PilotFish TECHNOLOGY®

CASE STUDY

Implementing High Performance, High Volume Interfaces with the XCS eiConsole

Large Canadian Life Insurer Processes Millions of Policies in Minutes

OVERVIEW:

Canadian Life Insurance carriers are increasingly implementing the CLEIDIS/CITS standards. These XML standards offer a refinement of the broader ACORD XML TXLife transaction set. Among the messages defined by CLEIDIS, in-force policy transmittal is one of the most commonly discussed – and one of the most challenging.

In this message, in-force policy information is transmitted from the carrier to their agents and MGAs. It applies to Life, Critical Illness, Long Term Care, and Disability lines of business. The transaction is very descriptive, including information on contract terms and values, as well as detail on the insured, beneficiary, owner, payor, broker, and servicing MGA.

For a large carrier with millions of in-force policies, the amount of raw data required to generate this transaction can be enormous. This size of the final message is even larger, with the verbose nature of both XML and the ACORD schema often tripling the length of the output.

The size of the transaction is only one aspect of this demanding effort. Further complicating matters, a large carrier will work with dozens of distributors, each of which must receive a different subset of policies – those that they have a part in managing. A given policy may be transmitted to one or many distributors, with the endpoints determined on-the-fly based on dynamic routing criteria.

For a very large carrier in Canada, these considerations warranted an intense product evaluation and planning period. Over the course of more than two years, the carrier considered their options. After poring over hundreds if not thousands of pages of formal proposals, and several extensive proof-of-concept exercises, the carrier selected PilotFish Technology as their implementation partner.

On a nightly basis, the carrier provides a fixed-width file extract containing information about all of the policies in the company's book of business. This extract contains a number of data records for each policy, including detail information on associated Parties and Coverages. The number of policies in this file is between 1 and 2 million, with a file size in the range between 10 and 20 GB.

Supplementing this policy information, the carrier also maintains a "hierarchy file". This file, actually represented in a relational database, contains information about the relationships between agents and various distribution channels, as well as details about those individuals and organizations. This database contains roughly 50-300 distributors and their associated agents, with the total size of approximately 15-30 MB.

When a set of policy-related records is processed, the distribution hierarchy information must be used to both enrich the data stream with distributor and agent level information as well as to direct the policy information to the correct outbound extract. Each outbound extract has not only a unique set of policies, but a unique FTP destination and encryption protocol and/or PGP key.

Each policy, and related parties, must be translated into a CITS-compliant ACORD XML representation and aggregated into the associated distributor-specific file. The final file for each distributor must represent an ACORD and CITS-compliant TXLife transaction. The file is then named appropriately per the CITS standard, compressed via the ZIP algorithm, and staged for FTP transmission to the various distributors.

The production of 50-300 outbound CITS XML files, totaling up to 30 GB of data, from the in-force Policy extract and associated hierarchy file must be handled within a 6-7 hour batch window.

INTERFACE IMPLEMENTATION:

In early 2009, PilotFish worked with the carrier to implement the initial, prototypical version of the CITS In-Force interface. This exercise demonstrated how the capabilities of the XCS eiConsole are be used to implement an interface which would generate CITS-compliant output from smaller samples of the aforementioned extracts.

In this first phase, PilotFish used the XCS File Specification Editor – a core component of the XCS eiConsole – to convert the policy administration system extract into a generic XML representation. Once the policy extract is converted into XML, it is forked into discrete units of work. These units of work, each a set of XML records, are then transformed into ACORD/CITS XML.

The transformation from generic XML to CITS XML is implemented using XSLT generated by the XCS Data Mapper. The XCS Data Mapper offers drag-and-drop mapping between any two data formats. The insurance-specific features of the Data Mapper, including the management of ACORD/CITS "type codes" and "Relation" element types, enhanced much-needed developer productivity. The color-coded, graphical representation of the transaction also provided a convenient vehicle for the joint review and discussion what at times included complex mapping logic.

Once the business analysis was complete, and the logical mapping rules from the policy administration system to the CITS transaction were well-understood, the development of the transformation required approximately a person-week of development effort.

After the successful execution of the first phase of the project, confidence in the data transformation capabilities ran high. However, all parties remained painfully aware of the elephant in the room – a daunting performance requirement that would push the boundaries of XML transaction processing.

A series of benchmarks were conducted against the interface before any optimization was performed. Based on these tests, PilotFish concluded that the desired performance may only be guaranteed through the implementation of a clustered XCS eiPlatform environment.

In the proposed architecture, one XCS eiPlatform instance serves as the Master. The Master accepts the initial input file and converts it into the generic XML representation. As the policies are converted into XML, they are provided to a set of clustered Worker XCS eiPlatform instances. Worker instances perform the computational-intense transformation over a given set of policies, and then return them to the Master for re-aggregation.

Subsequent to the initial design of the production implementation, PilotFish unveiled NTM (New Threading Model) Acceleration for the XCS eiConsole and XCS eiPlatform products. This enhancement marked the first time that the fundamental mechanics of the integration platform had been re-engineered. This enhancement reduces potential synchronization bottlenecks and improves the ability of the XCS eiPlatform to take advantage of a server's available resources.

Benchmarking of other, PilotFish NTM-accelerated interfaces indicated that the CITS interface may not require clustering at all. Instead, it would take advantage of this improved framework and allow for gigabytes of throughput on a single server.

In the prototype, the interface was designed as three XCS eiConsole routes. The first route was responsible for parsing the Hierarchy file and generated a fast, in-memory representation for subsequent reference. The second route was responsible for parsing and transforming individual policies from the administration system extract into individual CITS fragments. Distinct data transformations were implemented for generating Party, Holding, and

Relation elements of the CITS feed. The third route was responsible for combining the file-based fragments and generating the distributor-specific CITS XML files.

In the production implementation, the interface was simplified to include only a single route. The data flow is as follows:

1. On a periodic basis, the Listener component polls Policy extract files.
2. The Hierarchy database is loaded in-memory and converted to a set of Java value objects (POJOS), indexed by Agent and Distributor identifiers.
3. The Policy File is processed through an iterative file read routing, wherein:

    a. Each line of the file is read into memory

    b. The line of the file is converted into a corresponding XML representation

    c. When all of the lines for a particular policy have been read and converted to XML:

        i. The agent identifier is found and used to lookup the associated distribution channel in the Hierarchy index.

        ii. All relevant agent and distributor data is appended to the XML stream.

        iii. The complete XML fragment is added to a memory-based cache of Policy-level fragments organized by Distribution Channel.
4. Each Distribution Channel cache is populated with a growing set of Policy XML documents. Once this cache reaches a configured capacity (e.g., 25 policies), the XML documents are combined into a single XCS eiPlatform transaction and fired. A transaction attribute is set to store the identifier of the associated distribution channel. A counter of sent policies is incremented by the batch size.
5. Each transaction, consisting of a batch of Policies and related data in an XML form, arrives in the Transform component. Each is transformed via XSLT into an ACORD and CITS-compliant XML fragment containing Party, Relation, and Holding data.
6. As CITS XML fragments, each batch arrives in the Transport component. When this component receives a fragment:

    a. The distribution channel identifier is acquired from a Transaction Attribute

    b. An index is searched by distribution channel identifier to locate the appropriate CITS file stream.

        i. If such an entry is not found, a new file stream is created, including the generation of the ACORD TXLife / CITS header.

    c. The CITS document fragment is appended to the appropriate file stream.

    d. A count of received policies is updated.
7. When the Listener has completed processing all lines in the Policy file:

    a. All remaining caches are flushed into the system.

    b. Final counts are updated.

    c. Input files are deleted.
8. When the Transport identifies that the Listener has completed its work, and that the "received" counter is equal to the "sent" counter:

a. The ACORD TXLife / CITS footer is added to each file stream.

b. Files are closed.

c. Encryption is performed, typically via PGP

d. ZIP versions of the files are created.

9. The process completes.

PERFORMANCE TEST PLATFORM AND RESULTS:

The development of this interface was conducted largely on a developer-class laptop running Windows Vista. The specific configuration that was used for development is as follows:

- Dell XPS M1730 (Laptop)
- Intel Core 2 Extreme CPU X7900 @ 2.8 GHz
- 3070 MB RAM
- Windows Vista Ultimate Edition, 32-bit
- Sun Java 1.6

In this environment, the full sample Policy (~10 GB) and Hierarchy file (~29 MB) were processed in approximately 2 hours.

PilotFish also secured access to a server environment representative of the recommended deployment platform for the production implementation. This IBM 3650-M2 server had the following characteristics:

- IBM eServer xSeries 3650-M2
- 2x Quad Core Xeon X5570 @ 2.93GHz
- 32 GB RAM
- 4 x 73 GB 10K SAS HDD (RAID 10)
- Windows Server 2003, 64-bit
- Sun Java 1.6

In the representative server environment, the full file of 1,000,000 policies was processed in approximately 20 minutes. Tests of smaller versions of the file indicate that performance is O(N) (linear).

Given this, the observed rate of processing 1,000 polices / second indicates a capacity to handle more than 25,000,000 policies in the 6-7 batch window – far more than the number of in-force policies of even the very largest North American carriers.

PERFORMANCE TUNING TECHNIQUES

In order to achieve the outstanding performance referenced in the previous section, a number of performance tuning techniques were employed.

NTM Acceleration

The reimplementation of the interface on the NTM-accelerated version of the XCS eiPlatform and XCS eiConsole provided an immediate and marked performance boost.

NTM permits more fine-grained control over transaction execution threads that was previously possible in older versions of the XCS eiPlatform, where thread pool and queue sizes were maintained globally. To achieve maximum performance on this interface, the "per route" threading model was selected over the "per stage" and "global" options. Each route was configured with an unlimited transaction queue size and a thread pool with a maximum number of threads equivalent to the number of cores available in the operating environment.

XSLT Caching

In the default configuration, XSLT transformations are re-initialized from disk prior to transformation execution. While this permits for easy modifications to transformation rules "on the fly", in does incur significant performance overhead. This overhead is avoided by enabling the "Cache XSLT" option of the XSLT transformation module. When this option is enabled, each thread maintains its own, cached version of the Transformer used to execute the XSLT transformation against the transaction data. This eliminates the recurring overhead of parsing the XSLT transformation and eliminates the need to synchronize access to the transformer.

Also, in the default configuration, the interpreted Xalan transformation engine is used to parse and execute XSLT. Through various trials, it was discovered that Compiled Xalan offered significant performance gains over the default option or the Saxon XSLT engine.

Batched XSLT Transformations

The distributor-specific "caches" for Policy XML data are used to reduce the total number of distinct transactions that must be managed within the system, and to manage the size of the XML data being transformed to ACORD / CITS by any given XSLT execution. This cache capacity was made configurable in the interface's Listener modules. Benchmarking indicates that a capacity of 20-25 policies is optimal.

I/O Buffering

The interface has two significant potential I/O bottlenecks – reading the inbound Policy Extract and writing to each of the (potentially many) distributor-specific CITS XML files. In order to avoid a slow-down in the single-threaded Listener component, or resource-contention in the Transport, great attention was paid to the efficiencies that can be implemented in these areas.

When reading the inbound file, performance generally improves as the size of the buffer is increased. As a general rule of thumb, and default for the interface, 1/20th of the available memory in the JVM is allocated as a file read buffer. In practice, buffering 500 MB – 1 GB provided good results in both the laptop and server environments.

When writing the outbound files, similar file write buffers are configured. The larger the available buffer, the faster an individual CITS fragment can be appended to the appropriate file. Since multiple concurrent "file write" operations may be pending for the same distributor-specific file, buffering also helps to alleviate the time that is spent blocked and waiting on exclusive access to a particular file.

Benchmarks indicate that performance generally continues to improve as these buffers grow, so long as the buffer doesn't exceed the final CITS XML file size. However, since there is a buffer for each distributor, the memory footprint expands geometrically as the size or number of distributors is increased. For instance, a 10 MB file write buffer will require 300 MB of available heap space for 30 distribution channels. For laptop-based benchmarks, a 1 MB buffer was used. In the server environment, a 10 MB buffer was configured.

While these two buffers are the only configurable buffering points, the associated Listener and Transport were tuned such that all buffers and memory allocations are "right-sized" for the expected data.

Adaptive Throttling

The Listener component was instrumented with an "adaptive throttling" mechanism. This mechanism uses the delta between the current count of sent / completed policies to determine the effective backlog in the system. When the backlog exceeds a configured number of policies, the listener thread will yield until the system is again operating within the desired parameters. This throttling provides a more steady flow of transactions through the system. Allowing a backlog of 250 to 500 policies has proven optimal in most tests, regardless of the test platform.

Reduced Logging

Logging was reduced from the "Debug" level to the "Info" level. This eliminates most instances of threads blocking on Log4J debugging calls.

On-the-Fly XML Conversion

Rather than relying on the typical File Specification Engine to convert the Policy extract into a similar XML representation, this conversion was accomplished as the data was processed in the listener.

Thread Priority

When running most performance tests, the priority of the Windows process running the XCS eiConsole / eiPlatform was elevated in Task Manager from "Normal" to "High" or "Realtime". The performance impact of this change was much more noticeable in the 2-core laptop environment than the 8-core server environment. In the 8-core environment, performance was more typically gated by disk than CPU.

SUMMARY

The project was an overwhelming success. Very real performance concerns, not only held by the client but shared by some on the PilotFish staff, were eliminated. The final performance of the production interface bests the documented requirements by approximately 1000%, and does so without any requirement to cluster XCS eiPlatform servers.

In under an hour, the carrier can generate a set of encrypted, compressed, and CITS-certified messages for transmission to their many distribution partners.