ソフトウェアテストシンポジウム 2006
JaSST'06: Japan Symposium on Software Testing 2006

サービス連携システム向けテストツールの一提案

住友 千紗† 吉田 英嗣† 山本 恭弘† 横山 和俊†

†（株）NTT データ 技術開発本部 〒104-0033 東京都中央区新川 1-21-2

E-mail: †{ sumitomoc, yoshidaei, yamamotoysd, yokoyamakz }@nttdata.co.jp

あらまし サービス指向アーキテクチャの基盤技術であるビジネスプロセスエンジンは，複数の Web サービスの連携情報を記述したビジネスプロセスを解釈・実行し，Web サービスを連携させたサービス連携システムを実現する．誤ったビジネスプロセスの実行は，実行される Web サービスの提供者全員に損失を与える恐れがあるため，ビジネスプロセスの品質向上は重要な課題である．我々は，Java などで開発されたシステムのテストで用いられている手法を，サービス連携システムのテストに適用した．その結果，レビュー稼動や修正稼動の削減，後工程での故障の発生防止およびテスト工程の稼動削減および品質向上に効果があることがわかった．

キーワード SOA：Service-Oriented Architecture，ビジネスプロセス，静的検証，動的検証

A proposal of a testing tool for service-integrated systems

Chisa SUMITOMO† Eiji YOSHIDA† Yasuhiro YAMAMOTO† and Kazutoshi YOKOYAMA†

† Research and Development Headquarters NTT DATA Corporation 21-2 , Shinkawa 1-chome , Chuo-ku, Tokyo, 104-0033 Japan

E-mail: †{ sumitomoc, yoshidaei, yamamotoysd, yokoyamakz }@nttdata.co.jp

Abstract In service-oriented architecture (SOA), a business process engine is commonly used to integrate web services by executing a business process, a description of how to integrate web services. Since executing a wrong business process may cause losses to every provider of executed web services, improvement of business process quality is one of the most significant challenges. In this paper, we have applied a technique used for testing a Java-based system to test business processes using the specifications of web services included in the processes as executed services. As a result, we found that running cost was decreased and quality was improved.

Keyword SOA：Service-Oriented Architecture， BusinessProcess， Static verification, Dynamic verification

1. はじめに

近年，サービス指向アーキテクチャ(SOA：Service-Oriented Architecture) [1]の基盤技術として，ビジネスプロセスエンジン[2]が注目されている．ビジネスプロセスエンジンは，複数のWeb サービス[3]の連携情報を記述したビジネスプロセス[4]を解釈・実行し，Webサービスを連携させたサービス連携システムを実現する．ビジネスプロセスエンジンを用いたサービス連携システムでは，ビジネスプロセスの記述を変更するだけでWeb サービスの連携を変更できるため，ビジネスの変化に柔軟に対応できるという利点がある[5]．一方，誤ったビジネスプロセスの実行は，実行されるWebサービスの提供者全員に損失を与える恐れがあるため，ビジネスプロセスの品質向上が重要な課題となっている．しかし，現状のビジネスプロセスモデリングツールでは，記述したビジネスプロセスに対する文法検証を実施できる程度である．本稿では，Java などで開発されたシステムのテストで用いられている手法を，サービス連携システムのテストに適用することにより，システム品質が向上し開発工数も削減した事例を報告する．

2. サービス連携システムのテスト

本章では Web サービスとビジネスプロセスの特徴について示すとともに，サービス連携システムにおいて要求されるテストの課題を示す．

2.1. Web サービス

Web サービスは，コンピュータシステムの機能に標準のインタフェースを持たせ，他のコンピュータシステムから利用可能にしたものである．Web サービスは，入出力インタフェースの仕様[6]についてのみ公開されていることが多い．

2.2. ビジネスプロセス

ビジネスプロセスは図 1**エラー! 参照元が見つかりません。**に示すように,開始ノード(S),終了ノード(E),制御フローで結合されるWebサービス呼出ノード

(N1,N2,N3,Ne)，実行条件(遷移条件)を評価する判断ノード（N4,N5），およびそれらの実行順序を指定する遷移（実線矢印：通常遷移，破線矢印：例外遷移）の集合であり，BPEL4WS[7]といったビジネスプロセス記述言語に従いXML 形式で記述される．また，以下の特徴を持つ．

(1) ビジネスプロセスの開始には入力を，終了には出力を定義することができる．
(2) Web サービス呼出ノードには呼出対象の Web サービスに与える入力および Web サービスからの出力を定義する．
(3) ビジネスプロセスの開始に与える入力および Web サービスからの出力は，ビジネスプロセス上で定義された変数に格納することができる．変数は，判断ノードにて遷移条件の判断に利用することができる．
(4) Web サービスで発生した例外を，遷移条件の判断に利用することができる（**エラー! 参照元が見つかりません。**における破線矢印）．
(5) 利用する Web サービスは互いに独立して動作しており，連携はビジネスプロセスによってのみ行われる．

2.3. サービス連携システムのテストにおける課題

サービス連携システムでは，他者提供の Web サービスをビジネスプロセスの記述により連携させるため，一般のシステムのテストと比較して，以下の二つの特徴的な課題がある．

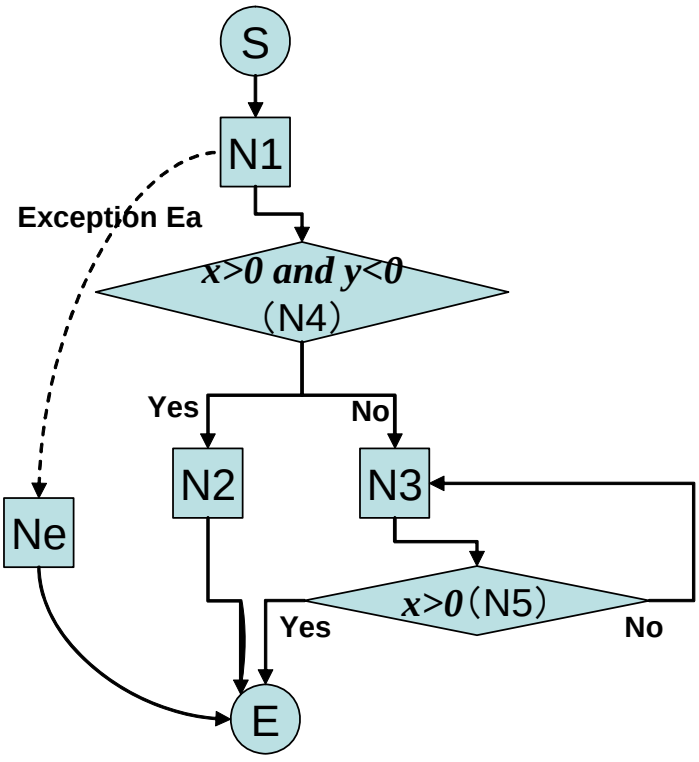

図 1　ビジネスプロセスの例

【課題 1】Web サービスはソースコードがなく，インタフェース等の仕様のみ与えられる．Web サービスを呼び出すビジネスプロセスに対しては，文法検証に加えて，Web サービスの仕様との整合性を静的にテストする必要がある．しかし，レビューによる検証では，ビジネスプロセスが利用する Web サービス数の増加に伴い，検証漏れが発生する恐れがある．したがって，検証漏れを防止し，早い段階で潜在エラーを検出するために，人手を介さずに自動的に検証することが求められる．

【課題 2】誤った順序に基づく Web サービスの実行は，連携対象となる Web サービスの所有者全員に損失を与える恐れがあるため，システム全体の動作テストとして，ビジネスプロセス上のパス（Web サービスの実行順序) をすべてテストすることが求められる．しかし，ビジネスプロセス上のパスは，ビジネスプロセス上の Web サービス呼出ノードや判断ノードの数，判断ノードのネスト数に従って膨大な数となるため，試験項目やテストケースの漏れの発生やテスト工数の増加が懸念される．したがって，試験項目やテストケースの自動生成およびテストの自動実行の実現が求められる．

3. ビジネスプロセス設計試験支援ツール biz∫

3.1. biz∫の概要

我々は，2.3で述べた二つの課題を解決するため，ビジネスプロセスの設計試験支援ツール「biz∫」を開発している．biz∫の構成を図 2**エラー! 参照元が見つかりません。**に示す．

biz∫は Eclipse ベースのツールであり，ビジネスプロセスの設計・製造・試験工程の支援／自動化を実現することで，開発効率化を目指している．biz∫の各機能の概要を説明する．

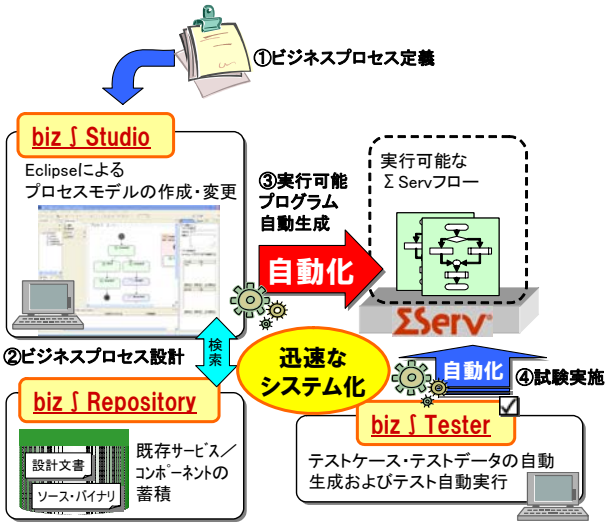

図 2　biz∫の構成

(1)　biz∫Studio／biz∫Repository

(a)　ビジネスプロセスエディタ

ビジネスプロセス設計を支援するエディタである．本エディタ上でフローの記述単位・実行単位であるWebサービス呼出ノード，遷移，遷移条件，入出力データ等を定義することで，ビジネスプロセスを設計することができる．その際，biz∫Repositoryに格納されている既存のWebサービスを検索し，状態に割り当てることで，そのWebサービスを呼び出す状態を定義できる．画面例を図 3**エラー! 参照元が見つかりません。**に示す．

(b)　静的検証機能

課題 1 を解決する機能として，エディタで記述したビジネスプロセスとWebサービスの整合性を自動的に静的検証する機能を提供する．それに加えて，一般のソースコードに対する静的検証機能と同様に，記述したビジネスプロセスに対して，各要素のスキーマやプロジェクト特有の規約との整合性検証機能，遷移条件や各要素に対する制約に対する設定した値の妥当性検証機能も実現している．ビジネスプロセスに対して検証機能を実行すると，エラー内容が一覧表示される（図 4参照）．

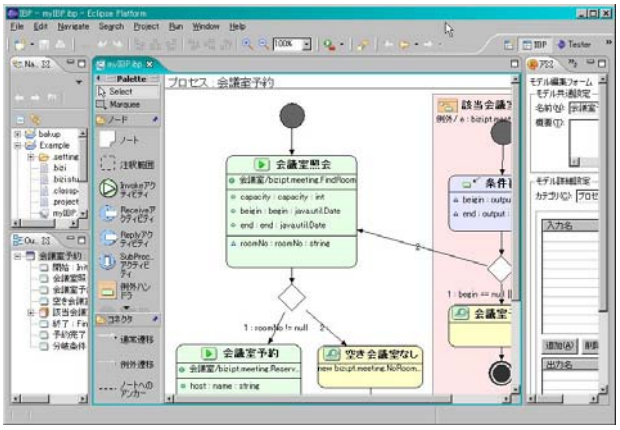

図 3　ビジネスプロセスエディタ例

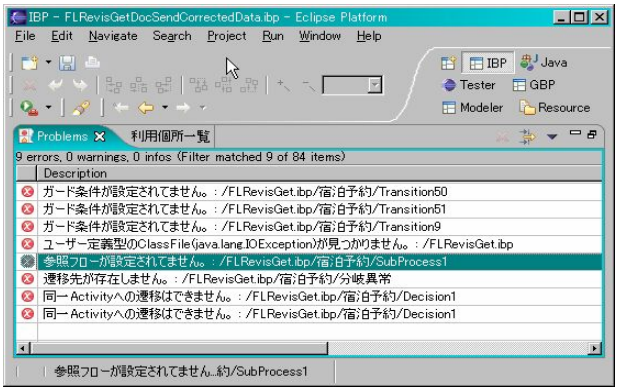

図 4　検証結果画面

(c)　プログラム生成機能

エディタで記述したビジネスプロセスから，ビジネスプロセスエンジン上で動作するフロープログラムを生成する．現在は，当社で開発した複数の情報システムを接続し，高信頼な連携サービスを提供するためのプラットフォームであるΣServ®[5]上で実行可能なフロープログラムを生成する．

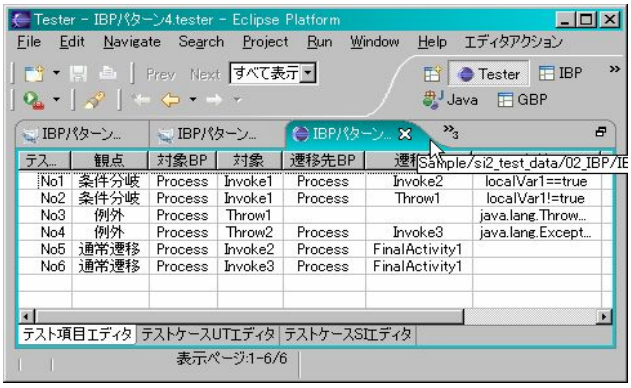

図 5　テスト項目一覧

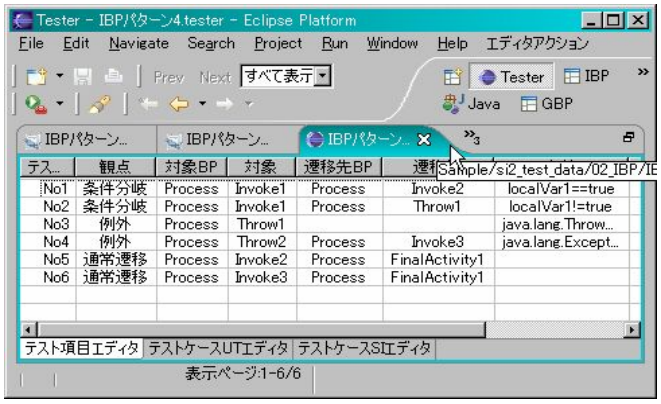

図 6　テストケース一覧

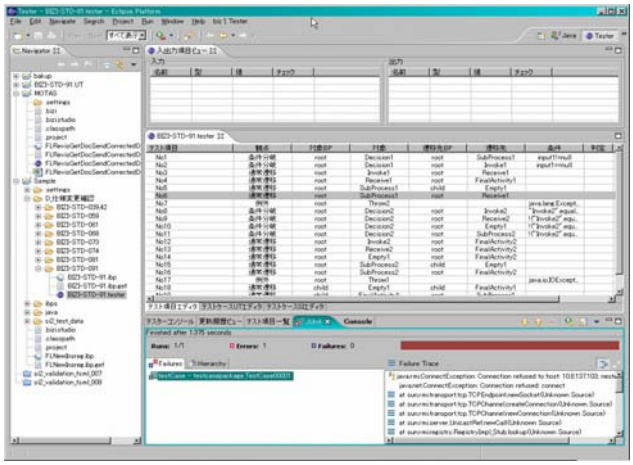

図 7　テスト結果画面例

(2)　biz∫Tester

(a)　テスト項目，テストケース自動生成機能

ビジネスプロセス上の隣接する 2 つのWebサービス間のパスをテスト項目とする．テスト項目の抽出観点を複数条件網羅と例外補足として，ビジネスプロセスの全パス検索を実施し，ビジネスプロセスのテスト項目を自動生成する．また，ビジネスプロセスをリンク網羅またはパス網羅検索し，各テスト項目を消化できるテストケースを自動生成する．テスト項目一覧例を図 5に，テストケース一覧例を図 6に示す．

(b)　テスト実行機能

Σ Serv®上では，定義されたビジネスプロセスから生成されたフロープログラムにしたがって，クライアントから要求を受け取ったり，プログラムコンポーネントを順次呼び出したりして，ビジネスプロセスを走行させ，結果を返却する．したがって，テストケースに従った試験を実行するためには，クライアントとなるドライバと，プログラムコンポーネントとなるスタブが必要である．biz∫Testerでは，ユーザが設定した入出力データをもとに，それらを送受信するスタブやドライバを自動生成する．そして，自動生成したスタブやドライバおよび生成されたフロープログラムをΣ Serv®上に配備し，試験を自動実行する．biz∫Testerでは，フロー遷移の確認を行うテストだけでなく，サーバントの入出力値とユーザが設定したデータの整合性を確認するテストも実施できる．テスト結果は，テスト項目ごとにテスト項目エディタ上で表示する．テスト実行画面例を図 7**エラー! 参照元が見つかりません。**に示す．

3.2. 課題への対処

3.2.1. 事前条件と事後条件

前節で述べた機能の中で，2.3における課題を解決する，特徴的な機能の詳細について説明する．近年，OWL-S [8]に代表されるように，Webサービスの事前条件，事後条件，発生例外といったWeb サービスの詳細仕様を記述・公開する方法が提案され普及しつつある．そこで本稿ではWeb サービスの仕様として,表 1**エラー! 参照元が見つかりません。**や表 2**エラー! 参照元が見つかりません。**に示すような入出力インタフェース，発生例外，事前条件，事後条件を利用することにより，従来に比べて精度の高い整合性検証と厳密な絞込みを行えるテストケース設計を実現する．事前条件とは，対象のWeb サービスを呼び出す時点で真になっていなければならない条件であり，Web サービス実行の前提を示す．また事後条件とは，Web サービスの実行に成功した場合に満足される条件であり，Web サービス実行の効果を宣言する．

3.2.2. Web サービス仕様との整合性検証

課題 1 を解決するため，Webサービスの仕様記述を用いた，ビジネスプロセスとWebサービスの整合性検証機能を開発した．実現した整合性検証機能を以下に示す（詳細は，[9]を参照のこと）．

(a)　入出力インタフェースの検証

ビジネスプロセス上の入出力パラメータの数およびそれぞれの型が Web サービスの入出力インタフェースの定義と一致しているかを検証する．

(b)　発生例外の検証

Web サービスの仕様で記述されている発生例外に基づき，ビジネスプロセス上で当該発生例外に基づく遷移が定義されているかを検証する．

(c)　事前条件，事後条件，遷移条件の組合せ

全パスに対してWebサービス間の事前条件・事後条件や遷移条件の組合せを抽出する．それらの組み合わせを数理計画ソルバー[10][11][12]によって解くことによって，遷移不可能な遷移を検出する．

3.2.3. ビジネスプロセステスト支援

課題 2 を解決するため，ビジネスプロセスから全パスを網羅するテスト項目とそれらを消化するテストケースを自動生成する機能を開発した．その際，Web サ

ービス仕様である出力パラメータ，事前条件，事後条件を考慮して実行不可能な遷移パスをテストケースから除外し，テストすべきテストケースの絞込みを実現する．

(1) テストケース設計手順

本稿で提案するテストケース設計の手順を以下に示す．

【手順 1】隣接 Web サービス間のパスの作成

ビジネスプロセス上の隣接する 2 つのWebサービス間のパス($A \rightarrow B$)を考える．$A \rightarrow B$上に判断ノードがある場合，判断ノードから遷移条件の全ての組合せの集合($C_{A \rightarrow B}$) を抽出する．判断ノードがない場合，$C_{A \rightarrow B}$ は空集合とする．そして，$C_{A \rightarrow B}$の各要素に，Aの事後条件とBの事前条件を追加する．$C_{A \rightarrow B}$が空集合の場合，A の事後条件とBの事前条件を持つ要素を$C_{A \rightarrow B}$に追加する．

【手順 2】パス式の作成

すべての隣接する Web サービス間に手順 1 の処理を適用し,ビジネスプロセスの開始から終了に至るすべてのパスを表現するパス式を作成する．

【手順 3】代入・条件リストの作成

パス式より，テストケースを抽出し，ビジネスプロセスの入力，テストケース上の各 Web サービスの出力，テストケースの遷移条件から,変数へのデータの代入・条件判定順序を表すリストを作成する．

【手順 4】テストケースの実行可能性検証

代入・条件リストに存在するすべての条件を数理計画ソルバーによって解く．解が存在しなければ，実行不可能であり，当該テストケースをテスト対象から除外する．

上記手順では，判断ノードの条件だけでなく Web サービスの出力パラメータ，事前条件，事後条件という制約も考慮されている．よって判断ノードの遷移条件のみを考慮して作成したテストケース集合に比べ，提案手法では，実行不可能な遷移パスが除外されたテストケース集合を作成することができる．

(2) テストケース設計例

(1)の処理を図 1**エラー! 参照元が見つかりません。**に適用した場合の例を以下に示す．

【手順 1】図 1**エラー! 参照元が見つかりません。**のN1 からN3 間の遷移($N1 \rightarrow N3$)を考える． N1 の事後条件が表 1**エラー! 参照元が見つかりません。**のCase2 に示す$x + y > 0$, N3 の事前条件が表 2**エラー! 参照元が見つかりません。**のCase2 に示す$y > 0$ であれば，$C_{N1 \rightarrow N3}$は表 3**エラー! 参照元が見つかりません。**のようになる．

【手順 2】すべての隣接するWeb サービス間に上記の処理を適用し,各組合せの集合の各要素に識別子をつけることにより,図 8に示すグラフ表現を作成することができる．図 8に示すグラフ表現に対し，ビジネスプロセスの開始から終了に至るパスの集合を表現するパス式を導出する．一般に知られているグラフのノード削減アルゴリズム[13]を利用することで，式(1)を得ることができる．

$$a(f + g)k + a(c + d + e)\ h^{*}i + abj \quad (1)$$

式(1) で + はor を表し，例えば，a(f + g)k でafkまたagkのパスを表す．また * はループを表し，h^{*}はhを 0 回以上任意に繰り返すことを表している．また，式(1) に示すパス式において，ループ回数を制限することによりパス式から有限のテストケースの集合を得ることができる．

表 1 N1 の Web サービス仕様

	Case1	Case2
入力	x, y	x, y
出力	x	x
事前条件	$y < 0$	$y < 0$
事後条件	$x + y = 0$, $y < 0$	$x + y > 0$
発生例外	Ea, Eb	Ea, Eb

表 2 N3 の Web サービス仕様

	Case1	Case2
入力	x, y	x, y
出力	x	x
事前条件	$x < 0$	$y > 0$
事後条件	$y < 0$	$x + y > 0$
発生例外	-	-

表 3 $C_{N1 \rightarrow N3}$

識別子	条件
c	$x \leqq 0, y < 0, x + y > 0, y > 0$
d	$x \leqq 0, y \geqq 0, x + y > 0, y > 0$
e	$x > 0, y \geqq 0, x + y > 0, y > 0$

【手順 3】式(1) に含まれるadhiのパスを検証することを考える．但し，a,d,h,iの各遷移条件が表 1**エラー! 参照元が見つかりません。**のCase2, 表 2**エラー! 参照元が見つかりません。**のCase2 の仕様とビジネスプロセス上の判断ノードより表 4であり，ビジネスプロセスの開始ポイント(S) における入力がx, yであるとする．表 4とビジネスプロセスの入力，Web サービスの出力を遷移順序に従い並べ，表 5に示す代入・条件リストを作成する．さらに表 5のリストの先頭から，ビジネスプロセスの入力およびWeb サービスの出力の変数がどの条件式で利用されるかを考え，変数名を別名に置き換える．置き換えた表を表 6に示す．

【手順 4】表 6に存在するすべての条件を数理計画ソルバーによって解く．表 6では，x_1, x_2, x_3, x_4, y_1

の解は存在しないため，adhiのパスは実行不可能であり，テスト対象外となる．

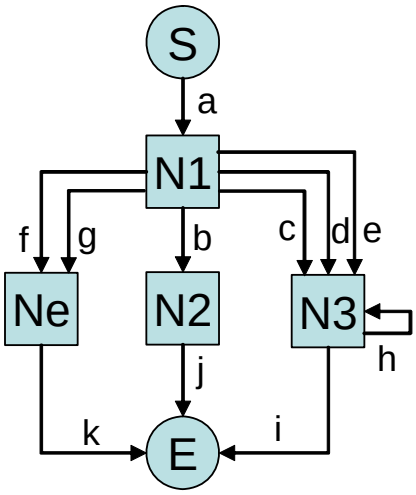

図 8　パスのグラフ表現

表 4　パス条件リスト

	条件
a	$y<0$
d	$x\leqq 0, y\geqq 0, x+y>0, y>0$
h	$y>0, x+y>0, x\leqq 0$
i	$x+y>0, x>0$

表 5　代入・条件リスト

	変数	条件
S	x, y	
a		$y<0$
N1	x	
d		$x\leqq 0, y\geqq 0, x+y>0, y>0$
N3	x	
h		$y>0, x+y>0, x\leqq 0$
N3	x	
i		$x+y>0, x>0$

表 6　別名置換済代入・条件リスト

	変数	条件
S	x_1, y_1	
a		$y_1<0$
N1	x_2	
d		$x_2\leqq 0, y_1\geqq 0, x_2+y_1>0, y_1>0$
N3	x_3	
h		$y_1>0, x_3+y_1>0, x_3\leqq 0$
N3	x_4	
i		$x_4+y_1>0, x_4>0$

4. 評価

4.1. 評価の目的

biz∫の実アプリケーション適用性とbiz∫による開発効率化についての評価を行った．評価アプリケーションであるAシステムを例として挙げ，biz∫の適用方法と，実際にbiz∫を利用して開発を実施することによる効果についての評価を行った．Aシステムの構成概要を図 9に示す．評価段階では，Webサービス仕様との整合性検証およびテストケース削除機能は実装されていなかったため，それ以外の静的テスト（検証）機能はおよびテスト項目・テストケース生成機能，テスト実行機能に関する評価について報告する．

4.2. 評価アプリケーション概要

評価アプリケーションとしてAシステムを例に挙げ，biz∫がどのように利用可能かを検討し，その結果から実際に biz∫により開発を実施して適用に対する効果(開発効率化の程度)を評価した．

Aシステムは，複数のシステム間の業務連携を可能とするシステムである．Aシステム内には複数の他システムとの接続を制御するシステムがあり，その制御システム内に存在するΣServ®を利用したワークフロー制御機能サーバにより他システムとの接続制御が行われる．つまり，ワークフロー制御機能サーバが，各システムとの接続を行うためのコンポーネントなどを組み合わせたシステムとなっている．

本評価では，Aシステム全体における，ワークフロー制御機能の開発について，biz∫を利用した場合のビジネスプロセスの設計工程からテスト工程における稼動および成果物の品質の観点で評価した．

Aシステム開発における設計工程からテスト工程の流れは，以下である．

(1) フロープログラム設計書作成

分岐条件,変数や呼び出すコンポーネントを検討し，ビジネスプロセスを設計する．そして，その結果を設計書として文書化する．

(2) 設計書レビュー

設計書が要件を満たしているかを確認する．

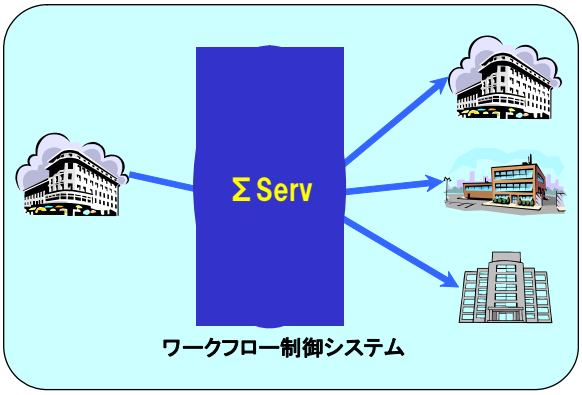

図 9　Aシステム構成概要

(3) プログラム製造

プログラム設計書の記載事項を把握し，それに基づいてプログラムを生成する．

(4) セルフチェック，ソースレビュー

ソースレビューチェックリストを基に，まず担当者が自己チェックし，仕様の反映漏れを防ぐ．レビュー時は，設計書およびソースレビューチェックリストを用いて確認を行う．

(5) 試験項目票の作成

プロジェクトで決められた抽出観点に基づいて，テスト項目を作成する．biz∫におけるテスト項目一覧に相当する．

(6) 試験項目票レビュー

プログラム設計書に記載された機能に対する試験項目の網羅性，及びソースレベルでの機能実現に対する試験観点の有効性をチェックし，試験項目票の抽出レベルを確認する．また，異常系についてのチェックも行う．

(7) 試験手順書の作成

試験項目を消化するテストケースを作成する．biz∫におけるテストケース一覧に相当する．

(8) 試験手順書のレビュー

テストケースが全テスト項目を網羅しているかを確認する．

(9) テストプログラム作成

スタブやドライバを生成する．

(10) 試験実施

(5)で生成したスタブとドライバを用いたデバッガによる試験を実施する．

(11) 試験結果レビュー

上記試験に於ける試験実施方法の正当性，及び試験結果確認の妥当性等をレビュー検証し，試験に問題がなかった事の確認を行う．

4.3. 結果と考察

4.3.1. 静的検証機能

(1) 抽出（防止）可能な PD 工程でのエラー割合

評価プロジェクトにおける開発成果物のレビューによって検出された問題のうち，本機能によって発生防止が可能な問題の割合は全体の 11 %であった（図 10**エラー! 参照元が見つかりません。**参照）．検出した問題はPD工程で混入する割合が高く，主な内容は，設計誤りや表現上の記述不良であり，主なエラー原因は，基本的な部分での確認不足等などであった．これらのことから，本機能により，主に軽微なエラーを発見できると考えられる．

(2) 抽出（防止）可能な M・UT 工程でのエラー割合

評価プロジェクトにおけるM・UTにおける故障の原因となってしまっていた問題のうち，本機能をPD工程で利用することによって発生防止が可能な問題の割合は，全体の 71%であった（図 11**エラー! 参照元が見つかりません。**参照）．検出した問題の内容は，プログラム設計やコーディングでの誤り，データ定義エラー等であり，主なエラー原因は，ケアレスミスの様な単純なエラーであった．これらのことから，本機能により，本来UTで発生するエラーを未然に防ぐことが可能であると考えられる．

静的検証機能により，各工程での問題の摘出漏れを少なくすることが出来，それにより，後工程で発生していたエラー（故障）を大幅に防止することが可能となるため，品質の向上および修正稼動の削減が見込める．さらに静的検証に掛かる稼動は，無視してもかまわない時間で終わるため，レビュー前に本機能を利用することで，問題を事前に発見・修正でき，レビュー稼動の削減も見込める．

4.3.2. ビジネスプロセステスト実行支援機能

評価プロジェクトに対してテスト項目自動生成機能を適用した結果，テスト項目抽出基準に従ってコンディションカバレッジで抽出されることと網羅率が100%であることを確認した．同様にテストケース自動生成機能を適用した結果，抽出された試験項目がすべてテストケースに含まれていることを確認した．また，フロープログラムすべての試験項目およびテストケースを抽出しPDFで出力するまでの時間を計測した結果，どちらの場合も，従来と比較して 9 割以上短縮できることがわかった（図 12参照）．

次に，評価プロジェクトに対してテスト実行機能を適用し，試験実行に要する時間を計測した．ただし，biz∫では，テスト項目抽出，テストケース抽出，テスト実行（スタブ・ドライバ生成含む）を一連の操作で行うため，それらの操作時間も含んでいる．その結果，全テスト項目の消化時間を，従来と比較して 9 割以上短縮できることがわかった（図 12参照）．

ただし，本評価では，4.2で述べたテスト工程のうち，(5)(7)(9)(10)の作業に要した時間のみを計測したが，本来なら自動生成されたテスト項目やテストケース，テスト結果のレビューを実施する必要がある．全体のレビュー時間も含めた場合の評価は今後実施する予定であるが，従来と比較して 5 割～ 7 割程度の削減になると想定している．

ビジネスプロセステスト実行支援機能を利用することで，テスト項目の抽出からテスト実行に至る全フェーズで稼働を削減することができる．それにより，仕様変更等による再試験実施の稼動を減らすことができるだけでなく，試験実施者は仕様と製造成果物の整合性の確認に集中することができ，品質の向上にも寄与できる．

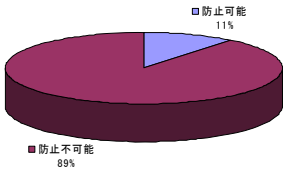

図 10　防止可能なエラーの割合(PD 工程)

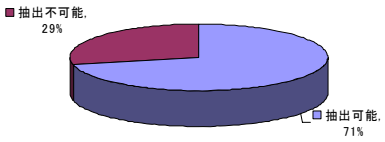

図 11　防止可能なエラーの割合（M・UT 工程）

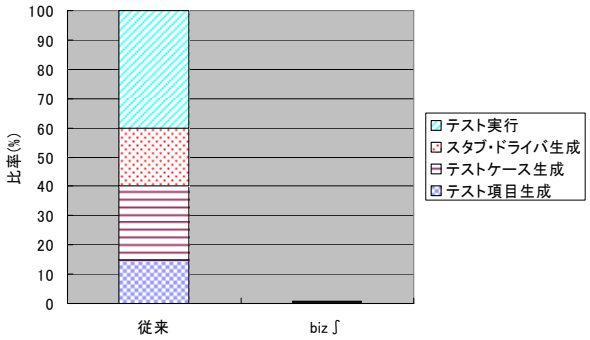

図 12　各機能の実行時間の割合

5. 今後の課題・展開

現在，本稿で提案したテスト機能を実装したプロトタイプを開発中であり,当社のプロジェクトに適用し,本手法を評価する予定である．今後は，ソフトウェアテストを効率よく進められるモデル設計手法やテスト自動実行手法，性能試験工程の自動化技術の開発やさらに上位工程での成果物検証技術の開発を進めていく予定である．

文　献

[1] D. Krafzig et al, “Enterprise SOA”, printice hall, pp.1-2, November 2004.

[2] M. Kloppmann et al, Business process choreography in WebSphere: Combining the power of BPEL and J2EE, http://www.research.ibm.com/journal/sj/432/kloppmann.html

[3] Ethan Cerami, Web サービスエッセンシャルズ，オライリー・ジャパン, pp4-5,2002.

[4] Tony Andrews et al, ”Business Process Execution Language for Web Services (BPEL4WS) Version 1.1”, ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf , May 2003.

[5] 吉田英嗣，横山和俊，松田栄之，“サービス連携プラットフォームΣ Serv”, pp.47-49, NTT 技術ジャーナル，2003 年 7 月号

[6] Erik Christensen et al, ”Web Services Description Language (WSDL) 1.1”, World Wide Web Consortium, March 2001.

[7] Matjaz B. Juric et al, ”Business Process Execution Language For Web Services”, Packt Publishing, September 2004.

[8] OWL-based Web service ontology, http://www.daml.org/services/owl-s/1.0/

[9] 山本恭弘，吉田英嗣，坂田祐司，横山和俊，”サービス連携システムにおけるテスト設計手法の提案”，pp.47-49，FOSE2005，November 2005.

[10] lp solve -Mixed Integer Programming (MIP) solver-, http://groups.yahoo.com/group/lp solve/

[11] GLPK (GNU Linear Programming Kit), http://www.gnu.org/software/glpk/glpk.html

[12] cream, http://bach.istc.kobe-u.ac.jp/cream/

[13] Boris Beizer, ソフトウェアテスト技法，日経 BP 出版センター, pp. 346-347,1994.