TX²

Technical Solution:
Parsing of External Input when using Message Broker

Since the acquisition of Crossworlds, and in particular the integration of the Adapters into the WBI architecture, a new decision has had to be made by architects and designers: "where to parse". In the case of dedicated application adapters such as SAP and Siebel, the decision is pretty clear cut. The parsing of the physical format is done in the adapter, which generates XML business objects for use in the rest of the system. Also, when using ICS the decision is mandated by the fact that ICS must receive business objects. However, historically Message Broker designers have typically used the TDS parser to parse file input (where the contents of the file have been placed on a queue as a blob).

The JText adapter, when used in conjunction with Message Broker, now offers an alternative to this approach, with the file content being parsed in the adapter, then transferred to Message Broker as an XML Business Object. Configuring the Jtext adapter to parse the file is done using Business Object schemas, Managed Objects and DataHandlers that parse the file content. Data Handlers are supplied for separated values, fixed-width fields, and tagged (name, value pairs).

A major worry of the traditional Message Broker designer is performance of the adapter approach. It is intuitive that parsing once using TDS will be much faster than parsing into XML, and then parsing the XML in Message Broker (not to mention the performance impact of the XML expansion factor – typically 7 or 8 times the size). However, the actual results are indeed counter-intuitive.

In the case where the data is already in XML format when it reaches Message Broker there is a further decision on parsers: parse the XML based on a scheme stored in the MRM, or dynamically parse the XML (without a schema). For "safety" reasons, designers often like the security of parsing against an MRM schema so that errors in the data can be detected. However, where the XML has been generated by another part of the same system (application / infrastructure) this level of checking seems unnecessary, and the performance advantages of using the dynamic XML parser are well worth considering.

The tests below were the result of investigations of performance issues processing "medium size" files (about 1MB), containing multiple records (about 3,000) in a multi-level structure. For each transaction in the file, the top level object had 2 child objects, plus an array of objects, and within each array element was a further structure containing an array. The fields within the records were comma separated. The system in question had to process about 1,000 files per night, so a processing time of several seconds for each file was significant.

Initial investigations into any framework overhead, or concerns about time spent building the output message, quickly demonstrated that most of the processing time was actually being spent parsing the input messages. Several tests of the flow were then run using different input formats: the original Comma Separated Values format, and XML (Business Object) format.

Test Results Flow 1 (simple processing of each transaction):
Test file (message), about 3,200 records - 1.2MB if in csv format, 7.7MB when in XML format:

TDS MRM - 9.0 seconds per message
XML MRM - 7.6 seconds per message
Dynamic XML - 1.9 seconds per message

Test Results Flow 2 (input message generates large number of output messages):
Test file (message), about 3,200 records - 1.2MB if in csv format, 7.7MB when in XML format:
Each message generates 1,925 output messages (each output message about 1.5k)

TDS MRM - 19.0 seconds per message (100 output messages/sec)
XML MRM - 17.6 seconds per message
Dynamic XML - 12.0 seconds per message (160 output messages/sec)

We conclude from these results (the absolute time difference between the parsing types remains constant) that the parsing itself is dramatically lower using dynamic XML parsing of the XML inbound data.

In the test case above, the file format included a “self describing structure” with field contents defined in “REM” lines within the file. This meant that a custom parser had to be written. The performance of this parser (written in java, but with optimal performance in mind) was very much faster than the difference in time between TDS and XML parsing. In the test cases above, the java custom parser averaged less than 1 second to parse the file. Thus, a one second conversion from CSV to XML outside message broker saved 7 seconds of processing within Message Broker.

Of course, the transactional scope of the records in the file needs to be taken into account. Typically however, we would postulate that normally each record is a separate transaction, or a group of a small number of records is one transaction. Where files have large numbers of records, it is unlikely the file represents one transaction. Parsing the file into separate business objects that are each one transaction makes a lot of sense, and can avoid problems with large numbers of records all being processed within one transaction.

The adapters DataHandler architecture is very flexible, making it easy to modify the supplied data handlers or write new ones. However, they are themselves not without problems. Any changes involve writing java code, and the java classes then need to be deployed and managed on systems where the adapters are running; this is a manual deployment task with no tools to assist in managing deployed code. The supplied data handlers (for Jtext are also rather limited, and most real projects require customisation of the DataHandler).

The overall conclusion of our tests ? External parsing to separate business objects, together with dynamic parsing XML input to the Message Broker reduced the overall nightly processing time from over 11 hours to under 5 hours.

