Oracle→OpenCyc Interface

Yeb Havinga

June 12 2002

Contents

1 Introduction **1**
1.1 Overview . . . 1
1.2 See also . . . 1

2 Installation **2**
2.1 Prerequisites . . . 2
2.2 Getting the files . . . 2
2.2.1 Third party jars . . . 2
2.2.2 Oracle→OpenCyc interface files . . . 3
2.3 Loading the stuff in Oracle . . . 3
2.3.1 Prepare Oracle . . . 3
2.3.2 Load the jars into Oracle . . . 4
2.3.3 Resolve CycAccess . . . 4
2.3.4 Load CycJsproc.java . . . 5
2.3.5 Load the CYC package . . . 5

3 Usage **6**
3.1 The first query . . . 6
3.2 Oracle puts data in OpenCyc . . . 7
3.3 Oracle gets data from OpenCyc . . . 8
3.4 Type mapping . . . 9
3.5 Method summary . . . 9
3.5.1 askwithvariable . . . 10
3.5.2 getbackchainrules . . . 10
3.5.3 converselist . . . 10

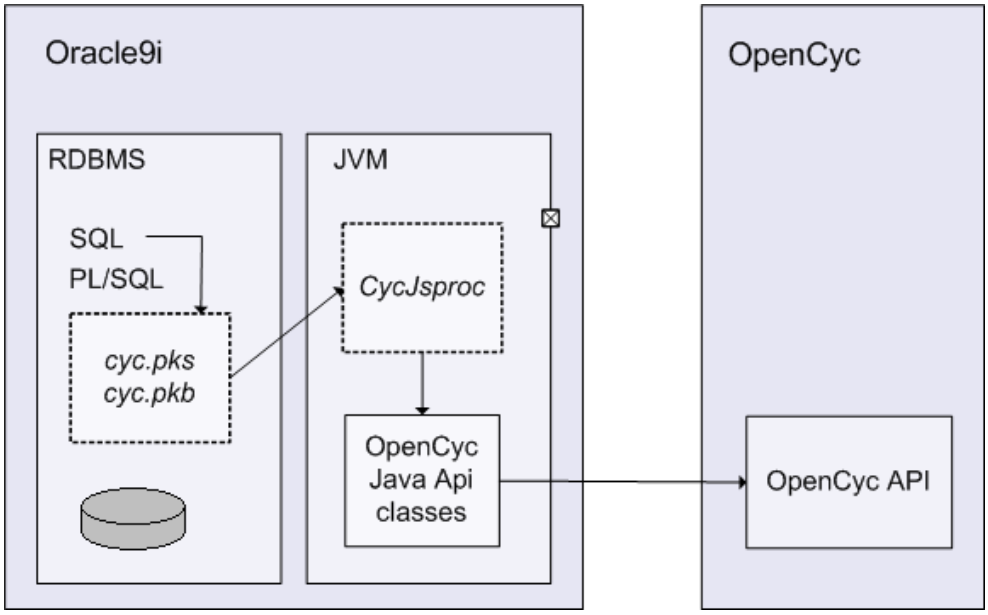

Figure 1: architecture

3.6 Debugging 10
3.7 Exceptions 10

4 End notes **10**
4.1 Quality of this software 10

1 Introduction

This software is kind of new. Expect procedures and parameters to change.. This document contains information on how to install and use the Oracle→OpenCyc interface.

1.1 Overview

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*
- J2SDK

2.2 Getting the files

2.2.1 Third party jars

These jars are used by the OpenCyc Java Api interface, and need to be loaded into Oracle. The list is fairly small, because the `oracle-opencyc.jar` contains only a subset of the full org/opencyc tree.

- jakarta-oro-2.0.6.jar
- violinStrings.jar
- jdom.jar (jdom.jar is somewhere in jdom-b8/build after unzip)
- xmlParserAPIs.jar (in xerces 2.0.1 bin)

You can put these files anywhere. This manual is written like you've put them in your `opencyc-0.6.0/lib` directory.

2.2.2 Oracle→OpenCyc interface files

The Oracle→OpenCyc interface consists of the following files :

`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.
`oracle-opencyc-src.tgz`	A gzipped tar file of the java sources in the previous jar. You only need this if you want to compile the above jar yourself.
`loadjars.sh`	A shell script for easy loading the third party jars into Oracle.
`cycjsprocs.jar`	Contains CycJsprocs.java, which contains the Java Stored Procedures that wrap the methods in the OpenCyc Java Api to Oracle call and dat
`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.

You can get these files with CVS from SourceForge. Look in the org/opencyc/oracle directory.

2.3 Loading the stuff in Oracle

2.3.1 Prepare Oracle

Create a user to hold the Java classes to be loaded. 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 using the same user. In this guide you won't see a TNS connect string either, because I installed Oracle and OpenCyc on the same server. If you know how to use TNS, you'll know what to do.

```
$ sqlplus system/manager (or the right password)

SQL> create user cyctest identified by cyctest
     default tablespace users temporary tablespace temp;

SQL> grant connect,resource,javauserpriv to cyctest;
```

(if you want to execute the examples below, access to the scott demo user tables is necessary)

```
SQL> grant select any table to cyctest;

SQL> quit/
```

IMPORTANT! DON'T FORGET THE JAVAUSERPRIV PRIVILEGE!!

2.3.2 Load the jars into Oracle

Edit the `loadjars.sh` script so it contains the right place where you put the downloaded jar files. Do a `chmod +x loadjars.sh` if it's not executable (I don't know what CVS does with file permissions, just to be sure), and execute it:

```
bash$ ./loadjars.sh
```

You should now get stuff like

```
arguments: '-f' '-verbose' '-oci8' '-u' 'cyctest/cyctest' '/home/yourhome/opencyc-0.6
creating : resource META-INF/MANIFEST.MF
arguments: '-f' '-verbose' '-oci8' '-u' 'cyct/cyct' '/home/yourhome/opencyc-0.6.0/lib
creating : class org/jdom/EntityRef
arguments: '-f' '-verbose' '-oci8' '-u' 'cyctest/cyctest' '/home/yourhome/opencyc-0.6
arguments: '-f' '-verbose' '-oci8' '-u' 'cyctest/cyctest' '/home/yourhome/opencyc-0.6
arguments: '-f' '-verbose' '-oci8' '-u' 'cyctest/cyctest' 'oracle-opencyc.jar'
creating : resource META-INF/MANIFEST.MF
creating : resource META-INF/MANIFEST.MF
created  :CREATE$JAVA$LOB$TABLE
loading  : resource META-INF/MANIFEST.MF
loading  : resource META-INF/MANIFEST.MF
loading  : class org/jdom/EntityRef
etc..
etc..
```

on your screen. Ignore the

```
Error while loading resource META-INF/MANIFEST.MF
    ORA-00001: unique constraint (CYCT.SYS_C0012106) violated
```

because the manifest files are not used. Get a cup of coffee, this will take 15-30 minutes. Use `ps aux | grep loadjava` to check if the load has finished. The `loadjava` command in `loadjars.sh` doesn't contain the `-resolve` parameter. So right now, the only check on the classes being done by Oracle is whether they are Java classes or not.

2.3.3 Resolve CycAccess

Connect to Oracle with the schema user you loaded the classes in, and resolve CycAccess with the following SQL command:

```
SQL> alter java class "org/opencyc/api/CycAccess" resolve;
```

If you get the message that there were errors, which probably 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.

2.3.4 Load CycJsproc.java

Once `CycAccess` is resolved, the Java Stored Procedure wrapper methods can be loaded. You're still in the CVS org/opencyc/oracle directory? Then:

```
$ jar xvf cycjsprocs.jar CycJsprocs.java
$ loadjava -f -resolve -verbose -oci8 -u cyctest/cyctest CycJsprocs.java
```

Note that this time a java source instead of class is loaded. Also, the class is now resolved at load time. If this fails, check the user_errors again.

2.3.5 Load the CYC package

Once `CycJsproc.java` is loaded and resolved, you can load the CYC PL/SQL package. First load the package specification. Note that this file also creates the type `cyclist_type`

```
$ sqlplus cyctest/cyctest @cyc.pks
```

Now load the package body

```
$ sqlplus cyctest/cyctest @cyc.pkb
```

You should get the gentle message `Package body created.`

3 Usage

At this point, the power of OpenCyc is at your Oracle fingertips. 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 starts with BEGIN and ends with END) where a SQL query could be executed. This is the way a PL/SQL procedure is called from an SQL frontend. Now to the first question *is Dog a collection?*

```
SQL> select cyc.isquerytrue( '(#$isa #$Dog #$Collection)', 'InferencePSC') from dual
```

this proceduces the following output

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

For the type mapping between the different Cyc and Oracle types, see 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.
```

Assert that these people are Employees in OpenCyc's HumanSocialLifeMt

```
SQL> select cyc.assertgaf( '(#$isa #$' || ename || ' #$Employee)',
  2  'HumanSocialLifeMt' )
  3  from scott.emp;

CYC.ASSERTGAF('(#$ISA#$'||ENAME||'#$EMPLOYEE)','HUMANSOCIALLIFEMT')
--------------------------------------------------------------------------------
(#$isa #$SMITH #$Employee)
(#$isa #$ALLEN #$Employee)
(#$isa #$WARD #$Employee)
```

```
(#$isa #$JONES #$Employee)
(#$isa #$MARTIN #$Employee)
(#$isa #$BLAKE #$Employee)
(#$isa #$CLARK #$Employee)
(#$isa #$SCOTT #$Employee)
(#$isa #$KING #$Employee)
(#$isa #$TURNER #$Employee)
(#$isa #$ADAMS #$Employee)
(#$isa #$JAMES #$Employee)
(#$isa #$FORD #$Employee)
(#$isa #$MILLER #$Employee)

14 rows selected.
```

GAF stands for 'Ground Atomic Formula'. Though `CYC.ASSERTGAF` is a function, and shouldn't change anything in the database, it's very handy that Oracle allows it to change things outside the database, in this case. It's very fundamental in Oracle that functions do not change the database state. And I think if you know a bit PL/SQL, calling cyc.assertGaf from a procedure in PL/SQL LOOP might be bit more proper. Assertgaf returns it's input, because functions are supposed to return something.

3.3 Oracle gets data from OpenCyc

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

```
SQL> select cyc.isquerytrue( '(#$isa #$SCOTT #$Employee)', 'InferencePSC') from dual
  2  /

CYC.ISQUERYTRUE('(#$ISA#$SCOTT#$EMPLOYEE)','INFERENCEPSC')
----------------------------------------------------------
                                                         1
```

Who are all the employees?

```
 SQL> select column_value from
  2  the( select cyc.askwithvariable(
  3          '(#$isa ?X #$Employee)',
  4          '?X',
```

```
  5          'InferencePSC' )
  6       from dual );

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

14 rows selected.
```

Because this is a REAL oracle resultset, it can be used in all wild Oracle SQL constructs. Union, order, group by etc etc. Don't forget to end the connection at the end of the Oracle session

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

3.4 Type mapping

OpenCyc	Java	Oracle
-	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.)

3.5 Method summary

At this point, you really must check the CycJsproc source.

3.5.1 askwithvariable

3.5.2 getbackchainrules

3.5.3 converselist

To get all hypothesized constants.

```
select column_value from the
(select cyc.converselist( '(constant-complete "HYP-")' ) from dual)

select cyc.converselist( '(cyc-kill #$' || column_value || ')' ) from the
(select cyc.converselist( '(constant-complete "HYP-")' ) from dual)
```

3.6 Debugging

In the beginning of the `CycJsprocs.java` source you'll find the method `makeConnection()`. At the end of this method is the call to `CycAccess.traceOn`. The trace of CycAccess is default off, so if you comment this call 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.

3.7 Exceptions

All java exceptions are catched and thrown to the caller. 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 CYC query, look good at the ORA-29532 errors!

4 End notes

4.1 Quality of this software

There should be no problems installing this software, I tested the installation procedure while writing this documentation. However, please be aware that this is a very first version of this interface. The current version must be seen as 'demonstrate that the interface works' and nothing more. I did a few tests with bigger result sets and I haven't had one strange error yet, both Oracle Java support and the OpenCyc Java

API (and OpenCyc ofcourse :) seem so robust that this interface software simply cannot trigger an 'internal error' or something. One more note: Cyclists get converted to java.lang.Arrays, which in turn are converted to a oracle.sql.ARRAY, which then are converted into a Oracle TABLE OF VARCHAR2. This is the most important conversion of the whole interface, and is the first I got working, and currently only supports un nested lists.