2010-01-31

Project specification

BEng Essay
KTH
Henrik Feldt 2010

Title	.Net implementation of File Transactions

Background

The Castle Project is an open source project aimed at making enterprise application development easier. As such transactions are important, but file transactions have been lacking for a long time. The goal of a transactional file system is to satisfy the ACID properties, i.e. transactions should be **A**tomic in terms of committing/rolling back, give a **C**onsistent view of the file system, be **I**solated from each other and **D**urable after commit [1]. Microsoft has implemented NTFS transactions called TxF, but these do not have bindings for .Net, accessible to programmers in an easy-to-use manner. What would it take to access them?

The common language runtime (.Net/Mono/rotor) can use the API from kernel32.dll by means of P/Invoke – a mechanism through which managed C# can invoke unmanaged code – in this case a C-flavoured interface. I have already implemented most of the transaction API as a class FileTransaction which interops with System.Transactions and hence is enlisted in any current transaction scope.

Using inversion of control[1] it's possible to wrap interceptors around instances of services. The interceptors start and stop transactions as the wrapped objects' methods are called – rolling the transaction back if an exception is thrown or remains unhandled within the intercepted method body. A developer programming operations acting on the file system would be interested in knowing the status/structure or state of files and directories within a given transaction in which work is being done; as such all invocations of IO-APIs would have to "know" what transaction is being used. As .Net is not aware of file transactions, the IO interfaces need to be implemented as aware of transactions. The grunt-work of making IO implementations aware of file transactions has been done.

Taking the above into consideration, the project aims to complete the implementation, write accurate documentation and analyze the implementation in terms of software design choices and performance.

[1] Inversion of control is a collection technology for miscellaneous object oriented practices. For example, a competent IoC engine can facilitate disposal of object graphs in a manner consistent to how the programmer needs the graph to behave (singleton/one-instance-only, thread-static/per-thread, transient/one instance per call, per-web request etc), how to instantiate objects with its dependencies (which are instances of objects), weaving together methods in a proxy – programming to interface, interception of method calls.
http://jonathan-oliver.blogspot.com/2009/03/dddd-eric-evans-interviews-greg-young.html Accessed 2010-02-10
http://www.infoq.com/interviews/Architecture-Eric-Evans-Interviews-Greg-Young Accessed 2010-02-10

2010-01-31

Abstract

The project aims to complete the implementation, write accurate documentation and analyze the implementation in terms of software design choices and performance. Documentation will be written online at using.castleproject.org with the report forming a complement; a project post mortem of sorts. The project will aim to investigate state-of-the-art in the area of file transactions by relating these new developments to the current C# implementation. Performance analysis will be similar to stress-testing[2] and load-testing[3] [4] but aim to create developer usage guidelines and knowledge about costs associated with using the framework. Analysis could also facilitate code improvements. In the end a released version of the file transaction framework should be made available together with its online documentation.

Plan

The order of affairs is something akin to the below:

1) Finish writing any lingering method implementations for namespaces: Castle.Services.Transaction[5], Castle.Services.TransactionMangagement[6], Castle.Facilities.AutomaticTransactionFacility[7].

2) Test the framework on non transactional systems (Mono, Windows XP)

3) Write public documentation of the system that documents how to use the framework classes

4) Send press releases about the finished framework

5) Write an essay detailing the design decisions, choke-points and a post-mortem of the project of releasing the component.

Theme/Keys Transactional File System, Kernel, Distributed Transactions, .Net, C#, Inversion of Control, Software Architecture

References

[1]: Principles of transaction-oriented database recovery
Theo Harder, Andreas Reuter
ACM Computing Surveys (CSUR)
Volume 15 , Issue 4 (December 1983)
Pages: 287 – 317
Year of Publication: 1983

http://social.msdn.microsoft.com/forums/en-US/windowstransactionsprogramming
http://castleproject.org/castle/projects.html
http://msdn.microsoft.com/en-us/magazine/cc163388.aspx
http://higherlogics.blogspot.com/2009/11/easy-file-system-path-manipulation-in-c.html
http://www.links.org/files/capabilities.pdf
http://en.wikipedia.org/wiki/P/Invoke
http://groups.google.com/group/castle-project-devel

Supervisor Mads Dam

[2] Spike drive-usage/CPU-usage by pushing as much load as the system can take onto it, to test against deadlocks, livelock, memory leakages, race-conditions and other programming flaws.

[3] Test the system under load for a prolonged period (usually hours) to check against memory leakages and stability-problems, but leave some “breathing room”, i.e. don’t push the system to its limits.

[4] http://www.youtube.com/watch?v=XQ5NvCpQnqg

[5] Here are implementations

[6] This is for transaction management; threads, contexts etc

[7] This is for inversion of control