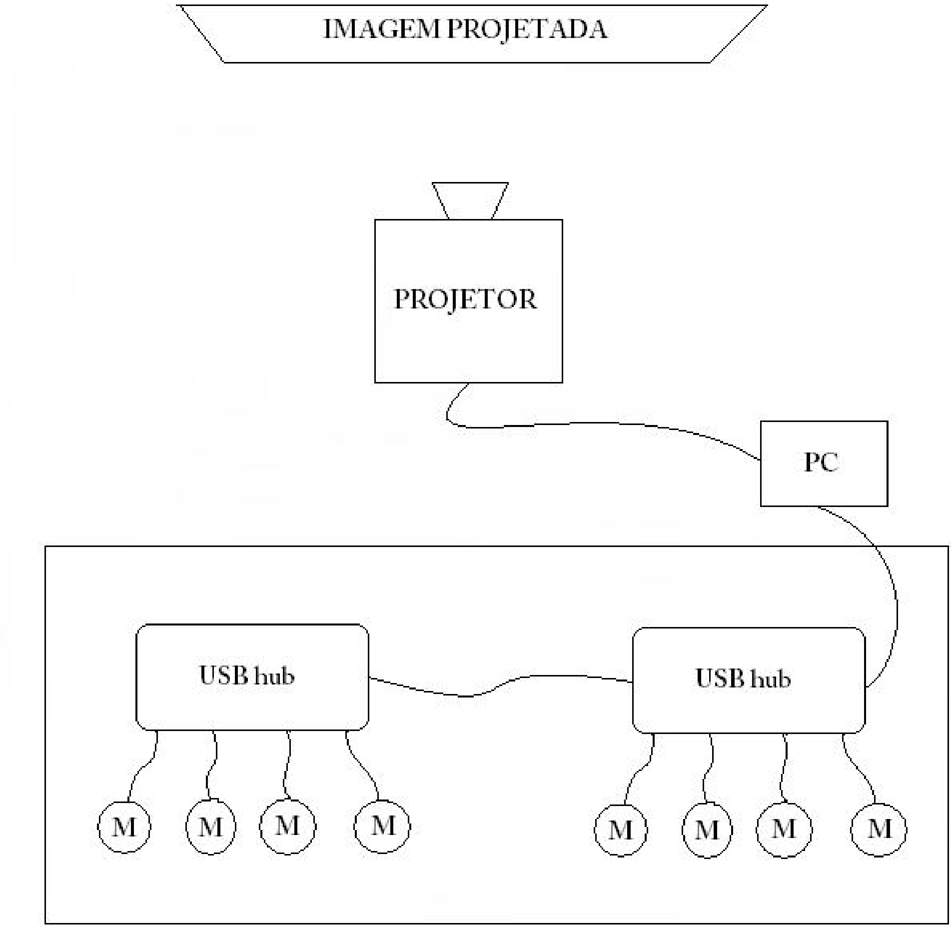

Fig.9: Disposição dos equipamentos no ambiente

Além disto, os candidatos de um grupo também conseguem ouvir as discussões e as decisões do outro grupo de forma que eles poderão, posteriormente, comentar sobre as decisões de cada um.

Enquanto isto, todas as ações feitas por cada candidato serão constantemente exibidas no telão e os avaliadores, neste caso os psicólogos, podem monitorar e tirar conclusões comportamentais dos candidatos a todo momento.

4. UMA ANÁLISE CRÍTICA DA API

A API *MultiPoint* conseguiu atender as necessidades requeridas pelo projeto em questão. Foram feitas com sucesso a inserção dos múltiplos dispositivos e o rastreamento e coleta de dados dos mesmos pelo aplicativo criado. Pode-se dizer que a API realmente faz o que se propõe a fazer, deixando a cargo do programador apenas o tipo de aplicação que utilizará dos seus benefícios.

Uma grande vantagem apresentada pela API é a sua simplicidade e modularidade. A inserção de apenas algumas DLL’s e um simples tratamento padrão inicial são necessários para criar uma aplicação básica.

Apesar de API ter sido considerada satisfatória para o desenvolvimento do software, foi observado que a mesma possui alguns problemas e restrições que podem vir a se tornar um grande empecilho dependendo do tipo de aplicativo que se planeja implementar. Visto que, em geral, estes problemas só se tornam claros quando o programador começa efetivamente a trabalhar com a tecnologia, alguns dos problemas e restrições encontradas são listados a seguir com breves explicações:

- Exigência de tela cheia: pelo modo como a API foi implementada a tela cheia é indispensável para que não existam problemas como aparição do mouse do sistema operacional ao mesmo tempo que os múltiplos dispositivos de mouse criados pela API, gerando confusão para os usuários, e problemas com a posição relativa em que o mouse está clicando para que os eventos sejam tratados de forma correta. Isso traz um grande prejuízo, por exemplo, para aplicações que não tem como objetivo serem executadas em tela cheia ou que desejem usar os múltiplos ponteiros apenas em janelas internas.
- Limitação de movimentação dos cursores na tela: recurso não oferecido pela API. Seria de grande importância para aplicações que tivessem como requisito que determinados dispositivos de mouse fiquem em regiões delimitadas da tela, por exemplo, para evitar que certas áreas da tela sejam clicadas por cursores indesejáveis.
- Problemas no reconhecimento de dispositivos: pode-se dividir esse tópico em dois tipos diferentes:

    - Dispositivos adicionados de forma errada (reconhecimento dos que não estão efetivamente conectados ou que não são dispositivos de mouse): pelo observado, a API detecta os dispositivos de acordo com os *drivers* em uso nas portas USB, a partir destes são filtrados os que são efetivamente dispositivos de mouse e então adicionados. Porém essa filtragem ainda não funciona com exatidão. Alguns dispositivos que não são mouse, quando conectados ao USB, foram entendidos pela API como mouse e então cursores adicionais foram criados na aplicação erroneamente. Outro exemplo são drivers de mesas digitalizadoras que apenas pelo fato de estarem instalados na máquina fazem a API detectar como um mouse mesmo que a mesa não esteja conectada a USB.
    - Dispositivos não adicionados: já que o reconhecimento se dá através da detecção de *drivers* USB, dispositivos que usem outras entradas como entradas PS/2 do PC não são detectados.
- Critério não estabelecido para a ordem de inserção dos dispositivos na aplicação: Não se pode afirmar, até que a aplicação seja aberta, qual dispositivo ganhará qual ID para o seu reconhecimento. Isso faz com que se torne praticamente obrigatório os tratamentos de exceção necessários da aplicação feitos durante a execução da mesma e não previamente. Ex.: Se for de interesse que apenas o mouse do controlador da atividade consiga clicar num botão específico. Ao iniciar a aplicação de alguma forma deve-se descobrir qual o ID desse mouse para passar a informação de que esse ID é que tem o privilégio sobre a ação de clicar naquele botão. Não pode-se afirmar previamente que o ID do mouse do controlador da atividade será um número específico.
- Lentidão: dependendo do tipo de aplicação feita com a API o consumo de CPU da máquina pode ficar sobrecarregado e os dispositivos simulados pelo *MultiPoint* começam a “não responder” de imediato criando “fantasmas” e ações com atrasos. Os “fantasmas” são causados por um retardo na renderização dos cursores na tela.
- Documentação fraca: por motivos desconhecidos a Microsoft não disponibiliza uma documentação completa com todas as classes da API e como elas funcionam, detalhadamente. São encontrados disponíveis apenas documentos com passo a passo de como iniciar uma aplicação e pequenos vídeos tutoriais.

Esse fator prejudica bastante o conhecimento de como a API funciona de fato o que a torna uma caixa preta.

Observa-se então que a tecnologia é de grande utilidade, porém ainda apresenta muitos aspectos que podem ser melhorados para que o leque de possibilidades de utilização da mesma se abra cada vez mais. Segundo a Microsoft, atualizações ainda estão por vir e talvez num momento próximo alguns dos problemas salientados acima estarão sanados e funcionalidades novas estarão disponíveis para que o programador tenha melhores e diferentes opções para trabalhar.

5. CONCLUSÕES E TRABALHOS FUTUROS

5.1. CONCLUSÕES

Foi possível visualizar como a tecnologia Microsoft *MultiPoint* pode ser utilizada para se criar aplicações que suportam múltiplos dispositivos de mouse em uma mesma tela. Utilizando esta tecnologia foi possível desenvolver uma aplicação voltada para dinâmicas de grupo ecológicas.

Para o desenvolvimento desta aplicação, a tecnologia *MultiPoint* atendeu à todas as necessidades porém se mostrou bastante limitada e muitas vezes confusa de se trabalhar. A documentação fraca e os diversos problemas encontrados ao longo do desenvolvimento do aplicativo proposto contribuem para uma experiência negativa com a tecnologia Microsoft *MultiPoint*. Acredita-se que em novas versões do *MultiPoint* os problemas comentados sejam sanados e uma melhor documentação seja feita. Assim, desenvolver aplicações com deste tipo será uma experiência mais positiva.

Para o levantamento dos requisitos do sistema foi necessário um trabalho em conjunto com outras áreas como a Psicologia. Por se tratar de um software voltado para a utilização em análise comportamental, por profissionais de Recursos Humanos e psicólogos, foi necessário esse esforço em conjunto para um correto desenvolvimento e testes das funcionalidades do aplicativo.

Como o aplicativo necessita que um cenário específico de uma empresa seja criado para ser possível aplicar a dinâmica de grupo ecológica, não se realizou testes com candidatos. Além disto, realizar uma dinâmica de grupo gera custos financeiros para uma empresa. Todavia, psicólogos consultados a testar e comentar o aplicativo concluíram que a quantidade de informações que podem ser extraídas com o uso deste é bastante superior a que se tem normalmente. Esse retorno positivo mostrou que o aplicativo tem potencial para até mesmo ser um produto comercialmente viável.

Com base nisto e no que foi visto ao longo do projeto, entende-se que o objetivo do projeto foi alcançado e que o aplicativo proposto cumpre o propósito para o qual ele foi criado.

5.2. TRABALHOS FUTUROS

Os trabalhos futuros têm dois focos principais em relação ao aplicativo apresentado. O primeiro foco se deve as dificuldades encontradas em questões específicas quem envolvem o *MultiPoint* e como contornar da melhor forma essas dificuldades para obter uma interface mais amigável e mais útil. O segundo foco está na inserção de novas funcionalidades na ferramenta, de forma a trazer mais benefícios e um leque de possibilidades muito maior para quem a está utilizando.

Temos então sobre as dificuldades encontradas as seguintes sugestões de trabalhos:

- Criar uma classe que funciona como um interpretador entre os eventos gerados pelos mouses emulados e os eventos gerados pelas funções nativas do mouse padrão do sistema operacional. Isso seria extremamente útil para quem está fazendo uma aplicação utilizando o *MultiPoint* com integração *Flash*, fazendo com que fique automático toda e qualquer ação gerada pelos eventos de mouse sobre os componentes providos pelo *Flash*.

- Tentando solucionar o problema da falta de teclado pra inserção de valores quaisquer, sugere-se criar uma classe que trate de forma genérica essa quantidade de opções e seus valores. De forma a tornar possível que através de um arquivo de dados XML, a modificação fique simples por parte do psicólogo que administra a atividade sem que seja preciso mexer diretamente no código da aplicação. Outra opção seria criar uma espécie de teclado virtual que aparece ao lado da caixa de texto que se deseja editar com as opções de caracteres necessários, podendo ser um teclado numérico ou alfanumérico, por exemplo.

Quanto à melhoria através de novas funcionalidades da ferramenta tem-se para trabalhos futuros a ideia de resgatar os dados extraídos da atividade e exportá-los para um arquivo externo ao ambiente, um arquivo XML por exemplo. Desta forma, é possível proporcionar uma facilidade para que os psicólogos e avaliadores da dinâmica consigam analisar as ações posteriormente em outro ambiente de análise de dados.

Futuramente também deve ser feito um estudo sobre a disposição das informações das atividades na tela, de forma que a atividade fique mais organizada, permitindo uma maior densidade de informações e proporcionando a inserção dos *scripts*, atualmente distribuídos em papel, dentro da atividade sem que prejudique a visualização dos elementos na tela e sem que atrapalhe a realização da atividade por parte dos candidatos.

Além das sugestões de trabalhos relacionadas efetivamente com o desenvolvimento do software, citadas anteriormente, acredita-se que o aplicativo ainda carece de uma análise de usabilidade e experimentos em simulações. Se possível, realizar seleções de candidatos, para que possa ser avaliada a satisfação por parte do psicólogo do uso do software, bem como a sua eficiência.

Assim como a análise e validação do aplicativo proposto de forma aprofundada, sugere-se a criação de novos produtos utilizando a tecnologia adotada, Microsoft MultiPoint, para ressaltar e confirmar que ao ser aplicada em vários contextos ela traz inúmeros benefícios para diferentes nichos de usuários, mostrando assim, todo seu potencial.

6. REFERÊNCIAS

[1] Microsoft Unlimited Potential Website
http://www.microsoft.com/unlimitedpotential/TransformingEducation/*MultiPoint*.mspx
Acessado em: 16 de julho de 2009

[2] TechFlash: Seattle Techonology New Source
http://www.techflash.com/*MultiPoint*_games_put_brothers_in_running_for_big_Microsoft_prize48245562.html
Acessado em: 16 de julho de 2009

[3] Digital Inspiration
http://labnol.blogspot.com/2006/12/attach-multiple-mice-to-one-computer.html
Acessado em: 16 de julho de 2009

[4] United Nations Millennium Development Goals
http://www.un.org/millenniumgoals/
Acessado em: 16 de julho de 2009

[5] *MultiPoint* General Information.doc
Disponível em:
http://www.microsoft.com/unlimitedpotential/TransformingEducation/*MultiPoint*.mspx
Acessado em: 28 de julho de 2009

[6] Issues and Updates in the *MultiPoint* SDK v1.1
Disponível em:
http://www.microsoft.com/unlimitedpotential/TransformingEducation/*MultiPoint*.mspx
Acessado em: 28 de julho de 2009

[7] Getting Started with the *MultiPoint* SDK Version 1.1
Disponível em:
http://www.microsoft.com/unlimitedpotential/TransformingEducation/*MultiPoint*.mspx
Acessado em: 28 de julho de 2009

[8] Developing with the *MultiPoint* SDK Version 1.1: The Map and Quiz Samples
Disponível em:
http://www.microsoft.com/unlimitedpotential/TransformingEducation/*MultiPoint*.mspx
Acessado em: 28 de julho de 2009

[9] Developing *Flash* Applications with the Microsoft *MultiPoint* SDK Version 1.1: The BeezMath Sample
Disponível em:
http://www.microsoft.com/unlimitedpotential/TransformingEducation/*MultiPoint*.mspx
Acessado em: 28 de julho de 2009

[10] Adobe - Help Resource Center - ActionScript 3.0 Reference
http://livedocs.adobe.com
Acessado em: 15 de setembro de 2009

[11] Lt. Col. Chetan Dewan; *Customising Open-Source Rendering Engine for Visual-based Simulation.* Junho 2008.

[12] Joyojeet Pal, Udai Singh Pawar, Eric A. Brewer, Kentaro Toyama; *The Case for Multi-User Design for Computer Aided Learning in Developing Regions.*

[13] Udai Singh Pawar, Joyojeet Pal and Kentaro Toyama; *Multiple Mice for Computers in Education in Developing Countries*.

[14] Customizable educational games for the classroom. Disponível em: http://mymousegames.com/
Acessado em: 03 de novembro de 2009.

[15] CHIAVENATO, I. Recursos Humanos. São Paulo: Editora Atlas, 1983

[16] ALENCAR, E.T.S e GOMES, A. A. Recursos Humanos e a aplicação de testes psicológicos para seleção de pessoal. Apresentação de pesquisa e painel no I Congresso da União Latino - Americana de Psicologia, São Paulo/SP, 2005

[17] IVANCEVICH, Jonh M. Organizações. Comportamentos e Estruturas de Processos. São Paulo: Atlas, 1995.

[18] ANDRADE, S.G. Teoria e Prática de Dinâmica de Grupo: Jogos e Exercícios. São Paulo: Casa do Psicólogo, 1999.