ソフトウェア工学 138－3
（２００２．７．１１）

WSFLを用いたWebサービスフロー記述の自動検証技法

中島 震
法政大学 経営学部 / 科学技術振興事業団 さきがけ21

あらまし　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。手軽に既存のサービスを連携できることがWebサービスの本質である。そのためには、Webサービスだけでなく、連携記述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実行する実体であり、連携記述は分散協調システムと見做すことができる。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。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を用いる方法を提案する。特に、WSFL (Web Services Flow Language) の記述を対象としてSPINを用いた検証実験の方法を示す。モデル検査検証技術がWebサービス連携の検証ツールとして利用できることを報告する。

キーワード　Webサービス、形式手法、振舞い仕様、モデルチェッキング

Verification of Web Service Flows written in WSFL

Shin NAKAJIMA
Hosei University and PRESTO21, JST

Abstract　Web service is an emerging software technology to use remote services in the Internet. As it becomes pervasive, some “language” to describe Web service flows is needed to combine existing services flexibly. The flow essentially describes distributed collaborations and is not easy to write and verify, while the fault that the flow description may have can only be detected at runtime. The faulty flow description is not desirable because a tremendous amount of publicly shared network resources are consumed. The verification of the Web service flow prior to its execution in the Internet is mandatory. This paper proposes to use the software model-checking technology for the verification of the Web service flow descriptions. For a concrete discussion, the paper adapts WSFL (Web Services Flow Language) as the language to describe the Web service flows, and uses the SPIN model-checker for the verification engine. The experiment shows that the software model-checking technology is usable as a basis for the verification of WSFL descriptions.

Keywords　Web Service, Formal Methods, Behavioural Specifications, 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[1][15]。Webサービスは電子商取り引き(EC)や電子政府などの新しいビジネスや業務の形態として期待が高い。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做すことができる。

ところが、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。現状の技術では、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[12]。

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)[10]の記述を対象として、モデル検査検証ツールとしてSPIN[5][6]を用いた検証実験の方法を提案する。モデル検査検証技術がWebサービス連携の検証ツールとして利用可能であることを報告する。

2 Webサービスアグリゲーション

Webサービス利用が増加すると共に、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きている。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)[14]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の必要な情報をWSDL (Web Services Description Language)[2]で表現しUDDIに登録する。利用者はUDDIに問い合わせを行なって、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を用いた表現既約が決められている。

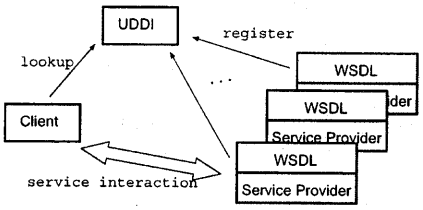

図 1. Web Service

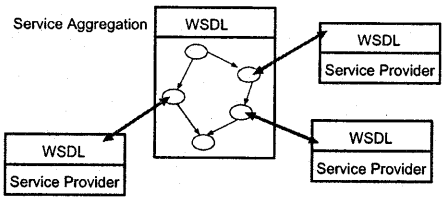

図 2. Web Service Aggregation

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なう。個々の飛行機会社やホテルのひとつひとつを独立したWebサービス提供者とみなすと、旅行代理店は、図2の模式的に示したようなサービスアグリゲーションを行なう。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)[10]

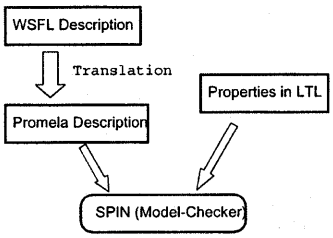

図 3. Verification Process

と XLANG[13] 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”として作成することができる。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の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等が考えられる。

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”になって初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、従来にない、次のような問題がある [12]。

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。

3 検証の方法

3.1 概要

本稿では、ソフトウェアに対するモデル検査検証の技術 [3] を用いて、Web サービス連携記述を形式検証する方法を提案する。技術的な詳細を議論するために、Web サービス連携記述言語として WSFL[10] を、モデル検査検証ツールとして SPIN[5][6] を用いる。

図 3 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等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的に行なうことができる。

WSFL は、次節で紹介するように、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の 1 種である。したがって、WSFL の検証とは、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 [4][8][11] と同様な方法で実現できる。なお、XLANG は式ベースの仕様記述パラダイムを採用した言語であり、時間やトランザクションなどの多様な概念を持つ。そのため、本稿の方法をそのまま XLANG に適用することは難しいと考えられる。本稿では、SPIN を想定するが、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。

3.2 形式化

3.2.1 WSFL

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 [9] を基にしている。WSFL は WSDL の振舞い拡張であり、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

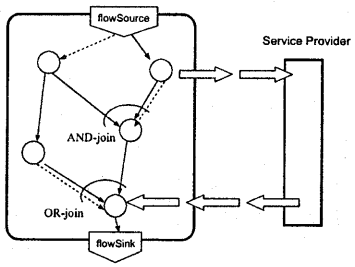

図 4. WSFL Basis

4 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。図 4 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。制御フローを表すために、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。また、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。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)[7] と同様な方法、すなわち、実行状況を表すスナップショット[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。

3.2.2 Promela/SPIN

SPIN[6] は、G.J.Holzmann たちによって開発され公開されているオートマトンベースのモデル検査検証ツールである。Promela と呼ぶ入力仕様言語を提供し、検証の対象システムを、チャネル通信オー

[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。

```
#define  NBUF    1
#define  MSG0    10
#define  MSG1    11
#define  ACK0    12
#define  ACK1    13

chan  sender   =[NBUF] of { short };
chan  receiver =[NBUF] of { short };

proctype Sender()
{ short any;
again:
  do
   :: receiver!MSG1 ->
      if :: sender?ACK1 -> break
         :: sender?any
         :: timeout
      fi
 od;
 do
  :: receiver!MSG0 ->
     if :: sender?ACK0 -> break
        :: sender?any
        :: timeout
     fi
 od;
 goto again
}

proctype Receiver()
{ short any;
again:
  do
   :: receiver?MSG1 -> sender!ACK1; break
   :: receiver?MSG0 -> sender!ACK0
   :: receiver?any
  od;
P0:
  do
   :: receiver?MSG0 -> sender!ACK0; break
   :: receiver?MSG1 -> sender!ACK1
   :: receiver?any
  od;
P1: goto again
}

init { atomic { run Sender(); run Receiver() } }
```

図 5. Promela Example

トマトン群として表現することを可能にする。図5にPromelaで記述したABPの例[5]を示す。

図5の例は、Senderと Receiverと名付けられた2つの非同期通信オートマトン(Promelaプロセス)と対象システム全体を初期化する特別なプロセスinitからなる。SenderとReceiverはsenderとreceiverと呼ぶバッファ長1のチャネルを用いて情報を交換する。情報は全てshort型のデータであり、可読性を向上させるために記号(#define)を導入した。チャネルを用いたデータの送受信はCSPと同様な構文で表される。すなわち、receiver!MSG1のような!構文が送信を、sender?ACK1にあるような?構文が受信を示す。

SPINは、非同期チャネル通信を行なうPromelaプロセス群をBüchiオートマトンに変換し、内部表現に対して網羅的な探索を行なうことで、デッドロック等の不具合を発見する。また、SPINは、LTL(Linear Temporal Logic)として表現された論理式をBüchiオートマトンに変換し、検証対象Büchiオートマトンとの積に対する探索によって、与えたLTL式が成り立つか否かを判定する。

3.2.3 変換テンプレート

次に、WSFL記述をPromelaに変換する手法を議論する。以下、WSFLを表現した具体的なXML文書と対応するPromela構文の関係を示す変換テンプレートの概要を示す。

前提として、Promelaで表現するための枠組を次のように考える。

- WSFLアクティビティをPromelaプロセスに対応
- WSFL制御リンクとデータリンクをPromelaチャネルに対応

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。

```
<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。

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

冒頭で述べた方針により、リンクひとつに対してPromelaチャネルをひとつ対応させる。今回の実験では、short型の2つの引数を持つようにした。第1引数はリンクを伝播する値を、第2引数は値の生成元となるアクティビティの識別子を持たせた[2]。

```
chan <CName> = [NBUF] of { short, short };
```

アクティビティは次のようなPromelaプロセスに変換する。

```
proctype <AName> (<Formal Parameters>)
{
 <Wait Loop>
 <Join Step>
 <Materialize Step>
 <Perform Step>
 <Call Exit Step>
 <Control Step>
 <Propagate Step>
}
```

[2]変換時の不用意なPromela記述への不具合混入を検知することが目的であり、WSFLの変換に本質的なものではない。

当該アクティビティの全ての入出力リンクを引数として与える方針を採用した。これにより、プロセスとチャネルを分離することができ、可能であればプロセス記述が再利用しやすくなる。逆に、プロセス生成時に、次の init の記述例に示すように、具体的なチャネルを与える必要がある。言い替えれば、init を見れば、WSFL フローモデルのトポロジーを把握できるようにした。

```
init {
  atomic{ run <AName>(<CName>, ...);
          ...                          }
}
```

他要素の説明は以下の通り。

- <Wait Loop>

  引数として与えられた入力チャネルをチェックして全ての入力リンク情報が確定することを待つ。

- <Join Step>

  制御リンクの情報を読みだし、合流条件が成立するかを判定する。成立する場合に以下の処理を行なう。

- <Materialize Step>

  データリンクの情報を読みだし、実行主体への入力データ整形を行なう。

- <Perform Step>

  実行主体を起動する。起動先サービス提供者のチャネルに処理起動に必要な情報を送り、必要に応じて、起動結果の値を受け取る。WSFL 記述での指定方法に応じて、手続き呼出しの場合、一方向メッセージ送信の場合、非同期通知引き取りの場合などがある。

- <Call Exit Step>

  実行主体の結果情報と終了条件を比較して、繰り返し処理を続けるか否かを判定する。

- <Control Step>

  出力となる制御リンクに流す情報を決定する。WSFL の制御リンクに与えられた遷移条件に関する処理も行なう。

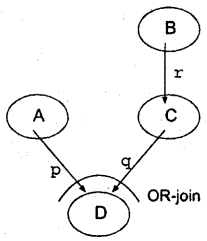

図 6. Dead-Path Elimination

- <Propagate Step>

  引数として与えられた全ての出力リンクに適切な情報を伝播させる。

WSFL 記述で<providedBy>に他サービス提供者名が指定される場合、上記の<Perform Step>で当該フローモデルに処理依頼あるいは値引き取りなどの処理を行なう。WSFL ではサービス提供者は複数のポートを持ち、各ポートが操作を定義している。Promela 記述では、交換するデータ値、サービス提供者名とポート名と操作名をコード化した情報ならびに送信プロセス識別子を引数として持つ通信チャネルを割り当てた。

```
chan <ChName> =
     [NBUF] of { short, short, short };
```

3.2.4 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 6 は仕様書から引用した例であり、DPE の必要性を説明する。図 6 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ひとつだけ持つアクティビティがあり、その制御リンク

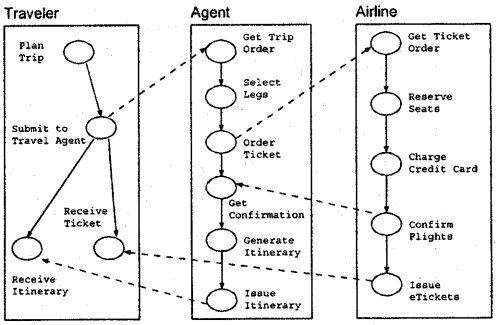

図 7. Ticket Order Example

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。

本稿の Promela 記述では、DPE を正しく扱うために、`join condition` の評価対象となる論理値を拡張して forced を導入した。すなわち、DPE の開始条件が満たされると制御リンクに沿って forced を伝播する。forced は論理値としては false であるが、さらに下流に DPE を継続する時は forced を伝播する[3]。先の Promela プロセスの構成要素中、`<Join Step>`と`<Control Step>`に対して、forced 値の処理ならびに下流への伝播処理を記述する。

3.3 性質検証

図 7 は WSFL 仕様書 [10] 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。仕様書記載の WSFL 記述は半完成であるが、これを適切に補い、前節の変換手順にしたがって Promela 記述を得た。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なう。

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))`) を行なう。他フロー (サービス提供者) に処理依頼や処理待ちをしている場合、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行なう。さらに、検証対象に特有な性質を LTL 式として表現しチェックする。

図 7 の `Agent` が `IssueItinerary` メッセージを生成しないという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。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)
```

なお、図 7 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が、デッドロックがないことの確認を行なうためには全ての状態空間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 の深さの探索を行ない障害がないことを確認した。一方、構成するフロー単独で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 で検証が終った。

3.4 考察

WSFL 仕様書 [10] の大部分はメッセージのタイプや WSDL との関係を説明した箇所である。本稿のでは 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振舞い検証を行なった。すなわち、メッセージや実際のサービス提供者の計算結果等の値やタイプに関するチェックを行なっていない。WSFL の記述対象が本質的に分

[3]仕様書記載の DPE 処理には曖昧さがあり、DPE によって辿るリンクを明記していない。しかし、データリンクに論理値を伝播することの意味が不明確であるため、伝播対象を制御リンクと解釈した。

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。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。

本稿で提案したような変換アプローチによる検証方式では、検証の結果をもとのWSFL記述にフィードバックする方法が明確でないという問題点が指摘される。すなわち、検証の対象はPromela記述であり、そこで見つけた不具合がWSFLのどこに起因するかを見つけることが難しい。しかし、変換テンプレートで議論したように、WSFLからPromelaへの変換は直接的である。Promela記述での問題点は特定チャネルに情報が詰まるとか特定のプロセスが待ちループにい続ける、等の理由からくるデッドロックとして見つかる。チャネルとリンク、プロセスとアクティビティは、良い対応関係にあるため、もとのWSFL記述での問題箇所を指摘することは難しくはない。

WSFLからPromela記述を得るための変換テンプレートを整理する段階で、操作的意味の最も複雑な部分であるDPEの記載が曖昧であることがわかった。実際、いく通りかの解釈を考えることができそうである。操作的な意味に絡むこと、デッドロックの源になること等から明確な記述が望まれる。本稿で提案した検証の枠組とSPINを利用して、いくつかのバリエーションを実験することができる。より曖昧さの少ないWSFL仕様の改訂案を検討していきたいと考えている。

4 おわりに

Webサービスアグリゲーションの形式検証が必要であることを述べ、ソフトウェアに対するモデル検査検証技法を用いた検証実験を、WSFLならびにSPINを具体例として行なった。WSFLからPromela(SPINの入力仕様言語)への変換規則を整理し、不具合を自動的に見つけることが可能であることを例を用いて示した。今後、変換テンプレートならびに逆変換の規則を整理しWSFLとSPINを統合したツールを開発する。

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, 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] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.

[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. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

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

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

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

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

[11] S. Nakajima. CCSによるビジネスフロー図の検証. In *Proc. 1st JSSST Workshop on FOSE*, pages 135–140, December 1994.

[12] S. Nakajima. On Verifying Web Service Flows. In *Proc. SAINT 2002 Workshop*, pages 223–224, February 2002. Presented at WebSE 2002 [15].

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

[14] UDDI Web Site. UDDI Techinical White Paper. `http://www.uddi.org`. September 2000.

[15] WebSE 2002. International Workshop on Web Service Engineering. Nara, February 2002.