

Universidade do Minho  
Conselho de Cursos de Engenharia  
Licenciatura em Engenharia Informática

## **Laboratórios de Informática IV - Projecto**

Ano Lectivo de 2007/08



escola de engenharia



departamento de  
informática

# **Criação de Animações em SceneBeans controladas por Redes de Petri**

**Márcio André Da Silva Lima (47105)**

**Mário Nuno Ferreira São João (47078)**

**Pedro Emanuel Pereira Soares(47034)**

**Rui Filipe Veiga Rebelo Da Silva(47084)**

Julho, 2008

Data de Recepção	
Responsável	
Avaliação	
Observações	

## **Criação de Animações em SceneBeans controladas por Redes de Petri**

**Márcio André Da Silva Lima (47105)**

**Mário Nuno Ferreira São João (47078)**

**Pedro Emanuel Pereira Soares(47034)**

**Rui Filipe Veiga Rebelo Da Silva(47084)**

Julho, 2008

# Resumo

Este trabalho possui três grandes componentes: XML, Java e CPN. Para criarmos a animação, usamos o XML. É com ele que importamos imagens, definimos acções, movimentos. É a base de teste para o nosso programa em Java, o qual foi construído para simplificar o processo que é fazer animações em XML (dado o seu carácter repetitivo, muita coisa pode ser parametrizada). Para controlar a animação, usamos o CPN. Será ele que vai gerar eventos que desencadeará toda a actividade da animação.

**Área de Aplicação:** Demonstração de cenários animados.

**Palavras-Chave:** Java, XML, CPN, animações.

# Conteúdo

<b>Resumo</b>	<b>i</b>
<b>Conteúdo</b>	<b>iii</b>
Lista de Figuras . . . . .	iv
<b>1 Introdução</b>	<b>1</b>
1.1 Contextualização . . . . .	1
1.2 Apresentação do Caso de Estudo . . . . .	1
1.3 Motivação e Objectivos . . . . .	2
1.4 Estrutura do Relatório . . . . .	2
<b>2 Concepção</b>	<b>3</b>
2.1 SceneBeans . . . . .	3
2.1.1 Animação . . . . .	6
2.2 Java . . . . .	9
2.3 Redes De Petri Coloridas . . . . .	11
2.4 Caso De Estudo . . . . .	16
2.5 Opções Tomadas . . . . .	17
2.5.1 Forall . . . . .	17
2.5.2 Fila De Espera . . . . .	17
2.5.3 Tempo de subida . . . . .	18

<b>3 Conclusões e Trabalho Futuro</b>	<b>19</b>
<b>Bibliografia</b>	<b>19</b>
<b>Referências WWW</b>	<b>21</b>
<b>Lista de Acrónimos</b>	<b>22</b>
<b>A Anexos</b>	<b>23</b>
A.1 Anexo 1 . . . . .	24
A.1.1 Código Comentado - JAVA . . . . .	24

# Lista de Figuras

2.1	Exemplo da animação . . . . .	7
2.2	Exemplo da animação e legenda . . . . .	7
2.3	Declaração de função . . . . .	11
2.4	colset ScenarioInfo . . . . .	11
2.5	colset Agent . . . . .	11
2.6	colset Passenger . . . . .	12
2.7	Transição geTicket . . . . .	12
2.8	Representação da captura de eventos . . . . .	14
2.9	Exemplo de uma transição que espera por um evento . . . . .	15

# 1 Introdução

## 1.1 Contextualização

No âmbito da disciplina de LI4, e de um total de 50 projectos, escolhemos o 6 “Animações em SceneBeans”.

Com este trabalho, o objectivo era construir uma ferramenta que permitisse simplificar o processo de criação de animações em SceneBeans. Estas animações foram criadas para servir de base, para um estudo de um check-in de um aeroporto, sendo a animação deste controlada por uma Rede Petri Colorida.

A animação é uma técnica usada para validação de sistemas de software, permitindo fazer demonstrações para clientes de uma maneira simples de perceber. Tipicamente, o cliente não tem grandes conhecimentos da área de informática, e ainda menos de programação.

É aqui que o SceneBeans (tecnologia JavaBeans) entra em acção, permitindo demonstrar aos clientes determinada situação ou comportamento de alguma aplicação.

## 1.2 Apresentação do Caso de Estudo

Para este trabalho, decidimos usar como caso de estudo, o check-in de um aeroporto. O programa é escalável ao ponto de permitir vários check-ins e vários utilizadores, dando hipótese de demonstrar ao cliente várias situações diferentes.

### **1.3 Motivação e Objectivos**

O principal objectivo é tornar mais fácil a tarefa fastidiosa que é criar animações em SceneBeans. Achamos interessante o desafio de tornar um processo demorado numa acção intuitiva e praticamente automática. Tínhamos também como objectivo adicional a criação de um pequeno exemplo (animação) para o nosso programa.

### **1.4 Estrutura do Relatório**

Este relatório é composto por três partes: Introdução, Concepção e Conclusão. Na introdução fizemos uma breve descrição do problema, para facilitar a compreensão do resto do relatório. Na concepção explicamos detalhadamente o nosso trabalho, os nossos problemas e soluções.

Por último, na conclusão expressamos as nossas ideias depois destes meses de trabalho.

Para além destas três secções principais, temos ainda anexos, um índice, um resumo e outras subsecções.

## 2 Concepção

### 2.1 SceneBeans

O objectivo deste trabalho é simplificar o processo de criação de animações 2D feitas em SceneBeans. Estas são utilizadas para facilitar a compreensão de cenários de utilização de um dado sistema.

O processo de criação de uma animação complexa em SceneBeans pela maneira habitual é bastante trabalhoso. Mais concretamente, terão de ser feitos todos os objectos, o posicionamento inicial e movimentos de cada um deles e finalmente todos os eventos que podem acontecer, tudo isto através de escrita manual de código pouco intuitivo e bastante repetitivo.

Seguidamente, passaremos um excerto de código SceneBeans no sentido de mostrar genericamente a sintaxe do código.

```
1 <animation width="500" height="500">
2
3 <define id ="checkIL">
4   <transform type="translate">
5     <param name="x" value="0" />
6     <param name="y" value="0" />
7     <primitive type="sprite">
8       <param name = "src" value="CIL.jpg"/>
9     </primitive>
10  </transform>
11 </define>
```

Este código cria uma imagem objecto com o jpg “CIL.jpg”.

```

1 <forall var="N" values="0" sep=",">
2   <transform type="translate">
3     <param name="translation" value="(0,200-100*${N})" />
4     <paste object="checkIR" />
5   </transform>
6 </forall>

```

Este código posiciona a imagem anterior no sítio pretendido. No caso de precisarmos de mais imagens destas na animação, o código iria repetir-se, alterando apenas a posição do objecto.

Esta situação também se verifica para movimentos associados a objectos, demonstrado no exemplo seguinte.

```

1 forall var="D" values="1,2,3,4" sep=",">
2
3 <define id ="bag${D}">
4   <transform type="translate">
5     <param name="x" value="0.0" />
6     <param name="y" value="440.0" />
7     <transform type="scale">
8       <param name="x" value="1.0" />
9       <param name="y" value="1.0" />
10      <primitive type="sprite">
11        <param name = "src" value="bag.png" />
12      </primitive>
13    </transform>
14  </transform>
15 </define>

```

Neste excerto, vai-se buscar a imagem, criando-se o objecto “bag”. Os primeiros parâmetros x e y vão definir a posição inicial do objecto. Os seguintes x e y são a escala do objecto. Para cada “D”, é feito um novo objecto “bag”.

```

1 <forall var="N" values="1," sep=",">
2   <seq id="bag_${D}_${N}" event ="Done_bag_${D}_${N}" state="stopped">
3     <behaviour algorithm="move" id="bag_a_${N}_${D}">
4       <param name="from" value="(-135-(${N}*25.0+75.0))*1.0"/>
5       <param name="to" value="(-135-(${N}*25.0+75.0))*1.0"/>
6       <param name="duration" value="0.5"/>
7     </behaviour>

```

```

8 <behaviour algorithm="move" id="bag_b_{$N}_{$D}">
9   <param name="from" value="75.0"/>
10  <param name="to" value="0.0"/>
11  <param name="duration" value="0.375"/>
12 </behaviour>
13 </seq>
14 </forall>

```

Para cada N, cria uma sequência que tem dois movimentos. Este forall será aplicado para outras três sequências de movimento para a bag, alterando apenas a variável value. Este código não foi apresentado por ser repetitivo. No fim teremos N x D sequências.

Dentro do behaviour temos uma origem, um destino e uma duração do movimento.

```

1 <forall var="N" values="1,2,3,4" sep=",">
2   <command name="bag_{$D}_{$N}">
3     <reset behaviour="bag_{$D}_{$N}" />
4     <start behaviour="bag_{$D}_{$N}" />
5   </command>
6 </forall>

```

Aqui é criado um comando para cada movimento de um objecto bag, ou seja, para cada combinação de D com N( D x N comandos diferentes). O reset serve para reiniciar o movimento e o start para o começar.

```

1 <transform type="translate">
2   <param name="x" value="0"/>
3   <animate param = "x" behaviour="bag_b_{$N}_{$D}" />
4   <animate param = "x" behaviour="vbag_{$D}_{$N}" />
5   <param name="y" value="0"/>
6   <animate param = "y" behaviour="bag_a_{$N}_{$D}" />
7   <paste object="bag{$D}" />
8 </transform>

```

Aqui estão a ser aplicados os behaviours ao objecto bag, definindo também a coordenada afectada pelo movimento.

### 2.1.1 Animação

Aqui será explicado como funciona a animação inserida no nosso problema - um check-in de um aeroporto. A animação consiste no seguinte:

O cliente entra no terminal pela porta de entrada seguindo para a sala de espera. Uma vez na sala de espera, o cliente pode aguardar que um check-in fique livre ou então desistir e sair do sistema. Se decidir esperar, assim que tiver um livre, sai da sala de espera e dirige-se para ele.

Uma vez no check-in, procede à amostragem do bilhete e do passaporte que serão verificados e validados pelo agente. Isto corresponde na animação ao aparecimento de um T em cima do check-in; se o bilhete for inválido aparece na animação um T em cima do check-in ao lado do T e abandona o sistema pela porta de entrada; caso contrário aparece um tick verde ao lado do bilhete, procedendo-se à mostragem do passaporte. Esta acção é simbolizada por um P em cima do check-in, sendo a validação do passaporte igual ao bilhete.

Caso o bilhete e o passaporte tenham sido validados o cliente é interrogado por um agente , que na nossa animação é representado por um boneco de cor azul que aparece em cima do check-in, em seguida é dado como apto, sendo mostrado um "print...".O cliente segue para a porta checked out pronto para o embarque.

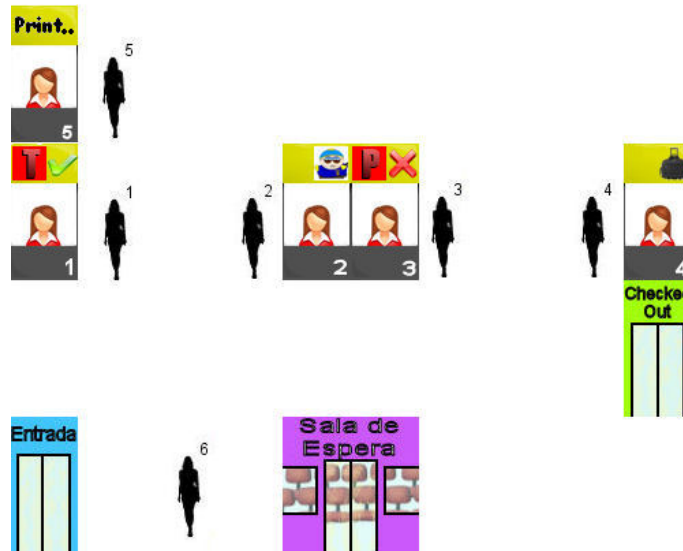


Figura 2.1: Exemplo da animação



Figura 2.2: Exemplo da animação e legenda

Na imagem acima podemos ver um caso em que temos 6 clientes e 5 check-ins.

O cliente número1 encontra-se no check-in1 e apresentou o bilhete que foi validado estando pronto para em seguida mostrar o passaporte, situação essa em que já se encontra o cliente3 que está no check-in3 e ao qual não lhe foi validado o passaporte.

Ao cliente2 que se encontra no check-in2, já lhe foram validados o bilhete e o passaporte estando agora a ser interrogado pelo agente. O cliente4 está no check-in4 e está na fase de entrega da mala no check-in.

O cliente5 que se encontra no check-in5 ao qual já lhe foram verificados passaporte e bilhete e interrogado pelo agente estando por isso pronto a dirigir-se para a porta checked out.

O cliente6 acaba de chegar ao check-in e está a dirigir-se para a sala de espera onde irá aguardar para que algum check-in fique livre.

## 2.2 Java

Nesta secção pretendemos apresentar a razão pela qual resolvemos utilizar a linguagem Java e também explicar sucintamente o funcionamento do programa.

Para a realização deste trabalho optamos por usar a linguagem Java. Esta opção foi tomada em parte, pelo facto de as ferramentas de animação serem também elas baseadas na linguagem Java. Tivemos também em conta o facto de todos os membros do grupo estarem mais familiarizados com esta linguagem, a possibilidade de programar por classes, a facilidade de criação de formulários e manipulação de strings. Foi a junção destes factores que facilitou em certo modo a construção deste programa. Este, é responsável, fundamentalmente pela criação de Strings com as partes de código em XML que vão ser colocadas nos dois ficheiros gerados pelo programa.

Decidimos criar dois ficheiros .XML para termos duas vertentes, uma com o cenário e outra com os vários comportamentos e definições dos objectos.

Para a nossa aplicação ser mais intuitiva e facilmente utilizada, decidimos construir um interface gráfico (formulário) para o utilizador final. Este formulário tem dois campos: números de check-ins e tamanho do set.

Assim, o programa ao receber a informação do número de check-ins que o utilizador quer, calculamos qual o tamanho aconselhável para a animação, ou seja, quantas filas e colunas de check-in são necessárias , existindo as seguintes hipóteses:

Até 12 check-ins:

Tamanho : 500x500

Colunas: 4

Linhas: 3

Até 30 check-ins:

Tamanho : 750x750

Colunas: 6

Linhas: 5

Até 64 check-ins:

Tamanho : 1000x1000

Colunas: 8

Linhas: 8

Após 64 check-ins:

Tamanho: 1000x1000+100\*(cada 8 check-ins a mais)

Colunas: 8

Linhas: 8 + 1 por cada 8 check-ins a mais

Este tamanho é apenas o aconselhado, sendo possível altera-lo para qualquer valor . Apesar de não haver limite de check-ins imposto pelo programa, a partir de um certo número a animação não fica totalmente visível, ou então ao fazer o escalonamento, a animação torna-se imperceptível.

Após ser calculado o tamanho, colunas e linhas os check-ins são dispostos por colunas.

Para animar todos os comportamentos possíveis do caso de estudo, foi preciso definir vários objectos, por exemplo o bilhete, o passaporte, etc.

Após definidos todos os objectos é necessário animá-los, usando também a divisão de comportamentos para cada coluna. Assim sendo, a sua criação é facilitada, pois é necessário ter um movimento para cada check-in de cada objecto.

Após todos os movimentos estarem criados e o seu correspondente código XML também, é necessário criar o código para definir os comandos para cada comportamento.

Ao ser gerada a animação, os comandos eventos e desenho são gerados automaticamente (para o movimento “x” temos o evento “Done\_x” e o comando “x”).

Colocamos na secção Anexos os principais métodos criados, com o respectivo comentário. Esta secção será bastante útil para ter uma melhor e mais pormenorizada percepção do programa.

## 2.3 Redes De Petri Coloridas

A petri-net feita no CPN-Tools [1] controla o estado da animação em cada momento.

Nas declarations estão determinados os colsets, variáveis e funções necessárias para a petri-net.

Os comandos que movimentam a animação no SceneBeans são invocados nas transições da CPN-Tools.

Por exemplo, o comando SceneBeans que controla a animação de entrega do bilhete do cliente ao Check-in é o comando “tick\_D\_N” em que D e N são os números específicos do cliente e do Check-in correspondentes.

Para a CPN-Tools utilizar este comando primeiro é necessário criar uma nova declaração de função nas declarações.

Assim, é criada a declaração:

```
▼ fun mkCmd_tick(s:ScenarioInfo)=  
  let val psgrname = Int.toString(#number (#passenger s))  
      val chin= #ci (#agent s)  
  in "tick_"^psgrname^"_"^chin end;
```

Figura 2.3: Declaração de função

Esta declaração recebe um ScenarioInfo como parâmetro assim definido, contém um Passenger e um Agent.

```
▼ colset ScenarioInfo =  
  record passenger:Passenger *  
  agent: Agent;
```

Figura 2.4: colset ScenarioInfo

O Agent contém duas strings, uma de nome e outra com o ID desse agente (check-in).

```
▼ colset Agent = record name:STRING*ci:STRING;
```

Figura 2.5: colset Agent

O Passenger contem uma string com o seu nome, uma string de números NUMID, um INT que é o numero de utilizador number e um TICKET.

```
▼ colset Passenger= record
name: STRING *
passport: NUMID *
number: INT *
ticket : TICKET;
```

Figura 2.6: colset Passenger

Voltando à explicação da declaração inicial, é criada uma variável psgname à qual é atribuída o numero do passenger descrita no campo number, e também é criada uma variável chin à qual é atribuído o número do agent “ci” numa string.

Estas duas strings são depois usadas para construir uma outra string que é a identificação do comando a ser usado. Seguidamente, a declaração cria diferentes comandos conforme o scenarioinfo que recebe como parâmetro. Esta declaração vai ser invocada na transição getticket a qual tem este código associado desta maneira:

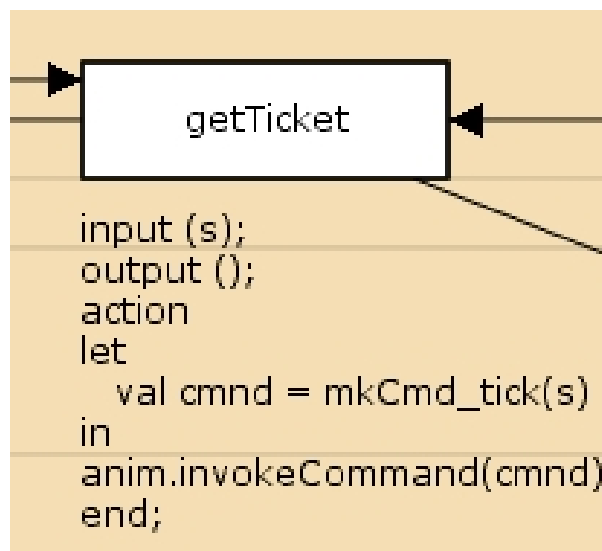


Figura 2.7: Transição geTicket

Este código cria uma variável `cmd` com a string criada pela declaração `mkCmd_tick(s)`, e seguidamente invoca o comando da animação que corresponde à string `"cmd"`, através da invocação `anim.invokeCommand(cmd)`.

Devido ao facto de o `cpn` não conhecer o tempo que demora cada movimento, por vezes ele era iniciado sem o movimento anterior ter acabado. Quando isto ocorria assistíamos, em vez de um movimento natural, a uma mistura dos dois em simultâneo, quando eram movimentos do mesmo objecto, ou então movimentos de objectos fora do contexto por exemplo a entrega do bilhete antes de o passageiro ter chegado ao check-in.

Para resolver este problema optamos por utilizar movimentos que lançam eventos, sendo assim possível capturar os eventos no `cpn` e sequênciá-los. O processo de captura e espera de eventos tem, essencialmente, 4 passos:

1. Utilizar uma transição ("Capture Event") que está sempre à escuta de eventos lançados na animação. Assim que detecta um novo evento adiciona-o à lista de eventos - "Incoming Events".

Esta captura de eventos faz-se com a função `anim.getNextEvent()`.

2. Na transição onde queremos esperar que um evento seja lançado antes de passar para essa transição temos de a ao lugar "Incoming Events" para ter acesso aos eventos já lançados.

3. O lugar "Incoming Events" envia a lista dos eventos a transição.

4. A transição verifica se o evento de que está à espera está na lista recebida. Caso esteja, a transição é efectuada. Caso não esteja, continua a espera. Esta verificação é efectuada com a função `existEvent()` que recebe por exemplo um `ScenarioInfo` e uma `String` com o nome do movimento de que estamos á espera e devolve `True` ou `False`. Dentro da função é criada uma `String` com o evento que queremos verificar. Para isto usa-se a função

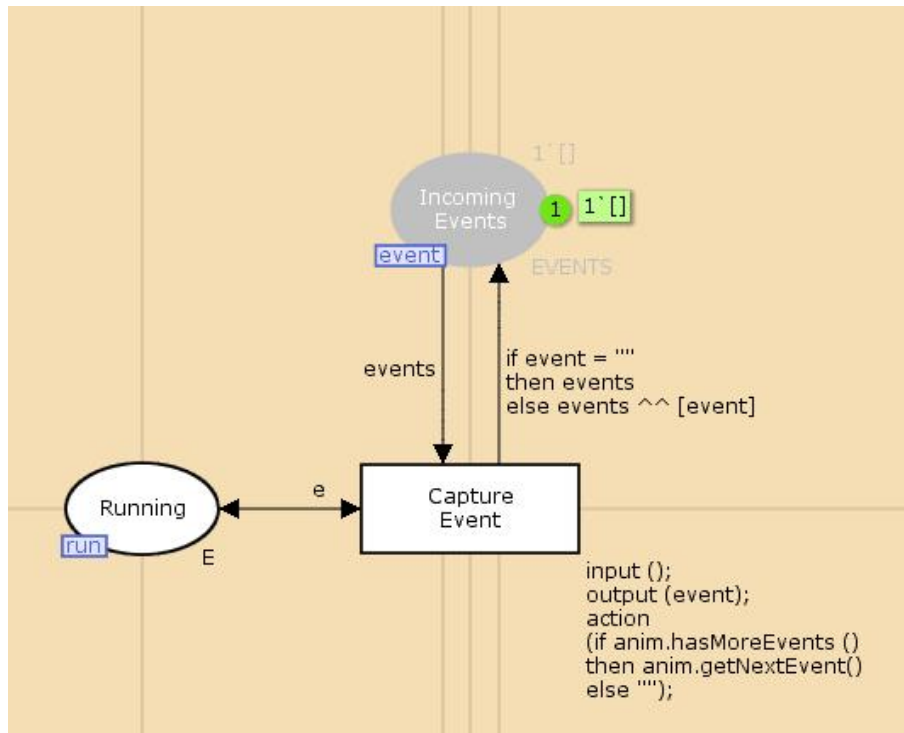


Figura 2.8: Representação da captura de eventos

mkEvent() que recebe um ScenarioInfo e uma String do movimento e cria uma nova String do evento correspondente.

5. Se o evento esperado está na lista "events", a transição retira-o da lista e devolve-a ao lugar "Incoming Events". Para isto usamos a função filter() que recebe a lista dos eventos, a String que identifica o movimento que estamos a espera e também pode receber um ScenarioInfo. A função cria a String do evento com a função mkEvent() já explicada, e faz um filter dessa String à lista de eventos.

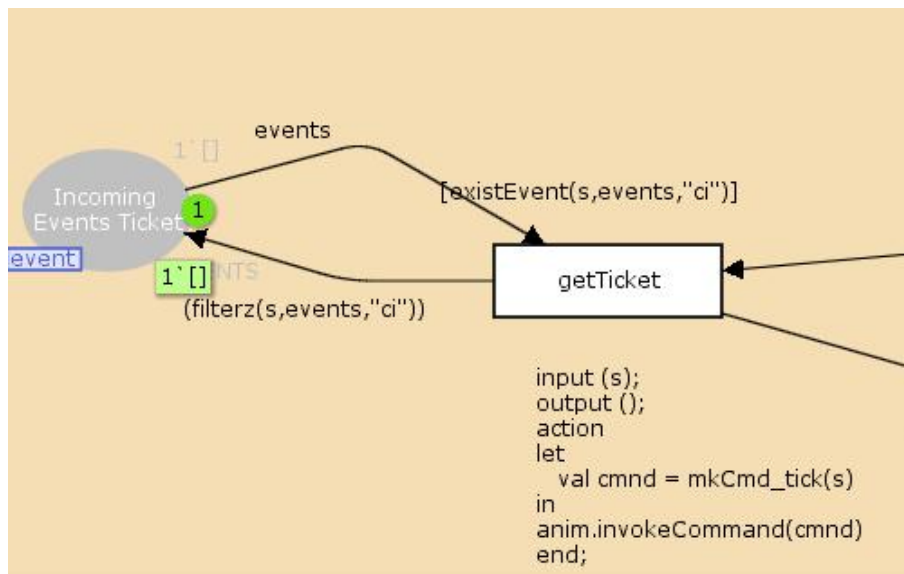


Figura 2.9: Exemplo de uma transição que espera por um evento

No exemplo apresentado a transição getTicket está à espera do evento “Done\_ci\_utilizador\_checkin” para ser efectuada.

## 2.4 Caso De Estudo

Escolhemos para objecto de estudo o check-in de um aeroporto, pois pareceu-nos o que melhor se adaptava ao problema a que nos propusemos, o de criar uma animação.

O check-in do aeroporto tinha muitas mais possibilidades que o problema que inicialmente tínhamos em mente , que era o de um elevador. Como achamos que nos limitava muito as possibilidades optamos pelo primeiro, que é bem mais completo.

Com esta escolha obtivemos a possibilidade de ter uma fila de espera (sala de espera), ocorrências durante o check-in, tal como o bilhete ou o passaporte falhar. A mala inicialmente também ia ser animada, ideia que depois foi posta de parte pois desviava a tenção do foco principal, o check-in.

## 2.5 Opções Tomadas

Nesta secção pretendemos apresentar as nossas opções face aos diferentes problemas que foram surgindo durante a execução deste trabalho. Em cada uma das dificuldades que fomos encontrando, decidimos a melhor de a superar, em grupo.

### 2.5.1 Forall

Nesta secção pretendemos apresentar as nossas opções face aos diferentes problemas que foram surgindo durante a execução deste trabalho.

Uso de foralls:

A utilização de foralls no código da animação tem o benefício de o tornar mais pequeno, mas por outro lado de mais difícil compreensão. Ao usar o forall criamos uma variável e no nome do movimento colocamos, nome do movimento \_  $\{ \text{variável do forall} \}$ . Por exemplo,  $v_{fci\_D}$  ou seja definimos D movimentos  $v_{fci}$ . Com isto evitamos a repetição de enormes quantidades de código, fazendo D várias iterações sobre o que escrevemos. O problema surge quando definimos os valores do movimento . Se eles forem diferentes para cada movimento, o seu cálculo torna-se mais complicado pois temos que usar a variável do forall ao definir os valores, surgindo por vezes expressões como :  $(-150.0 - (\{ N \} * 16.68 + 83.0)) * 1.0$ .

Apesar da desvantagem de complicar bastante a compreensão e elaboração do código, decidimos aplicá-lo. O forall permite, de certa forma, trazer para o XML alguma liberdade e escalonamento, pois foi a partir desta “ferramenta” que conseguimos por um número variável de check-ins e utilizadores.

### 2.5.2 Fila De Espera

O uso da Sala da Espera na animação foi decidido por uma questão de simplificação da animação.

Animar uma fila de espera é complicado pois existe uma enorme quantidade

de movimentos possíveis para cada um dos utilizadores na fila. A opção de não representar estes movimentos e utilizar a sala de espera foi tomada pois a criação dos movimentos para a fila de espera iria tornar a animação ainda mais pesada sem necessidade, visto os movimentos da fila de espera não serem grande ponto de interesse na nossa simulação.

### **2.5.3 Tempo de subida**

O tempo de subida não é importante e numa animação grande a subida de um utilizador até ao último check-in seria demasiado maçadora.

Desta forma, definimos uma constante de um segundo para o tempo de subida, sendo esta independente da distância, não atrasando de forma desnecessária a simulação.

## 3 Conclusões e Trabalho Futuro

O objectivo deste trabalho é a elaboração de uma ferramenta, que simplifique a criação de animações em SceneBeans e, posteriormente, o controlo dessa animação através de redes de Petri.

O caso de estudo escolhido foi o processo de check-in de um aeroporto, pelo facto de ser um exemplo viável e capaz de utilizar as várias propriedades do XML.

Alcançamos este objectivo através da criação de um programa em JAVA, que cria código SceneBeans conforme as especificações do utilizador (é proporcionado ao utilizador a escolha do número de check-ins e de clientes do sistema).

A animação criada é controlada em todos os estados pelas Redes de Petri, e pode também, em todos os momentos, ser visualizado o seu progresso através do BRITNeY.

Com esta aplicação a complexidade da criação de animações SceneBeans torna-se mais rápida e intuitiva, trazendo bastantes benefícios para o utilizador.

Este trabalho permitiu-nos adquirir capacidades no que diz respeito à interpretação de problemas e à maneira como interligamos vários conceitos informáticos (Java, XML e CPN).

Resta-nos referir que nos deu bastante satisfação a realização deste trabalho, dado que enriquecemos o nosso conhecimento nas linguagens utilizadas.

Por fim, gostaríamos de dizer que a ajuda dos coordenadores foi fundamental, particularmente a do coordenador Óscar Ribeiro.

## Bibliografia

- [1] K. Jensen, L. M. Kristensen, L. Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *Int. Journal on Software Tools for Technology Transfer, Springer*, 9(3-4):213–54, 2007.
- [2] O. R. Ribeiro, J. M. Fernandes. On the Use of Coloured Petri Nets for Visual Animation. *CPN*, 237–56, 2007.
- [3] S. Robertson, J. Robertson. *Mastering the Requirements Process*. Addison Wesley, second edition, 2006.

## Referências WWW

01 **<http://www-dse.doc.ic.ac.uk/Software/SceneBeans>**

Página onde para além dos ficheiros necessários à utilização do próprio SceneBeans, tínhamos alguns exemplos práticos e guias de utilização.

02 **<http://wiki.daimi.au.dk/cpntools/cpntools.wiki>**

Web site que contém o CPN Tools, exemplos e informações sobre as Redes de Petri.

03 **[http://wiki.daimi.au.dk/britney/\\_sitemap.wiki](http://wiki.daimi.au.dk/britney/_sitemap.wiki)**

Página que disponibiliza uma o BRITNeY Suite, e alguns dados sobre a aplicação.

# Lista de Acrónimos

**XML** Extensible Markup Language

**CPN** Coloured Petri Nets

## **A Anexos**

Aqui será apresentado algum código que achamos útil para a percepção do nosso trabalho. Apresenta-se sob a forma de comentário uma breve descrição sobre cada método usado.

## A.1 Anexo 1

### A.1.1 Código Comentado - JAVA

#### Classe Animation

Esta é a classe principal para a criação de animações, interagindo com as outras.

```
1 //Variáveis globais:

3 //Classe Checkin
4 Checkin chin= new Checkin();

6 //Classe Behaviour
7 Behaviour bh = new Behaviour();

9 //Quantidade de Check-ins
10 int cis ;

12 //Tamanho da animação
13 int size;

15 //Quantidade de utentes
16 int dudes;

18 //Inicialização da Classe
19 public Animation(){size=0;}

21 /*Métodos básicos que retornam os valores das variáveis
22 utilizadas.*/

24 public int getSize ()
25 public int getCis ()
26 public int getDudes ()
27 public void setCis(int a)
28 public void setDudes(int a)
29 public Checkin getCI ()
30 public Behaviour getBH ()
31 public void setCI (Checkin c)
32 public void setSize(int h)

34 //Código de inicialização da animação para o XML
35 public String toString()
```

```

37 /*Retorna a String com o código dos comportamentos, comandos,
38 eventos e desenho da animação, respectivamente.*/
39     public String behs()
40     public String cmnds()
41     public String events()
42     public String draw()

```

## Classe Behaviour

Esta classe é usada para a criação da maior parte do código do xml "beh.xml".

```

1 //Variaveis Globais:
2 //Nome usado para os passageiros
3     String id="dude";

5 //Quantidade de utentes
6     int total;

8 //Tamanho da imagem usada
9     int im_h=50;

11 //Velocidade dos movimentos
12     float speed=50;

14 /*TreeMap com os comportamentos em x e y do passageiro
15 que vao ser escritos no "draw" do xml da animação
16 usando o 'forall'.*/

18 String -> id , Integer -> quantidade
19     TreeMap<String,Integer > cmnds_xFa
20 = new TreeMap<String,Integer >();

22     TreeMap<String,Integer > cmnds_yFa
23 = new TreeMap<String,Integer >();

25 /*ArrayList com os comportamentos em x e y que vão ser escritos
26 no "draw" do xml da animação.*/
27     ArrayList<String> cmnds_x=new ArrayList<String>();
28     ArrayList<String> cmnds_y=new ArrayList<String>();

```

```

31 /*Método que recebe a String que é o nome dado ao cliente e
32 que retorna o código XML que faz com que seja desenhado.*/
33     public String printDraw(String id)

35 /*Método que recebe uma TreeMap de String ids de um objecto
36 e um ArrayList de String ids dos movimentos associados a esse
37 objecto tanto para x como para y.
38 Retorna o código necessário para o objecto ser desenhado no XML.*/
39     public String printDraw(TreeMap<String,ArrayList<String>> x
40 ,TreeMap<String,ArrayList<String>> y)

43 /*Método que cria o código do draw em forall dos movimentos dos
44 objectos dados como parâmetro.*/
45     public String drawFa(String id,TreeMap<String,
46 Integer > cmnds,char t)

48 //Retorna a String com o código inteiro do draw para o utente.
49     public String drawM()

51 /*Métodos utilizados para alterar/retornar as variáveis globais
52 desta classe.*/
53     public void settotal_cis(int i)
54     public void setCmndsxFa(TreeMap<String,Integer> a)
55     public void setCmndsyFa(TreeMap<String,Integer> a)
56     public TreeMap<String,Integer> getCmndsxFa ()
57     public TreeMap<String,Integer> getCmndsyFa ()
58     public ArrayList<String> getCmndsx()
59     public ArrayList<String> getCmndsy()
60     public void setCmndsx(ArrayList<String> s)
61     public void setCmndsy(ArrayList<String> s)
62     public void setcis(int s)
63     public String getfile()
64     public String getid()
65     public int gettotal()
66     public void setfile(String s)
67     public void settotal(int s)
68     public void setsize(int s)
69     public void setid(String s)

```

### **Classe subBehaviour**

Este método, a par com o anterior, cria código XML para a animação.

```

1 /*Escala que temos que utilizar para obter o tamanho
2 pretendido para a animação.*/
3     float scale_x;
4     float scale_y;

6 //ArrayList com o id dos movimentos do utente.
7     ArrayList<String> cmnds_x=new ArrayList<String>();
8     ArrayList<String> cmnds_y=new ArrayList<String>();

10 /*ArrayList dos movimentos que lançam eventos no fim de executados,
11 gerados automaticamente.
12 Por exemplo, para o movimento "x" cria o evento "Done_x".*/
13     ArrayList<String> events=new ArrayList<String>();

16 /*TreeMap onde cada objecto tem um ArrayList
17 com o id dos movimentos a ele associado.*/
18 String -> id ArrayList<String> -> movimentos
19     TreeMap<String,ArrayList<String>> cmndsvari
20 = new TreeMap<String,ArrayList<String>>();

22 /*TreeMap que tem o movimento e o evento que esse
23 movimento lança.*/
24 String -> id do movimento String -> evento a lançar
25     TreeMap<String,String> s_events
26 = new TreeMap<String,String>();

28 //ArrayList do id dos comandos a ser gerados.
29     ArrayList<String> cmnds=new ArrayList<String>();

31 /*TreeMap com o id dos movimentos e quantos objectos usam esse
32 movimento para gerar comandos usando o forall.
33 String -> id do movimento,
34 Integer -> número de objectos que usam este movimento*/
35     TreeMap<String,Integer > cmndsFa
36 = new TreeMap<String,Integer >();

38     TreeMap<String,Integer > cmnds_xFa
39 = new TreeMap<String,Integer >();

41     TreeMap<String,Integer > cmnds_yFa
42 = new TreeMap<String,Integer >();

44 /*Método que usa o TreeMap s_events para gerar o código
45 que lança eventos.*/

```

```

46     public String gera_s_events()

48 /*Método que usa o ArrayList events para gerar o código
49 que lança eventos de nome automático.*/
50     public String gera_events ()

52 /*Método que cria código usado para formar um objecto
53 à escala, usando as variáveis globais
54 scale_x e scale_y. Este método abre também uma imagem.
55 String id -> id que vai ser usado no objecto.
56 String source -> nome da imagem que vai ser aberta.
57 int init_x,int init_y -> posição inicial onde vai ser
58 colocado o objecto.*/
59     public String abrescaleim(String id,String source,
60 int init_x,int init_y)

62 /*Método que gera código para fazer um objecto a partir
63 de uma imagem.
64 String id-> id que vai ser usado no objecto.
65 String source -> nome da imagem que vai ser aberta.
66 int init_x,int init_y -> posição inicial onde vai ser
67 colocado o objecto.*/
68     public String abreim(String id,String source,
69 nt init_x,int init_y)

71 /*Este método cria código para escalar.
72 float x float y -> valores que vao escalar a imagem.*/
73     public String scale(float x, float y)

75 /*Método que cria código para fazer uma translação do objecto.
76 int x int y -> posição para onde vai ser feito o translate.*/
77     public String translate(int x,int y)

79 /*Método que cria código para fechar operações de
80 transform e define, respectivamente.*/
81     public String c_translate()
82     public String c_define()

84 /*Método que cria código para definir um objecto.
85 String id -> id do objecto*/
86     public String define(String id)

88 /*Método que cria código para usar o forall com valores
89 que aumentam uma unidade em cada valor.
90 String var -> variável que vai ser usada no forall.

```

```

91 int init_value -> valor inicial do forall.
92 int values-> quantidade de valores do forall.
93 int jump -> variaçao dos valores (jump = 2 entao values="1,3,5")*/
94     public String forall(String var,int init_value,int values)
95     public String forall(String var,int init_value,
96 int values,int jump)

99 /*Método que gera código para fazer o include de algo a partir
100 de um ficheiro.*/
101     public String include(String source,String id)

103 /*Método que gera código para fechar operações de include e
104 de forall, respectivamente.*/
105     public String c_include()
106     public String close_forall()

108 /*Método que cria código para criar primitivas usando um ficheiro.
109 String Source -> origem do ficheiro*/
110     public String source(String source)

112 /*Método que cria código para criar primitivas.
113 String type -> tipo da primitiva.
114 String value -> valor da primitiva.*/
115     public String primitive(String type,String value)

117 /*Método que junta vários arrays vindos de uma String
118 numa só String.*/
119     public String arraytoString(String [])

121 /*Método que cria código para iniciar uma sequência.
122 String id-> id da sequência.*/
123     public String seq(String id)

125 /*Método que cria código para iniciar uma sequência
126 que lança um evento.
127     String id-> id da sequência.
128     String event -> id do evento a lançar.*/
129     public String seq(String id,String event)

131 //Método que cria código para fechar operações de sequência.
132     public String c_seq()

134 /*Método que gera código para fazer uma sequência de
135 dois movimentos(com forall), sendo o primeiro em x

```

```

136 e o segundo em y.
137 String id -> id da sequência.
138 String event -> evento a ser lançado pela sequência.
139 int qtd -> quantidade de valores do forall.
140 int fromx -> valor inicial do x.
141 int toInix -> valor inicial destino do x
142 float saltox -> salto que é dado em cada valor do forall.
143 int fromy -> valor inicial do y.
144 int toy -> valor destino do y.
145 int fst -> selecção dos movimentos que usam o forall.
146 int tf -> flag que indica se o movimento é feito da direita
147 para esquerda ou de cima para baixo.*/
148     public String seqFA(String id,String event,
149 int qtd,int fromx,int toInix,float saltox,
150 int fromy,int toy,int fst,int tf)

152 /*Método bastante parecido com o anterior, alterando
153 apenas as variáveis recebidas.
154 String id -> id da sequência.
155 String event -> evento a ser lançado pela sequência.
156 int fromx -> valor inicial do x.
157 int tox -> valor destino do x.
158 int fromy -> valor inicial do y.
159 int toy -> valor destino do y.*/
160     public String seq(String id,String event,
161 int fromx,int tox,int fromy,int toy)

163 /*Método que cria o código para criar um movimento,
164 recorrendo ao forall.
165 String id -> id do movimento.
166 int qtd -> quantidade de valores do forall.
167 int from -> valor inicial do x.
168 int toIni -> valor destino do x.
169 float salto -> salto que é dado em cada valor do forall.
170 int xouy -> flag que indica se o movimento é em x ou y.
171 xouy = 0 para movimento em x.
172 int add -> flag que indica se é para criar
173 comando para este movimento.*/
174     public String bhvrFA(String id,int qtd,int from,
175 int toIni,float salto,int xouy,int add)

177 /*Método igual ao anterior, escolhendo-se agora a duração
178 do movimento.
179 float time -> duração do movimento.*/
180     public String bhvr(String id,int from,int to,int xouy,

```

```

181 int add, float time)

183 /*Método parecido aos anterior, sendo que agora é lançado
184 também um evento gerado automaticamente.*/
185     public String enbhvr(String id, float from, float to,
186 int xouy, int add, float time)

188 /*Método que cria o código para gerar um movimento
189 com tempo dado pelo utilizador. Os valores to e time são
190 dados como string para facilitar o uso em forall,
191 pois os movimentos são mais complexos.
192 Por exemplo: to = "100+({N}*12)". Como o tempo
193 não pode ser calculado (pois varia consoante o valor do
194 forall) tem que ser também submetido.
195 String time -> duração do movimento*/
196     public String bhvr(String id, float from, String to,
197 int xouy, int add, String time)

199 /*Método igual ao anterior, mas com flags.
200 int add -> flag que indica se é para criar comando para
201 este movimento.
202 int addD -> flag que indica se é para adicionar à lista
203 dos desenhados.*/
204     public String bhvr(String id, float from, float to,
205 int xouy, int add, int addD)

207 //Método igual ao anterior, mas com o tempo como parâmetro.
208     public String bhvr(String id, float from,
209 float to, int xouy, int add, int addD, float time)

211 //Método que gera código para os forall, usando o TreeMap cmndsFa.
212     public String gera_cmndsFa()

214 //Método igual ao anterior, mas que usa dois ArrayLists.
215     public String gera_cmnds()

217 /*Método gerador de comandos.
218 String id -> id do movimento que o comando vai executar.*/
219     public String command(String id)

221 /*Método que cria o código para o draw, usando o ArrayList
222 cmnds_x e cmnds_y.
223 String id -> id do objecto a desenhar.*/
224     public String behs(String id)

```

```

226 //Método que adiciona um comando a lista de comandos.
227     public void addCmndVari (String id,ArrayList<String> acmnds)

229 /*Métodos que alteram e devolvem as várias variaveis globais.
230 Também as adicionam aos vários TreeMap e ArrayList.*/
231     public TreeMap<String,ArrayList<String>> getcmndsvari ()
232     public void setcmndsvari (TreeMap<String,
233 ArrayList<String>> a)
234     public void addEvent(String id)
235     public void addsEvent(String id,String event)
236     public void addCmndsFa(String id,int n)
237     public ArrayList<String> getCmnds()
238     public ArrayList<String> getCmndsx()
239     public ArrayList<String> getCmndsy()
240     public TreeMap<String,Integer> getCmndsFa()
241     public TreeMap<String,Integer> getCmndsxFa()
242     public TreeMap<String,Integer> getCmndsyFa ()
243     public void addCmnds (String id)
244     public void setCmnds (ArrayList<String> a)
245     public void setcis(int s)
246     public void setscale_x(float s)
247     public void setscale_y(float s)
248     public String getid()
249     public int gettotal()
250     public void setttotal(int s)
251     public void setid(String s)
252 }

```

## **Class Main**

Nesta classe foram invocadas a maioria dos métodos, para que tivéssemos, no final, um código XML bem estruturado e a funcionar, para posteriormente ser usado na simulação. Todas as as Strings foram gravadas em ficheiro para serem interpretadas pelo SceneBeans.