QL Integration into Scala and Excel

Martin Dietrich
E.ON Global Commodities

• Over 1000 professionals, active on over 20 exchanges in more than 40 countries

• Over 1000 counterparties in more than 50 countries

• 850,000 trades in 2011

• Market energy, manage commodity risk and provide asset optimization services for the E.ON Group and its third party customers

• Main trading activities: Power, Gas, Emissions, Oil, Coal, Storage
  • Spot, physical forward, options, futures, spread, swaps
  • Swaps, virtual storage, swing gas
  • Physical coal, own fleet of vessels
What makes it special?

- Asset-backed trading
- Permanent obligation to mark and hedge E.ON‘s asset portfolio
- Physical delivery with hundreds of physical constraints in fuel supply and power generation
- Limited liquidity with a significant market share in physical positions
- Simple products like options and forwards
- Complex and structured products like VPP and Swing
Example: Swing Contract

- Periodic delivery within a given delivery period at a given strike price
- Buyer has the right to exercise nomination at short notice (day ahead)
- Min and max number of exercises
- Min and max volume per sub period (month)
- Min and max volume for the whole period (gas year)
- Coupled American style options – flexible but limited exercise
- Complex optimization problems solved by dynamic or linear programming
Why QuantLib

- Demand in financial and numerical open source library
- Advanced, mature and tested
- Not reimplementing pricing engines, volatility modelling, Brownian bridge and many more
Why not exclusively QuantLib

- Commodity markets are different
- Additional financial engineering requirements
- Want to leverage functional programming languages
- Access identical logic and underlying market data regardless of client
- Big data, half-hourly profiles or forward curves
- Interacting with pricing engines from ETRM, Excel or just a simple browser
- Access the power and performance of a grid from the desktop
- Agile development
Technology Stack

TeamCity → Excel → play → QuantLib

LexiFi

Jenkins
Development Dependencies

WebSocket

JSON

eet.apps.quantlib-swig

OS/Arch dependent dll

WebSocket

JSON

eet.apps.quantlib-swig

OS/Arch dependent dll
Why Excel-DNA

- Integrating .Net into Excel
- Packaging tool for script files and assemblies to generate a single XLL
- 32/64-bit support
- Asynchronous non-blocking calls
- Task-based operations (.Net 4.0)
- Per-call WebSocket using WebSocket4Net
- Message transfer via JSON using Json.NET
- Automatically resizing the result range
public static object AnalyticHestonNpv(
    String optionType, Double strike, ...)
{
    VanillaOption option = new VanillaOption(
        _optionType: optionType,
        _strike: strike, ...);

    return RxExcel.Obsrve(
        "AnalyticHestonNpv",
        new object[] { optionType, strike, ... },
        () => AnalyticHestonNpvTask(option));
}
public static object AnalyticHestonNpvVerbose(
    [ExcelArgument("Is the options type: can be CALL or PUT.",
        Name="Option Type")]
    String optionType,
    [ExcelArgument("Is the options strike.",
        Name="Strike")]
    Double strike, ...)
{
    VanillaOption option = new VanillaOption(
        _optionType: optionType,
        _strike: strike, ...);

    return RxExcel.Observed(
        "AnalyticHestonNpv",
        new object[] { optionType, strike, ... },
        () => AnalyticHestonNpvTask(option));
}

[ExcelFunction("Returns the npv ...")]

private static Task<Double> AnalyticHestonNpvTask(VanillaOption option)
{
    var tcs = new TaskCompletionSource<Double>();
    var websocket = new WebSocket(string.Format("ws://\{0\}:\{1\}/analyticHestonNpv", address, port));
    websocket.Opened += (sender, args) => websocket.Send(...);

    EventHandler<MessageReceivedEventArgs> handler = null;
    handler = (sender, args) =>
    {
        tcs.TrySetResult(...);
        websocket.MessageReceived -= handler;
        websocket.Close();
    };

    websocket.MessageReceived += handler;
    websocket.Open();

    return tcs.Task;
}
Why WebSockets

- Stateless protocol
- Real-time full-duplex communication (sending and receiving at a time)
- Alternative to long polling or Comet
- Less bandwidth usage
- Initial HTTP request with an upgrade request to the WebSocket protocol
- Independent in and out streams
- No request/response cycle
Why favouring JavaScript Object Notation

• JSON is a text-based data format for data exchange

• Lightweight – no tags, no attributes, less bandwidth-intensive

• Limited data types (strings, numerics, Booleans, arrays, objects, nulls)

• Java and .Net APIs at hand for (de)serialization

• Can be persisted in NoSQL databases like MongoDB
{ "instrument": { "exercise": { "dates": [ "2013-09-26T18:00:00" ], "exerciseType": "European" }, "instrumentCurrency": "EUR", "maturity": "2013-09-26T18:00:00", "premium": { "cashFlows": [ [ "2013-08-21T00:00:00", -15000.0 ] ], "currency": "EUR" } }, }
Continuous Integration – the Plugin

### Build Steps

<table>
<thead>
<tr>
<th>Build Step</th>
<th>Description</th>
<th>edit</th>
<th>more</th>
</tr>
</thead>
<tbody>
<tr>
<td>NuGetConfig</td>
<td>Command Line&lt;br&gt;Custom script: svn export <a href="http://sm036g0.dom1.e-ssi.net">http://sm036g0.dom1.e-ssi.net</a>...&lt;br&gt;Execute: Only if build status is successful</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NuGetInstall</td>
<td>Command Line&lt;br&gt;Custom script: ...	runk\install\Packages.bat trunk&lt;br&gt;Execute: Only if build status is successful</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NuGet (disabled)</td>
<td>NuGet Installer&lt;br&gt;Solution: trunk\X11Pricing.sln&lt;br&gt;Execute: Only if build status is successful</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSBuild</td>
<td>MSBuild&lt;br&gt;Build file: trunk\X11Pricing.mssbuild&lt;br&gt;Targets: CIBuild&lt;br&gt;Execute: Only if build status is successful</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Configuration Steps

1. General Settings
2. Version Control Settings
3. Build Steps (3)
4. Build Failure Conditions
5. Build Triggers
6. Dependencies
7. Build Parameters (1)
Continuous Integration – the Plugin

```xml
<Project InitialTargets="CheckFolders"
    DefaultTargets="LocalBuild" ...>
    <PropertyGroup>
        ...
        <ExcelDnaPath>$(NuGetFolder)\Excel-DNA.0.30.3\tools</ExcelDnaPath>
    </PropertyGroup>
    ...
    <Target Name="PackageXll">
        <Message Text="=== copy dna, xll and config ==="/>
        <Copy SourceFiles="$(RootPath)\$(ProjectName)\$(ProjectName)-AddIn.dna"
            DestinationFiles="$(OutputPath)\$(ProjectName)-AddIn.dna" />
        ...
        <Exec Command="$(ExcelDnopath)\ExcelDnaPack.exe $(OutputPath) ...">/
        <Message Text="=== copying artifact and its config ===" />
        <!-- versioned artifact-->
        <Copy SourceFiles="$(OutputPath)\$(ProjectName)-AddIn-packed.xll"
            DestinationFiles="$(ArtifactPath)\$(Artifact)" />
    </Target>
</Project>
```
Continuous Integration – the Plugin

```xml
<DnaLibrary Name="X11Pricing Add-In"
    RuntimeVersion="v4.0">
    <ExternalLibrary Path="X11Pricing.dll"
        LoadFromBytes="true" Pack="true"/>
    <Reference Path="WebSocket4Net.dll" Pack="true"/>
    <Reference Path="Newtonsoft.Json.dll" Pack="true"/>
    ...
</DnaLibrary>

<configuration>
    <appSettings>
        <add key="play.http.address" value="localhost"/>
        <add key="play.http.port" value="9000"/>
    </appSettings>
</configuration>
```
Why Play

• Full-stack web framework for scala
  • Integrated HTTP server, build system and cache
  • Asynchronous I/O

• Stateless web application

• Live code and configuration changes

• Remote debugging in single threaded environment

• Type safety

• Build-in support for JSON validation

• Build-in support for WebSockets
Exposing a WebSocket with Play

• Specifying the routes

```scala
# Routes
# This file defines all application routes (Higher priority routes first)
# ~~~

# Home page
GET /analyticHestonNpv com.eon.pricing.server.Server.analyticHestonNpv
```

• Exposing the WebSocket

```scala
object Server extends Controller {

  implicit val simpleFactoryReads = (
    (__ ", "OptionType") .read[Option.Type] ~
    (__ ", "Strike") .read[Double] ~
    ...
  ) (SimpleFactory)

  def analyticHestonNpv = WebSocket.async[JsValue] { request =>
    Akka.future {
      val out = Enumerator.imperative[JsValue]()
      val in = Iteratee.foreach[JsValue] { msg =>
        msg.validate[SimpleFactory] match {
          case JsSuccess(value, _) =>
            val option = new EquityOption(value)
            out.push(
              JsValue.create(Json.obj("Value" -> option.analyticHestonNpv))
            )
          ...
        }
      }
    }
  }
```
Exposing QuantLib to Play

- SWIG
  - Simplified Wrapper and Interface Generator
  - Java extension to SWIG writes the Java Native Interface (JNI)
  - SWIG wraps C++ code using Java proxy classes
  - Embedded 32/64bit dll delivered with the jar file, extraction on the fly → no need for a separate dll deployment

- QuantLib in a multi-threaded environment
  - SWIG/QuantLib Objects are not shared between different threads
  - Deregister observer during garbage collection via call back hook
  - Thread local singleton pattern
Continuous Integration - QuantLib

<table>
<thead>
<tr>
<th>Build a Visual Studio project or solution using MSBuild</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSBuild Version</strong></td>
</tr>
<tr>
<td><strong>MSBuild Build File</strong></td>
</tr>
<tr>
<td><strong>Command Line Arguments</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Build a Visual Studio project or solution using MSBuild</td>
</tr>
<tr>
<td><strong>MSBuild Version</strong></td>
</tr>
<tr>
<td><strong>MSBuild Build File</strong></td>
</tr>
<tr>
<td><strong>Command Line Arguments</strong></td>
</tr>
</tbody>
</table>
Continuous Integration - QuantLib

```xml
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <RootPath>$(MSBuildProjectDirectory)</RootPath>
    <ProjectName>QuantLib</ProjectName>
    <SolutionFile>$(ProjectName)_vc10.vcxproj</SolutionFile>
    <Configuration Condition=" '$(Configuration)' == '"">Release</Configuration>
    <Platform Condition=" '$(Platform)' == '"">Win32</Platform>
    <PlatformConfig>$(Platform)\$(Configuration)</PlatformConfig>
    <TargetName>QuantLib-vc100-$(Platform)-mt</TargetName>
    <OutputDir>.uild\vc100\$(PlatformConfig)</OutputDir>
  </PropertyGroup>

  <Target Name="CIBuild" DependsOnTargets="Clean;Compile" />

  ...
</Project>
```
## Continuous Integration - SWIG

### Build

- **Copy artifacts from another project**
  - **Project name**: QuantLib-SNAPSHOT
  - **Which build**: Copy from WORKSPACE of latest completed build

- **Artifacts to copy**: lib/**, ql/**
- **Target directory**: QL
  - **Flatten directories**: Optional

### Execute Windows batch command

- **Command**: `call .\build.bat`

  See the list of available environment variables

### Use builders from another project

- **Template Project**: sbt 0.12.0

  Use all the builders from this project.
Continuous Integration - SWIG

```bash
swig.exe -java -c++ -outdir org/quantlib -package org.quantlib -o quantlib_wrap.cpp ../SWIG/quantlib.i

call :createDll Win32
call :createDll x64

:createDll
set _os=%

if _os==Win32 call %vcDir%\vcvarsall.bat

if _os==x64 call %vcDir%\vcvarsall.bat amd64

c /bigobj /EHsc -O2 quantlib_wrap.cpp -I"..." %quantlibDir%\lib\QuantLib-vc100-%_os%-%mt.lib -FeQuantLibJNI-%_os%.dll -MD -LD

seq(
  name : "quantlib-swig",
  version : "1.9-SNAPSHOT",
  name : "eet.applications.quantlib-swig",
  javaSource in Compile <<= baseDirectory / "Java",
  crossScalaVersions : Nil,
  crossPaths : false,
  mappings in (Compile, packageBin) <<= baseDirectory map{ base =>
    Seq(
      base / "Java" / "QuantLibJNI-Win32.dll" -> ""lib\static\Windows\x86\QuantLibJNI.dll"",
      base / "Java" / "QuantLibJNI-x64.dll" -> ""lib\static\Windows\amd64\QuantLibJNI.dll"
    )
  }
)```
Artifactory

- Central artifact repository for local and remote repositories
- Integrates with maven, ivy and NuGet
Debugging

• Start from VS in debug mode - debug your c# code

```xml
<Project ToolsVersion="4.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    ...
    <StartProgram>$(ProgramFiles)\Microsoft Office\Office14\EXCEL.EXE</StartProgram>
  </PropertyGroup>
</Project>
```
Debugging

- Run play in debug mode
- Attach remote debugger - debug your scala code
Hands-On

• Pricing a set of vanilla gas options from a spread sheet

• Sending a pricing request from a web browser

• Pricing a vanilla option from LexiFi
Conclusion

- QuantLib can be integrated into multi-language/architecture system
- High throughput
- Scalable with standard web components
- Continuous Integration and TDD
- Central pricing server
Links and Tutorials

- Principles of Reactive Programming
  https://www.coursera.org/course/reactive

- Functional Programming Principles in Scala
  https://www.coursera.org/course/progfun