ソフトウェア工学　139－10
(2002. 10. 11)

WSFLの記述パターンとデザインチェッカ

中島　震†

† 法政大学/科技団さきがけ 21
E-mail: †nkjm@i.hosei.ac.jp

あらまし　インターネットの本格化と共に登場したWebサービスは他と独立に自律並行実行する実体である．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サービスを組み上げるサービス・アグリゲーションに関心が集まり，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，データリンクの取り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の考え方にしたがった範囲内で“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．また，モデル検査検証ツールSPINについて，WSFL仕様の分析作業の中での使い方ならびに，WSFL記述チェッカのエンジンとしての利用方法を紹介する．
キーワード　Webサービス，ワークフロースキーマ，デッドロック解析，モデルチェッキング

Patterns in WSFL Descriptions and Validation Checker

Shin NAKAJIMA†

† Hosei University/PRESTO, JST
E-mail: †nkjm@i.hosei.ac.jp

Abstract　Web service is an emerging software technology to use remote services in *the Internet*. 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 the SPIN model-checker for the verification of descriptions written in WSFL. Since WSFL is a net-oriented specification language, any WSFL descriptions, though syntactically correct, sometimes show faulty global behaviours. Such faulty descriptions can be detected by means of the proposed method. Further, the paper reports some anomaly in the definition of the WSFL operational semantics in regard to the handling of dataflows, which was identified during the experiments on using the model-checker. The paper presents a remedy to the anomaly as well as discussions on the role of the model-checking technology for use in the behavioural analysis of Web service flows.
Key words　Web Service, Workflow Schema, Deadlock Analysis, Model-Checking

1. はじめに

インターネットの本格化と共に，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．個々のWebサービスは他と独立に自律並行実行する実体である．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[1]．そこで，Webサービスを組み上げるサービス・アグリゲーションに関心が集まり，WSFL[11]やXLANG[17]といったWebサービスフロー記述言語が提案・公開されている．

ところが，Webサービス連携記述に不具合がある場合，現状の技術では“実行時”にならないと不具合が見つからない．実行環境のインターネットという公共資源のトラフィックを無駄使いする結果となる．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[14]．特に，WSFL[11]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．ワークフロースキーマへのモデル検査検証技法の応用[4][8][13]と同様な考え方で，WSFL記述の大域的

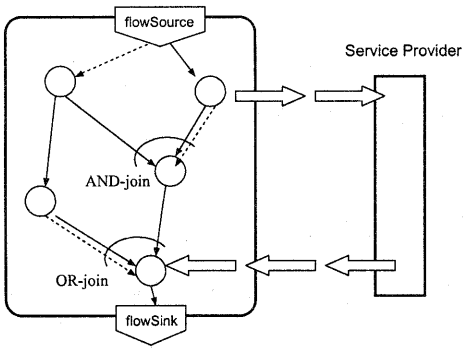

図 1　WSFL の基本モデル
Fig. 1　WSFL Basis

な振舞い検証が可能であることを示した [15] [16].

一方, データリンクの取り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. また, WSFL の考え方にしたがった範囲内で "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. また, 文献 [15] [16] で提案した方法をもとに, モデル検査検証ツール SPIN [5] の利用方法について述べ, 汎用モデル検査ツールの有用性を報告する.

以下, WSFL の概要を紹介し不具合を持つ WSFL 記述例を示す. ついで, WSFL の問題点が値確定の伝播と値の伝播の扱いに起因することを述べ, 2 つの解決案を議論する. 特に, 本稿で一般化 DPE と呼ぶ方法が, WSFL 記述の検証を可能にする操作的な意味という視点で優れていることを述べる. 最後に, ネット指向パラダイムに基づく計算モデルの関連研究との比較を行なう.

2. WSFL について

仕様書 [11] にしたがって WSFL の言語仕様を紹介する. WSFL の構文は XML で定義されているが, その操作的な意味は, PM-flow に基づく. WSFL 仕様書に加えて PM-flow の論文 [10] 記載の操作的な意味を補って説明する.

2.1 WSFL の概要

WSFL は, F. Leymann が提案した Web サービスフロー記述言語である. F. Leymann たちが約 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したがったワークフロー記述言語モデル [10] 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 が提供する概念を利用する. また, WSFL の具体的な構文も WSDL と同様に XML で定義されている.

WSFL の中心となるのはフローモデルであり, 図 1 にフローモデルの簡単な例を模式的に表した. 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 1 からわかるように, 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. 制御の流れを表すために, アクティビティには AND-join や OR-join 等の合流条件を付加することができる. このようなフローモデルは, 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.

WSFL のフローモデルは, CPN (Coloured Petri Nets) [6] 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(注1)の変化によって操作的な意味を付与している. 操作的な意味にしたがって, WSFL の "インタプリタ" を構築することができる.

WSFL は, フローモデルの他に, 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.

2.2 基本要素と操作的な意味

XML 文書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 1 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` の指定メッセージを生成した時に遷移することを示す. 図 1 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`), 等がある. また, 具体的な実行主体の決定方法は `implement` で指定する. 属性として終了条件が指定された場合, 条件が成立した時に実行主体を起動することをやめる. 逆に, 条件が成立しない限り実行主体の起動を繰り返す. すなわち, ループを表現する.

```
<activity name=''...'' exitCondition=''...''>
  <input name=''...'' ... />
```

(注1)：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.

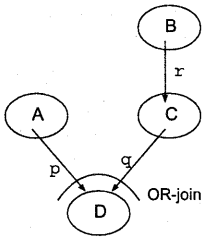

図 2　デッド経路除去の例
Fig. 2　Example of Dead-Path Elimination

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

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, 第一にある時点でのリンク値から発火可能 (実行可能) なアクティビティを求め, 第二に発火可能なアクティビティを実行する手順からなる. したがって, 以下の 2 つのフェーズからなる繰り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 join 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 処理を行な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, ServiceProvider 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.

2.3　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ひとつに Dead-Path Elimination (DPE) がある. 図 2 は仕様書 [11] から引用した例であり, DPE の必要性を説明する.

図 2 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.

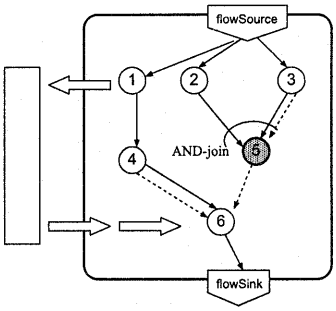

図 3　デッドロックするフローモデル
Fig. 3　Deadlocked Flow Model

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ひと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 join アクティビティあるいは終端の flowSink に到達した時に DPE 伝播を終了する.

2.4　不具合を持つ WSFL 記述例

図 3 に, WSFL 仕様書記載の構文的な規則にしたが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なう.

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, 出力の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の値が確定するのを待っている状態で止まってしまい, デッドロック状態となる. 図 3 に示したように, 例えば, 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不具合の源となる.

先に述べた DPE 処理を考慮しても, このデッド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種類のものである.

図 3 のようなフローモデルは構文的には正しい WSFL 記述であるが, 大域的な解析によって不具合の存在がわかる例である. 実際, 文献 [15] [16] に示した方法に従ってモデル検査ツー

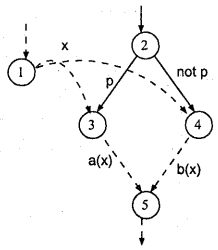

図4 条件式
Fig. 4 Conditional Expression

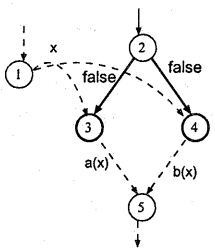

図5 不具合のある条件式
Fig. 5 "Buggy" Conditional

ルSPIN [5] を用いることで不具合をデッドロックとして検知することができる. 一方, この問題は, WSFLの操作的な意味が, 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していることを暗黙のうちに仮定していることに起因する. また, このようなフローモデルを排除する規則がないことが問題である.

3. WSFLの問題点と解決案

本節ではWSFLの問題点をまとめる. ついで, 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2つの解決案を示し比較検討する.

3.1 値確定の伝播と値伝播

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4にWSFLフロー例を示した. ここで, 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. アクティビティ4が発火可能とならないため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評価対象とならない.

次に, 図5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, アクティビティ5は決して実行されないことが正しい振舞いである. 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は制御リンクを辿って進むため, 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.(注2). 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, 図4と図5とを区別することができないことになる.

図3, 図4, 図5の例から, 以下がわかる. すなわち, 制御リンクとデータリンクが正しいフロー記述の上では密接に関係するにも関わらず, 表層的には両者を独立に配置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を記述でき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, 見かけの上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することにある. 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良い. 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していることを仮定しているだけである.

3.2 解決案

前節で述べたように, 図4のような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について, 意図通りの振舞いを示さない. 一方, DPEは強制伝播という点で解決の手立てを提供するように思える. 本節では, DPEの考え方を利用する方法を考察する.

a) 制御を伴うデータリンク

第1に, DPEが制御リンクに対して有効に働くことを利用した方法として, 制御の流れを必ず伴うようなデータリンクを導入することが考えられる(注3).

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

図4を参照する. アクティビティ1から3, 3から5, などのデータリンクを上記の `dataWithControlLink` に置き換える. 5はOR-joinを持つjoinアクティビティとする. また, 3から5 (4から5) については, 発火した場合に制御リンクにのせる論理値を *true* とする. これによって, アクティビティ5は, 2の出力結果に依存した正しい入力データ値を選択することが可能になる.

次に, この方法を図5に適用した場合を考察する. AND-joinのjoinアクティビティ3と4が共に *false* となるためDPEを

(注2): プログラム依存グラフの観点からいうと, アクティビティ2が5の支配節であるという情報を使えない, ということ.
(注3): オリジナルWSFLではデータリンクが常に制御リンクを伴うようなフローモデルを作成する "お作法" で対応することになる.

開始する. DPEによってアクティビティ5 (OR-join) に *false* 値が双方から伝播し, 5も発火不能となる(注4). WSFLの操作的な意味によると, アクティビティ5は実行されず, その結果, 当該フローモデルは正常終了する(注5). すなわち, 図5が持つ論理的な不具合を検知することができない.

b) 一般化DPE

ネットワーク環境下での実行に先立つ記述の検証時には不具合を検知できるような操作的な意味が必要である. 以下, リンクについてはWSFLの仕様通り2種類のものを採用するが, データリンクに沿ってDPEと同様な処理を行な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考察する.

- 開始の条件 (WSFLの既存DPE開始条件)
  - Joinアクティビティのjoin条件が *false*
  - ただ一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)
  - 次のJoinアクティビティに到達
  - フローモデルの終端 (sinkFlow) に到達
- 処理の本体 : 値確定を強制伝播

(1) 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

(2) データリンクの場合, *NULL* と解釈するが, アクティビティの発火可能性は制御リンクの計算方法を適用

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しても良い.

上記の操作的な意味を図4と図5に当てはめる. 図4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5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つの入力共に *NULL* 値が伝播する. この時, 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関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れば良い.

次に, 図3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れば良い.

c) 考　　察

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することが理想的である. 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の操作的な意味を分析し, WSFL記述の書き方を整理して記述パターンを集める方針を採用する. 記述パターンとしては, 条件分岐を特殊例として含む多重分岐 (switch文) や並行fork-joinなどがある(注6). WSFLを作成する利用者はこれらの記述パターンを参考にして所望の振舞いを示すフローを作成する. ついで, 作成したフローの大域的な振舞いの妥当性をモデル検査検証技術で確認するという方法が, 標準提案されているWSFLの利用という観点から好ましいと考える.

WSFLならびに基になったPM-graphの考え方を振り返ると, WSFLがデータリンクに対して, 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 [18] など, 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. 図1の模式図にあるように, データは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ようである. すなわち, 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. 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.

4. 関連研究

2つの観点, 第1にWSFLの仕様という観点, 第2にモデル検査ツールの役割という観点, から議論する.

第1に, WSFLはネット指向の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. 特に, 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 [12]. 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. 自由選択ネットはデータフローモデルと同型であることが知られている [9]. デッドロックの可能性がある.

(注4) : その結果, 5を起点とする新たなDPEが開始される.
(注5) : 他に発火可能なアクティビティがないとする.

(注6) : この2つはフローグラフの構造は同じであるが処理の意図が異なる. 大域的な解析により意図を確認する.

データフロー計算モデル [7] 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ーがある. 基本アクターを組み合わせて 条件分岐 を表現する. 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, 基本アクターの組み合わ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 [18]. アクティビティと遷移リンクが基本要素である. 本質的に遷移リンクは制御リンクである. 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, デッドロックについては, WPDL の仕様範囲外である. 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. また, 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.

第 2 に, モデル検査ツールの役割という観点から考察する. 当初, モデル検査検証ツール SPIN [5] を, 上に述べたように, WSFL の仕様を分析理解する補助ツールとして使った [15] [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なえた. WSFL 記述から SPIN の入力仕様記述を自動生成するツールと組み合わせることで "WSFL 記述専用チェッカ" の内部エンジンとして利用することができる. これは, 現在, 試作中である.

本稿の作成中, BPEL4WS (Business Process Execution Language for Web Services) が公表された [3]. WSFL [11] と XLANG [17] の経験から考案された. 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. 条件分岐等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 [3] 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っている. 本稿の手法が意味定義を与えるための方法論として有効であると考えられる.

5. おわりに

本稿では, モデル検査検証ツールを用いて 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. ついで, 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考案することを目的として, 2 つの方式を検討した. 記述の事前検証と相性の良い意味として一般化 DPE と呼ぶ方式が優れていることを報告した. WSFL は標準案ではないが広く公開されている仕様である. 公開仕様に対して具体的な不具合の指摘と改善提案を行ない, 特に, WSFL 記述の検証を可能にする操作的な意味を与えたことが本稿の成果である. なお, 一般化 DPE の方法は, WSFL だけでなく, 現在デッドロックを仕様の中で考察していない WPDL を改訂する際にも良い指針を与えると考えている. データフローの取り扱い方法によってはデッドロックの可能性が表面化するからである.

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, pages 857–862, September 2001.

[2] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Service Description Language (WSDL). W3C Web Site, 2001.

[3] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services (v1.0). `http://www.ibm.com/developerworks/library/ws-bpel/`. August 2002.

[4] H. Eertink, W. Janssen, P.O. Luttighuis, W. Teeuw, and C. Vissers. A Business Process Design Language, In *Proc. FME FM'99*, pages 76–95, September 1999.

[5] G.J. Holzmann. The Model Checker SPIN. *IEEE Trans. Soft. Engin.*, vol.23, no.5, pages 279–295, May 1997.

[6] K. Jensen. *Coloured Petri Nets 1*. Springer-Verlag, 1992.

[7] R.E. Filman and D.P. Friedman. *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.

[8] C. Karamanolis, D. Giannakopoulou, J. Magee, and S.M. Wheater. Model Checking of Workflow Schemas. In *Proc. IEEE EDOC 2000*, pages 170–179, September 2000.

[9] K.M.Kavi, B.P.Buckles, and U.N.Bhat. Isomorphism Between Petri Nets and Dataflow Graphs. *IEEE Trans. Soft. Engin.*, vol.13, no.10, pages 1127–1134, October 1987.

[10] F. Leymann and W. Altenhuber. Managing Business Processes as an Information Resource. *IBM System Journal*, vol.33, no.2, pages 326–348, 1994.

[11] F. Leymann. Web Services Flow Language (WSFL 1.0). IBM Corporation, May 2001.

[12] 村田 忠夫. ペトリネットの解析と応用. 近代科学社 1992.

[13] 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, pages 135–140, December 1994.

[14] S. Nakajima. On Verifying Web Service Flows. In *Proc. SAINT 2002 Workshop*, pages 223–224, February 2002.

[15] 中島 震. WSFL を用いた Web サービスフロー記述の自動検証技法. 情報処理学会ソフトウェア工学研究会, July 2002.

[16] S. Nakajima. Verification of Web Service Flows with Model-Checking Techniques. In *Proc. CW 2002*, IEEE, to appear, November 2002.

[17] S. Thatte. XLANG – Web Services for Business Process Design. Microsoft Corporation, May 2001.

[18] Workflow Management Coalition. Interface 1: Process Definition Language Process Model. WfMC TC-1016-P (v1.1), October 1999.