3N-4

実応用を想定したAWSミドルウェアの評価

平本　真道†　木村　泰輔†　大友　浩照†　大谷　真†
湘南工科大学†

1.はじめに

AWS(自律型 Web サービス)はインターネットで商取引を柔軟に実行するための新たな基盤技術であり、一昨年そのミドルウェアのプロトタイプが完成した[1]。しかし AWS として仕様の妥当性を含めた評価が十分に行われていない。本研究では、昨年[2]に続き実際の商取引を想定した実験アプリケーションを作成して、ユーザプログラム開発の容易性やミドルウェアの現仕様の評価を行った。

2.AWS(自律型 Web サービス)

従来の Web サービスはシステムを横断したビジネスプロセスモデル（BPM）を記述しておき、すべての関連システムがその BPM に従って動作する必要がある。AWS はこの制限を外し、独自の BPM を持つ個々のシステムが他のシステムと動的に商取引メッセージ交換ができる点に特徴がある。AWS を使った商取引アプリケーションの実行環境を提供するのが AWS ミドルウェアで、一昨年プロトタイプが完成した[1]。AWS ミドルウェアでは、個々のシステムが持つ BPM を記述し、オペレーションに関連する処理を行うアプリケーションプログラムを書き、若干の付加的なファイルを作成するだけで、AWS による商取引アプリケーションが作成できる。

3.評価アプリケーション

昨年に引き続き AWS ミドルウェアの現仕様を評価するために複数の評価アプリケーションを作成することにした(表１)。AWS ミドルウェアによるモデルの変形後の動作の評価のために、BPM 変形を前提とした X01 と X02 を、BPM 変形後のビジネスプロセスの分岐処理の機能の評価のために、変形後に実行可能な分岐とそうでない分岐の両方を持つ BPM の Y01 及び Y02 を考案した。現在まで、X01 及び X02、Y01 及び Y02 の作成とそれを使った実験が終わっている。

4. X01 と X02

(1)X01 と X02 の概要

図 1 に X01 と X02 の動作概要と、動的協調後の X01 の概要を示す。AWS ミドルウェアは取引参加者の BPM を相互に協調変形させることで取引を可能にする。変形後のモデルどおりにモデル駆動が行われるかを評価するために、協調変形によって分岐が消滅して動作が一本化する X01 と、取引相手としてのシステム X02 を作成した。

表 1　作成した評価アプリケーション一覧

名前	ねらい	概要
X01	変形後のモデルで正しくモデル駆動が行えるか評価	動的協調で動作が一本化するBPM
X02	X01に取引相手システムを提供	X01の動的協調の相手BPM
Y01	変形後のモデルで正しく分岐処理が行えるかどうか評価	動的協調後に分岐を持つBPM
Y02	x01に取引相手システムを提供	Y01の動的協調の相手BPM

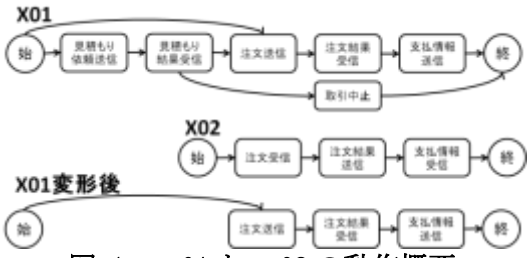

図 1　ex01 と ex02 の動作概要

(2)BPM（ビジネスプロセスモデル）

AWS でいう BPM とはオペレーションに応じた状態遷移のことである。図 2 は AWS の規定に従って XML で X01 の BPM を記述したものである。AWS ミドルウェアがこの BPM を入力して動的協調及びアプリケーションの自動駆動を行う。

```
<operations>
 <operation name="Estimate" format="Est01"><pattern>output<pattern></operation>
 <operation name="EstAns" format=" Ans01"><pattern>input<pattern></operation>
 <operation name="Cancel" format=" Cancel01"><pattern>output<pattern></operation>
 <operation name="Order" format="Order01"><pattern>output<pattern></operation>
 <operation name="Response" format=" Res01"><pattern>input<pattern></operation>
 <operation name="Payment" format=" Pay01"><pattern>output<pattern></operation>
</operations>
<behavior>
  <states>
   <state no="0"><next operation="Estimate">1</next>
                 <next operation="Order">3</next> </state>
   <state no="1"><next operation="EstimAns">2</next></state>
   <state no="2"><next operation="Order">3</next>
                 <next operation="Cancel">4</next> </state>
   <state no="3"><next operation="Response">4</next></state>
   <state no="4"><next operation="Payment">5</next></state>
   <state no="5"></state>
  </states>
  <first>0</first>
  <last>5</last>
</behavior>
```

図 2　X01 の BPM の XML 記述

Evaluation of AWS middleware based on realistic applications
†Masamichi Hiramoto, Taisuke Kimura, Hiroaki Otomo, Makoto Oya, Shonan Institute Technology

(3)プログラム本体

図3にX01のアプリケーション本体AppX01を示す。AppX01は6つのオペレーションに対応した6つのメソッドを持つ。各メソッドはconfigによって各オペレーションに対応づけられている。AppX01は現仕様で変形後のモデルに合わせて自動的にモデル駆動が実行できるかの調査が目的なので、各メソッドは遷移先を指定する命令を持たない。送信OPに対応するメソッドはデータコンテナクラスに対して情報、見積希望商品名(name)等を渡す。データコンテナは受け取った情報(name)等を含むXML電文を生成する。受信OPに対応するメソッドはデータコンテナから受信した情報、お届け予定日(date)等を取り出して処理する。データコンテナクラスとはXML電文の詳細をアプリケーションから隠蔽するためのクラスである。

```
public class AppX01 extends AWSFramework {
 public void apEstimate(EstimData out) {
   out.name = 見積希望商品名; out.num = 個数; }
 public void apEstimAns(EstimAnsData in) {
  お届け予定日(in.date)と合計見積価格(in.price)を処理する}
 public void apCancel (Cancel out) {
   out.cancelReason= キャンセル理由; }
 public void apOrder(OrderData out) {
   out.item= 注文商品名; out.qty = 個数; }
 public void apResponse(ResResult in) {
   お届け予定日(in.date)と請求金額(in.price)を処理する}
 public void apPay(PaymentData out) {
   out.payDate= 支払日時; }
}
```

図 3　X01のアプリケーション本体(AppX01)

(4)実験結果

X01は動的協調後のモデルに合わせて正常に動作した。これにより、AWSミドルウェアによる動的協調とモデルの変形を行った場合でも、特に指定せずに協調後のモデルに合わせてモデル駆動が実行できることが検証できた。

5. Y01とY02

図4にY01とY02の動作概要と動的協調後のY01の概要を示す。動的協調後のモデルに、消滅した分岐が存在するケースで、変形後のモデルに合わせて分岐制御がたやすく行えるかどうか検証するためにY01とY02を作成した。

注文結果送信OPは二つの遷移先を持ち、両者が実行可能な場合は遷移先を任意で選択できることが望ましい。注文結果送信OPはユーザの操作で遷移先を制御する際に、遷移可能OPの数をミドルウェアのAPIの getNextOperationLength()で獲得し、もし1個だけであればそのOPを遷移先に指定し、複数であれば実行可能なOP名をAPIの getNextOperation()で取得してその中から一つをユーザが選択して実行する。

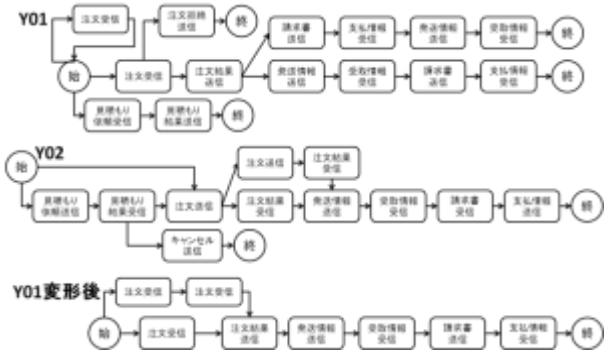

図 4　Y01とY02の動作概要

```
public class AppY01 extends AWSFramework {
 public void apOrder(OrderData in) {
   ......
 public void apOrdRes(OrdResData out) {
   データコンテナに注文結果の情報を格納する
   もし実行可能OPが一つなら
     string nextOp = getNextOperation[1];
     setNextOperation(nextOp);
   実行可能OPが複数なら
     getNextOperation[]で実行可能OPを取得する。
     実行したいOPをsetNextOperation()で指定する。
 }
 public void apBill(PaymentData in) {
   ......
```

図 5　Y01のプログラム本体(AppY01)

6. まとめ

X01とX02によりモデルの動的協調後もミドルウェアが利点であるモデル駆動が正しく動作することが確認できた。更にY01とY02ではミドルウェアのAPIを活用することでモデルの動的協調に合わせてビジネスプロセスの流れの分岐制御を柔軟に行えることが検証できた。実験の余禄として現仕様のミドルウェアでは一部の非決定性オートマトンの実行が不可能であることが示唆された。この問題を解決するためのAWSミドルウェアの仕様改定が今後の課題である。

本研究は科研費（21500110）の助成を受けたものである。

参考文献

[1]伊東,塚本,高木,木村,大谷:AWSミドルウェアの研究 -アプローチと構成-, 情報処理学会第71回全国大会講演論文集, pp.1-509-510, 2009

[2]平本, 木村, 大友, 大谷:実応用を想定したAWSミドルウェアの評価, 情報処理学会第72回全国大会, 2010