MSS Synchronizer High Level Overview

Overview

MSS is a specialist in the Enterprise IT legacy modernization space. With over 50 highly complex modernization projects successfully under our belt, we have learnt from each of our projects and developed our technology accordingly.

Quite often our customers wish to modernize their applications and systems but for a number of reasons are simply not able to do this with a “big bang” approach. In response to customer demand, our Research and Development team has created The MSS Synchronizer. Using the Synchronizer, a customer can perform a rewrite and gain significant project and operational benefits:

The MSS Synchronizer

- Allows for progressive deployment of the new application while continuing to run the business with the old application. (Both will run side by side in production until the new one is proven ready to take over completely)
- Ensures that all non-obsolete legacy data elements have been mapped to the new database
- Ensures that all non-obsolete legacy transactions have an equivalent function in the new application
- Ensures that all non-obsolete program logic has been carried forward into the new application

In other words, the MSS Synchronizer ensures success, or from the negative perspective, insures against errors and omissions that lead to cost overruns, delivery overruns, and lost functionality.

Technical Overview

The MSS Synchronizer is a tool that, firstly, allows you to take any database and create a clone of it on a remote system. We populate the clone database and keep it updated using software that traps any updates on the mainframe and sending information across a link to update the remote database. In fact we create 2 databases on the target, a ‘mirror’ database that is a simple clone of the mainframe DB, and a target database in the final format. This allows fine control of the comparison process as well as offloading the data mapping overhead from the mainframe.

The way we trap database updates on the mainframe is to use our pre-compiler technology to inject code into programs intercepting database update statements and sending the changes to the messaging infrastructure. This allows us to copy the data across giving more control than simply copying the updates using the database audit. We can use the same mechanism to capture the events that trigger those updates, in other words incoming messages or reads from disk files that drive batch programs.

Once we have the data in the remote database we can then use it to allow us to (A) keep the database in sync, and (B) (in fact the more important function) to check on how the remote software is working. The underlying premise is that these mainframe programs are being re-written to work on a dissimilar architecture but that the re-write is using the same basic data. The data will be in a different (potentially radically different) format so we have service routines that allow us to transform the mainframe data into the final target data format.

We can phase in adoption of the re-written application and we keep track of which functions run in the old environment and which in the new. The re-written transactions can run on the remote system at the same time as others still run on the mainframe and the software will allow a mixture of mainframe and remote transactions controlled dynamically. We provide an administrative facility to enable this as well as report on activity and performance of the system.

The key feature is that, when remote transactions are initiated, they will be intercepted and equivalent transactions run on the mainframe. This enables the results of the mainframe transaction to be compared against the remote target system transaction. Thus any bugs in the rewritten software can be highlighted immediately and, not only can they be found and highlighted, but also debugging information provided – it is readily available in the form of "what should have happened".

The mainframe software is likely to have been in existence for many years and therefore can be taken as the master in this relationship so the presumption is that the rewritten transaction, if the results are different from the mainframe transaction, is at fault. Any errors will be reported and can be debugged immediately.

This happens in real time so that, as soon as a fault is found with a remote program, we can switch to only using the mainframe program for that particular transaction and/or flag the database data (and any related information) as "in quarantine" and not to be updated on the remote system until the problem is resolved.

The whole system forms a real time check on a rewrite project by implementing effectively a continuous parallel run while providing a very flexible means of phasing in rewritten transactions that replace mainframe transactions. This is a way of keeping control of large rewrite projects which would otherwise have no easy way of being tested which is, we believe, the source of most rewrite project failures, of which there are a high number.

Top Four Customer Benefits

1. The real-time check on the validity on the new system as compared to the old system.
2. The availability of data in real time so that not only the mirror data (the clone of the mainframe data) is updated but the data is updated in the target database as if the rewritten transactions had already been in place.
3. The ability to capture the message data and other debugging data to be able to debug any program faults immediately.
4. A gradual implementation of the rewritten applications as they become available.