# 4
# O Framework MsA – Materialização de sameAs

Motivados pelo objetivo de facilitar a tarefa de interligar URIs diferentes que representam uma mesma entidade na Web de Dados, desenvolvemos o Framework MsA – Materialização de sameAs: uma ferramenta responsável pela manutenção de ligações owl:sameAs entre uma base de dados local e dados publicados na Web. Sua utilização pode ser bastante útil para incentivarmos a publicação desse tipo de ligação. Este capítulo apresenta uma descrição completa desta nova ferramenta, incluindo motivação, especificação, arquitetura, e detalhes de seu funcionamento e implementação.



## 4.1 Motivação do Framework MsA

### 4.1.1. O problema da manutenção de ligações owl:sameAs

Atualmente, a publicação de dados na Web de Dados é realizada a partir de triplificações de bases de dados locais. Para interligá-los com dados externos, uma prática é realizar consultas por URIs de entidades externas semelhantes em cada processo de triplificação. Essas entidades, quando identificadas, são publicadas como objeto de ligações owl:sameAs. Porém, em uma estratégia ingênua, essas ligações são geradas novamente sempre que ocorrer a triplificação dos dados da base, mesmo que não tenham ocorrido mudanças significativas desde a última publicação.

Para evitar trabalho desnecessário, a solução seria armazenar as ligações owl:sameAs juntamente com os dados da base local a ser publicada. Para aplicar essa solução, desenvolvemos o Framework MsA – Materialização de sameAs.

O framework automatiza as tarefas de identificação de ligações, materialização das ligações identificadas, e manutenção das ligações owl:sameAs materializadas. O processo envolvido em cada uma dessas tarefas será detalhado a seguir.

### 4.1.2. Identificação

Para materializar ligações, é necessário identificar seus sujeitos e objetos. Ou seja, quais registros locais geram URIs locais e quais são suas URIs externas correspondentes.

Com esse objetivo, o framework realiza a identificação dos registros locais a partir de gatilhos no banco de dados. Eles identificam novos registros, além de atualizações e remoções.

Para a identificação das URIs externas, foi necessária uma solução genérica, na qual o usuário pudesse definir módulos específicos para realizar consultas externas em diferentes domínios. Para isso, precisamos de estratégias para identificar o módulo correspondente a um determinado registro, e a partir do SGBD, dispará-lo. Por exemplo, poderíamos executá-lo na função associada ao gatilho que identifica a inserção de novos registros. Todavia, ela só finalizaria a execução após o módulo terminar com sucesso. Isso pode ser um problema, por exemplo, quando o acesso à Web estiver indisponível. Nesse caso, a transação que disparou o gatilho poderia nunca terminar.

Esse problema poderia ser evitado disparando o módulo em segundo plano. Porém, cada execução geraria um processo “fantasma” no sistema operacional, uma vez que eles nunca seriam finalizados pela aplicação que os disparou.

A fim de evitar os problemas anteriores, optamos por uma solução independente do SGBD. Um programa permanece executando em segundo plano e buscando por novos registros, atualizações e remoções na base de dados. Ao identificar alguma dessas situações, executa o módulo correspondente. Por fim, materializa a ligação encontrada. Todos esses passos podem ocorrer sem causar interferências maiores em transações na base de dados, e sem sobrecarregar o servidor com processos abandonados.

### 4.1.3. Materialização

Ao identificar uma URI externa de uma entidade equivalente a algum dos registros da base de dados interna, essa URI deverá ser armazenada localmente. Ao armazená-la, teremos a materialização de uma ligação owl:sameAs na base local.

Para buscar essa URI externa, utilizamos o valor de alguma coluna, pertencente a alguma tabela da base do usuário. Esse valor corresponde ao registro da entidade que será o sujeito da ligação owl:sameAs a ser publicada futuramente. Sendo assim, também é necessário guardar a referência do registro, e da tabela/coluna, que originou a materialização.

Uma vez que esse registro pode ser de qualquer tabela existente na base, o desafio é criar uma chave estrangeira que interligue a URI externa materializada com qualquer tabela/coluna da base de dados local. Em um SGBD comum, para a criação de chaves estrangeiras, é necessário especificar tabela e coluna referenciadas, além de só permitir chaves entre colunas que possuam um mesmo tipo de dados. Por exemplo, se fizéssemos referências às chaves primárias das tabelas do usuário, teríamos uma solução amarrada à estrutura dessas tabelas.

Também não é possível criar mais de uma chave estrangeira em uma única coluna de uma tabela, o que impede que um único identificador referencie todas as demais tabelas do usuário (afinal, a ligação materializada pode ser de qualquer uma delas).

Essas restrições foram um desafio para a elaboração de uma solução genérica e pouco intrusiva que armazenasse as correspondências entre ligações materializadas e qualquer registro da base de dados local. Consideramos como não intrusivas as soluções que fazem o mínimo de alterações possíveis nas tabelas do usuário.

Na solução que utilizamos no framework, há duas alterações desse tipo que se mostraram inevitáveis: acrescentamos gatilhos e uma nova coluna em cada tabela mapeada pelo usuário. Essas alterações foram cruciais, tanto para a materialização, quanto para a identificação de alterações que podem necessitar de manutenção nas ligações materializadas. Porém, há um ponto negativo: assumimos que o framework terá permissão para criar colunas nas tabelas existentes na base de dados local do usuário, apesar de sabermos que isso nem sempre é verdade.

Para contornar esse tipo de problema, é necessário uma outra estratégia que mantenha a correspondência entre as URIs externas obtidas e os registros locais correspondentes, mas não altere a estrutura das tabelas do usuário.

Como estratégia alternativa, a solução atual poderia ser substituída por outra menos intrusiva: ao invés de criar colunas nas tabelas do usuário, seriam

criadas tabelas relacionamentos para cada tabela monitorada. Ou seja, quando um usuário criasse um mapeamento de um módulo de resolução de entidades com uma de suas tabelas, o framework se encarregaria de criar uma nova tabela relacionamento, que teria uma chave estrangeira para a tabela do usuário, e outra para uma tabela de controle da ferramenta.

Não optamos por essa solução por ser muito dependente da estrutura das tabelas mapeadas. Modificações nas colunas de chave primária dessas tabelas afetariam a estrutura de controle do framework, pois essas chaves seriam referenciadas pelas tabelas de relacionamento criadas para cada mapeamento. Logo, essa estratégia também possui um ponto negativo a ser solucionado.

### 4.1.4. Manutenção

Uma vez materializadas, é necessário manter as ligações coerentes e consistentes, mesmo após alterações tanto na base local quanto na base externa. Para isso, precisamos monitorar tais alterações.

Existem diferentes tipos de alterações e, para cada tipo, há uma estratégia específica de monitoramento e correção das ligações envolvidas. Na seção 4.4 serão listadas as estratégias de manutenção de ligações para cada atualização possível nas bases de dados envolvidas.

## 4.2 Requisitos e Casos de Uso

Antes de implementarmos a ferramenta, realizamos um levantamento dos requisitos necessários para a instalação, implementação e execução do framework.

Os requisitos funcionais são:

1. Ser instalado em qualquer base de dados;
2. Permitir o uso de módulos de resolução de entidades desenvolvidos pelos usuários;
3. Funcionar com módulos de resolução de entidades desenvolvidos em qualquer linguagem de programação;
4. Permitir a escolha de tabelas/colunas a serem monitoradas;
5. Buscar URI a partir do valor de qualquer tabela/coluna monitorada;

6. Identificar operações de remoção, inserção e atualização para qualquer tabela monitorada;
7. Buscar URI correspondente e materializá-la sempre que houver uma inserção de registro em uma tabela monitorada;
8. Buscar e atualizar URI sempre que houver uma atualização de registro em uma tabela/coluna monitorada;
9. Remover URI sempre que houver uma remoção de registro em uma tabela monitorada;
10. Atualizar URI materializada caso sofra alguma modificação em sua fonte de origem;
11. Manter a consistência entre os dados do framework e da base de dados em que foi instalado.

Os requisitos não funcionais são:

1. Funcionar apenas com o SGBD PostgreSQL;
2. Funcionar apenas no ambiente de desenvolvimento Linux;
3. Necessitar da instalação prévia do framework Java SE;
4. Procurar apresentar mensagens durante a execução do Monitor.

A Figura 3 apresenta o diagrama com os casos de uso de um administrador de uma base de dados, usuário do Framework MsA.

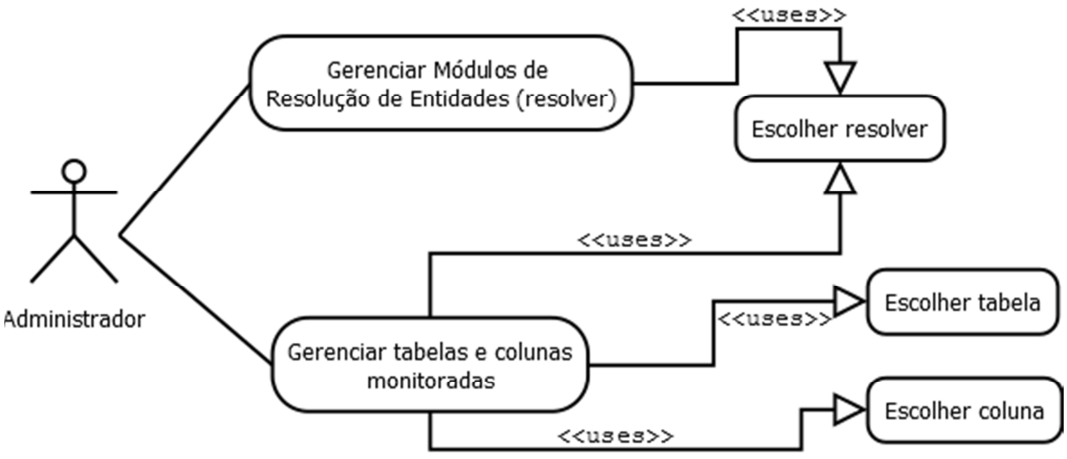


Figura 3: Diagrama de Casos de Uso do Framework MsA

## 4.3 Arquitetura do Framework MsA

### 4.3.1. Visão Geral da Arquitetura

A Figura 4 apresenta os componentes e o funcionamento do framework. Ele fica instalado na base de dados do usuário e, em segundo plano, monitora a ocorrência de modificações, tanto nos dados locais, quanto nos dados externos.

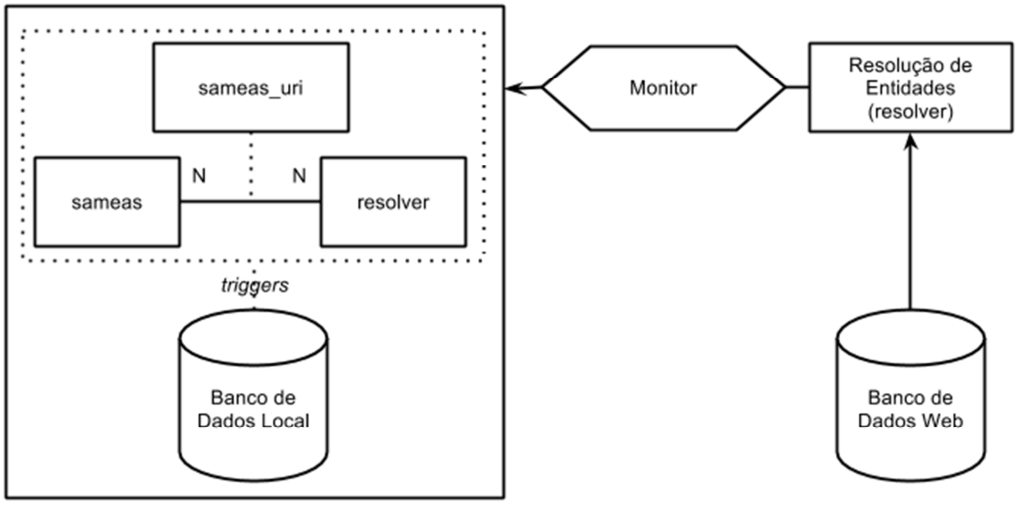


Figura 4: Visão geral da arquitetura do Framework MsA

Esse monitoramento é realizado pelo Monitor. Ele coordena os demais componentes para manter a consistência dos dados, mediando a comunicação entre dados locais e externos.

A comunicação com os dados externos é realizada através do componente Resolução de Entidades (resolvedor). Ele é responsável por obter URIs em fontes de dados disponíveis na Web. Pode ser composto por vários módulos, um para cada fonte de dados a ser consultada. Esses módulos deverão ser desenvolvidos, ou configurados, no caso de um módulo já existente, pelo usuário.

A escolha do resolvedor correto e o armazenamento das URIs externas identificadas quando o mesmo for executado são tarefas realizadas com o uso das tabelas específicas da arquitetura (*sameas*, *sameas_uri* e *resolver*). Elas contêm dados auxiliares para a execução do framework e armazenam as URIs obtidas na Web de Dados através dos módulos de resolução de entidades. Essas URIs podem ser publicadas juntamente com os dados do banco de dados local, em forma de ligações owl:sameAs. As seções seguintes descrevem os papéis de cada componente representado.

### 4.3.2. Resolução de Entidades

Chamamos de Resolução de Entidades (resolvedor) o conjunto de módulos responsáveis por buscar URIs na Web. Ou seja, deverão ser módulos, desenvolvidos pelo usuário, responsáveis por identificar entidades da base de dados externa que são semelhantes às entidades da base de dados interna.

Cada módulo deste componente deverá receber uma string a ser consultada na Web e retornar uma string com a URI encontrada (ou vazio quando a consulta não retornar nenhum resultado).

Eles podem ser desenvolvidos em qualquer linguagem de programação, ou até mesmo compostos por qualquer ferramenta de identificação de entidades semelhantes que o desenvolvedor julgar eficiente para a sua implementação. Por exemplo, para mapear dados internos com uma base de dados publicada em RDF, pode-se utilizar o Silk Framework descrito no capítulo 3.

### 4.3.3. Tabelas resolver, sameas e sameas_uri

As tabelas *resolver*, *sameas* e *sameas_uri* são específicas do modelo para controlar e armazenar os dados obtidos da Web. A Figura 5 apresenta a estrutura de cada uma dessas tabelas.

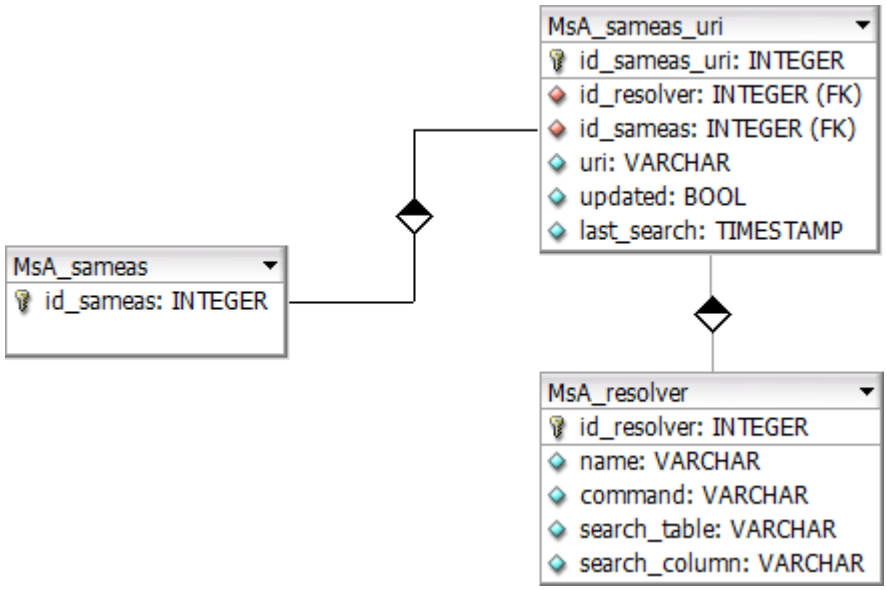


Figura 5: Modelagem de dados das tabelas de controle do Framework MsA

A seguir, descrevemos e especificamos cada uma das colunas representadas.

Tabela 1: Especificações técnicas das tabelas de controle

| Coluna | Tipo | Nulo | PK | FK |
|---|---|---|---|---|
| **Tabela: resolver** | | | | |
| id_resolver | serial | N | PK | - |
| name | varchar | N | UK | - |
| command | varchar | N | - | - |
| search_table | varchar | N | - | - |
| search_column | varchar | N | - | - |
| **Tabela: sameas** | | | | |
| id_sameas | serial | N | PK | - |
| **Tabela: sameas_uri** | | | | |
| id_sameas_uri | serial | N | PK | - |
| id_sameas | integer | N | - | sameas |
| id_resolver | integer | N | - | resolver |
| uri | varchar | - | - | - |
| updated | boolean | N | - | - |
| last_search | timestamp | - | - | - |

Tabela 2: Descrição do uso de cada coluna das tabelas de controle

| Coluna | Descrição |
|---|---|
| **Tabela: resolver** | |
| id_resolver | Identificador do resolvedor na base de dados. |
| name | Nome único do resolvedor. |
| command | Comando que dispara o resolvedor. |
| search_table | Tabela que será monitorada para obtenção de URI externa. |
| search_column | Coluna da qual o valor será utilizar para localizar uma URI. |
| **Tabela: sameas** | |
| id_sameas | Identificador gerado para cada registro monitorado. |
| **Tabela: sameas_uri** | |
| id_sameas_uri | Identificador da URI na base de dados. |
| id_sameas | Identificador de um registro monitorado na base local. |
| id_resolver | Identificador do resolvedor responsável por buscar uma URI. |
| uri | URI externa correspondente ao registro. |
| updated | Indicador do sucesso na obtenção da URI externa. |
| last_search | Data e hora da última consulta realizada na base externa. |

Para cada módulo de resolução de entidades desenvolvido, o usuário precisa mapeá-lo com a tabela e coluna que possui os valores locais que serão utilizados na busca por URIs externas. A tabela *resolver* armazena dados desses mapeamentos configurados pelo usuário. Uma tupla dessa tabela mapeia uma coluna qualquer (*search_column*), de uma tabela qualquer (*search_table*) da base de dados, com o comando a ser executado (*command*). O comando aciona o

módulo que obterá as URIs correspondentes aos registros do par tabela/coluna mapeado. Toda tabela que estiver mapeada na tabela *resolver* terá um gatilho associado à coluna mapeada, de forma a controlar as operações ocorridas nos registros, mantendo a consistência das ligações externas correspondentes.

Além disso, em cada tabela registrada como *search_table*, será criada uma coluna *id_sameas*. Essas colunas são chave estrangeira da tabela *sameas*, tabela que armazena os identificadores únicos dos registros do banco de dados local que terão ligações com fontes de dados externas. Essa tabela possui apenas uma coluna: *id_sameas* (chave primária), servindo apenas para permitir que um registro possa estar relacionado a mais de uma URI externa – por exemplo, obtidas por diferentes módulos de resolução de entidades.

Por fim, a tabela *sameas_uri* armazena as URIs obtidas da Web, sendo a tabela que deverá ser consultada para publicação das ligações owl:sameAs. Ela também funciona como uma fila para a materialização de ligações, aonde é realizado o controle de quais registros estão com suas ligações desatualizadas.

Caso um registro local ainda não tenha sido procurado na Web (*updated* possui o valor *false*), seu módulo de resolução de entidades correspondente é acionado para obtenção da URI externa correspondente. Ao obtê-la, as colunas *updated* e *last_search* são atualizadas com *true* e data/hora da obtenção, respectivamente, e a nova URI é armazenada na coluna *uri*. Porém, se a URI local não for encontrada na Web, apenas a coluna *last_search* é atualizada, e novas buscas serão realizadas periodicamente até obter algum resultado.



### 4.3.4. Triggers

Quando o usuário mapeia um módulo de resolução de entidades com a tabela e coluna da base de dados local correspondente, o framework, além de criar a coluna *id_sameas*, cria dois gatilhos para cobrirem as possíveis alterações a serem identificadas nesta tabela. São eles:

- **MsA_tg_before**: identifica operações do tipo *insert* ou *update*;
- **MsA_tg_after**: identifica operações do tipo *delete*.

Esses gatilhos, quando disparados, executam a função principal *MsA_tg_main*, responsável por realizar as manutenções necessárias para garantir a

consistência das ligações materializadas, de acordo com cada tipo de alteração identificado.

Ou seja, as tabelas monitoradas, ao receberem alguma operação em seus registros, acionarão o gatilho correspondente, que chamará a função principal. Essa função realizará as devidas manutenções de acordo com a operação identificada. Essas operações e suas manutenções correspondentes estão resumidas na listagem a seguir e serão detalhadas na seção 4.4.

- **Inserção**: Sinaliza que a URI externa correspondente ao novo registro precisa ser buscada. A URI será buscada utilizando o(s) resolvedor(es) correspondente(s) à tabela que acionou o gatilho, e caso seja encontrada, uma nova ligação será materializada;
- **Atualização**: Caso a atualização tenha ocorrido na coluna monitorada, sinaliza que a URI externa correspondente ao registro precisa ser atualizada. A URI antiga é desmaterializada e uma nova é buscada utilizando o(s) resolvedor(es) correspondente(s) à tabela que acionou o gatilho;
- **Remoção**: Remove a URI externa correspondente ao registro removido.

### 4.3.5. Monitor

Inicialmente, o framework envolveria apenas a utilização de gatilhos que desempenhariam dois papéis: monitoramento de tarefas e resolução de entidades. Porém, quando um gatilho realizasse uma busca externa por uma URI, não seria possível determinar quanto tempo duraria essa busca. Também não seria possível garantir que ela terminaria com sucesso. Ambas as situações poderiam fazer com que transações do usuário fossem perdidas. Por exemplo, poderiam impedir que o usuário realizasse operações simples em uma tabela, apenas porque o acesso para consulta dos dados externos na Web esteve por alguns momentos fora do ar. Logo, abandonamos essa estratégia.

Escolhemos a solução que separa a responsabilidade de obter a URI em uma estratégia externa ao SGBD. Com isso, separamos o framework em duas partes: uma estrutura que sinaliza quais registros precisariam ter suas URIs atualizadas (tabelas de controle e armazenamento); e uma implementação responsável por

verificar esses registros, buscar suas ligações na Web e materializá-las no banco de dados local.

A etapa de busca das ligações na Web é de responsabilidade dos módulos de resolução de entidade, e as demais são responsabilidades do Monitor. Ao ser inicializado, o Monitor permanece executando em segundo plano e realiza consultas periódicas na tabela *sameas_uri*, buscando por registros desatualizados, ou seja, que exigem uma nova busca externa por suas ligações owl:sameAs. Esses registros serão buscados através do resolvedor correspondente e atualizados com a nova URI encontrada. Caso nenhuma URI seja encontrada, serão realizadas buscas periódicas, até que a consulta retorne algum resultado.

Em resumo, o Monitor é responsável pela comunicação entre o banco de dados externo e o banco de dados local. Ele verifica quais registros locais estão com suas ligações owl:sameAs vazias ou desatualizadas e aciona o módulo de resolução de entidade (resolvedor) correspondente para realizar consultas na Web. Após obter a URI por meio do resolvedor, o Monitor a armazena na tabela *sameas_uri*.

A Figura 6 apresenta o diagrama de classes do Monitor. Quando inicializado, a classe Main instancia um objeto da classe GerenteTarefas e um objeto da classe FilaTarefas. São consideradas tarefas as entidades do banco de dados local que possuem um módulo de resolução de entidades associado, mas ainda não estão com a URI da entidade externa correspondente materializada.

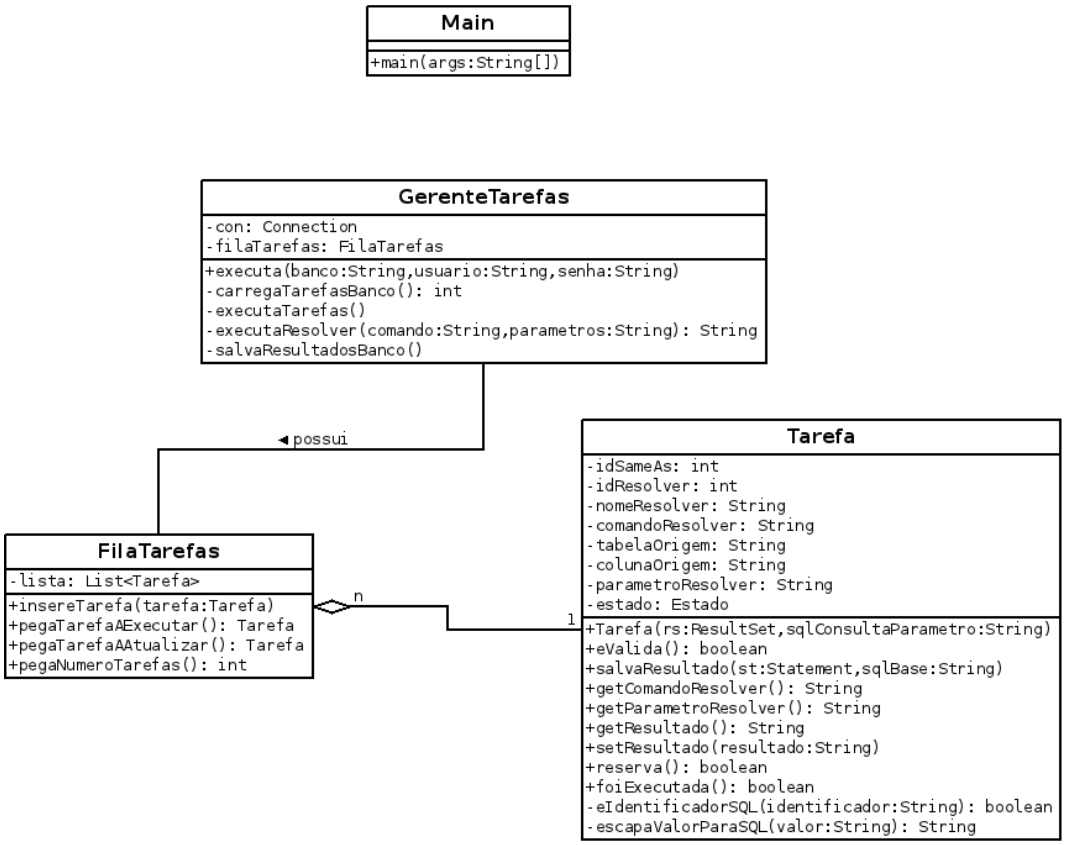


Figura 6: Diagrama de Classes do Monitor

O objeto GerenteTarefas instanciado consulta periodicamente a tabela *sameas_uri* e, ao identificar alguma entidade com consulta à Web pendente, instancia um objeto da classe Tarefa (para cada entidade pendente) com as propriedades correspondentes. Esse objeto é então inserido na fila de tarefas através do método insereTarefa do objeto FilaTarefas.

Após preencher a fila, o objeto GerenteTarefas executa o comando da propriedade comandoResolver de cada tarefa que constar na fila. O resultado será a URI encontrada na Web de Dados. Por fim, para materializar a URI encontrada como resultado, executa o método salvaResultado do objeto Tarefa. Esse é o método responsável por materializar a URI na base local.

## 4.4 Monitoramento das Bases de Dados

### 4.4.1. Alterações na Base Local (SGBD)

#### 4.4.1.1. Inserção: novo registro foi cadastrado

Novos registros locais (em tabelas mapeadas) deverão ter suas URIs externas correspondentes consultadas na Web. Ao encontrá-las, elas deverão ser materializadas com referências para o novo registro local inserido. Esse tipo de alteração não precisa de manutenção de dados locais, uma vez que se trata de um dado novo inserido. Logo, terá como consequência apenas a inserção da nova URI (materialização) e suas relações com a base local.

Para que isso ocorra, uma inserção na base local dispara a função (*stored procedure*) do gatilho principal do framework, que cria um identificador único para o registro inserido (*id_sameas*). Além disso, armazena o identificador na tabela *sameas* e na coluna *id_sameas* da tabela que sofreu a inserção.

A mesma função consulta a tabela *resolver* para verificar se a tabela que sofreu a inserção está registrada como *search_table* de algum dos módulos de resolução de entidades cadastrados. Caso esteja, obtém o identificador único do resolvedor correspondente (*id_resolver*).

Para cada resolvedor encontrado, a função insere, na tabela *sameas_uri*, o *id_resolver* e o *id_sameas* do novo registro. Por padrão, a coluna *id_sameas_uri* (identificador da URI na base local) será preenchida com um valor sequencial (referente ao último identificador inserido), e a coluna *updated* será preenchida com o valor *false*. As demais colunas permanecem com o valor nulo.

Em suas verificações periódicas na tabela *sameas_uri*, o Monitor busca por registros que estejam com a coluna *updated* com o valor *false*; e com a coluna *last_search* com o valor nulo (nenhuma consulta externa foi realizada), ou, supondo uma periodicidade de consulta definida para intervalos diários, com alguma data anterior à data atual (nenhuma consulta externa foi realizada hoje). Consideramos que registros que se encaixam nessas restrições estão aguardando uma nova busca de URI equivalente na Web de Dados. Em uma dessas verificações, o novo registro atenderá a essas restrições e será identificado pelo Monitor.

Ao identificar o registro, através do *id_resolver* associado (tabela *sameas_uri*, coluna *id_resolver*), o Monitor obtém o comando (tabela *resolver*, coluna *command*) que executa o módulo de resolução de entidades correspondente ao registro.

Ele obtém também o valor que consta na coluna mapeada (*search_column*) e o utiliza como parâmetro para executar o módulo.

O módulo é então executado e retorna uma *string* com o resultado obtido (ou o valor nulo, caso não haja nenhum resultado).

Se o módulo retornou alguma URI como resultado, o registro da tabela *sameas_uri*, que contém o *id_sameas* e o *id_resolver* utilizados, terá a coluna *uri* atualizada para a URI externa encontrada (materialização), e a coluna *updated* será finalmente atualizada para *true*.

Se o módulo não retornar nenhum resultado (a URI externa correspondente não foi encontrada), essas colunas não serão atualizadas.

Em ambos os resultados, a coluna *last_search* será atualizada para o valor do dia atual, no qual o resultado foi obtido (mesmo que o resultado seja nulo). Esta coluna é utilizada pelo framework pois, caso uma URI externa equivalente não tenha sido localizada no momento da inserção, será necessário buscá-la na Web repetidas vezes, até encontrá-la para a materialização. É preciso definir a periodicidade em que essas buscas deverão ser realizadas, de forma a evitar requisições exaustivas em fontes de dados externas. Essa periodicidade deve variar de acordo com o tipo de parceria existente entre os responsáveis pela base interna e os responsáveis pela fonte externa consultada e pode ser definida por um parâmetro.

Sendo assim, caso não tenha sido obtido nenhum resultado na última execução do módulo, e supondo uma periodicidade de consulta diária, o Monitor, em uma de suas verificações na tabela *sameas_uri*, identificará que o registro ainda está com a coluna *updated* com o valor *false* e que a última execução foi realizada há um dia. Com isso, executará novamente o módulo de resolução de entidades, repetindo diariamente esse procedimento, até que algum valor seja retornado.

#### 4.4.1.2. Atualização: registro antigo foi modificado

Uma modificação em um registro local que já possua uma ligação materializada correspondente pode tornar as duas entidades da ligação não mais equivalentes. Isso pode ocorrer quando o valor da coluna modificada tiver sido utilizado para localizar a URI equivalente na Web de Dados. Nesse caso, será necessário "desmaterializar" a ligação antiga, e materializar uma nova. Para isso, realizamos uma nova consulta externa a fim de encontrar a entidade correspondente ao novo valor. Se não for encontrada em um primeiro momento, assim como na inserção, serão necessárias novas consultas com uma periodicidade especificada por parâmetro.

Para esse comportamento funcionar no framework, uma atualização em uma tabela que está mapeada com um módulo de resolução de entidades dispara a função do trigger principal, que obtém, na coluna *id_sameas* da tabela, o identificador único do registro atualizado.

Caso ainda não exista um identificador (pode ocorrer, por exemplo, quando o registro já existia na base antes da instalação do framework), a função cria um novo, e o armazena na tabela *sameas* e na coluna *id_sameas* da tabela que sofreu a atualização.

A mesma função consulta a tabela *resolver* para verificar se a tabela que sofreu a atualização está registrada como *search_table* de algum dos módulos de resolução de entidades cadastrados. Caso esteja, obtém o identificador único do resolvedor correspondente (*id_resolver*), e o valor da coluna *search_column*.

Antes de prosseguir com os demais passos, a função verifica se a diferença entre o registro atual e o registro atualizado está no valor da coluna mapeada em *search_column*.

Caso não esteja, finaliza a execução, pois significa que a alteração identificada não afeta alguma possível ligação já existente com uma URI externa, pois o valor utilizado para obtê-la não foi alterado.

Caso contrário, atualiza para *false* a coluna *updated* da tupla da tabela *sameas_uri* correspondente ao registro atualizado, e atualiza para nulo o valor da coluna *last_search*, forçando uma nova execução do respectivo módulo de resolução de entidades.

O Monitor verifica a tabela *sameas_uri* e identifica o registro que precisa ter sua URI atualizada. A partir daqui, os demais passos são idênticos aos da inserção.

#### 4.4.1.3. Remoção: registro antigo foi apagado

Ao remover um registro local, também deverá ser removida sua ligação materializada. Como mantemos a referência entre os registros locais e suas URIs externas correspondentes, esse tipo de manutenção é realizado em cascata.

Ou seja, quando o usuário remove um registro de uma tabela que está mapeada com um módulo de resolução de entidades, a remoção dispara a função do gatilho principal do framework, que obtém, na coluna *id_sameas* da tabela, o identificador único do registro removido. Após obter o identificador, o remove da tabela *sameas* do framework. Como a chave estrangeira da coluna *id_sameas* da tabela *sameas_uri* (que faz referência à tabela *sameas*) possui a opção “on delete cascade”, todas as URIs relacionadas ao *id_sameas* removido serão automaticamente apagadas da tabela *sameas_uri*. Com isso, todas as ligações owl:sameAs do registro removido também deixarão de existir.

### 4.4.2. Alterações na Base Externa (Web de Dados)

#### 4.4.2.1. Inserção: nova URI foi publicada

Uma nova URI publicada na Web de Dados pode ser equivalente a alguma entidade já existente na base local. Porém, como ainda não havia sido publicada, a ligação owl:sameAs entre as duas também não havia sido identificada para a materialização.

Esse problema também depende da realização de consultas periódicas por entidades externas que sejam equivalentes às internas que ainda não possuem ligação materializada. Como as consultas disparadas pela ocorrência de um novo registro local só deixarão de ocorrer quando, finalmente, for encontrada uma entidade equivalente, alguma delas identificará a nova entidade publicada e materializará a nova ligação.

Sendo assim, a materialização das ligações com esse novo registro é abrangida pelas estratégias de manutenção para inserção e atualização descritas

nas seções anteriores. Ou seja, como o Monitor verifica quais registros da base de dados local ainda não obtiveram resultado ao consultar uma URI externa equivalente, ele continuará executando os módulos correspondentes aos registros que se encontram nessa situação.

Com isso, após a nova URI constar na Web, o módulo de resolução de entidades, executado periodicamente pelo Monitor, a encontra e atualiza o registro local correspondente, materializando uma nova ligação.

#### 4.4.2.2. Atualização: registro da URI já identificada foi modificado

Caso a base de dados externa, por algum motivo, modifique alguma propriedade (ou até a própria URI) de uma entidade já publicada, consideramos que houve uma "atualização" de algum registro externo. Com isso, as ligações materializadas que faziam referência à URI dessa entidade ficarão desatualizadas. Para atualizar essas ligações, precisamos realizar novas consultas externas pelos registros que já estão com suas URIs materializadas.

Caso as entidades externas não tenham sofrido nenhuma modificação, essas novas consultas serão redundantes, pois as novas URIs serão idênticas às anteriores. Essa redundância é necessária para garantir que as entidades das ligações permanecem sendo equivalentes.

Aproveitando as soluções anteriores, para essas novas consultas serem realizadas, o Monitor marca como não encontradas, periodicamente, as URIs já materializadas como um novo registro a ser consultado (updated com valor false na tabela sameas_uri). Essa periodicidade pode ser definida por parâmetro (nos experimentos, definimos para um intervalo de trinta dias).

Feito isso, os registros já materializados ficam pendentes e o Monitor executa novamente seus módulos de resolução de entidades correspondentes. Serão realizadas novas consultas por suas URIs externas equivalentes. Caso encontre uma nova URI, substitui a anterior pela encontrada, atualizando a materialização da ligação. Caso não encontre, atualiza a anterior para o valor nulo (deixou de existir), mantém a coluna *updated* (correspondente à URI) com o valor *false*, e o Monitor, em uma de suas verificações periódicas, aciona novamente o módulo de resolução de entidades para aquele registro local. Como nos demais casos, o

módulo será acionado periodicamente até que seja encontrada a nova URI equivalente.

#### 4.4.2.3. Remoção: URI antiga deixou de ser publicada

Consideramos como “remoção” quando uma base de dados externa deixa de publicar algum dos seus registros anteriormente existente na Web de Dados. Do ponto de vista do framework, esse caso é idêntico ao anterior. A única diferença é que, na atualização, uma URI deixou de existir, e uma nova foi criada com o mesmo registro ao qual ela correspondia. No caso da remoção, ela apenas deixou de existir, e o registro não estará mais na Web de Dados.

Com isso, os procedimentos de materialização e manutenção serão equivalentes aos de atualização, pois o Monitor continuará procurando por uma URI substituta, mesmo que ela nunca venha a existir.



## 4.5 Instalação e Inicialização

### 4.5.1. Infraestrutura

O Framework MsA foi desenvolvido a partir das linguagens Shell Script, Java e SQL: desenvolvemos os scripts de instalação em Shell Script, a lógica de programação do Monitor em Java, e os scripts para a instalação do framework na base de dados do usuário em SQL.

Para realizar a busca de URIs, é necessário que o usuário desenvolva pelo menos um módulo de resolução de entidades. Esse módulo pode ficar instalado no diretório que for de sua preferência.

Para utilizá-lo, após a instalação, o usuário precisa especificar o par tabela/coluna a ser monitorado pelo framework, e o comando que executará o módulo responsável por buscar URIs externas equivalentes aos registros dessa coluna.

A comunicação entre a fonte de dados externa e a base de dados local é realizada após a inicialização do Monitor do framework através de um script que o aciona e o mantém executando em segundo plano.

### 4.5.2. Instalação do Framework

Como primeiro passo para a instalação, é necessário indicar o nome da base de dados na qual o framework irá executar, além do nome do usuário que será utilizado para acessar essa base. Esses dados serão utilizados para a criação das estruturas do framework no banco de dados, e também para monitoramento de suas alterações.

As estruturas são criadas a partir de um script específico que contém os comandos SQL necessários. Ou seja, o usuário configura os parâmetros de acesso à sua base e executa tal script. O script solicita a senha de acesso ao SGBD (se houver) e executa os comandos de criação na base local configurada.

Feito isso, a base do usuário passa a conter as tabelas do framework (*sameas*, *sameas_uri* e *resolver*) e a função principal (*MsA_tg_main*) que realiza alterações nos dados de controle de acordo com cada operação monitorada.

Com a estrutura principal do framework instalada, o usuário já pode mapear os módulos de resolução de entidades nas tabelas e colunas correspondentes. Um outro script é responsável pela criação desses mapeamentos. O usuário informa o nome único a ser dado ao módulo; a tabela e a coluna (respectivamente), de onde serão consultados os valores a serem buscados na Web de Dados; e o comando que será utilizado para executar o seu módulo.

Esse script de mapeamento armazena os dados do módulo na tabela resolver (nome, comando, tabela, coluna) e cria um identificador interno único para referenciá-lo. Além disso, na tabela informada pelo usuário, cria uma coluna *id_sameas* que conterá, para cada registro, um identificador interno único. Essa coluna é criada como chave estrangeira da tabela *sameas*, aonde são administrados esses identificadores. Esse funcionamento permite que um mesmo registro possua mais de uma URI equivalente. Isso pode ser útil, por exemplo, para mapear dois módulos de resolução de entidades diferentes com uma mesma tabela.

Ainda nessa tabela informada pelo usuário, cria também os dois gatilhos do framework: *MsA_tg_after* e *MsA_tg_before*. Eles utilizarão a função principal *MsA_tg_main* criada na instalação.

Para finalizar o mapeamento, o identificador único (*id_sameas*) de cada registro já existente na tabela do usuário e o identificador do *resolvedor* mapeado

(*id_resolver*) são inseridos na tabela *sameas_uri*. As colunas *uri* e *last_search* terão o valor nulo. Já a coluna *updated* terá o valor *false* (default), indicando que a URI precisa ser atualizada. Todas essas colunas serão preenchidas futuramente pelo Monitor.

### 4.5.3. Inicialização do Monitor

Após criado pelo menos um mapeamento, o usuário só precisa inicializar o Monitor para que o framework passe a materializar e manter ligações owl:sameAs. Para inicializá-lo, há um script que executa a classe principal do framework em segundo plano. Após sua execução, o Monitor permanecerá executando na máquina do usuário, monitorando possíveis alterações periodicamente.

A periodicidade desse monitoramento da base interna pode ser definida por parâmetro. Por padrão, esse parâmetro está definido para um intervalo de vinte segundos. Caso identifique alguma alteração, esse intervalo diminui temporariamente para três segundos, até que nenhuma outra alteração seja identificada. Essa diminuição acelera a execução quando houver muitas alterações seguidas na base local.

Diferente da periodicidade anterior, definimos o padrão da periodicidade de monitoramento da base externa para um intervalo de trinta dias (também pode ser alterado por parâmetro). O intervalo é bem maior, pois essa tarefa consiste em realizar muitas requisições em um servidor externo. Porém, esse intervalo deverá variar de acordo com a parceria e com a frequência de publicação da base externa consultada.

Após os procedimentos anteriores, com suas devidas periodicidades, o usuário possui o framework executando na sua máquina e instalado na sua base de dados. Com isso, alterações realizadas na coluna e tabela mapeada refletirão em execuções do módulo de resolução de entidades correspondente.

## 4.6 Comentários sobre as Etapas de Desenvolvimento do Framework

### 4.6.1. Tecnologias

Inicialmente, estudamos as tecnologias que poderiam ser utilizadas na construção do framework. Foram levadas em consideração as tecnologias utilizadas atualmente pelo estudo de caso realizado (mais detalhes no próximo capítulo). Logo, escolhemos o PostgreSQL como o SGBD para a instalação do framework.

Como ambiente de desenvolvimento, escolhemos o sistema operacional Linux por suas facilidades de administração. Por fim, a linguagem Java foi uma escolha de projeto para permitir a portabilidade do código.

### 4.6.2. Especificação de Requisitos

Após a escolha das tecnologias, iniciamos a especificação dos requisitos necessários para o desenvolvimento do framework. A especificação foi voltada para o desenvolvimento de um modelo genérico, que permitisse a instalação em bases de dados heterogêneas.

### 4.6.3. Planejamento da Estratégia

Uma vez especificados os requisitos, iniciamos o planejamento da estratégia de desenvolvimento. Inicialmente, o planejamento envolvia apenas a utilização de gatilhos com a linguagem Java, que teriam dois papéis: monitor de tarefas e módulo de resolução de entidades. Porém, nessa etapa, foi possível notar que essa estratégia não seria viável. O problema ocorre, por exemplo, quando um gatilho não obtém sucesso ao acessar a Web em busca de uma URI. Nesse momento, ele pode afetar a finalização de transações do usuário na base local, que estarão aguardando o seu término.

Também planejamos o desenvolvimento de um monitor que seria disparado pela função do gatilho apenas quando necessário. Mas o problema anterior continuaria acontecendo. E, mesmo que a função do gatilho finalizasse sem aguardar a resposta do monitor (o monitor faria a materialização), cada processo externo criado pelo SGBD se tornaria “zumbi”. Isso ocorre porque o

SGBD não auxilia esse tipo específico de operação, fazendo com que os processos disparados por um gatilho não tenham seus recursos devidamente liberados ao finalizar.

Sendo assim, optamos por separar a responsabilidade de obter a URI em uma estratégia externa ao SGBD. Separamos o desenvolvimento em duas partes: desenvolver uma estrutura que sinalizasse quais registros precisariam ter suas URIs atualizadas; e desenvolver um monitor responsável por obter essa informação, buscar e guardar as URIs encontradas. Esse monitor permanece executando em segundo plano, em busca de novas tarefas.

### 4.6.4. Modelagem de Dados

Com as estratégias determinadas, iniciamos a criação do modelo de dados de controle do framework. Foi desenvolvido um modelo genérico o suficiente para permitir que qualquer tabela pudesse ter uma chave estrangeira para ela, garantindo o uso das facilidades da linguagem SQL, como por exemplo, gatilhos e operações em cascata.

### 4.6.5. Desenvolvimento da Arquitetura

Para desenvolver o Monitor, foi necessário antes desenvolver uma estrutura de controle que pudesse administrar a informação de quais registros precisavam ter suas URIs atualizadas, seja por motivo de inserção ou de atualização na base de dados. Nessa etapa, desenvolvemos a função principal (stored procedure) responsável por monitorar qualquer tipo de operação e administrar os dados das tabelas de controle do framework. Também desenvolvemos o script que cria gatilhos nas tabelas monitoradas, fazendo uso dessa função.

### 4.6.6. Implementação do Monitor

Esta etapa foi crucial para o funcionamento do framework. Criamos o diagrama de classes e implementamos as classes responsáveis pela comunicação entre a base de dados e os módulos de resolução de entidades. Essas classes fazem consultas e gravam registros na base de dados do usuário.

Também desenvolvemos o script de inicialização do framework. Ao inicializa-lo, ele permanece executando em segundo plano, fazendo verificações periódicas e constantes na base de dados.

### 4.6.7. Testes

Para a realização dos testes, criamos um módulo de resolução de entidades para o estudo de caso. O módulo será melhor detalhado no próximo capítulo.

Os scripts de teste foram criados utilizando esse módulo. Os casos de teste envolvem configuração do framework e operações diversas na base de dados local, verificando assertivas de entrada e saída para cada operação.

Os testes foram satisfatórios, cobrindo todos os casos possíveis para uso do framework.

### 4.6.8. Documentação

Com tudo pronto, iniciamos o detalhamento do passo-a-passo para o usuário utilizar o framework em sua base de dados.

O passo-a-passo com todos os procedimentos de instalação e execução necessários para o usuário estão registrados no arquivo readme.txt, disponibilizado juntamente com o código do projeto.