Oracle→OpenCyc Interface
release 0.71

3rd August 2004

Contents

1 Introduction

This document contains information on how to install and use the Oracle OpenCyc Interface. The interface enables access to OpenCyc from within the Oracle RDBMS. It is possible to 'copy' data from Oracle into OpenCyc, and it is possible to 'ask' OpenCyc information. There are three ways to access OpenCyc from Oracle:

Java stored procedure	Access to the complete OpenCyc Java API.
PL/SQL	Access to Java stored procedures in `CycJsprocs.java`
SQL	Access to functions in `cyc.pkb`

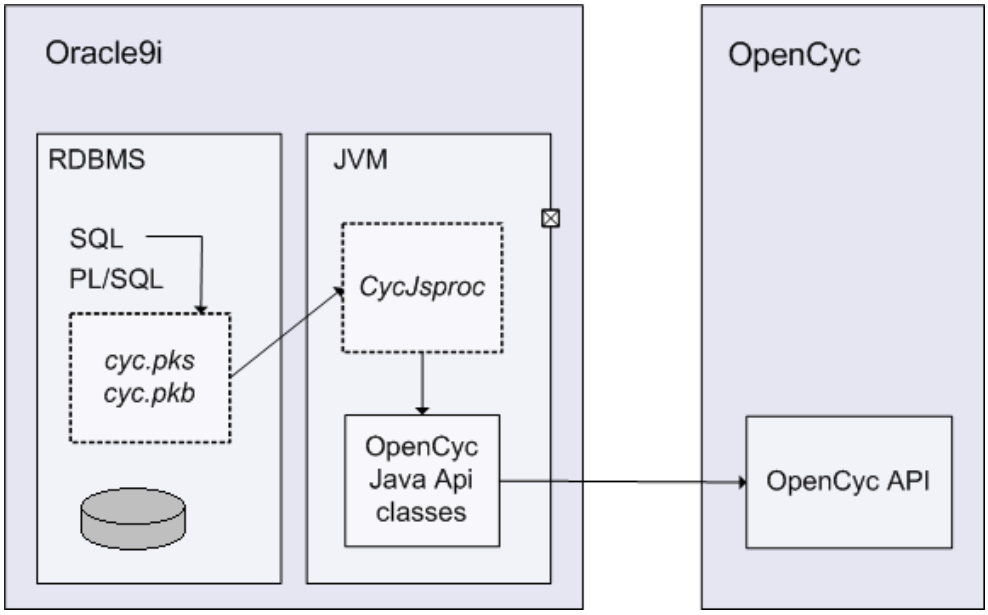

Figure 1: architecture

Writing Java stored procedures gives access to all the features supplied by the OpenCyc Java API, which includes using CycObjects like Cyclist, CycAssertions and CycLobjects such as terms, formulas, predicates, narts etcetera. A small subset of this functionality is visible from SQL. If you want features, go Java stored procedures. If you want to do things quick, write SQL statements.

1.1 Overview

In figure 1, the two rectangles with dotted lines are the Oracle→OpenCyc interface.

1.2 See also

- OpenCyc Api Documentation
- OpenCyc Java Api Documentation
- Oracle Java Developer's Guide
- Oracle Java Stored Procedures Developer's Guide (especially part 3)

- Oracle JDBC Developer's Guide and Reference (part 10 and 18 →connecting to internal driver)

2 Installation

2.1 Prerequisites

You need the following installed on a linux server :

- OpenCyc
- Oracle9*i* (preferably release 2 for speed) or Oracle 10g[1]

2.2 Get the files

The files are downloadable from SourceForge with CVS (see the CVS link on the OpenCyc Sourceforge Project page.) and alternatively in a tarball named `ooi-`x`.tgz` where x is the version number[2] from `http://wwwhome.portavita.nl/~yeb/` This tarball contains the following files:

`install.sh`	A shell script to install the stuff into Oracle.
`oracle-opencyc.jar`	This is a modified version of the official opencyc.jar, which has a slightly modified CycAccess.java to remove references to the Fipa Agent classes, and doesn't contain unused (by Oracle) classes.
`CycJsprocs.java`	The Java Stored Procedures that wrap the methods in the OpenCyc Java Api to Oracle call and data types.
`cyc.pks`	The PL/SQL CYC package specification.
`cyc.pkb`	The PL/SQL CYC package body, contains call specifications for the Java Stored Procedures in CycJsprocs.java.

and some sql scripts that are used by the install script.

2.3 Installing it into Oracle

Untar `ooi-0.71.tar.gz` :

[1] The release of Oracle10g I tested is also MUCH slower than 9 release 2. (2 secs vs 78msecs on a query from TOAD.)

[2] At the time of this writing it's 0.71

```
~$ tar zxvf ooi-0.71.tar.gz
```

Change into the directory and start the install script:

```
~$ cd ooi-0.71
~/ooi-0.71$ ./install.sh
```

It will ask for the directory where OpenCyc was installed. Then it will ask for the password of the Oracle SYSTEM user. The password of the system user is needed to create the new user/schema to contain the java objects. The next few sections describe the important parts of the script, in case you want to do it manually.

2.3.1 Create user cyctest (install.sh)

The java classes of the OpenCyc Java API have to be loaded into the schema of a user. Though it's possible to separate the oracle user who's schema contains the jars from the users that actually use it, the examples in this guide put and use all the stuff into the schema of a sinlge user named `cyctest`. The `install.sh` script doesn't use a TNS connect string, so if you want to install the stuff on a remote Oracle server you need to add the TNS identifier.

The user needs the grant `javauserpriv`.

2.3.2 Load the jars into Oracle (install.sh)

Next in the install script is loading some of the jars supplied in the opencyc-0.x.0/lib directory. Loading in Oracle 9i release 1 will take 15 to 30 minutes, in 9iR2 this is done in max one minute. There should be no errors, but if there were, which might occur if you try this with other jars a while after I write this documentation, you can view the errors with

```
SQL> select * from user_errors
```

This will probably show that there were references to unresolved classes. Find these classes, load them, and try the resolve command again. You can view the status of all loaded java classes with the SQL command

```
SQL> SELECT dbms_java.longname(object_name) as name, status, created
FROM user_objects
WHERE object_type='JAVA CLASS'
```

If the status is `VALID` is means that the class is resolved and can be used (called) by the database. Status `INVALID` means that it hasn't been resolved (yet). *Please note*: The order of loading without resolving doesn't matter. But the order of resolving can be important, if not all necessary classes are loaded at before the first 'resolve' attempt. It can happen that errors dissappear after dropping the user and loading all classes from scratch, though this happens very rarely.

2.3.3 Load CycJsproc.java (install.sh)

Once `CycAccess` is resolved, the 'Java Stored Procedure wrapper methods java source file' is loaded by the `install.sh` script. Note that this time a java source instead of class is loaded. Also, the class is now resolved at load time.

2.3.4 Load the CYC package (install.sh)

After `CycJsproc.java` is loaded and resolved, the CYC PL/SQL package is loaded.

2.3.5 Small test (install.sh)

When the CYC package is loaded, a small test is done by executing `test.sql` which asks opencyc for its current time. The following output should be displayed:

```
SQL*Plus: Release 9.2.0.1.0 - Production on Wo Dec 11 19:24:10 2002

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.


Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production

make connection to opencyc

PL/SQL procedure successfully completed.

ask for the time

CYCS_TIME
```

```
--------------------------------------------------------------------------------
(SecondFn 12 (MinuteFn 24 (HourFn 19 (DayFn 11 (MonthFn December (YearFn 2002)))
)))


end connection to opencyc

PL/SQL procedure successfully completed.

Disconnected from Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
```

3 Usage

At this point access to OpenCyc is enable from Oracle. Every function in the CYC package can be used in every SQL query (issued from SQLPlus, or for example a PHP script in a web page), and in PL/SQL you can call every function or procedure.

3.1 The first query

All the following commands can be performed in sqlplus in the cyctest schema. Connect to the database, and in the database session, connect to cyc

```
SQL> BEGIN cyc.makeconnection(); END;
  2  /
```

`Begin` and `end` in SQL? Well, it's actually PL/SQL. Oracle allows you to give 'anonymous' PL/SQL blocks (a block can be recognized by BEGIN and END) where a SQL query could be executed. This is the way a PL/SQL procedure is called from an SQL frontend.[3] Now to the first question *is Dog a collection?*

```
SQL> SELECT cyc.isquerytrue(
            '(#$isa #$Dog #$Collection)', 'InferencePSC')
       FROM DUAL
```

this proceduces the following output

[3]The same syntax is also used when calling PL/SQL from e.g. PHP ($query = "BEGIN ... END"') or from a handy Oracle client like TOAD.

```
CYC.ISQUERYTRUE('(#$ISA#$DOG#$COLLECTION)','INFERENCEPSC')
----------------------------------------------------------
                                                         1
```

For the type mapping between the different Cyc and Oracle types, see the source `CycJsproc.java` and `cyc.pkb`, or the summary below. Note that a query like this could also be put in e.g. a before trigger, and raise an exception if something is not true.

3.2 Oracle puts data in OpenCyc

Before OpenCyc can say anything interesting about your data, you have to put some information in your database into OpenCyc. This is easy. In the Oracle demo user SCOTT's schema is a table EMP. This table contains 14 employees, with the following names

```
SQL> SELECT ename FROM scott.emp;

ENAME
----------
SMITH
ALLEN
WARD
JONES
MARTIN
BLAKE
CLARK
SCOTT
KING
TURNER
ADAMS
JAMES
FORD
MILLER

14 rows selected.
```

The first thing that will be added is a Microtheory to put SCOTT's stuff in. This microtheory will be called `#$OOITestMt`. Execute the following PL/SQL anonymous block:

```
DECLARE
  genmts_tab cyclist_type;
BEGIN
  genmts_tab := cyclist_type('BaseKB', 'HumanSocialLifeMt' );
  cyc.createMicrotheory(
    'OOITestMt',
    'A Microtheory to test the Oracle OpenCyc Interface.',
    'Microtheory',
genmts_tab );
END;
/
```

Now assert that each person that is named ENAME in SCOTT.EMP is an employee:

```
SELECT cyc.assertWithTranscript(
       '(#$isa #$OOITest:' || ename || ' #$Employee)', 'OOITestMt' )
  FROM scott.emp;

CYC.ASSERTWITHTRANSCRIPT('(#$ISA#$OOITEST:'||ENAME||'#$EMPLOYEE)','OOITESTMT')
--------------------------------------------------------------------------------
(#$isa #$OOITest:SMITH #$Employee)
(#$isa #$OOITest:ALLEN #$Employee)
(#$isa #$OOITest:WARD #$Employee)
(#$isa #$OOITest:JONES #$Employee)
(#$isa #$OOITest:MARTIN #$Employee)
(#$isa #$OOITest:BLAKE #$Employee)
(#$isa #$OOITest:CLARK #$Employee)
(#$isa #$OOITest:SCOTT #$Employee)
(#$isa #$OOITest:KING #$Employee)
(#$isa #$OOITest:TURNER #$Employee)
(#$isa #$OOITest:ADAMS #$Employee)
(#$isa #$OOITest:JAMES #$Employee)
(#$isa #$OOITest:FORD #$Employee)
(#$isa #$OOITest:MILLER #$Employee)

14 rows selected.
```

3.3 Oracle gets data from OpenCyc

Now OpenCYC knows about the employees, we can ask stuff... *Is Smith a person?*

```
  SELECT cyc.isQueryTrue(
         '(#$isa #$OOITest:SMITH #$Person)', 'InferencePSC' )
    FROM dual;

CYC.ISQUERYTRUE('(#$ISA#$OOITEST:SMITH#$PERSON)','INFERENCEPSC')
----------------------------------------------------------------
                                                              1
```

Who are all the employees known to OpenCyc?

```
  SELECT *
    FROM TABLE(
  SELECT cyc.askwithvariable(
         '(#$isa ?X #$Employee)', '?X', 'InferencePSC' )
    FROM DUAL);

COLUMN_VALUE
--------------------------------------------------------------------------------
OOITest:SMITH
OOITest:ALLEN
OOITest:WARD
OOITest:JONES
OOITest:MARTIN
OOITest:BLAKE
OOITest:CLARK
OOITest:SCOTT
OOITest:KING
OOITest:TURNER
OOITest:ADAMS
OOITest:JAMES
OOITest:FORD
OOITest:MILLER

14 rows selected.
```

Because the type of this result is an Oracle SQL resultset, it can be used in all Oracle SQL constructs. Union, order, group by etc etc.

3.4 Oracle removes data from OpenCyc

Removing of knowledge is either removing axioms or removing constants (with all axioms asserted on those constants).

3.4.1 Killing constants

Completely deleting a constant and all the knowledge asserted on the constans is done with the `kill` procedure. There is not a functional variant to enable `kill` from an SQL query, because that would make it too easy. Now, first make a query that returns a list of the OpenCyc constant names of all the constants you want to remove. For example

```
  SELECT *
    FROM TABLE(
  SELECT cyc.askWithVariable(
         '(#$isa ?X #$Employee)', '?X', 'OOITestMt' )
    FROM DUAL);
```

If it returns the right list of constants to kill, cut and paste it into the following procedure.

```
BEGIN
    FOR r IN (
        SELECT COLUMN_VALUE AS cons FROM TABLE (
             SELECT cyc.askwithvariable(
               '(#$isa ?X #$Employee)',
               '?X',
               'OOITestMt'
             ) FROM DUAL
        )
    )
    LOOP
        cyc.kill( r.cons );
    END LOOP;
END;
```

3.5 End the connection to OpenCyc

Don't forget to end the connection at the end of the Oracle session

```
SQL> begin cyc.endconnection(); end;
```

3.6 Type mapping

OpenCyc	Java	Oracle
list	java.lang.Array↔oracle.sql.ARRAY	VARRAY (aka PL/SQL Table)
-	java.lang.?	DATE
-	boolean	NUMBER in [0,1]

In this case, 1 means true. (yes, Oracle SQL doesn't know booleans. PL/SQL does however.)

3.7 Method summary

This is a list of the methods specified in the package specification `cyc.pks`:

```
PROCEDURE makeConnection;

PROCEDURE makeConnection( hostname_in IN VARCHAR2 );

PROCEDURE endConnection;

PROCEDURE makeCycConstant( constant_in IN VARCHAR2 );

PROCEDURE createMicrotheory(
    mtname_in IN VARCHAR2,
    comment_in IN VARCHAR2,
    isamt_in IN VARCHAR2,
    genlmts_in IN cyclist_type );

PROCEDURE createMicrotheorySystem(
    mtname_in IN VARCHAR2,
    isamt_in IN VARCHAR2,
    genlmts_in IN cyclist_type  );

PROCEDURE assertGaf(
    gaf_in IN VARCHAR2,
    mt_in IN VARCHAR2 );
```

```
PROCEDURE unassertGaf(
    gaf_in IN VARCHAR2,
    mt_in IN VARCHAR2 );

FUNCTION assertGaf(
    gaf_in IN VARCHAR2,
    mt_in IN VARCHAR2 )
RETURN VARCHAR2;

PROCEDURE assertWithTranscript(
    sentence_in IN VARCHAR2,
    mt_in IN VARCHAR2 );

FUNCTION assertWithTranscript(
    sentence_in IN VARCHAR2,
    mt_in IN VARCHAR2 )
RETURN VARCHAR2;

FUNCTION isQueryTrue(
    query_in IN VARCHAR2,
    mt_in IN VARCHAR2 )
RETURN NUMBER;

FUNCTION askWithVariable(
    query_in IN VARCHAR2,
    variable_in IN VARCHAR2,
    mt_in VARCHAR2 )
RETURN cyclist_type;

FUNCTION askWithVariable(
    query_in IN VARCHAR2,
    variable_in IN VARCHAR2,
    mt_in IN VARCHAR2,
    backchain_in IN NUMBER )
RETURN cyclist_type;

FUNCTION askWithVariables(
    query_in IN VARCHAR2,
```

```
    variables_in IN VARCHAR2,
    mt_in IN VARCHAR2 )
RETURN cyclist_type;

FUNCTION getBackChainRules( predicate_in IN VARCHAR2 )
RETURN cyclist_type;

FUNCTION converseList( command_in IN VARCHAR2 )
RETURN cyclist_type;

FUNCTION converseString( command_in IN VARCHAR2 )
RETURN VARCHAR2;

FUNCTION converseObjectToString( command_in IN VARCHAR2 )
RETURN VARCHAR2;

FUNCTION converseEscapedList( command_in IN VARCHAR2 )
RETURN VARCHAR2;

PROCEDURE converseVoid( command_in IN VARCHAR2 );

FUNCTION converseVoid( command_in IN VARCHAR2 )
RETURN cyclist_type;

FUNCTION getKnownConstantByName( name_in IN VARCHAR2 )
RETURN VARCHAR2;

PROCEDURE kill( constant_in IN VARCHAR2 );

FUNCTION escapeList( list_in IN VARCHAR2 )
RETURN VARCHAR2;

PROCEDURE truncate_collection ( collection_name_in IN VARCHAR2 );

PROCEDURE openAskWithVariable(
    query_in        IN VARCHAR2,
    variable_in     IN VARCHAR2,
    mt_in           IN VARCHAR2,
```

```
    backchain_in    IN NUMBER,
    bindings_out    OUT generic_curtype       -- reference cursor
);
```

3.8 Debugging

In the beginning of the `CycJsprocs.java` source you'll find the method named `makeConnection()`. At the end of this method is the call to `CycAccess.traceOn`. The trace of CycAccess is default on. If you do not want a lot of logging, comment this call and reload CycJsprocs.java. Standard output (System.out.println) is dumped by oracle in trace files in the directory
`$ORACLE_BASE/admin/<instancename>/udump`. Find the last trace file with `ls -l --sort=time -r` and then monitor the contents with less or tail -f.

3.9 Exceptions

In Oracle, all java exceptions appear as ORA-29532 errors. At the end of the text of the error message you should see the java error. So, if there is an error in your OpenCyc formula, look good at the ORA-29532 errors!
If you get only errors when calling cyc functions; check whether you issued a `cyc.makeconnection()`!