Webサービスフロー記述のモデル検査検証

中島 震[†,††]

Webサービスの実体は他と独立に自律並行実行するサーバであり，利用者は既存のWebサービスを連携させることで所望の機能を得ることができる．Webサービスを組み合わせるサービス・アグリゲーション記述が重要であり，Webサービスフロー記述言語が提案・公開されている．その代表例であるWSFL（Web Services Flow Language）はネット指向ワークフロー言語であり，構文的に正しくても大域的な不具合を生じる可能性を排除できない．モデル検査検証技法によってWSFL記述の大域的な振舞いを検証することができる．一方，データリンクの取扱いや検証という観点から見た場合にWSFLの操作的な意味には問題点がある．本稿では，WSFLの仕様を分析し不具合が発生する原因を探る．WSFLの考え方に従った範囲内で“保守的な”拡張を検討し，WSFL記述の検証を可能にする操作的な意味を提案する．Webサービスフロー記述の正しさ検証への応用という観点から，汎用モデル検査ツールの有用性について議論する．

Model-checking of Web Service Flow

Shin Nakajima[†,††]

Web service is an emerging software technology to use remote services in the Internet environment. Each Web service is an autonomous server ready to use. As the technology becomes pervasive, some “language” to describe Web service flows is needed to combine existing services flexibly. WSFL (Web Services Flow Language), one of the languages proposed for a standard, is a net-oriented flow language. This paper proposes to use the software model-checking technology for the verification of descriptions written in WSFL. The checker can find faulty behaviours inherent in WSFL descriptions that are syntactically correct. This paper reports some anomaly in the definition of WSFL operational semantics, and proposes a remedy. And the paper discusses the importance of model-checking tools in the study of Web service flow verification issues.

1. はじめに

インターネットの本格化とともに，地理的に分散しているサービスをネットワーク経由で利用するWebサービスが登場した[1)]．Webサービスは電子商取引き（EC）や電子政府などの新しいビジネスや業務形態の基盤として期待が大きい．一般に，Webサービスを利用するためには，所望のサービスを発見し，複数のサービスを組み合わせて連携させる．手軽に，既存のサービスを連携できることが本質である（サービスアグリゲーション）．そのためには，Webサービスだけでなく，Webサービス連携の記述方式が大切な役割を果たす．個々のWebサービスは他と独立に自律並行的に実行するため，連携の記述を分散協調システムと見なすことができる．

Webサービス連携記述は一種の分散協調システムであるため，正しい記述を作成することが難しい．現状では，連携記述の“実行時”にならないと不具合が発見できない．不具合があるWebサービス連携記述は，実行環境であるインターネットという公共資源を無駄に使うことになる．したがって，実行に先だってWebサービス連携記述の正しさを何らかの基準を用いて確認しておく必要がある[15)]．

本稿では，Webサービス連携記述の正しさを確認するための技術として，並行システムを対象とするモデル検査検証技術[3)]を用いることを提案する．具体的には，Webサービス連携記述言語として提案されているWSFL（Web Services Flow Language）[12)]の記述を対象として，モデル検査検証ツールSPIN[7)]を用いた検証方法を検討する．特に，WSFL[12)]はネット指向ワークフロー言語であるため，構文的に正しくても大域的な不具合を生じる可能性を排除できない．また，

† 法政大学
Hosei University
†† 科学技術振興事業団さきがけ21
PRESTO, JST

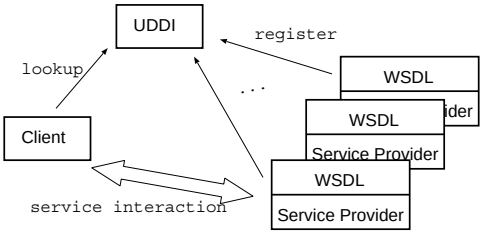


図 1 Web サービス
Fig. 1 Web service.

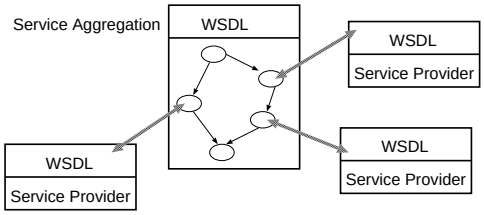


図 2 Web サービス・アグリゲーション
Fig. 2 Web service aggregation.

データリンクの取扱いや検証という観点から見た場合に WSFL の操作的な意味には問題点がある．WSFL の言語仕様を調査して不具合が発生する原因を探り，WSFL 記述の検証を可能にする操作的な意味を提案する．さらに，Web サービスフロー記述の正しさ検証への応用という観点から，汎用モデル検査ツールの有用性について議論する．

2. Web サービスフローと WSFL

2.1 サービス・アグリゲーションの検証

Web サービス利用が増加するとともに，Web サービスの諸側面を標準化することによって，さらに発展させる動きが活発になっている．**図 1** に Web サービスの典型的な利用形態を示した．UDDI（Universal Description, Discovery and Integration）[18] は 1 種のディレクトリサービスを提供するグローバルなサーバである．各サービス提供者は自身のサービスを利用するために必要なアクセス情報を WSDL（Web Services Description Language）[2] で表現し UDDI に登録する．利用者は UDDI に問合せを行って，要求に合致するサービスへのアクセス情報を得る．UDDI から得た WSDL 記述をもとに所望の Web サービスを利用する．WSDL は W3C で標準化が進められているサービスの記述形式であって，XML を用いた表現規約が決められている．

次に，仲介業の役割を果たす Web サービスの代表例として旅行代理店を考える．旅行代理店は，複数の飛行機会社，複数のホテル，さらにレンタカー会社や鉄道会社と連携して，利用者の希望に合った旅程を具体化して必要な予約や発券業務を行う．個々の飛行機会社やホテルの 1 つ 1 つを独立した Web サービス提供者と見なすと，旅行代理店は，**図 2** に模式的に示したようなサービスアグリゲーションを行う．自身も適切な WSDL を外部に公開することで，UDDI へ登録したり，利用者からのアクセスを受け付けたりする．

Web サービスの面白さは，既存 Web サービスを組み合わせて，手軽に，図 2 のようなサービスアグリゲーションを実現できることである．しかし，サービスアグリゲーションは，独立に作動している複数の Web サービスを連携させることであり，連携には，データならびに制御の流れを明示しなければならない．分散協調システムの記述言語が必要となる．実際，WSFL（Web Services Flow Language）[12] と XLANG [17] といった Web サービスフローの連携記述言語が提案されている．すなわち，サービスアグリゲーションを，WSFL あるいは XLANG の “プログラム” として作成する．しかし，分散協調システムを記述することになるため，従来の逐次的なプログラム作成に比べて難しさが増すという問題点がある．たとえば，並行実行している Web サービスの起動制御処理の記述がデッドロックに陥るなどの不具合を考えることができる．

ここで，Web サービス連携記述に不具合が混入している場合を考える．現状の WSFL や XLANG が想定している範囲では，連携記述の “実行時” になってはじめて不具合が判明する．不具合のある Web サービス連携記述は，ソフトウェア工学の立場から好ましくない．さらに，実際に，インターネット上で実行させると，従来は表に出てこなかった次のような問題がある [15]．

(1) 誤り個所に至るまでに行ったサービス実行が無駄になり，Web サービス提供者の実行結果をロールバックする必要があるなど影響範囲が広い．
(2) 不要なネットワークトラフィックの発生によりインターネットという公共資源を無駄に使うことになる．

したがって，実行に先だって Web サービス連携記述の正しさを何らかの基準を用いて確認しておく必要がある．

2.2 WSFL

WSFL（Web Services Flow Language）は Leymann が提案した Web サービスフロー記述言語である [12]．構文と操作的な意味を紹介する．

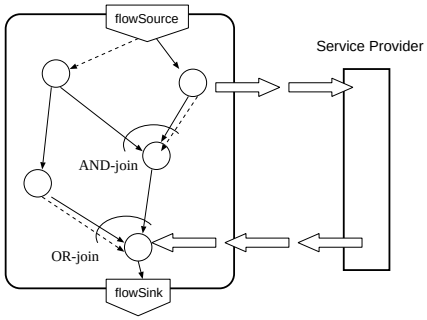


図 3 WSFL の基本要素
Fig. 3 WSFL basis.

2.2.1 WSFL の概要

WSFL は Leymann らが約 10 年前に研究していた PM-flow と呼ぶネット指向仕様記述パラダイムに従ったワークフロー記述言語モデル[11]を基にしている．WSFL は WSDL の振舞い拡張であり，メッセージ定義など WSDL が提供する概念を用いる．また，WSFL の具体的な構文は WSDL と同様に XML で定義されている．

図 3 は WSFL の中心となるフローモデルの模式的な例である．各 Web サービスの起動事象をアクティビティと呼びワークフローグラフのノードに，アクティビティ間のデータフローと制御フローとをリンクに対応させる．アクティビティでは，外部のサービス提供者への処理起動や値の引き取りなどの計算処理を記述する．

図 3 から分かるように，フローモデルは開始点を示す `flowSource` から始まり `flowSink` で終了するグラフである．制御の流れを表現するために，アクティビティには `AND-join` や `OR-join` などの合流条件を付加することができる．

WSFL のフローモデルは，CPN（Coloured Petri Nets）[8]と同様な手法，すなわち，実行状況を表すスナップショット☆の変化によって操作的な意味を付与している．操作的な意味に従って，WSFL の “インタプリタ” を構築することができる．

フローモデルは，それ自身が WSDL を定義することができ，外部の他フローモデルのアクティビティからアクセスされる．なお，WSFL は，フローモデルのほかに，複数のフローモデルを結合して大きな処理にまとめるグローバルモデルがある．グローバルモデルは構造的な側面の表現手段であるため以下の議論からは省略する．

☆ CPN ではマーキングと呼ぶ情報であって，実行スレッドを表すトークンが存在するプレースのマルチセットで表現する．実行とはマーキングの変化のことである．

2.2.2 基本要素と操作的な意味

XML を用いた具体的な構文と操作的な意味について説明する．

第 1 に，WSFL のデータリンクは，固有の名前（`name`）ならびにリンク元（`source`）とリンク先（`target`）のアクティビティ名を属性として持つ．図 3 のようなダイアグラム表現では破線で示す．

```
<datalink name=''...''
          source=''...'' target=''...''/>
```

第 2 に，WSFL の制御リンクは，データリンクが持つ属性に加えて，遷移条件を表現する 2 つの属性を持つ．`transitionCondition` は遷移条件を論理式で表す．また，リンク元アクティビティが `result` 指定メッセージを生成したときに遷移することを示す．図 3 のようなダイアグラム表現では実線で示す．

```
<controlLink name=''...''
   source=''...'' target=''...''
   transitionCondition=''...''
   result=''...''/>
```

第 3 に，WSFL のアクティビティは，固有の名前（`name`）と終了条件（`exitCondition`）を属性として持つ．アクティビティは多くの構成要素を持ち，入力データ（`input`）と出力データ（`output`）の生成方法，合流条件（`join condition`），実行主体（`performedBy`），実行主体への入力データ整形（`materialize`），などがある．また，具体的な実行主体の決定方法は `implement` で指定する．

```
<activity name=''...''
          exitCondition=''...''>
  <input name=''...'' ... />
  <output name=''...'' ... />
  <performedBy ... />
  <implement>  ...  </implement>
  <join condition=''...'' ... />
  <materialize ... />
</activity>
```

属性として終了条件が指定された場合，条件が成立したときに実行主体の起動を終える．逆に，条件が成立しない限り実行主体の起動を繰り返し，ループを表現する☆☆．

WSFL のフローモデルは上記の制御リンク，データリンク，アクティビティからなる．操作的な意味は，その時点でのリンク値から発火可能（実行可能）なア

☆☆ WSFL ではフローがサイクルを持つことはない．

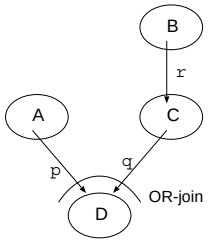


図 4 デッド経路除去
Fig. 4 Dead-path elimination.

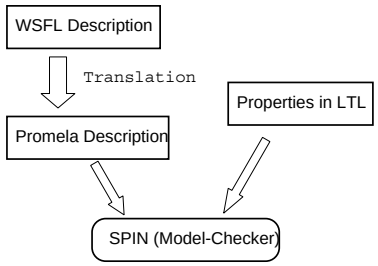


図 5 検証の過程
Fig. 5 Verification process.

クティビティを求め，発火可能なアクティビティを実行する手順からなる．ただし，すべての入力制御リンクの値が確定して，はじめて合流条件を評価する．したがって，以下の 2 つのフェーズからなる繰返しサイクルで処理が進行し，発火可能なアクティビティがなくなると当該フローモデルの実行が終了する．

(1) 発火可能なアクティビティの決定
- 入力となる制御リンクの真偽値，当該アクティビティの合流条件，などから決定．
- 条件が偽になる場合も以下に述べる DPE 処理を行うことで見かけの処理停止を除去．

(2) アクティビティの実行
- 入力データリンクからの値引き取り，外部サービス提供者の起動などによる処理，など．
- 上記処理の結果，関連する出力リンクの値を更新．

特に，発火可能なアクティビティが同時に複数あってもよく，逆に，これが並列性の源になっている．

2.2.3 デッド経路の除去

WSFL の操作的な意味を複雑にしている要因の 1 つに DPE（Dead-Path Elimination）がある．**図 4** は仕様書 12) から引用した例であり，DPE の必要性を説明する．

図 4 において，アクティビティ A の実行結果として `p` が *true* で，B の結果は `r` が *false* とする．`r` が *false* であるから，C は実行されることはない．そのため，D の `join condition` も決して評価されない．ところが，D の条件は OR であり，さらに，p が *true* であることから条件が成立する．すなわち，論理的には D は実行可能であるが，`join condition` が評価対象にならないことから実行に至らない．

WSFL の DPE は，このような見かけの障害状態を除去する手段である．DPE は，`join condition` が *false* となるか，あるいは入力制御リンクを 1 つだけ持つアクティビティがあり，その制御リンクの遷移条件が *false* になる場合に下流方向に除去処理を進める．また，次の合流アクティビティあるいは終端の `flowSink` に到達したときに DPE 伝播を終了する．

3. モデル検査ツールの利用

WSFL は，先に述べたように，ネット指向仕様記述パラダイムに従ったワークフロースキーマ記述言語の 1 種である．したがって，WSFL の検証を，ワークフロースキーマやビジネスフローあるいはビジネスプロセスの検証[5),9),14)]と同様な方法で実現できる．すなわち，ソフトウェアに対するモデル検査検証の技術[3)]を用いる．本稿では，SPIN [7)]を用いるが，他モデル検査検証ツールに置き換えることは難しいことではない．

3.1 検証方法の概要

図 5 は検証方法の概要を示す．第 1 に，WSFL 記述を SPIN の入力仕様言語である Promela 記述に変換する．第 2 に，当該連携記述に固有の性質を時相論理の式として表現する．第 3 に，SPIN を用いて，当該の連携記述がデッドロックなどの不具合を持たないこと，さらに，時相論理式として表現したアプリケーション固有の仕様を満たすことを確認する．SPIN はモデル検査検証ツールであるため，この確認作業を自動化することができる．

なお，本稿で扱う検証とは，WSFL フローモデルの範囲，特に，リンクに沿った情報の流れについて形式化した振舞い仕様についてである．メッセージや実際のサービス提供者の計算結果などのデータ値やタイプに関するチェックは範囲外である．WSFL の記述対象が本質的に分散協調システムであるため，制御やイベント送受の流れに着目した振舞い検証の重要性が高いと考えたためである．データ値やタイプに関する検証は別の手段で扱わなければならない．

3.2 変換方法

WSFL 記述のモデル検査検証を行うために SPIN の入力仕様言語 Promela に変換する．この変換は WSFL の操作的な意味を Promela のチャネル通信オートマ

トンにコード化することである．以下，詳細な変換方法を議論する．

第 1 に，次のような対応関係で，WSFL 記述を表現することとする[16]．

- 制御ならびにデータリンクを Promela のチャネルに対応させ，
- WSFL アクティビティを Promela プロセスに対応させる．

本稿では，チャネルはリンクを伝播する値とその値の送信元アクティビティに対応する Promela プロセスに与えた識別子の 2 つを引数とした．識別子は冗長であるが，SPIN が出力するモデル検査結果を解析する際に補助的な情報となる．

```
chan LinkName = [NBUF] of { short, short };
```

次に，WSFL アクティビティに対応する Promela プロセスのスケルトンを示す．

```
proctype ActivityName ( chan inA1, outB1, ... )
{
    local variable declarations
    activity body
}
```

本 Promela プロセスは当該アクティビティの入力および出力リンクに対応する Promela チャネルを仮引数として持つ．ハードコーディングを避けたことになり，具体的な入出力リンク名が何であるかを気にする必要がない．また，WSFL ではアクティビティは接続するリンクの情報を持たないで，リンク側に接続情報を記述する方針と一致している．一方，具体的な接続関係は，すべての Promela プロセスを初期化する `init` プロセスに記述することになる．別の言葉では，`init` プロセスがすべてのトポロジ情報を持つことになる．これによって，Promela 記述の可読性が向上すると考えている．

Promela プロセスは内部の処理記述に必要な局所変数定義の後，アクティビティの操作的な意味を忠実に反映した処理本体を持つ．処理本体を構造化し 6 つのステップに分けた．第 1 の `waitLoop:` は入力制御リンクに対応するチャネルにメッセージが到着するのを待つ．入力制御リンクを複数持つ場合はすべてのリンクの値が確定するまで待ち，その後，2 番目の `joinStep:` にうつる．以下は，チャネル `inA1` が入力制御リンクの場合の記述断片である．

```
waitLoop:
 do
  :: ((s1 == NIL) && inA1?[x1,FromSourceName])
      -> inA1?x1,FromSourceName;
      atomic{ s1 = nonNIL;
          if
            :: all the control channels have messages
                  -> goto joinStep
            :: else -> skip
          fi
      }
  ...
 od;
```

第 2 の `joinStep:` は入力制御リンクの値をもとに，指定の合流条件（`join condition`）を評価する．2.2.3 項で述べた DPE を実現するために，制御リンクは `CFORCED` を伝播する．`CFORCED` は合流条件の計算に際しては論理偽（`CFALSE`）と同じ扱いとする．

```
joinStep:
    if the message is CFORCED,
      then it is changed to CFALSE;
    if
    :: <join condition> becomes true  -> c = CTRUE
    :: else -> c = CFALSE
    fi;
```

一方，WSFL 記述が合流条件を持たない場合は `joinStep:` について合流条件の評価部を省略して，DPE の場合に下流方向に `CFORCED` を伝播するだけでよい．

```
joinStep:
    if the message is CFORCED, then c = CFORCED;
```

上記処理の結果，合流条件が成立する場合，本アクティビティが発火する．第 3 の `materializeStep:` では入力データリンクから値を引き取る．さらに，WSFL の `<materialize>` に記述された内容からデータに対する適切な処理を書き込む．以下はチャネル `inA2` が入力データリンクである場合の記述断片である．

```
materializeStep:
  if
   :: (c == CTRUE) -> inA2?x2,FromSourceName2; ...
          execution body for <materialize>
   :: else -> skip
  fi:
```

当該アクティビティの発火による実行では，指定サービスプロバイダ起動などを行う必要があり，第 4 の `performStep:` で関連処理を記述する．また，exit 条件を用いた繰返し処理も表現する．下記断片では exit 条件の成立と不成立とが非決定的に起こるという一般的な場合になっている．

```
performStep:
  if
   :: (c == CTRUE) ->
       progressPerform:
       execution body according to <implement>;
      if
        :: goto callExit
        :: goto progressPerform
      fi;
       callExit:
       epilogue code;
   :: else -> skip
```

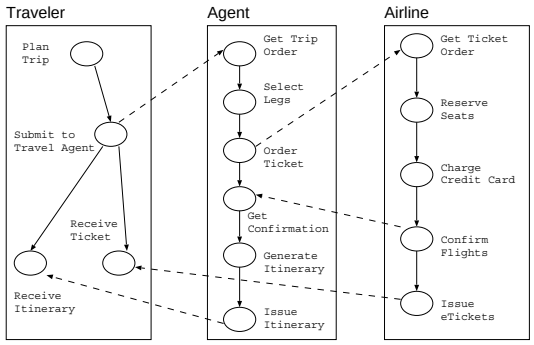


図 6 航空券購入の例題
Fig. 6 Ticket order example.

```
fi:
```

当該アクティビティの実行終了後，出力側の制御ならびにデータリンクに流す値を決定する必要がある．5 番目の `controlStep:` に記述する．一般的なテンプレートでは，WSFL 記述に対する網羅的な振舞いを調べるために，出力制御リンクの値は `CTRUE` と `CFALSE` を非決定的に選択するように表す．次節で紹介する**図 6** の例題のように，制御リンクの値が必ず真になる WSFL 記述の場合は，非決定性のない記述を与える．また，DPE によって下流に値伝播する場合の値もここで決める．以下では，変数 `y1` がある出力制御リンクの値であって論理値を非決定的に設定し，変数 `y2` が決まった値を持つデータリンクの値を保持する場合である．なお，DPE による値伝播の場合は変数 `y1` に`CFORCED` を設定しておく．

```
controlStep:
  if
   :: (c == CTRUE) -> atomic{
          if :: y1 = CTRUE :: y1 = CFALSE fi;
          y2 = some definite value; ...
        }
   :: else -> atomic{ y1 = CFORCED; ... }
  fi:
```

最後に，`controlStep:` で設定した値を，実際に出力チャネルにメッセージとして流すことで，本 Promela プロセスの処理を終了する．以下はチャネル `outB1` が出力制御リンクである場合の記述断片を示す．

```
propagateStep:
  atomic{
    outB1!y1,From ActivityName; ...
  }
}
```

ところで，WSFL 記述の `<performedBy ... />` が外部サービスプロバイダを起動している場合，Promela プロセスのスケルトンは，起動や返却値引き取り処理を持つ必要がある．本処理の詳細な情報は `<implement> ... </implement>` で与えられる記述に従って作成する．たとえば，起動と値の引き取りを同期的に行う処理の場合，次のようなチャネルへのメッセージ送信（`agent!...`）と受信（`agentReturn?...`）の組として表現することができる．

```
agent!VALUE,TRIP_ORDER,FromTraveler;
agentReturn?y,TRIP_ORDER,FromAgent;
```

なお，外部サービスプロバイダとの情報交換を行う Promela チャネルは，このメッセージ送受信形式に合わせて，以下のような情報から構成されるとした．

```
chan ProviderName = [NBUF]
        of { short, short, short };
```

すなわち，WSFL のサービスプロバイダは複数のポートを持ち，各々が複数の操作を受け付けることができる．チャネルが受け渡すメッセージは，送受信データ，操作とポートの指定，送信元アクティビティの識別子，の 3 つの情報からなる☆．

3.3 検証の例

図 6 は WSFL 仕様書 12) から引用した例題の模式図で，以下の検証作業を説明するための具体例である．これは，旅行予約に関する例題であって，`Traveler`，`Agent`，`Airline` の 3 つの WSFL フローモデルから構成される．旅行者（`Traveler`）が旅行代理店（`Agent`）に旅程作成ならびに航空券購入を依頼する．`Agent` は航空会社（`Airline`）に航空券を手配する．`Airline` は `Agent` には予約確認を返し，航空券を `Traveler` に直接送付する．`Agent` は `Airline` からの予約確認を受け取った後，旅程を作成し `Traveler` に送付する．`Traveler` は航空券と旅程の双方を同時並行的に受け取る．この例題での制御は単純で，`Agent` と `Airline` はともに無条件で逐次実行し，`Traveler` は 2 つの返事を並行して待つ．そのため，制御に関連する不具合が起きる可能性は低い．

仕様書記載の WSFL 記述は半完成であるが，これを適切に補ってシナリオを完成させた．次いで 3.2 節に示した変換方法に準じて人手変換により Promela 記述を得た．この過程で，本例題での制御の流れが逐次的な流れで決定的に発生することを利用して Promela 記述を単純化した．3 つのフローモデルを合計すると Promela 記述の規模は約 700 行である．

性質の検証は次のステップで行う．

(1) 単独フロー記述のチェック
3 つのフローモデルを個々にチェックする．デッドロックの有無や処理が `flowSink` に到達することのチェック（`[](<> flowSink)`）を行う．他フロー（サービス提供者）に処理依頼や処理待

☆ 3 番目の識別子は “デバッグ” 用情報である．

ちをしている場合，関連する正しい機能を “環境” として与えてチェックする．

(2) 統合フロー記述のチェック
3つのフローモデルを統合した全体のフローをチェックする．デッドロックの有無を判定する．さらに，検証対象に特有な性質を LTL（Linear Temporal Logic）式として表現しチェックする．

次に，簡単な不具合を例にとってモデル検査検証の効果を説明する．図 6 のフローについて，`IssueItinerary` メッセージを生成しないような誤った振舞いを持つ `Agent` を作成したと仮定する．この場合，単独フロー記述のチェックでは，`Agent` は処理を正常に終えるため不具合を発見できない．一方，統合フロー記述では `Traveler` が `IssueItinerary` メッセージ待ちになり処理を進行しない．すなわち，デッドロックとして不具合を発見することができる．

また，LTL で表現する仕様として，たとえば，`Traveler` の場合は次のような性質がある．

```
[](SubmitToTravelAgent ->
    (<>ReceiveTicket && <>ReceiveItinerary)
```

この式は，`SubmitToTravelAgent` があると，将来のある時点（`<>`）で `ReceiveTicket` があり，かつ，別のある時点（`<>`）で `ReceiveItinerary` が起こる，ことを示している．将来起こると期待している 2 つのイベントは同時に起こってもよい．

なお，図 6 の例は制御が単純であるため DPE が起きない．また，SPIN は状態空間を探索する際に検証に必要な遷移と状態だけを選択的に求める．そのために，検証する性質が成り立つか否かによって探索空間が大きく変わる．しかし，デッドロックがないことの確認を行うためには状態空間すべてを展開しなければならない．3 つのフローモデルを統合した全体について，SPIN は状態数が約 28 万で遷移数が約 47 万の状態空間に対して最大 195 の深さの探索を行い不具合がないことを確認した．一方，構成するフロー単独は状態空間が小さい．たとえば，`Airline` 単独では状態数 201 で遷移数が 586 の状態空間に対して深さ 122 で検証が終わった．

なお，現状の CPU パワーや主記憶容量で SPIN を用いたモデル検査を実行する場合，100 万状態を超えると実行効率が低下することが経験的に知られている．そのため，個別フローならびに統合フローの検証に分けて，ボトムアップに小さいフローモデルから確認する方法を採用すると全体的な作業の効率が良いと考えられる．

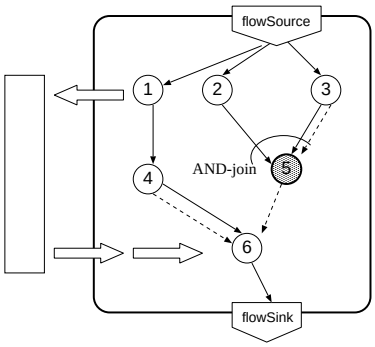


図 7 デッドロックとなるフローモデル
Fig. 7 Deadlocked flow model.

4. 操作的な意味の見直し

WSFL では単純な例であっても制御リンクとデータリンクを柔軟に設定したフローが意図どおりの振舞いを示さないことがある．本章では WSFL の問題点をまとめる．次いで，検証可能な操作的な意味を与えることを目標として，正しい意図を反映したフローモデルを作動させるための解決案を検討する．

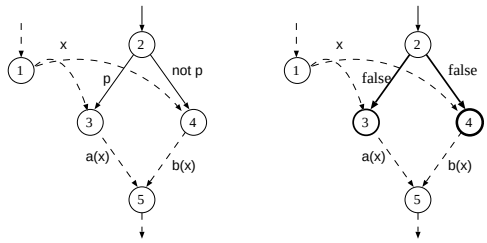
4.1 値確定の伝播と値伝播

図 7 に，WSFL 仕様書記載の構文的な規則に従って作成したフローモデルであっても，実行上の不具合を回避できない例を示す．本フローモデルは，1 から 6 までの 6 つのアクティビティを持つ．アクティビティ5 は 2 と 3 からの制御リンクに対して `AND-Join` の制御を持ち，6 への出力となるデータリンクを持つ．アクティビティ5 は，双方の制御リンクが確定しかつ $true$ の場合に実行可能となる．実行終了後，確定した値を出力データリンクに流す．アクティビティ6 は 4 と 5 からのデータリンクの値から何らかの処理を行う．

ここで，アクティビティ5 のいずれかの入力制御リンクが $false$ になり，その結果，`AND-join` も $false$ になる場合を想定する．当然，アクティビティ5 は実行されないため，出力データリンクの値は確定しない．アクティビティ6 は 5 からのデータリンク値が確定するのを待つ状態で止まる．前章で述べたモデル検査ツールを利用する場合はデッドロック状態として検出することができる．

図 7 に示したように，たとえば，アクティビティ1 で外部のサービスプロバイダに処理依頼し 6 で値を引き取る場合，アクティビティ6 が実行できないことは，外部サービスプロバイダにも影響を与えることになる．すなわち，本フローに閉じない深刻な不具合を引き起こす．

先に述べた DPE 処理を考慮しても，このデッド



(a) 正しい例 (b) 誤りのある例

(a) Valid (b) Invalid

図 8 条件分岐式
Fig. 8 Conditional expressions.

ロックは解消しない．アクティビティ5 の `AND-join` が $false$ になるという条件で，DPE を開始する．しかし，DPE は制御リンクを下流方向に，$false$ 値を強制的に流すだけである．決して，アクティビティ5 の出力データリンクに適切な値を設定するものではない．データ値はアクティビティの実行によってはじめて確定する．

次に，WSFL の不具合を説明する最も簡単な例として以下の条件分岐式を考える．

```
if p then a(x) else b(x) endif
```

図 8 に WSFL フロー例を示した．(a) でアクティビティ2 の出力制御リンクについて `p` が $true$ になる場合を想定する．アクティビティ3 が実行可能になり 5 の入力データリンクに所望の計算結果が流れることを期待する．また，アクティビティ4 は発火可能とならないためその出力データリンクの値が定まらない．条件分岐式の意図から明らかなように，アクティビティ3 と 4 のいずれかが発火可能になり実行する．いずれか確定した出力データを引き取る合流ノードの役割を担うのがアクティビティ5 である．一方，WSFL の操作的な意味では，アクティビティ5 は制御リンクを持たないため発火可能とならない．

次に，図 8 (b) のようにアクティビティ2 の出力制御リンクが同時に $false$ になる場合は，アクティビティ5 は決して実行されない．WSFL の操作的な意味では，アクティビティ3 と 4 とが発火可能でないため，両者から下流に DPE 処理が発生する．ところが，DPE は制御リンクをたどって進むため，データリンクによってつながっているアクティビティ5 には情報が伝わらない．すなわち，アクティビティ5 は発火可能にならないし，DPE によって排除されることもない．その結果，WSFL の操作的な意味では，両方ともアクティビティ5 が発火可能にならないという点で図 8 の (a) と (b) とを区別することができない．

図 7 や図 8 の例から，次のことが分かる．すなわち，正しいフロー記述の上では，制御リンクとデータリンクが密接に関係する．しかし，WSFL では，表層的には両者を独立に設定可能な言語要素としているため，不具合を持つフロー構造が構文的に許されることが誤り混入の原因である．

操作的な意味の観点からは，値伝播と値確定の伝播（メタレベルの伝播）が混乱していることに原因がある．先に述べた DPE は，見かけのうえで発火不能なアクティビティを発火可能にする機構である．$false$ を伝播するような記載があるが，その本質は，未確定の論理値を強制的に伝播し，発火可能なアクティビティを求めることにある．すなわち，DPE は $false$ という具体的な値伝播の形をとって，強制的な値確定というメタレベルの伝播を実現しているといってよい．しかし，データリンクに沿ったメタレベルの伝播については何も規定していない．発火可能なアクティビティの入力データリンクはすべて値が確定することを仮定している．

4.2 解 決 案

データリンクに沿って DPE と同様な処理を行う方法として一般化 DPE を導入する．特に，WSFL 記述の検証という観点からの操作的な意味と解析手法を検討した．

- 開始の条件（WSFL の既存 DPE 開始条件）
  合流アクティビティの合流条件が $false$
  OR ただ 1 つの入力制御リンクが $false$
- 終了の条件（WSFL の既存 DPE 終了条件）
  下流の次の合流アクティビティに到達
  OR フローモデルの終端（`flowSink`）に到達
- 処理の本体：値確定を強制伝播
  制御リンクの場合，$false$ と解釈し，アクティビティの発火可能性を計算
  OR データリンクの場合，$NULL$ と解釈

上記によって，いくつかの入力データリンクが $NULL$ 値を持つアクティビティが制御の観点から発火可能と判定されることがある．これを NULL アクティビティと呼ぶ．本稿で提案する操作的な意味では，$NULL$ 値を適切に解釈（値が未確定）して NULL アクティビティを実行することとする．記述の解析という観点からは，$NULL$ 値の到達を不具合発生と解釈すればよい．

上記の操作的な意味を図 8 にあてはめる．図 8 (a) の場合，先ほどと同じ条件下で，アクティビティ4 は発火不能となるため一般化 DPE によってアクティビティ5 に $NULL$ 値が伝播し，5 は発火可能となる．ア

クティビティ5は条件分岐の合流ノードであるため，実行時に *NULL* 値を無視して，アクティビティ3からの値を選択する．図8 (b) の場合，アクティビティ3と4から開始した一般化 DPE によってアクティビティ5の2つの入力ともに *NULL* 値が伝播する．このとき，制御の流れからアクティビティ5は発火可能である．条件分岐の合流ノードであるにもかかわらずすべての値が未確定であるため，記述の解析という観点で不具合と判断し，デッドロック状況として報告する．

次に，図7の例に一般化 DPE を適用する場合を考える．アクティビティ6は制御リンクの流れから発火可能になり，実行時に，アクティビティ5からの *NULL* データをチェックすることで不具合を検出することができる．

上記の一般化 DPE の機能を実現するために，先に述べた Promela プロセスの記述を変更する必要がある．以下に，主な変更点をまとめる．

(1) `waitLoop`: では制御リンクだけではなくデータリンクを含むすべての入力リンクに値が確定することを待つ．

(2) `materializeStep`: では入力データリンクからの値引き取り処理が不要になるため，次のように修正する．

```
materializeStep:
   if
    :: (c == CTRUE) ->
        execution body for <materialize>
    :: else -> skip
   fi:
```

(3) `controlStep`: では DPE の開始条件が満たされた場合，出力データリンクに対して，*NULL* 値を伝播するように設定する．以下の断片では，y1が出力制御リンクで，y2が出力データリンクの場合を示した．

```
controlStep:
   if
    :: (c == CTRUE) -> atomic{
           if :: y1 = CTRUE :: y1 = CFALSE fi;
           y2 = some definite value; ...
          }
    :: else -> atomic{
           y1 = CFORCED; y2 = NULL; ...
          }
   fi:
```

5. 考　　察

WSFL ならびにもとになった PM-graph の考え方を振り返ると，データリンクをきちんと取り扱っていないことが理解できる．WfMC で標準化されているワークフロースキーマ言語 WPDL [19]などを見ても，一般にワークフローではアクティビティ間の制御の流れを重要視する．データをアクティビティの外部で保持管理することを暗黙のうちに想定しているワークフロースキーマ言語が多い．WSFL の場合に即して説明すると，図3の模式図にあるように，上流アクティビティで外部サービスプロバイダを起動する．その計算結果を下流のアクティビティで獲得する．2つのアクティビティ間では逐次実行などの制御の流れを明示するだけでよいとする立場である．

一方，WSFL ではアクティビティ間のデータリンクを明示的に導入した．WSFL 仕様の作成者はデータリンクの使い方を暗黙に限定しているようである．すなわち，共有データを保持するデータプールの役割を果たすアクティビティを設け，当該アクティビティから利用側アクティビティに出力データリンクを設定する．この場合，データリンクから値を引き取る際，データはすでに確定しているため，本稿で扱ったような不具合は発生しない．

ところが，WSFL の仕様ではデータリンクと制御リンクの設定方法を制限する規則が明示されていない．そのために，データ値が確定しているか否かによって，アクティビティの処理が進行できるか否かが変わる．すなわち，不具合が混入する問題点が生じたといえる．

理想的には，構文的に正しい記述であれば，大域的な不具合を持たないフローモデルを作成できることを保証できるとよい．しかし，WSFL が基礎におくネット指向ワークフロー記述のモデルで，これを保証することは難しい．ネット指向記述よりも高いレベルの並列処理記述言語を考案する必要がある．

一方，本稿では，WSFL 仕様の考え方をもとに一般化 DPE の方法を検討した．これによって，現状の WSFL 仕様では不具合となるフロー記述についても記述者の意図どおりの操作的な意味を与えることが可能な場合が増えることを議論した．本稿での議論の中心は，一般化 DPE が万能であるということでも新規性があるという点でもない．現状の WSFL の操作的な意味ならびに一般化 DPE による意味のどちらも Promela で表現する方式を示すことで，モデル検査検証ツールによる確認が可能になるようにしたことである．特に，WSFL はネット指向ワークフロー言語であるため，構文的に正しくても大域的な不具合を生じる可能性を排除できない．また，データリンクの取扱いや検証という観点から見た場合に WSFL の操作的な意味には問題点がある．WSFL の仕様を分析し不具合が発生する原因を探り種々の考察を行うためにモデル検査検証ツールを用いることで，検討作業が効率化

できたと考えている．

モデル検査ツールの役割という観点から考察をまとめる．当初，モデル検査検証ツールSPIN[7]をWSFLの仕様を分析理解する補助ツールとして使った[16]．その後，改訂案を検討する際にアイデア実験のテストベッドツールとして利用することで，複雑にからむ並行処理やDPE方式の改訂実験が容易に行えた．WSFL記述からSPINの入力仕様記述を（半）自動生成するツールと組み合わせることで“WSFL記述専用の振舞いチェッカ”の内部エンジンとして利用することができる．WSFL記述が不具合を持つか否かを確認するために，モデル検査検証技術に基づく大域的な振舞いチェックツールによる記述の事前チェックが必須である．このようなツールが実現可能であることを3.2節で変換方法を示すことで議論した．

なお，本稿では，リンクに沿った情報の流れについて形式化した振舞い仕様を対象としている．WSFLの記述対象が本質的に分散協調システムであるため，制御やイベント送受の流れに着目した振舞い検証の重要性が高いためである．データ値やタイプに関する検証は本稿の範囲外である．振舞いならびにデータ値やタイプについての検証を統合して考察することは今後の研究課題としたい．

6. 関連研究

WSFLはネット指向並行計算記述モデルである．従来より，ペトリネット，データフロー，などの計算モデルが知られている．また，WSFLはPM-flowをベースとしてXMLによる表層構文を与えたワークフロースキーマ記述言語である．ワークフローの分野ではWfMC（Workflow Management Coalition）がワークフロープロセス定義言語の標準案を議論している．以下，デッドロックに関連する制御の側面について比較検討する．

ペトリネットは，構造的な性質によってクラス分けされている[13]．状態機械，マークグラフ，自由選択ネットなどがある．状態機械やマークグラフは競合に起因するデッドロックがない．一方，自由選択ネットはデッドロックの可能性がある．また，データフローモデルと同型であることが知られている[10]．

データフロー計算モデル[6]は長い研究の歴史があるため，バリエーションが豊富である．基本的なモデルでは6種類の細粒度の基本アクタがある．基本アクタを組み合わせて条件分岐などの制御の側面を表現することができる．データフローのダイアグラム表現により直観的な記述が可能である．一方，基本アクタの組合せ方法によってはデッドロックが発生する．関数プログラミングのパラダイムを採用した上位言語を提供することで不具合を持つ構造を避ける．

WfMCでは，WPDL（Workflow Process Description Language）と呼ぶネット指向ワークフロースキーマ言語を定義している[19]．基本要素はアクティビティと遷移リンクである．遷移リンクは本質的に制御リンクである．また，アクティビティには，SPLIT-JOINがあり，適切な性質を与えることで，条件分岐を表現することができる．ところが，デッドロックの取扱いはWPDLの仕様範囲外としている．デッドロックがないようなワークフロースキーマを設計することが前提である．デッドロック検知あるいは解消はWPDL仕様を満たす具体的なシステムが実現する機能であるとして仕様から外している．したがって，本稿で述べたようなチェックツールが別途必要である．

本稿をまとめる過程で，BPEL4WS（Business Process Execution Language for Web Services）が公表された[4]．WSFL[12]とXLANG[17]の経験から考案された新しい仕様である．BPEL4WSでもアクティビティの並行性と同期を表現するためにフローの考え方（<flow>）を踏襲している．リンク（<link>）は制御を表現するものとし，データの受渡しは大域的に宣言されたコンテナ（<container>）を経由する．条件分岐などの制御は特別な意味を持つアクティビティ（<switch>）で表現する．結果として，本稿で指摘したWSFLフローモデルが持つ問題点に言及した仕様になっている．ところが，仕様書[4]は操作的な意味を明確に規定していない．例題による説明で終わっている．また，大域的な共有資源であるコンテナを導入したために共有変数の参照制御に絡む問題が発生する．本稿のモデルチェッカを用いる手法が厳密な意味定義を検討するための方法論として有効であると考えられる．

7. おわりに

本稿では，モデル検査検証ツールSPINを用いてWSFLの仕様を分析することで，不具合を持つWSFL記述を排除することが難しいことを指摘した．次いで，条件分岐のような明らかに正しいWSFL記述を実現できるような操作的な意味を検討した．記述の事前検証と相性のよい意味として一般化DPEと呼ぶ方式を報告した．現状のWSFLの操作的な意味ならびに一般化DPEによる意味のどちらもPromelaで表現する方式を示すことで，モデル検査検証技術に基づく大域的な振舞いチェックツールが実現できることを報告し

た．また，モデル検査検証ツールが，WSFL 仕様の調査・理解，改訂案の検討などを行う過程で実験ツールとして有用であった．

参 考 文 献

1) 青山幹雄：ソフトウェアサービス技術へのいざない，情報処理，Vol.42, No.9, pp.857–862 (2001).
2) Christensen, E., Curbera, F., Meredith, G. and Weerawarana, S.: Web Service Description Language (WSDL), W3C Web Site (2001).
3) Clarke, E., Grumberg, O. and Peled, D.: *Model Checking*, The MIT Press (1999).
4) Curbera, F., Goland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S. and Weerawarana, S.: *Business Process Execution Language for Web Services* (*v1.0*) (Aug. 2002).
5) Eertink, H., Janssen, W., Luttighuis, P.O., Teeuw, W. and Vissers, C.: A Business Process Design Language, *Proc. FME FM'99*, pp.76–95 (Sep. 1999).
6) Filman, R.E. and Friedman, D.P.: *Coordinated Computing*, McGraw-Hill (1984). 雨宮直人，尾内理紀夫，高橋直久（共訳）：協調型計算システム，マグロウヒル (1986).
7) Holzmann, G.J.: The Model Checker SPIN, *IEEE Trans. Soft. Engin.*, Vol.23, No.5, pp.279–295 (1997).
8) Jensen, K.: *Coloured Petri Nets 1*, Springer-Verlag (1992).
9) Karamanolis, C., Giannakopoulou, D., Magee, J. and Wheater, S.M.: Model Checking of Workflow Schemas, *Proc. IEEE EDOC 2000*, pp.170–179 (Sep. 2000).
10) Kavi, K.M., Buckles, B.P. and Bhat, U.N.: Isomorphism Between Petri Nets and Dataflow Graphs, *IEEE Trans. Soft. Engin.*, Vol.13, No.10, pp.1127–1134 (1987).
11) Leymann, F. and Altenhuber, W.: Managing Business Processes as an Information Resource, *IBM System Journal*, Vol.33, No.2, pp.326–348 (1994).
12) Leymann, F.: *Web Services Flow Language* (*WSFL 1.0*), IBM Corporation (May 2001).
13) 村田忠夫：ペトリネットの解析と応用，近代科学社 (1992).
14) 中島　震：CCS によるビジネスフロー図の検証，日本ソフトウェア科学会 第 1 回 FOSE，pp.135–140 (Dec. 1994).
15) Nakajima, S.: On Verifying Web Service Flows, *Proc. SAINT 2002 Workshop*, pp.223–224 (Feb. 2002).
16) Nakajima, S.: Verification of Web Service Flows with Model-Checking Techniques, *Proc. Cyber World 2002*, pp.378–385, IEEE (Nov. 2002).
17) Thatte, S.: *XLANG — Web Services for Business Process Designm*, Microsoft Corporation (May 2001).
18) UDDI Web Site: UDDI Techinical White Paper (Sep. 2000). http://www.uddi.org
19) Workflow Management Coalition: Interface 1: Process Definition Language Process Model, WfMC TC-1016-P (v1.1) (Oct. 1999).

(平成 14 年 8 月 13 日受付)
(平成 15 年 1 月 7 日採録)

中島　震（正会員）

昭和 56 年東京大学大学院理学系研究科物理学専攻修士課程修了．同年 NEC 入社．同社コンピュータ技術本部，研究開発グループ主管研究員を経て，平成 14 年より法政大学教授．平成 13 年より科学技術振興事業団さきがけ 21「機能と構成」領域研究員を兼務．分散ソフトウェア工学，形式仕様と検証，オブジェクト指向技術，組み込みソフトウェアに関する研究に従事．学術博士（東京大学）．平成 13 年度より本学会ソフトウェア工学研究会幹事．日本ソフトウェア科学会，ACM 各会員．