

# OWL-DBC

The Arrival of Scalable and Tractable OWL Reasoning for  
Enterprise Knowledge Bases

URL: [<http://trowl.eu/owl-dbc/>]

Copyright ©2013 the University of Aberdeen. All Rights Reserved

This document is provided for information purpose only and its contents are subject to change without notice. This document is not guaranteed to be error-free.

**The University of Aberdeen is a registered charity organisation in Scotland, UK. Oracle is a registered trademark of Oracle Corporation and/or its affiliates.**

# Overview

This white paper introduces OWL-DBC, a set of JAVA APIs that connects the TrOWL ontology reasoning system with the native inference capability in Oracle Spatial and Graph, an option for Oracle Database. The RDF Semantic Graph feature of Oracle Spatial and Graph provides comprehensive RDF graph data management, SPARQL querying and persisted inferencing in Oracle Database. OWL-DBC allows TrOWL to interoperate with Oracle Spatial and Graph for specialised and more expressive reasoning beyond RDFS, OWL 2 RL and EL profiles, and user-defined rules supported by Oracle Spatial and Graph.

The integration of TrOWL and Oracle Spatial and Graph combines the strengths of both to provide efficient and scalable SPARQL query answering services against ontologies in the World Wide Web Consortium (W3C) standard Web Ontology Language OWL 2, with different levels of expressive power. In order to provide optimal performance for different scenarios, it offers a variety of configurations that allow users to fine-tune their systems. As a whole package, OWL-DBC is especially suitable for enterprise semantic knowledge bases with tight, complex schemata and large amount of data. It enables business applications, such as social network and linked data applications, that use/reuse sophisticated/complex ontologies to exploit efficient and scalable SPARQL query answering services, no matter if the ontology annotated data fits in memory or not.

## **PART 1**

In “What is OWL-DBC” we introduce the OWL-DBC, TrOWL and Oracle Spatial and Graph.

## **PART 2**

In “Why OWL-DBC” we introduce the motivation of using OWL-DBC.

## **PART 3**

In “How OWL-DBC works” we introduce the architecture of OWL-DBC.

## **PART 4**

In “Configuring OWL-DBC” we introduce the configuration options provided by OWL-DBC.

## **PART 5**

In “Best Practices for OWL-DBC” we show with examples how OWL-DBC handles different scenarios.

## **PART 6**

In “OWL-DBC with Real World Ontologies” we show evaluation results of OWL-DBC on some real world ontologies.

## PART 1 What is OWL-DBC?

OWL-DBC is a set of JAVA APIs that connects the TrOWL reasoning system<sup>1</sup> with native inference engine in Oracle Spatial and Graph and provides combined SPARQL query answering services for enterprise semantic knowledge bases. It has the following advantages:

- It enhances the native, forward-chaining based reasoning in Oracle Spatial and Graph and its support for W3C OWL 2 RL and EL profiles with TrOWL's cutting edge syntactic approximate reasoning services, improving its support to complex ontologies in expressive ontology languages.
- It combines the TrOWL reasoning system with the scalable Oracle Spatial and Graph triple/quad store, allowing TrOWL to handle extremely large data sets.
- It provides tractable, sound and practically complete materialisation for OWL 2 DL ontologies, which can be further exploited to perform efficient and scalable SPARQL query answering services.
- It provides multiple configurations that can be used to optimise the performance of the system, depending on the size, language profile, and other factors of the knowledge base and its users.

In later sections of this white paper, we will introduce the TrOWL reasoner and Oracle Spatial and Graph, explain the motivation and architecture of OWL-DBC, review the configuration options, demonstrate the use of OWL-DBC with examples and show its performance on real world ontologies.

### Introduction to TrOWL/REL Reasoner

TrOWL is a Tractable reasoning system for the W3C standard Web ontology Language OWL 2.<sup>2</sup> The approach of TrOWL is to offer tractable support for all the expressive power of OWL 2 by using quality guaranteed language transformations. Currently, TrOWL supports semantic approximation to transform OWL 2 DL ontologies into OWL 2 QL for SPARQL query answering, and syntactic approximation from OWL 2 DL to OWL 2 EL for reasoning tasks like classification, subsumption checking, instance retrieval.

The TrOWL/REL (or simply REL) reasoner, the syntactic approximation component of TrOWL, is an OWL 2 DL reasoner implemented in Java. Its foundation is an optimised OWL 2 EL materialisation algorithm, allowing REL to provide tractable reasoning for OWL 2 EL ontologies. On top of that, further approximate reasoning algorithms are devised to provide tractable reasoning services for OWL 2 DL ontologies. The entire approximate reasoning procedure is soundness-guaranteed. Hence the reasoning results of REL are always correct. Although known to be incomplete in theory, evaluation shows that, REL can perform reasoning on existing benchmarks very efficiently with very high recall (over 99%).

TrOWL/REL is specialised at efficient reasoning for ontologies with complex terminologies and in expressive languages. Users with such ontologies, or require efficient real time reasoning services, can benefit most from TrOWL/REL.

---

<sup>1</sup> <http://trowl.eu/>

<sup>2</sup> <http://www.w3.org/TR/owl2-overview/>

## Introduction to Oracle Spatial and Graph

As part of Oracle Spatial and Graph, an option for Oracle Database Enterprise Edition, Oracle delivers advanced RDF Semantic Graph data management and analysis (formerly Oracle Database Semantic Technologies). With native support for W3C standards – RDF and OWL are standards for representing and defining semantic data and SPARQL is a query language designed specifically for graph analysis – application developers benefit from the industry’s leading open, scalable graph data platform and its fine-grained security. Graphs are becoming central to a new category of social network and linked data applications common in health sciences, finance, media and intelligence communities. RDF Semantic Graph includes: storing and loading RDF/OWL data and ontologies; inference using OWL 2 and user-defined rules; and SPARQL 1.1 query and update.

RDF Semantic Graph includes a native, forward-chaining inference engine for efficient, scalable and persisted inference that supports W3C RDF, RDFS, OWL 2 RL and EL profiles, and user-defined rules. Support for the emerging W3C Simple Knowledge Organization System (SKOS) standard on RDF enables easy sharing of controlled and structured vocabularies such as thesauri, taxonomies, and classification schemes. Inference can be done using any combination of these supported entailment regimes.

Unique capabilities of RDF Semantic Graph include the ability to optimise inference performance for large owl:sameAs cliques with a compact data structure for inference, perform incremental inference to update entailments efficiently after triple inserts, and parallel inference on multi-core or multi-CPU architectures. Support for proof generation provides the derivation of inferred triples and validation can be performed to detect inconsistencies in the original data model and in the entailment.

## **PART 2 Why OWL-DBC?**

This combination synergises the advantages of both TrOWL/REL and Oracle Spatial and Graph. The main advantage of TrOWL/REL is its efficiency in dealing with ontologies with complex TBoxes and/or in expressive languages. However TrOWL/REL stores all the runtime data in main memory, making it less capable when dealing with large scale ontologies, e.g. ontologies with very large ABoxes. On the other hand, Oracle Spatial and Graph provides comprehensive data management, querying and persistent inferencing for RDF triples and quads in Oracle Database that scales for the largest data sets. Its built-in OWL 2 RL and EL profile inference rule sets, however, do not support the expressive power utilised in the most complex ontologies.

OWL-DBC will combine the efficiency and expressiveness of TrOWL with the scalability of Oracle Spatial and Graph to provide a tractable and scalable reasoning and querying infrastructure for enterprise knowledge bases.

## PART 3 How OWL-DBC works

Figure 1 illustrates how the TrOWL/REL and Oracle Spatial and Graph are connected in OWL-DBC. The OWL-DBC suite includes the REL reasoner of the TrOWL infrastructure, and an OWL-DBC API. The OWL-DBC API bridges the TrOWL/REL reasoner and the Jena Adapter for Oracle Database API such that data and query results can be exchanged between TrOWL/REL and Oracle Spatial and Graph. Both OWL-DBC API and Oracle Database can read from OWL files.

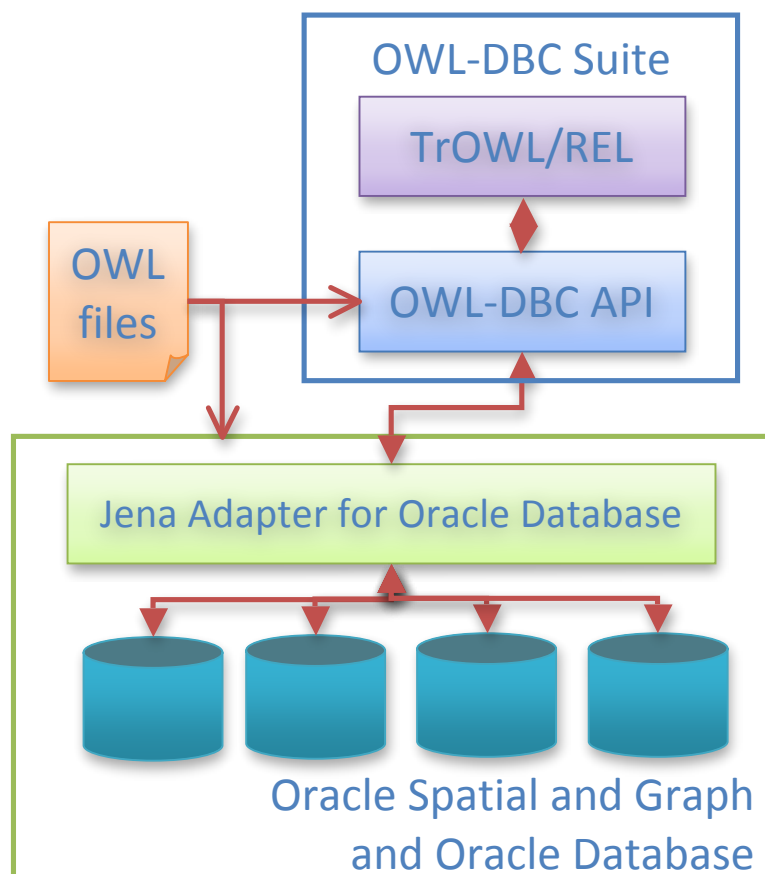


Figure 1. Integrate TrOWL/REL reasoned with Oracle Spatial and Graph via OWL-DBC

## PART 4 Configuring OWL-DBC

In order to provide optimal performance in different scenarios, OWL-DBC offers a variety of configurations that allow users to customise their systems. In this section we review the basic configuration options:

- **Customise language profile:** REL supports OWL 2 EL and OWL 2 DL with different algorithms and different levels of quality guarantees. In OWL-DBC, users can invoke the following method to set the language profile of the ontology:

```
public void setProfile(Profile profile)
```

where Profile is an enumeration type defined with elements including OWL\_2\_DL, OWL\_2\_EL, OWL\_2\_QL and OWL\_2\_RL. If this method is not called, the OWL\_2\_DL will be used as the default language profile. The currently specified language profile can be retrieved with the following method:

```
public Profile getProfile()
```

- **Customise saved results:** In our test we realised that the saving of inference results from the OWL-DBC connected reasoner to Oracle Database constitutes a significant part of the overall time. To minimise the amount of results saved to Oracle Database, we allow the users to decide, whether they want to save concept subsumptions (TBox reasoning results), class assertions and object property assertions (both are ABox reasoning results) or not. In the next two sections, we will further elaborate on this. In addition, OWL-DBC also allows users to decide if they want to save direct or indirect results for some of the inferences. For example, the following method controls whether direct or all sub/super concept relations should be saved:

```
public void setSubType(boolean subType)
```

If the saved subsumption type is not explicitly specified by the above method, OWL-DBC will use `false` as the default value, meaning that all inferable sub/super concept relations will be saved. The following getter can retrieve the current saving results configuration for sub/super concept relations:

```
public boolean isSubType()
```

In addition to the above methods, we also have methods for configuration of other kind of inference, e.g. the class assertions.

- **Customise inference saving mode:** Oracle Spatial and Graph provides different saving modes such as incremental mode and bulk mode for REL inference results. When the amount of inference results to be saved to Oracle Database is inevitably large, it is recommended to choose the bulk mode API and OWL-DBC provides the configurations for users to decide which saving mode they want to use. Such decision can be made via the following method:

```
public void setSavingmode(InferenceSavingMode savingmode)
```

where `InferenceSavingMode` is an enumeration type defined with elements `Incremental` and `Bulk`, in which `Incremental` will be used as the default saving mode. The currently specified saving mode can be retrieved with the following method:

```
public Profile getSavingmode()
```

Furthermore, when the bulk saving mode is used, users can specify a set of bulk saving options with the following method:

```
public void setBulkSavingOptions(String options)
```

For a detailed list of options, we refer the readers to Oracle® Spatial and Graph RDF Semantic Graph Developer's Guide<sup>3</sup>. When no option is specified, the default value will be empty. Similar as above, users can also get the current options with the corresponding get method.

Additionally, user can also choose to use an intermediate pipe as a buffer between REL and Oracle Spatial and Graph. The buffer holds the data saved from REL and waits for Oracle to read these results out simultaneously. To control the throughput of such saving mode, the size of the piped-buffer can be specified with the following method, with 1024 bytes as the default size:

```
public void setPipeBufferSize(int pipeBufferSize)
```

User can also save the inference results to an alternative Oracle semantic model, instead of the original one where the knowledge based is loaded (see the next section for more detail).

---

<sup>3</sup> [http://docs.oracle.com/cd/E11882\\_01/appdev.112/e11828/toc.htm](http://docs.oracle.com/cd/E11882_01/appdev.112/e11828/toc.htm)

## PART 5 Best Practices for OWL-DBC

### Best practice on dealing with small TBox and small ABox

When dealing with small TBox and small ABox, all inferences can be performed in main memory thus the best practice is to use TrOWL/REL to compute the materialisation and then save them in Oracle Spatial and Graph. This includes the following steps:

- Loading the ontology from either Oracle Database or OWL files into REL:

```
// create an OWL-DBC object based on an Oracle semantic
model
OWLDBC odbc = new OWLDBC(modelOracleSem);
// load ontology from file
pathodbc.loadOntology( filePath );
// or, load ontology from Oracle semantic model directly
odbc.loadOntology();
// or, load ontology from Oracle semantic model via file
buffer
odbc.loadOntology( fileBuffer );
```

- Specifying the inference saving mode and saving options. One example of using bulk mode with nested loop joins is as follows. Note that if users choose to save inference incrementally, then this step can be omitted:

```
odbc.setSavingmode(InferenceSavingMode.Bulk);
odbc.setBulkSavingOptions("IZC_JOIN_HINT=USE_NL
MBV_JOIN_HINT=USE_NL MBT_JOIN_HINT=USE_NL");
```

- Performing REL reasoning to compute the materialisation of the ontology. This step will be automatically computed prior to the next step;
- Saving both materialisation results back into Oracle Database triple store:

```
// perform inference and save results back.
// The three parameters indicate whether
// we want to save class hierarchies,
// individual types, individual relations, respectively
odbc.saveInferences(true, true, true);
```

It is also possible to use an alternative method if you want to directly specify the saving options. In this case, the saving mode will be used regardless of the configuration:

```
odbc.saveInferences(true, true, true, savingOptions);
```

In addition to the default in-memory pipe buffer option, it is possible to use an intermediate file, instead of the main memory, as the buffer between REL and Oracle Database. In this method, `szFileAsBuffer` is the buffer file:

```
odbc.saveInferences(true, true, true, savingOptions,
szFileAsBuffer);
```

By default, inference results are always saved into the *same* Oracle semantic model where the knowledge base was originally loaded. But users can also specify an alternative destination by using the following method, in which `modelDest` is the alternative Oracle semantic model:

```
odbc.saveInferences(true, true, true, savingOptions,
szFileAsBuffer, modelDest);
```

When any of the above arguments is `null`, the default option applies.

- Running queries against TBox and ABox stored in Oracle Database. For example, the following query prints all the asserted and inferred types in the ontology:

```
String queryString =
" PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
" +
" PREFIX wine:<http://www.w3.org/TR/2003/PR-owl-guide-
20031209/wine#> "+
" SELECT ?x ?y"+
" WHERE{ ?x rdf:type ?y ."+
" FILTER ( !isBLANK(?x) && !isBLANK(?y)) } ";

Query query = QueryFactory.create(queryString) ;
QueryExecution qexec = QueryExecutionFactory.create(query,
model) ;
try {
    ResultSet results = qexec.execSelect() ;
    ResultSetFormatter.out(System.out, results, query);
}
finally {
    qexec.close() ;
}
```

## Best practice on dealing with small TBox and large ABox

When the ABox is too large to be loaded and processed in main memory, REL can only compute the TBox reasoning results. Thus the best practice is to use TrOWL/REL to compute the materialisation of only TBox and then merge them with ABox data managed in Oracle Database. This includes the following steps:

- Loading an ontology or only ABox into Oracle Database<sup>4</sup>:

```
// create an Oracle instance
Oracle oracle = new Oracle(JDBC_URL, UserName, PassWord);
// create/initialize an oracle semantic model
ModelOracleSem model =
ModelOracleSem.createOracleSemModel(oracle, ModelName);
// read the ontology or ABox into Oracle semantic model
InputStream in = FileManager.get().open( filePath );
model.read(in, "N-TRIPLE");
```

- Loading only the TBox from either Oracle Database or OWL files into REL:

---

<sup>4</sup> Bulk load API can be used if the data size is very large. See [http://docs.oracle.com/cd/E11882\\_01/appdev.112/e11828/sem\\_jena.htm](http://docs.oracle.com/cd/E11882_01/appdev.112/e11828/sem_jena.htm)

```
// create OWL-DBC object for an Oracle semantic model
OWLDBC odbc = new OWLDBC(model);
// load TBox from file path
odbc.loadTBox( filePath );
// or, load TBox from Oracle semantic model directly
odbc.loadOntology();
// or, load TBox from Oracle semantic model via file buffer
odbc.loadOntology( fileBuffer );
```

- Specifying the inference saving mode and saving options. If the user wants to save inference in parallel, the following option can be used, where `dop` denotes the degree of parallelism:

```
"PARALLEL = dop"
```

- Performing REL reasoning to compute the materialisation of the TBox. This step will be automatically computed prior to the next step;
- Saving TBox materialisation results back into Oracle Database with the ABox knowledge

```
// this time, we only need to save the TBox results
odbc.saveInferences(true, false, false);
```

- Running ABox inferences against the combined Oracle Database with the native Oracle Database inference engine. The procedure is similar as the previous example. The major difference is that, after creating the query, instead of directly executing the query, we need to invoke the native OWL inference engine through the Attachment setting. Here the OWLPRIME rulebase is chosen:

```
Attachment attachment = Attachment.createInstance(
new String[] {}, "OWLPRIME",
InferenceMaintenanceMode.NO_UPDATE,
QueryOptions.DEFAULT);

GraphOracleSem graph = new GraphOracleSem(oracle,
szModelName, attachment);
graph.analyze();
graph.performInference(); // parallel inference can be
                          // chosen on a balanced hardware

Query query = QueryFactory.create(queryString) ;
QueryExecution qexec =
QueryExecutionFactory.create(query, new
ModelOracleSem(graph) );
```

- On a balanced hardware, parallel execution can be used to improve the performance of inference and querying. For example, one can set a `dop` (degree of parallelism) in the following API call to enable parallel inference, before calling the `performInference()`:

```
String inferenceOptions="dop=4";
graph.setInferenceOption(inferenceOptions);
```

And parallel querying can be enabled by adding the following SPARQL namespace prefix definition to the beginning of a SPARQL query. For details, please refer to Chapter 7 Jena Adapter in Oracle® Spatial and Graph RDF Semantic Graph Developer's Guide.

```
PREFIX ORACLE_SEM_FS_NS: <http://oracle.com/semtech#dop=8>
```

## Best practice on dealing with large EL Ontologies

When the ontology is in EL, Oracle Semantic RDF Graph can use its native OWLPrime together with the SNOMED component to perform materialisation. However given the fact that every step of the inference process needs to be persisted in Oracle Database, it may not be as efficient as in-memory reasoners. REL implements dedicated in-memory algorithm for EL ontologies. Thus the best practice is to use TrOWL/REL to compute the materialisation of EL ontology (if possible) and then save the results into Oracle Database. This includes the following steps:

- Loading an ontology into Oracle Database;
- Loading the ontology from either Oracle Database or OWL files into REL;
- Specifying the inference saving mode and saving options. In addition the parallel option we mentioned earlier, users need to specify the language profile for REL:

```
// set profile  
odbc.setProfile(Profile.OWL_2_EL);
```

And also, the size of piped buffer between Oracle Database and REL:

```
// set pipe buffer size  
odbc.setPipeBufferSize(1024);
```

In our implementation we have used a multi-threading piped buffer data transfer and empirical results demonstrated that a small pipe size such as 1024B can be quite effective.

- Performing REL reasoning to compute the materialisation of the ontology. Based on the profile, special algorithm or approximation will be applied.
- Saving materialisation results back into Oracle Database.

## PART 6 OWL-DBC with real world ontologies

We present the reasoning performance of OWL-DBC on three real world ontologies, i.e. the WINE ontology, the MGED ontology and the SNOMED CT ontology. The WINE ontology is an OWL DL show case ontology with a rather complex TBox and a small ABox (for ABox reasoning test we generated two synthetic ABoxes). The MGED ontology is an ontology that provides standard terms for the annotation of microarray experiments. The SNOMED CT ontology is a widely used large scale biomedical ontology in EL. The statistics of these ontologies are shown below and the ontologies are included in the OWL-DBC release.

Ontology	Concept No.	Object Property No.	Individual No.
Wine	138	17	206
Wine + synthetic ABox (1,000 assertions)	138	17	1,206
Wine + synthetic ABox (1,000,000 assertions)	138	17	1,000,206
MGED	229	102	658
SNOMED CT	383,836	62	0

We perform the following tests:

Test	Ontology	Loading Mechanism	Reasoning Regime Oracle Spatial and Graph	Query
1	Wine	Entire ontology	No further reasoning	Named concept subsumptions
2	Wine + Synthetic ABox (1,000 assertions)	TBox only	OWLPrime	Types of all individuals
3	Wine + Synthetic ABox (1,000 assertions)	Entire ontology	No further reasoning	Types of all individuals
4	Wine + Synthetic ABox (1,000,000 assertions)	TBox only	OWLPrime	Types of all individuals
5	MGED	Entire ontology	No further reasoning	Types of all individuals
6	MGED	TBox only	OWLPrime	Types of all individuals
7	SNOMED CT	Entire ontology	No further reasoning	Materialisation only, no query

When we load the entire ontology with OWL-DBC, we perform full materialisation for TBox subsumptions, ABox types and relations. Hence no further reasoning is needed

after the results are saved into Oracle Database. When we load only the TBox with OWL-DBC, we perform TBox classification only. Hence OWL 2 reasoning provided by Oracle Spatial and Graph is required to get ABox reasoning results.

The test results<sup>5</sup> are summarised in the following table:

Test	Ontology	Loading Mechanism	Materialisation Time	Query Time	Result #	Recall
1	Wine	Entire ontology	3.43s	0.34s	927	100%
2	Wine + Synthetic ABox (1,000 assertions)	TBox only	40.65s	0.96s	9144	96.2%
3	Wine + Synthetic ABox (1,000 assertions)	Entire ontology	9.06s	0.35s	9502	100%
4	Wine + Synthetic ABox (1,000,000 assertions)	TBox only	437.62s	132.02s	7601047	N/A
5	MGED	Entire ontology	3.10s	0.40s	4633	100%
6	MGED	TBox only	4.98s	0.31s	4663	100%
7	SNOMED CT	Entire ontology	863.7s	N/A	N/A	100%

The materialisation time in the above table includes the loading time, TrOWL/REL reasoning and saving time, and Oracle Spatial and Graph materialisation time (if the loading mechanism is TBox Only). Tests 1-3 and 5-6 do not have parallelisation enabled because the size of the ontology is small. Test 4 uses a degree of parallelism (DOP) 8 when performing OWLPrime inference as well as querying in Oracle Spatial and Graph. Test 7 uses a DOP of 8 when saving the inferred results of REL into Oracle Spatial and Graph.

It's also worth mentioning that performing the same task of Test 7 with Oracle Spatial and Graph alone using its native inference engine took 1957.1s. Compared with results of Test 7, it is demonstrated that using OWL-DBC to connect with TrOWL/REL can improve reasoning performance of Oracle Spatial and Graph for even very big EL ontology.

Most of these tests are included in the release of OWL-DBC<sup>6</sup>.

<sup>5</sup> The tests are run in a computer used as both server and client. The laptop is running Linux 2.6.18-194.el5 X86\_64 with a dual quad core 2.4GHz CPU (Intel Xeon E5620), 32GB RAM, and four 1TB SATA (7200 RPM) disks.

<sup>6</sup> The SNOMED CT ontology is a licensed ontology hence is not included in the release.

## Conclusion

OWL-DBC combines the strength of TrOWL/REL reasoner and Oracle Spatial and Graph, offering efficient and scalable semantic reasoning and SPARQL query answering services. A wide range of configuration options are provided to improve the flexibility and usability of the system. Evaluation on real world ontologies has also demonstrated its effectiveness.

To find out more about OWL-DBC and how you can benefit from it, please visit our website or contact Dr. Jeff Z. Pan (<http://homepages.abdn.ac.uk/jeff.z.pan/pages/>).