Instituto Superior de Engenharia do Porto

Departamento de Engenharia Informática

Relatório do Projecto de Licenciatura,

Julho, 2004

Desenvolvimento de um Interface Gráfico em Java Swing

Tema:

Desenvolvimento de um interface gráfico em Java Swing

Autor:

Pedro Vieira da Silva

Orientador:

Eng. Nuno Silva

Data de realização:

Julho, 2004

Desenvolvimento de um Interface Gráfico em Java Swing

Índice

Índice de figuras

Agradecimentos

Quero agradecer a todas as pessoas que me apoiaram durante estes meses, nomeadamente, à minha família e à minha querida M... que me “aturou” nos momentos mais difíceis. Aos meus amigos que sempre demonstraram interesse, curiosidade e apoio no decorrer deste projecto.

Uma palavra especial para o meu orientador, Eng. Nuno Silva, que desde o início depositou grande confiança em mim e sempre demonstrou grande disponibilidade e paciência para me ajudar.

A todos estas pessoas quero demonstrar a minha profunda gratidão...

Desenvolvimento de um Interface Gráfico em Java Swing

1. Introdução

No mundo de trabalho actual, cada vez mais é necessário a adaptação do trabalhador aos métodos de trabalho já existentes e à reutilização dos meios disponíveis de forma a satisfazer as suas necessidades. É nesse sentido que este deve estar preparado para escolher e recolher correctamente informação de forma a efectuar uma análise do ambiente que o rodeia para, posteriormente, estar ciente das ferramentas que possui para implementar os seus próprios métodos de trabalho, conforme os seus objectivos.

Este projecto visa, essencialmente, um estudo profundo de uma ferramenta já existente, o *MAFRA Toolkit*, que será determinante para a implementação posterior de novas soluções.

Este estudo compreende várias fases que determinam o decorrer deste projecto:

- A preparação dos requisitos de software e a instalação dos mesmos;
- A análise da implementação da estrutura do mafra-toolkit;
- A análise do comportamento da aplicação a nível de utilizador;
- A especificação e determinação de comportamentos e/ou anomalias observado(a)s;
- A especificação de soluções teóricas possíveis;
- O estudo e pesquisa das razões desse comportamento a nível de código e estrutura de classes;
- A especificação prática de soluções e implementação das mesmas;
- O desenvolvimento do espírito crítico, garantindo assim o aparecimento de novas soluções no sentido de melhorar as implementações já existentes.

Este relatório é composto por mais 4 capítulos. No capítulo seguinte descreve-se a aplicação subjacente ao projecto, sendo dividido logicamente entre uma abordagem à aplicação propriamente dita, seguida duma componente descritiva de noções básicas sobre a aplicação.

Segue-se um capítulo de análise de comportamentos observados a nível de utilizador e por fim o capítulo de desenvolvimento onde são apresentadas implementações de soluções relacionadas com o capítulo de análise.

2. Mafra

2.1 A Aplicação

MAFRA Toolkit (***MA****pping* ***FRA****mework Toolkit*) é uma aplicação que permite a criação de relações semânticas entre uma ontologia fonte e uma ontologia destino. As relações semânticas são especificadas a um nível conceptual e serão usadas a nível dos dados para transformar instâncias definidas de acordo com a ontologia fonte em instâncias da ontologia destino.

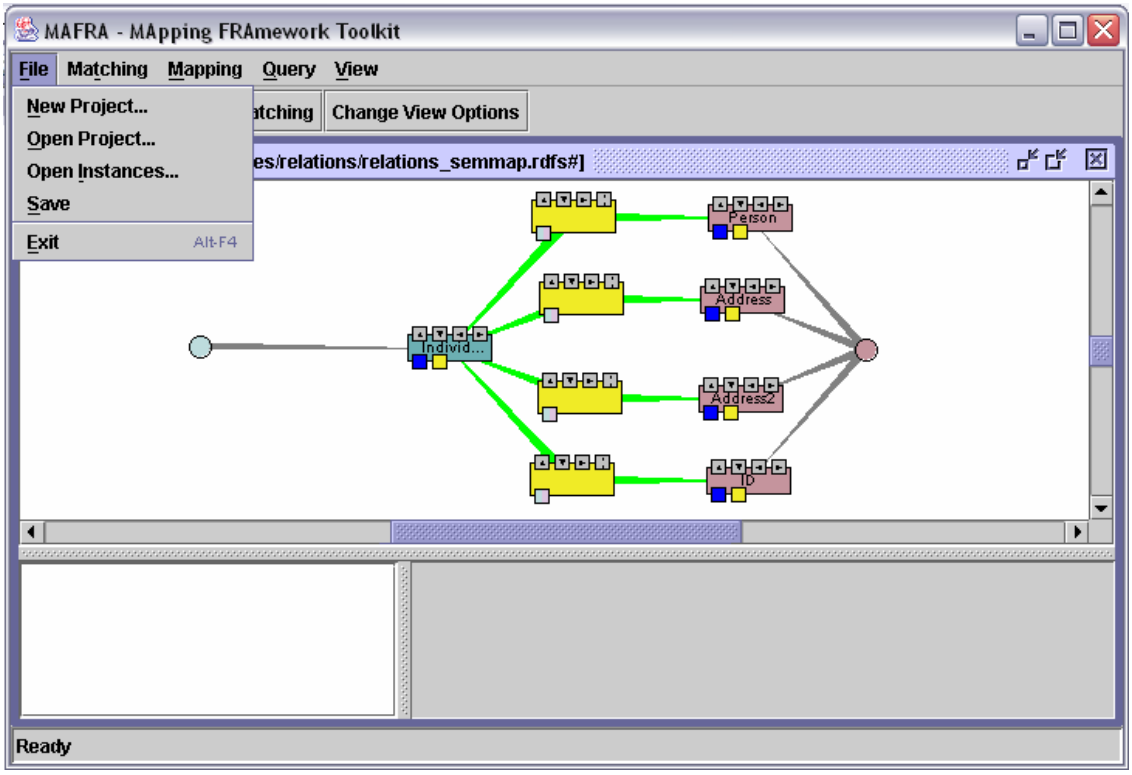

Figura 1 - *Screenshot* do Interface do MAFRA Toolkit

A Figura 1 representa o interface da aplicação MAFRA Toolkit onde é possível observar um exemplo de um projecto aberto.

O MAFRA Toolkit é uma implementação específica do MAFRA - *Mapping FRAmework. O objectivo do MAFRA* é a caracterização e interligação de todas as fases do processo de mapeamento de ontologias, incluindo análises prévias das ontologias e das suas semelhanças, especificação, representação, e execução do mapeamento. Inclui também fases complementares ao processo fundamental como seja a evolução, negociação e análise dos resultados da execução. Todo este processo está representado pela Figura 2.

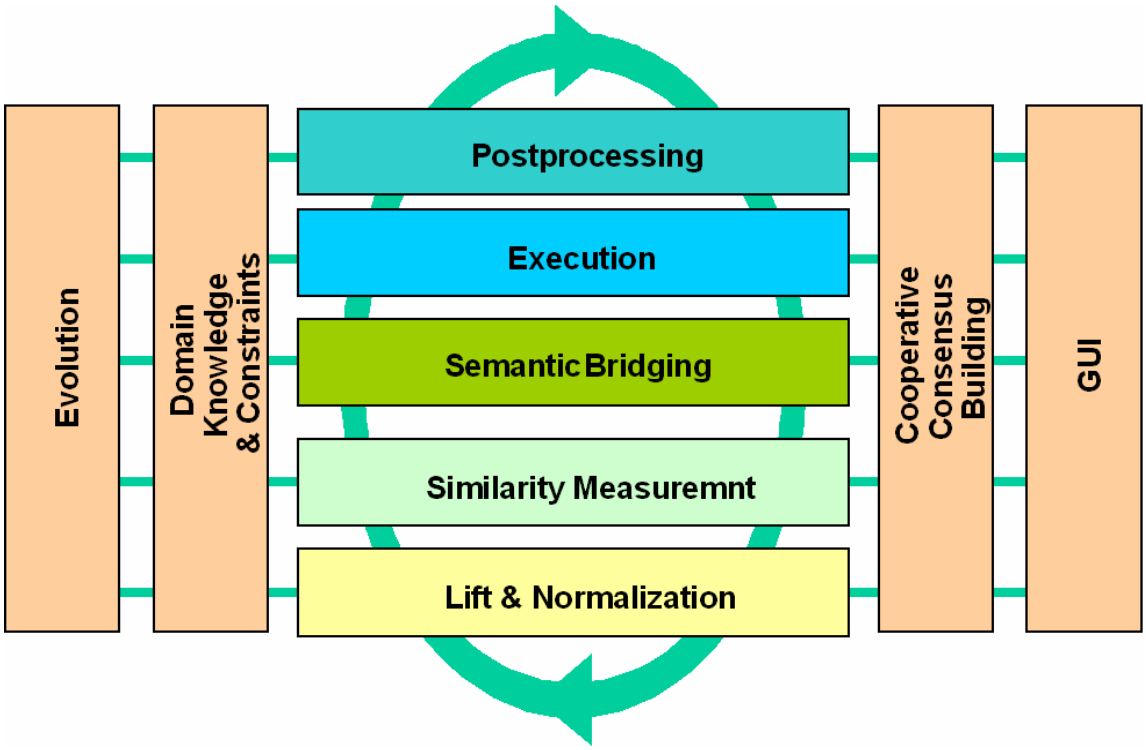

Figura 2 - MAFRA – *Mapping FRAmework*

O MAFRA Toolkit adopta uma arquitectura aberta por forma a observar, maximizar e responder aos requerimentos da Web Semântica (*Semantic Web*), nomeadamente heterogeneidade, evolução e distribuição.

Uma das capacidades do *MAFRA Toolkit* diz respeito à sua abordagem orientada ao serviço que defende que as capacidades de um sistema de mapeamento de ontologias depende do tipo de transformações existentes no sistema de transformação.

Os serviços são depois responsáveis pelas transformações das instâncias, mas também proporcionam suporte para outras tarefas de mapeamento de ontologias como especificação automática de relações semânticas, de negociação e de evolução.

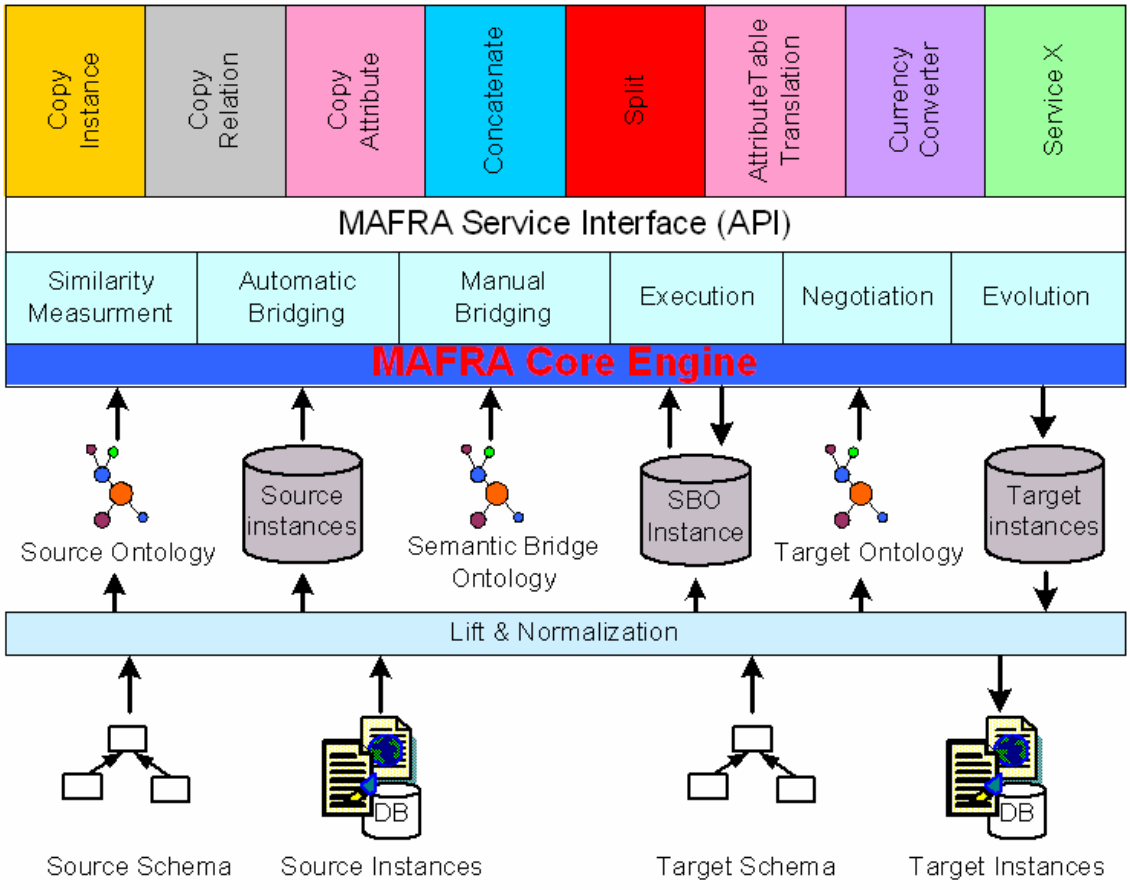

Figura 3 - MAFRA Toolkit - Arquitectura Orientada ao Serviço

A Figura 3 demonstra a arquitectura orientada ao serviço em que se baseia o MAFRA Toolkit.

2.2 Noções Básicas

No MAFRA existe uma série de conceitos básicos que são necessários considerar antes de se proceder à análise e desenvolvimento. Esse conceitos incidem sobre as entidades gráficas que são apresentadas ao utilizador com as quais este pode interagir criando, alterando e interligando-as entre si de forma a definir os seus comportamentos de acordo com os seus objectivos.

As próximas secções descrevem estas entidades quer em termos de componente conceptual quer em termos de representação gráfica, que é aquela que é relevante neste projecto.

2.2.1 Ontologies

A definição mais comum de ontologia, definida em 1984 por Thomas Gruber, é:

"ontologia é a especificação explícita duma conceptualização"

Desde então tem vindo a ser continuamente posta em causa, quer em termos de conteúdo (não diz tudo o que deveria dizer) quer em termos práticos (as implementações existentes não contemplam nem têm objectivo de contemplar tudo o que contém).

Como tal, no contexto do MAFRA e do MAFRA Toolkit esta definição foi substituída por:

"Ontologias é uma especificação explícita, formal e parcial duma conceptualização partilhada"

Em termos práticos, o conceito de ontologia varia drasticamente de aplicação para aplicação de comunidade para comunidade ou mesmo de autor para autor. Várias vertentes distintas existem em domínios como a Semantic Web ou nos sistemas baseados em conhecimento.

Fazer uma descrição exaustiva de conteúdo e forma de ontologia nas suas diferentes vertentes não faz parte dos objectivos deste projecto e como tal faz-se apenas uma descrição do tipo utilizado no MAFRA Toolkit, uma vez que o MAFRA tem uma abordagem muito mais genérica.

2.2.2 Concepts

Embora se esteja a tratar com ontologias é possível, analogamente, estabelecer um paralelismo entre um *concept* e uma tabela de uma base de dados ou mesmo até entre este e uma classe de objectos. Na verdade, em diversas linguagens de representação de ontologias (ex. RDFS, DAML+OIL, OWL) é adoptado um modelo hierárquico e orientado a objectos (*concepts* herdam propriedades dos seus *super concepts*).

Assim como uma classe pode derivar de outra e vice-versa, o *concept* pode também ter *super-concepts* e *sub-concepts*. Embora se esteja a fazer esta analogia, a natureza dos *concepts*, no contexto deste projecto, não tem comportamentos nem características idênticas ao de uma classe em linguagens de programação orientadas a objectos.

Um *concept* possui também *Properties To*(Para) e *Properties From*(De), assim como *Matches* e *Concept Bridges* com que está relacionado em diferentes níveis do processo de mapeamento. Estas entidades irão ser abordadas posteriormente em 2.2.4.

Graficamente, um *concept* é representado através dum rectângulo, tal como apresentado na Figura 4.

Figura 4 - Representação Gráfica de um Concept

O rectângulo adquire uma cor específica de acordo com o facto de se tratar duma entidade da ontologia fonte ou da ontologia destino. O quadrado tem diversos componentes de manipulação das características do *concept* representados por pequenos botões icónicos da função que executam:

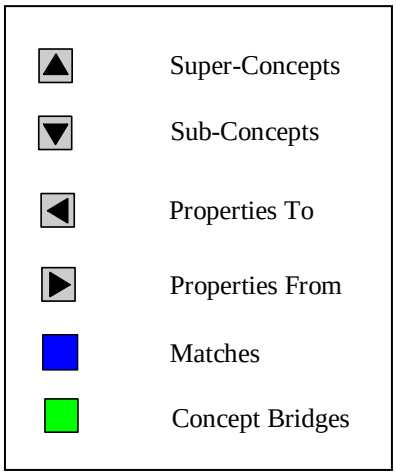

Figura 5 - Botões de um Concept e Respectivas Funções

Cada um destes botões representados pela Figura 5 corresponde a uma chamada gráfica de visualização de modo a mostrar/esconder as ligações que este *concept* possui às entidades relativas ao botão seleccionado.

Por exemplo, se for pressionado o botão verde ir-se-á visualizar todas as *concept bridges* que estão ligadas a este *concept*.

2.2.3 Properties

Uma *property* tem de estar sempre relacionada com um *concept* e, como o próprio nome indica, representa uma propriedade do mesmo. Pode-se dar como exemplo, o caso de um *concept* representar um indivíduo, é possível afectar-lhe uma *property* que represente a idade desse indivíduo.

No caso do MAFRA Toolkit, que utiliza uma linguagem de representação de ontologias baseada no RDFS, as *properties* são entidades de primeira ordem. A principal consequência é a modelação centrada na propriedade. Isto é, é a *property* que define quais os *concepts* que interliga. O *concept* em que é definida é o *Domain Concept* e o *concept* valor o *Range Concept*.

Graficamente *Property* tem a forma hexagonal e possui três botões (Figura 6).

Figura 6 - Representação Gráfica de um Property

O botão azul tem a função idêntica ao de um *concept* que serve para mostrar/esconder todos os *matches* que esta *property* possui.

Dos dois restantes botões acima, o que representa uma seta para a esquerda utiliza-se para mostrar/esconder os "*Domain Concepts*", ou seja, os *concepts* que têm esta propriedade. No botão que representa a seta para a direita são mostrados/escondidos os "*Range Concepts*".

2.2.4 Semantic Bridges

As *Semantic Bridges* são um dos elementos da ontologia representados graficamente mais importantes do MAFRA Toolkit, e ainda mais atendendo ao objectivo deste projecto. Elas contém informação não só das entidades origem e destino como também das especificações, condições, correspondências e transformações que as entidades origem irão sofrer.

Como a própria designação indica, estabelecem pontes semânticas entre as ontologias.

2.2.4.1 Concept Bridges

Basicamente estabelecem pontes de interligação entre *concepts* e assim como estes, também podem ter *super* e *sub-bridges*. *Concept Bridges* são responsáveis por criar instâncias de *concepts* da ontologia destino. Através dos botões, delas podem também derivar as *property bridges* e as *In-Alternative Bridges* (não irão ser abordadas, visto não estarem no âmbito deste projecto). Possuem também um botão para mostrar/esconder os *concepts* que lhes estão relacionados (Figura 7) .

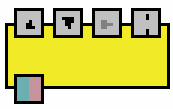

Figura 7 - Representação Gráfica de um Concept Bridge

2.2.4.2 Property Bridges

Estabelecem pontes de interligação entre *properties* e é nelas onde estão especificadas as transformações propriamente ditas. *Property Bridges* são responsáveis por criar *properties* valor nas instâncias dos *concepts* criados através de *Concept Bridges*. Por esse motivo, *Property Bridges* são associadas e são executadas no contexto de *Concept Bridges*.

Em exemplo concreto, imaginando que se estava perante uma *property* chamada "info" e uma outra chamada "nome" ambas derivadas de um *concept* origem e um *concept* destino respectivamente. Pretendia-se efectuar uma transformação de modo a que o conteúdo de "info", onde está a *string* "Aa Bb", fosse copiado para o conteúdo de "nome" onde a transformação seria a eliminação de espaços nessa *string*. O resultado no conteúdo da *property* "nome" seria: "AaBb". Este exemplo concreto serve para demonstrar a aplicação de um serviço da *property bridge* que neste caso seria uma operação simples a nível de *strings*.

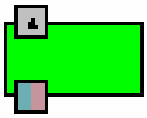

Figura 8 - Representação Gráfica de um Property Bridge

Como a *property bridge* está associada à *concept bridge*, possui um botão para mostrar/esconder as *concept bridges* às quais pertence.

O botão bicolor mostra/esconde todas as *properties* ligadas a esta *property bridge* (Figura 8).

2.2.5 Edges

Graficamente, os *Edges* servem para representar as várias ligações e seus tipos entre todas as entidades. O exemplo da Figura 9 mostra vários tipos de *Edges* a interligar entidades:

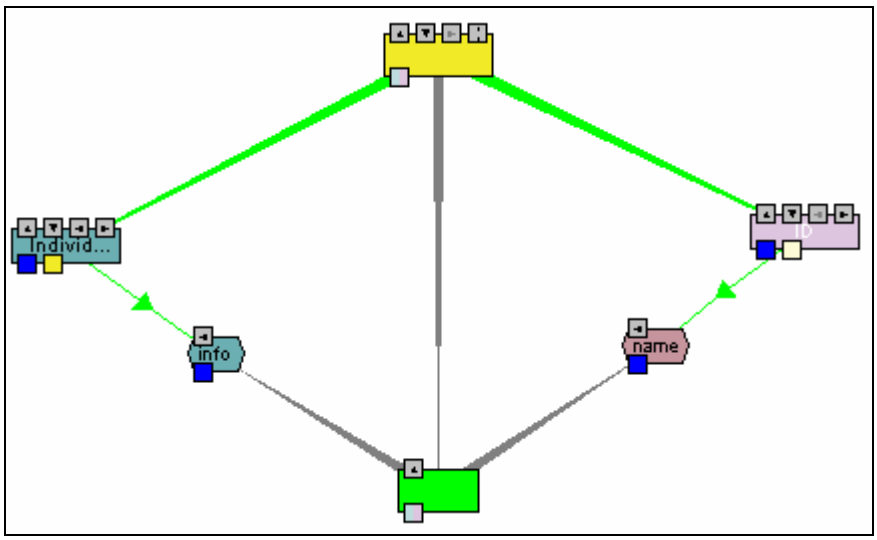

Figura 9 - Representação Gráfica de Ligações entre Entidades

Neste exemplo pode-se visualizar uma *Concept Bridge* ligada por três *Edges* a 2 *Concepts* e uma *Property Bridge*. Por sua vez, de cada um dos *Concepts* derivam as *Properties* "info" e "name" respectivamente estando depois estas *Properties* ligadas à *Property Bridge*.

2.2.6 Matches

Matching é uma outra forma de ver as ligações que há entre as entidades. Para isso, existe uma outra entidade, o *Match*, que interliga a entidade origem e entidade destino. Neste tipo de ligação a visualização torna-se mais simples visto que não são especificadas as características da ligação ficando somente especificada a origem, destino e a lista de *semantic bridges* correspondentes. Ou seja, um *Match* corresponde a uma relação mais simples e, portanto, incompleta entre entidades de ambas as ontologias. Ao contrário de *Semantic Bridges* em que um conjunto (de cardinalidade

variável) de entidades da ontologia fonte podem estar relacionadas com um conjunto (de cardinalidade variável) de elementos da ontologia destino, a cardinalidade dum *Match* é sempre 1:1. Ou seja, um *Match* representa a ligação duma e uma só entidade da ontologia fonte a uma e uma só entidade da ontologia destino. Pode ser ainda referido que para expandir um M*atch* é possível fazê-lo tanto através da origem como do destino. A Figura 10 representa um *Match* esquematicamente.

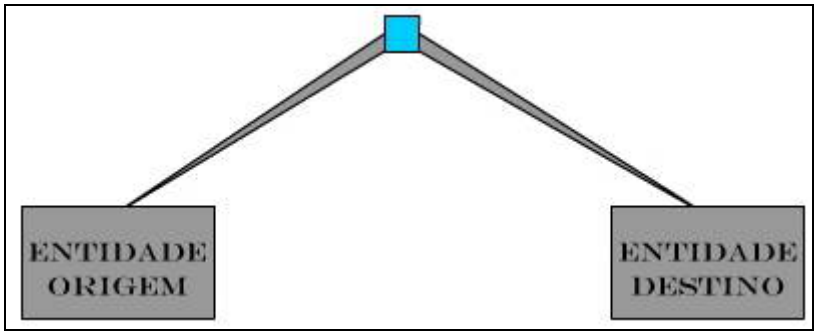

Figura 10 - Representação Esquemática de um Match

Os *matches* podem ligar 2 *concepts* (Figura 13), duas *properties* (Figura 12) ou uma *property* e um *concept* (Figura 11).

A mesma entidade (*property* ou *concept*) pode estar ligada por um ou mais *matches*.

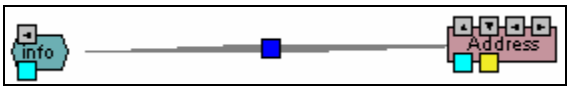

Figura 11 - Rep. Gráfica de um Match (Property - Concept))

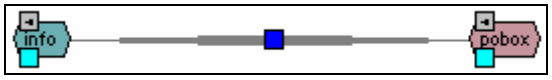

Figura 12 - Rep. Gráfica de um Match (Property – Property)

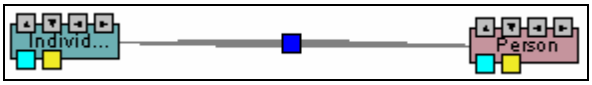

Figura 13 - Rep. Gráfica de um Match (Concept – Concept)

2.2.7 Interacção com as Entidades Gráficas

Uma das funcionalidades do MAFRA Toolkit relativamente à interacção com o utilizador, é a capacidade de interagir com as entidades gráficas através do botão do lado direito do rato, afectando a(s) entidade(s) seleccionada(s) ou mesmo comportamentos mais generalizados.

Afecto a uma ou mais entidades seleccionadas, através desta funcionalidade, o utilizador tem ao seu dispor 3 opções:

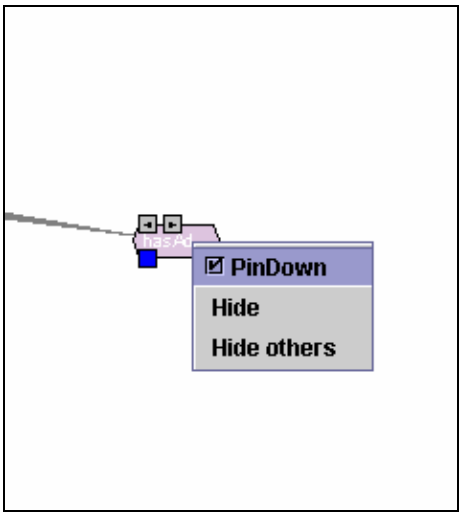

Figura 14 - Interacção com Entidades

Pela Figura 14, a primeira opção permite que o utilizador fixe a posição de uma entidade no espaço gráfico, isto é, o processamento do *layout* gráfico das entidades passa a ser feito tendo em conta a posição dessa entidade escolhida pelo utilizador, adaptando-se à mesma.

A segunda opção permite esconder uma ou mais entidades seleccionadas.

A terceira opção permite esconder todas as entidades exceptuando as que foram seleccionadas.

O aplicação oferece ainda uma outra série de opções que não dizem respeito a uma entidade específica mas sim a comportamentos de visualização e a afectação de grupos mais gerais de entidades. Essas opções estão expressas na Figura 15.

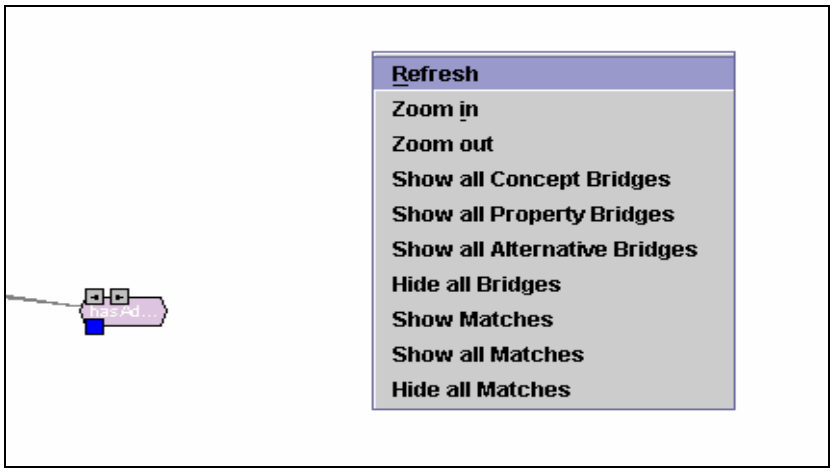

Figura 15 - Interacção com o Interface Gráfico

3. ANÁLISE

Uma vez preparado o cenário para a utilização em concreto da aplicação MAFRA Toolkit e reunidas todas as condições para que seja possível trabalhar correctamente com a mesma, este capítulo visa, essencialmente, analisar comportamentos observados no MAFRA Toolkit. Embora muitos desses comportamentos não sejam necessariamente comportamentos errados, a realidade é que podem ser observados comportamentos nem sempre desejados pelo utilizador. Em seguida, proceder-se-á à decomposição e descrição individual de cada um deles.

É ainda relevante referir que este capítulo se caracteriza apenas por se cingir à análise de comportamentos e problemas não apresentando quaisquer soluções. Ficando essa matéria relegada para o capítulo seguinte que trata exactamente da abordagem prática às questões que surgem desta análise.

3.1 Criação de projectos

A fase inicial de análise, começa por uma identificação de problemas mais básicos relacionados com a aplicação. Um dos primeiros problemas observados está relacionado precisamente com a criação de projectos.

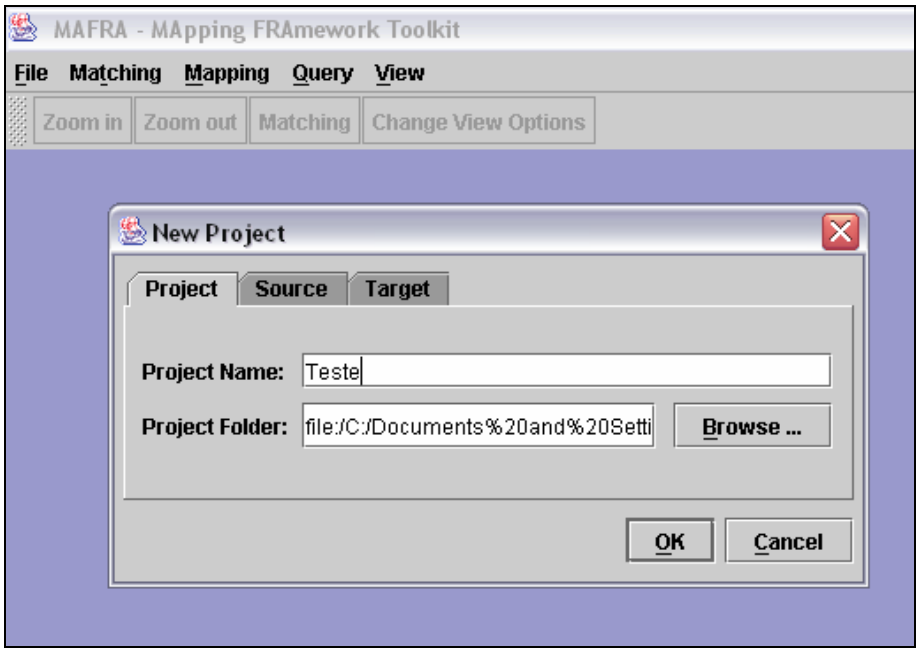

Figura 16 - Diálogo para criação de um novo projecto no MAFRA Toolkit

A Figura 16 representa a caixa de diálogo para criação de novos projecto no MAFRA Toolkit. O problema que se verifica neste caso, prende-se com a existência de espaços, na *path* do ficheiro que se pretende criar. Ou seja, caso exista algum espaço, o projecto não é criado correctamente e é lançada uma excepção de erro.

Analisando uma porção de código retirada do ficheiro "**HMAFRAProject.java**",

```
1.  . . .
2.  public void saveProjectFile() throws Exception
3.  {
4.  TransformationUtil.saveXML ( (XMLDocument)m_xmlProjectFile,
    m_strProjectFileURI );
5.  }
6.  . . .
```

é possível verificar que na linha 4 é chamado um método que recebe como segundo parâmetro, a *string* membro "m_strProjectFileURI".

Esse parâmetro refere-se à *path* onde irá ser criado o projecto. Esta *string* é proveniente da caixa de diálogo onde o utilizador insere o *path* de criação.

Matéria relacionada: 4.1.1 - Criação de Projectos

3.2 Gravação de projectos

Outro aspecto que não foi detectado inicialmente mas que também representa um problema de funcionamento, surgindo um pouco como consequência da solução encontrada para a criação de projectos, foi aquando da gravação de projectos criados com espaços na *path* no MAFRA Toolkit.

Ou seja, uma vez que já era possível criar novos projectos com espaços nas *paths* que definem o local físico da sua criação, surge agora a questão de os gravar ou regravar no mesmo ou em outro local físico cuja *path* contenha espaços.

Seleccionando a opção de gravação do projecto para criação do ficheiro de *mapping* (ficheiro de projecto*)*, o método "saveMapping", que pertence ao objecto do

tipo da classe “Mapping”, é encarregue de gravar esse ficheiro no espaço físico que é dado, uma vez mais, por uma *string* de pa*th*.

Considere o código seguinte:

```
7.      . . .
8.      public void saveMapping() throws Exception
9.      {
10.     KAONUtils.saveOIModel(this.m_oimodelSBOInstance,
11.     m_strMappingPhysicalURI );
12.     }
13.     . . .
```

Verifica-se que este método (“saveMapping”) limita-se a chamar um outro método passando-lhe por um dos parâmetros a *string* que contém a *path* do projecto em questão. Focando o problema em questão, o conteúdo desta *string* provém da criação de um novo projecto onde foi dado o caminho físico para a criação do mesmo com espaços.

Matéria Relacionada: 4.1.2 - Gravação de Projectos

3.3 Properties

Relembrando um pouco as relações de uma *property* relativamente às outras entidades gráficas neste contexto, como podemos observar na Figura 17 verifica-se que uma *property* pode estar graficamente ligada a uma outra *property* (via *Match*) ou a um ou mais *concepts* podendo estes representarem *Range* ou *Domain Concepts*.

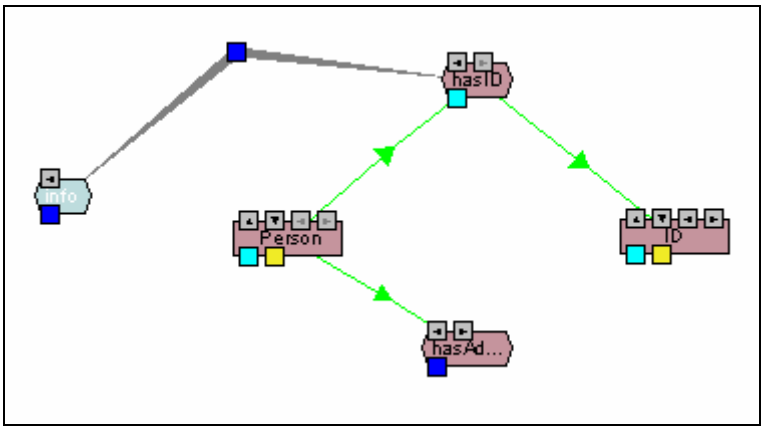

Figura 17 - Ligações Possíveis de uma Property

Na Figura 17 pode-se observar a *property* "hasID" ligada ao seu *Domain Concept* "Person" e ao seu *Range Concept* "ID". Verifica-se também que a mesma *property* se encontra ligada, através de um *Match*, à *property* "info".

O comportamento de uma *property*, que é observado quando lhe são escondidas todas as ligações às outras entidades, pode ser demonstrado pela Figura 18.

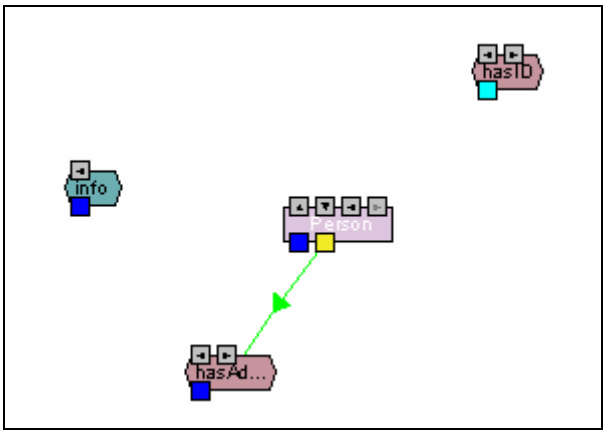

Figura 18 - Exemplo de Properties Órfãs

Na Figura 18 verifica-se que existem duas *properties* visíveis que não têm qualquer ligação com outras entidades, ou seja, estão órfãs.

Matéria relacionada: 4.2.10.2 - "Allow Orphan Properties"

3.4 Concepts

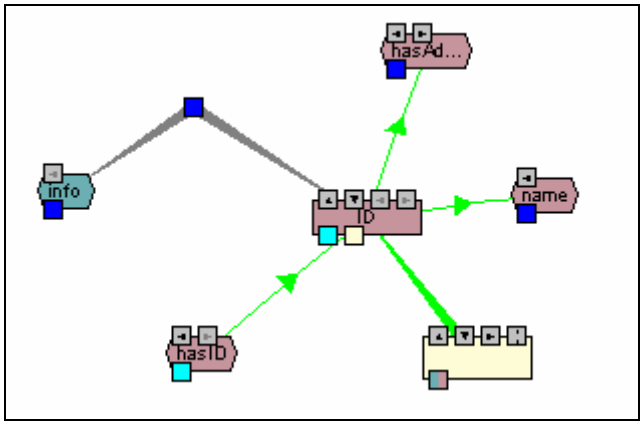

Figura 19 - Ligações Possíveis de um Concept

Na Figura 19 observa-se o *concept* "ID" ligado, através de um *Match*, à *property* "info", tem como "*properties to"* a *property* "hasID", como "*properties from"* as *properties* "hasAd..." e "name" e por fim está ligado a uma *concept bridge*.

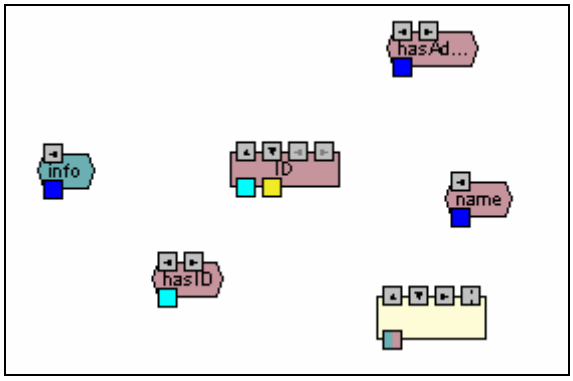

Figura 20 - Exemplo de um Concept Órfão

Após serem quebradas todas as ligações do *concept* "ID" às outras entidades (Figura 20), verifica-se um comportamento semelhante ao das *properties*, ou seja, o *concept* continua visível. É ainda relevante salientar que também através da Figura 20 se pode comprovar o comportamento já referido anteriormente relativamente às *properties*.

Matéria relacionada: 4.2.10.1 - "Allow Orphan Concepts"

3.5 Concept Bridges

No capítulo 2 caracterizou-se as c*oncept bridges* por serem pontes de interligação entre dois os mais *concept*s de diferentes ontologias.

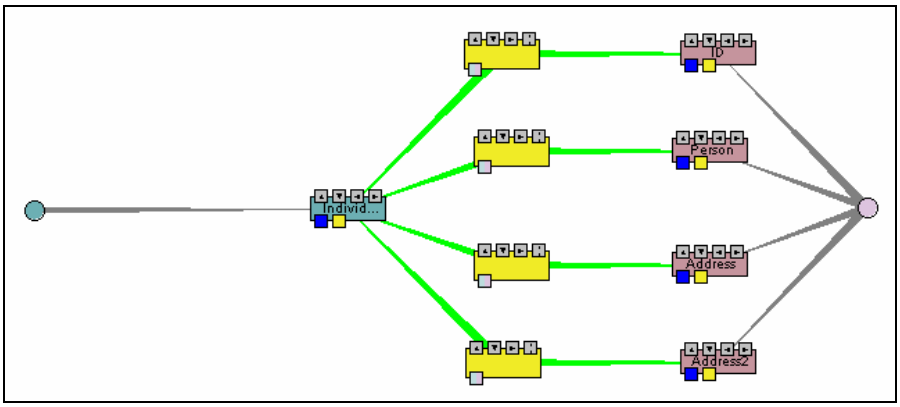

Figura 21 - Concept Bridges a interligar Concepts

Na Figura 21 pode-se observar o *concept* "Individ..." ligado a 4 *concept bridges* diferentes em que cada uma delas possui uma ligação para 1 *concept* diferente.

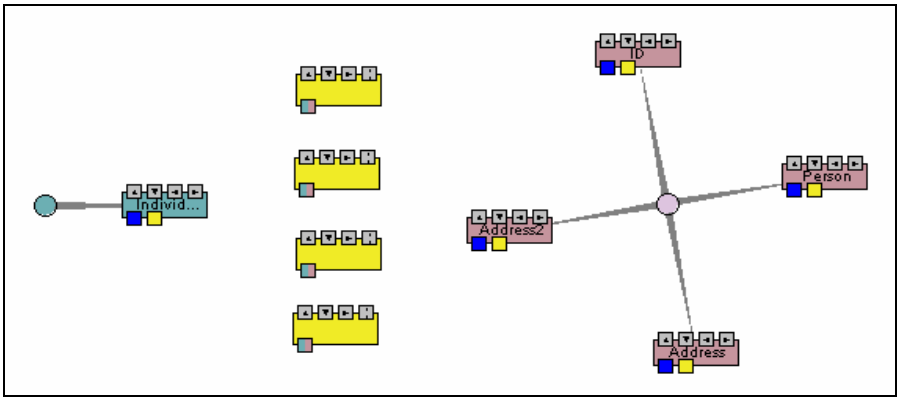

Figura 22 - Exemplo de Concept Bridges Órfãs

Através da Figura 22 repara-se que ao serem quebradas as ligações das *concept bridges* com os *concepts*, elas permanecem também visíveis, ou seja, órfãs.

Matéria Relacionada: 4.2.9.1 - "Allow Orphan Concept Bridges"

3.6 Property Bridges

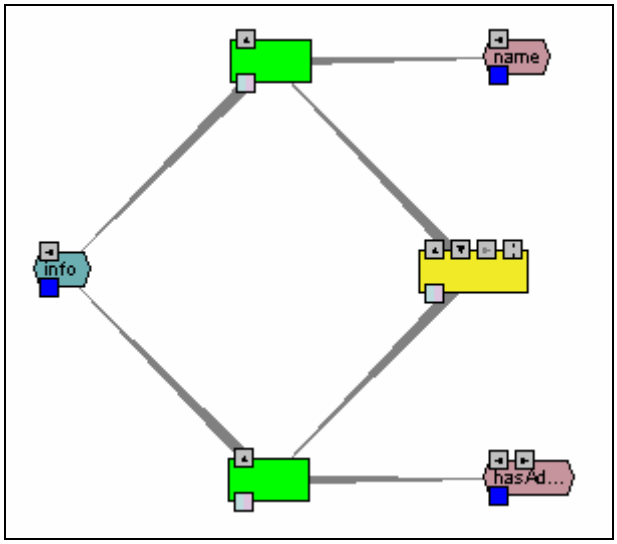

Figura 23 - Ligações possíveis de Property Bridges

Na Figura 23 estão representadas as ligações possíveis que as *property bridges* podem ter relativamente a outras entidades.

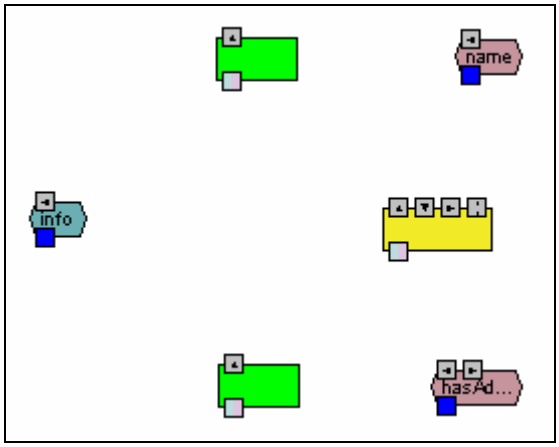

Figura 24 - Property Bridges Órfãs

Pela Figura 24, à imagem das outras entidades representadas e já analisadas, as *property bridges* apresentam um comportamento semelhante, ou seja, permanecem visíveis ou órfãs quando lhes são quebradas as ligações.

Matéria relacionada: 4.2.9.2 - "Allow Orphan Property Bridges"

3.7 Matches

Ao contrário das questões de incerteza que advieram da observação do comportamento das entidades atrás analisadas, os *matches* apresentam, claramente, um comportamento defeituoso.

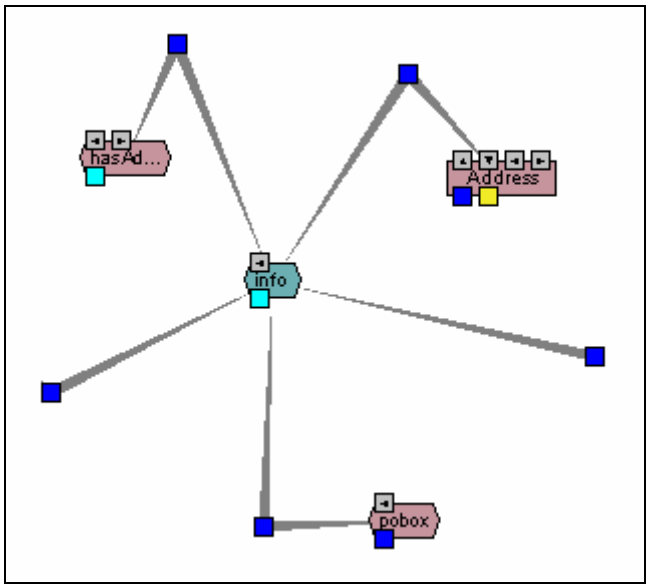

Figura 25 - Má abertura de Matches

A Figura 25 apresenta 3 *Matches* entre entidades, dois dos quais interligam *properties* e um outro que interliga uma *property* e um *concept*.

Depois vislumbram-se *Matches* incompletos, ou seja, verifica-se que pelo menos dois *Matches* apresentam falhas numa das suas extremidades.

Através desta análise gráfica, concluí-se que há um comportamento errado aquando da expansão dos *Matches* de uma entidade.

3.8 "Hide All Bridges"

Como foi descrito em 2.2.7, o utilizador tem a possibilidade de interagir com o ambiente gráfico através de uma série de opções disponibilizadas com o botão do lado direito do rato. Uma dessas opções, "Hide All Bridges", como o próprio nome indica, tem, supostamente, a capacidade para excluir do ambiente gráfico todas as *bridges*, sejam elas *concept bridges* ou *property bridges*.

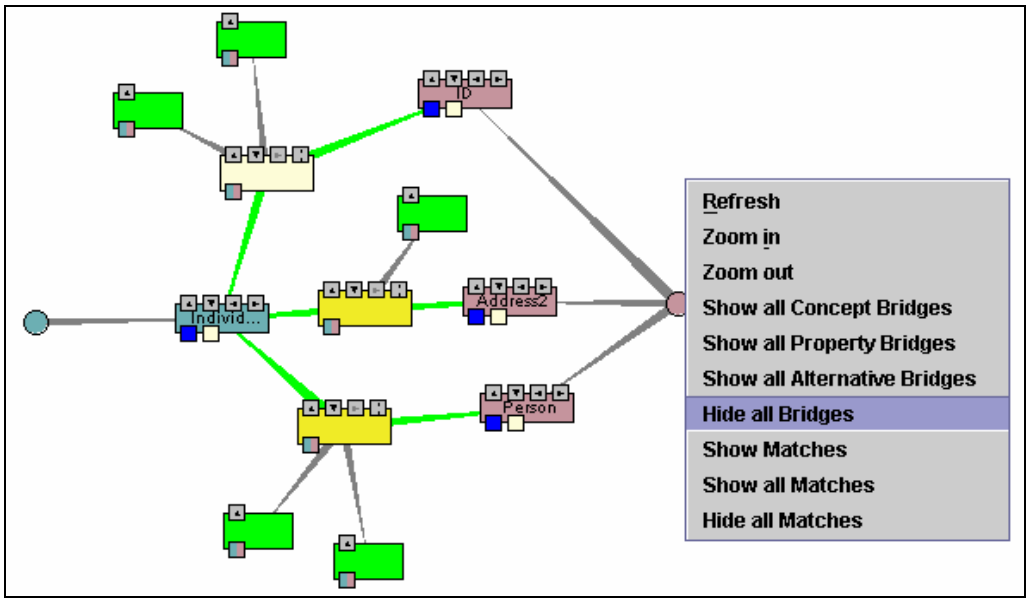

Figura 26 - Várias Semantic Bridges a Interligar Entidades

Na Figura 26 pode-se observar 3 *concept bridges* a interligar *concepts* e 5 *property bridges* derivadas das *concept bridges*. Percebe-se também que o utilizador se prepara para activar a opção "Hide All Bridges". O comportamento esperado será obviamente o desaparecimento das 8 *semantic bridges* e respectivas ligações(*Edges*), ficando apenas os *concepts* visíveis.

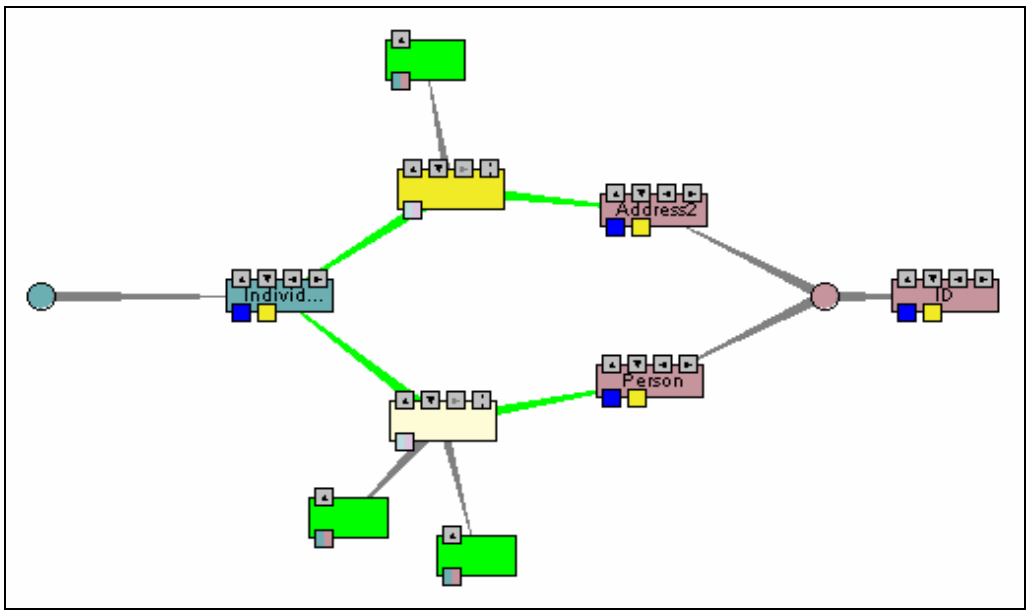

Figura 27 - Semantic Bridges após chamada da função "Hide All Bridges"

Analisando a Figura 27 infere-se que o resultado gráfico, após a chamada da função “Hide All Bridges”, não foi o esperado. Note-se que apenas uma das *concept bridges* e respectivas *property bridges* foram escondidas, permanecendo as restantes *semantic bridges* visíveis.

É facilmente perceptível que se está perante um comportamento parcialmente incorrecto, uma vez que apenas parte das *semantic bridges* é escondida.

Matéria relacionada: 4.6 - “Hide All Bridges”

3.9 “Hide All Matches”

Outra opção que é motivo de análise é a opção “Hide All Matches” que tem como função esconder todos os *Matches* que estão visíveis.

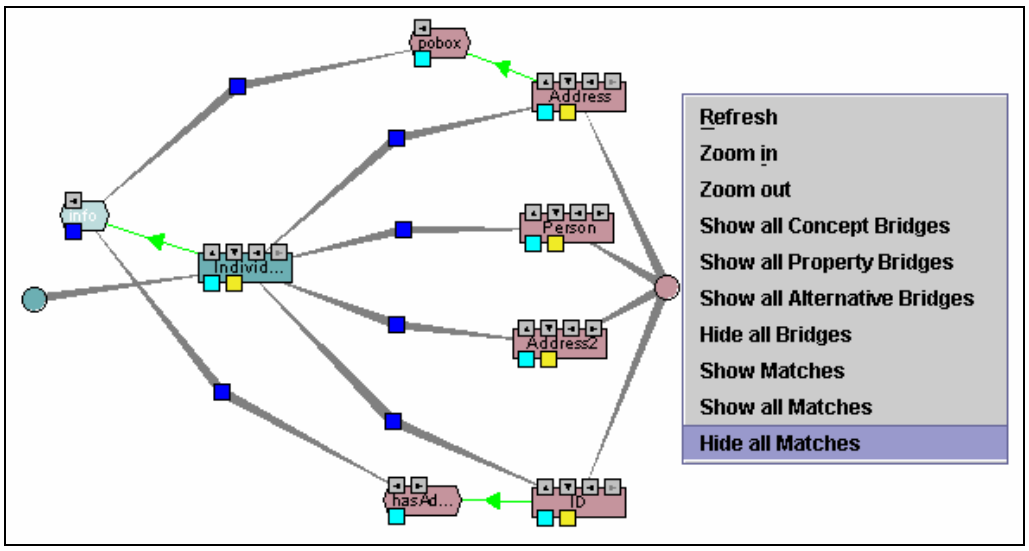

Figura 28 - Vários Matches a interligar Entidades

A Figura 28 apresenta 6 *Matches* dos quais 4 interligam um *concept* com *concepts* e 2 interligam *property* com *property*. Nesta altura o utilizador prepara-se para activar a opção “Hide All Matches” para esconder todos os 6 *Matches* visíveis.

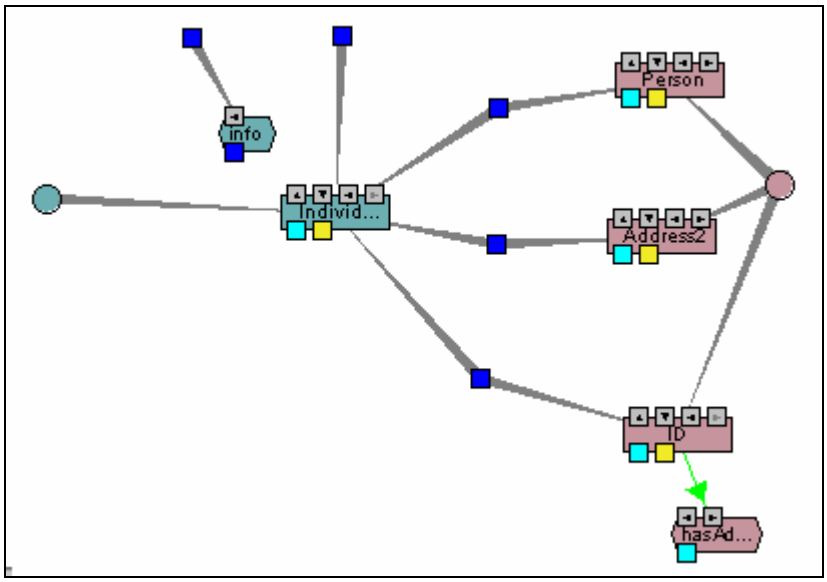

Figura 29 - Matches após chamada da função "Hide All Matches"

O comportamento da função não é, de todo, o esperado; alguns *Matches* desapareceram, outros perderam um *edge* e um dos *concepts* também foi perdido. Importa referir que este comportamento não é linear, ou seja, após chamadas repetidas desta opção com estados gráficos idênticos, o resultado varia. Isto é, no exemplo concreto da Figura 28, se for escolhida a opção “Hide All Matches” uma segunda vez, o resultado, provavelmente, será diferente do resultado apresentado na Figura 29.

Desta análise conclui-se que esta opção tem um comportamento completamente incorrecto e que, não obstante desse facto, apresenta também resultados inesperados e até, poder-se-á dizer, um pouco estranhos.

Matéria relacionada: 4.4.1 - “Hide All Matches”

3.10 Considerações

Ainda no âmbito da análise de comportamentos, importa referir alguns aspectos e conclusões que são relevantes para uma melhor compreensão desta matéria, sendo que, é possível sintetizá-los da seguinte forma:

- Relativamente a *Concepts*, *Properties* e *Semantic Bridges*(*Concept Bridges* e *Property Bridges*), levanta-se a questão: Será legítimo que, após a quebra de todas as ligações de uma destas entidades com outras entidades, ela

permaneça visível? A resposta para esta questão será abordada no próximo capítulo.

- Um *edge* nunca existe isolado. Ou seja, qualquer ligação que for quebrada entre duas entidades, o comportamento esperado será sempre a perda de visibilidade do edge que as interliga.

- Um *Match* nunca existe isolado ou apenas com uma das extremidades. Isto é, não faz sentido, em termos gráficos, que um *Match* apareça isolado na medida em que o mesmo não representa absolutamente nada. Também não faz sentido que um *Match* apareça apenas com uma entidade numa das extremidades. Mais concretamente, estando perante qualquer um deste dois casos, não se torna possível retirar qualquer informação ou informação completa sobre o que representa o *Match*.

- Ainda relativamente aos *Matches* surge uma outra questão: O que fazer quando se expande os *matches* de uma entidade? Devem-se tornar visíveis todos os *matches*, e respectivas entidades, que lhe dizem respeito mesmo as que estejam escondidas, ou não?

- Para retrair ou mostrar uma ligação de uma para uma entidade ou de uma para muitas entidades, pode-se utilizar tanto o botão afecto a essa ligação contido numa extremidade da mesma, assim como o botão contido na outra extremidade. Isto é, há sempre duas maneiras de quebrar uma mesma ligação. Por exemplo, na Figura 30, observa-se a *property* "hasID" ligada ao seu *Domain Concept* através de um *Edge*. Caso se pretendesse proceder à remoção dessa ligação haveria duas hipóteses:

    - carregar no botão "Domain Concepts" da *property* "hasID"

    - carregar no botão "Properties From" do *concept* "Person"

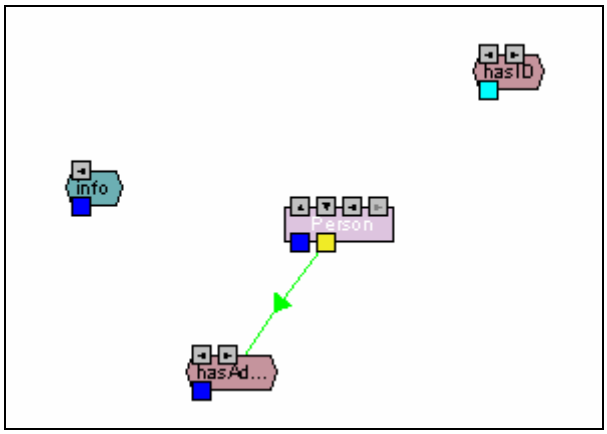

Figura 30 - Exemplo ilustrativo de quebra de ligações

4. Desenvolvimento

4.1 Paths

4.1.1 Criação de Projectos

Após várias tentativas, chega-se à conclusão que o problema da *path* na criação de projectos, está precisamente na *string* que a especifica.

Reapreciando o código referido no capítulo anterior em 3.1, verificamos que

```
14.     . . .
15.     public void saveProjectFile() throws Exception
16.     {
17.     m_strProjectFileURI = m_strProjectFileURI.replaceAll("%20"," ");
18.     TransformationUtil.saveXML (m_xmlProjectFile, m_strProjectFileURI );
19.      }
20.     . . .
```

a linha 17 foi agora adicionada antes da chamada do método e a mesma constitui a solução para o problema.

A função da linha 17 é transformar a *string* "m_strProjectFileURI", que contêm a *path* para o novo projecto, de forma a que todas as ocorrências da *substring* "%20" sejam substituídas por espaços.

De alguma forma, internamente, existiria um conflito com esta *substring* uma vez que a mesma está sempre presente nas *paths* quando há espaços.

4.1.2 Gravação de Projectos

Sendo este problema praticamente uma consequência da solução de 4.1.1, uma vez que sendo agora possível criar projectos com espaços nas suas *paths*, tem de ser

também tornado possível gravá-los com essas mesmas *paths*, a solução para este problema, foi encontrada baseando-se na anterior em 4.1.1.

Utilizando novamente o código de 3.2,

```
21.     . . .
22.     public void saveMapping() throws Exception
23.     {
24.     m_strMappingPhysicalURI = m_strMappingPhysicalURI.replaceAll("%20"," ");
25.     KAONUtils.saveOIModel(this.m_oimodelSBOInstance,
26.     m_strMappingPhysicalURI );
27.     }
28.     . . .
```

verifica-se que a linha 24 foi adicionada.

À imagem da solução encontrada em 4.1.1, torna-se necessário, novamente, uma manipulação de *strings*. O caso é idêntico, onde é feita uma substituição de todas as ocorrências da *substring* "%20" por espaços, na *string* membro "m_strMappingPhysicalURI".

Isto torna possível a gravação do projecto sem quaisquer problemas.

4.2 Diálogo

A resposta a algumas das questões levantadas no capítulo anterior em 3.10 através da análise de comportamentos das entidades, apresenta-se agora de uma forma clara. A solução encontrada para solucionar estas dúvidas, passa pela criação de uma caixa de diálogo global, que disponibilize ao utilizador uma série de opções que lhe permitam definir o comportamento desejado neste contexto.

Para a criação dessa caixa de diálogo adoptou-se um modelo que contém um painel de propriedades ou *properties panel* onde são criados os separadores ou *tabs* que se relacionam com as opções que cada um deles irá conter.

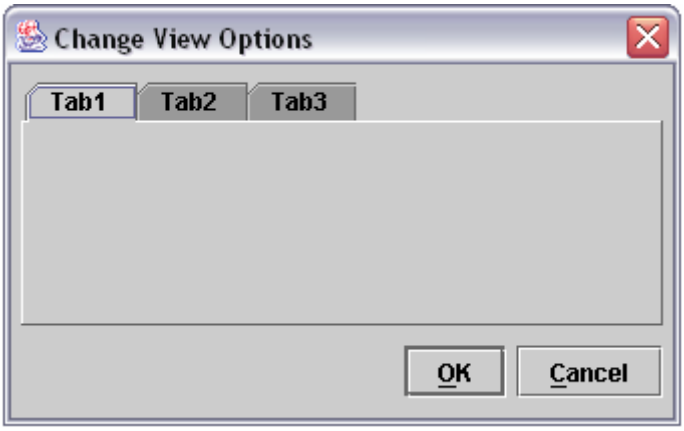

Figura 31 - Caixa de Diálogo Básica

A nova caixa de diálogo foi apelidada com o título “Change View Options”, que em português significa “Alterar Opções de Visualização”. A Figura 31, representa o estado básico da mesma, apresentando 3 *tabs* que fazem parte de um *properties panel* e os dois botões tradicionais “OK” e “Cancel”.

Para criar e chamar a nova caixa de diálogo, é necessário seguir uma série de passos.

4.2.1 Ficheiro “gui_configuration.xml”

Este ficheiro contém uma estrutura XML onde estão especificados, entre outros, aspectos como construção de menus, itens, separadores, botões, etc. A cada um desses itens e botões estão acções associadas. Neste caso concreto, pretende-se adicionar um botão e um item de menu para chamar a nova caixa de diálogo como se pode observar na Figura 32 e na Figura 33.

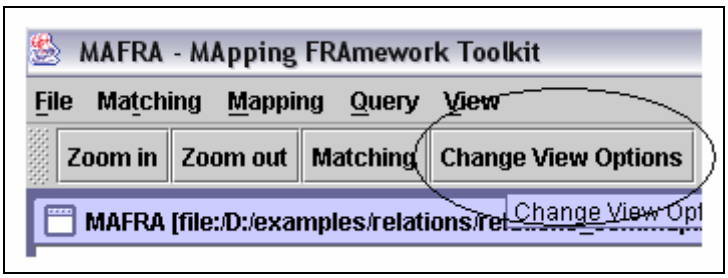

Figura 32 - Botão da Barra de Menu “Change View Options”

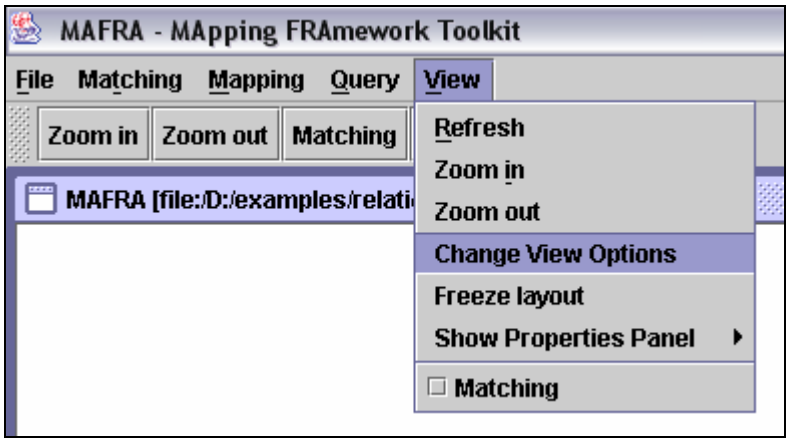

Figura 33 - Item de Menu "Change View Options"

A nível de código, para se obter o resultado observado na Figura 33,

```
29.     . . .
30.     <menu actionID="menu.view">
31.     <menuitem actionID="action.mafra.refresh"/>
32.     <menuitem actionID="action.mafra.zoomIn"/>
33.     <menuitem actionID="action.mafra.zoomOut"/>
34.     <menuitem actionID="action.mafra.changeViewOptions"/>
35.     <menuitem actionID="action.mafra.freezeLayout"/>
36.     <modalmenu actionID="action.mafra.changesPropertiesPanelViewPolicy"/>
37.     <separator/>
38.     . . .
```

note-se que a linha 30 indica a criação do submenu "View" e é precisamente dentro desse submenu que estão a ser colocados os items das opções de visualização. Entre essas opções destaca-se a nova opção adicionada pela linha 34 onde é criado um novo item de menu com a acção correspondente à nova caixa de diálogo. Isto é, o novo item de menu tem a tarefa de invocar a acção "action.mafra.changeViewOptions" que, por sua vez, invoca e mostra a nova caixa de diálogo.

Um pouco mais abaixo, analisando mais uma porção de código do mesmo ficheiro,

```
39.     . . .
40.     <toolbar key="main_toolbar">
41.     <button actionID="action.mafra.zoomIn"/>
42.     <button actionID="action.mafra.zoomOut"/>
43.     <button actionID="action.mafra.changesMAFRAProcessPhase" type="toggle"/>
44.     <button actionID="action.mafra.changeViewOptions"/>
45.     </toolbar>
46.     . . .
```

observa-se a criação do novo botão (Figura 32), na barra principal de botões, que irá invocar a nova caixa de diálogo. A linha 44 foi adicionada para, juntamente com as linhas anteriores formar o conjunto de botões que são apresentados na barra principal.

É ainda de salientar que lhe foi associada uma acção idêntica à do novo item do submenu “View”, uma vez que ambos têm a mesma tarefa de chamar a nova caixa de diálogo.

4.2.2 Ficheiro “mafra.xml”

Aquando da criação do novo item de menu e do novo botão atrás mencionados, a cada um deles foi associada uma *string* que corresponde a uma dada acção, sendo neste caso, a *string* “action.mafra.changeViewOptions”. Mas na realidade, a nível gráfico, o texto que aparece no novo botão e no novo item de menu, é diferente.

Neste caso em concreto, o texto que aparece em ambos os casos é : “Change View Options” e não “action.mafra.changeViewOptions”.

Analisando o seguinte código proveniente do ficheiro “mafra.xml”, percebe-se porquê.

```
47.     . . .
48.     <phrase key="action.mafra.changeViewOptions" value="Change View Options"/>
49.     <phrase key="action.mafra.changeViewOptions.short" value="Change View Options"/>
50.     <phrase key="action.mafra.changeViewOptions.long" value="Allows The User To
51.     Change Some View Options"/>
52.     . . .
```

Neste contexto, este ficheiro é útil para criar uma relação entre uma acção criada e um texto correspondente.

Se for analisada a linha 48, repara-se que à acção referida anteriormente está associado um texto com o valor “Change View Options”. É este texto que irá aparecer em detrimento de “action.mafra.changeViewOptions”.

A adição das duas linhas abaixo, 49 e 50, permitiu também definir o texto que irá aparecer quando se coloca o ponteiro do rato sobre o novo botão ou sobre o novo item de menu e também o texto explicativo que irá aparecer na barra de estado do interface da aplicação.

4.2.3 Classe “MAFRAModule”

Outra das questões que surge após a criação gráfica dos controlos que invocam a nova caixa de diálogo e a associação dos mesmos a uma determinada acção é, precisamente, onde está definida essa acção e o que faz.

```
53.     . . .
54.     addAction(new NewProject(this));
55.     addAction(new OpenProject(this));
56.     . . .
57.     addAction(new ChangeViewOptions(this));
58.     . . .
```

Na classe “MAFRAModule” estão definidas, entre outras, acções relacionadas por exemplo com botões e itens de menu. Ou seja, o que acontece quando um desses elementos é seleccionado.

Na linha 57, uma nova acção é adicionada. Ela refere-se à criação de um objecto do tipo “ChangeViewOptions”.

A partir desta altura, obtém-se então uma nova acção criada e relacionada com a classe “ChangeViewOptions”.

4.2.4 Classe “ChangeViewOptions”

Após a acção ser lançada, a classe “ChangeViewOptions” está encarregue de criar o objecto da nova caixa de diálogo e torná-la visível, colocando todos os seus controlos precisamente no mesmo estado em que estavam desde a última vez que esta foi aberta.

```
59.     . . .
60.     ChangeViewOptionsDlg vOpDlg = new ChangeViewOptionsDlg(
61.     m_mafraModule, m_harmonisationType );
62.     vOpDlg.setTitle(m_localizationManager.getPhrase("Change View Options"));
63.     vOpDlg.setVisible(true);
64.     . . .
65.     mafraViewable.changeAlwaysShowMatchesState(
66.     vOpDlg.getCbAllMatchesStateControl() );
67.     . . .
```

Nas linha 60 pode-se verificar a criação de um objecto do tipo “ChangeViewOptionsDlg” que corresponde à caixa de diálogo propriamente dita. Em seguida, após lhe ser dado o título, observa-se que é na linha 63 que ela é tornada visível graficamente. Nas linhas 65 e 66 está um exemplo de como um dos controlos da caixa de diálogo neste caso, “CbAllMatchesStateControl”, é actualizado conforme o seu estado (activado ou não).

4.2.5 Classe “ChangeViewOptionsDlg”

A classe “ChangeViewOptionsDlg” representa, como já foi referido, a caixa de diálogo, onde é feita a implementação do seu aspecto gráfico e a criação e disposição dos seus controlos.

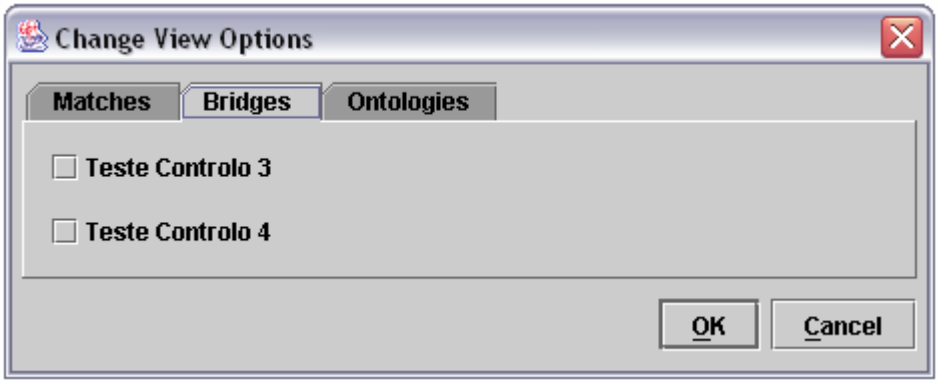

Figura 34 - Caixa de Diálogo Com Controlos de Teste

Como não é de grande relevância apresentar o código de construção do *layout* apresentado na Figura 34, através dela infere-se que incluído como controlo dentro da caixa de diálogo, está um painel de propriedades ou *properties panel*, no qual existem 3 separadores, “Matches”, “Bridges” e “Ontologies”. No exemplo da Figura 34 observa-se ainda que dentro do separador “Bridges” estão visíveis 2 controlos do tipo *CheckBox*.

4.2.6 Classe “ChangeViewOptionsPanel”

Uma característica importante da caixa de diálogo é o *Properties Panel*. É importante na medida em que representa um novo objecto de um nova classe que é incluído dentro do *layout* da caixa de diálogo.

O *Properties Panel* integra os separadores ou *tabs*, sendo que cada um deles é responsável por disponibilizar as opções que contextualmente lhe dizem respeito.

Analisando novamente a Figura 34, o *properties panel* representa parte significativa da área total da caixa de diálogo.

As opções que estão disponibilizadas por cada um dos *tabs* do *properties panel* são as seguintes:

Tab “Matches”:

- “When expanding a Match, Show All Related Matches even if not visible”

Tab “Bridges”:

- “Allow Orphan Concept Bridges”
- “Allow Orphan Property Bridges”

Tab “Ontologies”:

- “Allow Orphan Concepts”
- “Allow Orphan Properties”

Cada uma destas opções, corresponde a controlos do tipo *CheckBox* de modo a serem seleccionadas ou não.

A nível de código, é importante referir que esta classe para além de implementar especificações a nível de controlos e *layout* do *properties panel*, implementa também 3 *inner-classes* protegidas em que cada uma delas, representa um separador ou *tab*. Isto é, por exemplo, o separador “Matches” é uma *inner-class* da classe *ChangeViewOptionsPanel* .

4.2.7 Classe “MAFRAViewable”

A classe “MAFRAViewable” tem, neste contexto, a tarefa de definição de variáveis globais que irão afectar e ser afectadas directamente pelas opções seleccionadas ou não da caixa de diálogo.

Embora não sejam variáveis globais na sua verdadeira essência, a classe define métodos públicos que as tornam disponíveis às outras classes para consulta ou alteração.

```
68.     . . .
69.     boolean bAlwaysShowMatchesState = false;
70.     boolean bOrphanPropertiesState = false;
```

```
71.     boolean bOrphanConceptBridgesState = false;
72.     boolean bOrphanPropertyBridgesState = false;
73.     boolean bOrphanConceptsState = false;
74.     . . .
75.     public boolean getOrphanConceptsState()
76.     {
77.         return bOrphanConceptsState;
78.     }
79.     public void changeOrphanConceptsState( boolean orphanConceptsNewState )
80.     {
81.         bOrphanConceptsState = orphanConceptsNewState;
82.     }
83.     . . .
```

O código acima representado, é extraído precisamente da classe "MAFRAViewable" no local onde algumas variáveis, importantes para a matéria em questão, são definidas e também algum código referente a alguns métodos que se relacionam com elas.

Se for analisada uma dessas linhas, a título de exemplo a linha 73, está definida uma variável do tipo *boolean* com o nome "bOrphanConceptsState". Como o nome assim o indica, esta variável irá corresponder ao estado da *CheckBox* da opção "Allow Orphan Concepts" do *tab* "Ontologies" da caixa de diálogo.

Mais abaixo, pode-se observar que existem dois métodos (linhas 75 e 79), que estão directamente relacionados com a variável definida na linha 73 e que são usados por classes externas para consultar e alterar o seu valor, respectivamente.

Em suma, o estado destas variáveis(true/false) é alterado indirectamente pelo utilizador, através da caixa de diálogo, e são elas que vão determinar o comportamento das entidades.

4.2.8 Implementação das opções do *tab* Matches

4.2.8.1 "When expanding a Match, Show All Related Matches even if not visible"

Antes de se proceder à explicação da implementação desta solução, convém referir que, nesta altura, está-se a assumir que a solução para a má abertura de *matches* já foi implementada. Ou seja, esta matéria trata já os matches a um nível mais elevado de implementação.

Após alguma ponderação sobre o comportamento dos *matches*, chega-se à conclusão que o utilizador poderia pretender visualizar os *matches* de duas maneiras distintas:

- Quando se expandem os *matches* de uma entidade, apenas se mostram os *matches* cujas entidades da outra extremidade estejam visíveis na altura
- Quando se expandem os *matches* de uma entidade, devem ser mostrados todos os *matches* relacionados com ela, tornando visíveis todas as outras entidades que não estejam visíveis na altura mas que correspondam às extremidades desses *matches.*

Neste sentido, convencionou-se que seria correcto criar uma opção que permitisse escolher entre estes dois tipos de comportamento.

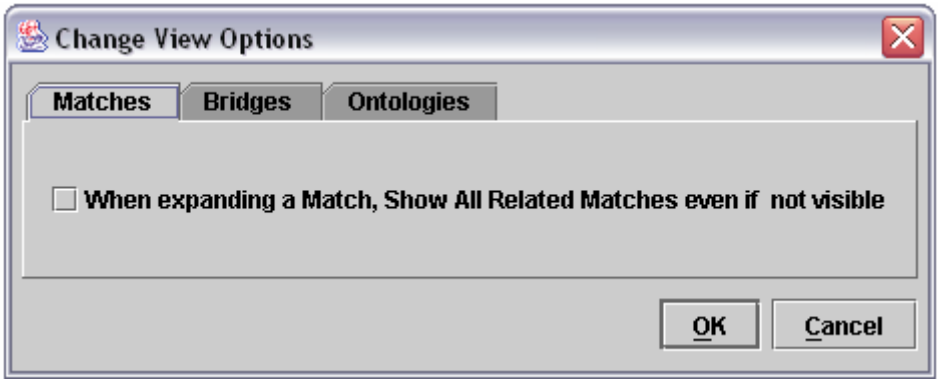

Figura 35 - Dialog Box - Tab Matches

Na Figura 35 observação uma opção do tipo *CheckBox* colocada dentro do *tab* "Matches".

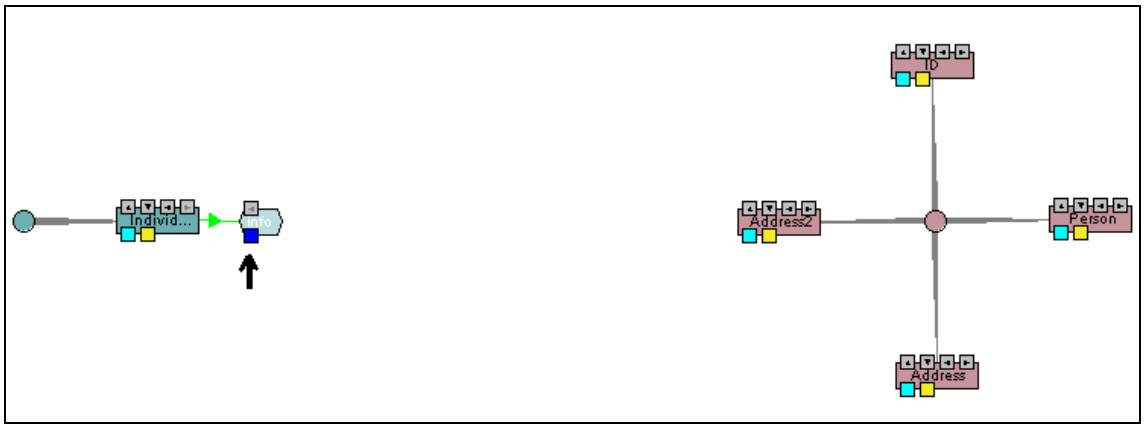

Figura 36 - Momento antes de expandir os Matches de "info"

Na Figura 36 o utilizador prepara-se para expandir, através do botão azul, todos os *matches* da *property* "info".

A Figura 37 apresenta o resultado desse expansão dos *matches* de "info".

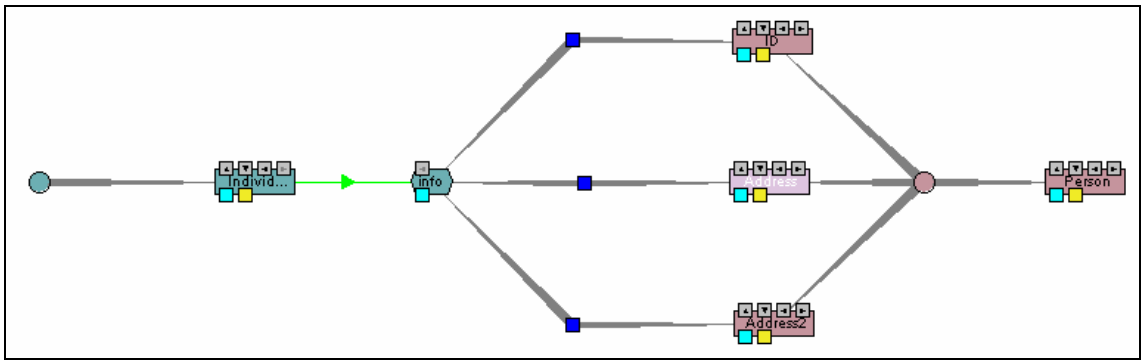

Figura 37 - Resultado da expansão de matches (simplificada)

Observe-se que a *property* "info" tem ligação por *match* a 3 dos 4 *concepts* que estão visíveis. Este comportamento está correcto e é aceitável mediante um tipo de configuração.

O comportamento deu-se desta forma porque foi afectado directamente pelo facto de a opção "When expanding a Match, Show All Related Matches even if not visible" não estar seleccionada. Isto é, apenas foram criados e mostrados os *matches* relacionados com as entidades que estão visíveis.

De qualquer modo, não se pode afirmar peremptoriamente que a *property* "info" apenas tem estes *matches*. E eis a questão que se coloca: "Será que existem mais *matches* para a *property* "info"?

Seleccionada agora a *CheckBox* da opção "When expanding a Match, Show All Related Matches even if not visible" como se pode observar na Figura 38,

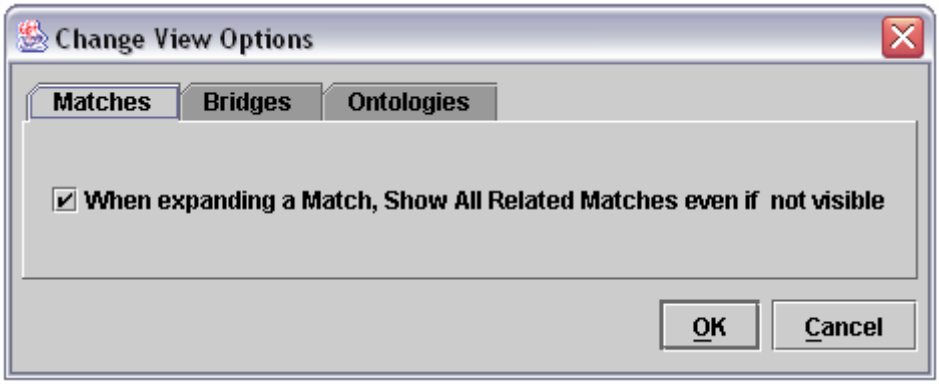

Figura 38 - Dialog Box - Tab de Matches - opção seleccionada

observe-se agora o comportamento gerado mediante o mesmo ponto de partida demonstrado na Figura 36.

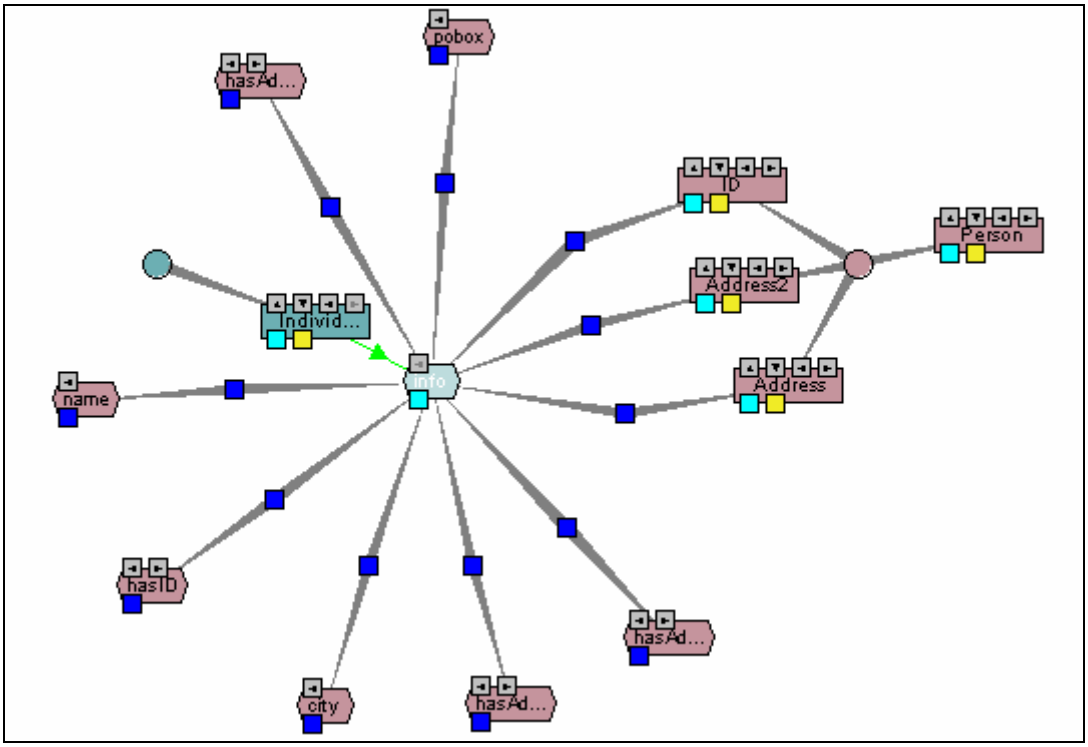

Figura 39 - Resultado da expansão de matches completa

A resposta à pergunta colocada anteriormente é afirmativa, ou seja, sim, existem mais *matches* para a *property* “info”.

Isto ficou comprovado pelo comportamento observado na Figura 39, em que é visível que a *property* “info” para além de ter *matches* com os 3 *concepts* já verificados, apresenta também mais 7 *matches* de interligação com outras entidades.

É importante salientar que embora neste exemplo, os *matches* que se tornaram visíveis posteriormente fossem todos *properties*, há uma independência completa deste comportamento em relação às entidades. Isto é, podíamos inicialmente inverter os papeis dos *concepts* e das *properties* que o comportamento iria ser novamente coerente com o apresentado.

Em resumo, esta dupla solução adequa-se na perfeição às necessidades de um utilizador avançado da aplicação e que pretenda visualizações e comportamentos específicos relacionados com os *matches*.

A nível de implementação desta solução, uma vez que as entidades que podem ser interligadas por *matches* são os *concepts* e as *properties*, o dois métodos que expandem os *matches* de cada um deles são relativamente idênticos e ambos utilizam o mesmo método para expansão de entidades de um *match*. Ou seja, a maneira como a expansão é feita, tanto num *concept* como numa *property*, é idêntica nos dois casos. Sendo assim torna-se claro que a tarefa de expansão de *matches* pertence à própria entidade (*concept* ou *property*) mas a tarefa de decisão sobre a visibilidade desses *matches* e das entidades que os interligam fica relegada para a própria entidade *match*.

4.2.8.2 Método “expandEntities” da classe “MatchNode”

O método “expandEntities” tem a tarefa de expandir as extremidades do próprio objecto (*match*) e tornar as entidades que lhe dizem respeito também visíveis caso ainda não estejam. Note-se que este método é chamado em ciclo pelo método “expandMatches” da entidade em questão até que não haja mais *matches* para essa mesma entidade.

No âmbito desta nova opção introduzida na caixa de diálogo, este método sofre quase uma total redefinição. A nível de pseudocódigo, essa redefinição segue uma

estrutura consideravelmente diferente da sua implementação anterior, apresentando uma lógica totalmente repensada para se poder adaptar a diferentes comportamentos.

A lógica deste método pode ser especificada, simplificadamente, da seguinte maneira:

```
84.     .... PARA A ENTIDADE ORIGEM VISIVEL ....
85.     Se a entidade origem estiver visível
86.         Se a entidade destino estiver visível
87.             então cria nó de match e edges de ligação
88.         Se a entidade destino não estiver visível
89.             Se a CheckBox estiver seleccionada ( permite tornar visível o destino )
90.                 então torna visível a entidade destino, cria o nó e edges de ligação
91.             Caso contrário ( não permite tornar visíveis entidades escondidas )
92.                 destroí o match (que é o próprio objecto)
93.     .... PARA A ENTIDADE DESTINO VISIVEL ....
94.     Se a entidade destino estiver visível
95.         Se a entidade origem estiver visível
96.             então cria nó de match e edges de ligação
97.         Se a entidade origem não estiver visível
98.             Se a CheckBox estiver seleccionada ( permite tornar visível a origem )
99.                 então torna visível a entidade origem, cria o nó e edges de ligação
100.            Caso contrário (não permite tornar visíveis entidades escondidas)
101.                destroí o match (que é o próprio objecto)
```

A lógica apresentada acima, repete-se para os dois casos; ou seja, se for expandido um *match* de uma entidade origem (já estava visível) desencadeia-se um processo idêntico ao processo que se desencadearia caso o ponto de expansão fosse a entidade destino (já estava visível).

4.2.9 Implementação das opções do *tab* Bridges

Relativamente às *semantic bridges*, através da observação e análise dos seus comportamentos, torna-se também evidente a disponibilização, através da caixa de diálogo (Figura 40), de opções que permitam definir comportamentos distintos para estas entidades.

Figura 40 - Dialog Box - Tab Bridges

4.2.9.1 "Allow Orphan Concept Bridges"

Esta opção tem a função de permitir, ou não, que *concept bridges* possam permanecer visíveis no ambiente gráfico sem qualquer ligação a outra ou outras entidades.

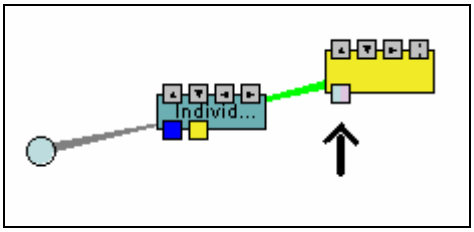

Figura 41 – Ligação (Concept - Concept Bridge) antes de ser quebrada

Na Figura 41 pode-se observar apenas uma ligação entre um *concept* e uma *concept bridge*. Nesta altura o utilizador prepara-se para quebrar essa ligação entre as duas entidades.

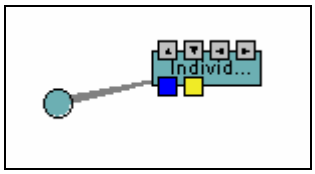

Figura 42 – Ligação (Concept-Concept Bridge) quebrada (sem "Allow Orphan")

Na Figura 42 apresenta-se o resultado da quebra da ligação e é visível que a *concept bridge* foi escondida. Este comportamento deve-se ao facto de a opção "Allow Orphan Concept Bridges" estar desactivada. É precisamente por esse facto que a *concept bridge* desaparece visto que não são permitidas *concept* bridges órfãs.

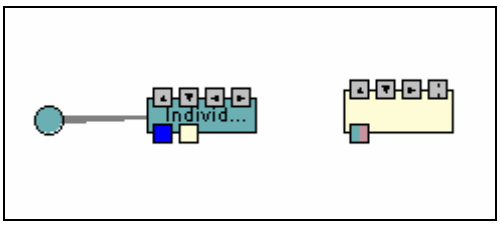

Figura 43- Ligação (Concept-Concept Bridge) quebrada (com "Allow Orphan")

Na Figura 43 demonstra-se o resultado da mesma acção levada a cabo pelo o utilizador, mediante o mesmo cenário inicial ilustrado na Figura 41.

Neste caso concreto, a opção "Allow Orphan Concept Bridges" está activada, logo, uma vez que são permitidas *concept bridges* órfãs, a *concept bridge* permanece visível, embora sem ligações, no ambiente gráfico.

Relembrando um pouco a fase de análise, o comportamento das *concept bridges* era, inicialmente, o comportamento demonstrado na Figura 43 e o utilizador não tinha qualquer forma de definir que não queria deixar *concept bridges* sem ligações visíveis.

4.2.9.2 "Allow Orphan Property Bridges"

À imagem das *concept bridges*, esta opção tem a função de permitir, ou não, que *property bridges* possam permanecer visíveis no ambiente gráfico sem qualquer ligação a outra ou outras entidades.

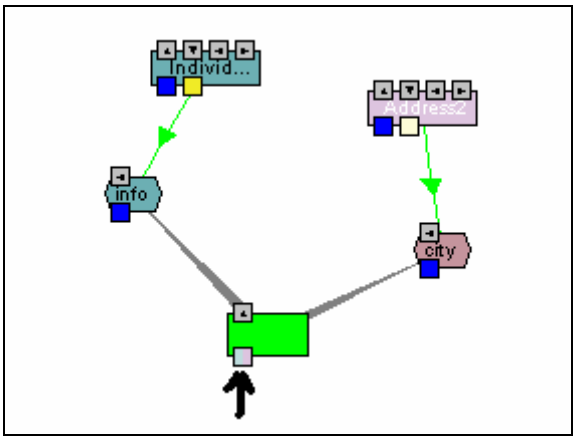

Figura 44 - Ligação (Properties - Property Bridge) antes de ser quebrada

Na Figura 44 observa-se uma *property bridge* ligada a duas *properties*. O utilizador prepara-se para quebrar as duas ligações através do botão "properties" da *property bridge*.

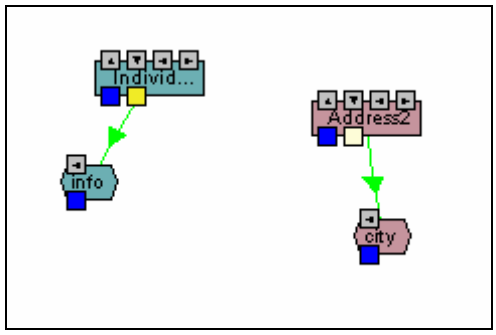

Figura 45 - Ligação (Properties-Property Bridge) quebrada (sem "Allow Orphan")

Pela Figura 45, a *property bridge* desapareceu quando foram removidas as ligações. Este comportamento é normal atendendo ao facto de que a opção "Allow Orphan Property Bridges" está desactivada.

Perante o mesmo cenário inicial, através da Figura 46 pode observar-se o comportamento da *property bridge* com a opção "Allow Orphan Property Bridges" activada.

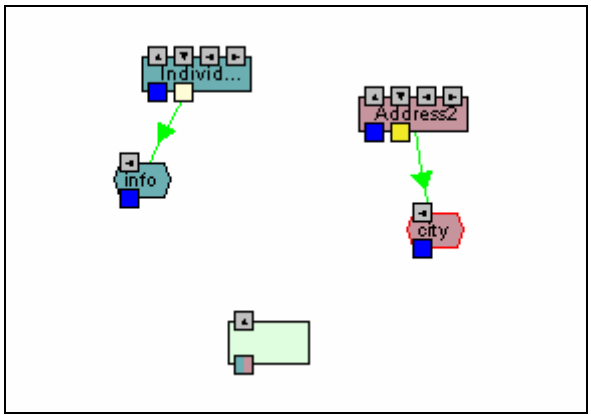

Figura 46 - (Properties - Property Bridge) quebrada (com "Allow Orphan")

Uma vez que já é permitido visualizar *property bridges* órfãs, a *property bridge* manteve-se visível, embora sem qualquer ligação.

4.2.10 Implementação das opções do *tab* Ontologies

A Figura 47 demonstra o *tab* "Ontologies" da caixa de diálogo.

Figura 47 - Dialog Box - Tab Ontologies

4.2.10.1 "Allow Orphan Concepts"

Esta opção permite manter ou não a visibilidade de *concepts* consoante o comportamento escolhido. Na Figura 48 observa-se um *concept* ligado a 4 *properties*.

A Figura 49 demonstra o resultado da quebra das ligações do *concept* com a opção "Allow Orphan Concepts" desactivada.

A Figura 50 demonstra o resultado da quebra das ligações do *concept* com a opção “Allow Orphan Concepts” activada.

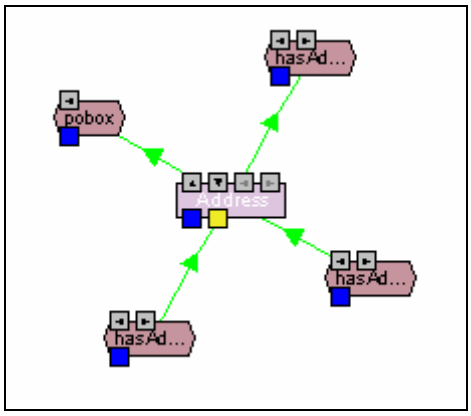

Figura 48 - Ligações (Concept - Properties) antes de serem quebradas

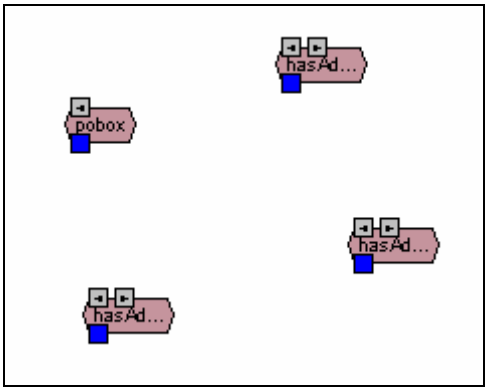

Figura 49 - Ligações (Concept - Properties) quebradas (sem “Allow Orphan”)

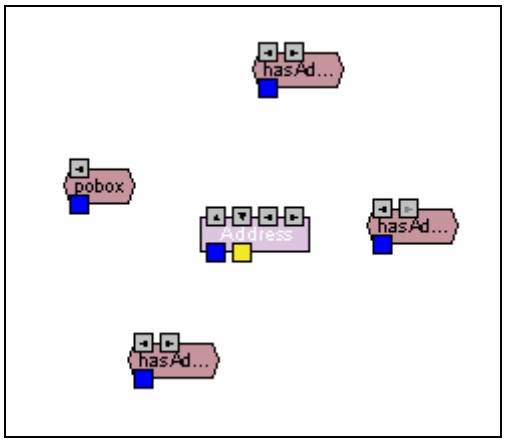

Figura 50 - Ligações (Concept - Properties) quebradas (com "Allow Orphan")

4.2.10.2 "Allow Orphan Properties"

Esta opção permite manter ou não a visibilidade de *properties* consoante o comportamento escolhido. Na Figura 51 observa-se 2 properties ligadas a 1 *concept*.

A Figura 52 demonstra o resultado da quebra das ligações do *concept* com a opção "Allow Orphan Properties" desactivada.

A Figura 53 demonstra o resultado da quebra das ligações do *concept* com a opção "Allow Orphan Properties" activada.

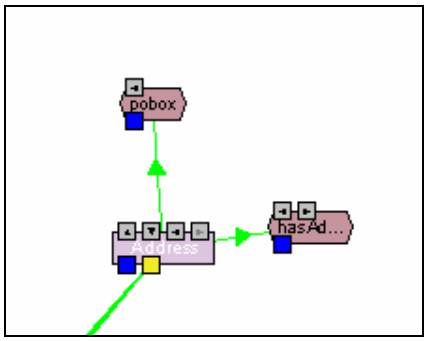

Figura 51 - Ligações (Properties - Concept) antes de serem quebradas

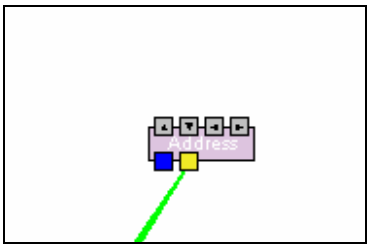

Figura 52 - (Properties - Concept) quebradas (sem "Allow Orphan")

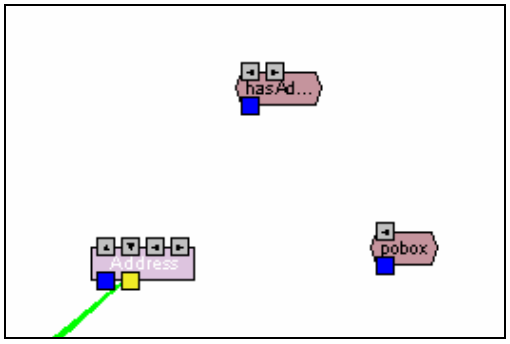

Figura 53 - (Properties - Concept) quebradas (com "Allow Orphan")

4.3 "Validate Visibility"

Uma das questões importantes que surgiram durante a implementação de soluções do género "Allow Orphan...", foi a questão das competências de visibilidade. A questão que surge conduz à pergunta:

Quem é que tem a função e a competência para mostrar ou esconder uma entidade aquando de uma dada acção, mediante as condições de visualização no momento adoptadas?

Numa implementação inicial, esta questão não surgiu, uma vez que o comportamento de todas as entidades era linear no sentido de permitir que as entidades sem ligações ficassem órfãs, isto é, visíveis mesmo sem ligações a outras entidades.

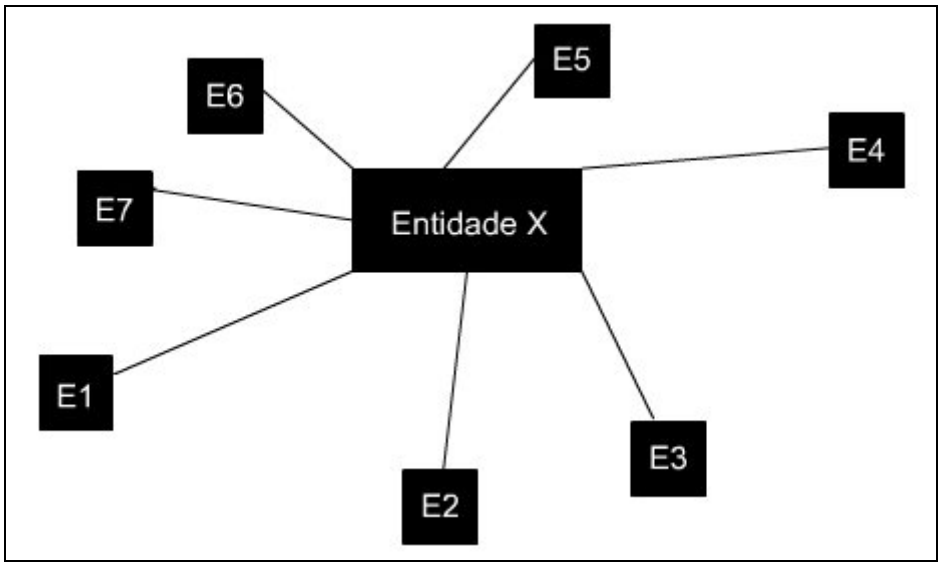

Figura 54 - Esquema de ligações entre entidades

Supondo que se está presente num cenário em que não são permitidas entidades órfãs e, pela Figura 54, supondo que a "Entidade X" quebra a sua ligação entre a entidade "E6", obviamente que a entidade "E6" irá desaparecer. Quem quebrou a ligação foi a "Entidade X", logo terá sido a "Entidade X" que se encarregou da tarefa de apagar "E6" do ambiente gráfico?

A questão apresenta-se agora como relevante sendo que, há que atribuir competências a "alguém" para tratar de esconder entidades, sem ligações, quando estas não são permitidas.

Analisando a questão a um nível mais adaptado à realidade, rapidamente se chega à conclusão que um indivíduo inserido numa comunidade, deve saber que, quando existe um certo impedimento para a sua presença, este tem que ter a capacidade inata de se retirar.

A mesma situação se pode aplicar a entidades gráficas deste contexto e, por conseguinte, cada entidade deve ter a capacidade de se auto-analisar mediante uma alteração do estado de uma ou de várias das suas ligações.

Por consequência desta nova abordagem, é necessário criar uma nova forma de cada entidade ter a capacidade de se auto-analisar.

O método “validateVisibility” terá essa mesma função e terá de estar presente em cada uma das entidades.

Todas as entidades gráficas derivam de uma classe chamada “MAFRANode” e é nessa classe que o novo método terá de ser definido da seguinte forma:

```
102.    . . .
103.    public abstract void validateVisibility(MAFRAGraph mafraGraph);
104.    . . .
```

Estando o método definido de forma abstracta, obriga a que todas as classes que derivam de “MAFRANode” implementem este método. Deste modo, o método poderá ser implementado com características individuais para cada classe de cada entidade.

A título de exemplo, o código abaixo representa a implementação do método “validateVisibility” na classe “PropertyNode”, classe esta que representa a entidade *property* a nível gráfico.

```
105.    public void validateVisibility(MAFRAGraph mafraGraph) throws KAONException
106.    {
107.        if(!this.isConnected(mafraGraph))
108.        {
109.            if(!mafraGraph.getMAFRAGraphAnchor().getOrphanPropertiesState())
110.            mafraGraph.remove(PropertyNode.this);
111.        }
112.    }
```

Note-se que na linha 107 é feita uma verificação das ligações desta entidade com outras através do método “isConnected(...)”. Caso haja um retorno “false”, significará que esta *property*(*this*) não tem quaisquer ligações o que servirá de indicação para que ela se auto-destrua caso não sejam permitidas *properties* órfãs, facto esse que é verificado na linha 109.

4.4 Matches

4.4.1 “Hide All Matches”

A matéria analisada em 3.9, acabou por ser resolvida facilmente pela implementação do método “validateVisibility”.

Nesta altura, o método “Hide All Matches” sofre uma total reestruturação em função da existência do método “validateVisibility”.

Ao analisar o comportamento gerado aquando da chamada do método “Hide All Matches”, verifica-se que ao esconder todos os *matches* poderá verificar-se o caso em que uma ou mais entidades que estavam interligadas pelos *matches* tenham ficado órfãs sem que tenha sido validada a sua existência gráfica.

Torna-se claro então que mais uma vez o método “validateVisibility” é extremamente útil para esta situação. É que, ao iterar sobre todos os *matches*, pode-se também iterar sobre todos os *concepts* ou *properties* chamando o método “validateVisibility” para cada um deles em cada iteração.

Desta forma garante-se que, ao esconder todos os *matches*, todos os *concepts* ou *properties* irão validar-se a si próprios de modo a verificar a sua legitimidade gráfica.

4.4.2 “Validate Visibility” para um Match

Embora a importância do método “validateVisibility” já tenha sido enaltecida com grande destaque na sua tarefa relativamente a grande parte das soluções apresentadas, este método, relativamente aos *matches*, têm uma implementação ligeiramente diferente e que importa realçar.

Em praticamente todos os casos, exceptuando os *matches*, o método está implementado de uma forma semelhante ao exemplo apresentado anteriormente em 4.3.

No caso de um *match*, quando uma das suas extremidades é escondida, não basta apenas remove-lo mas é também necessário informar a outra extremidade, de que o *match* vai ser removido e logo ela também terá de se validar a si própria.

O seguinte código representa parte do método “validateVisibility” de um *match*:

```
113.        . . .
114.        mafraGraph.remove(MatchNode.this);
115.        if(sourceNode != null)
116.        {
117.                if(sourceEntity instanceof Concept)
118.                        ((ConceptNode)sourceNode).validateVisibility(mafraGraph);
119.                else
120.                        ((PropertyNode)sourceNode).validateVisibility(mafraGraph);
121.        }
122.        if(targetNode != null)
123.        {
124.                if(targetEntity instanceof Concept)
125.                        ((ConceptNode)targetNode).validateVisibility(mafraGraph);
126.                else
127.                        ((PropertyNode)targetNode).validateVisibility(mafraGraph);
128.            }
129.        . . .
```

Como se pode depreender do código, logo após a remoção do *match* (linha 114), a extremidade que fica visível, que pode ser tanto a entidade origem como a destino, será alvo de verificação.

A implementação deste método é de grande importância na medida em que permite controlar, aquando de uma dada acção, não só a legitimidade gráfica do próprio *match*, como também a das entidades das suas extremidades.

4.5 "Validate All Entities"

Este método foi criado com o intuito de corrigir erros de comportamento das restantes entidades quando ocorre um comando "Hide" sobre uma ou mais entidades.

O seguinte código corresponde ao método "validateAllEntities":

```
130.    public void validateAllEntities() throws KAONException
131.    {
132.        Iterator it = m_entityNodes.iterator();
133.        while(it.hasNext())
134.        {
135.            Node node = (Node) it.next();
136.            if( node instanceof ConceptBridgeNode )
137.                ((ConceptBridgeNode) node).validateVisibility(this);
138.            else if( node instanceof PropertyBridgeNode )
139.                ((PropertyBridgeNode) node).validateVisibility(this);
140.            else if( node instanceof ConceptNode )
141.                ((ConceptNode)node).validateVisibility( this );
142.            else if( node instanceof PropertyNode )
143.                ((PropertyNode)node).validateVisibility( this );
144.            else if( node instanceof MatchNode )
145.                ((MatchNode) node).validateVisibility(this);
146.        }
147.    }
```

Este método é muito útil e pode até ser visto como um *refresh* não com o intuito de reposicionamento das entidades mas com o intuito de "pedir" a cada uma deles que verifique o ser estado de visibilidade.

4.6 *"Hide All Bridges"*

Em análise anterior feita ao comportamento desta acção concluiu-se que o mesmo era incompleto, em que apenas parte das *semantic bridges* eram ocultadas.

A causa desse mau funcionamento, deve-se ao facto de, a nível de código, se estar a tentar distinguir entre *concept bridges* e *property bridges*. Contudo, essa distinção é totalmente desnecessária atendendo ao facto de que a acção "Hide All Bridges" compreende tanto umas como outras.

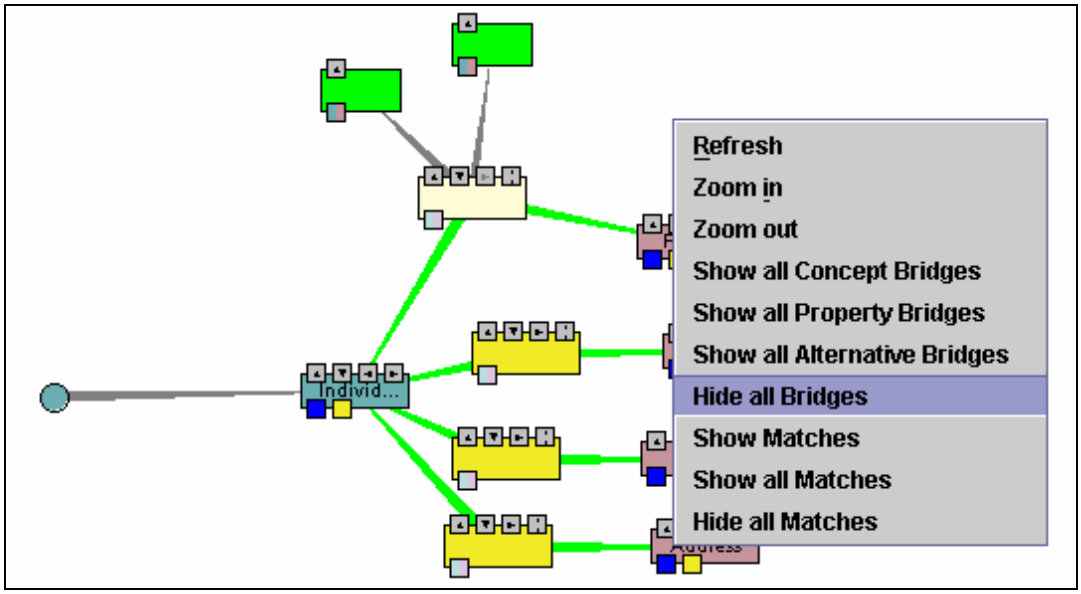

Figura 55 - Antes de Efectuar a acção "Hide All Bridges"

A Figura 55 demonstra o cenário inicialmente preparado para efectuar a acção "Hide All Bridges".

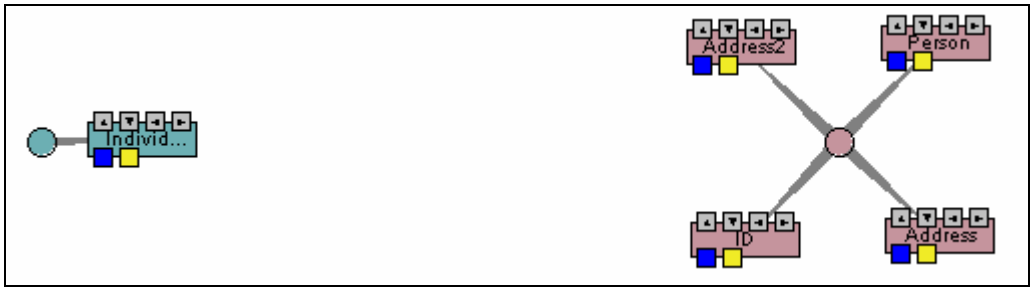

Figura 56 - Após Efectuada a acção "Hide All Bridges"

Repare-se que a Figura 56 mostra que o comportamento final observado é o esperado e diferente do comportamento anteriormente analisado.

A nível de código, o novo método “HideAllBridges” limita-se a construir uma lista de todas as *semantic bridges* visíveis, colocando depois essa lista como parâmetro na chamada ao método “hideNodes” que, por sua vez, estará encarregue de esconder as entidades que recebeu como parâmetro.

5. CONCLUSÃO

Os objectivos inicialmente propostos para este projecto assentavam claramente numa componente bastante prática que visava essencialmente o melhoramento do comportamento gráfico da aplicação MAFRA Toolkit.

Nessa fase inicial, esses objectivos foram-me demonstrados pelo meu orientador bem como as questões a resolver, inerentes aos mesmos.

Com o decorrer do projecto foram surgindo novos problemas e novas questões que foram alvo de discussão e reflexão crítica de modo a decidir sobre novos caminhos a tomar e novas soluções a implementar, de modo a que o caminho para atingir os objectivos iniciais fosse alargado em virtude de novos objectivos.

Desde cedo ficou bem patente a importância que teria a componente teórica deste projecto como sendo determinante para a componente prática. O processo de estudo da aplicação relativamente à sua estrutura e à sua implementação, foi sem dúvida o mais moroso mas também o mais importante. Isto é, a capacidade de aprendizagem sobre o MAFRA e MAFRA Toolkit por forma a interiorizar os seus conceitos e fundamentos básicos, foi crucial para o projecto.

É possível então concluir que um dos grandes objectivos deste projecto é a oportunidade que o aluno tem de adquirir conhecimento ao estudar e compreender estruturas e implementações já existentes, ganhando mais capacidade para sugerir novas implementações sólidas e concretas mediante a grande absorção de conhecimento sobre o cenário contextual.

A nível de resultados, propriamente ditos, foi possível levar a bom porto novas soluções que melhoraram, sem dúvida, o comportamento das entidades gráficas do MAFRA Toolkit e providenciando também a possibilidade ao utilizador de poder deliberar sobre esse comportamento.

Outro dos resultados práticos, foi a implementação de soluções de problemas que constituíam à partida erros de funcionamento e que condicionavam “a priori” a implementação de novas soluções.

Para concluir, este projecto não só permitiu encontrar soluções para as questões inerentes aos objectivos propostos, como também constituiu uma boa preparação para a vida profissional na medida em que proporcionou o estudo de um cenário real já existente, de modo a ser reavaliado e reutilizado não só para reestruturação de implementações já existentes como também para apresentação de novos métodos e novas ideias.

Desenvolvimento de um Interface Gráfico em Java Swing

6. BIBLIOGRAFIA

Microsoft Developer Network,

http://www.msdn.com/

OneSmartClick.Com, Java Programming,

http://www.onesmartclick.com/programming/java.html/

Source Forge,

http://sourceforge.net/

Project: Harmonise Mapping Framework,

http://sourceforge.net/projects/hmafra/

Nuno Silva - site pessoal,

http://www.dei.isep.ipp.pt/~nsilva/

Nuno Silva - Research and Development (R&D),

http://www.dei.isep.ipp.pt/~nsilva/id.htm

KAON – The KArlsruhe ONtology and Semantic Web tool suite,

http://kaon.semanticweb.org/

Java Swing: A Quick Tutorial for AWT Programmers,

http://www.apl.jhu.edu/~hall/java/Swing-Tutorial/

Ontology Working Group,

http://mged.sourceforge.net/ontologies/index.php/

Java Foundation Classes (JFC/Swing),

http://java.sun.com/products/jfc/

TortoiseCVS,

http://www.tortoisecvs.org/

Oracle.com - JDeveloper 10g,

http://www.oracle.com/tools/

Desenvolvimento de um interface gráfico em Java Swing

7. Anexo - Pré-Requisitos

7.1 Repositório SourceForge.net

O repositório da *SourceForge.net* fornece alojamento gratuito para milhares de projectos de desenvolvimento de *software open source*. Entre vários serviços fornecidos estão os serviços para os projectos alojados da *SourgeForge.net* e um número de serviços para a comunidade *open source*.

A *SourceForge.net* é parte da OSDN (*Open Source Development Network, Inc*). OSDN é uma subsidiária completamente possuída pela *VA Software Corporation.* Esta empresa produz a plataforma de *software* colaborativa da *SourceForge*, uma versão mais antiga, a qual é utilizada pela *SourceForge.net* para disponibilizar o seu *website*.

O MAFRA Toolkit está alojado neste repositório e por isso mesmo, é necessário a qualquer novo colaborador deste projecto que crie uma conta na SourceForge.net.

7.2 CVS

7.2.1 O que é?

CVS (*Concurrent Versions System*) é o sistema de controlo dominante de versionamento de software transparente de rede *open source*.

CVS é útil e acessível por toda a gente desde programadores individuais a grandes equipas distribuídas:

- O acesso do tipo cliente/servidor permite aos colaboradores ter acesso à última versão do código de qualquer local onde haja uma ligação à Internet.

- O seu modelo de *check-out*(*download*) relativamente ao controlo de versões, não permite conflitos artificiais bastante comuns neste tipo de modelo exclusivo.

- As suas ferramentas de cliente estão acessíveis na maior parte das plataformas.

7.2.2 Tortoise CVS

Note-se que pela Figura 57, a aplicação permite trabalhar com o CVS directamente através do explorador do Windows. Com o TortoiseCVS é possível fazer o *check-out* directo de módulos, fazer *updates*, *commits*(*uploads*) e ver diferenças de código carregando com o botão do lado direito do rato em cima dos ficheiros ou pastas pretendidas dentro do explorador. É possível ver o estado de um ficheiro(modificado ou não) através de uma camada colocada sobre o seu ícone no explorador. É possível etiquetar, ramificar, fundir, importar e ver os *logs* de um dado ficheiro.

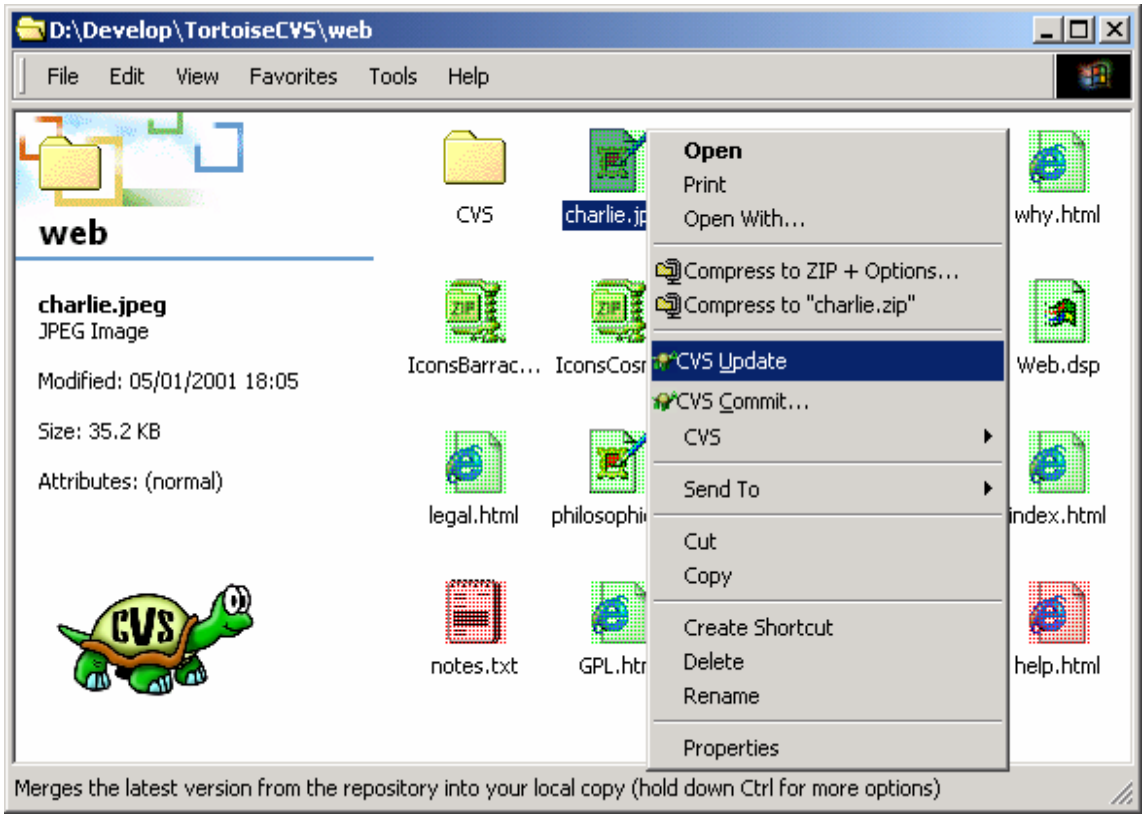

Figura 57 - Interface com o Tortoise CVS

O TortoiseCVS foi construído baseando-se no código fonte do WinCVS.

7.2.2.1 Configuração

Antes de utilizar as funcionalidades do TortoiseCVS em pleno, é necessário proceder-se à sua configuração para ser possível trabalhar com segurança e minimizar conflitos.

Para aceder às *sources* do mafra-toolkit é importante configurar o *TortoiseCVS* de maneira a povoar os campos com informação que permita aceder com sucesso ao pretendido.

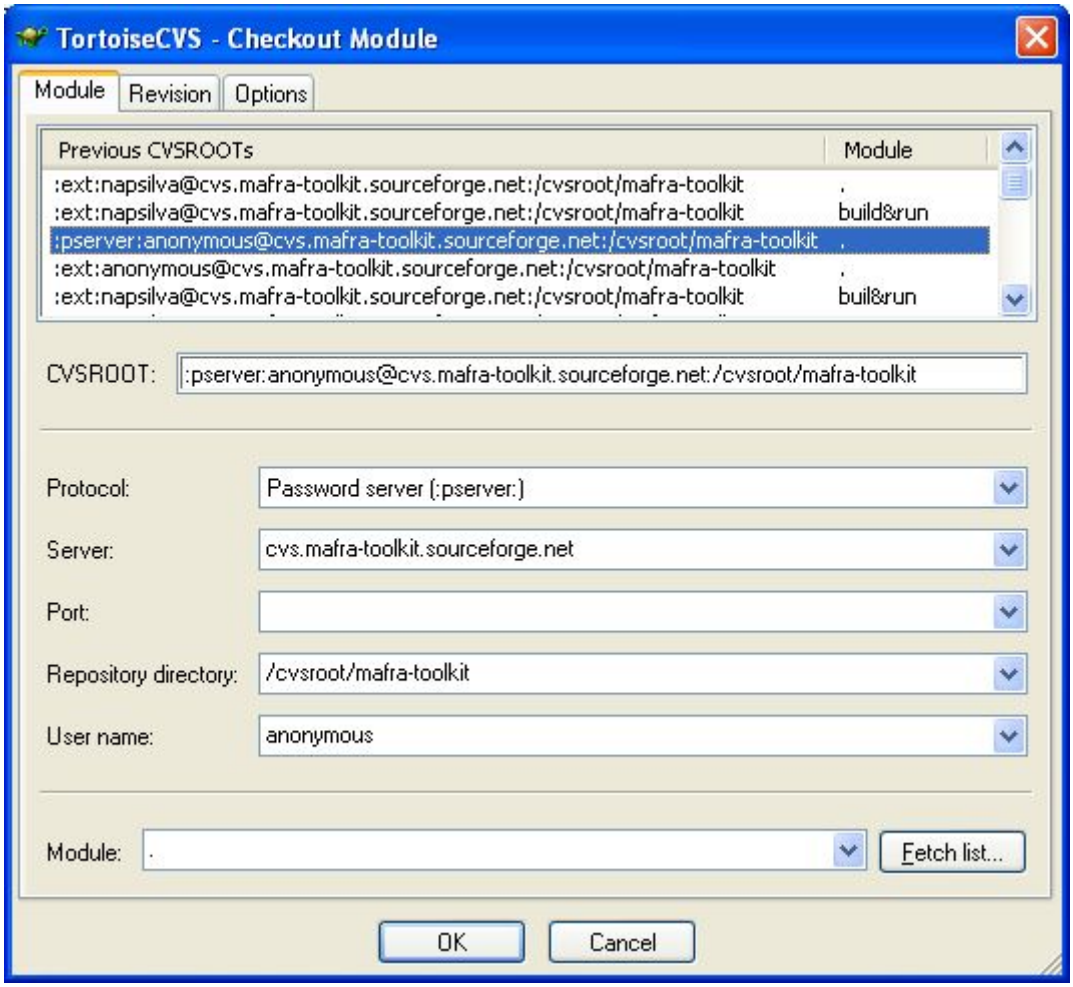

Figura 58 - Configuração do Tortoise CVS

É de salientar que no caso demonstrado na Figura 58 está-se a fazer um acesso restrito visto que se utiliza um "*username*" anónimo. Este tipo de acesso apenas permite fazer o *check-out* (*download*) das *sources*. Para se poder, depois de alterar ou não as *sources* efectuando o "*commit*" (*upload*), é necessário ter um acesso que permita realizar essa operação. Para isso, ter-se-á de utilizar a conta criada em

www.sourceforge.net, pedir permissões aos administradores do projecto e configurar o Tortoise CVS inserindo o nosso *username* e *password*.

7.3 Download da Aplicação MAFRA Toolkit

Uma vez instalado e configurado o TortoiseCVS, torna-se possível fazer o download da aplicação MAFRA Toolkit. Para isso é necessário criar uma nova pasta em qualquer área do disco duro e, no explorador, com o botão do lado direito do rato, fazer "CVS Check-out" para dar início ao download de todos os módulos de software que constituem a aplicação MAFRA Toolkit.

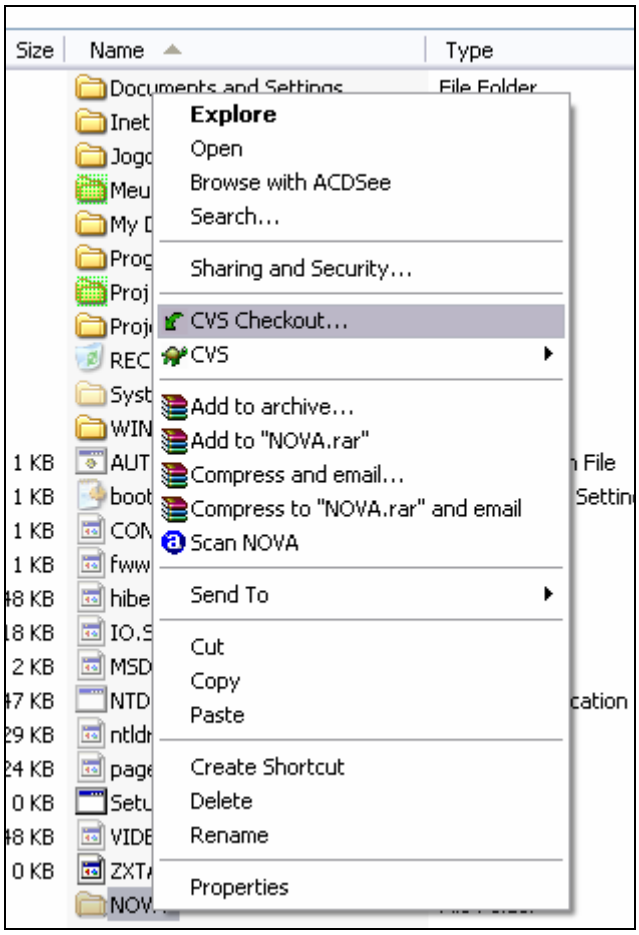

Figura 59 - Interface Tortoise CVS - check-out

Na Figura 59 está representado um exemplo de uma nova pasta criada através do explorador para efectuar o *checkout* da aplicação sobre ela, fazendo assim o *download* dos ficheiros da aplicação MAFRA Toolkit para dentro da mesma.

7.4 Ambiente de Desenvolvimento

Para levar a cabo a componente de análise e desenvolvimento deste projecto, torna-se relevante, para além do software atrás referenciado, encontrar um ambiente de desenvolvimento com capacidade.

7.4.1 Hardware/Software

O hardware utilizado acaba por ser uma componente importante na medida em que o comportamento gráfico do MAFRA Toolkit, a nível de velocidade de movimento das entidades, é directamente afectado por ela. Ou seja, a rapidez de posicionamento e reposicionamento das entidades gráficas varia de sistema para sistema, sendo talvez a capacidade do processador a principal responsável pelo aumento ou diminuição deste desempenho.

A nível de hardware este projecto foi implementado num sistema com os seguintes componentes relevantes:

- Processador : AMD Athlon 64
- Memória Principal : 1024 Mb DDR
- Placa Gráfica : Asus Geforce FX5200 128MB
- Disco : Seagate 160GB

A nível de ambiente de desenvolvimento integrado, inicialmente foi utilizado o Microsoft Visual J++ mas devido a problemas de compilação decidiu-se optar pela utilização do *Oracle JDeveloper 10g* para a realização da componente prática do projecto.

O *Oracle JDeveloper 10g* é indicado para o desenvolvimento de aplicações e *Web Services* e implementa já os últimos standards para o Java, XML e SQL.

Neste sentido, torna-se o ambiente ideal para criar e integrar no desenvolvimento da aplicação MAFRA Toolkit.

A nível de configuração do JDeveloper, para compilar as *sources* do MAFRA, é necessário configurar uma libraria para utilizar no mínimo a versão 1.4.2_01 do Java™ 2 Platform. Também neste caso o JDeveloper 10g se apresenta preparado, uma vez que já usa uma libraria com uma versão mais recente, tornando desnecessária esta configuração.