AWSにおけるビジネスプロセスモデル

大友 浩照† 伊東 正起† 吉川 恭平† 大谷 真†
湘南工科大学　情報工学科†

1. はじめに

AWS(自律型 Web サービス)[1]ではビジネスプロセスモデル(BPM)が制御の中心である。このため BPM の定義方法や XML による表現方法は AWS ミドルウェア開発を含め重要な検討項目となっている。従来の研究[2]では BPM は状態遷移表として表現されている。しかし無駄な要素があり複雑なモデルを表現するのに適していないとの問題点がある。本論文ではこの解決策として正規表現を用いた新たなBPM表現を提案することで、より簡潔で扱いやすい BPM 表現を実現する。またそれに応じ AWS ミドルウェア実装処理方式の改良点を述べる。

2. AWS

通常の Web サービスでは、サービス全体に対し BPM が事前定義されている。サービスに参加するシステムはこの BPM に従い構築されている。これに対し AWS では、各システムは独自の BPM を持ち、独自に開発・運用することができる。取引を行う際、互いの BPM を交換し協調変換することで、協調済 BPM を生成し、それに従いアプリケーションを実行し一連の取引を行う(図 1)[1][3]。

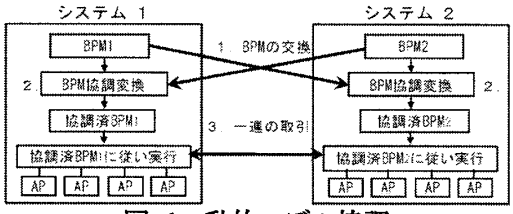

図 1　動的モデル協調

AWS における BPM とは、各システム独自の振舞いをオペレーション単位で定義したものである。図 2 は見積送信、結果受信、発注又はキャンセル、発注を行った場合は結果受信を行うオペレーションの順で取引を行う BPM の例である。

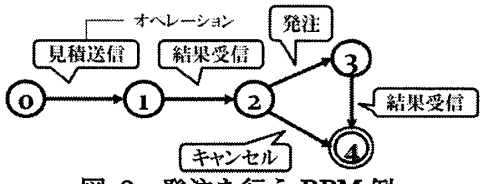

図 2　発注を行う BPM 例

3. 従来の BPM 表現

(1) BPM 表現

これまでの AWS ミドルウェアでは、BPM を状態遷移表の形でXMLを用いて表現していた[2]。これを状態遷移表型 BPM という。表 1 に示すとおりオペレーションの集合の定義の後に、状態の集合、各状態ごとにオペレーション実行後の遷移先状態、初期・終了状態を XML エレメントとして定義する。図 2 の BPM の記述例を図 3 に示す。

表 1　従来の BPM 表現

状態遷移表型BPMの形式定義	XMLの表現定義
BPM = {O, B}	BPM = <BPModel>～</BPModel>
O = オペレーション(op)の集合	O = <operations>～</operations>
op = 操作	op = <operation>～</operation> 属性:name ="オペレーション名" 属性:format="フォーマット名" 要素:<pattern>入出力パターン</pattern>
B ={S,s0, F}	B = <behavior>～</behavior>
S = 状態(s)の集合	S = <states>～</states>
s = {T}	s = <state>～</state> 属性:num ="状態番号" 要素:T
T =状態遷移	T = <next>～</next> 属性:operation="遷移オペレーション名" 要素:状態遷移先番号
s0 = 初期状態	s0 = <first>～</first> 要素:初期状態番号
F= 終了状態	F = <last>～</last> 要素:受理状態番号

```
<BPModel>
オペレーション集合記述部
<behavior>
<states>
  <state no="0">
   <next operation="見積送信">1</next>
  </state>
  <state no="1">
   <next operation="結果受信">2</next>
  </state>
  <state no="2">
   <next operation="発注">3</next>
   <next operation="キャンセル">4</next>
  </state>
  <state no="3">
   <next operation="結果受信">4</next>
  </state>
  <state no= "4"></state>
 </states>
 <first>0</first>
 <last>4</last>
</behavior>
</BPModel>
```

図 3　状態遷移表型の BPM 記述例

(2)問題点

BPM ではオペレーションの実行順序を規定することに意味がある。利用者の立場からはオペレーションだけを使い BPM を表現したい。しかし従来の表現方法では、状態はオペレーションの実行遷移を示す間接的なエンティティにもかかわらずそれを中心に定義しなければならない。例えば各状態の番号を定義しなければならず、初期・終

Business process model for AWS
†Hiroaki Otomo,Masaki Ito,Kyohei Yoshikawa,Makoto Oya,Shonan Institute of Technology

了状態を定義し、また全ての状態の遷移を定義しなければならない。このためモデル表現の記述が長くなり、複雑なモデルの設計が膨大なものになるとの問題点がある。

4. 新たに提案したBPM表現

(1) BPM表現

3(2)の問題を解決するため、正規表現の構文規則に対応したXML構文規則を定め、BPMを正規表現を用いて表現できるようにする。これを正規表現型BPMという。表現定義を表2に示す。

表 2 新たに提案したBPM表現

正規表現型BPMの形式定義	XMLの表現定義
BPM = {O, B}	BPM = <BPModel>～</BPModel>
O = オペレーション(op)の集合	O = <operations>～</operations>
op = 操作	op = <operation>～</operation> 属性:name ="オペレーション名" 属性:format="フォーマット名" 要素:<pattern>入出力パターン</pattern>
B = 正規表現<R>	B = <behavior>～</behavior>
アトム(実行する操作) <R> := <atom>	<atom/> 属性:operation="オペレーション名"
繰返し <R> := (<R>*) or <R> := (<R>?) or <R> := (<R>{min,max})	<repeat>～</repeat> 属性:min = "繰返し回数最小値" 属性:max = "繰返し回数最大値"
任意 <R> := ([<R>])	<optional>～</optional>
選択 <R> := (<R>|<R>)	<choice>～</choice>
結合 <R> := (<R>･<R>)	<sequence>～</sequence>

Oはオペレーション(op)を任意の数要素に持つ。opは属性としてオペレーション名、使用するメッセージフォーマット名、要素として入出力パターンを持たなければならない。Bは<behavior>で定義され、以下の要素によって正規表現を表現する。

・<atom>:実行するオペレーションを定義する。
・<choice>:要素から選択する処理を定義する。
・<repeat>:要素を一連の処理単位とし、最少回数、最大回数を指定し、繰返し処理を定義する。
・<optional>:要素の任意実行を定義する。
・<sequence>:要素を一連の処理単位とする。

<atom>以外の上記の要素は、お互いと<atom>を任意の数要素に含むことができる。正規表現型BPMによる図2のBPM記述例を図4に示す。

```
<BPModel>
オペレーション集合記述部
<behavior>
  <atom operation="見積送信" />
  <atom operation="結果受信" />
  <choice>
    <sequence>
      <atom operation="発注" />
      <atom operation="結果受信" />
    </sequence>
    <atom operation="キャンセル" />
  </choice>
</behavior>
</BPModel>
```

図 4 正規表現型のBPM記述例

(2)評価

新たに提案したBPM表現は従来のものより簡潔なものとなり、3(2)での問題点を解決できた。

5. AWSミドルウェア実装処理方式の変更

AWSで新たに提案したBPMを利用するには一度オートマトンへと変換する必要がある。一般にε遷移を含む非決定性有限オートマトン(ε-NFA)に変換するが、AWSではこのε遷移を含めることができない。[2]でもε遷移を含まない非決定性有限オートマトン(NFA)により処理を行っている。決定性有限オートマトン(DFA)に変換し利用する方法も考えられるが、この工程を伴う場合、BPMとして想定したオートマトンとは異なるものになってしまうことがある。

NFAに変換することで、新たに提案したBPMを利用できるようにする。一般に、ε-NFAからε除去を行いNFAを生成するが、その場合正規表現の長さにより処理に掛る時間は増える。そこで正規表現→NFAへ直接変換を行うことでε除去の計算を行わないアルゴリズムを実装することとする。

(1)変換アルゴリズム

ε遷移を含まないパターンを用意し、それに従い初期状態から順次、次の状態への遷移を生成する。用意するパターンは「選択・繰返し・結合」のパターンとする(図5)。

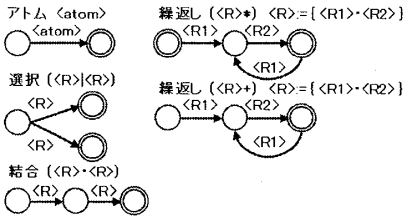

図 5 パターン

(2)結果

[2]と同じテストケースでテストした結果、同等のNFAが出力された。これにより変換アルゴリズムは正しく動作することが確認できた。

6. まとめ

正規表現を用いた新たなBPM表現方法を定義し、[2]のものと比べ評価を行った。正規表現→NFA変換アルゴリズムを実装しAWSミドルウェア実装処理方式の変更を行ない、新たに定義したBPMを利用できるようにした。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] 大谷,伊東,塚本,高木,木村,AWS(自律型Webサービス)とそのミドルウェア, 情報処理学会第71回全国大会講演論文集, pp.1-503-504, 2009

[2] 塚本,高木,木村,大谷,AWSミドルウェアの研究-動的モデル協調層-,情報処理学会第71回全国大会, pp.1-511-512, 2009

[3] 伊東,平本他,動的協調ミドルウェアの実装方法の研究, 情報処理学会第72回全国大会,2010