4
Testes do protótipo

Para testar a ferramenta, foi necessário implementar pelo menos um algoritmo de alinhamento e uma interface de usuário.

Escolhemos o algoritmo proposto por Leme [20], já que permite utilizarmos todas as funcionalidades expostas no capítulo 3 (como os passos, funções de similaridade e representação dos dados). A interface de uso recebeu o nome de *MatchmakingGUI* em alusão à abreviação GUI[8]. Além disso, criamos uma terceira ferramenta, a *MatchmakingBatch,* responsável por gerar alinhamentos de esquemas em lote. Essa última ferramenta foi integrada ao conjunto de comandos do *Matchmaking*.

4.1.
MatchmakingGUI – a interface gráfica do Matchmaking

A interface gráfica denominada *MatchmakingGUI* foi criada com o intuito de testar as funcionalidades da ferramenta descrita no capítulo 3. Essa interface faz uso da biblioteca de comandos (seção 3.2.3) do *Matchmaking* como canal de comunicação entre as partes.

Foi definido um conjunto limitado de oito casos de uso, considerados suficientes para avaliar o funcionamento da ferramenta. A Figura 17 ilustra o diagrama de casos de uso gerado. Cada caso de uso gerou uma tela na interface gráfica, descritas em maior detalhe nas próximas seções.

[8] *Graphical User Interface* – Interface gráfica do usuário

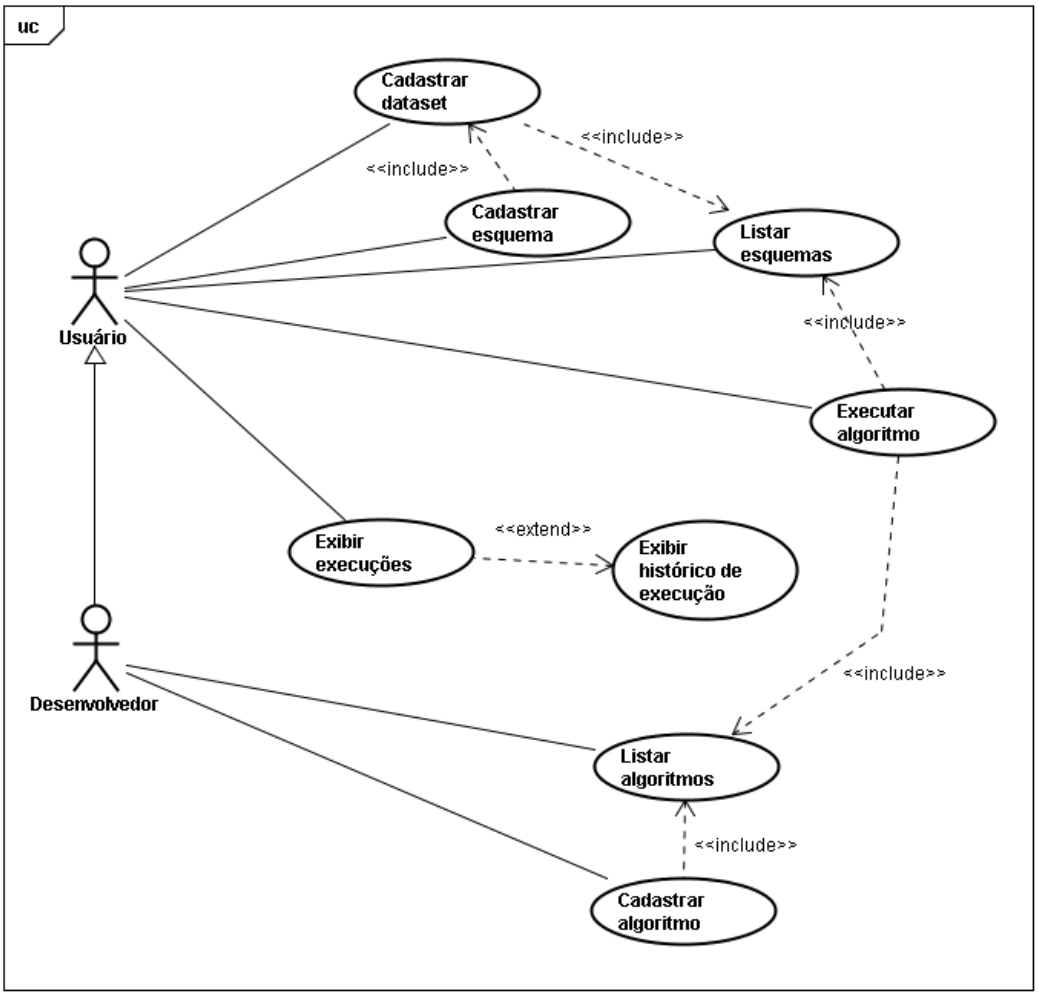


Figura 17 - Casos de uso do matchmakingGUI.

4.1.1. Listar esquema

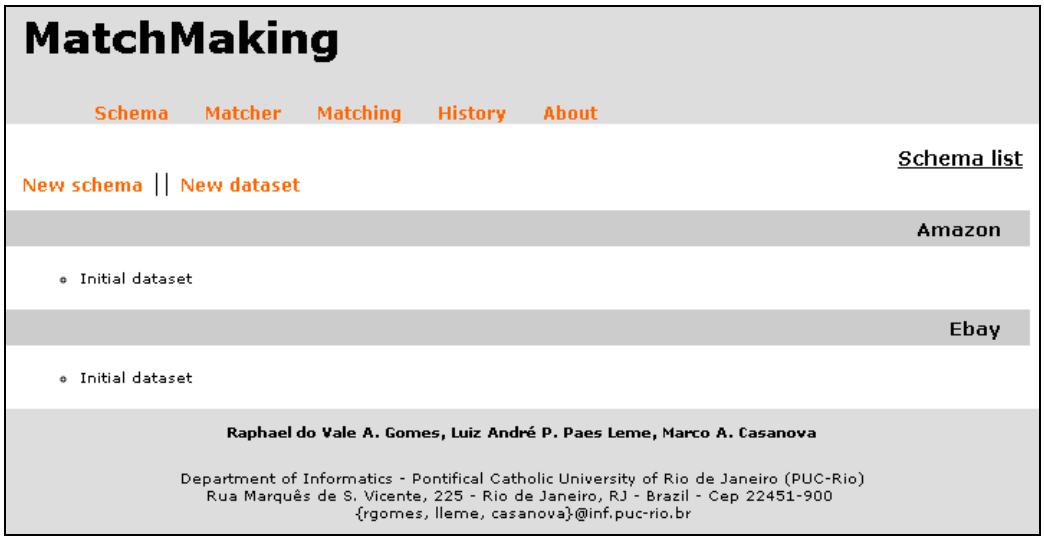


Figura 18 - Tela: listar esquema.

A tela “Listar esquemas” é responsável por apresentar todos os esquemas já cadastrados no *Matchmaking*. Para cada um destes, uma lista de *datasets* é apresentada por ordem de criação. A Figura 18 apresenta dois esquemas (*Amazon* e *eBay*) cada um com um *dataset* (*Initial dataset*). Esse *dataset* é criado no mesmo momento do esquema.

4.1.2. Cadastrar esquema

Figura 19 - Tela: cadastrar esquema.

A tela “Cadastrar esquemas” (Figura 19) permite o preenchimento de um nome para o esquema, o arquivo OWL e um ou mais *namespaces*. Apenas os elementos com *namespaces* cadastrados nessa tela serão considerados pela ferramenta. No ato do cadastramento do esquema, automaticamente a ferramenta também irá cadastrar um *dataset* de nome *Initial dataset*.

4.1.3. Cadastrar *dataset*

A tela “Cadastrar *datasets*” (Figura 20) é semelhante à tela “Cadastrar esquemas”. A diferença está em que não há cadastramento de *namespaces* (já realizado anteriormente) e é preciso escolher um dos esquemas previamente cadastrados. É importante notar que essa função só será bem sucedida se o *dataset*

for realmente do esquema. Para validar o atendimento a esta restrição, o *Matchmaking* compara elemento por elemento do esquema com os elementos apresentados no arquivo OWL enviado. Se houver mais ou menos elementos em qualquer um dos lados, a ferramenta informará erro. Em casos de evolução do esquema, pode haver uma perda ou ganho de elementos. Nesse caso, sugere-se que o usuário cadastre um novo esquema no *Matchmaking*, indicando, através do nome, que ele é uma evolução do anterior.

Figura 20 - Tela: novo *dataset*.

4.1.4. Listar algoritmos

Figura 21 - Tela: listar algoritmos.

A tela “Listar algoritmos” (Figura 21) apresenta todos os algoritmos cadastrados até o momento, informando quais são apenas passos (*Step Only*) e

quais não são. Nessa tela o usuário pode visualizar as informações de um algoritmo já cadastrado clicando sobre ele ou cadastrar um novo algoritmo.

4.1.5. Cadastrar algoritmo de alinhamento

A tela "Cadastrar algoritmo de alinhamento" apresentada na Figura 22 permite cadastrar um novo algoritmo. Como exibido no diagrama de casos de uso (Figura 17), esse recurso só deve ser utilizado pelo desenvolvedor do algoritmo já que requer informações disponíveis apenas pelo desenvolvedor. Um erro na etapa de cadastramento pode impossibilitar a execução do algoritmo.

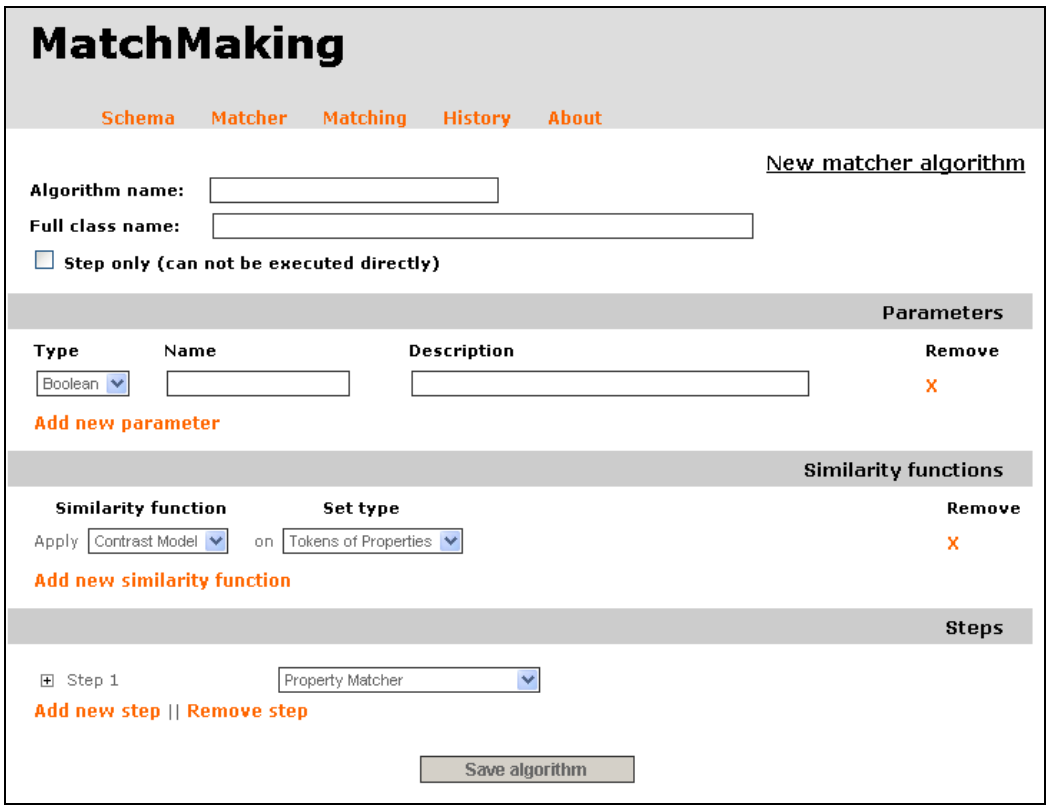


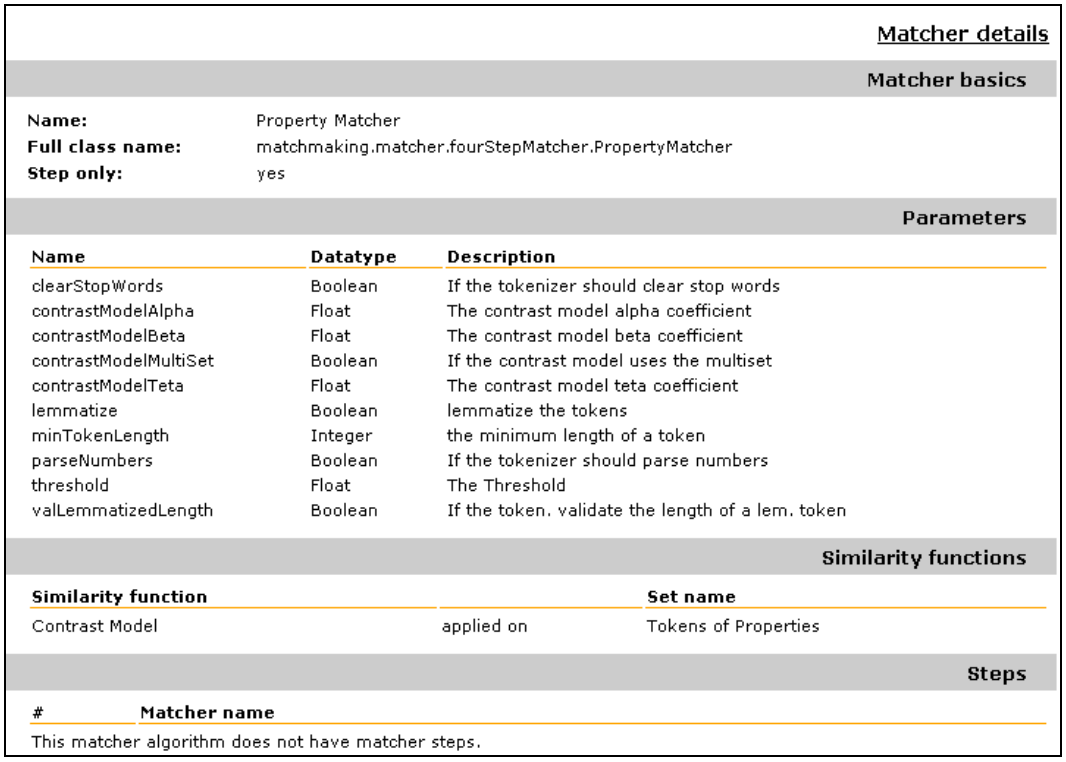
Figura 22 – Tela: cadastrar algoritmo de alinhamento.

O desenvolvedor precisa informar um nome que identifica o algoritmo, o nome completo da classe que estende *AbstractMatcher* e que será executada pelo *Matchmaking* quando for realizada uma operação de alinhamento, e informar se este algoritmo pode ser executado diretamente ou é apenas um passo de outro algoritmo. Na guia de parâmetros, o desenvolvedor deve informar cada um dos parâmetros necessários, o tipo de dados e uma descrição (opcional, para orientar o usuário no momento da operação de alinhamento). É de extrema importância que

o nome do parâmetro esteja correto, pois é através dele que o algoritmo deve solicitar um valor.

A guia de funções de similaridade permite que o desenvolvedor informe quais funções podem ser utilizadas pelo algoritmo de alinhamento: na primeira coluna o desenvolvedor deve informar a função e, na segunda, a representação de dados utilizada pela função. Se o algoritmo tentar utilizar um par composto por uma função de similaridade e uma representação de dados que não esteja cadastrada, o alinhamento será cancelado. A última guia apresenta os passos que um algoritmo pode utilizar. É possível informar um ou mais passos, mas é importante que a ordem seja coerente com a implementada pelo algoritmo. Cada passo está armazenado em um vetor e a ordem será a mesma deste cadastramento. A não utilização do nome do algoritmo foi uma decisão de implementação. Em uma situação hipotética, um mesmo algoritmo pode ser executado várias vezes em passos diferentes, com parâmetros diferentes, de modo que o nome do algoritmo não é uma opção.

4.1.6. Ver detalhes de um algoritmo



Matcher details

Matcher basics

Name:	Property Matcher
Full class name:	matchmaking.matcher.fourStepMatcher.PropertyMatcher
Step only:	yes

Parameters

Name	Datatype	Description
clearStopWords	Boolean	If the tokenizer should clear stop words
contrastModelAlpha	Float	The contrast model alpha coefficient
contrastModelBeta	Float	The contrast model beta coefficient
contrastModelMultiSet	Boolean	If the contrast model uses the multiset
contrastModelTeta	Float	The contrast model teta coefficient
lemmatize	Boolean	lemmatize the tokens
minTokenLength	Integer	the minimum length of a token
parseNumbers	Boolean	If the tokenizer should parse numbers
threshold	Float	The Threshold
valLemmatizedLength	Boolean	If the token. validate the length of a lem. token

Similarity functions

Similarity function		Set name
Contrast Model	applied on	Tokens of Properties

Steps

#	Matcher name

This matcher algorithm does not have matcher steps.

Figura 23 - Tela: ver detalhes de um algoritmo.

A tela “Ver detalhes de um algoritmo” (Figura 23) permite ao usuário conferir detalhes de um determinado algoritmo. Basicamente, a tela apresenta os mesmos dados que a tela de cadastramento (item 4.1.5), mas sem os campos de formulário. A informação é apenas para leitura.

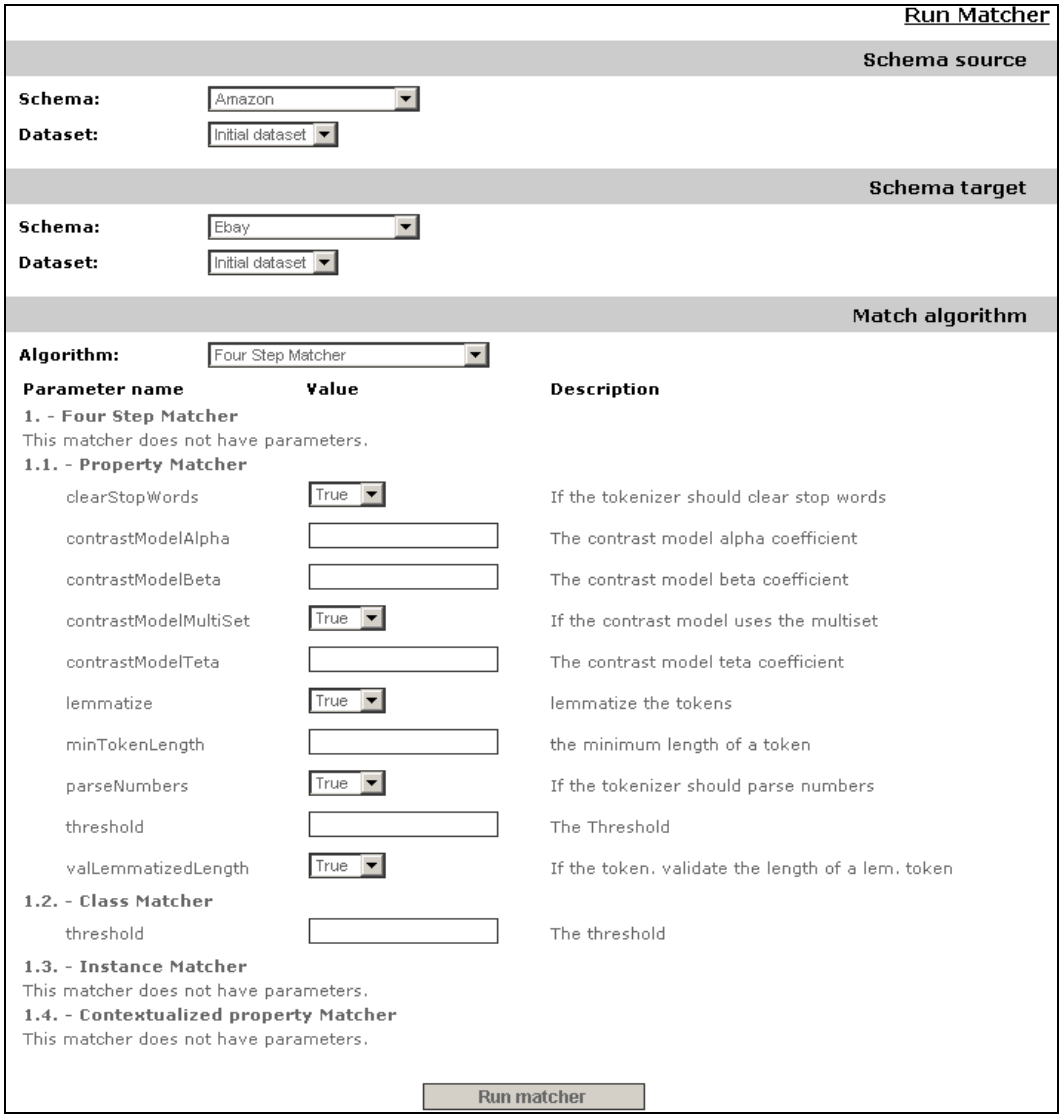


Figura 24 - Tela: iniciar operação de alinhamento.

4.1.7. Iniciar operação de alinhamento

A tela “Iniciar operação de alinhamento” (Figura 23), como o nome indica, permite iniciar uma operação de alinhamento de esquemas. O usuário deve escolher o esquema e *dataset* de origem e destino e o algoritmo. Depois de selecionado, o *MatchmakingGUI* automaticamente irá exibir as variáveis a serem preenchidas para cada um dos *passos* desse algoritmo. Se o parâmetro for do tipo booleano já será exibido uma caixa com as opções verdadeiro e falso.

Depois de tudo preenchido, o usuário deve solicitar a execução da operação. Vale lembrar que todos os parâmetros precisam ter valores e esses devem ser do tipo de dados informado pelo algoritmo. Se essa regra não for respeitada, o procedimento será cancelado.

4.1.8. Exibir execuções e histórico de execução

O conjunto de telas descritas nesse item é responsável por toda informação de proveniência do sistema. A Figura 25 apresenta a tela de histórico exibindo todas as operações de alinhamento já realizadas em ordem decrescente de data de processamento. Ela exibe o nome do algoritmo e a data e hora de início da operação.

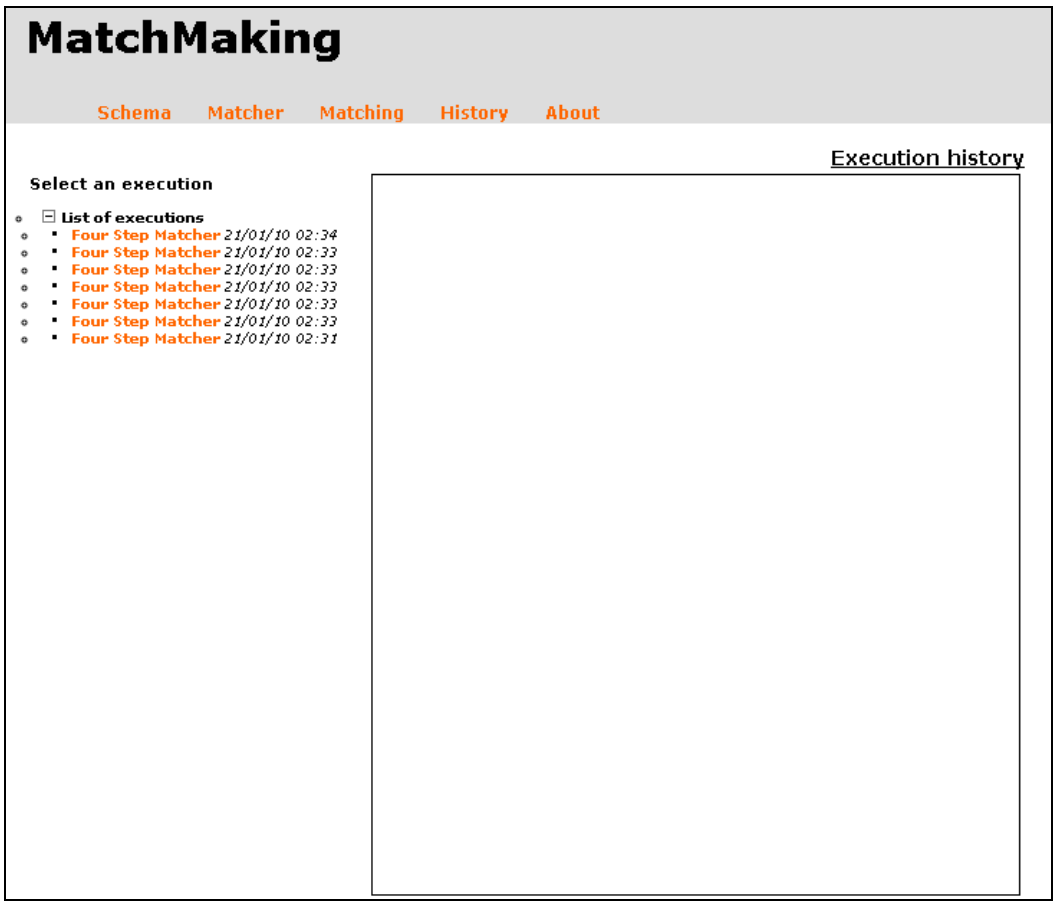


Figura 25 - Tela: exibir execuções.

Clicando sobre uma das operações de alinhamento, a caixa em branco da direita será preenchida com todas as informações existentes sobre o alinhamento. No caso da Figura 26, é apresentado a execução do *FourStepMatcher* (maiores

detalhes na seção 4.2). Nessa imagem, podemos verificar que toda a operação de alinhamento ocorreu em apenas nove segundos. Além disso, vemos Amazon e eBay (cada um utilizando seu *dataset Initial Dataset*) como esquemas origem e destino, respectivamente. Por fim, identificamos que nenhum parâmetro ou função de similaridade foi utilizado e, dos quatro passos listados, apenas o primeiro (*PropertyMatcher*) foi executado (por alguma decisão do algoritmo. Casos de erro são cancelados e não são apresentados). Também é possível verificar que não houve entradas de alinhamento registradas.

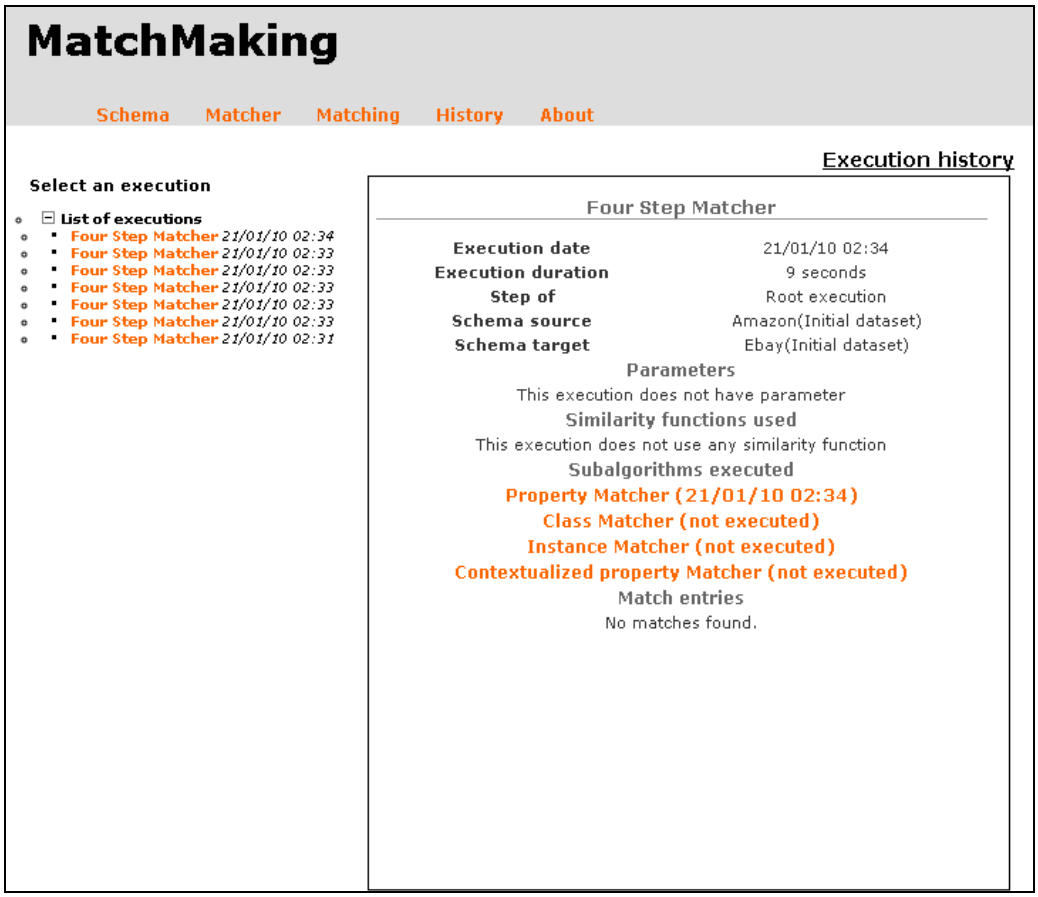


Figura 26 - tela: exibir histórico (1)

Clicando sobre os passos listados, é possível obter as mesmas informações detalhadas para cada um deles. No caso da Figura 27, foi solicitado o detalhamento do primeiro passo. Nesse caso, podemos ver que o tempo de processamento foi dos mesmos nove segundos (isso é coerente, já que os outros passos não foram executados), além da indicação de que esse algoritmo foi executado como um passo de outro (o *fourStepMatcher*). Também é possível ver a informação de cada um dos parâmetros informados pelo usuário e que funções de similaridade foram utilizadas (no caso aplicou-se a função *Contrast Model* na

representação de dados *Tokens of Properties*). Em seguida, temos a informação que não há passos para esse algoritmo (passos dentro de passos são algo permitido pelo *Matchmaking*). Por fim, nesse caso, houve entradas de alinhamento. Elas são exibidas no formato de quádruplas como descrito no capítulo 2.

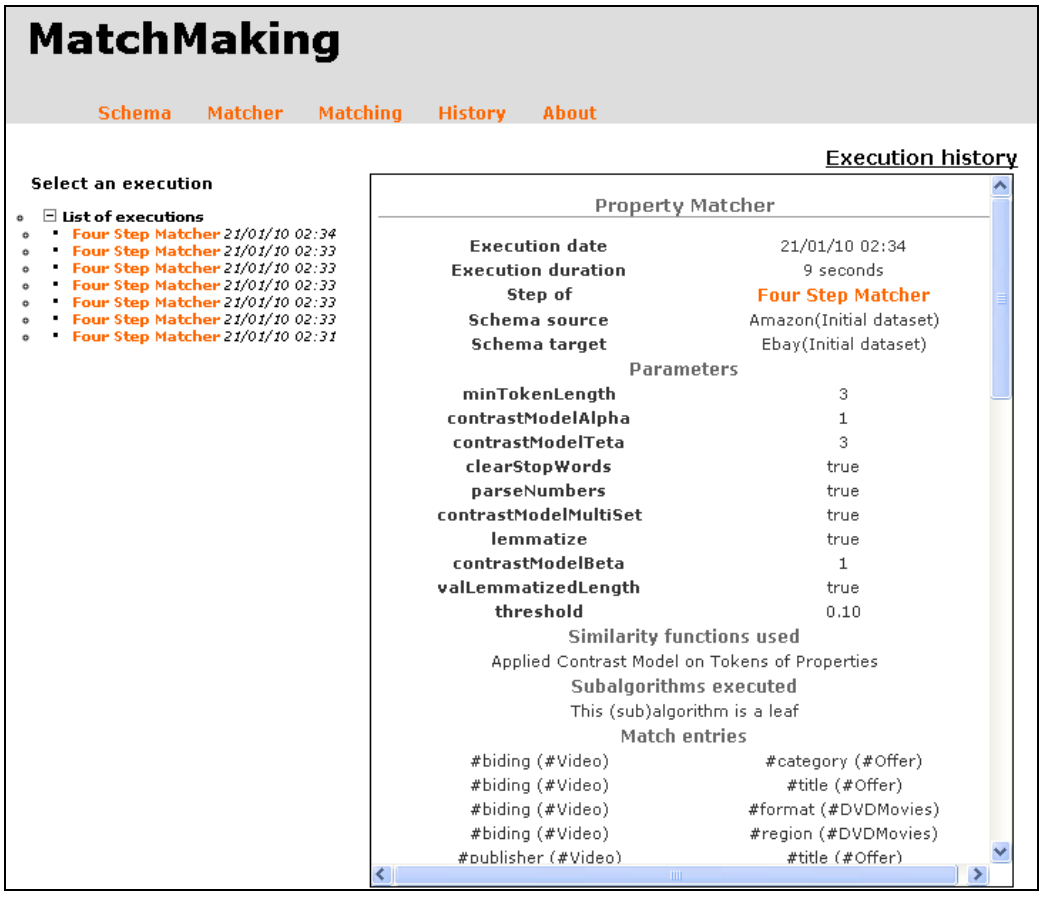


Figura 27 – Tela: exibir histórico (2).

4.2. Four Step Matcher – o algoritmo de alinhamento

Implementamos um algoritmo de alinhamento de esquemas para testar a ferramenta e validar se toda a biblioteca de código funcionava corretamente e se as funções disponíveis para os algoritmos eram suficientes e atendiam às necessidades.

Escolhemos o algoritmo proposto por Leme [20] por utilizar diferentes passos e funções de similaridade. Tomamos a liberdade de chamar o algoritmo de *FourStepMatcher* (alinhador de quatro passos) por questões sistemáticas. Para implementá-lo, criamos cinco classes (Figura 11 – página 38), que estendem *AbstractMatcher*: *FourStepMatcher*, *PropertyMatcher*, *ClassMatcher*,

InstanceMatcher e *ContextualizedPropertyMatcher*. Para o teste, desenvolvemos os três primeiros passos, conforme detalhado nas próximas seções.

4.2.1. FourStepMatcher

A classe *FourStepMatcher* é responsável por iniciar a execução de cada um dos passos e servir como meio de ligação entre cada um deles. Ou seja, é responsável pelo processo de comunicação entre os passos. A Figura 28 apresenta o diagrama de atividades do algoritmo. Nele, podemos ver como funciona a seqüência de atividades desse algoritmo: basicamente, para cada passo executado, a linha de execução volta à instância de *FourStepMatcher* para que possa tratar a informação e repassá-la ao próximo passo. Ao final da execução do quarto passo, o resultado final é gerado pela *FourStepMatcher*, normalizando as entradas de alinhamento obtidas nos passos (conforme descrito no item 2.2).

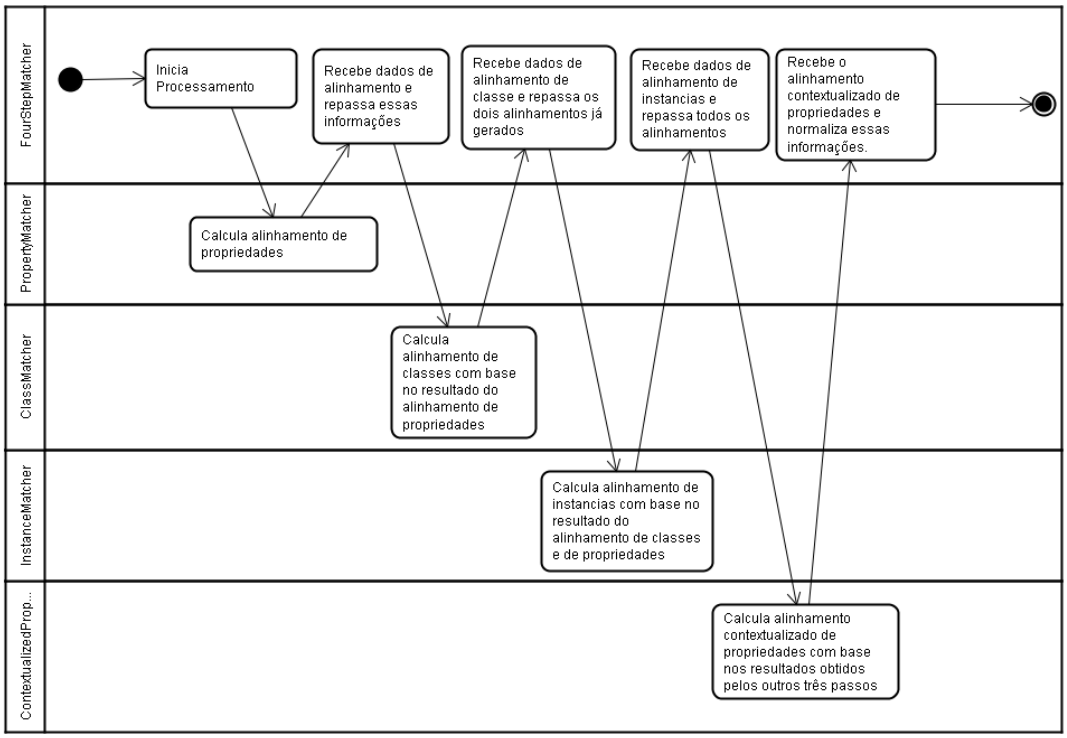


Figura 28 - Diagrama de atividades do *FourStepMatcher*.

4.2.2. PropertyMatcher

O *PropertyMatcher* é o primeiro passo do *FourStepMatcher*. Seu objetivo é conseguir alinhar as propriedades dos dois esquemas de modo que a margem de

erro seja mínima. Para isso, utiliza a função de similaridade *contrast model* aplicada a *tokens* obtidos dos valores de cada uma das propriedades. Intuitivamente, o *contrast model* calcula o grau de similaridade entre os elementos baseado no fato de que, quanto mais valores em comum, mais similares os elementos serão. A seção 2.2 apresenta mais detalhes sobre essa função de similaridade.

Para trabalhar com o *contrast model*, usamos a representação de dados *Tokens of Properties* que é responsável por obter todos os valores de cada uma das propriedades e gerar tokens. Ou seja, nesse caso, cada propriedade vai ter, como representação de seus dados, um conjunto de *tokens* derivado do seu conjunto de valores existente no *dataset*.

Para tornar o processamento mais personalizável de tal forma que seja possível avaliar possíveis variações, a *tokens of properties* possui uma série de parâmetros (Tabela 2) que podem ser informados. A variação desses parâmetros pode gerar resultados diferentes no resultado do alinhamento.

Tabela 2 - parâmetros de *tokens of properties*

Nome	Tipo de dado	Descrição
minTokenLength	Inteiro	Tamanho mínimo de um *token*. Se o tamanho for menor que esse valor, o *token* será ignorado.
parseNumbers	Booleano	Indica se valores numéricos devem entrar na lista de *tokens*.
clearStopWords	Booleano	Indica se *stopwords* (palavras muito comuns que pouco acrescentam semanticamente) devem ser ignoradas.
Lemmatize	Booleano	Indica se o *tokens of properties* deve converter um *token* para a sua estrutura mais básica (lexema)
valLemmatizedLength	Booleano	Indica se, após a conversão para lexema, o tamanho do *token* deve ser novamente validado.

A função de similaridade implementada também possui uma série de parâmetros que podem ser informados. A maioria desses parâmetros é melhor descrita na seção 2.2.

Tabela 3 - parâmetros de *contrast model*

Nome	Tipo de dado	Descrição
Teta	Double	Coeficiente θ da função
Alpha	Double	Coeficiente α da função
Beta	Double	Coeficiente β da função
C	Double	Variável utilizada para normalizar o resultado da similaridade.
Multiset	Booleano	Indica se um token repetido deve ser considerado como um elemento a mais (se for

		verdadeiro) ou só a primeira ocorrência é considerada.

Com a função de similaridade e a representação de dados pronta, a implementação do *PropertyMatcher* foi relativamente simples, bastando solicitar a geração dos dados pelo *tokens of properties* e depois repassar o conjunto de valores gerados para o *contrast model* e solicitar o grau de semelhança entre os elementos. Nessa implementação do primeiro passo, o algoritmo verifica o tipo de dados de cada uma das propriedades e somente solicita o calculo de similaridade se ambos forem de mesmo tipo de dados. Se o valor retornado pela função for maior do que o *threshold* (valor mínimo de corte) informado, ele será incluído como uma entrada de alinhamento.

O usuário não tem poder de preencher diretamente os valores de parâmetros da função de similaridade e da representação de dados. O preenchimento desses valores fica a cargo do algoritmo de alinhamento. No entanto, foi definido que o *PropertyMatcher* deve solicitar todos esses valores para o usuário (podemos ver isso na Figura 24 – página 53). O desenvolvedor é livre para decidir se esses valores devem ser preenchidos pelo usuário do algoritmo ou se o algoritmo já possuirá esses valores pré-definidos.

A Figura 29 ilustra a seqüência de operações do *PropertyMatcher* de forma simplificada. Após a solicitação de processamento do *FourStepMatcher*, o algoritmo cria uma instância para a origem e outra para o destino de *TokensOfProperties*. Depois, para cada um deles, solicitada a geração da representação de dados informando o esquema (que, no caso, já possui embutido qual *dataset* está sendo utilizado), o algoritmo que será executado e os valores dos parâmetros. Depois de geradas ambas as representações, o algoritmo irá instanciar o *contrast model*, que receberá informações de parâmetros, representação de dados e que algoritmo está sendo executado. Criada a instância, o *PropertyMatcher* pede todas as propriedades contextualizadas do esquema de origem. Para cada uma dessas, o algoritmo solicita as propriedades do esquema destino e verifica se ambas são de mesmo tipo de dados. Se forem, ele solicitará a comparação pela função de similaridade que retornará um valor indicando o grau de semelhança entre os dois elementos. Se esse grau for maior que o valor de corte, o algoritmo irá incluir esse par de propriedades como uma nova entrada de

alinhamento. Esse procedimento se repete até não existirem mais propriedades de origem para serem comparadas.

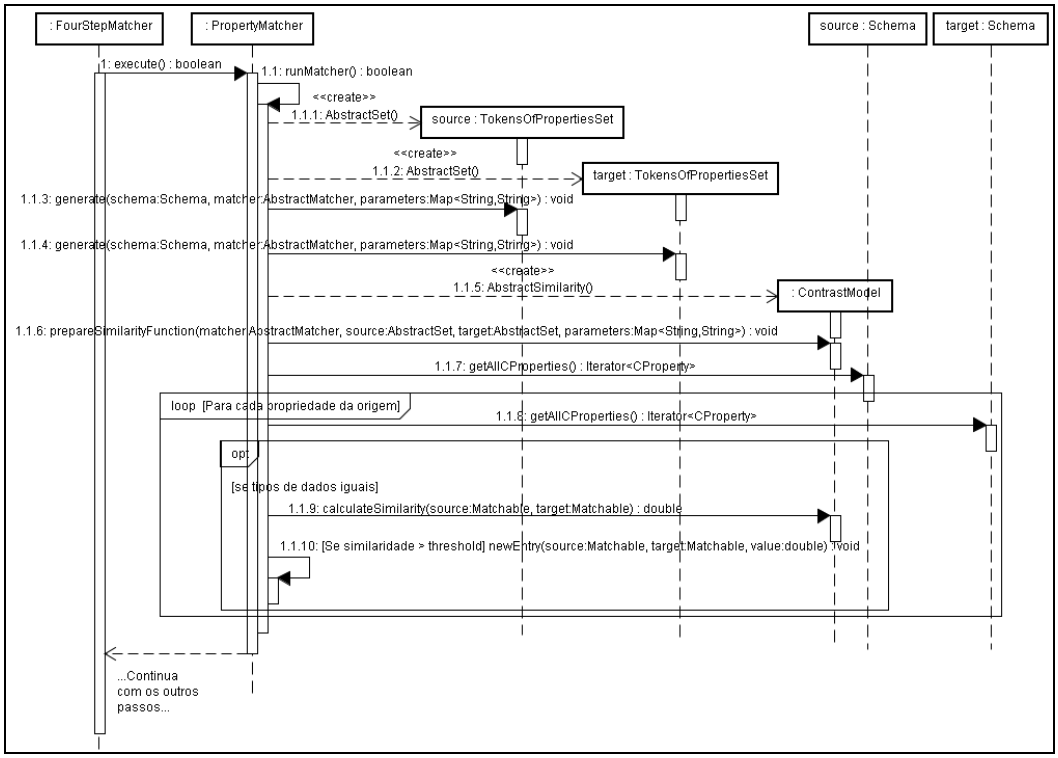


Figura 29 - Seqüência simplificada de operações do *PropertyMatcher*.

4.2.3. ClassMatcher

O *ClassMatcher* é o segundo passo do algoritmo implementado em nossos testes. Seu objetivo é conseguir, através dos dados obtidos no primeiro passo, alinhar classes dos dois esquemas. Para isso, ele deve usar uma representação de dados que indique quais propriedades da classe origem estão alinhadas com propriedades da classe destino. A *MatchablePropertiesOfClassesSet* foi criada com objetivo de gerar essa representação. No caso, utilizaremos o *Contrast Model* para alinhar as classes também.

Tabela 4 - Parâmetros de *MatchablePropertiesOfClasses*

Nome	Tipo de dado	Descrição
generateSource	Booleano	Indica se os dados gerados serão para o esquema de origem ou destino.
executionId	Inteiro	Indica o código da execução que será usado o alinhamento para gerar a representação.

A Tabela 4 apresenta os valores de parâmetros esperados pela representação de dados. Um fator importante é que a *MatchablePropertiesOfClasses* depende

dos dados de alinhamento gerados anteriormente. Além disso, o resultado gerado varia de acordo com o tipo de esquema que está sendo tratado (origem ou destino).

No caso, o conjunto de dados deve representar os elementos em comum entre duas classes dos dois esquemas. Para isso, seguimos o seguinte procedimento:

1. Se o esquema analisado é o de origem
    a. Para cada classe do esquema, obtemos suas propriedades e colocamos os códigos de identificação delas na representação de dados da classe.
2. Se o esquema analisado é o de destino
    a. Nesse caso, queremos relacionar as classes de destino com as propriedades de origem. Para isso, obtemos todo o conjunto de alinhamento da execução informada por parâmetro;
    b. Para cada classe do esquema de destino, obtemos suas propriedades e verificamos quais delas possuem pelo menos um alinhamento com uma propriedade do esquema de origem;
        i. Se ela possui alinhamento(s), colocamos o código de identificação das propriedades de origem que alinham com a propriedade de destino na representação da classe de destino;
        ii. Se não possui, colocamos o código de identificação da propriedade de destino na representação da classe de destino.

Com as representações geradas, o *Contrast Model* será capaz de avaliar quantas propriedades em comum e quantas diferentes cada classe de origem tem em relação a uma classe de destino. A geração dessa representação de dados é bastante eficiente já que usamos apenas os dados gerados pelo Matchmaking, sem precisar utilizar o *Jena*.

4.2.4. InstanceMatcher

O terceiro passo do algoritmo proposto tem o objetivo de alinhar as instâncias dos esquemas comparados. Intuitivamente, o algoritmo analisa as classes que foram consideradas equivalentes no passo anterior para comparar cada uma das instâncias da classe de origem com as da classe de destino. Para atingir maior precisão, o procedimento faz uso das informações obtidas no primeiro passo (alinhamento de propriedades) para comparar apenas as propriedades equivalentes entre as classes**.**

Para implementar esse passo, criamos a classe *InstanceMatcher* responsável pela execução e a representação *TokensOfInstancesSet* responsável por gerar o conjunto de dados que será comparado entre as instâncias.

A representação de dados criada gera *tokens* para cada instância da classe solicitada. Esses tokens são gerados apenas pelas propriedades que se alinham com as da classe comparada. Dessa forma, para cada par de classes, será gerado um conjunto de *tokens* de instância especifico para aquela comparação. Entretanto, se o mesmo conjunto de propriedades da instância for solicitado em outra comparação, o *Matchmaking* irá aproveitar o calculo anterior.

4.3. MatchmakingBatch – alinhamento de esquemas em lote

O *MatchmakingBatch* permite realizar mais de uma operação de alinhamento de esquemas de uma única vez, variando apenas os parâmetros dos algoritmos (e os próprios algoritmos) com objetivo de avaliar os resultados obtidos. A ferramenta está colocada como um comando interno do *Matchmaking* e, para ser utilizada, basta o usuário informar o caminho do arquivo XML que contêm as informações sobre a operação. Esse arquivo deve seguir o formato apresentado na Figura 30.

```
<matchmaking>
    <source id='81' dataset='1'></source>
    <target id='82' dataset='1'></target>
    <matching>
        <matcher id='71'> <!-- four step matcher -->
            <matcher id='63'> <!-- property matcher -->
                <parameter name='clearStopWords'
value='true'></parameter>
                <parameter name='contrastModelAlpha'
value='1'></parameter>
```

```
                <parameter name='contrastModelBeta'
value='1'></parameter>
                <parameter name='contrastModelMultiSet'
value='true'></parameter>
                <parameter name='contrastModelTeta'
value='3'></parameter>
                <parameter name='lemmatize' value='true'></parameter>
                <parameter name='minTokenLength' value='3'></parameter>
                <parameter name='parseNumbers' value='true'></parameter>
                <parameter name='threshold' value='0.15'></parameter>
                <parameter name='valLemmatizedLength'
value='true'></parameter>
            </matcher>
            <matcher id='64'> <!-- class matcher -->
            </matcher>
            <matcher id='65'> <!-- instance matcher -->
            </matcher>
            <matcher id='66'> <!-- contextualized property matcher -->
            </matcher>
       </matcher>
    </matching>
</matchmaking>
```

Figura 30 - Exemplo de alinhamento em lote.

O usuário deve informar inicialmente quais são os esquemas e *datasets* de origem e destino (através das suas chaves primárias). Depois, o usuário pode criar um novo nó do tipo *matching* para cada novo alinhamento que deseja fazer neste lote. Cada algoritmo e *passo* de algoritmo deve ser representado por um nó do tipo *matcher* que deve ter internamente todos os valores de parâmetros declarados pelo algoritmo e todos os passos adicionais, caso existam.

Ao solicitar a execução desse arquivo, o *Matchmaking* irá processar cada alinhamento em uma transação separada para que não se perca um alinhamento concluído e que poderia ser perdido se uma única transação para todos os procedimentos fosse utilizada. Para ver o resultado dos alinhamentos gerados pela ferramenta, o usuário deve ir para a tela de exibição de execuções (item 4.1.8).