4
Implementação do meta-alinhador

Para mostrar a viabilidade da proposta, foi implementada uma ferramenta de meta-alinhamento, denominada também de GNoSIS+. A ferramenta permite realizar o cálculo de similaridade entre conceitos de ontologias através da especificação de diferentes funções de similaridade a serem executadas. Como resultado, a ferramenta permite a geração automática ou manual de alinhamentos. A geração manual consiste na disponibilização de uma tabela ordenada de graus de similaridade calculada pela ferramenta para que o usuário possa escolher os alinhamentos corretos. Já a geração automática processa a tabela de similaridades e gera um arquivo de alinhamentos com as maiores similaridades encontradas.

Foi desenvolvido a ferramenta meta-alinhadora, o qual é apresentada neste capítulo com mais detalhes.

4.1
Arquitetura da ferramenta

A ferramenta GNoSIS+ foi desenvolvida como uma ferramenta de alinhamento de ontologias que pudesse utilizar diferentes algoritmos para cálculo de graus de similaridade. Assim, a ferramenta permite que novos algoritmos sejam incorporados ao cálculo de similaridades, à vontade do usuário.

A Figura 4.1 apresenta a arquitetura do sistema. A ferramenta é dividida em três grandes módulos (Módulo de Composição, Módulo Analisador e Módulo de Persistência), os quais podem possuir submódulos, além de uma API utilizada para extensão da ferramenta.

A ferramenta foi desenvolvida para processar ontologias descritas em linguagem OWL ou RDF, sendo sempre a entrada com pares de ontologias. Foi utilizada a linguagem Java e a API Apache Jena[1], a qual disponibiliza métodos básicos para leitura e gravação de documentos RDF ou OWL, além de permitir que máquinas de inferência (*reasoners*) sejam utilizadas para processamento das ontologias (Apache, 2011).

[1] http://incubator.apache.org/jena/

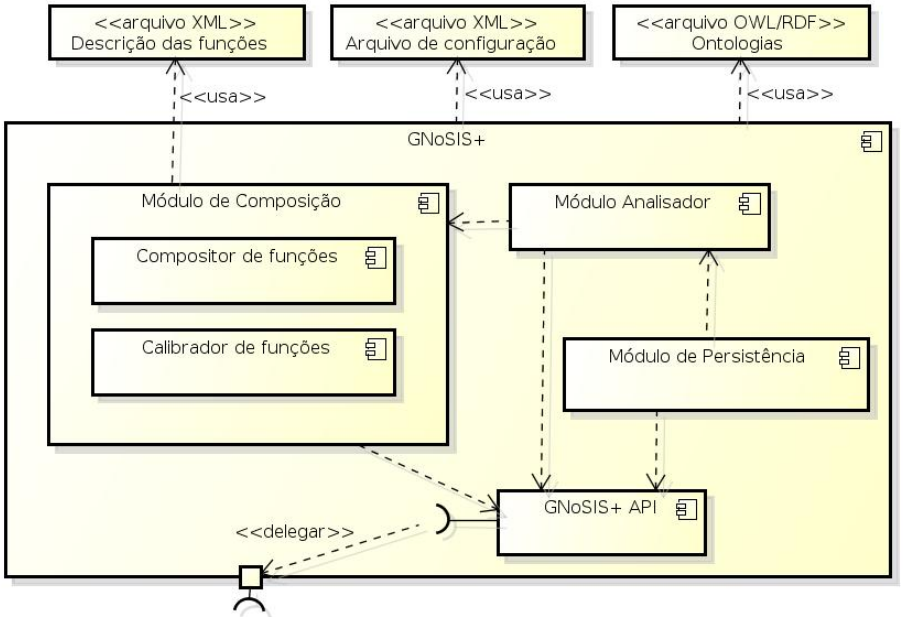

Figura 4.1: Arquitetura da ferramenta

Todos os três grandes módulos utilizam métodos disponíveis na GNoSIS+ API. A GNoSIS+ API pode ser considerada como uma camada de extensão da Jena API especializada para funções de similaridade. A GNoSIS+ API permite que a ferramenta GNoSIS+ possa ser integrada em outras aplicações e disponibiliza um conjunto de métodos para manipular os resultados calculados pelo GNoSIS+. Além disso, a API disponibiliza um conjunto de classes para que novas funções de similaridade possam ser integradas à ferramenta GNoSIS+.

As seções seguintes detalham cada módulo da ferramenta e, em cada seção, são também detalhadas algumas funcionalidades da API relacionadas ao módulo descrito na seção.

4.2 Módulo de composição

O módulo de composição de funções é responsável por compor as funções de acordo com a descrição das composições especificadas em um arquivo XML. A composição de funções é feita através da agregação de funções implementadas na ferramenta. A ferramenta GNoSIS+ já possui algumas funções de similaridade disponíveis e outras podem ser implementadas ou integradas com a GNoSIS+ API.

4.2.1 Arquivos de entrada

O GNoSIS+ recebe como entrada ontologias em formato OWL ou RDF, um arquivo com a descrição em formato XML das funções de similaridade que serão utilizadas para alinhar as duas ontologias de entrada e, opcionalmente, um arquivo com a descrição em formato XML dos alinhamentos conhecidos entre as duas ontologias de entrada.

A figura 4.2 exibe um arquivo XML simplificado para descrição de funções de similaridade. Neste arquivo, estão definidas as funções que serão utilizadas para alinhar as ontologias descritas pelo atributo *id* dos elementos XML *ontology*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<similarityCalc>
 <container name="Funcoes principais">
   <function weight="0.3">
     <class>br...ConceptNameSimilarity</class>
     <strategy>br...JaroWinklerEditDistance</strategy>
   </function>
   <container name="Funcoes Secundarias" weight="0.7">
     <function weight="0.5">
       <class>br...DirectSuperClassSimilarity</class>
       <combination>br...FirstMatchCombination</combination>
       <strategy>br...JaroWinklerEditDistance</strategy>
       <penalty>0.1</penalty>
     </function>
     <function weight="0.5">
       <class>br...DirectSubClassSimilarity</class>
       <combination>br...FirstMatchCombination</combination>
       <strategy>br...JaroWinklerEditDistance</strategy>
     </function>
    </container>
   </container>
   <ontologies>
    <ontology id="xml/Celo.owl"><concept>Cell</concept></ontology>
    <ontology id="xml/CeloModTermo.owl"><concept>Celula</concept></ontology>
    <pre-alignment>xml/examplesAlign.rdf</pre-alignment>
   </ontologies>
</similarityCalc>
```

Figura 4.2: Exemplo de documento XML para descrição de funções compostas

As funções de similaridade disponíveis na ferramenta são chamadas através do elemento XML *function*. Um *container* representa uma composição de funções ou containers. Caso o usuário deseje restringir quais entidades de cada ontologia terão seu grau de similaridade calculados, o arquivo XML conterá os elementos XML *concept* preenchido para cada entidade que se deseje alinhar. Por outro lado, caso o usuário deseje alinhar todas as entidades das duas ontologias, pode-se omitir o elemento XML *concept*.

Cada função do arquivo XML de descrição de funções possui, opcionalmente, seu peso associado (elemento XML *weight*). Caso os pesos de cada função ou container estejam definidos, o sistema utilizará esses valores para definir o grau de similaridade final de cada par de conceitos. Caso o usuário

deseje que o sistema calibre os pesos automaticamente, o elemento XML *pre-alignment* é preenchido com o caminho do arquivo de alinhamentos de exemplo fornecido pelos especialistas. A figura 4.3 exibe um trecho do arquivo de alinhamentos de entrada.

```
<Alignment>
  <xml>yes</xml>
  <level>0</level>
  <type>11</type>
  <onto1>http://gnosis.dcc.ufjf.br/Celo.owl</onto1>
  <onto2>http://gnosis.dcc.ufjf.br/CeloModTermo.owl</onto2>
  <uri1>http://gnosis.dcc.ufjf.br/Celo.owl</uri1>
  <uri2>http://gnosis.dcc.ufjf.br/CeloModTermo.owl</uri2>
  <map>
    <Cell>
      <entity1 rdf:resource="http://gnosis.dcc.ufjf.br/Celo.owl#Cell"/>
      <entity2 rdf:resource="http://gnosis.dcc.ufjf.br/CeloModTermo.owl#Celula"/>
      <measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
      <relation>=</relation>
    </Cell>
  </map>
  ...
</Alignment>
```

Figura 4.3: Exemplo de documento XML com alinhamentos de entrada

O arquivo de alinhamentos de exemplo descreve as correspondências conhecidas entre entidades das ontologias descritas por seus *namespaces* (elementos XML *onto1* e *onto2*) e por seu caminho físico (elementos XML *uri1* e *uri2*). O arquivo descreve um conjunto de elementos *Cell*. Cada elemento *Cell* representa uma correspondência entre duas entidades, onde os elementos XML *entity1* e *entity2* identificam uma correspondência entre a entidade da ontologia *onto1* e a entidade da ontologia *onto2*. O elemento XML *measure* descreve o grau de similaridade entre *entity1* e *entity2*. Por fim, o elemento XML *relation* descreve a relação entre as duas entidades, onde a relação denotada por "=" denota uma relação do tipo equivalência.

4.2.2 Funções disponíveis na ferramenta

Um conjunto de funções de similaridade foram implementadas e estão disponíveis para uso no GNoSIS+.

Funções de similaridade podem fazer uso de análise sintática para calcular o grau de similaridade entre entidades. Estas funções podem, então, incorporar qualquer medida de distância de termos. Assim, o usuário pode especificar para cada função, quando possível, qual algoritmo de distância de edição que será utilizado (ver elemento XML *strategy* da figura 4.2).

Foram implementados alguns algoritmos clássicos para medições sintáticas na ferramenta, as quais estão presentes na tabela 4.1.

Tabela 4.1: Algoritmos de distância de edição

Nome da classe	**Algoritmo**
DamerauLevenshteinEditDistance	Damerau-Levenshtein (Damerau, 1964)
EqualsEditDistance	–
HammingEditDistance	Hamming (Hamming, 1950)
JaroWinklerEditDistance	Jaro-Winkler (Winkler, 1999)
LevenshteinEditDistance	Levenshtein (Levenshtein, 1966)
TokenizerEditDistance	TFIDF (Robertson & Jones, 1976)

Cada classe da tabela 4.1 recebe duas cadeias de caracteres como entrada e computa a similaridade de acordo com cada algoritmo (ver apêndice A). Novos algoritmos para cálculo de edição de distância podem ser incorporados na ferramenta através da implementação da interface *IEditDistance*. Funções de similaridade que fazem uso de medidas de edição implementam uma interface chamada de *StringBasedFunction*.

Dentre essas funções de similaridade disponíveis no GNoSIS+, estão funções para analisar diferentes elementos de ontologias, como relacionamentos entre classes, instâncias, comentários de elementos, entre outros. As funções de similaridade atualmente disponíveis foram criadas para analisar somente um tipo de elemento de ontologias ou, em alguns casos, pequenos grupos de elementos. Além das funções disponíveis, novas funções podem ser integradas à ferramenta através da implementação de uma interface chamada de *ISimilarityFunction*. Assim, novos algoritmos mais complexos podem ser facilmente incorporados ao GNoSIS+. A tabela 4.2 apresenta as funções de similaridade disponíveis na ferramenta.

A seguir, é descrito o funcionamento básico de cada função de similaridade. Detalhes sobre os algoritmos, como seu pseudo-código, podem ser encontrados em (Souza et al, 2011).

- CommentSimilarity: Calcula a similaridade de duas entidades em relação à descrição contida nos campos de comentários das entidades.
- ConceptNameSimilarity: Calcula a similaridade de duas entidades em relação à similaridade entre seus termos;
- DirectCommonGraphSimilarity: Calcula a similaridade de duas entidades em relação aos identificadores comuns dos elementos que formam um subgrafo de todas suas relações;

Tabela 4.2: Funções de similaridade disponíveis no GNoSIS+

Nome da classe	**Elemento analisado**
CommentSimilarity	comentários
ConceptNameSimilarity	identificadores
DirectCommonGraphSimilarity	relacionamentos
DirectDataTypePropertybyNameSimilarity	propriedades
DirectDataTypePropertybyRangeEqualSimilarity	propriedades
DirectDataTypePropertybyRangeSimilarity	propriedades
DirectIndividualbyNameSimilarity	instâncias
DirectIndividualbyPropertySimilarity	instâncias
DirectIndividualSimilarity	instâncias
DirectObjectPropertybyNameSimilarity	propriedades
DirectObjectPropertybyRangeSimilarity	propriedades
DirectPropertybyNameSimilarity	propriedades
DirectPropertybyRangeDomainSimilarity	propriedades
DirectSubClassSimilarity	hierarquia
DirectSuperClassSimilarity	hierarquia

- DirectDataTypePropertybyNameSimilarity: Calcula a similaridade de duas entidades em relação à similaridade entre os termos de suas propriedades de tipo de dado;
- DirectDataTypePropertybyRangeEqualSimilarity: Calcula a similaridade de duas entidades em relação ao contra-domínio de suas propriedades de tipo de dado;
- DirectDataTypePropertybyRangeSimilarity: Calcula a similaridade de duas entidades em relação à similaridade entre os conjuntos que formam o contra-domínio de suas propriedades de tipo de dado.
- DirectIndividualbyNameSimilarity: Calcula a similaridade de duas entidades em relação à existência de instâncias com mesmo identificador;
- DirectIndividualbyPropertySimilarity: Calcula a similaridade de duas entidades em relação aos valores das propriedades de suas instâncias;
- DirectIndividualSimilarity: Calcula a similaridade de duas entidades em relação aos valores das propriedades e dos identificadores de suas instâncias;
- DirectObjectPropertybyNameSimilarity: Calcula a similaridade de duas entidades em relação à similaridade entre os termos de suas propriedades de tipo de objeto.

- DirectObjectPropertybyRangeSimilarity: Calcula a similaridade de duas entidades em relação à similaridade entre os termos que identificam o contra-domínio de suas propriedades de tipo de objeto.
- DirectPropertybyNameSimilarity: Calcula a similaridade de duas entidades em relação à similaridade entre os termos que identificam suas propriedades de tipo de dado e de objeto;
- DirectPropertybyRangeDomainSimilarity: Calcula a similaridade de duas entidades em relação à similaridade entre os conjuntos que formam o contra-domínio e domínio de suas propriedades de tipo de dado e de objeto;
- DirectSubClassSimilarity: Calcula a similaridade de duas entidades em relação à similaridade entre o conjunto de suas subclasses (ou super-propriedades, no caso de relacionamentos);
- DirectSuperClassSimilarity: Calcula a similaridade de duas entidades em relação à similaridade entre o conjunto de suas superclasses (ou sub-propriedades, no caso de relacionamentos).

4.3 Módulo analisador

O módulo analisador é responsável por calcular os graus de similaridade de acordo com a composição especificada pelo usuário e recebida do módulo de composição. Neste ponto, os pesos para cada função de similaridade já estão estabelecidos, seja por definição do usuário ou pelo calibrador de funções.

Sendo duas ontologias o e o', o analisador $a(e, e') = s$ gera um conjunto T de tuplas, tal que $|T| \leq |o||o'|$, uma vez que, $\forall e \forall e'(e, e', s) \in T$ se, e somente se, $s \geq \gamma, e \in o, e' \in o'$. O valor γ é denotado de *threshold* ou *valor mínimo de confiança* do grau de similaridade. O usuário pode especificar o valor mínimo em um arquivo de configuração do sistema. Caso não seja especificado, então $\gamma = 0$.

Para preencher o conjunto T, o analisador considera, ainda, os tipos de *combinação* e os valores das *penalidades* especificados pelo usuário.

4.3.1 Tipos de combinação

Neste trabalho, *combinação* é o processo de escolha de alinhamentos que uma função de similaridade realiza para calcular o grau de similaridade entre duas entidades.

Funções de similaridade que analisam subclasses de uma classe, por exemplo, podem gerar um grau de simlaridade de acordo com a média de todas as similaridades calculadas entre todas as subclasses de cada classe. Assim, suponha que uma classe c possua duas subclasses sc_1 e sc_2. Da mesma forma, uma classe c' possua duas subclasses sc'_1 e sc'_2. Ao se calcular as similaridades entre as subclasses, tem-se as tuplas $(sc_1, sc'_1, s_1), (sc_1, sc'_2, s_2), (sc_2, sc'_1, s_3), (sc_2, sc'_2, s_4)$. Ou seja, uma função de similaridade pode calcular a similaridade entre c e c' como a média aritmética das similaridades calculadas para cada subclasse:

$$\frac{s_1 + s_2 + s_3 + s_4}{4} \tag{4-1}$$

Uma outra forma de cálculo da similaridade entre c e c' pode ser feita como a média entre os graus de similaridade sem repetição de entidades. Dessa forma, haveria duas formas de escolha de tuplas: $\{(sc_1, sc'_1, s_1), (sc_2, sc'_2, s_3)\}$ ou $\{(sc_1, sc'_2, s_2), (sc_2, sc'_1, s_4)\}$. Ou seja, como cada classe possui duas subclasses, a similaridade entre essas classes seriam calculadas como a média da similaridade de duas tuplas somente.

Contudo, a escolhas das tuplas que formarão a similaridade final da função de similaridade pode ser realizada de diferentes formas. O GNoSIS+ permite que funções de similaridade possam ser implementadas utilizando estratégias de combinação (ver elemento XML *combination* na figura 4.2). Estas estratégias podem ser utilizadas por quaisquer funções de similaridade que calculam a similaridade através de um conjunto de similaridades, subclasses, superclasses, propriedades, instâncias, entre outros.

Duas estratégias estão implementadas na ferramenta, disponibilizadas pelas classes *FirstMatchCombination* e *DeepCombination*.

A classe FirstMatchCombination implementa uma estratégia de combinação gulosa. A estratégia consiste em ordenar o conjunto de tuplas de forma descrescente por grau de simlaridade. A partir de então, o algoritmo insere a primeira tupla no conjunto S de tuplas escolhidas, inicialmente vazio. Em seguida, a lista de tuplas é lida de cima para baixo e é escolhida uma nova tupla (x, y, z) de tal forma que $\forall(e, e', s) \in S, x \neq e \wedge y \neq e'$. A tupla escolhida é inserida em S. O processo termina quando não há mais tuplas a serem escolhidas. Embora seja uma estratégia simples, ela é uma estratégia rápida, uma

vez que não necessita verificar outras combinações de tuplas.

Considere, por exemplo, a tabela 4.3 que apresenta um conjunto de tuplas, onde cada tupla representa o grau de similaridade entre uma entidade e e e'.

Tabela 4.3: Conjunto de tuplas

$(e_1, e'_2, 0.8)$
$(e_2, e'_2, 0.7)$
$(e_1, e'_3, 0.6)$
$(e_2, e'_1, 0.4)$
$(e_1, e'_1, 0.3)$
$(e_2, e'_3, 0.3)$

Exemplo 4.1 *Seja o conjunto ordenado de tuplas da tabela 4.3, a aplicação da estratégia de combinação gulosa dá-se pelos seguintes passos:*

1. *escolha da tupla* $(e_1, e'_2, 0.8)$
2. *inserção da tupla* $(e_1, e'_2, 0.8)$ *em* S
3. *escolha da tupla* $(e_2, e'_1, 0.4)$*, uma vez que as tuplas anteriores possuem elementos* e_1 *ou* e'_2 *que já estão presente em* S*.*
4. *inserção da tupla* $(e_2, e'_1, 0.4)$ *em* S

Ao final do processo, $S = \{(e_1, e'_2, 0.8), (e_2, e'_1, 0.4)\}$ *e a similaridade final é dada por* $(0.8 + 0.4) \div 2 = 0.6$*.*

A estratégia gulosa não garante o melhor resultado. Vale ressaltar que encontrar o subconjunto de T que retorne o maior valor total é um problema NP completo (Souza, 1986) e pode ser impraticável calcular esse subconjunto quando a quantidade de tuplas for muito elevada.

A classe *DeepCombination* implementa uma estratégia força-bruta que testa todas as soluções. Esta estratégia garante um melhor resultado em detrimento do desempenho. Seja um conjunto ordenado de tuplas T, são criados todos os possíveis subconjuntos S_i de soluções viáveis de T, onde, para cada tupla $t = (t_e, t_{e'}, t_s)$ e $w = (w_e, w_{e'}, w_s)$ pertencentes a S_i temos que $t_e \neq w_e \wedge t_{e'} \neq w_{e'}$. Após criar todas as combinações possíveis entre as tuplas, o algoritmo seleciona o subconjunto S_i que possui a maior similaridade final.

Exemplo 4.2 *Seja o conjunto ordenado de tuplas da tabela 4.3, a estratégia de combinação força-bruta cria todos os possíveis subconjuntos* S_i *de soluções*

viáveis de T. Ao final do processo, para este exemplo, o subconjunto $S = \{(e_2, e'_2, 0.7), (e_1, e'_3, 0.6)\}$ *é escolhido, onde a similaridade final é dada por* $(0.7 + 0.6) \div 2 = 0.65$ *e não existe outro subconjunto com maior similaridade final.*

Novas estratégias de combinação podem ser introduzidas na ferramenta através da implementação de uma interface chamada de *ICombination*.

4.3.2 Penalidades

Quando é utilizado o processo de combinação em funções de similaridade que analisam conjuntos de entidades, conforme visto na seção 4.3.1, pode-se definir uma penalidade especificada pelo usuário (ver elemento XML *penalty* na figura 4.2). A penalidade pode ser considerada como um peso negativo aplicado a conjuntos de entidades que possuam diferença na sua cardinalidade.

Esse tipo de penalidade pode ser aplicado, por exemplo, em funções de similaridade que avaliam propriedades de uma classe. Caso a classe c possua x propriedades e o outra classe c' possua y propriedades, onde $x \neq y$, então o engenheiro de ontologias pode decidir penalizar o grau de similaridade calculado para essas duas classes.

Assim, seja uma penalidade Φ e g uma função que soma os graus de similaridade gerados pela função de combinação de ocorrências c, a qual combina as ocorrências de uma função de similaridade f aplicada a duas entidades e e e', o grau de similaridade final ρ de uma função de similaridade que permite utilizar penalidades é calculado como na equação 4-2.

$$\rho(e, e') = \frac{g(c(f(e, e')))}{l_{min} + \Phi(l_{max} - l_{min})} \tag{4-2}$$

Para compreender a equação acima, considere n_e o número de elementos relacionados à entidade e que foram analisados para calcular a similaridade $F(e, e')$ e $n_{e'}$ o número de elementos relacionados à entidade e' que foram analisados para calcular a similaridade $F(e, e')$. Por exemplo, seja e uma classe que possui 3 subclasses e e' uma classe que possui 4 subclasses e $F(e, e')$ uma função de similaridade que analisa a similaridade entre duas classes através de uma função de similaridade f aplicada às subclasses de e e e'. Neste exemplo, teríamos $n_e = 3$ e $n_{e'} = 4$. As variáveis l_{min} e l_{max} representam, respectivamente, o número mínimo e o número máximo de elementos relacionados com cada entidade, ou seja, $l_{min} = min(n_e, n_{e'})$ e $l_{max} = max(n_e, n_{e'})$.

A penalidade Φ é um número real pertencente a [0,1]. Caso $\Phi = 0$, não importa a diferença no número de elementos que possuem uma dada relação com uma entidade, resultando numa simples média aritmética. Caso contrário, a diferença no número de elementos é relevante para o cômputo da similaridade entre os elementos e, quanto maior o valor da penalidade, menor será o valor final de ρ.

Ressalta-se que penalidades podem ser aplicadas às funções que analisam qualquer conjunto de entidades que se relacionam com uma dada entidade, ou seja, pode ser aplicadas às funções que analisam conjuntos de instâncias, hierarquia, valores de propriedades, entre outras. A API do GNoSIS+ fornece uma classe abstrata *PenaltyFunction* que contém métodos básicos para esses tipos de funções, permitindo que penalidades possam ser aplicadas a novas funções de similaridade inseridas na aplicação.

4.4 Módulo de persistência

O módulo de persistência é responsável por persistir o conjunto de tuplas recebidas do módulo analisador. O GNoSIS+ possui classes para persistir em formatos TXT, HTML e AF (*Alignment Format*).

O AF é um formato definido por Euzenat (2003, 2004) que, segundo o autor, é um formato simples para que a maioria dos alinhadores consiga produzir alinhamentos neste formato. O arquivo XML apresentado na figura 4.3 é um exemplo de arquivo de alinhamento em formato AF. Novos formatos podem ser acrescentados na ferramenta através da API GNoSIS+.

Por fim, as correspondências podem ser processados pelo módulo de persistência para gerar alinhamentos de aridade $1:1$, $1:n$ ou $n:n$. A aridade do alinhamento pode ser escolhida pelo usuário através de um arquivo de configuração, sendo a aridade $1:1$ o padrão do sistema. Outras aridades podem ser implementadas através da API GNoSIS+.