



Solutions to Complex Data Integration



Instance Traceability and Regression Test Systems

2008



Contents

CONTENTS	2
EXECUTIVE SUMMARY	3
INTRODUCTION	3
WHAT IS INSTANCE TRACEABILITY?	3
INSTANCE TRACEABILITY AND REGRESSION TEST SYSTEM	4
INSTANCE TRACEABILITY - PRACTICAL APPROACHES	4
RAW INSTANCE TRACEABILITY VIA SML.....	4
VIA INSTANCE BREAK POINTS IN THE SML DEBUGGER.	5
INTEGRATED TRACEABILITY AND REGRESSION TEST SYSTEM.....	5
FUTURE DEVELOPMENTS	9
FURTHER INFORMATION	9



Executive Summary

As part of our work on comparing transforms of other types with TM transforms (we are also comparing with XSLT and evaluating an XSLT analyser), a number of improvements have been made for producing TM transforms when an alternative transform is available. In these cases, there is usually a set of regression tests results, source and target transforms, which we can compare with equivalent targets generated from the same sources using TM transforms.

ETL have had for some time a full regression test suite, which automatically compares results. During our recent work we have now added

1. Differences Comparator
2. Instance Traceability

The regression suite (including the differences comparator), together with instance traceability have now been added to TM Design Tool, to provide features that will allow equivalent transforms to be developed more quickly, by first highlighting the differences in the output and the TM statements that have caused the differences.

Introduction

One of the prime reasons for using the SML language to describe transforms is to reduce development and maintenance costs. SML does this in a variety of ways but principally by making the transforms simple, exposed, and visible. However, this is not the whole story. The Transform Design Tool also includes a number of very useful tools that leverage SML's advantages several stages further. This document describes just one of these toolsets, the instance traceability toolset. It describes the concepts involved and then proceeds to explain how they would be used in real life.

What is instance traceability?

It is important to distinguish instance traceability from meta-data traceability, especially as Transform Design Tool supports both these concepts. Meta-data traceability is a feature that allows a (suitably privileged) user of TM to discover what parts of an input document are used to form which parts of an output document, or, conversely, what parts of an output document depend upon which parts of an input document. The first of these we refer to as forward (meta-data) traceability, the second as backward traceability. This analysis is a static analysis, which is performed whenever a set of project code is generated. During this process, TM uses its complete understanding of models and SML transformation descriptions.

Note that this type of analysis is not limited to one project. A very important feature of the meta data traceability is that a series of projects that transform data through a series of stages may be linked together in a pipeline. A typical use of forward traceability is to perform impact analysis - we can ask questions such as, "If this field input data were to change - which fields further down the pipeline could possibly be affected?" A typical use of backward traceability is some kind of audit process - here we can ask questions such as, "Which fields in the original input data are being used to establish the figures which appear in some final report?" Note that, currently, the TM meta-data traceability feature is primarily concerned with asking the questions "Which" - or "What" - not to question "How." If the question begins with "Which" or "What" - i.e. "Which fields will be affected?" or, "Which fields are responsible?", then meta-data traceability is the correct tool to use. On the other hand, "How" questions - i.e. "How is this data being manipulated in the transform to form these final figures?" were until recently only answerable directly from the SML itself. However, the introduction of instance traceability now means that a tool is available to assist in answering those "How" questions - at least in the context of transforming specific instances. Instance traceability thus allows the system to produce a very intelligent trace showing exactly how any particular instance item in the target has been created.



Instance traceability and regression test system

A particularly powerful way of using the instance traceability feature is in conjunction with the new regression test system, which is integrated into the Transform Design Tool itself. This regression system allows you to set up a series of source items, expected target items and transforms, and automatically to execute the transforms and compare the results. This is, of course, an essential feature of any commercially developed system. However, in addition, the instance traceability feature can be used in conjunction with a special ETL supplied comparator, which both identifies differences and the code that created those differences.

Instance traceability - practical approaches

Transform Design Tool supports three different ways of using instance traceability:

Raw Instance Traceability via SML

Instance break points in SML debugger

Integrated Traceability and Regression test system

Raw Instance Traceability via SML

Two SML calls control the operation of raw SML traceability

```
OUTPUTOBJECTNUMBERS  
SETTRACKOBJECTS.
```

OUTPUTOBJECTNUMBERS takes up to two parameters: a Boolean flag, which should be set to true if you want to output parameters, and a string, which is the name of the object instance identifier that you wish to add to every item you create. By default, this is the string '_oid' as defined in net.etl.tmc. It is not advisable to change this string unless you have good reason to do so (i.e. if there will be name clashes in your model between '_oid' and some attribute name of your own.) OUTPUTOBJECTNUMBERS needs to be called once only during the process of the transform. Once it has been called, then all nodes in the target will be assigned unique numbers. In XML, this will look like:

```
<?xml version="1.0" encoding="UTF-8"?>  
<Ingredients _oid="1">  
  <RqdSubstance _oid="2">  
    <Substance name="H2S04" _oid="0" />  
    <Qty _oid="3">17.51624548775</Qty>  
  </RqdSubstance>  
  <RqdSubstance _oid="5">  
    <Substance name="Phosgene" _oid="4" />  
    <Qty _oid="6">15.1485148350</Qty>  
  </RqdSubstance>  
  <RqdSubstance _oid="8">  
    <Substance name="TeraHydrateChlorinate" _oid="7" />  
    <Qty _oid="9">8.09603959515</Qty>  
  </RqdSubstance>  
  .....
```

Note that _oid numbers will not necessarily form a complete series or be in order through the document.

Once we are creating numbered output, we can use SETTRACKOBJECTS track activities on numbered objects. SETTRACKOBJECTS takes three parameters:

- (a) itemsList - this is to list of items we wish to track. This can be a list of comma-separated numbers, a list of comma-separated node names, or a mixture of the two. This function also accepts a 'jobject' of type java.util.List. containing such a list. Finally it is also possible to specify '\$TRACKALL'. This means that all items are tracked.
- (b) doAll - By default this optional parameter is false. If set to true, then all actions on this object ID are tracked - including those which set attributes or relationships.



(c) outputFile - By default this is \transman\diag.track.txt. Note that this file is not created unless it does not exist. It will be expanded each time it is updated in reverse chronological order.

A typical output for tracking all objects will begin something like:

```
Tracking list for TestProject.V1.m0 at Mon Oct 11 10:18:33
=====
----- Activity for @=0'Transaction' -----
@=0 created In:2
Map Transaction<-trd:IrML
----- Activity for @=1'Trade' -----
@=1 created In:1
Map Trade<-trd:irTrade
Transaction <- trd:irPackage [1st iteration]
Transaction <- trd:IrML [1st iteration]
----- Activity for @=2'tradeDateTime' -----
@=2 created In:10
Map Trade<-trd:irHeader
Trade <- trd:irTrade [1st iteration]
Transaction <- trd:irPackage [1st iteration]
Transaction <- trd:IrML [1st iteration]
.....
```

This shows which line has been used to create, and which series of transform calls led to the creation of, the item in question. Note that parameters passed on the stack are also noted e.g.

```
----- Activity for @=528'MarketRateIndexObservation' -----
@=528 created
Map FloatingRateFormula<-FpML:paymentCalculationPeriod
CalculationEvent <- FpML:paymentCalculationPeriod [1st iteration]
AccrualFormula <- FpML:paymentCalculationPeriod [1st iteration]
CashSettlementFlow <- FpML:paymentCalculationPeriod [3rd iteration]
SettlementSchedule <- FpML:cashflows [4th iteration]
SettlementSchedule <- FpML:swapStream [1st iteration]
Stream <- FpML:swapStream [1st iteration]
Trade <- FpML:swap [2nd iteration]
  With keys and values
  P1: FLO
  P2: MODFOLLOWING
Trade <- FpML:trade [1st iteration]
Trade <- trd:irTrade [1st iteration]
Transaction <- trd:irPackage [1st iteration]
Transaction <- trd:IrML [1st iteration]
```

Via Instance break points in the SML debugger.

Tracking Instance creation via the debugger is very easy. All you have to do is add a data break point in the Debug | Conditional Breakpoint.

A data break point is specified by choosing the @ symbol from the list of drop down conditions. Note that the data break point may be specified by number, or by node name.

Once a break point has been set, then execution will pause after creation of the specified element, or during modification of the specified element. Normally, you will be able to navigate up and down the execution stack switching into the relevant transforms at will. If you wish the break point to only affect creation of the specified element to choose then C@ simple from the drop-down list.

Integrated Traceability and Regression test system

The Integrated Traceability and Regression test system is accessed from the Transform Design Tool using Tools|Regression Suite. Note that you should not include the SML calls



OUTPUTOBJECTNUMBERS & SETTRACKOBJECTS in code that is traced via the Integrated tool kit - there is no need to do so and you may confuse the system.

There are currently two tools in the Regression Suite
Load External Difference File...
Run Tests Now...

Run Tests Now is the main entry point to the regression test system. You will be invited to choose a properties file describing the regression test. Once that has been selected, the test system is run immediately. During running, you will not be able to use the Transform Design Tool - a message will be displayed on the status bar during this process. Note that when the regression test system is run directly from the Design Tool the Java code is not automatically rebuilt. You may want to rebuild the code manually first by pressing Control-B, if you have changed it. The actual behaviour of the regression test system is specified using a properties file - this allows the same regression system to be run externally from the Design Tool using a Batch file. A typical Batch file would look like:

```
java -Xmx400m tests.simple.SimpleRegTestRunner  
C:\om\tests\reg9Grp\controlFiles\reg9Grp.prop
```

by specifying an 'r' parameter, the build will be omitted. I.e.

```
java -Xmx400m tests.simple.SimpleRegTestRunner  
C:\om\tests\reg9Grp\controlFiles\reg9Grp.prop r.
```

A typical properties file contains the following items:

- A section containing information describing the repository. This does not have to be the Repository to which you will currently be connected with in TM Builder.

```
REPOS_URL      = \\om\tests\reg9Grp\controlFiles\Reg9TChemicalFactory  
REPOS_DRIVER   = TEXT  
REPOS_PASSWORD =  
REPOS_USER     =
```

- Global settings for this run
 - 'pc_speed' is used to scale the performance of your processor if you decide to indicate that tests must run between certain durations.
 - The hdrcomment is the comment that is added to each set of results. You can specify several directories to clear by separating each path with a ';' character.

```
resultfile     = \\om\tests\reg9Grp\results.txt  
pc_speed       = 1.0  
hdrcomment     = Regression tests for the group adapter  
directorytoclear = \\om\tests\reg9Grp\tgt
```

- Default settings for each test in this run. Any one of these may be over-ridden (as in the examples below)
 - comparator is the class used to perform the comparison - or maybe one of the three pre-defined constants, \$xml, \$rdbms or \$java
 - version is the version of the project which will be used - setting this to 2, for example, will mean that all tests test as version 2.
 - min and max are performance and in milliseconds.
 - PCC should be set to yes - it indicates the use of the full comparator
 - iTrace should be set to yes - it indicates the use of instance tracking
 - PCC.Detail should be set to yes - for each instance tracked
 - PCC.LaxNumeric allows numeric strings to be compared as numbers (i.e. 3.00 will compare as true with 3.0)

```
comparator     = $xml
```



```
src.url = jdbc:oracle:thin:@LOCALHOST:1521:PEN
src.driver = oracle.jdbc.driver.OracleDriver
src.class = net.etl.tm.helper.pgroup.TMPropertyGroupAdapter
tgt.class = $xml
src.user = CF_RECIPE
src.runScripts = n
exp.runScripts = n
our= 1
min = 80
max = 2000
PCC = y
iTrace = y
PCC.Detail = y
PCC.LaxNumeric = y
PCC.AsPercent = n
SHOWTESTS = Y
```

The regression suite may also be used to test performance. In this case, typically, the comparator will be turned off and each test may be run a number of times. The following settings may be used in such cases:

```
repeat = 10
skipcompare = n
longlived = Y
SHOWTESTS = N
StdOut = C:\\resources\\customers\\RegTest\\stdout.txt
```

Most of these entries are fairly self-explanatory. The StdOut entry redirects 'standard out' to well buffered file. (Allowing 'standard out' to go to a dos prompt in the windows operating systems is very inefficient). The long-lived flag runs using a TMTransformExecutor rather than a TMTransformer - this removes the overhead of restarting the system for each test.

lines to run tests - there is no limit to the number of tests which may be run, and they may be included in any particular file. Note that the tests are run in the order in which they are specified in this file. A typical test in this case a test of the project named 'Y.calculateIngredients' will be specified as follows:

```
TEST.Y.calculateIngredients.comment = copies data from two xml sources
with easy link
TEST.Y.calculateIngredients.tgt.url =
\\om\\tests\\reg9Grp\\tgt\\calculateIngredients.xml
TEST.Y.calculateIngredients.src.Property =
\\om\\tests\\reg9Grp\\properties\\chemistry.properties
TEST.Y.calculateIngredients.exp.url =
\\om\\tests\\reg9Grp\\expected\\modifiedcalculateIngredients.xml
TEST.Y.calculateIngredients.cxp.url =
\\om\\tests\\reg9Grp\\NumberedExp\\modifiedcalculateIngredients.xml
```

If we want to add an individual test override, then we can specify it here. For example, the following line tells us that this test should complete within 1000 milliseconds (rather than the 2000 milliseconds which has been set as default for the entire Test series).

```
TEST.Y.calculateIngredients.max = 1000
```

A similar override he can be added for any of the defaults. For example, this line would ensure that version 2 of this project is run:

```
TEST.Y.calculateIngredients.version = 2
```

Note that if we had wanted to run a series of tests on the same project name using different data sets, then we would have needed to add a dollar number prefix to the tests:



```
TEST.$0.TestProject.tgt.url =
\\resources\customers\CUST\TestProject\Instances\targets\1266.xml
TEST.$0.TestProject.exp.url =
\\resources\customers\CUST\TestProject\Instances\Goodsamples\1266.xml
TEST.$0.TestProject.src.url =
\\resources\customers\CUST\TestProject\Instances\InputSamples\1266.xml
TEST.$0.TestProject.cxp.url =
\\resources\customers\CUST\TestProject\Instances\corrected\1266.xml

TEST.$1.TestProject.tgt.url =
\\resources\customers\CUST\TestProject\Instances\targets\4567.xml
TEST.$1.TestProject.exp.url =
\\resources\customers\CUST\TestProject\Instances\Goodsamples\4567.xml
TEST.$1.TestProject.src.url =
\\resources\customers\CUST\TestProject\Instances\InputSamples\4567.xml
TEST.$1.TestProject.cxp.url =
\\resources\customers\CUST\TestProject\Instances\corrected\4567.xml
```

Note that in this case, the file specified as src.url (i.e. the source files), or as (exp.url the expected outcome files), will need to exist for the test to run successfully. (If you only want to test a run, but not perform comparisons, then there is no need for an expected file).

Once the regression test system has finished, you will be presented with a dialogue showing the results of the test run. The details should be fairly self-explanatory.

If any differences have been found, then closing this dialogue will place you directly into loading an external difference file. These files have the extension .itb, and will, by default, be placed in the \transman\dtraces directory. There will be one external difference file for each transformation test that reported a difference (i.e. if all the tests succeed perfectly then no .itb files will be created). This is why we provide a separate menu option Load External Difference File... - this also allows you to open a difference file that has been created by our regression test system in a batch mode.

Once the External Difference File has been loaded, the message area in the Design Tool (which normally contains compile warnings or execution code plan statements) will turn yellow and will contain lines describing the differences between expected and the actual created targets. Note that you can double-click each of these statements to navigate automatically to the correct line in the appropriate transform in the SML mapping pane.

In addition, you can right-click any of the selected differences. If you do this, three separate dialogues will be opened. The first is a complete description of the difference itself and the execution state of the transform system when the difference was created. In future versions of this feature it will be possible to click into the execution stack presented in this dialogue and move to the correct execution point.

The second is a window into the expected target which has been modified by our comparator system to contain object ID's. Within this window, the system will have moved to the point in the document where the difference has taken place and will have highlighted the relevant _oid. Note, incidentally, that this modified document will have assigned _oids to every node. Where these match with the actual target, they will be numbers. Where no such match has been found, then they will contain the text 'ERC!' to enable rapid identification of missing nodes.

Note that a twin monitor system is highly recommended for use with the integrated instance tracking. All three windows are automatically cleared if you right-click on another difference point, or build or close the project.



Future developments

A third option will be added to the Tools | Regression Suite. This will be a step by step wizard to guide you through the construction of a suitable regression test property file for your transform.

A generic version of this tool will become available for database, java and generic use.

Further Information

Website : www.etsolutions.com
Email : info@etsolutions.com
Telephone : +44 (0) 1912 894040