CSCI 693 Research Paper

Rules Engines

Comparing .NET and Java based rules engines

By:

Manali Kulkarni
MS Candidate, California State University, Chico, CA 95929-0410
mkulkarni3@mail.csuchico.edu

Date: Dec 17, 2009.
Abstract – The financial services industry is a fast paced one; it is dominated by incessant requirements changes, regulatory compliance mandates and new products. To handle this volatile and fast paced environment, it is the need of the hour to have tools that are quick, flexible and accurate. The financial services sector thrives on Rules Engines; they are employed by most financial giants today to facilitate their day to day workings. In this paper I intend to research the two categories of Rules Engines – Java based and .NET based. Beyond the comparison; I also intend to emphasize the need, relevance and applicability of Rules Engines in the financial services world.

Index words – Rules Engines, Rule-based programming, Business Rules

I. INTRODUCTION

Rules Engine is a software tool that provides the user with the ability to create, classify, modify, manage, test and deploy business rules [2]. Rules engines are an integral part of a Business Rules Management System (BRMS). Business rules change more frequently than application code and Rules Engine gives the Business Analyst the ability to modify and deploy new business rules into the production environment without having to deal with the complex task of code redesign. One of the biggest merits of a Rules Engine is its ability to segregate application code from business rules.

II. BACKGROUND

Rules Engines, as the name suggests, it is a software tool that employs rule-based programming. Rule-based programming was first perceived in the seventies for AI (Artificial Intelligence) systems. Since then it has found applications in diverse fields such as Healthcare, Telecommunications, Travel and Transportation, Manufacturing and Supply chain, Insurance and Financial Services among many others. In the realm of rule-based programming, rules are expressed as simple “If <condition>, then <action>” statements.
Most commercial rules engines are inference/production rules engines that use the forward chaining algorithm. Inference engine, as the name implies, analyses the data available and draws inferences from it which help move closer to the final goal. Forward chaining algorithm gathers facts/data, draws conclusions and performs actions in accordance with the conclusion. It essentially fires all the rules that match a given condition and continue to do so until the goal is reached or there are not more conditions to match [9].

III. RULES ENGINES IN THE FINANCIAL SECTOR

Business rules are dynamic in nature and more so in the financial sector; they change constantly and not seldom. An example of a simple rule change in the mortgage industry would be a credit score eligibility change –

Change the existing rule:

If the customer has credit score > 720 then approve loan of $500,000 to

If the customer has credit score > 680 then approve loan of $500,000.

Having worked in the financial industry for more than 6 years I have seen criteria such as the credit score eligibility change as frequently as multiple times in a day. The example above is a simple 2-digit change that can be extremely crucial to the business involved.

Along with changing business rules, financial firms are also required to abide by a number of regulatory compliance directives. These mandates can change as often as once a week or as rarely as once a year. However it is compulsory to comply with the directives and failing to do so invites heavy penalties from the enforcing agencies. Rules engines make it easy for
users to implement these seemingly intricate mandates using lucid “If, then” statements. An example of mortgage fees compliance rules would be “If State is NY then Closing Fee is $0”. This simple rule makes sure that the loan process does not go forward until the closing fee field is made zero.

Traditional application development is rendered redundant when it comes to dealing with the ever-changing nature of business rules. In the typical orthodox environment where application developers handle all modifications and treat them as “code change”, a simple credit score change like the one mentioned above would require a lot of unnecessary effort and time. Employing the usual conventional approach would mean the business rules and the logic is intertwined heavily with the application code, database structures and objects, thus making it a tedious process to modify them quickly. A relatively plain and lucid rule change would still require the developers to scour through the application code, locate the business rule, figure out the logic, modify it, pass it on to the QA (Quality assurance) team, have them test it for errors, then pass it on to the UAT (User acceptance testing) team, have them test it to make sure it meets client/user specifications and then deploy it to the production environment. This makes the process laborious, time-consuming, expensive (too much man power and hours involved) and incompatible with the supersonic speed at which e-businesses operate.

Rules engines provide the necessary relief. They separate application code from business logic. This solves a majority of the problems involved with the time complexity and cost involved with the orthodox method of application development. Popular commercial rules engines today have the ability to fire thousands of rules per second and accommodate up to 50,000 rules [5]. Rules engines are so transparent and easy to manage that the entire cycle of
locating business rules, modifying, testing and deploying them to production can be done by a business analyst alone.

IV. TYPES OF RULES ENGINES

The two most dominant platforms employed in the industry today are Sun Microsystems Java [7] and Microsoft’s .NET [8]. Aptly so, the two types of rules engines that are provided by vendors are also Java-based and .NET-based. Most application development today is carried out in these two languages and although the BRMS separates application code from business logic, to be able to conceive an integrated system the rules engine needs to be capable of reading .NET and Java objects along with communicating with multiple databases.

1. .NET based Rules Engines

Microsoft’s .NET framework™ is a popular platform employed at many firms. It is essentially the advanced Windows component for running cutting edge software applications and web services. The .NET framework supports 20 different languages and incorporates techniques such as IntelliSense to make it easier to build and maintain contemporary software applications.

Rules engines based on the .NET framework integrate seamlessly with the .NET architecture and its components. The rules management services can be called effortlessly from the inherent .NET languages C#, J#, VB.NET and ASP.NET and also can be deployed as web services.
2. Java based Rules Engines

Sun Microsystems Inc. J2EE is a framework defined for component-based development and deployment of Java enterprise applications [4]. Java 2 Enterprise Edition (J2EE) simplifies the software development process without compromising on the robustness of the application.

Rules engine systems can be easily integrated within the multiple tiers of a J2EE application. The Rules engine blends with the J2EE servers as stateless or stateful Enterprise Java beans (EJB) [4]

V. EXPERIMENTS AND ANALYSIS

My research was conducted on a prominent commercial rules engine provided by FICO™ (Fair Isaac Co.). Amongst the various products that FICO™ has to offer, a popular one used widely in the financial services industry is the Blaze Advisor® BRMS [10]. FICO’s Blaze Advisor® is the leading business rules engine choice boasting of big name clients such as HSBC, Aviva, ICICI bank, Sun and DMV among many others [3, 4].

FICO™ offers a 90-day free trial of the Blaze Advisor® .NET and Java-based rules engines. The trial version comes with a well designed UI and a default repository in addition to a complete manual which covers all the topics ranging from installation to deployment of the rule services.

I have conducted my research, analysis and subsequent comparisons on this trial version.
1. Experimental setup and technical details -

1.1. Java-based rules engine –
   - Version: 6.8
   - Size: 851 bytes
   - Installed on: Intel(R) Core Duo CPU @1.80 GHz, 2038 RAM, 32-bit Operating system, Windows Vista

1.2. .NET based rules engine –
   - Version: 6.7.1
   - Size: 1.78KB
   - Installed on: Intel(R) Pentium @1.60 GHz, 504 RAM, Windows XP SP2

2. Details of the experiments:

After installing both the rules engines, I created test data that could be compiled and run through the UI. I created a new project called “Loan Approval process” and wrote simple business rules such as credit score eligibility rules, income eligibility rules etc which are an important component of a loan application process I created a ruleset (a container for storing multiple rules) with 15 rules to start with. As I compiled and ran the ruleset, I noted the performance of both engines in terms of time required for execution. I increased the number of rules to 25 and eventually 50. With the number of rules increased, as anticipated the number of milliseconds to execute increased too, the results are discussed in the following section. Some of the other parameters considered for comparing the two engines were ease of installation, interoperability, architectural advantages, issues with JDK and .NET framework etc.
The following section discusses the results and subsequently compares the two engines.

VI. RESULTS AND COMPARISON

The results compare both rules engines based on the following parameters –

1. Ease of Installation: Although both engines had simple installation tasks, the Java-based engine scored over the .NET one because the .NET engine needed a preinstalled Microsoft Visual Studio and the .NET framework whereas the Java one simply needed the Java Virtual Machine (JVM).

2. Performance – Execution time: Test data consisting of 15 rules within a ruleset was initially created, compiled and run on both rules engines. The number of rules was increased to 25 and finally 50. Screenshots of the results at 15 and 50 rules are shown below, pointers indicate the number of rules executed and the time taken.
The results of the execution times are tabularized below.

<table>
<thead>
<tr>
<th></th>
<th>.NET</th>
<th>J2EE</th>
</tr>
</thead>
<tbody>
<tr>
<td>No of rules</td>
<td>Time</td>
<td>No of rules</td>
</tr>
<tr>
<td>15</td>
<td>3.0</td>
<td>15</td>
</tr>
<tr>
<td>25</td>
<td>4.7</td>
<td>25</td>
</tr>
<tr>
<td>50</td>
<td>5.9</td>
<td>50</td>
</tr>
</tbody>
</table>

**Table 1 .NET execution**

**Table 2 J2EE execution**
It is clear from Figure 1 that the .NET-based rules engine had much better execution times compared to the J2EE one.

3. Architecture: Both engines have the rules engine set in the middle tier which is the business tier as seen in Figure 2 and Figure 3. However with the .NET based rules engine, since all the application development happens in this tier, the presence of the rules engines overloads the application development block and causes technical snarls.

Figure 2: Architecture of Rules Engine in .NET framework (from [3])
With the J2EE, the EJB (Enterprise Java Beans) container houses the rules engines. The EJB container provides services such as state management, transaction management, thread management, resource pooling and security. It uses entity, session and message beans for various communications thus keeping the business tier (Application block) from overload.

Figure 3: Architecture of the Rules Engine in the Java-based structure (from [4])

4. Choice of programming languages: In terms of choice of programming language Microsoft’s Visual Studio and .NET framework offer programming in C#, VB and J#, in addition to 20 other programming languages. Whereas with J2EE, the only programming language available is Java.

5. Importing Business Objects Models (BOMs): The .NET based rules engine has the capability to import a variety of BOMs such as .NET, COM, Database, XML and
also Java BOMs. However the J2EE-based engine can import Java, database and XML BOMs but does not have the ability to import .NET BOMs.

Screenshots below show the import BOMs options in both tools.
6. Interoperability: In my opinion the biggest advantage of the .NET based rules engine over the J2EE one is its interoperability. The Java based engine interacts with applications and servers that conform to J2EE standards and uses specially designed APIs to communicate with other software, however it does not provide interoperability with .NET components. Whereas the .NET based engine connects seamlessly with non-.NET components such as ActiveX and other .dlls. Java code can be recast as J# code to use Java components with .NET.

7. Issues with the .NET framework:

- Memory leaks when using .NET framework 2.0
- Scroll bar moves slowly while using decision metaphors in RMA (Rules Maintenance Application) on .NET framework 2.0

*Solution:* Microsoft has provided hotfixes for both these bugs on their support website.

8. Issues with JDK (Java Development Kit):

- Incompatibility between JVMs and video cards. Results in null pointer exception referencing graphics.
- Builder IDE exits unexpectedly in JDK 1.5
- Large zip files can cause pickers to hang
- Hotspot virtual machine error when changing file name in the ‘Select a File’ dialogue box
Solution: Sun Microsystems has provided fixes for all these issues.

The JDK framework has more issues compared to the .NET framework. However both Sun Microsystems and Microsoft have provided the necessary fixes for the problems.

9. Execution modes: Both engines have two modes of execution namely

- RETE Execution mode: The RETE algorithm is an efficient method for computing the set of satisfied rules incrementally after each rule execution. RETE algorithm realizes a total indexing of the data base according to rule conditions [6].

- Sequential mode: In this mode, each rule is fired in a linear fashion.
The following table summarizes the results of the comparison. The check mark symbolizes which one fared well and the cross mark indicates the one that fared poorly.

**Table 3: Results of the comparative study**

<table>
<thead>
<tr>
<th></th>
<th>.NET</th>
<th>J2EE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ease of Installation</strong></td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Better performance</strong></td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Architectural benefits</strong></td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Programming languages</strong></td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Importing BOMs</strong></td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Interoperability</strong></td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Fewer issues with the framework</strong></td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

**VII. CONCLUSION**

Business Rules Engines are essentially software applications that contain definitions of Business rules [1]. Rules engines are widely used in diverse industries today. One of the biggest advantages of a rules engines is the fact that it moves business logic from being deeply embedded (hard coded) in application code and treat them as business rules so that they can be changed more easily making them more flexible and thus enhancing business agility [9].
The experimental analysis and subsequent comparisons performed on the .NET and J2EE based engines, the .NET based rules engine proved to be superior in comparison with the J2EE one.

Although performance analysis and other underlying framework issues are important considerations while selecting a rules engine, the primary deciding factor remains the existing platform (.NET or Java) employed. Since the cost of moving from one platform to another is very high, it is most likely that an existing .NET platform favors a .NET based engine and a J2EE platform gets a Java-based engine. This is where the interoperability with Java components offered by the .NET-based engine could be a crucial deciding factor and result in a decision favoring the .NET one.
REFERENCES


