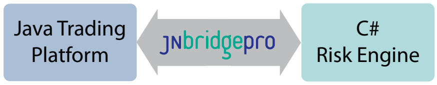
Case Study

JNbridge®

SPANNING JAVA & .NET

Major Financial Institution Bridges C# and Java to Assess Risk More Efficiently, Reap Significant Cost Savings



Business Challenge

Company's C#-based financial risk engines were never fully integrated with the rest of its Java-based backend infrastructure, resulting in time-consuming and expensive processing bottlenecks for analysts assessing risk in financial transactions.

Solution

JNBridgePro presents a fast, reliable and flexible alternative to rewriting applications, saving exponential time and costs.

> "JNBridgePro just worked. Actual time spent in computational processing is down from two hours to 12 minutes. The human element is greatly reduced too: The process requires only one analyst for 30 minutes, as opposed to three analysts spending over two-and-a-half hours each."
>
> *Developer at major financial institutution*

Situation

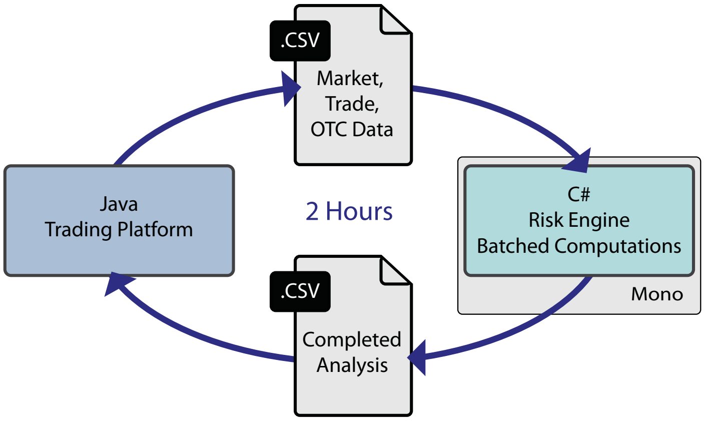
As any financial services company knows, assessing risk is key to a healthy and successful financial transaction. That's why this major financial institution pays very close attention to assessing volatility in financials using its risk engines.

"Our job is to assess the risk of the people doing the trade," said a developer with the company. "It's incredibly important that this is done correctly and accurately because we're on the line for anything that goes awry."

Challenge

Unfortunately, the company's risk engines were written in C#, while 85 percent to 90 percent of the company programmed in Java. (In fact, the actual trading infrastructure was in Java.) So, the process of exposing the risk engines to all the Java developers began with the Java side constructing a .csv file that contained all of the company's market data, trade data and OTC data, then passing the .csv file to a C# executable, which then ran on a batch timer and performed all the computations. The output was another .csv file that then was passed back to the Java side. The time spent in computational processing took two hours to complete. Added to this was the human element: Three analysts spent two-and-a-half hours each in the process. And if one of the OTC trades was off by a cent? "Well, we'd have to go through the entire process again," said the developer. "It was pretty terrible."

It's important to note here that, despite the size of the data being processed — around 10 megs of order data (about 50 million trades) a day — the data wasn't the bottleneck. Rather, the volume of computations was the culprit.



Before

Solution

Since its core trading platform is Java-based, the company needed to find an efficient way — for both risk-processing time and developer time — to access its C#-based risk engine. Whichever solution was chosen must not only deliver efficiency, but also must support

Financial Institution

About the Customer:

This company is a major financial institution that deals in the energy markets. Because its JNBridge implementation is very strategic, the company has chosen to remain anonymous.

SPANNING JAVA & .NET

About JNBridge:

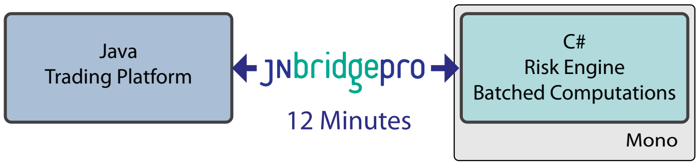
JNBridge has made seamless and cost-effective Java and .NET interoperability a reality since 2001. The company's award-winning bridging technologies, JNBridgePro and JMS Adapters for .NET and for BizTalk, enable rich cross-platform communications anywhere, so businesses can focus on innovation, not on solving interoperability issues. More than 600 organizations around the world rely on JNBridge, making it the most popular bridging solution in the industry. To learn more, please visit www.jnbridge.com.

JNBridge, LLC
info@jnbridge.com
jnbridge.com

Mono: The company's C# engine isn't running on a standard Windows box; rather, it replaced .NET with Mono and is running on Linux.

The company began searching for a solution that could help reduce processing time. "We took a look at three open-source products, went through their demos and found issues in all of them. One had fundamental mistakes in the execution of the code, another took a significantly longer time to call from Java, and the third simply didn't work," said the developer. "Since they were open-source products, we could have investigated further, but we were short on time."

The company found that JNBridgePro solved its interoperability issues efficiently and effectively. Now, Java collects all the risk data, provides it to the C# engine through the API that JNBridge exposes, then consumes the corresponding risk profiles from the call's result.



After

"JNBridgePro just worked," he concluded. "Actual time spent in computational processing is down from two hours to 12 minutes. The human element is greatly reduced too: The process requires only one analyst for 30 minutes, as opposed to three analysts spending over two-and-a-half hours each."

The company also now has the ability to do incremental processing for differentials. "Differentials are very difficult for humans running the inputs to perform, whereas providing a data-aggregating cache in front of the JNBridge-facilitated risk call allowed us to programmatically apply risk curves to the appropriate differentials," said the developer. "Without the capability of calling the C# code programmatically from the existing risk aggregators, this would have been impossible to perform without incurring increased operational risk."

Aside from the time savings and operational efficiency, the company estimates a hefty savings in infrastructure investment as well. By not converting its C# engine over to Java, the company estimates it saved approximately $100,000 in development costs, after licensing is taken into account.

With JNBridgePro's benefits clearly proven, the company now has plans to migrate all its processes to JNBridge over a two-year timeframe. "We have so many different processes — one C# processor for one type of risk and another for a different type of risk, and so on and so forth — that migration will take some time," he said. "But the time and cost benefits afforded by JNBridgePro are worth it."

Benefits

- *Dramatically reduces risk assessment process from approximately two hours to 12 minutes*
- *Enables C# and Java programs to interoperate seamlessly*
- *Estimated development cost savings of $100,000 by not converting C# engine to Java*