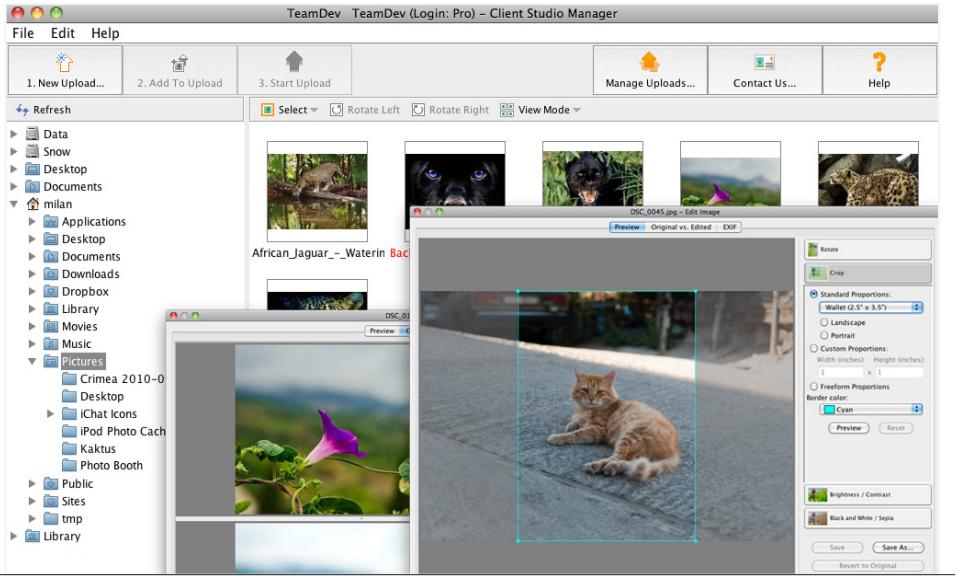
TeamDev

JNIWrapper Case Study

JNIWrapper provided enhanced interoperability between client and server sides of DigiPixArt's online digital photo management application.

Industry
Digital photography
Online photo printing service

Technology
JNIWrapper



Summary

DigiPixArt, specializing in online digital photo management and printing service, needed a more efficient and scalable way to handle user requests in the application providing the service. The solution was to make server-side mage processing distributed. To provide it, the company needed to replace the existing command line interface between the client and server with a more efficient bridging solution.

Challenge

Thousands of people are using online digital photo catalogs to share their photos with family, friends, and the world. Publishing photos on the Web with their subsequent editing requires stability and high performance of the service under a heavy load. DigiPixArt, a USbased provider of an online digital photography management and photo printing service, faced the problem in 2006. Using the service from DigiPixArt, professional photographers and organizations could upload photos via a provided Web interface and perform desired image editing operations, such as resizing, rotation, cropping, watermarking, etc. However, when there were thousands of photos to process, the Web application providing this service could not handle the increased load. DigiPixArt needed to redesign the internal architecture of the application to make it distributed in order to share data processing tasks among several servers instead of one.

For complex image manipulation, the Web application used opensource ImageMagick software with which it communicated via ImageMagick command line utilities. The future redesign required replacement of the console interface with an intermediate layer that would be more natural for the Java environment, in particular, be able to handle image processing errors, and provide effective communication between ImageMagick, written in the C programming language, and the Web application,

written in the Java language. Such solution was a Java wrapper for ImageMagick. DigiPixArt turned to TeamDev for stance in implementing an ImageMagick wrapper.

DigiPixArt required support of the ImageMagick wrapper for Mac and Windows platforms. In addition, both Mac and Windows implementations of the ImageMagick wrapper were to be built with the view of being later embedded in DigiPixArt's in-house desktop applications. High performance of the solution was also important since processing of a large number of large-size photos via an online service is always resource-intensive. Efficiency and correctness of bridging were the key requirements in this project as these were critical issues for correct data transfer and conversion. Primarily Rosetta tried several solutions before switching to ComfyJ (particularly J-Integra). But in the end none of these solutions satisfied the customer. Finally they tried ComfyJ—our bidirectional Java-COM bridge or working with COM technologies from Java programs—and were impressed with the scope of functionality they got with ComfyJ.

Solution

To implement the ImageMagick wrapper, TeamDev had two options to use: JNIWrapper, an in-house technology for integration of native code with Java, or JMagick, an open-source API for ImageMagick. After a thorough analysis of the two software alternatives, TeamDev opted for JNIWrapper.

There were several reasons for the choice. Though JMagick is a Java interface for ImageMagick, it is implemented in the form of Java Native Interface (JNI) and is very hard to maintain, mainly because it requires writing native code using time-consuming JNI programming techniques. Moreover, out-of-the-box, JMagick could not be used with an existing ImageMagick embedded in the application. JNIWrapper, on the other hand, obviates the need for a Java developer to write native code and, unlike JNI, supports complex native features on the Java side. So with all pros and cons considered, it was much faster and more reliable to implement the required functionality using JNIWrapper than to maintain JMagick.

During the investigation stage, TeamDev searched the ImageMagick source code documentation for a required set of functions that were to be supported in the ImageMagick wrapper. After the functions were identified, respective Java wrapper classes were written for them. The final result was implementation of the wrapper interface for calling native functions of ImageMagick.

Results

After being introduced, the ImageMagick wrapper eliminated the need in the command line interface and enhanced interoperability between the Web application and ImageMagick. Portability of the solution also proved to be high. When DigiPixArt wanted to use ImageMagick on the Linux platform, the migration took only two days, including testing. The ImageMagick wrapper was later successfully integrated into another DigiPixArt's applications, both desktop and server-side ones. When utilized inside of desktop applications, the ImageMagick wrapper showed good performance on all the supported platforms.

The ImageMagick wrapper is also highly extensible. When there is a need to add a new function to the wrapper, it takes very little time to do so, because a developer just needs to write necessary code in the Java language bypassing complex JNI programming.

Contact Information

TeamDev Ltd.

47 Nauky Ave.
Kharkiv 61103,
Ukraine

+1 425 223-3079
+380 57 766-4330
FAX +380 57 766-4343

sales@teamdev.com
www.teamdev.com