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、昨年度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まだ行われていない。本研究では、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など実際面からの評価を行った。

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、昨年度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

AWS ミドルウェアの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 ミドルウェアの基本機能の評価のために、単純な BPM を持つ Simple01 および 02 を、ビジネスプロセスの分岐処理の機能の評価のために、分岐を持つ BPM の Fork01 及び 02 を、ビジネスプロセスのループ処理の機能の評価のために、ループを持つ BPM の Loop01 及び 02 を考案した。現在まで、Simple01 及び Fork01 の作成及びそれを使った実験が終わっている。

4. Simple01

(1)Simple01 の概要

図 1 に Simple01 のシステム間での処理の流れを示す。オペレーション Order は注文依頼を実行する。注文依頼では Ford という名前のフォーマットに従った電文を送信する。フォーマット

表 1　評価アプリケーション一覧

名前	ねらい	概要
Simple01	基本機能の評価	分岐およびループを持たない直線的なBPM
Simple02	基本機能の評価	直線的でより長いBPM
Fork01	分岐処理の基本形の評価	一つの分岐を持つ
Fork02	分岐処理を複数持つ形の評価	複数の分岐を持つ
Loop01	ループ処理の単純形の評価	１回のループを持つ
Loop02	ループ処理の基本形の評価	任意の回数のループを持つ

とは AWS で定義された用語でオペレーションが送受信する電文の形式を意味する。Ford は XML 形式で商品名と個数を含む。オペレーション Response は注文受付を実行し、フォーマット Fres の電文を受信する。Fres は価格と納期を持つ。最後にオペレーション Payment は支払通知を実行し、支払日時を含むフォーマット Fpay の電文を送信する。

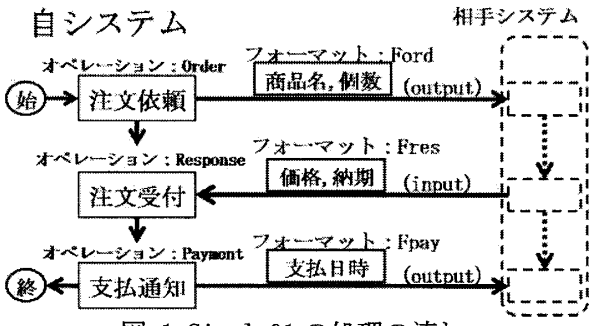

図 1 Simple01 の処理の流れ

(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１から作成した BPM を図２に示す。３つのオペレーションに応じて状態が０→１→２→３と変化する。ここで０は開始状態、３は終了状態である。

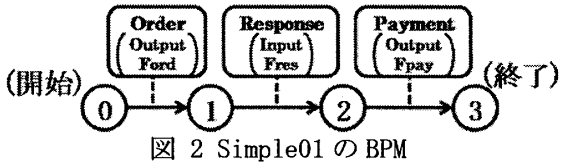

図 2 Simple01 の BPM

図 3 は AWS の規定に従って XML でこの BPM を記述したものである。3 つのオペレーションを持つ要素 operations と状態遷移情報を持つ要素 behavior で構成されている。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。

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

```
<operations>
  <operation name="Order" format="Ford">
          <pattern>output</pattern> </operation>
  <operation name="Response" format="Fres">
          <pattern>input</pattern> </operation>
  <operation name="Payment" format="Fpay">
          <pattern>output</pattern> </operation>
</operations>
<behavior>
  <states>
    <state no="0"> <next operation="Order">1</next> </state>
    <state no="1"> <next operation="Response">2</next> </state>
    <state no="2"> <next operation="Payment">3</next> </state>
    <state no="3"> </state>
  <states>
  <first>0</first>
  <last>3</last>
</behavior>
```

図 3　Simple01 の BPM の XML 記述

(3)プログラム本体

図 4 に Simple01 のアプリケーション本体 UserA を示す。UserA は apOrd()、apRes()、apPay()の 3 つのメソッドを持つ。それぞれは config によってオペレーション Order、Response、Payment に対応している。apOrd()は、データコンテナクラス O_ord に商品名(item)と個数(qty)を渡す。O_ord は受け取った item と qty を含む Ford 形式の XML 電文を生成する。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。同様に、apRes()は I_res から納期(date)と価格(price)を取り出し、apPay()は O_pay に支払日時(payDate)をセットする。図 4 のようにアプリケーション内では送受信の処理を実行する必要はない。またビジネスプロセスに従った処理の流れ制御も必要ない。これらは AWS ミドルウェアが BPM に従って自動的に実行してくれる。

```
public class UserA extends AWSFramework {
   public void apOrd(O_ord out) {
      out.item = 商品名; out.qty = 個数;}
   public void apRes(I_res in) {
      納期(in.date)と価格(in.price)を処理する}
   public void apPay(O_pay out) {
      out.payDate = 支払日時;}
}
```

図 4　Simple01 のアプリケーション本体(UserA)

(4)実験結果

Simple01 は正常に動作した。AWS ミドルウェアを使うと複雑なビジネスプロセスの流れ制御や非同期メッセージ通信を AWS ミドルウェアが行ってくれるのでアプリケーションをた易く開発できることが検証できた。また、本実験を通して BPM 表現の簡易化が必要なことも判明した[2]。

5. Fork01

図 5 に Fork01 の処理の流れを示す。Fork01 は 5 つのオペレーションからなる。Simple01 と異なり、途中(EstAns)で分岐が起こる。オペレーション EstAns(見積結果受付)はフォーマット AnsF の電文を受信し、電文の内容(見積結果)に応じて次の遷移先を決める。見積結果が妥当であれば Order(注文)にすすみ、妥当でなければ Cancel(注文中止)にすすむ。

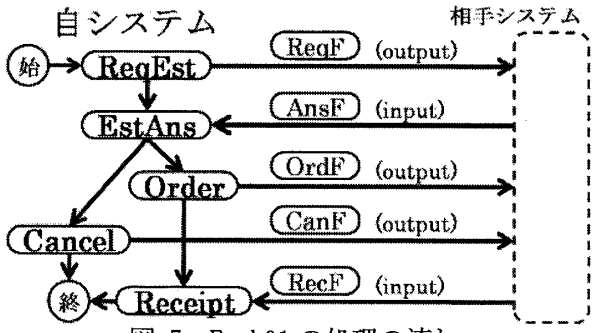

図 5　Fork01 の処理の流れ

アプリケーション本体(UserF01)で、ansAP()がオペレーション EstAns に対応する(図 6)。遷移先の指定は AWS ミドルウェアの API の一つである setNextOperation()で行う。見積結果が妥当だと判断し、Order に遷移先を指定するとき setNextOperation(“Order”)を実行し、見積結果が妥当でなく、Cancel に遷移先を指定するとき setNextOperation(“Cancel”)を実行する。

```
public class UserF01 extends AWSFramework {
 public void reqAp(O_ReqEst out){
   * * * * * * *
 public String ansAP(I_EstRes in){
   * * * * * * *
  Cancelに遷移するとき setNextOperation("Cancel");
  Orderに遷移するとき  setNextOperation("Order");
 }
  public void order (G_Order out) {
   * * * * * * * *
```

図 6　Fork01 のプログラム本体(UserF01)

6. まとめ

Simple01 により AWS ミドルウェアの基本的な利点が確認できた。更に Fork01 ではビジネスプロセスの流れの分岐制御をた易く行えることが検証できた。残りの評価アプリケーションの作成と評価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]大友, 伊東, 吉川, 大谷:AWS におけるビジネスプロセスモデル, 情報処理学会第 72 回全国大会, 2010