Integração de triplestore

Concepção/Especificação

04/10/10
1.0

Henrique Nunes

Histórico de versões do Relatório de Concepção/Especificação

Versão	Datas	Responsabilidades	Justificação
1.0	04/10/2010	Edição:Henrique Nunes Revisão: Aprovação	Versão inicial
		Edição: Revisão: Aprovação	

Índice

1. Introdução

Sistema a que se refere este relatório	Versão	Documento de Requisitos associado
Oobian – Integração de triplestore	1.0	

1.1. Objectivos

O objectivo principal é a integração/implementação de uma triplestore no projecto Oobian, obedecendo esta aos requisitos enumerados no documento Levantamento de triplestores.

1.2. Documentos relacionados

Outros documentos de entrada considerados para a actividade de Concepção	Breve descrição
Levantamento de triplestores	Descreve as opções ao nível de triplestores disponíveis no mercado à data, assim como os requisitos necessários a obedecer pela opção seleccionada

Outros documentos resultantes das actividades de concepção que completam este relatório	Breve descrição
Diagrama de classes	Descreve as classes criadas/refactorizadas e o seu relacionamento entre si e entre as classes já existentes.
Diagrama de sequência	Descreve a sequência de chamadas e operações relevantes na integração da triplestore.
Tabela de resultados preliminares	Agrega os resultados de medições de tempos para diversas ontologias usando a solução descrita.

2. Resumo da Actividade de Concepção

Com o objectivo de integrar uma triplestore no projecto Oobian, e após a escolha de uma solução entre as alternativas disponíveis no mercado, como descrito no documento Levantamento de triplestores, pretende-se a implementação e integração da framework Jena e correspondente triplestore TDB no Oobian.

Esta framework deve substituir a solução corrente, baseada na API do Protége, e possibilitar a manipulação de ontologias de tamanho elevado, sem despoletar erros de falta de memória. De facto, esta solução deverá permitir manipular ontologias de qualquer tamanho, independentemente da quantidade de memória disponível, mas com o inconveniente de maior frequência de acessos ao disco.

Para compensar este efeito, deve ser implementada uma cache que minimize os efeitos negativos na performance, decorrentes da maior frequência de acessos a disco.

3. Requisitos

3.1. Requisitos Funcionais Considerados

Os requisitos funcionais exigidos a esta solução são:

- Carregamento transparente das ontologias para a triplestore a partir dos respectivos ficheiros em repositório.
- Conversão total entre o modelo usado pela framework e o modelo da lógica de negócio do Oobian e vice-versa.
- Permitir o caching dos objectos convertidos para o modelo Oobian.
- Cumprir os requisitos anteriores sem penalizações graves na performance do Oobian.

3.2. Requisitos Funcionais Não Considerados

3.3. Requisitos Não Funcionais

4. Arquitectura do Sistema

4.1. Fundamentação e Abordagem, Restrições e Condicionantes

A solução considerada pode ser dividida em três componentes essenciais: triplestore, cache e tratamento do modelo de dados.

No que respeita à triplestore, convém referir que consiste em dois componentes semi-independentes:

- TDB – a triplestore propriamente dita. Para além do sistema de ficheiros, consiste num módulo que contém o código e a lógica para manipular datasets, grafos e modelos de ontologias.
- API Jena – a framework Jena é a API usada para comunicar com o TDB, fornecendo um modelo de dados bastante extenso, que descreve uma ontologia e seus componentes sob vários aspectos e permite fazer queries simples à triplestore.

Para a fundamentação da escolha desta solução particular ver documento Levantamento de triplestores

Para o sistema de cache, optou-se pelo Apache Java Caching System, uma vez que é um sistema bastante avançado mas simultaneamente simples. É altamente configurável, permitindo, caso se queira, configurar várias caches alternativas em diversas fontes, e necessita de poucas dependências.

Em relação ao tratamento do modelo de dados, a abordagem consistiu no refactor da classe que desempenhava este papel anteriormente em relação à API do Protege, uma vez que para além disto, também implementava o interface de web services que o cliente usa para comunicar com o servidor, bem como diversos métodos de apoio as estas funções.
Assim, toda a lógica nesta classe foi reorganizada da seguinte maneira:

- Uma classe específica de implementação do interface de web services – `IOWLAPIModel`
- Uma classe específica de conversão entre o modelo Jena e o modelo Oobian – `ModelBridge`
- Uma classe que implementa vários métodos de apoio às classes supra-mencionadas – `OntologyUtils`

Esta reorganização impunha-se, na medida em que toda esta lógica estava concentrada numa única classe, dificultando tanto a compreensão como a manutenção do código, para além de ser uma singularidade bizarra tendo em conta a organização modular do projecto.

5. Detalhes de Concepção

A integração da triplestore, implementação da cache e reorganização geral dos componentes envolvidos é ilustrada pelo seguinte diagrama de classes:

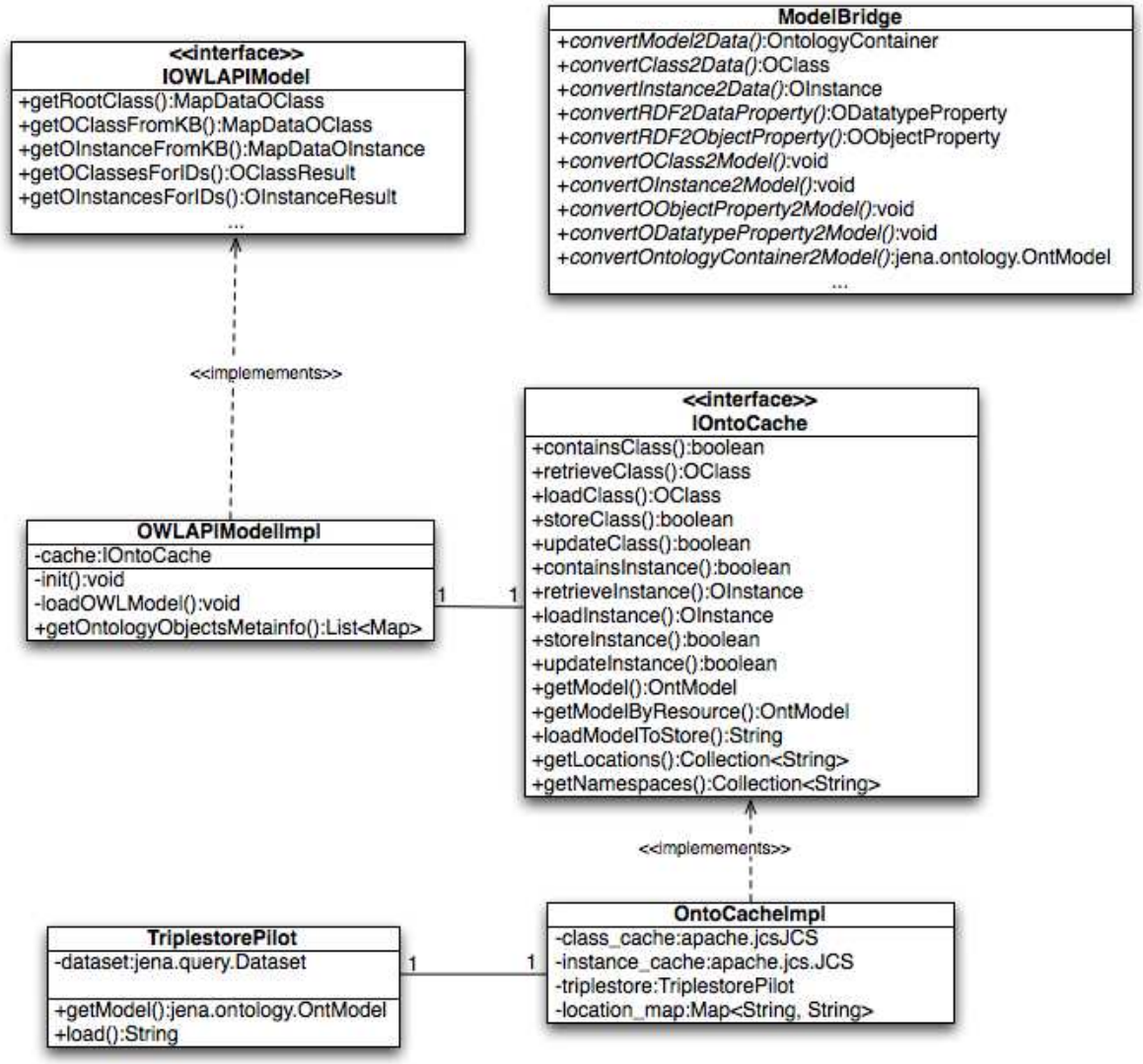


Figura 1 - Diagrama de classes

Assim, temos em primeiro lugar a classe `OWLAPIModelImpl` que implementa o interface `IOWLAPIModel`. Este interface é usado pelos web services, e especifica os métodos que os clientes usam na comunicação com o servidor. Para além de implementar este interface, a classe `OWLAPIModelImpl` implementa dois métodos relevantes:

- `loadOWLModel()` – este método destina-se a carregar os modelos das ontologias para a triplestore. A maioria da lógica de carregamento está abstraída nas classes `IOntoCache` e TriplestorePilot.
- `getOntologyObjectsMetainfo()` – método que se destina a extrair informação dos modelos contidos na triplestore, com vista à indexação desses conteúdos.

Como referido acima, grande parte da lógica de carregamento de ontologias está contida na classe `OntoCacheImpl` que implementa o interface `IOntoCache`. Este interface define os métodos que permitem comunicar com a cache. Em suma, a cache permite obter, carregar, actualizar e remover instâncias das classes `OClass` e `OInstance`. Para além disto, permite obter o modelo da

ontologia que disponibiliza a API da framework Jena através do URI de um resource e fornece um método que permite carregar uma ontologia contida num ficheiro para a triplestore.

A classe `OntoCacheImpl`, para além de implementar o interface acima descrito, possui os seguintes atributos relevantes:

- `class_cache / instance_cache` – estes atributos são as instâncias das caches de objectos `OClass` e `OInstance` respectivamente. Estes objectos são obtidos através de uma chamada a biblioteca JCS que, de resto, abstrai toda a lógica de caching.
- `triplestore` – este atributo é uma instância da classe `TriplestorePilot` que abstrai a lógica de criação de datasets e modelos de ontologias e, consequentemente, de acesso à triplestore.
- `location_map` – devido ao facto de não ser conhecido o namespace de cada ontologia aquando do seu carregamento para a triplestore, é necessário fazer um mapeamento entre o namespace da ontologia carregada e o seu nome interno na triplestore (concretamente, este nome é a path do ficheiro no repositório).

A classe `TriplestorePilot` contém a lógica de carregamento e acesso aos modelos ali contidos. Este acesso é feito através dos seguintes métodos:

- `getModel()` – este método permite obter um objecto que representa um modelo de uma ontologia – um graph ao nível da triplestore. Este modelo é útil porque é através dele que se acede à API Jena.
- `load()` – carrega uma ontologia em ficheiro para a triplestore. Neste método está contida a lógica que permite aceder à triplestore – através das classes disponibilizadas pelo componente TDB: `TDBFactory` e `TDBLoader` – e carregar a ontologia. O processo pode ser descrito sucintamente:
  - Criar um `Dataset` através da classe `TDBFactory`.
  - Pedir um named model à triplestore através do dataset criado. Se não existir um modelo com o nome fornecido, é devolvido um novo.
  - Carregar a ontologia para o modelo através da classe `TDBLoader`.

O único atributo relevante desta classe é uma instância da classe `Dataset`, que é o ponto de comunicação com a informação contida na triplestore. Na prática, é usada para obter tanto modelos novos como modelo preenchidos.

Por fim, a classe `ModelBridge` contém toda a lógica que permite converter o modelo de dados usado pela API do Jena no modelo usado pelo Oobian e vice-versa. Este processo é conseguido sobretudo através dos métodos especificados no diagrama para esta classe, mas também através de outros contidos na classe `OntologyUtils` que não sofreu alterações de grande relevo, daí não se encontrar no diagrama. O único aspecto relevante da classe `ModelBridge` é o facto de todos os métodos serem static, dispensando-se assim a necessidade de haver instanciação desta classe, uma vez que não traria benefícios aparentes, antes pelo contrário complicando a organização do código, já que é usada por várias classes diferente, em hierarquias diferentes. É por esta razão que não existem ligações envolvendo esta classe no diagrama.

Para melhor detalhar a sequência de chamadas aos componentes envolvidos e melhor ilustrar o funcionamento da solução, consideremos o seguinte diagrama de sequência:

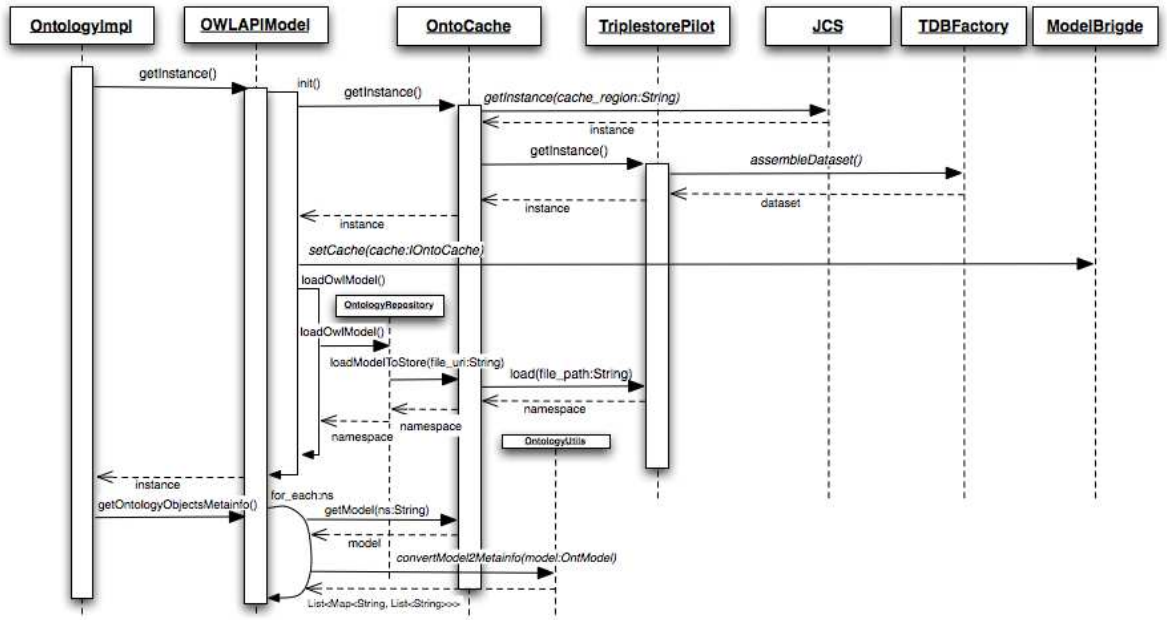


Figura 2 - Diagrama de sequência

Segue-se agora uma descrição da sequência ilustrada no diagrama:

A classe `OWLAPIModelImpl` é instanciada pela classe `OntologyImpl`. Durante a instanciação é executado o método `init()`, que contém o grosso da lógica de inicialização. Neste método é instanciada a classe `OntoCacheImpl` que, por sua vez, inicializa as caches e instancia a classe `TriplestorePilot`. Durante a instanciação da classe `TriplestorePilot`, esta instancia o atributo dataset através do método `assembleDataset()` que devolve o dataset contido na triplestore.

Após obter a instância de uma `IOntoCache`, o método `init()` prossegue passando essa instância à classe `ModelBridge` que a usa extensivamente durante o processo de conversão.
Em seguida o método `init()` chama o método `loadOntModel()`. Este método chama o método homónimo da interface `OntologyRepository` (no presente caso, esta interface é implementada pela classe `FileOntologyRepository`) que, por sua vez, chama o método `loadModelToStore()` na instância da cache para cada ficheiro a ser carregado. A cache chama o método `load()` na classe `TriplestorePilot`, que carrega a ontologia presente no ficheiro cujo path lhe fôr passado. Depois de carregada a ontologia e determinado o namespace, este é propagado de volta à classe `OWLAPIModel`, terminando os métodos `loadOwlModel()` e `init()` respectivamente.

Após obter uma instância da classe `OWLAPIModelImpl`, a classe `OntologyImpl` invoca o método `getOntologyObjectsMetainfo()` daquela. Para cada namespace – logo, para cada ontologia – este método invoca o método `getModel()` da cache para obter o modelo correspondente, passando-o de seguida ao método `convertModel2Metainfo()` da classe `OntologyUtils`. Nesta classe (e recorrendo aos métodos da classe `ModelBridge`) o modelo passado é convertido no modelo de dados do Oobian e é extraída a informação relevante para posterior indexação.

A partir deste ponto, a execução decorre sem modificações relativamente ao estado anterior à implementação desta solução.

6. Conclusões preliminares

Após a implementação descrita acima, verificaram-se os resultados preliminares contidos na seguinte tabela:

	Demo_Full	GK	Prima (1Mb)	Stepahead (2mb)	Primavera (9mb)
AVG MODEL	1	3	3	0	0
TOTAL MODEL	501	4118	6159	2006	5524
EXT AVG MODEL	1	1	0	1	0
EXT TOTAL MODEL	1	1	0	1	0
AVG CLASS	54	51	104	36	67
TOTAL CLASS	1098	7417	7310	257	4880
EXT AVG CLASS	60	52	104	36	67
EXT TOTAL CLASS	1213	7541	7310	257	4880
AVG INSTANCE	5	13	13	0	0
TOTAL INSTANCE	408	4118	6159	2006	5524
EXT AVG INSTANCE	0	1	0	0	0
EXT TOTAL INSTANCE	1	1	0	1	0
TOTAL	4477	12353	25028	1613865	572276

Tabela 1- Resultados preliminares para várias ontologias. Valores em milisegundos.

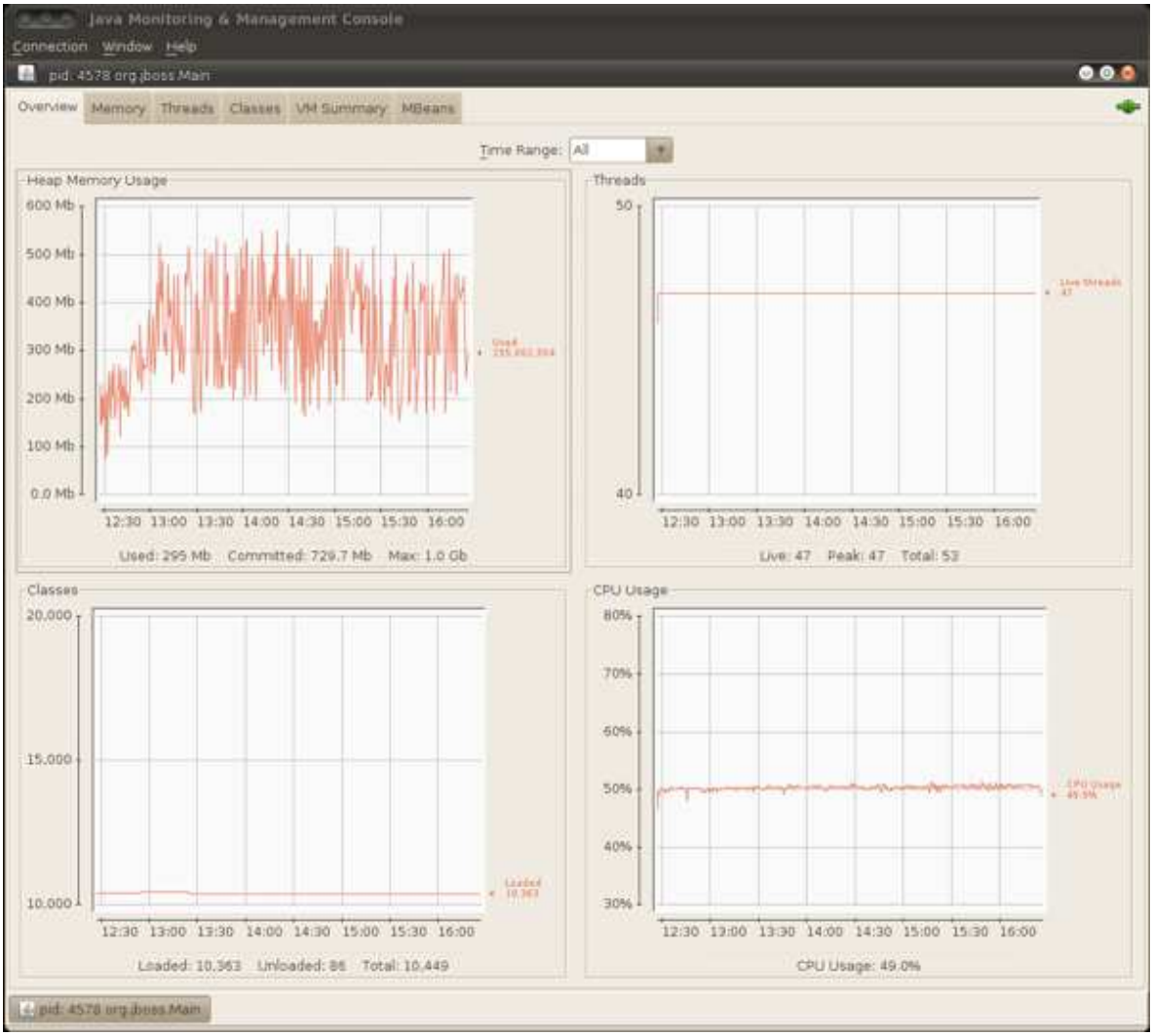
Esta tabela deve interpretada da seguinte maneira:

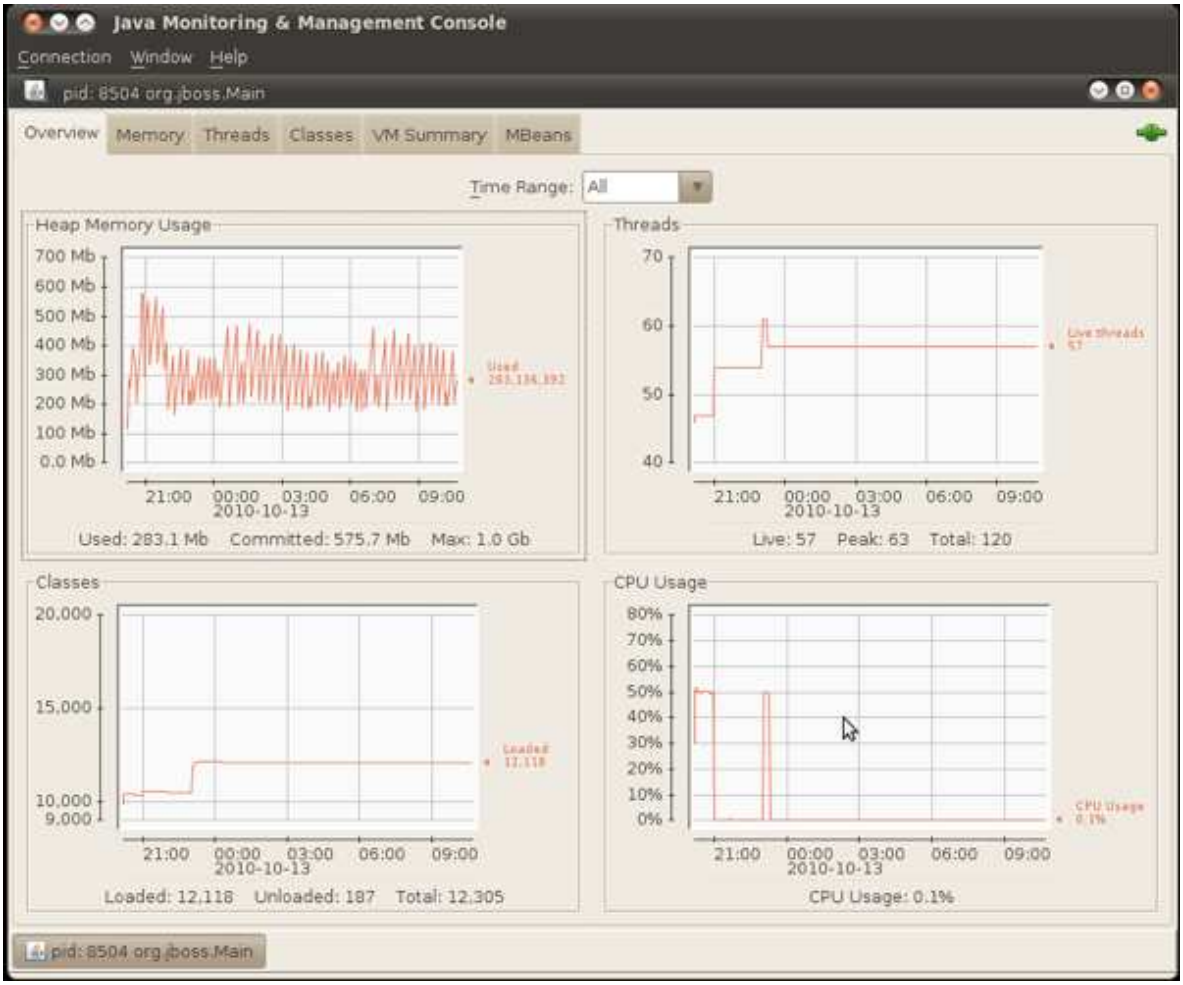
- Na coluna da esquerda encontram-se os vários parâmetros medidos, organizados por objecto:
  - AVG MODEL - tempo médio da duração de uma operação no modelo Jena durante a conversão.
  - TOTAL MODEL – tempo total gasto em invocações do modelo Jena durante a conversão.
  - AVG CLASS/INSTANCE – tempo médio da duração da conversão de uma classe/instância durante a conversão para o modelo Oobian.
  - TOTAL CLASS/INSTANCE – tempo total gasto na conversão de todas as classes/instâncias de uma ontologia
  - EXT * - mesmas métricas descritas acima mas relativas ao processo de extracção de meta-informação.
  - TOTAL – tempo total gasto na conversão e extracção de meta-informação para a ontologia indicada (não inclui os tempos de carregamento da ontologia para a triplestore).

Considerando os valores presentes na tabela, verifica-se uma inconsistência relativamente à ontologia Stepahead. De facto, esta ontologia demora muito mais tempo do que seria desejável, como se pode ver pelo tempo total (~26 minutos). Este valor elevado deve-se, muito provavelmente, ao número elevado de propriedades inversas presente na ontologia, que tem um impacto negativo na performance.

Também se testaram ontologias com um número de triplos na ordem dos milhões, de modo a avaliar os limites do sistema com datasets de proporções mais aproximadas ao esperado em ambiente real.

Seguem-se algumas imagens ilustrando o comportamento do sistema enquanto carrega e processa as ontologias.





Como se pode ver pelas imagens, o sistema escala bem a nível de memória, mantendo a quantidade de memória alocada estável, independentemente do número de triplos. A nível de CPU, as observações são semelhantes.

O principal ponto fraco é a performance ao nível do tempo. O sistema demora várias horas a fazer a conversão e extracção inicial quando lida com grandezas na ordem dos milhões de triplos. É de esperar que a performance sofra uma deterioração linearmente proporcional ao aumento do número de triplos.

Aspectos a melhorar:
- triplestore em ambiente 64-bit
- melhorar código de conversão e extracção
- estratégia de caching durante a conversão mais eficiente -> David propôs fazer caching permanente em memória da estrutura da ontologia i.e. classes e propriedades, e aceder à triplestore apenas para obter informação sobre as instâncias.
- para o carregamento inicial das ontologias, o formato utilizado deveria ser NTriples, uma vez que RDF/OWL implica o parsing do XML, acrescentando tempo ao processo. No entanto, isto implica converter o RDF/OWL para NTriples.

Riscos e Vulnerabilidades:
- impacto negativo na performance com ontologias contendo números elevados de propriedades inversas.

- de um modo geral, a performance a todos os níveis (load/conversão/extracção) varia muito dependendo da ontologia. A estrutura e conteúdo da ontologia em si afectam consideravelmente a performance do sistema, isto independentemente do número de triplos, que são um factor por si só também.

o

7. Referências

Alertamos para o facto das cópias em papel poderem estar desactualizadas.

Consulte o uebe.Q para obter a última versão.