BRMS を適用するためのフィージビリティ・スタディ

― ビジネスルール設計方式と実行性能 ―

李 東昇
(株) 富士通システムズ・イースト
li.dongsheng@jp.fujitsu.com

要旨

経営やビジネス環境の変化に追随するため、情報システムの開発・変更を即応させなければならない。手段として、超高速開発[1] [4] [5] [6]という考え方とそれを支える超高速開発ツールが注目を浴びている。BRMS (Business Rule Management System) やノンプログラミングツールは、その代表である。BRMS の適用に向けて、下流工程での主な課題として「ルール設計方式の指針はどう決めるか」と「ルール方式導入による性能的に不安はないか」の２つがある。これらの課題に対してビジネスルールの組み合わせ方と非機能要件項目[2] (性能や保守性等) に着目しフィージビリティ・スタディを行った。本論文では疑似的な業務モデルを想定し、性能測定を通してルール設計方式の選択指針について述べる。

１．背景

経営環境の変化や技術の進化に伴い、情報システムに俊敏な対応が求められている。具体的には、短期開発やビジネスの変化対応力を実現するために、BRMS の適用が求められている。商談においても、RFP ((Request For Proposal) に BRMS 適用が条件となることが多く、今後の SI 技術には欠かせないツールとなってきている。

産業業界ではメインフレームからの脱却を目指しているお客様が増えている。これらのお客様は基幹システムのマイグレーションの検討に当たり、「現在の業務システムが将来に渡って継続的に運用できるか」と「プログラム仕様のスパゲッティ化によるメンテコストはどう削減できるか」という課題に直面している。解決手法として BRMS の導入が検討されている。

BRMS は、アプリケーションの業務ロジック部分を「ルール」として定義し、「ルールエンジン」により管理・実行されるシステム・ソフトウェアのことである[3]。「ルール」は業務ユーザーでも理解可能な形式で記述でき、記述した「ルール」はプログラムに自動変換され、アプリケーションから利用・実行できる。業務ロジックの改修が必要な場合には、ルールの変更のみで迅速に対応できる点がメリットである。

富士通グループでは、オープンソースツール (JBossBRMS) [4]にも取組んでいる。しかし実適用の実施例がないため、業務ルール実装方式の適用検討、基本的な使い方のノウハウ及び性能の基礎値を確認する必要があった。本論文では JBossBRMS を使いビジネスルールの組み合わせによりアプリケーションを開発する際に必要となる、使い方のノウハウ及び性能の基礎値を得るために行った検討とその結果について論述する。

２．課題

JbossBRMS を業務システムに安心して適用できるようにするためには、ＢＲＭＳの開発標準が無いため、各現場にて調査し、方式設計や業務適用指針の検討などを個別に行わなければならない。下流工程での BRMS 開発標準検討の一環として、まずもっとも重要な以下の２つの課題にフォーカスした。

課題１：ルール設計方式 (スプレッドシート方式／マスタ FACT 方式) の選択指針がない。

JBossBRMS のビジネスルールは、スプレッドシートでコード実装する方式と、マスタデータをマスタＦＡＣＴに展開してパターンマッチングする方式の、２種類の実装方式がある。どちらの方式が適切か、開発標準での観点からルール設計方式の方針を決めなければならない。

課題２：実行性能に関する定量的なガイドがない。

ルールの実行性能については定量的な評価を実施するため、サーバー上でほかのアプリケーションが動いていない“無風状態”と、トランザクションが集中している“高負荷状態” の２つの状態の性能を計測し、性能劣化の状況を検証する。また、メモリの配置目処はどのぐらいすべきかについて、ヒープサイズの Max 値を抑制し、必要最低限のメモリサイズを確認する。これらの計測値をまとめて、BRMS 導入時の定量的な参考値として提示しなければならない。

３．課題の対応策

郵便番号情報を検索する業務モデルを実装し、非機能要件項目(性能や保守性等)を検証することにより、ルール方式の選択方式を明確にする。具体的に、Ｗｅｂアプリケーションにルール呼び出しを実装したアプリケーション構成とする。ルールの呼び出しは Servlet から API で行い、ルールのステートフルセッションプールを使用する。トランザクションの発生は、テストツールとしてオープンソースの JMeter を使用して行う。

一般に業務データの操作パターンは作成・検索・更新と削除（CRUD）で定義することができる。BRMS は基本的に処理条件の複雑さに依存する処理となる。したがって DB の処理性能とは異なり、条件の複雑さによる検証が必要である。今回はルールを偏りなく一般化するため、検索のパターンを選定した。

ルールの設計方式について、スプレッドシートでコード実装した方式（以下、ルール Base と略す）と、マスタデータをマスタ FACT に展開してパターンマッチングする方式（以下、マスタ FACT と略す）の、２種類の実装方式を検証する。

図１は「条件項目＝10」の場合のルール Base の実装仕様のイメージで、「CONDITION」欄は条件項目で、「ACTION」欄は検索結果である。ルールのロード処理は、初回のみルール全体が Production Memory へ展開される。その後、ルールの実行はセッション取得、FACT Insert、Rule Fire とセッション解放 の順で行われる。

図２は「条件項目＝10」の場合のマスタ FACT の実装仕様で、マスタ FACT とトランザクション FACT をルールにより検索のためのパターンマッチングを行う。ルールのロード処理については、セッションプール取得時に重複したマスタロードを防止するため、プール内の FACT が空であれば、マスタの Insert を行う。マスタの Insert 方法として、アプリ側は一旦 ArrayList にマスタデータを格納後、それをルールエンジン側に渡す。そして、ルールエンジン側で ArrayList から抽出して Insert する。ルールの実行はルール Base と同じである。

RuleTable 郵便番号マスタ照会_1										
CONDITION	CONDITION	CONDITION	CONDITION	CONDITION	CONDITION	CONDITION	CONDITION	CONDITION	CONDITION	ACTION
$fc:照会データ										
照会 郵便番号桁 ==″$1″	″$1″ ==″0″	全国地方 公共団体コード ==″$1″	旧郵便番号5桁 ==″$1″	カナ都道府県名 ==″$1″	カナ市町村名 ==″$1″	カナ町域名 ==″$1″	都道府県名 ==″$1″	市町村名 ==″$1″	町域名 ==″$1″	$fc.set照会結果($1);
郵便番号7桁	備考5	全国地方 公共団体C	旧郵便番号 5桁	カナ都道府県名	カナ市町村名	カナ町域名	都道府県名	市町村名	町域名	町域名
0600000	0	01101	060	ﾎｯｶｲﾄﾞｳ	ｻｯﾎﾟﾛｼﾁｭｳｵｳｸ	ｲｶﾆｹｲｻｲｶﾞﾅｲﾊﾞｲ	北海道	札幌市中央区	北一条東	true
0640941	0	01101	064	ﾎｯｶｲﾄﾞｳ	ｻｯﾎﾟﾛｼﾁｭｳｵｳｸ	ｱｻﾋｶﾞｵｶ	北海道	札幌市中央区	旭ケ丘	true
0600041	0	01101	060	ﾎｯｶｲﾄﾞｳ	ｻｯﾎﾟﾛｼﾁｭｳｵｳｸ	ｵｵﾄﾞｵﾘﾋｶﾞｼ	北海道	札幌市中央区	大通東	true

図１　ルール Base の実装仕様のイメージ（条件項目＝10）

・マスタFACT

```
public class 郵便番号マスタ{
    public String 全国地方公共団体コード;
    public String 旧郵便番号5桁;
    public String 郵便番号7桁;
    public String カナ都道府県名;
    public String カナ市町村名;
    public String カナ町域名;
    public String 都道府県名;
    public String 市町村名;
    public String 町域名;
    public String 削除フラグ;}
```

・トランザクションFACT

```
public class 照会データ extends 郵便番号マスタ{
    public String 照会郵便番号7桁;
    public String 照会結果; }
```

左記の2ファクトを、下記ルールでパターンマッチングする。

・条件項目＝10の場合

```
rule ″郵便番号照会 Rule"
  when
    $Fact : 照会データ( $pram1:照会郵便番号7桁, $pram2:全国地方公共団体コード,
                $pram3:旧郵便番号5桁, $pram4:カナ都道府県名, $pram5:カナ市町村名,
                $pram6:カナ町域名, $pram7:都道府県名, $pram8:市町村名, $pram9:町域名 )
    $Mast : 郵便番号マスタ( 郵便番号7桁==$pram1, 全国地方公共団体コード==$pram2,
                旧郵便番号5桁==$pram3, カナ都道府県名==$pram4, カナ市町村名==$pram5,
                カナ町域名==$pram6, 都道府県名==$pram7, 市町村名==$pram8, 町域名==$pram9,
                削除フラグ==″0″)
  then
    $Fact.set照会結果(true);
    retract($Fact)
end
```

図２　マスタ FACT の実装仕様のイメージ（条件項目＝10）

実装仕様とルールのロード処理を比べると、図１のルール Base の設計方式は実装しやすさのみならず、“ルールの見える化”も優れている。マスタデータがない限り、ルール Base の設計方式を選択すべきである。(保守性)

既存マスタ情報の活用を考慮すると、図２のマスタ FACT の設計方式はマスタ DB を容易に取り込むので、マスタ FACT の方が有効であると考えられる。(移行性)

ルールの実行性能に関しては定量的な参考値を明確にするため、“無風状態”と“負荷状態” の２つの状態の処理性能を計測し、処理性能のフィージビリティを検証した。結果として、性能要件としてのレスポンスタイム（3 秒以内）には問題がないことを確認した。但し、ルールの実行性能への影響要因等についてはいくつの考慮点がある。その中で特に BRMS 特有の特徴と GC（ガーベージ・コレクション）による性能への影響について考察する。(性能)

１）“無風状態”でのルールロード時間とルール実行時間の計測

JMeter から１多重で 20 回の HTTP リクエストを連続（応答があり次第、次の要求を即座に）送信し、レスポンスタイムを計測する。図３は、条件項目数＝４（ルール Base とマスタ FACT）のロード時間とルール実行時間を計測した結果である。

ルール Base の場合、ルールの読み込み性能は、母数が大きくなるほど１ルールあたりの読み込み性能も劣化する傾向が見られる(120,000 件の場合は 19.1 ミリ秒)。マスタＦＡＣＴの場合、マッチング用ルール（DRL 言語）のルールロード時間が約 5,000 ミリ秒かかるが、１件当たりの読込み性能は対象のデータ件数による違いは見られない。JBossBRMS は、小規模なルールでも読込みオーバーヘッドが大きいので、事前にセッションプールにロードすることが必要である。実行性能は、どちらでも今回検証したルール件数１２万件までにおいてばらつきはあるものの、顕著な劣化は見られなかった。

（実測値）

ルールBaseの設計方式

No.	ルール数	ルールロード（ミリ秒）	ルール実行（ミリ秒）	読込み時間/rule
1	10,000	18,289.40	2.08	1.8
2	30,000	82,603.40	1.45	2.8
3	120,000	2,286,594.20	2.91	19.1

マスタFACTの設計方式

No.	データ件数	ルールロード（ミリ秒	マスタＦＡＣＴ展開（ミリ秒）	ルール実行（ミリ秒）	読込み時間/件
1	10,000	4,938.40	526.50	1.24	0.05
2	30,000	4,850.40	1,271.75	1.68	0.04
3	120,000	4,803.80	4,686.80	2.05	0.04

図３　“無風状態”の実行時間とロード時間

２）負荷状態でのルール実行時間の計測

3 万ルールが登録されている環境で、JMeter により HTTP リクエストを発生させて、「Rush 状態※」でのルールの処理速度を計測する。多重度は、50 多重、100 多重と 200 多重にして、それぞれ 75,000、150,000 と 300,000 のサンプルトランザクション数（多重度×500 リクエスト/回×３回）に対して計測結果を統計分析した。

※セッションプールの初期確保数を多重度分とし、このプールに事前にルールを読み込む為に、Rush 走行前に暖気運転として全スレッドから初回要求を送信し、その後にＲｕｓｈ状態となる。

図４は、ルール Base とマスタ FACT の「Rush 状態」でのルール実行時間とその分布（１ミリ秒未満、20 ミリ秒未満とそれ以外)を計測・統計した結果である。ルール実行時間は、セッション取得 ＋ FACT Insert ＋ Rule Fire ＋ セッション解放 の一連の処理の合計時間を指す。

（実測値）

ルールBaseの設計方式

No.	テストケース	ルール実行（ミリ秒）	1ミリ秒未満	20ミリ秒未満	それ以外
1	50多重	13.84	58.1%	20.1%	21.8%
2	100多重	23.75	57.9%	11.6%	30.5%
3	200多重	32.71	61.5%	7.7%	30.7%

マスタFACTの設計方式

No.	テストケース	ルール実行（ミリ秒）	1ミリ秒未満	20ミリ秒未満	それ以外
1	50多重	9.15	71.6%	14.6%	13.9%
2	100多重	14.03	68.4%	12.8%	18.8%
3	200多重	19.20	67.4%	10.4%	22.2%

図４　“”負荷状態”のルール実行時間と分布

ここでは、1,000 ミリ秒を閾値として、これを越えるデータを GC（ガーベージ・コレクション）による例外データとみなして統計から除外する。ルール Base の検索時間は 50 多重で約 14 ミリ秒と高速を維持して、200 多重でも約 33 ミリ秒という 2.3 倍ほどの数値に落ち着いている。また、マスタ FACT の検索時間がルール Base の約６割で、ルール Base よりマスタ FACT の処理性能が良かった。これで、負荷状態でも両方式とも良い性能を維持することが可能であると確認できる。時間と TPS を計測・統計した結果である。

図５は、200 多重において、Ｒｕｓｈ状態での経過時間(横軸：秒)と、その時点で受信した処理のルール実行時間（縦軸：ミリ秒）の分布図である。1,000 ミリ秒以上のデータは GC による例外データだと考える。

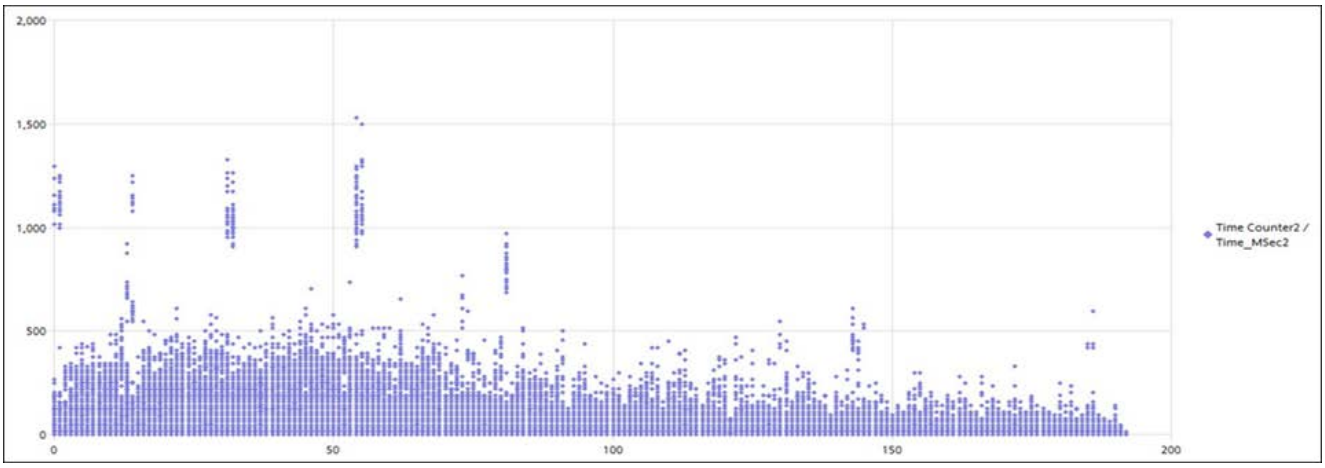

図５　２００多重の経過時間とルール実行時間の分布図

３）セッションプールの MAX 値と Idle 値による影響分析

セッションプールの作成時間は今回の計測では意識するレベルの時間は要しなかった。ルール Base の場合はルール全体で初回のみ Production Memory へ展開され、各セッションプールからは共有領域として参照されるため、Idle 分で起動されるセッションプールだけでなく MAX 値で新規展開されるセッションプールにおいてもルールロード時間は考慮しなくてよかった。マスタ FACT の場合は、アイドル状態のセッションプールは Production Memory の内容を維持できるので２回目以降は高速で処理できるが、Idle 値を超えた MAX 値までのセッションプールは毎回新規展開とセッションプールのクリアが行なわれるせいで、毎回マスタ FACT の展開分をロスすることとなった。環境としての資源が許されるなら、セッションプールの設定は Idle 値と MAX 値を同一にすることを勧める。MAX セッションプール数を超えるリクエストが集中した場合には、セッションプールの空き待ちによるロスが発生する。表１は、ルール Base　３万ルールでの「待ち無し」と「待ち有り」のセッション取得時間の比較であり、「待ち有り」は「待ち無し」の約 35 倍であった。この結果から、可能な限り、最大同時接続数をセッションプール数として確保できる方が望ましいと言える。

表１　**セッション取得時間の比較**

[セッション取得時間]　　単位:ミリ秒

No.	テストケース	①待ち無し	②待ち有り	①:②
1	50多重	0.86	29.94	1 ： 34.81
2	100多重	1.46	58.13	1 ： 39.82
3	200多重	2.41	87.30	1 ： 36.22

※待ち無し･･･起動時に多重数と同数のセッションプールを確保
待ち有り･･･最大セッションプールを10に制限して起動

４）GC による影響分析

“負荷状態”でのテストと同等の条件下で、ヒープサイズを抑制し必要最低限なメモリ容量を検証した。またその状況下での処理性能を測定した。検証した結果としてヒープサイズの最低必要量は、ルール Base の場合約 1.5GB、マスタ FACT の場合約 7GB であった。それを下回ると OutOfMemory Exception が発生する。

表２はルール Base　３万ルール、200 多重のテストケースに対して、ヒープサイズを抑制なしの 40GB、抑制ありの 13GB、抑制ありの 1.5GB、の３つの場合におけるルール実行時間と TPS を計測した結果※である。抑制ありの 1.5GB で計測した場合、かなりのＧＣが発生しており、TPS が下がった大きな原因になると言える。

表２　メモリ抑制ありとなし場合の“負荷状態”でのルール実行時間と TPS

テストケース	項目	抑制無し	抑制有り(13GB)	抑制有り(下限)	無 : 13GB : 下限
ルールBase 3万ルール200多重	YGC回数	11	20	140	1 : 1.8 : 12.7
	FGC回数	0	1	3	0 : 1 : 3
	ルール実行時間	32.71	24.20	19.24	1 : 0.7 : 0.6
	TPS	521	457	384	1 : 0.9 : 0.7

※ルール実行時間（平均値）を計算時、1,000ミリ秒を閾値として、これを越えるデータをGCによる例外データとみなして集計から除外する。

４．ルール設計方式のガイド（以下、本ガイドと略す）

１）ルール設計方式の選択指針

図６は、ルール Base 方式とマスタ FACT 方式に対する一部の非機能要求項目を比較したものである。保守性はルール Base 設計方式の方が優れている。移行性とルール管理の効率化はマスタ FACT のほうが有利である。ルール設計方針の選択例として、ルール数が少ない場合は可視化の良いルール Base が効果的と考える。既存マスタを活用したい場合はマスタ FACT 設計方式を選択すべきである。もちろん、複雑なルール設計の場合は、両方式を併用するハイブリッド方式も考えられるだろう。

	設計方式		
比較項目	ルールBase	マスタFACT	備　　考
性能	○	○	今回の測定したようにルールBase設計方式よりもマスタFACT設計方式の方が処理若干有効であるが、両方ともミリ秒のレベルであった。
保守性	○	△	“ルールの見える化”と実装しやすさはルールBase設計方式の方が優れている
移行性	△	○	・既存マスタ情報の活用 マスタFACTを設定する場合、他システムで利用中のマスタDBを容易に取り込むことが可能である
効率性	×	○	・ルール管理の効率化 コードから名称へ、逆に名称からコードへ変換するような双方向の変換が求められる場合に、ルールBase方式では双方向の変換ルールをそれぞれ記述する必要があるが、マスタFACT方式では一方向だけの記述で良い

図６　ルール設計方式の非機能要件項目の比較

２）ルールの実行性能の定量的な参考値及び考慮点

今回は主に３万ルールを基準として計測した。“無風状態”と“負荷状態”ともにミリ秒の実行性能なので、性能要件としてのレスポンスタイム（3 秒以内）には十二分な性能結果が出たことで問題なく活用できると考える。但し、ルールロード時間を減らすため、ルール実行サーバーでは事前にルール環境の立上げを進める。また、セッションプールに関するパラメータの設定や GC によって影響があることも確認できた。可能な限り最大同時接続数をセッションプールの MAX 値と Idle 値として設定することが望ましい。最後にヒープ領域をふんだんに使用するため、適正なサイズを確保しないと GC の影響を受ける。性能担保のためにも運用に合わせた、この領域のチューニングが重要である。

５．おわりに

超高速開発における BRMS の実用化について、JBossBRMS についての基本的な使い方のノウハウ及び性能の基礎値を確認することができた。これは JBossBRMS を顧客のシステムに適用する際の参考になりうる。ここ数年、BRMS についての問い合わせや相談などが急増している。これは、俊敏性をもとめた超高速開発への期待に比べ、一般書籍を始め、社内外に BRMS の適用に関する技術情報が少ない状況にあるためである。BRMS は、システムの一部かもしれないが、ビジネス全体から見ると重要度は高く、商談を発展させる切り札と考える。BRMS によるアプローチがビジネスを新しい方向へ導く鍵となる。今後ますます注目を浴びていく[5] [6] [7]と考える。

今回のフィージビリティ・スタディでは方式の使い分けと性能について検証した。しかし、実際の業務ルールのすべてを本当に表現することが可能か否か、ルール件数の量的面と業務の複雑さからの検証はまだ今後の課題として残っている。

参考文献

[1]「超高速開発のためのルールベース開発技術の研究」. LS 研成果報告書. 2014 年度
http://jp.fujitsu.com/family/lsken/activity/w

ork-group/14/abstract/outline_04.html
[2] 「経営に活かす IT 投資の最適化」. 情報処理推進機構（IPA）. 2011 年 4 月
http://www.ipa.go.jp/sec/reports/20110427.html
[3] 森田　武史、山口　高平.「業務ルール管理システム BRMS の現状と動向.「人工知能学会誌」 29(3), 277-285, 2014-05-01
[4] 「Red Hat Jboss BRMS」.
http://www.jboss.org/products/brms/overview
[5] 井上英明.「超高速開発」が日本を救う.「日経コンピュータ」. 2012 年 3 月 15 日号.
http://itpro.nikkeibp.co.jp/article/NC/20120309/385541/?ST=NC
[6] 井上英明.「みずほもすなる超高速開発」. ITpro By 日経コンピュータ. 2015 年 10 月 16 日
http://itpro.nikkeibp.co.jp/atcl/watcher/14/334361/082400358/?ST=system&P=1
[7] 井上英明.「広がる超高速開発　限界と常識を打ち破る」. ITpro By 日経コンピュータ. 2016 年 1 月 4 日
http://itpro.nikkeibp.co.jp/atclact/active/15/121700150/121700001/